MINISTERUL EDUCAŢIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA

АНАТОЛ ГРЕМАЛЬСКИ
СЕРЖИУ КОРЛАТ
АНДРЕЙ БРАЙКОВ

12

Информатика

Учебник для 12-го класса

Ştiinţa, 2025

CZU

Acest manual este proprietate publică, editat din sursele financiare ale Fondului special pentru manuale.

Manualul școlar a fost elaborat în conformitate cu prevederile Curriculumului la disciplină, aprobat prin Ordinul ministrului educației și cercetării nr. 906, din 17 iulie 2019.

Manualul a fost aprobat prin Ordinul ministrului educației și cercetării nr. 1928, din 27.12.2024, ca urmare a evaluării calității metodico-științifice.

_____

(Denumirea instituției de învățământ)

EVIDENȚA UTILIZĂRII MANUALULUI:

Anul de folosire	Numele și prenumele elevului	Anul de studii	Aspectul manualului	
			La primire	La returnare

Dirigintele clasei verifică dacă numele, prenumele elevului sunt scrise corect.
Elevii nu vor face niciun fel de însemnări în manual.
Aspectul manualului (la primire și la returnare) se va aprecia de diriginte cu unul dintre calificativele: nou, bun, satisfăcător, nesatisfăcător.

ÎNTREPRINDEREA
EDITORIAL-POLIGRAFICĂ
ȘTIINȚA

str. Academiei, nr. 3; MD-2028, Chișinău, Republica Moldova
tel.: +373 22 739616 (anticamera)
+373 22 739983; +373 60966868 (vânzări)
e-mail: prini_stiinta@yahoo.com
www.editurastiinta.md

Responsabilă de ediţie: Natalia Ciobanu
Redactor: Tatiana Şarşova
Corector: Maria Cornesco
Redactor tehnic: Nina Duduciuc
Machetare computerizată și copertă: Romeo Șveț

Descrierea CIP a Camerei Naţionale a Cărţii

Дорогие друзья!

Информатика как наука и информационно-коммуникационные технологии как совокупность методов и средств, основанных на достижениях этой науки, постоянно развиваются. В этом мы можем убедиться на собственном опыте: каждый год появляется новое оборудование и цифровые устройства, новые программные продукты и новые онлайн-сервисы. Повседневная практика еще раз доказывает нам, что средства информационно-коммуникационных технологий проникают во все сферы экономической и социальной жизни, существенно способствуя повышению производительности труда, меняя способы работы, обучения, отдыха, общения с другими людьми. Эти средства помогают нам распространять информацию о себе и узнавать информацию о других, реализовать свой творческий потенциал. Чтобы не потеряться в этом многообразии цифрового оборудования и продуктов, эффективно и результативно использовать программные продукты, иметь возможность безопасного доступа к цифровым сервисам, очень важно глубоко разбираться в принципах, лежащих в основе информатики и информационно-коммуникационных технологий. Также важно формировать и развивать собственные навыки использования этих технологий для передачи, получения, хранения и обработки числовой, текстовой, аудио- и видеоинформации, создания самых разнообразных цифровых продуктов: компьютерных программ, баз данных, веб-сайтов и веб-страниц, мультимедийных произведений.

Поэтому независимо от выбранного вами лицейского профиля – гуманитарного или реального, спортивного или художественного – цифровые навыки играют ключевую роль в достижении ваших целей личностного развития, успешной профессиональной карьеры и полноценной жизни. Формирование и развитие этих навыков является основной задачей данного учебника. В частности, к концу 12 класса вы сможете:

- разрабатывать и реализовывать алгоритмы решения часто встречающихся в повседневной жизни задач;
- выбирать и реализовывать на компьютере наиболее распространенные методы программирования;
- разрабатывать и реализовывать компьютерные модели самых разнообразных объектов, применять часто используемые численные методы;
- систематизировать и обрабатывать информацию с использованием систем управления базами данных.

Также изучение материалов, включенных в учебник, будет способствовать формированию следующих навыков и ценностей:

- стремление к познанию себя и окружающего мира с использованием цифровых средств;
- корректность и последовательность в применении цифровых инструментов, инициативность и настойчивость в алгоритмизации задач и реализации алгоритмов;
- любознательность и интерес, критический и творческий подход к познанию мира с помощью компьютерного моделирования;
- соблюдение правил безопасности, эргономики, этики и дизайна при создании и распространении цифровых продуктов.

Изучение одного из модулей по выбору, описанных в главе 5, поможет освоить методы и инструменты, специфичные для цифровой обработки, а также развить мышление и креативность при интеграции информатики с другими областями знаний. Дополнительные материалы, представленные в главе 6, являются факультативными и предназначены для углубленного изучения отдельных тем по информатике.

Изучение модулей по выбору и расширений рекомендуется осуществлять методом «перевернутого класса». Этот метод основан на самостоятельном изучении материала учащимися, а преподавателю отводится роль консультанта и наставника.

Основные образовательные ресурсы, рекомендуемые ученикам для изучения выбранных ими модулей и расширений, являются частью данного учебника и доступны для скачивания на веб-странице Центра информационных и коммуникационных технологий в образовании: http://www.ctice.gov.md.

Желаем вам успехов,
авторы

СОДЕРЖАНИЕ

Глава 1

Функции и процедуры

1.1 Подпрограммы

Известно что любая сложная задача может быть решена путем ее разбиения на ряд подзадач. Для решения каждой подзадачи записывается соответствующая последовательность операторов, называемых **подпрограммами.** Таким образом, задача решается с помощью основной программы, в которой для решения подзадач вызываются подпрограммы. Когда в основной программе встречается обращение к подпрограмме, управление передается первому оператору вызванной подпрограммы (*рис. 1.1*). После завершения выполнения оператора подпрограммы управление передается оператору основной программы, следующему за оператором вызова подпрограммы.

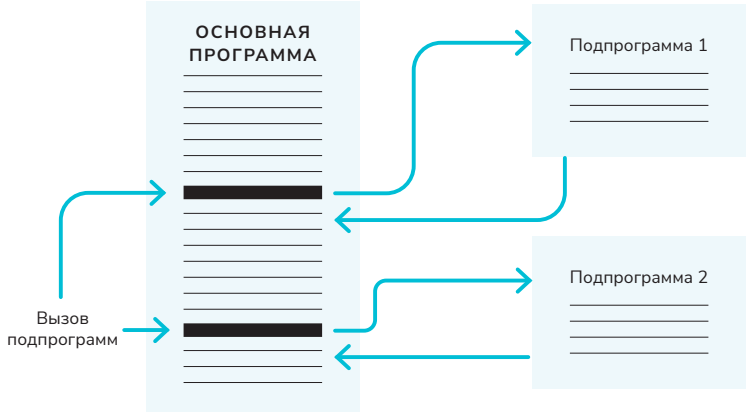


Рис. 1.1. Взаимодействие между программой и подпрограммой

Pascal

В языке PASCAL существуют два вида подпрограмм: функция и процедура.

<*Подпрограммы*> ::= { <*Функция*>; | <*Процедура*>; }

Функция – это подпрограмма, которая вычисляет и возвращает некоторое значение. Язык PASCAL содержит ряд стандартных функций, известных любой программе: `sin`, `cos`, `eof` и т.д. Помимо этого, программист может создавать собственные функции, к которым можно обращаться так же, как и к стандартным функциям. Таким образом, концепция функции расширяет понятие **выражения** в языке PASCAL.

Процедура – это подпрограмма, которая осуществляет обработку данных, переданных в момент обращения к ней. В языке PASCAL существуют стандартные процедуры: `read`, `readln`, `write`, `writeln` и другие. Помимо этого, программист может создавать собственные процедуры, к которым можно обращаться так же, как и к стандартным процедурам. Таким образом, концепция процедуры расширяет понятие **оператора** в языке PASCAL.

C++

В языках C/C++ подпрограммы носят название функций. Языки C/C++ содержат множество предопределенных функций, объединенных в библиотеки. Функции определенной библиотеки становятся доступными разрабатываемой программе путем включения в нее данной библиотеки. Напомним, что включение библиотеки осуществляется с помощью директивы `#include`:

```
#include <Имя библиотеки>
```

Например, для того, чтобы использовать функции ввода/вывода, разрабатываемая программа должна содержать директиву:

```
#include <iostream>
```

Для того, чтобы использовать математические функции, разрабатываемая программа должна содержать директиву:

```
#include <math.h>
```

Помимо предопределенных функций, объединенных в библиотеки, программист может создавать и свои собственные функции. Функции, определенные программистом, вызываются точно так же, как и предопределенные функции.

В языках C/C++ различаем две категории функций:

1) Типизированные функции. Вызов таких функций осуществляется путем включения их имени в любое выражение C/C++. При вызове функция вычисляет и возвращает значение, тип которого указан явно в объявлении функции.

Примеры: `sin`, `cos`, `abs`, `fabs` и др.

2) Нетипизированные функции (`void`). Вызов функции без типа осуществляется путем включения ее имени в любом месте программы, где может появиться оператор. Будучи вызванной, функция обрабатывает данные, переданные ей во время вызова.

Примеры: `scanf`, `printf`, `eoln` и др.

Очевидно, что понятие *типизированной функции* расширяет понятие **выражения**, а понятие *нетипизированной функции* – понятие **оператора**.

Pascal C++

Независимо от используемого языка программирования, подпрограммы объявляются в разделе описаний программы. Очевидно, что вызовы функций и процедур включаются в исполняемую часть программы.

Подпрограмма может обращаться и к самой себе, такое обращение называется **рекурсивным**.

Вопросы и задания

1. Объясните термины *основная программа* и *подпрограмма*.
2. Как взаимодействуют программа и подпрограмма?
3. (PASCAL) В чем разница между процедурами и функциями?
4. (C/C++) В чем разница между типизированными и нетипизированными функциями?
5. **Систематизируйте свои знания!** (PASCAL) Как происходит обращение к функции? В каких местах программы может появиться обращение к функции? Как происходит обращение к процедуре? В каких местах программы может появиться обращение к процедуре?
6. **Систематизируйте свои знания!** (C/C++) Как вызвать типизированную функцию? Где в программе могут происходить вызовы типизированных функций? Как вызвать нетипизированную функцию? Где в программе могут происходить вызовы нетипизированных функций?
7. **Обратите внимание!** (PASCAL) Назовите тип аргумента и тип возвращаемого результата следующих предопределенных функций `abs`, `chr`, `eof`, `eoln`, `exp`, `ord`, `sin`, `sqr`, `sqrt`, `pred`, `succ`, `trunc`. Назовите тип фактических параметров процедур `read` и `write`.
8. **Обратите внимание!** (C/C++) Назовите тип возвращаемого результата следующих типизированных функций из библиотеки `<math.h>`: `abs`, `fabs`, `exp`, `sin`, `sqr`, `sqrt`, `trunc`.
9. **Проведите исследование!** (PASCAL) Какую обработку данных осуществляют процедуры `read` и `write`?
10. **Проведите исследование!** (C/C++) Какую обработку данных осуществляют нетипизированные функции `scanf` и `printf` языка C или их эквиваленты `cin` и `cout` языка C++?

1.2 Функции

Pascal

На языке PASCAL функция **объявляется** следующим образом:

```
function f(x1, x2, ..., xn) : tr ;
D;
begin
   ...
   f := e;
     ...
end;
```

Первая строка – это заголовок функции, состоящий из:

f – имя функции;

(x_1, x_2, ..., x_n) – произвольный список формальных параметров, являющихся аргументами функции;

t_r – тип результата; может быть простым или ссылочным.

За заголовком следует **тело функции**, состоящее из произвольных локальных описаний D и составного оператора **`begin ... end`**.

Локальные описания состоят из следующих разделов (некоторые из них могут отсутствовать) **`label`**, **`const`**, **`type`**, **`var`**, **`function/procedure`**.

Имя функции f должно появиться хотя бы один раз в левой части некоторого оператора присваивания: $f := e$. Последнее значение, присвоенное функции f, возвращается в место ее вызова.

Как правило, формальный параметр из списка (x_1, x_2, ..., x_n) имеет вид:

$v_1, v_2, ..., v_k : t_p$

где $v_1, v_2, ..., v_k$ – идентификаторы, а t_p – имя типа.

Обращение к функции f осуществляется следующим образом:

$f(a_1, a_2, ..., a_n)$

где $(a_1, a_2, ..., a_n)$ – **список фактических параметров**. Обычно фактическими параметрами являются выражения, значения которых передаются в функцию. Соответствие между фактическим параметром и формальным параметром осуществляется по положению, занимаемому ими в двух списках. Тип значения фактического параметра должен быть **совместим** с типом формального параметра.

Пример:

Pascal
```
Program P97;
 {Объявление и использование функции Putere }
 type Natural=0..MaxInt;
 var a : real;
     b : Natural;
     c : real;
     s : integer;
     t : integer;
     v : real;
 function Putere(x : real; n : Natural) : real;
  {вычисление x в степени n  }
 var p : real;
     i : integer;
 begin
   p:=1;
   for i:=1 to n do p:=p*x;
   Putere:=p;
 end; { Putere }
 begin
   a:=3.0;
   b:=2;
   c:=Putere(a, b);
   writeln(a:10:5, b:4, c:10:5);
   s:=2;
   t:=4;
   v:=Putere(s, t);
   writeln(s:5, t:4, v:10:5);
   readln;
 end.
```

Функция `Putere` имеет два формальных параметра: `x` типа `real` и `n` типа `Natural`. Функция возвращает значение типа `real`. В теле функции описаны локальные переменные `p` и `i`.

При обращении к функции `Putere(a,b)` значения `3.0` и `2` фактических параметров `a`, `b` передаются формальным параметрам – соответственно `x` и `n`. Отметим, что тип параметра `a` совпадает с типом параметра `x`, а тип `b` – с типом `n`.

В случае обращения к функции `Putere(s,t)` тип фактических параметров `s,t` не совпадает с типом формальных параметров – соответственно `x` и `n`. Однако такое обращение корректно, так как соответствующие типы совместимы с точки зрения присваивания.

На языках C/C++ **функция объявляется** следующим образом:

C++

```
<Тип результата> <Имя функции> (x1, x2, ..., xn)
D;
{
   ...
   return <Результат>
 }
```

Первая строка — это вызов функции, состоящий из:

<*Тип результата*> — тип возвращаемого результата. Это должно быть имя простого или ссылочного типа.

<*Имя функции*> — имя функции.

(x_1, x_2, ..., x_n) — опциональный список формальных параметров, представляющих аргументы функции.

За заголовком следует **тело функции**, состоящее из опциональных локальных объявлений и составного оператора `{...}`.

Локальные объявления в основном представляют собой объявления переменных. Переменные, объявленные в теле функции, называются **локальными переменными**.

Имя функции используется для ее вызова. При вызове функция возвращает вызывающей программе значение <*Результат*>, указанное после ключевого слова `return`.

Обычно формальный параметр из списка (x_1, x_2, ..., x_n)) имеет вид:

```
t1 v1, t2 v2, ..., tn vn
```

где пары t_1 v_1, t_2 v_2, ..., t_n v_n формируются из <*Имя типа*> и <*Идентификатор*>.

Вызов функции имеет вид:

```
<Имя функции> (a1, a2, ..., an)
```

где (a_1, a_2, ..., a_n) – это **список фактических параметров**. Обычно фактическими параметрами являются выражения, значения которых передаются в функцию. Соответствие между фактическим параметром и формальным параметром осуществляется по положению, занимаемому ими в двух списках. Тип значения фактического параметра должен быть **совместим** с типом формального параметра.

Пример:

C++

```
// Program P97;
// Объявление и использование функции Putere

#include <iostream>
    using namespace std;
double putere (double x, int n)
```

```
{
    double p = 1.0;
    for (int i = 1; i <= n; i++)
        p = p * x;
    return p;
}

int main()
{
    double a = 3.0, c;
    int b = 2;
    c = putere(a, b);
    cout << a << "^" << b <<" = "<< c << endl;
    int s = 2, t = 4;
    c = putere(s, t);
    cout << s << "^" << t <<" = "<< c << endl;
    return 0;
}
```

Функция `putere` имеет два формальных параметра: `x` вещественного типа с двойной точностью и `n` типа **`int`**. Функция возвращает значение вещественного типа с двойной точностью. Локальные переменные `p` и `i` объявляются в теле функции.

При выполнении вызова `putere(a,b)` значения 3.0 и 2 текущих параметров `a`, `b` передаются формальным параметрам `x` и `n` соответственно. Следует отметить, что тип `a` совпадает с типом `x` и тип `b` совпадают с типом `n`.

В случае вызова `putere(s,t)` типы реальных параметров `s`, `t` не совпадают с типами формальных параметров `x` и `n` соответственно. Однако вызов корректен, поскольку соответствующие типы совместимы с точки зрения присваивания.

Вопросы и задания

1 **Применяйте!** Даны следующие объявления

Pascal

```
function Factorial(n : integer) : integer;
var p, i : integer;
begin
     p:=1;
     for i:=1 to n do p:=p * i;
     Factorial:=p;
end;
```

C++

```
int Factorial(int n)
// Вычисление n!
{
    int p = 1;
    for (int i = 1; i <= n; i++) p = p * i;
    return p;
} // Конец Factorial
```

Назовите тип формального параметра и тип результата, возвращаемого функцией. Укажите переменные, объявленные в теле функции. Напишите программу, выводящую на экран значения $n!$ для $n = 2, 3$ и 7.

2 В каком месте основной программы содержатся объявления функций?

3 **Обратите внимание!** Прокомментируйте следующую программу:

Pascal

```
Program P98;
     { Ошибка }
function Factorial(n : 0..7) : integer;
var p, i : integer;
begin
     p:=1;
     for i:=1 to n do p:=p*i;
     Factorial:=p;
end; { Factorial }
begin
     writeln(Factorial(4));
     readln;
end.
```

C++

```
// Program P98
// Ошибка

#include <iostream>
    using namespace std;
enum par {v1 = 1, v2 = 2, v3 = 3, v4 = 4, v5 = 5, v6 = 6, v7 = 7};

int factorial(enum par n)
{
    int p = 1;
    for (int i = 1; i <= n; i++) p = p * i;
    return p;
}
int main()
{
    cout << factorial(4) << endl;
    return 0;
}
```

4 **Практикуйтесь!** Дан заголовок функции:

Pascal

```
function F(x : real; y : integer; z : char) : boolean;
```

C++

```
bool F(double x, int y, char z)
```

Какие из следующих вызовов функции корректны?

a)
```
F(3.18, 4, 'α')
```

b)
```
F(4, 4, '4')
```

c)
```
F(4, 4, 4)
```

d)
```
F(4, 3.18, 'α')
```

e)
```
F(3.18, 4, 4)
```

f)
```
F('3.18', 4, '4')
```

g)
```
F(15, 21, '3')
```

h)
```
F(15,21,3)
```

5 **Программируйте!** Напишите функцию, которая возвращает:

a) сумму действительных чисел *a*, *b*, *c*, *d*;
b) среднее арифметическое целых чисел *i*, *j*, *k*, *m*;
c) минимальное из чисел *a*, *b*, *c*, *d*;
d) количество гласных в строке символов;
e) количество согласных в строке символов;
f) корень уравнения $ax + b = 0$;
g) наименьший делитель целого числа $n > 0$, отличный от 1;
h) наибольший общий делитель натуральных чисел *a*, *b*;
i) общее наименьшее кратное натуральных чисел *a*, *b*;
j) последнюю цифру в десятичном представлении целого числа $n > 0$;
k) количество цифр в десятичном представлении целого числа $n > 0$;
l) цифра старшего разряда в десятичной записи целого числа $n > 0$;
m) количество появлений заданного символа в строке символов.

6 **Программируйте!** Даны следующие объявления:

Pascal
```
const nmax=100;
type Vector=array [1..nmax] of real;
```

C++
```
#define nmax 100
double Vector[nmax];
```

Напишите функцию, которая вычисляет:

a) сумму элементов массива;
b) среднее арифметическое элементов массива;
c) максимальный элемент массива;
d) минимальный элемент массива.

7 **Анализируйте и применяйте!** Даны следующие типы данных:

Pascal

```
type Punct=record
       x, y : real
     end;
Segment=record
       A, B : Punct
     end;
Triunghi=record
      A, B, C : Punct
     end;
Dreptunghi=record
        A, B, C, D : Punct
      end;
Cerc=record
       Centru : Punct;
       Raza : real
     end;
```

C++

```
struct Punct {double x; double y;}
struct Segment {struct Punct A; struct Punct B;}
struct Triunghi {struct Segment A; struct Segment B, struct
Segment C;}
struct Dreptunghi
 {    struct Segment A; struct Segment B;
      struct Segment C; struct Segment D;
 }
struct Cerc { struct Punct; double Raza;}
```

Напишите функцию, вычисляющую:
a) длину отрезка (segment – отрезок);
b) длину окружности (cerc – круг);
c) площадь круга;
d) площадь треугольника (triunghi – треугольник);
e) площадь прямоугольника (dreptunghi – прямоугольник).

8 **Применяйте!** (PASCAL) Переменная *A* объявлена следующим образом:

```
var A : set of char;
```

Напишите функцию, возвращающую количество символов в множество *A*.

9 **Применяйте!** (C/C++) Переменная *A* объявлена следующим образом

```
string A;
```

Напишите функцию, возвращающую количество различных символов в строке символов *A*.

10 **Применяйте!** Напишите функцию, которая вычисляет в секундах разность между двумя моментами времени, заданными в часах, минутах и секундах.

11 **Интегрируйте знания и применяйте!** Треугольник задан координатами своих вершин. Напишите функции, которые для двух заданных треугольников проверяют:
a) одинаковы ли площади данных треугольников;
b) являются ли треугольники подобными;
c) находится ли первый треугольник внутри второго.

1.3 Процедуры/Нетипизированные функции

Pascal

Общая форма текста объявления процедуры:

```
procedure P(x1, x2, ..., xn);
D;
begin
...
end;
```

Заголовок процедуры включает:

P – имя процедуры;

$x_1, x_2, ..., x_n$ – необязательный список формальных параметров.

Тело процедуры содержит:

D — локальные (опциональные) объявления, сгруппированные по тем же правилам, как и в случае функций;

begin ... end – составной оператор, в котором имя процедуры не появляется в операторе присваивания.

Процедура может возвращать несколько результатов, но не через свое имя, а через специально предназначенные для этого переменные (следующие за словом **var**) из списка формальных параметров.

Параметры из списка, вводимые через объявления вида

$$v_1, v_2, ..., v_k : t_p$$

называются ***параметрами-значениями***. Они служат для передачи значений из основной программы в процедуру.
Формальные параметры, введенные в список через описания вида

var $v_1, v_2, ..., v_k : t_p$

называются ***параметрами-переменными*** и служат для возвращения результатов из процедуры в основную программу.

Запуск процедуры *P* выполняется путем ее вызова

$P\ (a_1, a_2, ..., a_n)$

где $a_1, a_2, ..., a_n$ – список **фактических параметров**. В отличие от функции, обращение к процедуре является оператором. Операторы вызова процедур вставляются в основную программу там, где требуется выполнить соответствующую обработку данных.

Для ***параметра-значения*** в качестве фактического параметра можно использовать любое выражение соответствующего типа, в частности константу или переменную. Изменения параметров-значений не передаются за пределы подпрограммы.

Для ***параметра-переменной*** в качестве фактического параметра можно использовать только переменную. Все изменения указанных параметров будут переданы в программу, которая вызвала соответствующую процедуру.

Пример:

Pascal
```
Program P99;
 {Объявление и использование процедуры Lac }
var  a, b, c, t, q : real;

procedure Lac(r : real; var l, s : real);
 { Длина окружности и площадь круга }
 {r - радиус; l - длина; s - площадь }
const Pi=3.14159;
begin
  l:=2*Pi*r;
  s:=Pi*sqr(r);
end; {Lac }

begin
  a:=1.0;
  Lac(a, b, c);
  writeln(a:10:5, b:10:5, c:10:5);

  Lac(3.0, t, q);
  writeln(3.0:10:5, t:10:5, q:10:5);

  readln;
end.
```

Процедура `Lac` имеет три формальных параметра: `r, l, s`. Параметр `r` является параметром-значением, а `l` и `s` – параметрами-переменными.

При выполнении процедуры `Lac(a,b,c)` значение `1.0` передается формальному параметру `r`, а адреса переменных `b` и `c` передаются формальным параметрам `l` и `s`. Таким образом, последовательность операторов

```
a:=1.0;
Lac(a, b, c)
```

эквивалентна последовательности

```
b:=2*Pi*1.0;
c:=Pi*sqr(1.0)
```

Аналогично оператор

```
Lac(3.0,t,q)
```

эквивалентен последовательности

```
t:=2*Pi*3.0;
q:=Pi*sqr(3.0)
```

C++

Общая форма текста объявления нетипизированной функции:

```
void <Имя функции> (x1, x2, ..., xn)
{
 D;
...
 return;
}
```

Заголовок функции содержит:
< *Имя функции* > — имя функции;
$x_1, x_2, ..., x_n$ — необязательный список формальных параметров.
Тело функции содержит:
D – локальные (опциональные) объявления, сгруппированные по тем же правилам, как и в случае типизированных функций;
{...} — составной оператор; он не содержит каких-либо операторов присваивания относительно имени функции.

Нетипизированная функция может возвращать несколько результатов через специально назначенные переменные (переменные-указатели типа `pointer`) в списке формальных параметров.

Параметры $x_1, x_2, ..., x_n$ в списке вводятся объявлениями вида $t_1\ v_1, t_2\ v_2, ..., t_n\ v_n$, где пары $t_1\ v_1, t_2\ v_2, ..., t_n\ v_n$ формируются из <*Типа имени*> и <*Идентификатора*>.

Параметры $t_1 v_1, t_2 v_2, ..., t_n v_n$ называются ***параметрами-значениями***. Они служат для передачи значений из основной программы в функцию.

Формальные параметры, которые вносятся в список посредством объявлений формы

$t_1{*}\ v_1,\ t_2{*}\ v_2 ..., t_k\ *\ v_k$

называются **параметрами-переменными** и служат для возврата результатов из текущей функции в вызывающую функцию.

Запуск нетипизированной функции осуществляется через ее вызов

$P(a_1, a_2, ..., a_n)$

где `P` – имя функции, а $a_1, a_2, ..., a_n$ – это список фактических параметров. В отличие от вызова типизированных функций, вызов нетипизированной функции является оператором, который записывается в том месте вызывающей функции, в котором ожидается эффект от его выполнения.

Для ***параметра-значения*** в качестве фактического параметра можно использовать любое выражение соответствующего типа, в частности константу или переменную. Изменения параметров-значений не передаются за пределы подпрограммы.

Для ***параметра-переменной*** в качестве фактического параметра можно использовать только переменную. Все изменения указанных параметров будут переданы в программу, которая вызвала соответствующую функцию.

Пример:

C++
```
// Program P99
// Объявление и использование процедуры Lac

#include <iostream>
#include <math.h>
    using namespace std;
double a, b, c, t, q;

void Lac(double r, double* l, double* s)
 //Длина окружности и площадь круга: r - радиус; l - длина; s - площадь
{
 #define  Pi 3.14159
  *l = 2 * Pi * r;
  *s = Pi * pow(r, 2);
 return;
}

int main()
{
  a = 1.0;
  Lac(a, &b, &c);
  cout << a <<" "<< b<< " "<< c << endl;

  Lac(3.0, &t, &q);
  cout << 3.0 <<" "<< t<< " "<< q << endl;
}
```

Функция Lac имеет три формальных параметра: r, l, s. Параметр r является параметром-значением, а l и s – параметрами-переменными.

При выполнении процедуры Lac(a,b,c) значение 1.0 передается формальному параметру r, а адреса переменных b и c передаются формальным параметрам l и s. Таким образом, последовательность операторов

```
a = 1.0;
Lac(a, &b, &c)
```

эквивалентна последовательности

```
b = 2 * Pi * 1.0;
c = Pi * pow(1.0, 2)
```

Аналогично оператор

```
Lac(3.0, &t, &q)
```

эквивалентен последовательности

```
t = 2 * Pi * 3.0;
q = Pi * pow(3.0, 2)
```

Вопросы и задания

1 **Обратите внимание!** Чем отличаются *параметр-значение* и *параметр-переменная*?

2 **Практикуйтесь!** (PASCAL) Даны объявления:

```
var  k, m, n : integer;
     a, b, c : real;
procedure P(i : integer; var j : integer;
            x : real; var y : real);
begin
   {...}
end.
```

Какие из следующих операторов вызова корректны?

a) `P(k,m,a,b)`

b) `P(3,m,a,b)`

c) `P(k,3,a,b)`

d) `P(m,m,a,b)`

e) `P(m,k,6.1,b)`

f) `P(n,m,6,b)`

g) `P(n,m,6,20)`

h) `P(a,m,b,c)`

i) `P(i,i,i,i)`

j) `P(a,a,a,a)`

Аргументируйте ваш ответ.

3 **Практикуйтесь!** (C/C++) Даны объявления:

```
int k, m, n;
double a, b, c;

void P(int i, int* j, double x, double* y)
{
    ...
}
```

Какие из следующих операторов вызова корректны?

a) `P(k,&m,a,&b)`

b) `P(3,&m,a,&b)`

c) `P(k,3,a,&b)`

d) `P(m,&m,a,&b)`

e) `P(m,&k,6.1,&b)`

f) `P(n,&m, 6, &b)`

g) `P(n,m,6,20)`

h) `P(a, m, b, c)`

i) `P(i, &i, i, &i)`

j) `P(a, &a, a, &a)`

Аргументируйте ваш ответ.

4 **Обратите внимание!** Прокомментируйте следующую программу:

Pascal
```
Program P100;
  {Ошибка}
var    a : real;
     b : integer;
procedure P(x : real; var y : integer);
begin
  {... }
end; { P }
begin
  P(a, b);
  P(0.1, a);
  P(1, b);
  P(a, 1);
end.
```

C++
```
// Program P100
// Ошибка
#include <iostream>
    using namespace std;
double a;
int b;
void P(double x, int* y)
{... }

int main()
{
 P(a, &b);
 P(0.1, a);
 P(1, &b);
 P(a, 1);
 return 0;
}
```

5 **Проведите исследование!** Что будет выведено на экран следующей программой?

Pascal
```
Program P101;
 {Параметр-значение и параметр-переменная}
var a, b : integer;

procedure P(x : integer; var y : integer);
begin
```

```
  x:=x+1;
  y:=y+1;
  writeln('x=', x, ' y=', y);
end; {P }

begin
  a:=0;
  b:=0;
  P(a, b);
 writeln('a=', a, ' b=', b);
 readln;
end.
```

C++

```
// Program P101
// Параметр-значение и параметр-переменная
#include <iostream>
    using namespace std;
int a, b;

void P(int x, int* y)
{
 x = x + 1;
 *y = *y + 1;
 cout <<"x= "<< x<<" y= "<< *y << endl;
 return;
}

int main()
{
 a = 0;
 b = 0;
 P(a, &b);
 cout <<"a= "<< a<<" b= "<< b << endl;
 return 0;

}
```

Аргументируйте ваш ответ.

6 **Программируйте!** Напишите процедуру/нетипизированную функцию, которая:

a) вычисляет корни уравнения $ax^2 + bx + c = 0$;
b) удаляет из строки указанный в вызове символ;
c) заключает строку символов между символами “#”;
d) упорядочивает компоненты массива, содержащего до 100 действительных чисел, в порядке возрастания;
e) упорядочивает компоненты файла, содержащего целые числа, в порядке убывания;
f) вычисляет и вносит в массив простые числа меньше заданного натурального числа n.

7 **Программируйте!** Даны следующие типы данных:

Pascal
```
type Data = record
              Ziua : 1..31;
              Luna : 1..12;
              Anul : integer;
            end;
     Persoana = record
                  NumePrenume : string;
                  DataNasterii : Data;
                end;
     ListaPersoane = array [1..50] of Persoana;
```

C++
```
struct Data {
              int Ziua;
              int Luna;
              int Anul;
            };
struct Persoana { string NumePrenume;
                  struct Data DataNasterii;
                } Pers[50];
```

Напишите процедуру/нетипизированную функцию, которая получает из главной программы список лиц и возвращает:

a) фамилии и имена тех, кто родился в день *z* месяца;
b) фамилии и имена тех, кто родился в месяц *l* года
c) фамилии и имена тех, кто родился в год *a*;
d) фамилии и имена тех, чья дата рождения *z.l.a*;
e) фамилию и имя самого старшего человека;
f) фамилию и имя самого младшего человека;
g) возраст каждого человека в годах, месяцах и днях;
h) список лиц старше *v* лет;
i) список лиц в алфавитном порядке;
j) список лиц, упорядоченный согласно дате рождения;
k) список лиц одного возраста (рожденных в одном и том же году).

8 **Интегрируйте знания и применяйте!** Напишите процедуру/нетипизированную функцию, которая:

a) создает резервную копию любого текстового файла;
b) удаляет из текстового файла пустые строки;
c) пронумеровывает строки текстового файла;
d) выполняет слияние двух текстовых файлов в один;
e) выполняет слияние n ($n > 2$) текстовых файлов в один.

9 **Интегрируйте знания и применяйте!** Назовем *большими* натуральные числа, содержащие более 50 значимых цифр. Объявите тип данных для обработки больших натуральных чисел и напишите процедуры/нетипизированные функции для их сложения и вычитания.

1.4 Области видимости (PASCAL)

Тело любой программы или подпрограммы называется **блоком**. Поскольку подпрограммы включены в основную программу и, в свою очередь, могут содержать другие подпрограммы, блоки могут быть **вложенными** (включенными один в другой). Такое вложение блоков называется **блочной структурой** программы PASCAL.

В таких структурах каждому блоку *i* соответствует некоторый уровень вложенности. Основной программе соответствует уровень вложенности 0, блоку, определенному в основной программе, – уровень вложенности 1. Блоку, определенному на уровне *n*, соответствует уровень вложенности $n + 1$.

В качестве примера на *рисунке 1.2* представлена блочная структура программы P105.

Pascal

```
Program P105;                                    {nivel 0}
 { Блочная структура программы }
var a : real;
{1}

   procedure P(b : real);                {nivel 1}
   var c : real;
   {2}

      procedure Q(d : integer);   {nivel 2}
      {3}
      var c : char;
      {4}
      begin
        c:=chr(d);
        writeln('In procedura Q  c=', c);
      end; {5}

   begin
     writeln('b=', b);
     c:=b+1;
     writeln('В процедуре P  c=', c);
     Q(35);
   end; {6}

   function F(x : real) : real;          {nivel 1}
   begin
     F:=x/2;
   end;

begin
   a:=F(5);
   writeln('a=', a);
   P(a);
   readln;
end {7}.
```

Рис. 1.2. Блочная структура программы на языке PASCAL

Как правило, любой блок на языке PASCAL содержит объявления меток, переменных, функций, параметров и т.д. При описании вводится имя, которое может быть меткой, или идентификатором. Объявление, находящееся во внутреннем блоке, может переопределить имя, описанное за его пределами. Таким образом, одно и то же имя в различных частях программы может обозначать различные объекты.

Под **областью видимости** некоторого объявления понимается текст программы, в котором введенные имена обозначают объект, определенный данным объявлением. Область видимости начинается сразу же после завершения объявления и заканчивается вместе с текстом соответствующего блока. Поскольку блоки могут быть вложенными, область видимости не является непрерывной зоной в тексте программы. Область видимости объявления из некоторого вложенного блока **перекрывает** область видимости объявления, в котором участвует это же имя, находящееся во внешнем блоке.

Например, в программе `P105` областью видимости объявления **`var`** `a : real` является текст, заключенный между пунктами `{1}` и `{7}`. Область видимости объявления **`var`** `c: real` состоит из двух фрагментов текста, заключенных между `{2}`, `{3}` и `{5}`, `{6}`. Областью видимости объявления **`var`** `c : char` является текст, заключенный между `{4}` и`{5}`.

Для нахождения объекта, обозначенного некоторым именем, необходимо знать области видимости.

Например, в программе `P105` идентификатор `c` из оператора

```
c:=chr(d)
```

обозначает переменную типа `char`. Тот же самый идентификатор из оператора

```
c:=b+1
```

обозначает переменную типа `real`.

Напомним, что объявление имени функции/процедуры завершается в конце заголовка. Таким образом, область видимости такого объявления включает и тело соответствующей функции/процедуры. Это делает возможным **рекурсивный вызов**: в тело функции/процедуры можно включать вызов ее самой, так как соответствующее имя находится в области видимости. Естественно, объявление любого формального параметра является видимым только в теле соответствующей подпрограммы.

Например, область видимости объявления procedure `Q` – это текст, находящийся между пунктами `{3}` и `{6}`. Областью видимости объявления d:integer является текст, находящийся между пунктами `{3}` и `{5}`.

Вопросы и задания

1. **Объясните!** Как определяется область видимости некоторого объявления?
2. **Примените!** Определите области видимости объявлений `b : real` и `x : real` в программе `P105` (*рис. 1.2*).
3. **Примените!** Определите блочную структуру представленной ниже программы. Для каждого объявления установите область видимости и определите объекты, которые обозначают идентификаторы `c` и `x` при каждом появлении.

Pascal
```
Program P106;
  {Переопределение констант }
 const c=1;
 function F1(x : integer) : integer;
 begin
  F1:=x+c;
 end; { F1 }

 function F2(c : real) : real;
 const x=2.0;
 begin
  F2:=x+c;
 end; { F2 }
 function F3(x : char) : char;
 const c=3;
 begin
  F3:=chr(ord(x)+c);
 end; { F3 }

 begin
  writeln('F1=', F1(1));
  writeln('F2=', F2(1));
  writeln('F3=', F3('1'));
  readln;
 end.
```

Что выведет на экран данная программа?

4 **Практикуйтесь!** Определите области видимости идентификаторов `P` и `F` в программе `P105` (*рис. 1.2*).

5 **Обратите внимание!** Прокомментируйте следующую программу:

Pascal
```
Program P107;
 { Ошибка }
 var a : real;

 procedure P(x : real);
 var a : integer;
 begin
  a:=3.14;
  writeln(x+a);
 end; { P }

 begin
  a:=3.14;
  P(a);
 end.
```

6 **Объясните!** Как определить, какой объект представляет имя, появляющееся в определенном блоке программы на языке PASCAL?

1.5 Области видимости (C/C++)

Тело программы или подпрограммы называется **блоком**. В некоторых языках программирования, например в языке PASCAL, подпрограммы включаются в основную программу и, в свою очередь, могут содержать другие подпрограммы, формируя **вложенные** (включенные один в другой) блоки.

В C/C++ тело программы состоит из функций. Обычно функции описываются одна за другой. Последней описывается основная функция `main ()`. Каждое объявление внутри функции определяет некоторый объект, чаще всего переменную, который доступен только внутри этой функции. Более того, внутри функций C/C++ могут быть определены блоки, которые являются составными частями некоторых операторов, например, циклов **`for`** и **`while`**.

Определения из этих блоков вступают в силу только во время выполнения соответствующих операторов и видимы только внутри них.

Переменные, объявленные внутри функций или операторов, называются *локальными переменными*. Переменные, объявленные вне функций, называются *глобальными переменными*.

В отличие от локальных переменных, которые видимы и на которые можно ссылаться только внутри функций или операторов, в которых они определены, глобальные переменные видимы, и на них можно ссылаться в любой из функций, следующих за объявлением соответствующих переменных.

Пример 1:

Дана следующая программа:

C++

```
// Program P102
// Области видимости
#include <iostream>
#include <math.h>
    using namespace std;

  void Tipartablou(int x[], int n)
  {
      for (int i = 0; i < n; i++) cout << x[i]<<" ";

  cout << endl;
  return;
}

  bool Esteprim(int q)
  {
      int h = 0;
      for (int i = 1; i <= q; i++)
      if (q % i == 0) h++;

    if (h == 2) return true; else return false;
  }
```

```
  void Numereprime(int x[],int n)
  {
      int p[n];
      int j = 0;
      for (int i = 0; i< n; i++)
       if (Esteprim(x[i]) == true)
       {
          p[j] = x[i]; j++;
       }
       Tipartablou(p, j);
       return;
}
```

```
  int main()
  {
      int a[10] = {27, 12, 13, 8, 9, 148, 7, 23, 55, 5};
      int n = 10;
  Tipartablou(a, n);
  Numereprime(a, n);
  return 0;
}
```

В этой программе объявлены следующие функции:

`Tipartablou` – нетипизированная функция, отображающая на экране первые `n` элементов массива `x`. В этой функции объявляется локальная переменная `i`, которую можно использовать только внутри цикла for.

`Esteprim` – функция логического типа, возвращающая значение `true`, если число `q` является простым числом, и `false` в противном случае. В этой функции объявлена локальная переменная `i`, которую можно использовать только внутри цикла **`for`**, и переменная `h`, которую можно использовать в любом операторе этой функции.

`Numereprime` – нетипизированная функция, отображающая на экране только простые числа из массива `x`. В функции объявлена локальная переменная `i`, которую можно использовать только внутри цикла **`for`**, переменная `j`, которую можно использовать в любом операторе этой функции, а также массив `p[n]`, размер которого определяется значением параметра, пересылаемого в функцию.

`main` – основная функция, которая присваивает значения элементам массива `a`, отображает на экране эти значения (вызовом функции `Tipartablou`), а затем отображает только простые числа массива a (вызовом функции `Numereprime`). В основной функции определены две переменные: массив `a` и простая переменная `n` типа **`int`**.

Подчеркнем тот факт, что внутри одной и той же программы, но в разных функциях программист может объявлять переменные с одинаковым именем. Конкретная переменная, которая будет использоваться в расчетах, зависит от того, где появляется обращение к ней.

Например, одно и то же имя переменной `i` встречается в объявлениях трех функций: `Tipartablou`, `Esteprim` и `Numereprime`. Соответствующие переменные, несмотря на то что они имеют одно и то же имя `i`, являются локальными по отношению к блокам, в которых они были объявлены, и, очевидно, их значения никак не зависят друг от друга.

Вопросы и задания

1. **Объясните!** Как определяется область видимости некоторого объявления?
2. **Примените!** Определите области видимости объявлений `int j = 0;` и `int i = 0;` в функции `Numereprime` программы `P102_C/C++`.
3. **Примените!** Определите структуру следующей программы. Укажите область действия каждого объявления и определите объекты, обозначаемые каждым появлением идентификаторов `c` и `x`.

C++
```
// Program P103
// Переопределение констант
#include <iostream>
#define C 1
    using namespace std;

int F1(int x)
 {
  return x + C;
 }

 double F2(double c)
 {
    #define X 2.0
    return X + c;
 }

 char F3(char x)
 {
    #define C '3'
    return (char)(int(x) + C);
 }

 int main()
 {
  cout << "F1=" << F1(1) << endl;
  cout << "F2=" << F2(1) << endl;
  cout << "F3=" << F3('1') << endl;
 }
```

Что выведет на экран данная программа?

4. **Практикуйтесь!** Определите области видимости идентификаторов `p`, `a` и `n` в программе `P102`.
5. **Обратите внимание!** Прокомментируйте следующую программу:

C++
```
// Program P104
// Ошибка
#include <iostream>
 using namespace std;
```

```
double a;

 void P(double x)
 {
 int a;
   a = 3.14;
  cout << x + a;
 }

 int main()
 {
  a = 3.14;
  P(a);
}
```

6 **Объясните!** Как определить, какой объект обозначен некоторым именем в программе C/C++?

1.6 Связь через глобальные переменные

Pascal

Вызов подпрограммы подразумевает передачу подлежащих обработке данных вызываемой функции или процедуре. После выполнения последнего оператора подпрограммы полученные результаты необходимо возвратить на место вызова. Мы уже знаем, что подлежащие обработке данные и полученные результаты могут передаваться через параметры. Формальные параметры указываются в заголовке функции или процедуры, а фактические – в той части программы, где производится вызов.

В дополнение к способу передачи данных через параметры, современные языки программирования допускают передачу данных между программой и вызываемыми подпрограммами через глобальные переменные.

Любая переменная является локальной по отношению к подпрограмме, в которой она объявлена. Переменная является **глобальной по отношению к подпрограмме**, если она объявляется в основной программе или во внешней подпрограмме, без повторного объявления в рассматриваемой подпрограмме. Так как глобальные переменные известны как в подпрограмме, так и за ее пределами, они могут использоваться для передачи данных, подлежащих обработке, и для возвращения результатов.

Пример:

Pascal

```
Program P108;
  {Связь через глобальные переменные  }
 var a,             {переменная глобальная в P }
     b : real;      {переменная глобальная в P, F }

 procedure P;
 var c : integer;  {переменная локальная в  P }
 begin
    c:=2;
    b:=a*c;
 end; { P }
```

```
function F : real;
var a : 1..5;     {переменная локальная в F }
begin
   a:=3;
   F:=a+b;
end; { F }

begin
   a:=1;
   P;
 writeln(b);     {на экран выводится 2.0000000000E+00 }
 writeln(F);     {на экран выводится 5.0000000000E+00 }
 readln;
end.
```

C++

```
// Program P108
// Связь через глобальные переменные
#include <iostream>
    using namespace std;
double a, b; // глобальные переменные
void P()
{
    int c = 2; // переменная локальная в  P
    b = a * c;
    return;
}
double F()
{
    int a = 3; // переменная локальная в  F
    return a + b;
}
int main()
{
    a = 1;
    P();
    cout << b << endl; // на экран выводится 2
    cout << F() << endl; // на экран выводится 5
    return 0;
}
```

Данные, подлежащие обработке, передаются процедуре/нетипизированной функции P через глобальную переменную a. Результат, полученный процедурой/нетипизированной функцией, возвращается на место вызова через глобальную переменную b. Значение аргумента функции F передается через глобальную переменную b. Отметим, что переменная a является локальной для F и не может использоваться для передачи данных в эту функцию.

Как правило, связь через глобальные переменные используется в тех случаях, когда несколько подпрограмм обрабатывают одни и те же данные. Например, функции, аргументы которых относятся к типу массив; процедуры, которые обрабатывают массивы или файлы с данными о работниках, учениках и т.д.

Вопросы и задания

1. Объясните термины *переменная глобальная* по отношению к некоторой подпрограмме и *переменная локальная* в некоторой подпрограмме.
2. **Примените!** (PASCAL) Назовите глобальные и локальные переменные, описанные в программе P105 (*рис. 1.2*).
3. **Проведите исследование!** Может ли локальная переменная одновременно являться и глобальной по отношению к какой-либо подпрограмме?
4. **Применяйте!** Назовите локальные и глобальные переменные, описанные в следующей программе. Что выведет на экран данная программа?

Pascal

```
Program P109;
   { Связь через глобальные переменные }
 var a : integer;

 procedure P;
 var b, c, d : integer;

 procedure Q;
 begin
    c:=b+1;
 end; { Q }

 procedure R;
 begin
    d:=c+1;
 end; { R }
 begin
   b:=a;
   Q;
   R;
   a:=d;
 end; { P }

 begin
   a:=1;
   P;
   writeln(a);
   readln;
 end.
```

C++

```
// Program P109
// Связь через глобальные переменные
#include <iostream>
    using namespace std;

 int a, b;

void R()
```

```
{
    a++;
    cout << " a in R() = "<< a << endl;
    return;
}

void Q()
{
   b = a + 2;
   cout << " a in Q() = "<< a << endl;
   cout << " b in Q() = "<< b << endl;
   return;
}

 void P()
{
    int b, d = 6;
    b = a;
    Q();
    R();
    a = d;
    cout << " b in P() = "<< b << endl;
    cout << " a in P() = "<< a << endl;
    return;
}

int main()
{
  a = 1;
  P();
  cout << " a in main() = "<< a << endl;
  cout << " b in main() = "<< b << endl;
  return 0;
}
```

5 **Программируйте!** Даны объявления:

Pascal
```
Type Ora=0..23;
     Grade=-40..+40;
     Temperatura=array [Ora] of Grade;
```

C++
```
#define Ora 24
int Grade;
int Temperatura[Ora];
```

Компоненты переменной типа `Temperatura` представляют собой значения температуры, измеряемой каждый час в течение 24 часов. Напишите процедуру/нетипизированную функцию, которая:

a) находит максимальное и минимальное значения температуры;
b) находит час (часы), когда была зарегистрирована максимальная температура;
c) записывает в текстовый файл час (часы), когда была зарегистрирована минимальная температура;

Связь с соответствующими процедурами осуществляется через глобальные переменные.

6 **Интегрируйте знания и применяйте!** Дан произвольный текстовый файл. Напишите функцию, которая:

a) возвращает количество строк в файле;
b) находит количество гласных в тексте;
c) находит количество слов в тексте (слова представляют собой последовательности символов, разделенных пробелами или символами конца строки);
d) возвращает среднюю длину строк текста;
e) возвращает среднюю длину слов текста;
f) возвращает число знаков препинания в тексте.

Связь с соответствующими процедурами/нетипизированными функциями осуществляется через глобальные переменные.

1.7 Побочные эффекты

Любая функция возвращает единственный результат – значение функции. Как правило, значения аргументов передаются функции через параметры-значения, а результат возвращается на место вызова через имя функции. Помимо этого, современные языки программирования допускают передачу аргументов через глобальные переменные и параметры-переменные.

Под **побочным эффектом** понимается присваивание (внутри функции) некоторого значения глобальной переменной или формальному параметру-переменной. Побочные эффекты могут непредвиденным образом влиять на ход программы, усложняя тем самым процесс ее отладки.

Ниже представлены примеры программ, в которых используются функции с побочными эффектами.

Pascal

```
Program P110;
 {Побочный эффект - присваивание значения глобальной переменной}
 var a : integer; { глобальная переменная }

 function F(x : integer) : integer;
 begin
   F:=a*x;
   a:=a+1;      {присваивание, влекущее за собой побочный эффект }
 end; { F }
 begin
   a:=1;
   writeln(F(1));  { на экран выводится 1 }
   writeln(F(1));  { на экран выводится 2 }
   writeln(F(1));  { на экран выводится 3 }
   readln;
 end.
```

C++

```
// Program P110
// Побочный эффект - присваивание значения глобальной переменной

#include <iostream>
    using namespace std;

 int a; // глобальная переменная

 int F(int x)
 {
   int tmp = a * x;
   a++;  // присваивание, влекущее за собой побочный эффект
   return tmp;
 }
 int main()
{
    a = 1;
    cout << F(1) << endl;  // на экран выводится 1
    cout << F(1) << endl;  // на экран выводится 2
    cout << F(1) << endl;  // на экран выводится 3
}
```

В программе P110 функция F возвращает значение выражения a*x. Наряду с этим операция присваивания a:=a+1 / a++ изменяет значение глобальной переменной a. Таким образом, для одного и того же значения 1 аргумента x функция возвращает различные результаты, что противоречит обычному определению функции.

Pascal

```
Program P111;
 {Побочный эффект - присваивание формальному параметру }
 var a : integer;

 function F(var x : integer) : integer;
 begin
     F:=2*x;
     x:=x+1;    { присваивание, влекущее за собой побочный эффект }
 end; { F }

 begin
   a:=2;
   writeln(F(a));     { на экран выводится 4 }
   writeln(F(a));     { на экран выводится 6 }
   writeln(F(a));     { на экран выводится 8 }
   readln;
 end.
```

C++
```
// Program P111
// Побочный эффект - присваивание формальному параметру

#include <iostream>
    using namespace std;
 int a; // глобальная переменная

 int F(int* x)
 {
   *x = 2 * (*x);
   return *x++;
 }
 int main()
{
 a = 2;
 F(&a);
 cout << a << endl; // на экран выводится 4
 F(&a);
 cout << a << endl;   // на экран выводится 8
 F(&a);
 cout << a << endl;   // на экран выводится 16
}
```

В программе P111 функция F возвращает значение выражения 2*x. Так как x – формальный параметр-переменная, то оператор присваивания x:=x+1изменяет значение фактического параметра, указываемого при вызове, т.е. значение переменной a из основной программы. Тот факт, что идентичные по тексту обращения F(a), F(a) и F(a) возвращают различные результаты, может привести к осложнениям в процессе отладки.

Для процедур/нетипизированных функций присваивание значений глобальным переменным приводит к побочным эффектам, аналогичным тем, которые обсуждались для таких присваиваний в случае функций. Поскольку стандартным способом возврата результатов из процедуры/нетипизированной функции являются формальные параметры-переменные, присваивание значений таким параметрам не считается побочным эффектом.

Побочные эффекты приводят к отклонениям от стандартного процесса связи, в котором участвующие переменные явно обозначаются как формальные параметры в объявлении и фактические параметры в вызове. Последствия побочных эффектов могут распространяться на область видимости глобальных объявлений и влиять на аналогичные объявления, появляющиеся при выполнении других подпрограмм. В таких условиях использование глобальных переменных становится рискованным. Поэтому при разработке сложных программ будут применяться следующие рекомендации:

1. Связь функций с вызывающей средой будет осуществляться путем передачи данных функции через значение формальных параметров и возврата единственного результата через имя функции.
2. Связь процедур/нетипизированных функций с вызывающей средой будет осуществляться путем передачи данных через формальные параметры-значения или параметры-переменные и возврата результатов через формальные параметры-переменные.
3. Глобальные переменные могут использоваться для передачи данных в подпрограммы, но их значения не должны ими изменяться.

Вопросы и задания

1. **Объясните!** Каковы причины возникновения побочных эффектов? К каким последствиям они могут привести?
2. **Проведите исследование!** Определите, что выведут на экран следующие программы?

Pascal

```
Program P112;
  {Побочные эффекты }
 var a, b : integer;

  function F(x : integer) : integer;
   begin
   F:=a*x;
   b:=b+1;
   end; { F }

  function G(x : integer) : integer;
   begin
     G:=b+x;
     a:=a+1;
   end; { G }
begin
 a:=1; b:=1;
 writeln(F(1));
 writeln(G(1));
 writeln(F(1));
 writeln(G(1));
 readln;
end.
```

C++

```
// Program P112
// Побочные эффекты
#include <iostream>
    using namespace std;

 int a, b;

 int F(int  x)
 {
   b++;
   return a + x;
 }
```

```
int G(int  x)
 {
   a++;
   return b + x;
 }

 int main()
{
 a = 1; b = 1;
    cout << F(1) << endl;
    cout << G(1) << endl;
    cout << F(1) << endl;
    cout << G(1) << endl;
}
```

Pascal

```
Program P113;
 {Побочные эффекты }
 var a : integer;
     b : real;

  function F(var x : integer) : integer;
   begin
     F:=x;
     x:=x+1;
   end; { F }

  procedure P(x,y:integer; var z:real);
   begin
     z:=x/y;
   end; { P }

begin
 a:=1;
 P(F(a), a, b);
 writeln(a, '   ', b);
 readln;
 end.
```

C++

```
// Program P113
// Побочные эффекты

#include <iostream>
    using namespace std;

 int a;
 double b;

 int F(int*  x)
 {
   *x = *x + 2;
   return *x;
 }

 void P(int  x, double y, double* z)
 {
   *z = x / y;
 }

 int main()
{
 a = 1;
 P(F(&a), a / 2.0, &b);
    cout << a <<" "<< b <<  endl;
}
```

Pascal

```
Program P114;
  {Побочные эффекты }
 var a, b : real;

  procedure P(var x, y : real);
  {Обмен значениями между переменными x и y }
   begin
     a:=x;
     x:=y;
     y:=a;
   end; { P }
```

```
begin
 a:=1; b:=2;
 P(a, b);
 writeln(a, b);
 a:=3; b:=4;
 P(a, b);
 writeln(a, b);
 readln;
end.
```

C++

```
// Program P114
// Побочные эффекты

#include <iostream>
    using namespace std;
double a, b;

void P(double* x, double* y)
  // Обмен значениями между переменными x и y
{
 a = *x;
 *x = *y;
 *y = a;
}

int main()
{
 a = 1; b = 2;
 P(&a, &b);
 cout << a <<" " << b << endl;
 a = 3; b = 4;
 P(&a, &b);
 cout << a <<" " << b << endl;
}
```

3 **Объясните!** Как можно избежать побочных эффектов?

1.8 Рекурсия

Рекурсией называется прямое или косвенное обращение подпрограммы к самой себе. Подпрограмма, которая вызывает саму себя, называется **рекурсивной**.

Например, функция *факториал*

$$f(n) = \begin{cases} 1, \text{ если } n = 0; \\ n \cdot f(n-1), \text{ если } n > 0 \end{cases}$$

может быть представлена на некотором языке высокого уровня, исходя из ее определения, в следующем виде:

Pascal

```
function F(n : integer) : integer;
begin
   if n=0 then F:=1
   else F:=n*F(n-1)
end; {F}
```

C++

```
int F(int n)
{
   if (n == 0) return 1;
   else return n * F(n - 1);
}
```

Вызов `F(7)` влечет за собой ряд обращений к функции `F` с фактическими параметрами `6, 5, ..., 2, 1, 0`:

```
F(7) -> F(6) -> F(5) -> ... -> F(1) -> F(0).
```

При вызове `F(0)` вычисляется непосредственное значение функции, что останавливает процесс повторений; после этого следует процесс возврата и вычисление значений функции `F` для аргументов `1, 2, ..., 6, 7`, последнее вычисленное значение `F(7)` возвращается на место первого вызова функции.

Функция

$$fib(n) = \begin{cases} 0, \text{ если } n = 0; \\ 1, \text{ если } n = 1; \\ fib(n-1) + fib(n-2), \text{ если } n > 1 \end{cases}$$

определяет числа *Фибоначчи* (Fibonacci). Следуя данному определению, получаем:

Pascal

```
function Fib(n:integer):integer;
begin
 if n=0 then Fib:=0
 else if n=1 then Fib:=1
      else Fib:=Fib(n-1)+Fib(n-2)
end; {Fib}
```

C++
```
int Fib(int n)
{
 if ( n==0) return 0;
   else if (n == 1) return 1;
     else return Fib(n - 1) + Fib(n - 2);
}
```

Каждый вызов функции Fib для n > 1 влечет за собой два вызова `Fib(n-1), Fib(n-2)` и т.д., например:

```
Fib(4) ->
    Fib(3), Fib(2) ->
        Fib(2), Fib(1), Fib(1), Fib(0) ->
            Fib(1), Fib(0).
```

Из данных примеров видно, что рекурсия используется для того, чтобы запрограммировать повторяющиеся вычисления. Повторение осуществляется с помощью подпрограммы, внутри которой есть вызов самой себя: когда процесс выполнения программы достигает соответствующего места, вызывается новое выполнение данной процедуры и т.д.

Очевидно, любая рекурсивная подпрограмма должна содержать условие остановки процесса повторения. Например, в случае функции factorial процесс повторений останавливается, когда n принимает значение 0; в случае функции `Fib` процесс останавливается, когда n принимает значения 0 или 1.

При любом вызове подпрограммы в память компьютера заносится следующая информация:

– текущие значения параметров, передаваемых через значение;
– местонахождение (адреса) параметров-переменных;
– адрес возврата, т.е. адрес оператора, который следует за оператором вызова.

Таким образом, при рекурсивном вызове занимаемая область памяти увеличивается очень быстро, что ведет к риску переполнения памяти компьютера. Этого можно избежать путем замены рекурсии на итерацию (операторы **for**, **while**, **repeat**). В качестве примера представим нерекурсивную форму функции для вычисления факториала:

Pascal
```
function F(n: Natural): Natural;
var i, p : Natural;
begin
   p:=1;
   for i:=1 to n do p:=p*i;
   F:=p;
end; {F}
```

C++
```
int F(int n)
{
 int i, p = 1;
 for (i = 1; i <= n; i++)
     p = p * i;
 return p;
}
```

Рекурсия особенно полезна в тех случаях, когда написание нерекурсивных алгоритмов является очень сложным: перевод программ с языка программирования высокого уровня на машинно-ориентированный язык, компьютерная графика, распознавание изображений и т.д.

Вопросы и задания

1. **Объясните!** Как выполняется рекурсивная подпрограмма? Какая информация заносится в память компьютера при выполнении рекурсивного вызова?
2. **Обратите внимание!** В чем разница между рекурсией и итерацией?
3. **Интегрируйте знания и применяйте!** Напишите нерекурсивную функцию *Fibonacci.*
4. **Программируйте!** Напишите рекурсивную подпрограмму, которая:
   a) вычисляет сумму $S(n) = 1 + 3 + 5 + \ldots + (2n - 1)$;
   b) вычисляет произведение $P(n) = 1 \times 4 \times 7 \times \ldots \times (3n - 2)$;
   c) инвертирует строку символов (переставляет символы в обратном порядке);
   d) вычисляет произведение $P(n) = 2 \times 4 \times 6 \times \ldots \times 2n$.
5. **Экспериментируйте!** Напишите программу, которая вводит с клавиатуры натуральные числа *m, n* и выводит на экран значения функции *Акерманна* (Ackermann):

$$a(m, n) = \begin{cases} n+1, \text{ если } m = 0; \\ a(m-1,1), \text{ если } n = 0; \\ a(m-1), a(m, n-1)), \text{ если } m > 0 \text{ и } n > 0. \end{cases}$$

   Вычислите *a*(*0, 0*), *a*(*1, 2*), *a*(*2, 1*) *и a*(*2, 2*). Попробуйте вычислить *a*(*4, 4*) и *a*(*10, 10*). Объясните сообщения, выводимые на экран.
6. **Программируйте!** Дано объявление:

Pascal
```
type Vector=array [1..20] of integer;
```

C++
```
int Vector[20];
```

   Напишите рекурсивную подпрограмму, которая:
   a) выводит на экран элементы массива;
   b) вычисляет сумму элементов массива;
   c) переставляет элементы массива в обратном порядке;
   d) вычисляет сумму положительных элементов массива;
   e) проверяет, есть ли среди элементов массива хотя бы один отрицательный;
   f) вычисляет произведение отрицательных элементов массива;
   g) проверяет, есть ли среди элементов массива хотя бы один, равный заданному числу.

7 **Создавайте!** Перепишите следующую функцию в нерекурсивном виде:

Pascal
```
function S(n:Natural):Natural;
 begin
   if n=0 then S:=0
   else S:=n+S(n-1)
 end; {S}
```

C++
```
int S(int n)
 {
   if (n == 0) return 0;
   else return n + S(n - 1);
 }
```

8 **Интегрируйте знания и применяйте!** Напишите рекурсивную функцию, которая возвращает значение `true`, если строка символов s соответствует следующему определению:

<Число> ::= *<Цифра>* | *<Цифра>* *<Число>*

Указание. Рекурсивная форма такой функции непосредственно вытекает из металингвистической формулы:

Pascal
```
function N(s : string) : boolean;
var  i : integer;
   p : boolean;
begin
   p:=(s<>'');
   for i:=1 to length(s) do
       p:=p and (s[i] in ['0'..'9']);
   N:=p;
end;
```

C++
```
bool N(char* S[])
{
   for (int i = 0; i < s.length(); i++)
   if (s[i] <'0" || s[i] >'9") return false;
   return true;
}
```

Нерекурсивный вариант следует из определения

<Число> ::= *<Цифра>* {*<Цифра>*}

9 **Интегрируйте знания и применяйте!** Даны следующие металингвистические формулы:

<Число> ::= *<Цифра>* {*<Цифра>*}
<Знак> ::= + | –
<Выражение> ::= *<Число>* | *<Выражение>* *<Знак>* *<Выражение>*

Напишите рекурсивную функцию, которая возвращает значение `true`, если строка символов s соответствует определению лексической единицы *<Выражение>*.

1.9 Синтаксис объявлений и вызовов подпрограмм

Pascal

В общем случае описание функций на языке PASCAL осуществляется с помощью следующих металингвистических формул:

<Функция> ::= *<Заголовок функции>; <Блок>*

| *<Заголовок функции>; <Директива>*
| **function** *<Идентификатор>; <Блок>*

<Заголовок функции> ::=

function *<Идентификатор>* [*<Список формальных параметров>*]: *<Идентификатор>*

Синтаксические диаграммы представлены на *рисунке 1.3*.

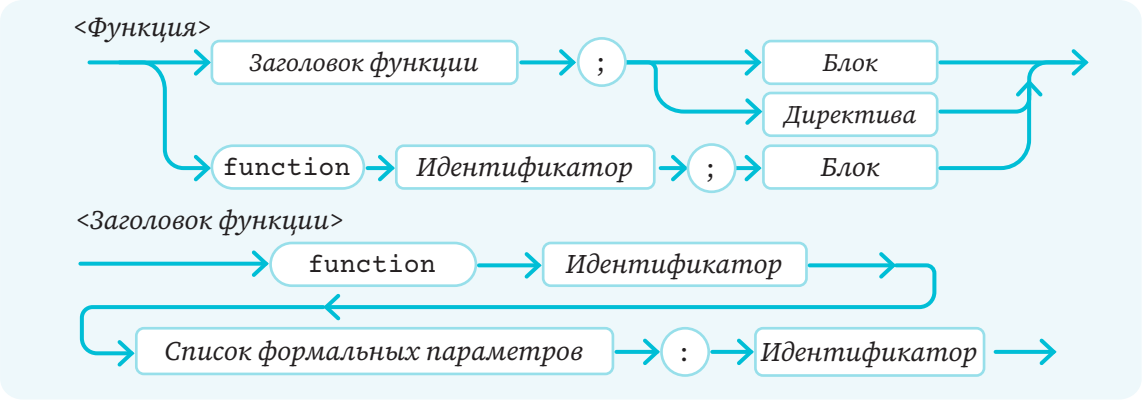


Рис. 1.3. Синтаксис объявления функции

Процедуры на языке PASCAL описываются с помощью следующих формул:

<Процедура>::=*<Заголовок процедуры>;<Блок>* | *<Заголовок процедуры>;<Директива>*

| **procedure** *<Идентификатор>;<Блок>*

<Заголовок процедуры> := **procedure** *<Идентификатор>* [*<Список формальных параметров>*]

Синтаксические диаграммы представлены на *рисунке 1.4*.

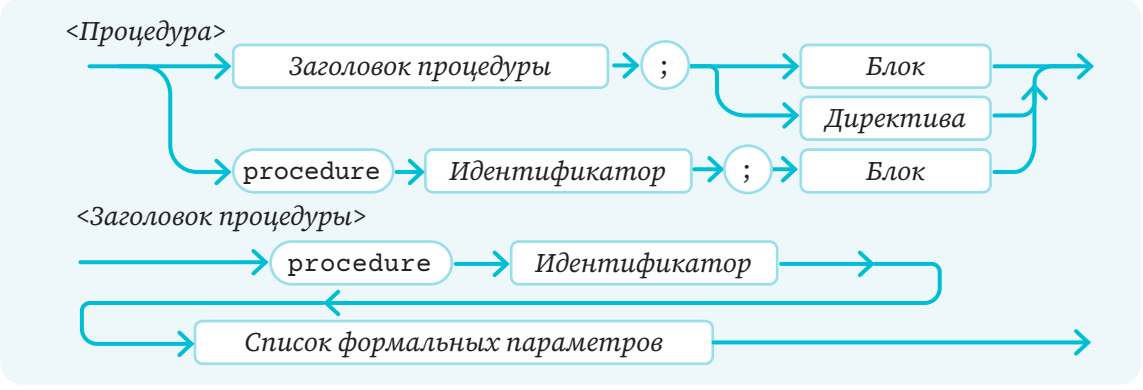


Рис. 1.4. Синтаксис объявления процедуры

Список формальных параметров имеет синтаксис:

<*Список формальных параметров*> ::=

(<*Формальный параметр*> {; <*Формальный параметр*>})

<*Формальный параметр*> ::=

[**var**] <*Идентификатор*> {, <*Идентификатор*>} : <*Идентификатор*>

| <*Заголовок функции*>

| <*Заголовок процедуры*>

Синтаксическая диаграмма представлена на *рисунке 1.5*.

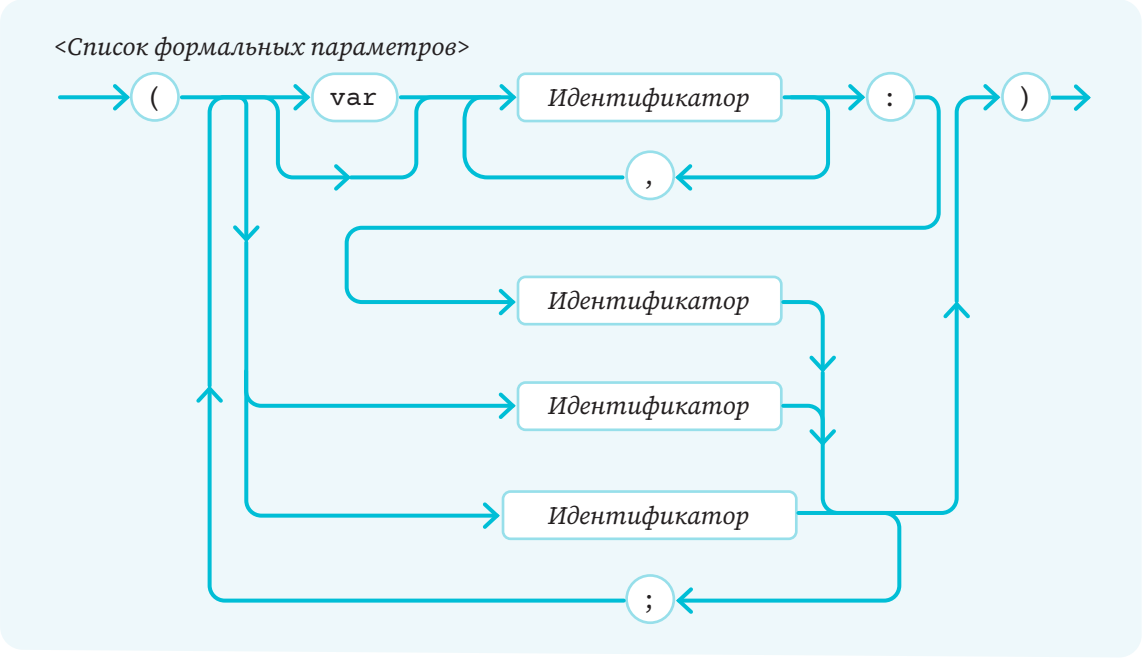


Рис. 1.5. Синтаксическая диаграмма <*Список формальных параметров*>

Отметим, что при отсутствии ключевого слова **var** идентификаторы из списка указывают **параметры-значения**. Слово **var** указывает **параметры-переменные**. Заголовок некоторой функции (процедуры) указывает на **параметр-функцию** (процедуру). В Turbo PASCAL такие параметры описываются явно, относятся к **процедурному типу** и имеют вид параметров-значений. Язык PASCAL расширяет обычный смысл понятия функции, допуская возврат значений не только через имя функции, но и через параметры-переменные. Оператор **вызова функции** имеет вид:

<*Вызов функции*> ::= <*Имя функции* > [<*Список фактических параметров*>]

а оператор **вызова процедуры**:

<*Вызов процедуры*> ::= <*Имя процедуры* > [<*Список фактических параметров*>]

Фактические параметры описываются с помощью формул:

<*Список фактических параметров*> ::=

(<*Фактический параметр*> {,<*Фактический параметр*>})

<*Фактический параметр*> ::= <*Выражение*> | <*Переменная*> | <*Имя функции*> | <*Имя процедуры*>

Синтаксическая диаграмма представлена на *рисунке 1.6.*

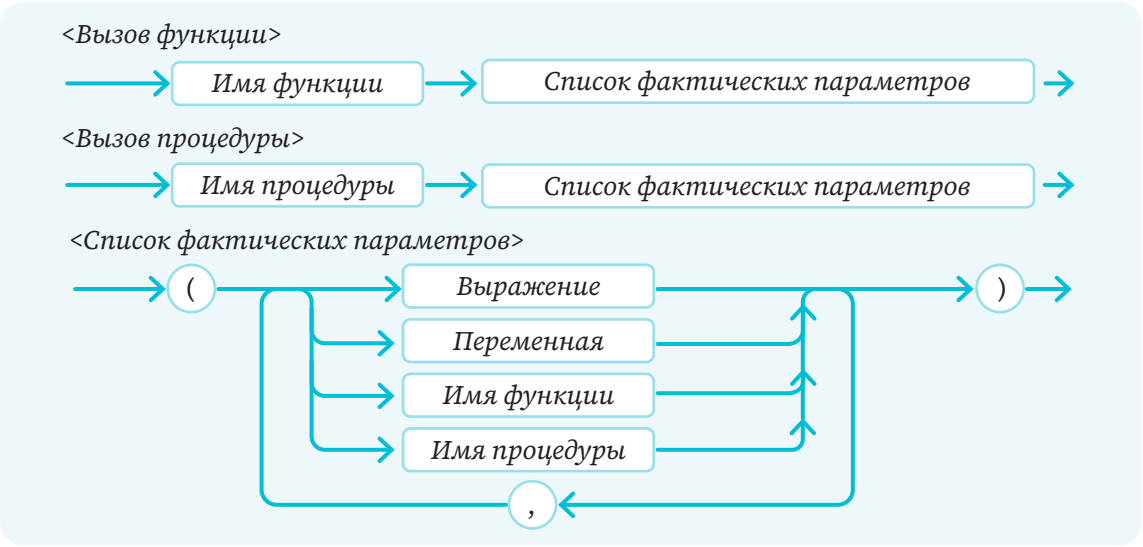


Рис. 1.6. Синтаксическая диаграмма вызова функции и процедуры

Связь между фактическими и формальными параметрами определяется позицией, которую они занимают в этих двух списках.

В случае **параметра-значения** в качестве фактического параметра может использоваться любое выражение, в частности константа или переменная. Соответствующее выражение должно быть совместимым с точки зрения присваивания с типом формального параметра. Изменения параметров-значений не передаются за пределы подпрограммы.

В случае **параметра-переменной** в качестве фактических параметров могут использоваться только переменные. Изменения параметров-переменных передаются за пределы подпрограммы.

В случае **параметра-функции** (**процедуры**) в качестве фактического параметра может использоваться любое имя функции (процедуры), заголовок которой имеет вид, указанный в списке формальных параметров.

C++

В общем случае описание функции на языках C/C++ осуществляется с помощью следующих металингвистических формул:

<*Функция*> ::= <*Заголовок функции*> <*Блок*>

<*Заголовок функции*> ::= <*Тип*> <*Идентификатор*> ([<*Список формальных параметров*>])

<*Тип*> ::= **`void`** | **`int`** | **`long`** | **`float`** | **`double`** | **`long double`** | **`char`** | **`*`**

Примечание. Множество типов данных, используемых в языках C/C++, не позволяет полностью описать их в рамках достаточно компактной металингвистической формулы. В дидактических целях в приведенную выше металингвистическую формулу включены только часто используемые типы.

Список формальных параметров объявляется с помощью формул:

<*Список формальных параметров*> ::= <*Формальный параметр*> {, <*Формальный параметр*>}

<*Формальный параметр*> ::=

<*Тип*> [**`*`** | **`&`**] <*Идентификатор*> | <*Заголовок функции*>

{, <*Тип*> [**`*`** | **`&`**] <*Идентификатор*>} | <*Заголовок функции*>

Вызов функции, так же как и **оператор вызова функции**, имеет синтаксис:
<*Вызов функции*> ::= <*Имя функции* > ([<*Список фактических параметров*>])

Список фактических параметров объявляется с помощью формул:
<*Список фактических параметров*> ::= <*Фактический параметр*> {,<*Фактический параметр*>}
<*Фактический параметр*> ::=<*Выражение*> | <*Переменная*> | <*Вызов функции*>
Связь между фактическими и формальными параметрами определяется позицией, которую они занимают в этих двух списках.

Внимание! В отличие от языка PASCAL, который был разработан для обучения основам алгоритмизации и программирования в школе, языки C/C++ были предназначены для профессионального программирования. Следовательно, синтаксис и семантика языков C/C++ гораздо сложнее, чем у языка PASCAL. Это обстоятельство делает невозможным включение в школьный учебник всех металингвистических формул и соответствующих синтаксических диаграмм. Для ознакомления с этими формулами и диаграммами мы рекомендуем тем ученикам, которые в будущем планируют стать профессиональными программистами, обратиться к системам поддержки сред программирования C/C++.

Вопросы и задания

1. **Объясните!** (PASCAL) Когда используется объявление вида
   `function` <*Идентификатор*>; <*Блок*>?

2. **Объясните!** (C/C++) Дана металингвистическая формула, описывающая заголовок некоторой функции C/C++. Какой тип функции описывается формулой
   `void` <*Идентификатор*> ([<*Список формальных параметров* >])?

3. **Объясните!** (C/C++) Даны металингвистические формулы, описывающие вызовы функций:
   a) <*Заголовок функции* > ::= <*Тип*> <*Идентификатор*> [<*Список формальных параметров*>]
   <*Список формальных параметров*> ::= (<*Формальный параметр*> {, <*Формальный параметр*>})
   b) <*Заголовок функции* > ::= <*Тип*> <*Идентификатор*>([<*Список формальных параметров*>])
   <*Список формальных параметров*> ::= (<*Формальный параметр*> {, <*Формальный параметр*>})

   Заметим, что в обеих металингвистических формулах список формальных параметров является опциональным. Чем отличаются соответствующие вызовы в случаях отсутствия формальных параметров?

4. **Исследуйте!** (PASCAL) Укажите на синтаксических диаграммах (*рис. 1.3* и *1.5*) пути, которые соответствуют описаниям функций из программы `P106`, параграф 1.4.

5. **Объясните!** В чем разница между параметром-значением и параметром-переменной?

6. **Исследуйте!** Укажите на синтаксических диаграммах (*рис. 1.4* и *1.5*) пути, которые соответствуют описаниям процедур в программе `P101`, параграф 1.3.

7. **Исследуйте!** (PASCAL) Укажите на синтаксических диаграммах (*рис. 1.3 – 1.6*) пути, которые соответствуют описанию и вызову подпрограмм в программе `P105`, параграф 1.4.

II
Раздел

Методы программирования

2.1 Сложность алгоритмов

Практическая ценность компьютерных программ в значительной степени зависит от сложности используемых в них алгоритмов. Напомним, что алгоритм представляет собой конечную последовательность известных операций (инструкций, команд), которые выполняются в строго определенном порядке, обеспечивая этим решение определенной задачи.

Известно, что один и тот же алгоритм может быть описан различными способами: логическими схемами, математическими формулами, текстом, записанным на привычном языке общения, и, наконец, с помощью языков программирования. В данном учебнике изучаемые алгоритмы будут описываться при помощи средств языков PASCAL и C/C++: операторов, функций, процедур и программ, которые могут быть выполнены на компьютере.

Сложность алгоритма характеризуется *объемом требуемой памяти* и *временем* (*продолжительностью*) *его выполнения*. Методы оценивания этих показателей изучаются в специальном разделе информатики, который называется **анализ алгоритмов**. В рамках этого раздела используются следующие обозначения:

n – натуральное число, характеризующее размер (объем) данных на входе изучаемого алгоритма. В большинстве случаев n представляет собой количество элементов обрабатываемого множества, степень решаемого уравнения, количество элементов обрабатываемого массива и т.п.

V(n) – объем внутренней памяти, необходимой для хранения используемых алгоритмом данных;

T(n) – время, необходимое для выполнения алгоритма. Обычно в нем не учитываются операции по вводу исходных данных и выводу результатов.

Очевидно, что объем памяти *V(n)* и время выполнения *T(n)* зависят, в первую очередь, от характеристики n входных данных. Указанная зависимость подчеркивается и тем фактом, что *V* и *T* представлены в виде функций от аргумента *n*.

Практическая применимость алгоритма возможна лишь тогда, когда расход памяти и требуемое время не превышают ограничений, накладываемых на них средой программирования и техническими характеристиками используемого компьютера.

Например, предположим, что для решения некоторой задачи существуют два различных алгоритма, обозначенных как A_1 и A_2. Объем памяти и время выполнения алгоритма A_1 составляют:

$$V_1(n) = 100n^2 + 4;$$
$$T_1(n) = n^3 \cdot 10^{-3},$$

а для алгоритма A_2:

$$V_2(n) = 10n + 12;$$
$$T_2(n) = 2^n \cdot 10^{-6}.$$

В приведенных формулах объем памяти измеряется в байтах, а время – в секундах.

Объем памяти и время выполнения алгоритмов A_1, A_2 для различных величин n представлены в *таблице 2.1*.

Таблица 2.1. Сложность алгоритмов A_1 и A_2

n	10	20	30	40	50
$V_1(n)$	9,77 Кбайт	39,07 Кбайт	87,89 Кбайт	156,25 Кбайт	244,14 Кбайт
$V_2(n)$	112 байтов	212 байтов	312 байтов	412 байтов	512 байтов
$T_1(n)$	1 секунда	8 секунд	27 секунд	64 секунды	125 секунд
$T_2(n)$	0,001 секунды	1,05 секунд	≈18 минут	≈13 дней	≈36 лет

Из *таблицы 2.1* видим, что алгоритм A_2 практически неприменим для входных данных с характеристикой n>30. Для таких данных время выполнения алгоритма A_1 намного меньше, но объем требуемой памяти $V_1(n)$ может превысить ограничение, накладываемое средой программирования (64 *Кбайта* для статических переменных программ в Turbo PASCAL).

Определение объема памяти $V(n)$ и времени выполнения $T(n)$ представляет особый интерес на этапе разработки соответствующих алгоритмов и программ. Очевидно, что именно на этом этапе можно заранее исключить из рассмотрения те алгоритмы, которые потребуют слишком больших объемов памяти или неприемлемого времени выполнения.

Отметим, что постоянное увеличение внутренней памяти современных компьютеров направляет основное внимание информатиков на изучение времени выполнения алгоритмов, или, другими словами, на их **временную сложность**.

Вопросы и задания

1. **Систематизируйте свои знания!** Объясните термин *сложность алгоритма*. Назовите показатели, характеризующие сложность алгоритмов.
2. **Проанализируйте!** Как Вы считаете, какие факторы влияют на сложность алгоритма?
3. **Анализируйте и применяйте!** От чего зависит расход памяти и время выполнения алгоритма? Когда возможно практическое применение алгоритма?
4. **Проведите исследование!** Алгоритмы A_1 и A_2 (смотрите *таблицу 2.1*) будут выполняться в среде программирования Turbo PASCAL, Free Pascal, C/C++. Как вы считаете, Какой из алгоритмов, по вашему мнению, нужно использовать в случае входных данных с характеристикой: *a)* n = 10; *b)* n = 20; *c)* n = 30? Для каких значений n алгоритм A_1 может быть использован в среде программирования, в которой вы работаете?

5 **Проведите исследование!** Сложность некоторого алгоритма, обозначенного как A_3, описывается формулами:

$$V_3(n) = 600n^3 + 18;$$
$$T_3(n) = 3^n \cdot 10^{-2}.$$

Для каких значений n алгоритм A_3 может выполняться в среде программирования, в которой вы работаете?

6 **Интегрируйте знания и применяйте!** Определите размер входных данных наиболее репрезентативных алгоритмов, разработанных вами в процессе изучения языка программирования высокого уровня.

2.2 Оценка объема памяти

Оценка объема памяти *V(n)* может быть осуществлена путем суммирования количества байтов, занимаемых каждой из переменных программы. Количество байтов, занимаемых переменной зависит от конкретной реализации языка. Так, в наиболее распространенных средах программирования память для переменных выделяется в соответствии с *таблицей 2.2*.

Таблица 2.2. Выделение внутренней памяти

Тип переменной PASCAL (C/C++)	Количество байтов		
	Turbo PASCAL	Free PASCAL	C/C++
`Integer (int)`	2	2 или 4, в зависимости от операционной системы	2 или 4, в зависимости от операционной системы
`Longint (int)`	4	4	8
`Real (float)`	6	4 или 8, в зависимости от операционной системы	4/8
`Boolean (bool)`	1	1	-/1
`Char (char)`	1	1	1
Перечисляемый тип (`enum`)	1	1 или 2, в зависимости от количества значений	4
Интервальный тип, только в языке PASCAL	Согласно базовому типу	Согласно базовому типу	–
Ссылочный тип (&)	4	4 или 8, в зависимости от операционной системы	4 или 8, в зависимости от операционной системы
`Pointer (*)`	4	4 или 8, в зависимости от операционной системы	Не фиксировано, зависит от типа данных, на которые указывет ссылка

Для структурированных типов данных объем памяти, необходимый для переменной, рассчитывается путем сложения количества байтов, выделенных для каждого компонента.

Pascal

Например, требования к памяти для переменных `A`, `B`, `p` и `s` в объявлениях Turbo PASCAL

```
var A : array[1..n, 1..n] of real;
    B : array[1..n] of integer;
    p : boolean;
    s : string[10];
```

или

$$V(n) = 6n^2 + 2n + 1 + 10 \text{ (байт)}$$

C++

Требования к памяти для переменных `A`, `B`, `p` и `s` в объявлениях C/C++

```
double A[n][n];
long long B[n];
bool p;
char s[10];
```

или

$$V(n) = 8n^2 + 8n + 1 + 10 \text{ (байт)}.$$

В общем случае объем памяти произвольной программы зависит не только от типа использованных переменных, но и от способа управления внутренней памятью компьютера. Соответствующие способы изучаются на продвинутых курсах информатики.

Вопросы и задания

1. **Исследуйте!** Пользуясь системой помощи среды программирования, в которой вы работаете, определите объем памяти, занимаемый простыми и составными переменными.
2. **Объясните!** Как вычисляется объем внутренней памяти, необходимой для хранения данных, используемых в определенном алгоритме?
3. **Учитесь учиться!** Пользуясь системой помощи среды программирования, в которой вы работаете, определите, как происходит управление внутренней памятью в процессе выполнения программы.
4. **Учитесь учиться!** Пользуясь системой помощи среды программирования, в которой вы работаете, определите как можно изменить объем внутренней памяти, выделенной для выполнения программы?

5 **Применіте!** Вычислите объем памяти, необходимый для переменных из приведенных ниже объявлений:

Pascal

a)
```
var A : array[1..n, 1..n] of integer;
    B : string;
    C : array [1..n, 1..n, 1..n] of boolean;
```

b)
```
type Vector  = array[1..n] of real;
     Matrice = array[1..n] of Vector;
var A, B, C : Matrice;
    D : Vector;
```

c)
```
type Elev = record
               Nume : string;
               Prenume : string;
               NotaMedie : real
            end;
     ListaElevi = array[1..n] of Elev;
var A, B, C : ListaElevi;
```

d)
```
type Angajat = record
                  NumePrenume : string;
                  ZileLucrate : 1..31;
                  PlataPeZi : real;
                  PlataPeLuna : real
               end;
     ListaDePlata = array[1..n] of Angajat;
var L1, L2 : ListaDePlata;
```

C++

a)
```
int A [n][n];
string B[256];
bool C [n][n][n];
```

b)
```
double A[n][n], B[n][n], C[n][n];
float D[n];
```

c)
```
struct Elev {
             string Nume;
             string Prenume;
             float NotaMedie;
           } A[n], B[n], C[n];
```

d)
```
struct Angajat {
                char NumePrenume[20];
                int ZileLucrate;
                float PlataPeZi;
                float PlataPeLuna;
                } L1[n], L2[n];
```

6 **Experimentirуйте!** Вычислите объем памяти, необходимый для приведенной ниже программы. Скомпилируйте эту программу для следующих последовательных значений константы n: 10, 100, 1000, 10000 и т.д. Объясните сообщения, отображаемые на экране.

Pascal
```
Program P146;
   { Размеры сегмента данных }
const n = 50;
type Matrice = array[1..n, 1..n] of real;
var A, B : Matrice;
begin
  {...ввод данных...}
  {...обработка матриц A и B...}
 writeln('Конец');
 readln;
end.
```

C++
```
// Program P146
// Размеры сегмента данных
#include <iostream>
 using namespace std;
#define n 50
float A[n][n], B[n][n];
int main()
{
  //...ввод данных...
  //...обработка матриц A и B...
 cout << "Конец";
}
```

7 **Анализируйте şi применяйте!** Дана следующая программа:

Pascal
```
Program P147;
 { Размеры стека }
var n : integer;

function S(n:integer):real;
begin
 if n=0 then S:=0
        else S:=S(n-1)+n;
end; { S }
```

```
begin
 write('n='); readln(n);
 writeln('s=', S(n));
 readln;
end.
```

C++

```
// Program P147
// Размеры стека
#include <iostream>
using namespace std;
long long n;

double S(long long n)
{
 if (n == 0)  return 0;
        else return S(n - 1) + n;
}

int main()
{
 do
 {
   cout << "n = ";  cin >> n;
   cout << "n = " << n << "  s = " << S(n) << endl;
 } while (n);
}
```

В приведенной программе сумма

$$S(n) = 0 + 1 + 2 + \ldots + n$$

вычисляется при помощи рекурсивной функции

$$S(n) = \begin{cases} 0, \text{ если } n = 0; \\ S\,(n-1)+n, \text{ если } n > 0. \end{cases}$$

Оцените объем памяти, необходимы программе P147. Определите максимальное значение n, при котором программа P147 выполняется без ошибок. Считается, что среда программирования выделяет данной программе 32 байта оперативной памяти.

8 **Экспериментируйте!** Дана программа P147 из предыдущего задания. Задавая n значения 10, 100, 1000, 10000 и т.д., определите, для каких из этих значений данная программа работает без ошибок. Оцените объем памяти, выделяемый данной программе средой программирования, в которой вы работаете.

9 **Проведите исследование!** Дана следующая программа. Задавая n значения 10, 100, 1000, 10000 и т.д., определите, для каких из этих значений данная программа выполнится без ошибок. Сравните объем памяти, выделяемый данной программе с объемом, выделяемым средой программирования в которой вы работаете.

Pascal

```
Program P148;
 { Размеры кучи (heap) }
type Vector = array[1..100] of real;
var p : ^Vector;
    i, n : longint;
begin
 write('n='); readln(n);
 writeln('Start');
 for i:=1 to n do new(p);
 writeln('Sfârșit');
 readln;
end.
```

C++

```
// Program P148
// Размеры кучи (heap)
#include <iostream>
using namespace std;
int main()
{
 double *p = new (nothrow) double[1000000];
 long long i = 0;
 do
 {
    double* p = new(nothrow) double[1000000];
    if (!p)  {  cout << "Невозможно выделить память\n";
                break;
             }
 } while (p);
}
```

2.3 Измерение времени выполнения алгоритма

Для уже разработанных программ время $T(n)$, необходимое алгоритму, можно определить с помощью прямых измерений.

Pascal

В языке PASCAL для измерения времени выполнения программы будем использовать программный модуль U7:

```
Unit U7; { Измерение времени }
interface
function TimpulCurent : real;
implementation
uses Dos;
var ore : word;
    minute : word;
    secunde : word;
    sutimi : word;
```

```
function TimpulCurent;
 { Возвращает текущее значение времени в секундах }
begin
GetTime(ore, minute, secunde, sutimi);
TimpulCurent:=3600.0*ore+60.0*minute+
              1.0*secunde+0.01*sutimi;
end; { TimpulCurent }
end.
```

Примечание. Программные модули (тексты PASCAL, начинающиеся с ключевого слова **unit**), можно скомпилировать отдельно и подключить к любой программе PASCAL с помощью клаузы **uses** (используй).

Программный модуль U7 дает программисту возможность использовать предопределенную функцию TimpulCurent, которая возвращает значение вещественного типа – текущее время, выраженное в секундах и сотых долях секунды. Показания системных часов компьютера (часы, минуты, секунды и сотые доли секунды) считываются при помощи процедуры GetTime, содержащейся в модуле DOS в средах программирования PASCAL.

C++

В языке программирования C/C++ для измерения времени используются типы данных и функции из библиотеки <sys/time.h>, которую, разумеется, необходимо включить в разрабатываемую программу:

```
#include <sys/time.h>
    using namespace std;
struct timeval start, stop;
double secunde;

double timpSecunde (struct timeval start, struct timeval stop)
{
    return (double)(stop.tv_usec - start.tv_usec) / 1000000 +
           (double)(stop.tv_sec - start.tv_sec);
}
```

Проанализируем программу P149, чтобы понять, как соответствующие функции используются для измерения времени выполнения подпрограммы:

Pascal

```
Program P149; { Время выполнения процедуры Sortare }
uses U7;type Vector = array[1..100000] of real;
      var   A  : Vector;
      i, n : longint;
      T1, T2 : real; { время в секундах }

procedure Sortare(var A:Vector; n:longint);
  { Сортировка элементов массива A }
var i, j : longint;
       r : real;
```

```
begin
  for i:=1 to n do
     for j:=1 to n-1 do
        if A[j]>A[j+1] then
           begin
             r:=A[j];
             A[j]:=A[j+1];
             A[j+1]:=r;
           end;
end; { Sortare }

begin
 write('Введите число элементов n=');
 readln(n);

 { массиву A присваивается значение (n, n-1, ..., 3, 2, 1) }
 for i:=1 to n do A[i]:=n-i+1;

 T1:=TimpulCurent;
 Sortare(A, n);
 T2:=TimpulCurent;

 writeln('Длительность выполнения', (T2-T1):7:2, ' секунд');
 readln;
end.
```

C++

```
// Program P149;
// Время выполнения нетипизированной функции Sortare

#include <iostream>
#include <sys/time.h>
    using namespace std;

int   A[10000], n;
struct timeval start, stop;
double secunde;

double timpSecunde (struct timeval start, struct timeval stop)
{
    return (double)(stop.tv_usec - start.tv_usec) / 1000000 +
           (double)(stop.tv_sec - start.tv_sec);
}

void Sortare (int *x, int n) {
// Сортировка элементов массива x
    int r ;
```

```
    for (int i = 1; i < n; i++)
        for (int j = 1; j < n; j++)
        if (x[j] > x[j + 1])
           { r = x[j];
             x[j] = x[j + 1];
             x[j + 1] = r;
           }
}

int main()
{
    cout << "Число элементов"; cin >> n;
    // // массиву A присваивается значение (n, n-1, ..., 3, 2, 1)
    for (int i = 1; i <= n; i++)
        A[i] = n - i + 1;
     gettimeofday(&start, NULL);
     Sortare (A, n);
     gettimeofday(&stop, NULL);
     cout <<"Длительность выполнения - " << timpSecunde(start, stop)
<< "секунд";
    return 0;
}
```

Подпрограмма `Sortare` упорядочивает элементы массива `A` пузырьковым методом. В этом методе массив `A` просматривается n раз, причем при каждом проходе выполняется $n-1$ сравнение соседних элементов `A[j]` и `A[j+1]`. Если `A[j]>A[j+1]`, то такие элементы обмениваются местами.

Для того чтобы избежать ввода с клавиатуры большого количества данных, в программе `P149` массиву `A` присваивается начальное значение

$$A = (n, n-1, n-2, \ldots, 3, 2, 1).$$

Например, для n=4 исходный массив имеет вид:

$$A = (4, 3, 2, 1).$$

В процессе сортировки получаем:

$$i = 1,\ A = (3, 2, 1, 4);$$
$$i = 2,\ A = (2, 1, 3, 4);$$
$$i = 3,\ A = (1, 2, 3, 4);$$
$$i = 4,\ A = (1, 2, 3, 4).$$

Время выполнения подпрограммы `Sortare` для компьютера с тактовой частотой 1,6 ГГц приведено в *таблице 2.3*, а соответствующий график – на *рисунке 2.1*.

Таблица 2.3. Время выполнения подпрограммы `Sortare`

n	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
$T(n), s$	2,56	10,23	22,35	40,21	59,45	91,12	123,11	160,19	203,92	248,77

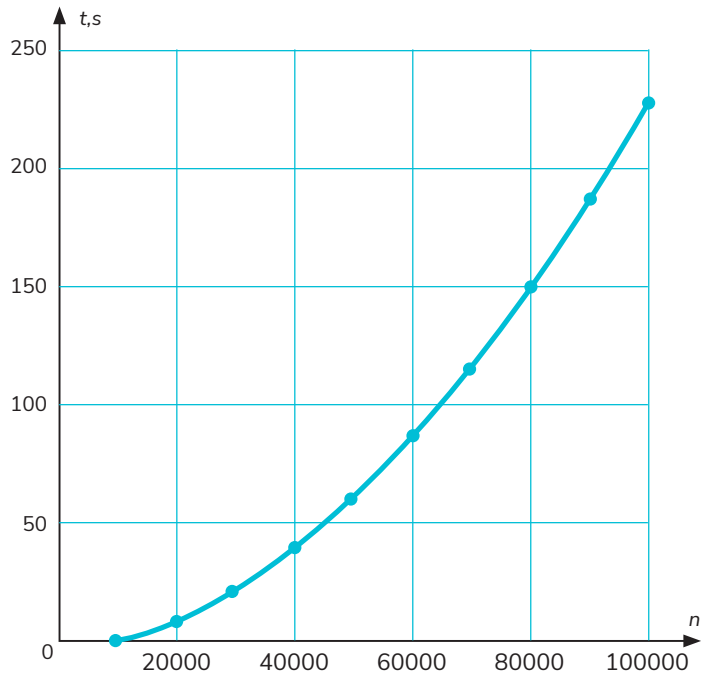


Рис. 2.1. Время выполнения подпрограммы `Sortare`

Подчеркнем тот факт, что современные операционные системы запускают несколько программ параллельно (одновременно). Чтобы не включать во время выполнения подпрограммы время, затрачиваемое параллельно работающими программами, их необходимо закрыть.

Но некоторые программы, особенно те, которые входят в состав операционной системы, закрыть невозможно. Например, нельзя закрыть драйверы экрана и клавиатуры. Следовательно, результаты измерения времени выполнения содержат определенные ошибки.

Чтобы уменьшить ошибки измерения времени выполнения:

1. Закройте текстовые процессоры, электронные таблицы, редакторы электронных презентаций, интернет-браузеры, мессенджеры, обучающие игры и т.д.
2. Скомпилируйте программу, по которой вы будете производить измерения, и сохраните ее на диске в формате объектного кода.
3. Закройте среду программирования.
4. Запустите скомпилированную программу (объектный код).
5. Входные данные подбирайте так, чтобы время выполнения превышало несколько сотых секунды.

Не забывайте!

Время выполнения любой подпрограммы зависит:

- от типа компьютера (тактовая частота, количество ядер микропроцессора, объем буферной памяти, объем оперативной памяти и т.п.);
- от типа операционной системы и ее конфигурации;
- от типа среды программирования и ее конфигурации.

Следовательно, время выполнения одной и той же подпрограммы может варьироваться от одного компьютера к другому.

Вопросы и задания

1. **Проанализируйте!** Какова связь между временем, необходимым для выполнения программы и тактовой частотой, количеством ядер процессора, объемом оперативной памяти?
2. **Экспериментируйте!** Измерьте время выполнения подпрограммы `Sortare` (смотри программу `P149`) для компьютера, на котором вы работаете. Постройте график, аналогичный представленному на *рисунке 2.1*.
3. **Проведите исследование!** Представьте графически на одном рисунке время выполнения приведенных ниже подпрограмм.

a)

Pascal

```
procedure N2(n : longint);
var i, j, q : longint;
          r : real;
begin
   for i:=1 to n do
     for j:=1 to n do
        for q:=1 to 300000 do
r:=1.0;
end; { N2 }
```

C++

```
void N2(long long n)
{
long long i, j, q;
double r;
for (i = 1; i <= n; i++)
 for (j = 1; j <= n; j++)
  for (q = 1; q <= 300000;
q++)
     r = 1.0;
} // N2
```

b)

Pascal

```
procedure N3(n : longint);
var i, j, k, q : longint;
             r : real;
begin
   for i:=1 to n do
     for j:=1 to n do
        for k:=1 to n do
           for q:=1 to 10000
do r:=1.0;
end; { N3 }
```

C++

```
void N3(long long n)
{
long long i, j, k, q;
double r;
for (i = 1; i <= n; i++)
 for (j = 1; j <= n; j++)
   for (k = 1; k <= n; k++)
     for (q = 1; q <= 10000;
q++)
      r = 1.0;
} // N3
```

c)

Pascal

```
procedure N4(n : longint);
var i, j, k, m, q : longint;
                 r : real;
begin
   for i:=1 to n do
     for j:=1 to n do
        for k:=1 to n do
           for m:=1 to n do
              for q:=1 to
400 do r:=1.0;
end; { N4 }
```

C++

```
void N4(long long n)
{
long long i, j, k, m, q;
double r;
for (i = 1; i <= n; i++)
 for (j = 1; j <= n; j++)
   for (k = 1; k <= n; k++)
    for (m = 1; m <= n; m++)
      for (q = 1; q <= 400;
q++)
       r = 1.0;
} // N4
```

Например, на *рисунке 2.2* соответствующие графики представлены для компьютера с тактовой частотой 1,6 ГГц, двумя ядрами, 6 Гбайт оперативной памяти, операционной системой Windows 10.

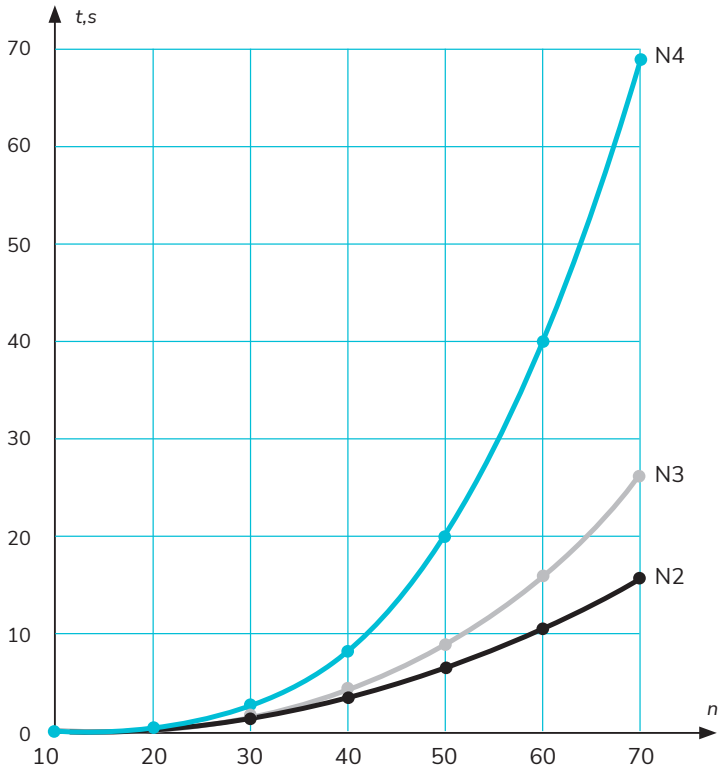


Рис. 2.2. Время выполнения подпрограмм `N2`, `N3` и `N4`

4 **Учитесь учиться!** Какова точность измерения времени выполнения с помощью функций, предоставляемых средой программирования, с которой вы работаете? Обоснуйте свой ответ.

2.4 Оценка времени выполнения алгоритма

При практической реализации любого алгоритма появляется необходимость в предварительной оценке времени его выполнения $T(n)$ до окончательного тестирования и отладки самой программы. К сожалению, теоретически очень тяжело найти точное выражение для $T(n)$. По этой причине обычно ищут верхний предел времени, требуемого для выполнения алгоритма, анализируя только самые неблагоприятные случаи.

Предположим, в учебных целях, что на выполнение любой из операций затрачивается не более Δ единиц времени. Такое же время затрачивается на индексирование элементов произвольного массива, на присваивание и выполнение оператора безусловного перехода. Конкретное значение величины Δ зависит от среды программирования, производительности используемого компьютера и составляет порядка 10^{-9} секунды. В дальнейшем будем оценивать время $T(n)$ по формуле

$$T(n) = Q(n) \cdot \Delta,$$

где $Q(n)$ – количество элементарных операций – сложения, вычитания, умножения, сравнения и т.д. – необходимых для решения некоторой задачи.

Допустим, что в выражении E встречается m операций и k вызовов функции F. Очевидно, что общее количество Q_E элементарных операций, необходимых для вычисления выражения E, определяется как

$$Q_E = m + k\,Q_F,$$

где Q_F – количество элементарных операций, необходимых для вычисления функции F.

Примеры:

	Выражение E	*Количество элементарных операций* Q_E
a)	`a*b+c`	2
b)	`(a<b)or(c>d)`	3
c)	`sin(x)+cos(y)`	$1 + Q_{\sin} + Q_{\cos}$
d)	`a+M[i]`	2
e)	`sin(x+y)+sin(x-y)`	$3 + 2Q_{\sin}$

Количество элементарных операций Q_I, необходимых для выполнения некоторого оператора, оценивается по формулам *таблицы 2.4.*

Таблица 2.4. Количество элементарных операций, необходимых для выполнения некоторых операторов

№	Оператор PASCAL Оператор C / C++	Количество элементарных операций
1.	Присваивание $v := E$ $v = E$	$Q_E + 1$
2.	Вызов процедуры P Вызов нетипизированной функции P	$Q_P + 1$
3.	Выбор **if** E **then** I_1 **else** I_2 **if** (E) I_1; **else** I_2	$Q_E + \max\{Q_{I_1}, Q_{I_2}\} + 1$
4.	Множественный выбор **case** E **of** I_1; I_2; ...; I_k **end** **switch**$(E)\{$ **case** I_1; **case** I_2; . . . **case** I_k $\}$	$Q_E + \max\{Q_{I_1}, Q_{I_2}, \ldots, Q_{I_k}\} + \kappa + 1$
5.	Цикл со счетчиком **for** v:=E_1 **to/downto** E_2 **do** I **for** (**int** v = E_1; v <= E_2;i++) I	$Q_{E_1} + Q_{E_2} + mQ_I + m + 1$
6.	Цикл с предусловием **while** E **do** I **while** (E) { I; }	$(m+1)Q_E + mQ_I + 1$
7.	Цикл с постусловием **repeat** I **until** E **do** { I; } **while** (E)	$mQ_I + mQ_E + 1$

8.	Составной оператор **begin** I_1; I_2; ...; I_k **end** { I_1; I_2; ...; I_k }	$Q_{I_1} + Q_{I_2} + \cdots + Q_{I_k} + 1$
9.	Оператор **with** v **do** I Не существует в C/C++	$Q_I + 1$
10.	Переход **goto** **goto**	1

Формулы из *таблицы 2.4* могут быть выведены исходя из способа выполнения каждого оператора. В таблице 2.4 используются следующие обозначения: v – переменная или имя функции; E – выражение; I – оператор. Количество повторений оператора I внутри цикла обозначено через m. Подчеркнем, что циклы в программе могут быть организованы и при помощи операторов выбора и условного либо безусловного перехода, хотя такое применение указанных операторов противоречит правилам структурного программирования.

Для примера оценим количество элементарных операций $Q(n)$, необходимых для упорядочивания элементов массива пузырьковым методом:

Pascal

```
procedure Sortare(var A:Vector; n:integer);
var i, j : integer;
       r : real;
{1}  begin
{2}  for i:=1 to n do
{3}     for j:=1 to n-1 do
{4}        if A[j]>A[j+1] then
{5}           begin
{6}             r:=A[j];
{7}             A[j]:=A[j+1];
{8}             A[j+1]:=r;
              end;
end; { Sortare }
```

C++

```
void Sortare(double *A, int n)
{                                          // 1
    for (int i = 1; i < n; i++)            // 2
        for (int j = 1; j < n; j++)        // 3
            if (A[j] > A[j + 1])           // 4
                {                          // 5
                   r = A[j];               // 6
                   A[j] = A[j + 1];        // 7
                   A[j + 1] = r;           // 8
                }
    return;
}
```

Операторы I_1, I_2, ... , I_8 подпрограммы `Sortare` помечены при помощи комментариев в соответствующих строках программы. Через Q_j будем обозначать количество элементарных операций, необходимых для выполнения оператора I_j. В соответствии с *таблицей 2.4* получаем:

$Q_6 = 2$;
$Q_7 = 4$;
$Q_8 = 3$;
$Q_5 = Q_6 + Q_7 + Q_8 + 1 = 10$;
$Q_4 = 4 + Q_5 + 1 = 15$;
$Q_3 = 0 + 1 + (n-1)Q_4 + (n-1) + 1 = 16n - 14$;
$Q_2 = 0 + 0 + nQ_3 + n + 1 = 16n^2 - 13n + 1$;
$Q_1 = Q_2 + 1 = 16n^2 - 13n + 2$.

Следовательно, количество элементарных операций

$$Q(n) = 16n^2 - 13n + 2,$$

а время выполнения процедуры `Sortare`

$$T(n) = (16n^2 - 13n + 2)\Delta .$$

Из рассмотренного выше примера замечаем, что порядок прохода (выполнения) операторов диктуется структурой самой программы. Очевидно, что сначала анализируются простые операторы, а затем – составные. Для вложенных операторов в первую очередь анализируются внутренние, а затем – внешние операторы.

Аналитические выражения для $T(n)$, полученные в результате анализа программ, могут быть использованы для экспериментального нахождения времени Δ, нужного для выполнения одной элементарной операции.

Например, для процедуры `Sortare` (смотри *таблицу 2.3*) $n = 10000$ и $T(n) = 2{,}56$ с. Из уравнения

$$(16n^2 - 13n + 2)\Delta = 2{,}56$$

получаем $\Delta \approx 1{,}6 \cdot 10^{-9}$ секунд.

Очевидно, что найденная величина применима только для компьютера с тактовой частотой 1,6 ГГц, использовавшегося в ходе измерения времени выполнения подпрограммы `Sortare`.

Например, в случае компьютера с тактовой частотой 3,2 ГГц получается величина $\Delta \approx 0{,}8 \cdot 10^{-9}$ секунд.

Вопросы и задания

1 **Экспериментируйте!** Определите с помощью программы `P149` значение Δ для компьютера и среды программирования, с которыми вы работаете.

2 **Практикуйтесь!** Определите количество элементарных операций Q_i необходимых для выполнения следующих операторов:

Pascal

a)
```
x:=2*a-6*(y+z);
```

b)
```
p:=not(a=b)and(c>d);
```

c)
```
p:=(a in R)and(b in P);
```

d)
```
if a>b then x:=0 else x:=a+b;
```

e)
```
case i of
   1: x:=0;
   2: x:=a+b;
   3: x:=a+b+c;
end;
```

f)
```
for i:=1 to n do A[i]:=2*A[i];
```

g)
```
for i:=1 to n do A[i]:=B[i+1]-C[i-2];
```

h)
```
i:=0; while i<n do begin i:=i+1 end;
```

i)
```
i:=n; repeat i:=i-1 until i=0;
```

j)
```
begin i:=0; s:=0; r:=0 end;
```

k)
```
with A do begin x:=0; y:=0 end.
```

C++

a)
```
x = 2 * a - 6 * (y + z);
```

b)
```
p = !(a == b) && (c > d);
```

c)
```
set <int> p; set <int> R;
p = (r.count(a) == 1) && (r.count(b) == 1);
```

d)
```
if (a > b) x = 0; else x = a + b;
```

e)
```
switch(i)
{
   case 1: x = 0;
   case 2: x =a + b;
   case 3: x = a + b + c;
}
```

f)
```
for (int i = 1; i<= n; i++) A[i] = 2 * A[i];
```

g)
```
for (int i = 1; i<= n; i++) A[i] = B[i + 1] - C[i - 2];
```

h)
```
i = 0; while (i < n) { i++; }
```

i)
```
i = n; do {i--; } while (i != 0);
```

j)
```
{ i = 0; s = 0; r = 0; }
```

k)
```
{ A.x = 0; A.y = 0; }
```

3 **Примените!** Оцените количество элементарных операций $Q(n)$ в подпрограммах `N2`, `N3` и `N4` (смотри упражнение 3 из предыдущего параграфа).

4 **Проанализируйте!** Для подпрограммы `Sortare`, выполняемой на компьютере с тактовой частотой $f_1 = 1{,}6$ ГГц, получено $\Delta_1 \approx 1{,}6 \cdot 10^{-9}$ с. Для такого же типа компьютера, но с тактовой частотой $f_2 = 3{,}2$ ГГц получено $\Delta_2 \approx 0{,}8 \cdot 10^{-9}$ с. Сопоставляя полученные данные, замечаем, что

$$\frac{f_2}{f_1} = \frac{3{,}2 \cdot 10^9}{1{,}6 \cdot 10^9} = 2 \quad \text{и} \quad \frac{\Delta_2}{\Delta_1} = \frac{0{,}8 \cdot 10^{-9}}{1{,}6 \cdot 10^{-9}} = \frac{1}{2}.$$

Как вы считаете, чем объясняется этот факт?

5 **Разработайте!** В процессе компиляции программ на языках высокого уровня каждый оператор транслируется в одну и более команд языка машинного кода. Количество получаемых в результате компиляции команд зависит от конкретной реализации среды программирования и типа используемого компьютера. Придумайте план эксперимента, который позволил бы оценить количество команд машинного кода, в которые транслируется каждый оператор языка высокого уровня, изучаемого вами.

6 **Экспериментируйте!** Известно, что время выполнения команд языка машинного кода зависит от их типа. Например, команда суммирования двух целых чисел выполняется быстрее, чем команда суммирования двух вещественных чисел. Вследствие этого, значения Δ, найденные в результате измерения времени выполнения программ написанных на языках высокого уровня, зависят от типа использованных данных. Проверьте экспериментально это утверждение для компьютера, на котором вы работаете.

7 **Проведите исследование!** Производительность компьютера измеряется в *Gips* – Гига-операций в секунду (10^9 операций в секунду). Для измерения этой характеристики производители компьютеров используют операции языка машинного кода. Например, вычислительная мощность современных персональных компьютеров составляет 1–5 *Gips*. Однако для программиста будет интересна не только вычислительная мощность, выраженная в операциях языка машинного кода, но и в операторах языков программирования высокого уровня. Разработайте план эксперимента, который позволит оценить эту характеристику для компьютера, на котором вы работаете.

2.5 Временная сложность алгоритмов

В информатике временная сложность характеризуется временем выполнения $T(n)$ или количеством элементарных операций $Q(n)$. Поскольку современные компьютеры обладают очень большой скоростью вычислений – порядка 10^9 ... 10^{11} команд в секунду, то проблема времени выполнения возникает только при больших значениях n. Следовательно, в формулах, выражающих количество элементарных операций $Q(n)$, представляет интерес только **доминирующий член**, т.е. тот, который быстрее остальных стремится к бесконечности с ростом n. Преобладание доминирующего члена над остальными демонстрируется в *таблице 2.5*.

Таблица 2.5. Значения доминирующих членов

n	$\log_2 n$	n^2	n^3	n^4	2^n
2	1	4	8	16	4
4	2	16	64	256	16
8	3	64	512	4096	256
16	4	256	4096	65536	65536
32	5	1024	32768	1048576	4294967296

Например, количество элементарных операций процедуры `Sortare` выражается формулой

$$Q(n) = 16n^2 - 13n + 2.$$

Доминирующий член в этом выражении $16n^2$. Очевидно, что для больших значений n количество элементарных операций

$$Q(n) \approx 16n^2,$$

а время выполнения

$$T(n) \approx 16n^2\Delta.$$

В зависимости от временной сложности, алгоритмы классифицируются на:
– полиномиальные алгоритмы;
– экспоненциальные алгоритмы;
– полиномиально-недетерминированные алгоритмы.

Алгоритм называется **полиномиальным**, если доминирующий член имеет вид Cn^k, то есть

$$Q(n) \approx Cn^k; \quad T(n) \approx Cn^k\Delta,$$

где:
n – это характеристика входных данных;
C – положительная константа;
k – натуральное число.

Временная сложность полиномиальных алгоритмов отражена в обозначении $O(n^k)$, которое читается как «алгоритм со временем выполнения порядка n^k», или короче, **«алгоритм порядка n^k»**. Очевидно, что существуют полиномиальные алгоритмы порядка n, n^2, n^3 и т.д.

Например, упорядочивание элементов произвольного массива методом пузырьковой сортировки представляет собой полиномиальный алгоритм порядка n^2. Это видно из графика для времени выполнения $T(n)$ процедуры `Sortare`, представленного на *рисунке 2.1*.

Алгоритм называется **экспоненциальным**, если доминирующий член имеет вид Ck^n, то есть

$$Q(n) \approx Ck^n; \quad T(n) \approx Ck^n\Delta,$$

где $k>1$. Временная сложность экспоненциальных алгоритмов отражена в обозначении $O(k^n)$.

Отметим, что экспоненциальными считаются также алгоритмы сложности $n^{\log n}$, хотя эта функция и не является экспоненциальной в строго математическом смысле.

Из сравнения скоростей роста экспоненциальной и полиномиальной функций (смотрите *таблицу 2.5*) следует, что экспоненциальные алгоритмы становятся неприменимыми даже при небольших значениях n. Для примера обратимся к *таблице 2.1*, в которой представлено время выполнения $T_1(n)$ полиномиального алгоритма порядка $O(n^3)$ и время выполнения $T_2(n)$ экспоненциального алгоритма порядка $O(2^n)$.

Полиномиально-недетерминированные алгоритмы изучаются в углубленных курсах информатики.

В зависимости от временной сложности считается, что задача **легкоразрешимая**, если для ее решения существует полиномиальный алгоритм. Задача, для решения которой не существует полиномиального алгоритма, называется **трудноразрешимой**. Подчеркнем, что указанная классификация относится только к асимптотическому анализу алгоритмов, то есть касается лишь больших значений n. Для малых n ситуация может сильно отличаться.

Например, предположим, что для решения некоторой задачи существуют два алгоритма, первый из которых полиномиальный со временем выполнения $T_1(n)$, а второй – экспоненциальный со временем выполнения $T_2(n)$:

$$T_1(n) = 1000n^2\Delta;$$
$$T_2(n) = 2^n\Delta.$$

Прямыми вычислениями можно проверить, что для $n = 1, 2, 3, \ldots, 18$ время $T_2(n) < T_1(n)$. Следовательно, в случае $n \le 18$ для применения предпочтительнее экспоненциальный алгоритм.

На практике разработка компьютерных программ предполагает выполнение следующих этапов:

- точное формулирование задачи;
- определение временной сложности задачи – является ли она легко- или трудноразрешимой;
- разработку соответствующего алгоритма и реализацию его на определенной вычислительной системе.

Очевидно, что в случае легкоразрешимых задач программист должен направить все усилия на разработку полиномиального алгоритма, т.е. алгоритма порядка n^k, стремясь к тому, чтобы параметр k принял как можно меньшее значение. В случае трудноразрешимых задач приоритет отдается тем алгоритмам, которые минимизируют время выполнения хотя бы для наиболее часто встречающихся на практике размеров входных данных. Методы классификации задач на легко- и трудноразрешимые изучаются в углубленных курсах информатики.

Вопросы и задания

1 **Практикуйтесь!** Укажите доминирующий член в следующих выражениях:

a) $12n + 5$;

b) $6n^2 + 100n + 18$;

c) $15n^3 + 1000n^2 - 25n + 6000$;

d) $2000n^3 + 2^n + 13$;

e) $n^{\log_2 n} + n^5 + 300\,n^2 + 6$;

f) $3^n + 2^n + 14n^3 + 21$;

g) $n^5 + 10n^4 + 200n^3 + 300n^2 + 1000n$.

2 **Систематизируйте свои знания!** Как классифицируются алгоритмы в зависимости от их временной сложности?

3 **Практикуйтесь!** Определите тип алгоритма по его временной сложности:

a) $Q(n) = 200n + 15$;

b) $Q(n) = 2^n + 25n^2 + 1000$;

c) $Q(n) = n^3 + 3^n + 6000n^2 + 10^6$;

d) $Q(n) = 3^n + 2^n + n^{10} + 4000$;

e) $Q(n) = n^{\log_2 n} + n^3 + n^2 + 1500$;

f) $Q(n) = 100n^2 + 15n^3 + 8^n + 900$.

4 **Объясните!** Чем, по вашему мнению,объясняется важность доминирующего члена при анализе временной сложности алгоритмов?

5 **Проведите исследование!** Определите порядок временной сложности алгоритмов, описанных подпрограммами `N2`, `N3` и `N4` (смотри задание 3 из параграфа 2.3). Сравните скорость увеличения времени выполнения $T_{N2}(n)$, $T_{N3}(n)$ и $T_{N4}(n)$, используя для этого графики на *рисунке 2.2*.

6 **Примените!** Рассмотрим алгоритм, состоящий из k вложенных циклов:

Pascal

```
for i1:=1 to n do
   for i2:=1 to n do
       ...
           for ik:=1 to n do P
```

C++

```
for ( int i1 = 1; i1 <=n; i1++ )
   for ( int i2 = 1; i2 <= n; i2++ )
       ...
           for ( int ik = 1; ik <= n; i2++
```

Количество элементарных операций Q_P, выполняемых в подпрограмме `P` – постоянная величина. Оцените временную сложность алгоритма.

7 **Интегрируйте знания и применяйте!** Придумайте алгоритм решения следующей задачи: Дано множество A, состоящее из n целых чисел. Определите, существует ли хотя бы одно подмножество B, $B \subseteq A$, сумма элементов которого равна m. Например, для A={-3, 1, 5, 9} и m=7, такое подмножество существует, а именно, B={-3, 1, 9}.

Оцените временную сложность разработанного алгоритма.

2.6 Итерация или рекурсия

Развитие информатики помогло установить, что многие практически важные задачи могут быть решены с помощью определенного набора стандартных приемов, называемых **методами программирования**: рекурсии, перебора, метода перебора с возвратом, эвристических методов и других.

Одним из самых распространенных методов программирования является **рекурсия**. Напомним, что рекурсия определяется как ситуация, когда подпрограмма обращается к самой себе либо непосредственно, либо посредством другой подпрограммы. Рассматриваемые методы, изученные в рамках темы «Функции и процедуры», называются соответственно **прямой рекурсией** и **косвенной рекурсией.**

В целом/в общем виде разработка любой рекурсивной программы возможна только тогда, когда соблюдается следующее **правило сходимости**: решение задачи должно вычисляться непосредственно либо вычисляться при помощи некоторых непосредственно вычисляемых значений. Другими словами, правильное определение любого рекурсивного алгоритма предполагает, что в процессе осуществления вычислений должны существовать:

– элементарные случаи, которые вычисляются непосредственно;

– случаи, которые хотя и не могут быть решены явно, однако процесс вычислений обязательно сводится к элементарному случаю.

Например, в рекурсивном определении функции факториал $fact$: $\boldsymbol{N}\to\boldsymbol{N}$,

$$fact(n) = \begin{cases} 1, \text{ если } n = 0; \\ n \cdot fact\,(n-1), \text{ если } n > 0, \end{cases}$$

различаем:

– Элементарный случай $n = 0$. В этом случае значение $fact(0)$ вычисляется непосредственно, а именно $fact(0)=1$.

– Неэлементарные случаи $n > 0$. В таких случаях значения $fact(n)$ не могут быть вычислены непосредственно, но процесс вычислений сводится к элементарному случаю $fact(0)$.

Например, для $n = 3$ получаем:

$$fact(3) = 3 \cdot fact(2) = 3 \cdot 2 \cdot fact(1) = 3 \cdot 2 \cdot 1 \cdot fact(0) = 3 \cdot 2 \cdot 1 \cdot 1 = 6.$$

Следовательно, рекурсивное определение функции $fact(n)$ является **сходящимся определением**. Напомним, что функция $fact(n)$ может быть представлена на языке высокого уровня в соответствии с рекурсивным определением в форме:

Pascal
```
function Fact(n:Natural):Natural;
begin
  if n=0 then Fact:=1
  else Fact:=n*Fact(n-1)
end;
```

C++
```
long Fact(int n)
{
    if (n == 0) return 1;
    else return n * Fact(n - 1);
}
```

В оперативной памяти в случае рекурсивных вызовов текущие значения аргумента и рекурсивной функции откладываются в стек.

В обыденном языке под **стеком** (в английском языке *stack*) подразумевается множество объектов одного вида (и одинакового размера), упорядоченно расположенных один над другим, с тем свойством, что операции добавления и извлечения объектов осуществляются только в его верхней части. Позиция в стеке, которую занимает последний добавленный элемент, называется **вершиной**. Стек без каких-либо элементов называется **пустым стеком**.

В информатике стек реализуется областью рабочей памяти, обладающей тем свойством, что операции ввода и извлечения данных выполняются только на одном ее конце.

Режим управления стеком в случае вызова `Fact(3)` показан на *рисунке 2.3*.

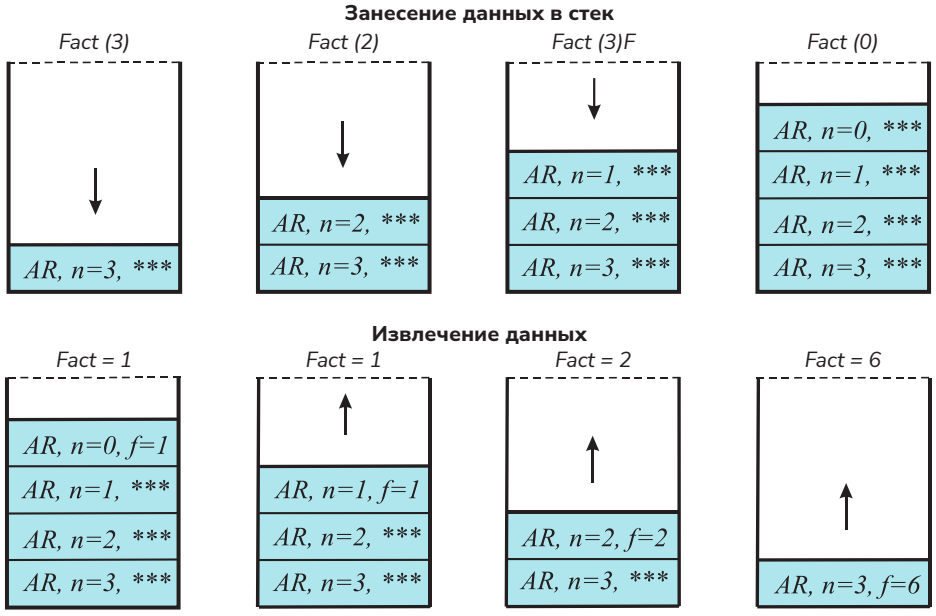


Рис. 2.3. Управление стеком в случае вызова функции `Fact(3)`:
AR – адрес возврата; *n* – текущее значение фактического параметра;
*** – место для запоминания значения *f* возвращаемого функцией `Fact`

Аналогично в рекурсивном определении функции *incons*: $\boldsymbol{N \to N}$,

$$incons\,(n) = \begin{cases} 1, \text{ если } n = 0; \\ n \cdot incons\,(n+1), \text{ если } n > 0, \end{cases}$$

можно выделить элементарный случай $n=0$ и неэлементарные случаи $n>0$. Но в отличие от функции *fact*(n), для $n > 0$ значения *incons*(n) не могут быть вычислены, поскольку процесс вычисления ведет к *incons*(∞).

Например, для $n=3$ получаем:

$$incons(3) = 3 \cdot incons(4) = 3 \cdot 4 \cdot incons(5) = 3 \cdot 4 \cdot 5 \cdot incons(6) = \cdots.$$

Следовательно, приведенное выше рекурсивное определение функции *incons*(*n*) является **расходящимся определением** и теоретически процесс вычислений будет продолжаться бесконечно. На практике же вычисления будут остановлены операционной системой в момент переполнения памяти, выделенной для стека, или превышения разрядности арифметического устройства.

Подчеркнем тот факт, что современные среды для разработки программ не выполняют автоматическую проверку сходимости рекурсивных алгоритмов, оставляя этот контроль программисту.

Как было показано в предыдущих главах, любой рекурсивный алгоритм может быть преобразован в итеративный и обратно. Выбор метода программирования – **итеративного** или **рекурсивного** – зависит от компетентности программиста. Очевидно, что при выборе должны учитываться преимущества и недостатки каждого метода, которые варьируются от случая к случаю. Для примера, в *таблице 2.6* представлены результаты сравнения обоих методов программирования для решения задач, связанных с компьютерной обработкой текстов.

Таблица 2.6. Сравнение итерации и рекурсии (компьютерная обработка текстов)

№	Характеристики	Итерация	Рекурсия
1.	Расход памяти	малый	большой
2.	Время выполнения	одинаковое	
3.	Структура программы	сложная	простая
4.	Трудоемкость написания программы	большая	малая
5.	Тестирование и отладка программ	простые	сложные

Рекурсивные алгоритмы рекомендуется применять для решения задач, описанных с помощью рекуррентных соотношений, таких как: синтаксический анализ текстов, обработка динамических структур данных, обработка изображений и т.п. Типичным примером задач подобного рода является грамматический анализ программ, написанных на языках высокого уровня, синтаксис которых, как известно, определяется при помощи рекуррентных соотношений.

Вопросы и задания

1. **Проведите исследование!** Объясните термин *методы программирования*.
2. **Систематизируйте свои знания!** В чем разница между *прямой рекурсией* и *косвенной рекурсией*? Приведите примеры.
3. **Объясните!** Какие условия должны быть соблюдены, чтобы определение рекурсивного алгоритма было правильным?
4. **Проанализируйте!** В чем разница между сходящейся и расходящейся рекурсиями?

5 **Практикуйтесь!** Даны следующие рекурсивные определения функций. Какие из этих определений являются сходящимися? Обоснуйте свой ответ.

a) $f: \boldsymbol{N} \to \boldsymbol{N}$, $f(n) = \begin{cases} 1, \text{ если } n = 0; \\ n + f(n-1), \text{ если } n > 0; \end{cases}$

b) $f: \boldsymbol{N} \to \boldsymbol{N}$, $f(n) = \begin{cases} 1, \text{ если } n = 0; \\ n + f(n), \text{ если } n > 0; \end{cases}$

c) $f: \boldsymbol{Z} \to \boldsymbol{Z}$, $f(i) = \begin{cases} 1, \text{ если } i = 0; \\ i + f(i-1), \text{ если } i \neq 0; \end{cases}$

d) $f: \boldsymbol{N} \to \boldsymbol{N}$, $f(n) = \begin{cases} 1, \text{ если } n = 0; \\ i + g(n-1), \text{ если } i > 0; \end{cases}$

e) $g: \boldsymbol{N} \to \boldsymbol{N}$, $f(n) = \begin{cases} 2, \text{ если } n = 0; \\ n + f(n-1), \text{ если } n > 0; \end{cases}$

f) $f: \boldsymbol{N} \to \boldsymbol{N}$, $f(n) = \begin{cases} n, \text{ если } n = 0; \\ (n \bmod 10) + f(n \operatorname{div} 10), \text{ если } n \geq 0. \end{cases}$

6 **Применяйте!** Запустите на выполнение следующую программу. Объясните сообщения, выводимые на экран.

Pascal

```
Program P150;
 { Переполнение стека }
type Natural = 0..Maxint;

function Incons(n:Natural):Natural;
 { Расходящееся рекурсивное определение }
begin
  writeln('Рекурсивный вызов  n=', n);
  if n=0 then Incons:=1
         else Incons:=n*Incons(n+1);
end; { Incons }

begin
  writeln(Incons(3));
  readln;
end.
```

C++

```
// Program P150;
// Переполнение стека
// Поскольку объем памяти, выделенный для стека в C/C++,
// значительно выше, чем объем, выделенный в Pascal, в функции
// была объявлена переменная tablou[n], которая прогрессивно
// потребляет память
#include <iostream>
    using namespace std;
int Incons(int n)
// Расходящееся рекурсивное определение
{
    double tablou[n];
    cout << "Рекурсивный вызов n = "<< n << endl;
    if (!n) return 1;
         else
         {   for (int i = 1; i <= n;  i++) tablou[i] = i;
             return n * Incons(n + 1);
         }
```

```
} // Incons

int main()
{
  cout << Incons(3) << endl;
}
```

Представьте на рисунке, аналогичном *рисунку 2.3*, способ управления стеком для случая вызова функции *Incons*(3).

7 **Проанализируйте!** Сумма $S(n)$ первых n натуральных чисел может быть вычислена при помощи итеративной функции:

$$S(n) = 0 + 1 + 2 + \ldots + n = \sum_{i=0}^{n} i$$

или рекурсивной функции

$$S(n) = \begin{cases} 0, \text{ если } n = 0; \\ n + S(n-1), \text{ если } n > 0. \end{cases}$$

Используя в качестве модели *таблицу 2.6*, укажите преимущества и недостатки алгоритмов вычисления суммы $S(n)$.

8 **Интегрируйте знания и применяйте!** Даны следующие металингвистические формулы:

<*Цифра*> ::= 0|1|2|3|4|5|6|7|8|9

<*Число*> ::= <*Цифра* > {<*Цифра*>}

<*Знак*> ::= +|–|*

<*Выражение*> ::= <*Число*> | (<*Выражение*>) | <*Выражение*><*Знак*><*Выражение*>

Напишите рекурсивную функцию, которая возвращает значение `true`, если строка символов *S* соответствует определению лексической единицы <*Выражение*> и `false` – в противном случае.

9 **Тематическое исследование** Напишите итеративную функцию, возвращающую значение `true`, если строка символов *S* соответствует определению лексической единицы <*Выражение*> из упражнения 8 и `false` – в противном случае. Используя в качестве модели *таблицу 2.6*, проведите сопоставительное сравнение итеративного и рекурсивного алгоритмов.

10 **Программируйте!** Напишите рекурсивную подпрограмму, вычисляющую сумму цифр произвольного натурального числа *n*. Исследуйте случаи $n \le$ `MaxInt` и $n \le 10^{250}$.

11 **Программируйте!** Черно-белые *изображения* (*рис. 2.4*) могут быть закодированы с помощью двоичной матрицы, $B=\|b_{ij}\|_{n\times m}$, $1 \le n, m \le 30$. Элемент b_{ij} указывает цвет соответствующей микрозоны: черный (b_{ij} =1) или белый (b_{ij} =0).

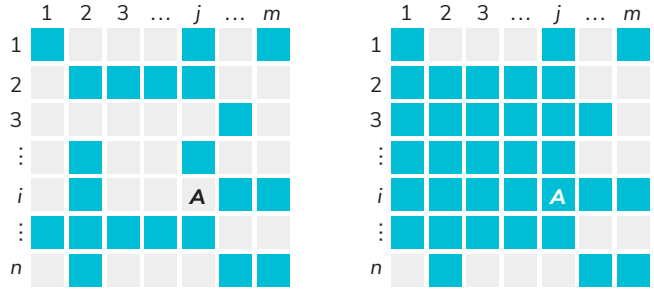


Рис. 2.4. Закраска замкнутой поверхности:
a – исходное изображение; *b* – изображение после закраски

Разработайте рекурсивную подпрограмму закраски замкнутой поверхности, определенной координатами (*i, j*) произвольной внутренней точки *A*.

12 **Программируйте!** Напишите подпрограмму, которая определяет, сколько объектов содержит черно-белое изображение. Изображение делится на микрозоны и кодируется при помощи двоичной матрицы $B = \|b_{ij}\|_{n \times m}$. Элемент b_{ij} определяет цвет микрозоны с координатами (i, j).

13 **Проведите исследование!** Проведите сравнительное исследование итеративных и рекурсивных алгоритмов решения задач из *заданий 11* и *12*.

2.7 Метод полного перебора

В методе полного перебора предполагается, что решение рассматриваемой задачи может быть найдено путем последовательного анализа элементов s_i, принадлежащих конечному множеству

$$S = \{s_1, s_2, \ldots, s_i, \ldots, s_k\},$$

называемому **множеством возможных решений**. В простейших случаях элементы s_i, $s_i \in S$, могут быть представлены значениями какого-нибудь порядкового типа: `integer (int)`, `boolean (bool)`, `char`, *перечисляемого* или *интервального*. В более сложных задачах для представления таких элементов используются массивы, записи или множества. Подчеркнем, что в большинстве задач возможные решения $s_1, s_2, \ldots, s_k$ не представлены явным образом и разработка алгоритма их нахождения ложится целиком на плечи программиста.

Общая схема алгоритма, основанного на методе полного перебора, может быть представлена с помощью цикла:

Pascal

```
for i:= 1 to k do
   if SolutiePosibila(si) then PrelucrareaSolutiei(si)
```

C++

```
for (int i = 1; i<= k; i++)
   if (SolutiePosibila(si)) PrelucrareaSolutiei(si)
```

где `SolutiePosibila` – это логическая функция, возвращающая значение `true`, если элемент s_i удовлетворяет условиям задачи, и `false` – в противном случае, а `PrelucrareaSolutiei` – это подпрограмма, осуществляющая обработку выбранного элемента. Обычно эта подпрограмма выводит решение s_i на экран.

Рассмотрим два примера, которые показывают достоинства и недостатки алгоритмов, основанных на методе перебора.

Пример 1: Рассмотрим натуральные числа, принадлежащие множеству {0, 1, 2, ..., n}. Напишите программу, определяющую, для скольких чисел K этого множества сумма цифр каждого числа равна m. В частности, для $n = 100$ и $m = 2$, множество {0, 1, 2, ..., 100} содержит 3 числа, удовлетворяющие условию задачи: 2, 11 и 20. Следовательно, $K = 3$.

Решение: Очевидно, что множество возможных решений S = {0, 1, 2, ..., n}. В следующей программе сумма цифр любого натурального числа i, $i \in S$ вычисляется с помощью функции `SumaCifrelor`. Выделение десятичных цифр из записи натурального числа i осуществляется справа налево путем деления нацело числа i и соответствующих частных на основание системы счисления 10.

Pascal

```
Program P151;
 { Сумма цифр натурального числа }
type Natural=0..MaxInt;
var i, K, m, n : Natural;

function SumaCifrelor(i:Natural):Natural;
var suma : Natural;
begin
  suma:=0;
  repeat
    suma:=suma+(i mod 10);
    i:=i div 10;
  until i=0;
  SumaCifrelor:=suma;
end; { Сумма цифр }

function SolutiePosibila(i:Natural):boolean;
begin
  if SumaCifrelor(i)=m then SolutiePosibila:=true
                       else SolutiePosibila:=false;
end; { Возможное решение }

procedure PrelucrareaSolutiei(i:Natural);
begin
  writeln('i=', i);
  K:=K+1;
end; { Обработка выбранного решения }

begin
  write('Введите n='); readln(n);
  write('Введите m='); readln(m);
  K:=0;
  for i:=0 to n do
    if SolutiePosibila(i) then PrelucrareaSolutiei(i);
  writeln('K=', K);
  readln;
end.
```

C++

```
// Program P151
// Сумма цифр натурального числа
#include <iostream>
    using namespace std;
int i, K = 0, m, n;

int SumaCifrelor(int i)
```

```
{
    if (i == 0) return 0;
    else return (i % 10) + SumaCifrelor(i / 10);
} // Сумма цифр

bool SolutiePosibila(int i)
{
    if (SumaCifrelor(i)== m) return true;
    else  return false;
} // Возможное решение

void PrelucrareaSolutiei(int i)
{
    cout << " i = " << i << endl;
    K++;
} // Обработка выбранного решения

int main()
{
    cout << " Введите n= "; cin >> n;
    cout << " Введите m= "; cin >> m;
    for (int i = 1; i <= n; i++)
        if (SolutiePosibila(i)) PrelucrareaSolutiei(i);
  cout << "K = " << K ;
}
```

Из анализа программы P151 следует, что временная сложность соответствующего алгоритма равна O(n).

Пример 2: Дано множество $P = \{P_1, P_2, ..., P_n\}$, состоящее из n точек ($2 \le n \le 30$) на евклидовой поверхности. Каждая точка P_j определяется своими координатами x_j, y_j. Напишите программу, выводящую на экран координаты точек P_a, P_b, расстояние между которыми максимально.

Решение: Множество возможных решений $S = P \times P$. Элементы (P_j, P_m) декартова произведения $P \times P$ могут быть получены при помощи двух вложенных циклов:

Pascal

```
for j:=1 to n do
  for m:=1 to n do
    if SolutiePosibila(Pj, Pm) then PrelucrareaSolutiei(Pj, Pm)
```

C++

```
for (int j = 1; j <= n; j++)
  for (int m = 1; m <= n; m++)
    if (SolutiePosibila(Pj, Pm)) PrelucrareaSolutiei(Pj, Pm)
```

Расстояние между точками P_j, P_m вычисляется по формуле:

$$d_{jm} = \sqrt{(x_j - x_m)^2 + (y_j - y_m)^2}$$

Pascal

```
Program P152;
 { Точки на евклидовой плоскости }
const nmax=30;
type Punct = record
               x, y : real;
             end;
     Indice = 1..nmax;
var P : array[Indice] of Punct;
    j, m, n : Indice;
    dmax : real; { максимальное расстояние }
    PA, PB : Punct;

function Distanta(A, B : Punct) : real;
begin
  Distanta:=sqrt(sqr(A.x-B.x)+sqr(A.y-B.y));
end; { Расстояние }

function SolutiePosibila(j,m:Indice):boolean;
begin
  if j<>m then SolutiePosibila:=true
          else SolutiePosibila:=false;
end; { Возможное решение }

procedure PrelucrareaSolutiei(A, B : Punct);
begin
  if Distanta(A, B)>dmax then
    begin
      PA:=A; PB:=B;
      dmax:=Distanta(A, B);
    end;
end; { Обработка решения }

begin
  write('Введите n='); readln(n);
  writeln('Введите координаты x, y точек');
  for j:=1 to n do
    begin
      write('P[', j, ']: '); readln(P[j].x, P[j].y);
    end;
  dmax:=0;
  for j:=1 to n do
    for m:=1 to n do
       if SolutiePosibila(j, m) then
          PrelucrareaSolutiei(P[j], P[m]);

  writeln('Решение: PA=(', PA.x:5:2, ' ', PA.y:5:2, ')');
  writeln('         PB=(', PB.x:5:2, ' ', PB.y:5:2, ')');
  readln;
end.
```

C++
```
// Program P152
// Точки на евклидовой плоскости
#include <iostream>
#include <math.h>
    using namespace std;
#define nmax 30

struct Punct {double x; double y;} P[30], PA, PB;
int m, n;
double dmax = 0; // максимальное расстояние

double Distanta(struct Punct A, struct Punct B)
{
    return sqrt(pow((A.x-B.x),2)+ pow((A.y-B.y),2));
} // Расстояние

bool SolutiePosibila(int j, int m)
{
  if (j != m)  return true;
         else return false;
} // Возможное решение

void PrelucrareaSolutiei(struct Punct A, struct Punct B)
{
  if ( Distanta(A, B) > dmax)
    {
        PA = A; PB = B;
        dmax = Distanta(A, B);
    }
} // Обработка решения

int main()
{
    cout <<" Введите n= "; cin >> n ;
    cout << "Введите координаты x, y точек  " << endl;
    for (int j = 1; j <= n; j++)
        {
        cout << " P[ " << j <<" ]: "; cin >> P[j].x >> P[j].y;
        }

    for (int j = 1; j<= n; j++)
        for (int m = j + 1; m <= n; m++)
          if (SolutiePosibila(j, m)) PrelucrareaSolutiei(P[j], P[m]);
    cout << " Решение: PA=(" << PA.x << " "<< PA.y << ")" << endl;
    cout << "          PB=(" << PB.x << " "<< PB.y << ")" << endl;
}
```

Временная сложность алгоритма, описанного с помощью программы P152 равна $O(n^2)$.

Рассмотрев данные примеры, замечаем, что в алгоритмах, основанных на методе перебора, вычисляется, явно или неявно, множество возможных решений S. В относительно простых задачах (*Пример 1*) элементы множества возможных решений могут быть перечислены непосредственно. В более сложных задачах (*Пример 2*) генерирова-

ние возможных решений требует разработки специального алгоритма. В общем случае, такие алгоритмы реализуют операции, связанные с обработкой конечных множеств:

– объединение;
– пересечение;
– разность;
– генерирование всевозможных подмножеств;
– генерирование элементов декартова произведения множеств;
– генерирование всевозможных перестановок, размещений или сочетаний элементов и т.п.

Главным достоинством алгоритмов полного перебора является то, что соответствующие программы относительно просты, а их отладка не требует сложных тестов. Временная сложность таких алгоритмов определяется количеством элементов k множества возможных решений S. Для большинства практически важных задач метод перебора приводит к экспоненциальным алгоритмам. Поскольку экспоненциальные алгоритмы неприемлемы в случае больших объемов входных данных, метод перебора применяется только в учебных целях или для разработки таких программ, время выполнения которых не является критичным.

Вопросы и задания

1. **Систематизируйте свои знания!** Объясните общую структуру алгоритмов полного перебора.
2. **Объясните!** Как может быть реализован перебор возможных решений с использованием циклов?
3. **Анализируйте и применяйте!** Оцените время выполнения программ `P151` и `P152`. Измените программу `P152` так, чтобы время ее выполнения уменьшилось примерно в два раза.
4. Приведите примеры программ, время выполнения которых не является критичным.
5. **Проанализируйте!** Какими преимуществами и недостатками обладают алгоритмы, основанные на методе полного перебора?
6. **Программируйте!** Рассмотрим множество $P = \{P_1, P_2, ..., P_n\}$, содержащее n точек $(3 \le n \le 30)$ на евклидовой плоскости. Каждая точка P_j определена своими координатами x_j, y_j. Напишите программу, которая находит такие три точки из множества P, для которых площадь соответствующего треугольника максимальна. Оцените время выполнения написанной программы.
7. **Программируйте!** Напишите функцию, которая, получив в качестве параметра натуральное число n, возвращает значение `true`, если n является простым, и `false` – в противном случае. Оцените временную сложность соответствующей функции.
8. **Программируйте!** В обозначении $(a)_x$ буква x представляет основание системы счисления, а буква a – число, записанное в соответствующей системе. Напишите программу, которая вычисляет, если существует, хотя бы один корень уравнения

$$(a)_x = b,$$

где a и b – натуральные числа, а x – неизвестное. Каждая цифра натурального числа a принадлежит множеству {0, 1, 2, ..., 9}, а число b записано в десятичной системе счисления. Например, корнем уравнения

$$(160)_x = 122$$

является $x = 8$, а уравнение

$$(5)_x = 10$$

не имеет решений. Оцените временную сложность разработанной программы.

9 **Программируйте!** В копилке находится N монет различных достоинств общим весом G граммов. Вес каждой монеты определенной стоимости дан в таблице, приведенной ниже:

Стоимость монеты, *лей*	**Вес монеты,** *грамм*
1	4,45
2	6,7
5	7,1
10	7,65

Напишите программу, которая определит:
a) минимальную сумму S_{min}, которая может находиться в копилке;
b) максимальную сумму S_{max}, которая может находиться в копилке.
Оцените временную сложность разработанной программы.

10 **Программируйте!** Напишите программу, которая определяет, сколько точек с целочисленными координатами содержится в сфере радиуса R с центром в начале системы координат. Подразумевается, что R – натуральное число, $1 \le R \le 30$. Расстояние d между точкой с координатами (x, y, z) и началом системы координат определяется по формуле $d = \sqrt{x^2 + y^2 + z^2}$. Оцените временную сложность разработанной программы.

2.8 Метод Greedy – алгоритм

Данный метод предполагает, что решаемые задачи имеют следующую структуру:
– задано множество $A=\{a_1, a_2, ..., a_n\}$, образованное из n элементов;
– требуется определить подмножество B, $B \subseteq A$, удовлетворяющее определенным условиям, чтобы оно могло быть принято в качестве решения.

В принципе, задачи рассматриваемого типа могут быть решены с помощью метода полного перебора путем последовательной генерации 2^n подмножеств A_i множества A. Недостаток метода полного перебора состоит в том, что время выполнения соответствующего алгоритма очень велико.

Чтобы избежать полного перебора всевозможных подмножеств A_i, $A_i \subseteq A$, в методе *Greedy* применяется **критерий (правило)** прямого выбора необходимых элементов из множества A. Обычно критерии отбора не заданы явно в описании задачи, и их формулировка является задачей программиста. Очевидно, что при отсутствии подобного критерия метод *Greedy* не может быть применен.

Общая схема алгоритма, основанного на методе *Greedy*, может быть представлена с помощью следующего цикла:

Pascal
```
while ExistaElemente do
begin
  AlegeUnElement(x);
  IncludeElementul(x);
end
```

C++
```
while (ExistaElemente)
{
  AlegeUnElement(x);
  IncludeElementul(x);
}
```

Функция `ExistaElemente` возвращает значение `true`, если во множестве A существуют элементы, удовлетворяющие критерию отбора. Подпрограмма `AlegeUnElement` извлекает из множества A такой элемент x, а процедура `IncludeElementul` записывает выбранный элемент в подмножество B. Первоначально подмножество B является пустым.

Как видим, в методе *Greedy* решение задачи ищется путем последовательного тестирования элементов множества A и включения некоторых из них в подмножество B. Образно говоря, подмножество B пытается «проглотить» «вкусные» элементы множества A, откуда и происходит название метода (*greedy* – жадный, ненасытный).

Пример. Дано множество $A=\{a_1, a_2, ..., a_i, ..., a_n\}$, элементы которого являются вещественными числами и хотя бы один из них удовлетворяет условию $a_i>0$. Напишите программу, которая вычисляет подмножество B, $B\subseteq A$, такое, чтобы сумма элементов подмножества B была максимальна. Например, для $A=\{21,5; -3,4; 0; -12,3; 83,6\}$ получим B={21,5; 83,6}.

Решение. Заметим, что, если подмножество B, $B\subseteq A$ содержит элемент $b\leq 0$, тогда сумма элементов подмножества $B \setminus \{b\}$ больше или равна сумме элементов B. Следовательно, правило отбора очень простое: на каждом шаге в подмножество B включается любой произвольный положительный элемент множества A.

В приведенной ниже программе множества A и B представлены векторами (одномерными массивами) `A` и `B`, а количество элементов каждого множества – целыми переменными соответственно `n` и `m`. Первоначально B – пустое множество, соответственно $m=0$.

Pascal
```
Program P153;
 { Техника Greedy }
const nmax=1000;
var A : array [1..nmax] of real;
    n : 1..nmax;
    B : array [1..nmax] of real;
    m : 0..nmax;
```

```
    x : real;
    i : 1..nmax;

Function ExistaElemente : boolean;
var i : integer;
begin
  ExistaElemente:=false;
  for i:=1 to n do
    if A[i]>0 then ExistaElemente:=true;
  end; { Элементы существуют }

procedure AlegeUnElement(var x : real);
var i : integer;
begin
  i:=1;
  while A[i]<=0 do i:=i+1;
  x:=A[i];
  A[i]:=0;
end; { Выбрать элемент }

procedure IncludeElementul(x : real);
begin
  m:=m+1;
  B[m]:=x;
end; { Включить элемент }

begin
  write('Dati n='); readln(n);
  writeln('Введите элементы множества A:');
  for i:=1 to n do read(A[i]);
  writeln;
  m:=0;
  while ExistaElemente do
    begin
      AlegeUnElement(x);
      IncludeElementul(x);
    end;
  writeln('Элементы множества B:');
  for i:=1 to m do writeln(B[i]);
  readln;
end.
```

Отметим, что процедура `AlegeUnElement` обнуляет компоненту массива `A`, содержащую элемент *x*, извлеченный из соответствующего множества.

C++

```
// Program P153
// Tehnica Greedy
#include <iostream>
    using namespace std;
#define nmax 1000

double A[nmax], B[nmax], k;
int n, m = 0, d;

int AlegeElementExistent()
{
    for (int i = 1; i<= n; i++)
        if (A[i] > 0) return i;
    return -1;
} // Выбор существующего элемента

void IncludeElementulAles(int k)
{
  m++;
  B[m] = A[k];
  A[k] = 0;
} // Включение выбранного элемента

int main()
{
    cout <<" Введите n= "; cin >> n ;
    cout << "Введите элементы множества A " << endl;
    for (int j = 1; j <= n; j++) cin >> A[j];
    do
    {
        d = AlegeElementExistent();
        if (d > 0) IncludeElementulAles (d);
    } while (d > 0);
    cout << " Элементы множества B: " << endl;
    for (int i = 1; i <= m; i++) cout << B[i] << " ";
}
```

В варианте C/C++ программы `P153` функция `AlegeElementExistent` комбинирует эффекты функций `ExistaElemente` и `AlegeUnElement`, а обнуление элементов массива `A`, переводимых в множество `B`, осуществляется в функции `IncludeElementulAles`.

Временная сложность алгоритмов, основанных на методе *Greedy*, может быть оценена в соответствии с общей схемой, представленной выше. Обычно время, затрачиваемое подпрограммами `ExistaElemente`, `AlegeUnElement` и `IncludeElementul`, составляет n. В составе цикла **`while`** эти подпрограммы выполняются не более n раз. Следовательно, алгоритмы, основанные на методе *Greedy*, представляют собой полиномиальные алгоритмы. Для сравнения отметим, что алгоритмы, основанные на переборе всех подмножеств A_i, $A_i \subseteq A$, являются алгоритмами порядка $O(2^n)$, т.е. экспоненциальными. К сожалению, метод *Greedy* применим только для задач, из условия которых может быть выведено правило, обеспечивающее прямой выбор нужных элементов из множества A.

Отметим тот факт, что метод *Greedy* не гарантирует нахождение оптимального решения, а точность метода зависит от критерия (правила) выбора, реализованного программистом.

Вопросы и задания

1. **Систематизируйте свои знания!** Объясните общую схему алгоритмов, основанных на методе *Greedy*.
2. **Проанализируйте!** Каковы преимущества и недостатки алгоритмов, основанных на методе *Greedy*?
3. **Практикуйтесь!** Оцените время выполнения программы `P153`.
4. **Тематическое исследование** Опишите в общих чертах алгоритм полного перебора, который определяет подмножество *B* из программы `P153`. Оцените временную сложность полученного алгоритма.
5. **Интегрируйте знания и применяйте!** **Хранение файлов на магнитных лентах.** Рассмотрим *n* файлов f_1, f_2, ..., f_n, которые должны быть записаны на магнитную ленту. Напишите программу, определяющую такой порядок размещения файлов на ленте, чтобы среднее время доступа к ним было минимальным. Предполагается, что частота обращения ко всем файлам (для чтения) одинакова, а на чтение файла f_i (i=1, 2, ..., n)затрачивается t_i секунд.
6. **Интегрируйте знания и применяйте!** **Непрерывная задача о рюкзаке.** Имеются *n* предметов. Для каждого предмета *i* (i=1, 2, ..., n) известен вес g_i и прибыль c_i, которая получается при транспортировке. Имеется рюкзак, в котором можно переносить один и более предметов, суммарный вес которых не превышает величины G_{max}. Напишите программу, которая определяет, каким образом необходимо загрузить рюкзак, чтобы суммарная прибыль *C* была максимальна. При необходимости переносимые предметы можно разделить на меньшие части.
7. **Интегрируйте знания и применяйте!** **Криковские подвалы.** После посещения знаменитых Криковских подвалов (*Хрубэ*)* один информатик сконструировал робота, который может двигаться по полю, разделенному на квадраты (*рис. 2.5*). Робот может выполнять следующие команды: `ВВЕРХ, ВНИЗ, НАПРАВО, НАЛЕВО`, в соответствии с которыми он перемещается в один из соседних квадратов. Если в квадрате имеется препятствие, например стена или бочка, то происходит столкновение и робот ломается.

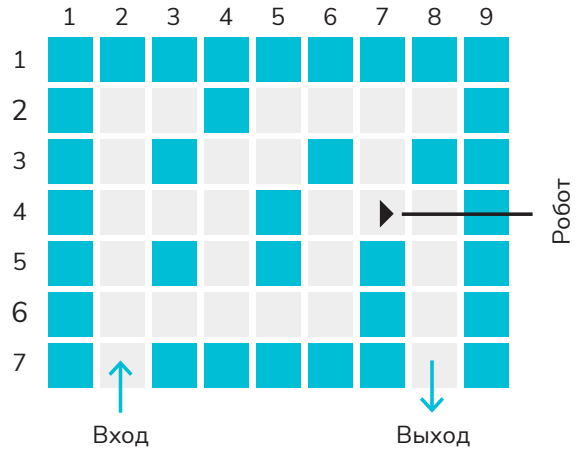


Рис. 2.5. Рабочее поле робота

* *Хрубэ* – подземное помещение или галерея, используемая для хранения продуктов питания. Лучшие вина Республики Молдова хранятся в подвалах Крикова уже несколько десятилетий.

Напишите программу, которая по известному плану подвалов проведет робота по подземным помещениям от входа в подвалы до выхода. Поскольку коллекция вин очень большая, нет необходимости посещать все подземные помещения Криковских подвалов.

Входные данные. План подвалов нарисован на листе бумаги в клетку. Закрашенные квадраты обозначают препятствия, а незакрашенные – свободные пространства. Квадраты по периметру плана, за исключением входа и выхода, закрашены по определению. В цифровом виде план подвалов представлен в виде массива A из m строк и n столбцов. Элементы $A[i, j]$ указанного массива имеют следующие значения: 0 – свободно; 1 – препятствие; 2 – вход в подвалы; 3 – выход из подвалов. Первоначально робот находится в квадрате, для которого $A[i, j]=2$.

Файл `HRUBE.IN` содержит в первой строке числа m, n, разделенные пробелом. В каждой из следующих m строк содержится по n чисел $A[i, j]$, разделенных пробелами.

Выходные данные. Файл `HRUBE.OUT` должен содержать в каждой строке по одной из команд `ВВЕРХ, ВНИЗ, НАПРАВО, НАЛЕВО`, записанных в порядке их выполнения роботом.

Ограничения. $5 \le m, n \le 100$. Время выполнения программы не должно превышать 3 секунды.

Пример:

`HRUBE.IN`

```
7 9
1 1 1 1 1 1 1 1 1
1 0 0 1 0 0 0 0 1
1 0 1 0 0 1 0 1 1
1 0 0 0 1 0 0 0 1
1 0 1 0 1 0 1 0 1
1 0 0 0 0 0 1 0 1
1 2 1 1 1 1 1 3 1
```

`HRUBE.OUT`

```
ВВЕРХ
НАПРАВО
НАПРАВО
НАПРАВО
НАПРАВО
ВВЕРХ
ВВЕРХ
НАПРАВО
НАПРАВО
ВНИЗ
ВНИЗ
ВНИЗ
```

Указание. На каждом шаге выбирайте из множества {`ВВЕРХ, ВНИЗ, НАПРАВО, НАЛЕВО`} по одной команде таким образом, чтобы робот всё время двигался только вдоль некоторой стены. Очевидно, что данный критерий (правило) выбора не гарантирует нахождение кратчайшего пути. Поскольку задача допускает несколько решений, программа рассчитает только одно из них.

Глава

III

Моделирование и численные методы

3.1 Понятие модели. Классификация моделей

Имитация процессов, предметов или явлений природы присуща человеческому обществу на протяжении его исторического развития. Первые рисунки, сделанные людьми каменной эпохи на стенах пещер, являются также и первой попыткой отобразить реальные предметы и явления природы с помощью изображений (*рис. 3.1*).

Глобус-макет нашей планеты также является имитацией реально существующего предмета. С его помощью мы узнаем о форме Земли, о ее движении, о местоположении континентов, океанов, стран и городов (*рис. 3.2*). Но элементы макета не представляют соответствующий ему реальный предмет полностью. Так, в случае глобуса-макета мы имеем дело со сферическим телом, через центр которого проходит ось, вокруг которой возможно вращение. На его поверхности напечатана различная информация о планете Земля. Глобус-макет отражает лишь определенные черты реального космического объекта, но отличается от него размерами, физическими свойствами, структурой и т.д. Глобус-макет – это лишь *модель*, являющаяся упрощенным представлением Земли/земного шара, позволяющая изучать лишь некоторые ее свойства.

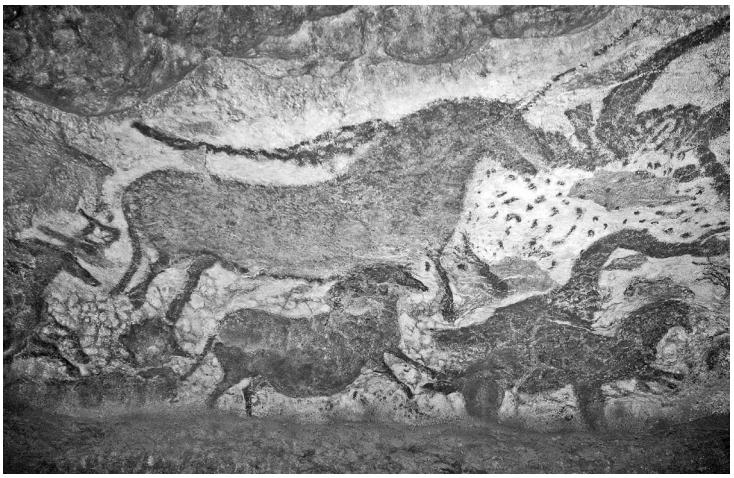


Рис. 3.1. Палеолитическое зооморфное изображение

Рис. 3.2. Модель глобуса Земли

Модель – это материальная либо идеальная, логико-математическая система, с помощью которой, путем аналогии, могут быть изучены свойства и действия реальной, значительно более сложной системы.

С давних времен модели используют для изучения природных явлений, сложных объектов и процессов, таких как молекулы, атомы, солнечная система, вселенная, атомный реактор, небоскреб и т.д. Удачная модель более удобна для исследований, нежели реальный объект. Более того, некоторые предметы и явления не могут быть изучены в оригинале. Например, сложно осуществить опыты над экономикой страны, невозможно осуществить опыты над планетами солнечной системы либо над уже прошедшими событиями.

Другим очень важным аспектом моделирования является возможность выделить, сосредоточить внимание на тех свойствах изучаемого реального объекта, которые определяют его суть. Модели также позволяют обучать правильному использованию реальных объектов и отрабатывать реакции, необходимые в различных ситуациях использования реальных объектов.

Опыты над реальными объектами могут быть невозможными либо опасными (продолжительность процесса во времени, возможность разрушения объекта). При изучения динамических объектов, характеристики которых меняются во времени, особую важность приобретает возможность прогнозирования состояния объекта под воздействием определенных факторов.

В сущности, хорошо построенная модель позволяет получить новые знания об изучаемом реальном объекте.

Процесс построения модели называется моделированием.

Существует несколько типов моделирования, которые могут быть объединены в две большие группы: *материальное моделирование* и *идеальное моделирование*.

В случае ***материального моделирования*** изучение оригинала осуществляется с помощью другого, более простого, материального предмета, называемого моделью, который обладает основными геометрическими, физическими, динамическими и функциональными характеристиками реального предмета. Примеры: макеты зданий, самолетов, военной техники и т.д.

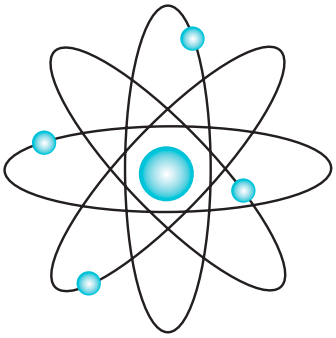
В случае ***идеального моделирования*** изучение оригинала осуществляется путем представления его свойств с помощью определенных концептов, схем, планов, структур, существующих лишь в воображении человека.

В общем, *идеальное моделирование* основывается на интуитивном представлении об изучаемом предмете. Например, опыт жизни каждого человека может служить личностной моделью окружающего мира. В случае, если модель не является интуитивной, а для ее создания используются различные символы, схемы, графики, формулы, то такое моделирование называют *символьным*. В категории символьного моделирования особое место занимает *математическое моделирование*. В данном случае изучение реального объекта осуществляется с помощью модели, описанной математическими средствами и понятиями. Классическим примером математического моделирования является описание и изучение математическими средствами основных законов механики Ньютона.

Для того чтобы изучить какой-либо процесс или явление, вовсе недостаточно создать его модель. В общем, модель описывает определенные закономерности, связи, свойства оригинала, однако цель моделирования заключается в получении на основе информации, предоставляемой моделью, новых данных об оригинале.

Вопросы и задания

1. **Систематизируйте свои знания!** Объясните понятие *модель*. Приведите примеры реально существующих в окружающем мире предметов и их моделей. В каждом конкретном случае уточните характеристики реальных предметов, которые отражены в их модели.
2. **Объясните!** Сформулируйте понятие *материальной модели*. Приведите примеры материальных моделей. Обоснуйте необходимость применения материальных моделей в различных областях науки и техники.
3. **Объясните!** Сформулируйте понятие *идеальной модели*. Приведите примеры идеальных моделей. Обоснуйте необходимость применения идеальных моделей.
4. **Интегрируйте!** В 1911-м году исследователь Е. Резерфорд предложил «планетарную (ядерную) модель атома». Объясните смысл слова «модель» в предложении Резерфорда, а также обоснуйте, почему эта модель называется планетарной.
5. Объясните смысл следующего высказывания: «Пиктограммы графического интерфейса пользователя представляют собой модели реальных объектов, обрабатываемых программными продуктами».



3.2 Математическая модель и математическое моделирование

Одной из базовых целей информатики как межпредметной науки является разработка методов решения сложных исследовательских и вычислительных задач с помощью компьютера. Первоначально информатика развивалась как область прикладной математики. Первые задачи, решаемые средствами информатики, были чисто математическими, а их решение сводилось к осуществлению большого числа сложных вычислений. В настоящее время информатика стала независимой наукой, основанной на математических законах, обладающей собственными методами и объектами исследований. Информатика изучает и решает сложные задачи из области математики, физики, химии, биологии, экономики, экологии, филологии и социологии. Однако независимо от области, которой принадлежит задача, информатика при ее решении основывается на математике. И, как следствие, для решения любой задачи на компьютере сначала необходимо с помощью математических понятий описать процессы и явления, происходящие в ней. В частности, это могут быть функции, уравнения, неравенства, системы уравнений и др.

Математическая модель представляет собой описание некоторого процесса либо явления с помощью математических понятий.

Пример 1: Даны два автомобиля. Один из них движется прямолинейно, траектория его движения описывается уравнением $x = 1/2$. Второй автомобиль перемещается по круговой траектории, описываемой окружностью с единичным радиусом и центром, совпадающим с началом системы координат (*рис. 3.3*). Требуется определить координаты точек возможного столкновения автомобилей.

Абстрагируя задачу, получим следующую формулировку: одна материальная точка движется по траектории, описываемой уравнением $x = 1/2$. Другая материальная точка движется по траектории, описываемой уравнением $x^2 + y^2 = 1$. Требуется найти решение системы уравнений:

$$\begin{cases} x = \dfrac{1}{2} \\ x^2 + y^2 = 1. \end{cases}$$

Графически данная задача представлена на схеме справа.

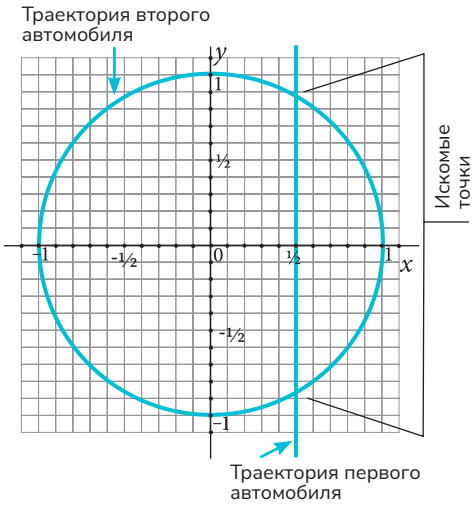


Рис. 3.3. Движение автомобилей

Алгоритм решения задачи:

1. Пусть $x = 1/2$.

2. Подставим это значение во второе уравнение системы и вычислим

$$y_1 = \sqrt{1 - \frac{1}{4}}; \qquad y_2 = -\sqrt{1 - \frac{1}{4}}.$$

3. Получим следующее решение задачи: $\left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right)$; $\left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)$.

Пример 2: На гладком столе находится металлический шар, прикрепленный к пружине (*рис. 3.4*). Пружина сжимается, без риска потери упругости, а затем отпускается. Требуется определить координаты шара через t секунд.

Если k – коэффициент упругости пружины, m – масса шара, x – величина упругой деформации пружины, то на основании *закона Гука* и *второго закона Ньютона* математическая модель системы *шар–пружина* имеет форму:

$ma = -kx$, где a – ускорение.

Обозначим через x_0 положение шара после сжатия пружины (первоначальная упругая деформация) (*рис. 3.5*). Требуется определить положение шара (величину упругой деформации) через t секунд. Для этого преобразуем предыдущую модель таким образом, чтобы отразить зависимость величины x (упругой деформации) от времени. Воспользуемся законом гармонического движения, предположив, что деформации в нашем случае малы и что силой трения можно пренебречь.

$$x(t) = x_0 \cos \sqrt{\frac{k}{m}}\, t.$$

Данная формула позволяет определить положение шара x (упругую деформацию пружины) в любой момент времени t в случае, если известны величины k, m и x_0.

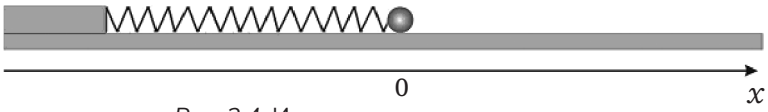


Рис. 3.4. Исходное состояние системы

Рис. 3.5. Состояние системы после сжатия пружины

Таким образом, общая задача определения упругой деформации в любой момент времени может быть решена с помощью, полученной выше формулы и следующей программы:

```
program cn01;
var
t, x0,k,m,x: real;
begin
 readln(x0,k,m,t);
 x:=x0*cos(sqrt(k/m*t));
 writeln('x=', x:0:6);
end.
```

```
// program cn01;
#include <iostream>
#include <math.h>
 using namespace std;
double t, x0,k,m,x;
int main()
{
    cin >> x0 >> k >> m >>t;
    x =x0 * cos(sqrt(k / m * t));
    cout << "x = " << x;
    return 0;
}
```

Вопросы и задания

1. **Систематизируйте свои знания!** Дайте определение понятию *математическая модель*. Является ли такая модель материальной?
2. **Интегрируйте!** Укажите несколько областей применения математических моделей. Обоснуйте необходимость их применения.
3. **Аргументируйте!** Обоснуйте необходимость представления математических моделей в виде компьютерных программ и их использования в дальнейшем.
4. **Объясните!** Напишите программу, которая на основании задачи из *примера 2* будет подсчитывать упругую деформацию пружины x через каждую секунду с момента начала движения и до момента t (t имеет целое значение). Значения x_0, k, m, t считываются с клавиатуры.
5. **Объясните!** Радиоактивный изотоп плутония-235 имеет период полураспада, равный 26 минутам. За этот период половина начального количества изотопов исчезает, распавшись на другие химические элементы. Опишите математическую модель, позволяющую подсчитать число циклов полураспада, необходимое для исчезновения k процентов от начального количества изотопов.
6. **Примените!** Эффект лекарства подсчитывается по формуле $r_k = \alpha r_{k-1} + 0{,}4^k$, где r_k – концентрация активных веществ через k часов после приема лекарства. Первоначально $r_0 = 1$, а $0 < \alpha < 1$. Из формулы следует, что r_k достигает своего максимального значения r_m по истечении m часов, затем его значение начинает уменьшаться. Напишите программу для определения того, через сколько часов эффект от приема лекарства достигнет максимального значения. Действительное число α считывается с клавиатуры.

3.3 Аналитические и имитационные решения

Для начала проанализируем следующую задачу:

Пример 1: Бассейн объемом в 100 $м^3$, первоначально содержащий 20 $м^3$ воды, заполняется с помощью насоса мощностью 15 $м^3$/час. Одновременно через устройство для фильтрации из бассейна вытекают 5 $м^3$/час воды. Требуется определить, через сколько часов бассейн наполнится.

Несомненно, существует несколько способов решения данной задачи. Один из самых простых состоит в имитации процесса накапливания воды в бассейне через каждый час с помощью следующей таблицы.

Время (часы)	Количество накачиваемой воды	Количество вытекшей воды	Количество воды в бассейне
0	0	0	20
1	15	5	30
2	30	10	40
3	45	15	50
4	60	20	60
5	75	25	70
6	90	30	80
7	105	35	90
8	120	40	100

Решение задачи было получено путем достаточно большого числа последовательных расчетов, с помощью которых объем воды в бассейне динамически реконструировался через каждый час. Безусловно, это не самый эффективный метод решения: потребовалось большое количество промежуточных результатов, а также неоправданно большое число операций.

Другое решение основано на применении формулы, позволяющей произвести прямой расчет конечного результата. Необходимое для наполнения бассейна водой время определяется как отношение разницы общего и начального объемов воды в бассейне к количеству прироста воды в течение часа:

$$t = \frac{100 - 20}{15 - 5} = 8.$$

В первом случае для решения был использован часто повторяющийся процесс, при котором объем воды определялся через каждый интервал времени (1 час), а новый результат рассчитывался с учетом данных, полученных на предыдущем этапе. Другими словами, был смоделирован процесс этапов наполнения бассейна. Такой процесс называется *имитацией*.

Имитация **является техникой решения задач, основанной на применении математических и логических моделей, которые описывают поведение реальной системы в пространстве и/или во времени.**

Имитационная модель **– это модель для решения задачи, основанного на технике имитации. Решения, полученные в процессе имитации, называются имитационными.**

Обычно имитационные модели используют в тех случаях, когда необходимы не только конечные, но и промежуточные результаты.

Второй способ решения рассмотренной задачи основан на прямых расчетах по формуле:

$$t = \frac{V_{бассейна} - V_{начальное\_кол}}{C_{насоса} - C_{стока}},$$ где $V_{бассейна}$ – общий объем бассейна;

$V_{начальное\_кол}$ – начальное количество воды в бассейне;

$C_{насоса}$ – мощность насоса;

$C_{стока}$ – пропускная способность стока.

Формула позволяет вычислить время наполнения бассейна *любого объема*, с *произвольным начальным количеством* воды, *насосом любой мощности*, превосходящей *пропускную способность стока*, которая, в свою очередь, тоже является произвольной.

Метод решения задач, основанный на применении формул, позволяющих делать прямой расчет конечного результата, не требующий анализа промежуточных состояний, называется аналитическим методом. Решения, полученные таким методом, называются аналитическими решениями.

Пример 2: Требуется написать программу для вычисления суммы первых n членов геометрической прогрессии, зная величину первого члена a_1, ($a_1 > 0$) и знаменатель прогрессии q, ($q < 1$).

Из математики известна формула $S_n = \frac{a_1(q^n - 1)}{q - 1}$, позволяющая напрямую, без нахождения всех членов прогрессии, вычислять требуемую сумму. Известна также и рекуррентная формула $a_n = q \times a_{n-1}$ для вычисления n-го члена прогрессии ($n \geq 2$). Следовательно, необходимый результат можно получить с помощью итеративного процесса, подсчитывая по рекуррентной формуле каждый последующий член прогрессии и добавляя его к сумме $S_n = a_1 + a_2 + \ldots + a_{n-1} + a_n$.

Pascal

Прямой расчет суммы

```
program cn02;
var S,a,q : real;
  n : integer;
begin
  readln(a, q, n);
  S:=a*(exp(n*ln(q))-1)/(q-1);
  writeln('S=', S:0:6);
end.
```

Итеративный расчет суммы

```
program cn03;
var S,a,q : real;
  i, n : integer;
begin
  readln(a, q, n);
  S:=0;
  for i:=1 to n do
    begin
      S:=S+a;
      a:=a*q;
    end;
  writeln('S=', S:0:6);
end.
```

C++

```
// program cn02
#include <iostream>
#include <math.h>
  using namespace std;

double S, a, q;
int n;

int main()
{
  cin >> a >> q >> n;
  S = a * (exp(n*log(q)) - 1)/
(q - 1);
  cout << "S = " << S;
  return 0;
 }
```

```
// program cn03;
#include <iostream>
  using namespace std;

double S, a, q;
int n;

int main()
{
  cin >> a>> q >> n;
  S = 0;
  for (int i = 1; i <= n; i++)
  {
    S = S + a;
    a = a * q;
  }
  cout << "S = " << S;
  return 0;
}
```

Каждый из приведенных выше методов имеет свои достоинства и недостатки. В некоторых задачах бывает достаточно сложно, а иногда и невозможно определить аналитическую формулу (например, координаты кометы или астероида в зависимости от времени). В других случаях бывает достаточно сложно создать адекватную имитационную модель, даже если использовать очень большое число промежуточных расчетов.

Аналитическое решение позволяет делать прямой расчет конечного результата, однако, не позволяет исследовать динамику его построения. Использование итеративного процесса (имитационного решения) позволяет динамически конструировать решение, в зависимости от данных задачи, а также на каждом новом шаге проводить проверку полученного промежуточного результата. В то же время для получения имитационных решений требуется значительно большее количество операций, чем для аналитических решений.

Выбор метода решения задачи зависит от многих факторов. Приведем основные из них:

- возможность нахождения аналитического решения;
- необходимость исследования промежуточных решений (состояний);
- время, необходимое для расчетов (в случае имитационного решения);
- погрешность имитационного решения (разница между точным решением и решением, полученным с помощью вычислительного эксперимента).

Вопросы и задания

1 **Объясните!** Дайте определение понятию *имитация*. Что такое имитационное решение?

2 **Систематизируйте свои знания!** Что понимается под аналитическим методом решения задачи? Что представляет собой аналитическое решение? Какими свойствами обладают аналитические решения?

3 **Проанализируйте!** Перечислите свойства имитационных решений. Какие из этих свойств предполагают использование компьютера для нахождения такого типа решений?

4 **Объясните!** Разработайте имитационный метод для нахождения n-го элемента числового ряда 1, 2, 3, 5, 8, 13, 21, ... Существует ли аналитический метод решения данной задачи?

5 **Интегрируйте знания и применяйте!** Решите следующие задачи имитационным методом:

a) Божья коровка в течение дня поднимается по столбу на 5 м, а в течение ночи – опускается по нему на 3 м. Подъем она начинает утром. Высота столба равна 15 м. Через сколько дней божья коровка достигнет вершины столба?

b) В условиях предыдущей задачи божья коровка начинает движение утром с высоты равной 6 м.

c) В условиях предыдущей задачи божья коровка начинает движение сразу после наступления ночи.

6 **Программируйте!** Напишите программу для вычисления суммы первых n членов арифметической прогрессии, зная величину первого члена a_1, ($a_1 > 0$) и шаг r, ($r > 0$).

3.4 Этапы решения задачи на компьютере

Инструменты информатики позволяют решать задачи как аналитическими, так и имитационными методами. Решение любой задачи независимо от применяемого метода состоит из нескольких этапов. Каждый из этих этапов очень важен.

Анализ задачи. Это этап изучения содержания задачи. На этом этапе определяется набор начальных данных и ожидаемый результат, устанавливаются связи между начальными данными и результатом. Также тут устанавливаются дополнительные ограничения на начальные данные и результаты.

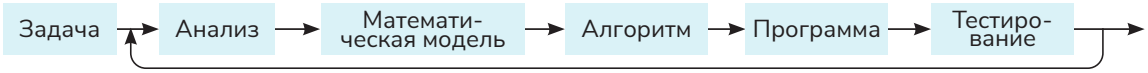
Создание математической модели задачи. На этом этапе начальные данные описываются с помощью математических структур. Описываются, применяя язык математики, отношения, позволяющие получить из начальных данных результат. В зависимости от задачи эти отношения могут быть рекуррентными (создается имитационная модель), либо позволяющими прямой расчет результата (аналитическая модель). На этом же этапе (если необходимо) задача разбивается на подзадачи. В данном случае математическая модель создается для каждой подзадачи в отдельности.

Разработка алгоритма. Алгоритм решения задачи на компьютере содержит множество команд, описанных в установленной форме (псевдокод, логическая схема и др.), которые необходимо выполнить для решения задачи, а также порядок их выполнения (шаги алгоритма). Если задача была разбита на подзадачи, то дополнительно к описанию подалгоритмов необходимо описать алгоритм, определяющий способы и условия их вызова.

Написание программы. Для того чтобы решение задачи было выполнено на компьютере, алгоритм необходимо преобразовать в форму, ему понятную, – ***программу***, написанную на ***языке программирования***. Шаги алгоритма представляются в виде операторов языка программирования, а порядок выполнения определяется их последовательностью и структурой. Начальные и промежуточные данные описываются с помощью структур данных, допустимых в языке программирования. В процессе написания программы возможны как синтаксические, так и семантические ошибки. Процесс их исправления также является частью этапа написания программы. Данный этап считается завершенным тогда, когда компиляция либо интерпретация программы проходит без ошибок.

Тестирование программы. Успешная компиляция еще не является гарантом правильного решения задачи. Для проверки правильности решения проводится ряд тестов, устанавливающих правильность результатов, генерируемых программой, в зависимости от простых, средней сложности и экстремальных начальных данных. Если для всех проведенных тестов программа генерирует правильный результат, то можно предположить, что задача решена верно. Если же в результате тестирования полученные результаты отличаются от ожидаемых, то необходимо возобновить решение задачи, начиная с первого этапа – *анализа задачи*.

Процесс решения задачи на компьютере можно проиллюстрировать с помощью следующей схемы:



Пример:

Задача:

В лабораторных условиях популяция вирусов, первоначально состоящая из N единиц и помещенная в стерильную среду, уменьшается каждый час на 50 процентов, если количество вирусов в начале часа четное, либо увеличивается на единицу, если количество вирусов в начале часа нечетное. В момент, когда количество вирусов становится меньше числа C – критического для выживания, данная популяция полностью исчезает.

Задание: Напишите программу, с помощью которой будет определено время в часах, необходимое для полного исчезновения в лабораторных условиях популяции из N ($N < 32\,000$) вирусов, имеющих критическое для выживания число C ($1 < C < N$).

Анализ задачи

Популяция, состоящая из четного количества вирусов, уменьшается вдвое в течение часа. Популяция, состоящая из нечетного количества вирусов, вырастает на единицу и стано-вится четной. Повторный рост популяции невозможен, а ее уменьшение вдвое происходит минимум один раз за каждые два часа. Следовательно, значение C ($C > 1$) будет достигнуто за конечное число часов.

Математическая модель

Число популяции в момент времени $t = 0$ задано числом $N_0 = N$.

Число популяции по истечении t часов стерилизации ($t > 0$) задается рекуррентной формулой:

$$N_t = \begin{cases} \dfrac{N_{t-1}}{2}, & \text{если } N_{t-1} \text{ четное} \\ N_{t-1} + 1, & \text{если } N_{t-1} \text{ нечетное} \end{cases}$$

Алгоритм

Шаг 0. *Инициализация:* Вводятся значения N, C. $t \Leftarrow 0$.*

Шаг 1.

a) По истечении одного часа: $t \Leftarrow t+1$.

b) Ремоделирование популяции: если $N \Leftarrow N$ div 2, иначе $N \Leftarrow N+1$.

Шаг 2. Проверка условия выживания: если $N < C$, выводится значение t. КОНЕЦ. В противном случае возобновляется выполнение *шага 1*.

---

* Здесь и далее выражение $a \Leftarrow b$ означает: a получает значение b.

Программа

```
program cn04;
var N,C,t: integer;
begin
 readln(N, C); t:=0;
 while (N>=C) do begin
   t:=t+1;
   if N mod 2 = 1 then N:=N+1 else N:= N div 2;
   end;
 writeln(t);
end.
```

```
// program cn04;
#include <iostream>
 using namespace std;
int N, C, t;

int main()
{
 cin >> N >> C;
 t = 0;
 while ( N >= C )
 {
   t++;
   if (N % 2 == 1) N++; else N = N / 2;
 }
 cout << t;
 return 0;
}
```

Тестирование

Для проверки правильности решения задачи были использованы следующие наборы данных:

Начальные данные	Результат	Начальные данные	Результат	Начальные данные	Результат	Начальные данные	Результат
512 2	9	31999 16	15	331 330	2	332 330	1
25768 235	10	31999 2	18	31997 2	19	3 2	3

Вопросы и задания

1. **Объясните!** Перечислите этапы решения задачи на компьютере. Объясните необходимость каждого этапа.
2. **Систематизируйте свои знания!** Какие методы описания алгоритма решения задачи вы знаете? Приведите примеры.
3. **Проанализируйте!** Каковы последствия разбиения задачи на подзадачи? Приведите примеры задач, которые могут быть разбиты на подзадачи. Укажите две задачи, содержащие одинаковые подзадачи.
4. **Примените!** Для следующих задач разработайте математические модели:
   a) Известны координаты трех вершин прямоугольника со сторонами, параллельными осям системы координат. Требуется определить координаты четвертой вершины.
   b) Решите предыдущую задачу в условиях, когда стороны прямоугольника могут располагаться произвольно относительно осей системы координат.
5. **Разработайте!** Для следующих задач опишите алгоритмы решения, воспользовавшись для этого известными методами:
   a) Задана последовательность целых чисел, количество которых не превосходит 100. Требуется расположить элементы последовательности в возрастающем порядке.
   b) Задана последовательность целых чисел, количество которых не превосходит 100. Требуется определить за один проход элемент с максимальным значением, а также число его повторений в последовательности.
6. **Программируйте!** Напишите программы для решения задач из заданий 4 и 5.
7. **Интегрируйте и применяйте!** Придумайте наборы тестов для программ, написанных в задании 6.

3.5 Приближенные числа. Абсолютная и относительная погрешности

Число a называется приближением числа A, если их значения отличаются незначительно и в расчетах число a может заменить A. Если имеет место отношение $a < A$, a называется приближением с недостатком, если же $a > A$ – приближением с избытком. Например, для числа π значение 3,14 является приближением с недостатком, а 3,142 – приближением с избытком. Действительно $3{,}14 < \pi < 3{,}142$. Приближенные значения возникают в результате проводимых измерений, а разница между приближенным и точным значениями может быть обусловлена множеством факторов: температурными условиями, давлением, влажностью, качеством измерительных приборов, квалификацией специалиста, проводящего измерения и др.

Под погрешностью Δ_a понимается разность $A–a$ (иногда и $a–A$). В зависимости от значений a и A, Δa может быть отрицательной либо положительной. Для получения точного числа A к приближенному числу a добавляется значение погрешности Δ_a:

$$A = a + \Delta_a.$$

Часто бывает известно приближение a и неизвестен знак приближения. В таких случаях используют *абсолютную погрешность*, определяемую следующим образом:

Абсолютная погрешность Δ приближенного значения a – это модуль разницы между точным значением A и приближенным значением a:

$$\Delta = |A - a| .$$

Абсолютная погрешность не является достаточным показателем для оценивания точности расчетов либо измерений. Пусть в результате измерения двух балок длиной в 20 м и 6 м были получены соответственно результаты в 20,5 м и 6,2 м.

Несмотря на то, что абсолютная погрешность в первом случае больше, очевидно всё же, что измерения первой балки были более точными, чем измерения второй. Для того, чтобы определить качество измерений (расчетов), абсолютная величина ошибки относится к единице длины (или, в общем случае, к соответствующей единице измерения).

Инфо+

Представление чисел в компьютере предполагает использование приближенных чисел с конечным количеством цифр. Участвующие в записи числа цифры, начиная с первой, отличной от нуля, называются *значащими цифрами.*

Пример: 0,04708 имеет четыре значащие цифры. Первые два нуля лишь фиксируют положение десятичной точки.

Приближение a ($a_n a_{n-1} a_{n-2} \cdots a_0$) точного числа A имеет k точных значащих цифр $a_i a_{i-1} \cdots a_{i-k+1}$, если:

$$|A - a| \le \frac{1}{2} 10^{i-k+1}.$$

Относительная погрешность δ приближенного значения – это отношение абсолютной погрешности Δ к модулю точного числа A ($A \neq 0$):

$$\delta = \frac{\Delta}{|A|}.$$

Пример: Длина балки равна 100 см. В результате ее измерения было получено значение 101 см. Расстояние между пунктами A и B равно 3 000 м. В результате его измерения было получено значение в 2 997 м. Необходимо определить относительную погрешность для каждого измерения и установить более точное из них.

Решение:

для первого измерения $\Delta = |100 - 101| = 1$ cm, $\delta = \frac{1}{100} = 0,01$;

для второго измерения $\Delta = |3\,000 - 2\,997| = 3$ m, $\delta = \frac{3}{3\,000} = 0,001$.

Так как относительная погрешность второго измерения меньше, оно является более точным.

Вопросы и задания

1. **Объясните!** Поясните понятие погрешности. Приведите примеры возникновения погрешностей в реальных ситуациях.
2. **Объясните!** Дайте определение понятию *абсолютная погрешность*. Приведите примеры.
3. **Проанализируйте!** Дайте определение понятию *относительная погрешность*. Приведите примеры. Как изменится формула для расчета относительной погрешности, если она должна выражаться в % от исследуемого точного значения?
4. **Примените!** Длина пути между двумя населенными пунктами согласно дорожному указателю равна 230 км. С помощью измерительного прибора во время движения автомобиля было зафиксировано расстояние в 230,7 км. Считая значение на указателе точным, определите абсолютную и относительную погрешность данного измерения.

5 **Примените!** Балка, точная длина которой равна 100 см, была измерена с абсолютной погрешностью, равной 2 см. Каковы возможные значения длины, полученные в результате измерений?

6 **Примените!** Точный объем сосуда равен 20 л. Измерения объема были проведены с относительной погрешностью, равной 0,001. Каковы возможные значения объема, полученные в результате измерений?

3.6 Источники вычислительных погрешностей

Источники погрешностей, появляющихся в процессе решения задач, очень разнообразны. Знание этих источников позволяет избежать их и минимизировать накопительный эффект погрешностей. Чаще всего встречаются следующие типы погрешностей:

a) *погрешности задачи*; b) *погрешности метода*; c) *погрешности входных данных*; d) *погрешности приближения*; e) *погрешности округления.*

Погрешности задачи. Этот тип погрешностей появляется в ситуациях когда выбранная математическая модель не описывает исследуемый объект абсолютно точно. Так, в *примере* 2 (3.2. Математическая модель и математическое моделирование) для построения математической модели системы *шар–пружина* было использовано уравнение гармонических колебаний в условиях *малых деформаций* и отсутствия *силы трения*. Следовательно, полученный с помощью формулы результат будет отличен от точного, и чем больше будет сила трения и деформация пружины в реальной системе, тем более значительной будет погрешность.

Погрешности метода. Этот тип погрешностей возникает в результате невозможности нахождения точного метода решения задачи, либо в результате ограничений, приводящих к необходимости использовать для решения задачи менее точные методы. В этом случае используются эвристические методы, что может привести к значительным различиям между генерируемым и точным решением задачи.

Хорошим примером для иллюстрации вышесказанного может служить решение *задачи о рюкзаке* методом *Гриди* (Greedy).

Погрешности входных данных. Математическое моделирование часто основывается на результатах, полученных опытным путем, то есть на числовых последовательностях, полученных в результате измерений. Эти значения не являются точными (например: расстояние, масса, скорость).

Пусть некоторое тело движется по траектории, описываемой на отрезке [0,1] функцией $f(x) = x^2 + x + 1$. Известно, что значение аргумента x вычисляется с абсолютной погрешностью, не превосходящей 0,01. Следовательно, если z – точное значение аргумента, то $|z - x| < 0{,}01$.

Вспомним!

Задача о рюкзаке:

Пусть дан рюкзак объемом S и n предметов объемами v_i, $i = 1...n$ и стоимостью c_i, $i = 1...n$. Необходимо заполнить рюкзак предметами из данного набора таким образом, чтобы суммарная стоимость положенных в рюкзак предметов была максимально возможной. Если применить для решения задачи метод *Гриди*, в большинстве случаев полученное решение будет отличаться от оптимального. Так, для набора из пяти предметов объемами 5, 7, 13, 20, 10 и стоимостью соответственно, 4, 8, 15, 23, 5, а также рюкзака объемом 30 единиц, метод *Гриди* сгенерирует решение, равное 27 (предметы 3, 2, 1). (Сортировка предметов производится в порядке убывания отношения стоимость/объем.) В действительности оптимальное решение равно 31 (предметы 2 и 4).

В этих условиях можно определить, в какой мере погрешность измерения величины x влияет на результаты вычислений:

$$|f(x)-f(z)| = |x^2+x+1-z^2-z-1| =$$
$$= |x^2+x-z^2-z| = |(x-z)(x+z+1)|.$$

Так как $x+z+1 \le 3$ şi $|z-x| < 0,01$, следует $|f(x)-f(z)| \le 0,03$.

Разность между значением функции для точного (z) и приближенного (x) аргументов не превосходит постоянное число. Отсюда следует, что погрешность результата не зависит от x, а только от точности, с которой он был измерен, и функция является устойчивой к погрешностям.

Погрешности приближения. Это тип погрешностей, генерируемых некоторыми математическими определениями и понятиями. Их наличие приемлемо в задачах, использующих понятия предела, сходимости и др. Появление этого типа погрешностей обосновано самой структурой определений, содержащих в себе элементы приближения.

Пример: Последовательность $\{x_n\}$ является сходящейся и имеет предел x, если для любого $\varepsilon > 0$, $\varepsilon \to 0$, существует ранг $n(n_\varepsilon)$ такой, что $\forall n > n_\varepsilon . |x-x_n| < \varepsilon$.

С точки зрения численного анализа ε отражает точность, с которой x_n приближает x.

В процессе вычисления предела используется последовательное приближение к точному пределу, который, однако, не достигается. Процесс прерывается тогда, когда разница (погрешность) становится меньше максимально допустимой погрешности (ε).

Необходимо отметить, что результаты, полученные численными методами, допустимы лишь в качестве количественных и не могут служить доказательством определенных математических высказываний.

Погрешности округления. Это особый тип погрешностей, обоснованный тем фактом, что в компьютере числовые значения, участвующие в расчетах, сохраняются с ограниченным количеством десятичных цифр после запятой. Примерами могут служить константа π, значения тригонометрических функций и др.

Вспомним!

Считается, что функция $f(x)$: $I \to R$ обладает свойством Лившица, если существует константа $m > 0$, такая, что: $|f(x) - f(z)| \le m^* |z-x|$, $\forall x, z \in I$.

Считается, что определение значений функции $f(x)$ устойчиво к погрешностям, если $f(x)$ обладает свойством Лившица. Другими словами, устойчивость функции к погрешностям предполагает малые изменения значений функции при малых изменениях аргумента.

Вспомним!

Приближение значений членов ряда $1/n$ к его пределу – 0.

$n = 1$ $\quad 1/n = 1$
$n = 2$ $\quad 1/n = 0,5$
...
$n = 501$ $\quad 1/n = 0,001996$
$n = 502$ $\quad 1/n = 0,001992$
$n = 503$ $\quad 1/n = 0,001988$
...
$n = 999$ $\quad 1/n = 0,001001$
$n = 1000$ $\quad 1/n = 0,001000$
$n = 1001$ $\quad 1/n = 0,000999$
...

Пример: Пусть в процессе вычислений можно оперировать лишь тремя цифрами после запятой. В этом случае для чисел:
$a = 0,2334$, $b = 0,2331$, $c = 0,233$

получим $rd(rd(a+b)+c) =$
$= rd(rd(0,4665)+0,233) =$
$= rd(0,467+0,233) =$ **0,7**;

$rd(a+rd(b+c)) =$
$= rd(0,2334+rd(0,4661)) =$
$= rd(0,2334+0,466) =$
$= rd(0,6994) =$ **0,699**.

Пусть $A = \{a_1, a_2, a_3, ..., a_n,\}$ множество всех чисел, которые могут быть представлены в компьютере. (Считается, что $a_1 < a_2 < a_3 < ... < a_n$.)

Любое из чисел $\frac{a_i + a_{i+1}}{2}$ отсутствует в заданном множестве A. В случае появления подобной ситуации при расчетах возникает погрешность, обусловленная заменой действительного результата – его самым близким приближением из данного множества A. Эта процедура называется *округлением.*

Функция округления, реализованная в компьютере, определена следующим образом:

$$\forall x \in R,\ a_i \in A,\ a_{i+1} \in A \quad x \in (a_i,\ a_{i+1}),$$

$$rd(x) = a_i \text{ если } x \in \left(a_i, \frac{a_i + a_{i+1}}{2} \right),$$

$$rd(x) = a_{i+1} \text{ если } x \in \left[\frac{a_i + a_{i+1}}{2}, a_{i+1} \right).$$

Применение функции округления приводит к отклонениям в основных законах арифметических операций, которые уже не являются ассоциативными, дистрибутивными.

В случае округления результатов вычислений, погрешности по модулю не превосходят значение $0{,}5 \cdot 10^{-n}$ (эти погрешности называются *абсолютными погрешностями* округления), где n есть число значащих цифр (которые могут быть восприняты) в процессе вычислений. Погрешности округления могут быть как положительными, так и отрицательными. В случае их чередования имеет место процесс погашения, в результате которого конечная погрешность не возрастает вместе с числом произведенных вычислений.

Вопросы и задания

1. **Систематизируйте свои знания!** Укажите основные источники погрешностей. Приведите примеры.
2. **Примените!** Приведите примеры погрешностей, обусловленных невозможностью точной формулировки задачи. Аргументируйте невозможность точной формулировки задачи.
3. **Проанализируйте!** Какие функции обладают свойством устойчивости вычисляемых значений к погрешностям?
4. **Примените!** Является ли устойчивым к погрешностям вычисление значений линейной функции? А функции $y = \sqrt{x},\ x \in [1,2]$?
5. Объясните суть погрешностей приближения.
6. **Исследование случая!** Решение некоторой задачи вычисляется итеративно, с помощью формулы: $x_0 = 0,\ x_{i+1} = \sqrt{2 + x_i}$. Достигнет ли точного результата последовательность вычисляемых значений? Определите опытным или аналитическим путем точное решение. Через какое количество итераций абсолютная погрешность решения станет меньше 0,0001?
7. **Объясните!** Почему результаты, полученные численными методами, не могут служить доказательством истинности математических высказываний?
8. **Объясните!** Какова причина возникновения погрешностей округления? Приведите примеры.
9. **Экспериментируйте!** Напишите программу, вычисляющую произведение и частное чисел 1,00000001 и 0,999999999. Значения сохраните с помощью переменных типа **`real`**. Проанализируйте полученные результаты. Объясните причины появления погрешностей.

3.7 Отделение корней алгебраических и трансцендентных уравнений

Решить алгебраическое или трансцендентное уравнение (далее просто *уравнение*) $f(x) = 0$ означает определить те значения переменной x, при которых равенство $f(x) = 0$ становится истинным. В случае несложных уравнений их решение можно относительно просто найти аналитическими способами, изучаемыми в лицейском курсе математики. Если же структура уравнения сложная, то решение уравнения становится достаточно трудной процедурой. Более того, в случаях, когда уравнение моделирует определенные реальные ситуации, явления, зависящие от нескольких параметров, а значения этих параметров известны лишь приближенно, понятие точного решения вообще теряет смысл. Поэтому бывает полезно знать и приближенные методы решения уравнений, а также алгоритмы, их реализующие.

Пусть дано уравнение

$$f(x) = 0, \qquad (1)$$

$f(x)$ – функция, определенная и непрерывная на интервале $a \le x \le b$.

Любое значение ξ, для которого выражение f (ξ) = 0 является истинным, называется нулем функции f (x), либо корнем уравнения $f(x) = 0$.

В дальнейшем предполагается, что уравнение (1) имеет различные (изолированные) корни, то есть для каждого корня существует соседняя область, не содержащая других решений.

Таким образом, решение уравнений численными методами разбивается на два этапа:

1. Отделение интервалов, на которых уравнение имеет единственное решение.

2. Уменьшение, насколько это возможно, каждого из изолированных интервалов (если требуется найти все решения уравнения) либо одного из них (если надо определить лишь одно решение).

Инфо+

Если функция $f(x)$ записана либо может быть приведена к форме полинома, уравнение $f(x) = 0$ называется ***алгебраическим***.

Пример: $3x-7 = 0$.

В противном случае, когда функция $f(x)$ не полиномиальная, уравнение называется ***трансцендентным***.

Пример: $\sin(2x) + \sqrt{\cos^2 x + e^x} = 0$.

Вспомним!

Теорема. Если функция $f(x)$, непрерывная на отрезке $[a, b]$, принимает на концах отрезка значения, разные по знаку ($f(a) \times f(b) < 0$), тогда на этом отрезке существует по крайней мере одна точка ξ такая, что $f(\xi) = 0$. Если на $[a, b]$ существует производная $f'(x)$, непрерывная и сохраняющая знак, тогда ξ является единственным решением уравнения $f(x) = 0$ на данном отрезке.

Аналитический метод. Для аналитического отделения корней можно воспользоваться свойствами производной.

Если решение уравнения $f'(x)=0$ можно легко вычислить, то для отделения корней уравнения $f(x)=0$, необходимо:

1. Найти различные корни $a \le x_1 \le x_2 \le \ldots, \le x_n \le b$ уравнения $f'(x)=0$;

2. При условии, что $a = x_0$ и $b = x_{n+1}$, вычислить значения $f(x_0), f(x_1), \ldots, f(x_{n+1})$. Отрезки $[x_i, x_{i+1}]$, $i = 0, \ldots, n$, для которых $f(x_i) \times f(x_{i+1}) < 0$, содержат хотя бы одно решение уравнения $f(x)=0$.

Пример 1: Определите число решений уравнения $e^x + x = 0$:

$$f'(x) = e^x + 1; \quad f'(x) > 0 \ \forall \ x \in R.$$

Так как $\lim\limits_{x \to -\infty} f(x) = -\infty$, $\lim\limits_{x \to \infty} f(x) = \infty$, то заданное уравнение имеет единственное решение.

Пример 2: Отделите корни уравнения $x^3 - 9x^2 + 24x - 19 = 0$ на отрезке $[0, 8]$.

Решение:

$$f(x) = x^3 - 9x^2 + 24x - 19;$$

$$f'(x) = 3x^2 - 18x + 24.$$

Решив уравнение $f'(x) = 0$, получим корни $x_1 = 2$, $x_2 = 4$.

x	$f(x)$	Знак $f(x)$
0	−19	−
2	1	+
4	−3	−
8	109	+

Таким образом, начальное уравнение имеет 3 корня, по одному на каждом из отрезков [0, 2], [2, 4], [4, 8].

Графический метод. Другая возможность отделения корней уравнения $f(x) = 0$ состоит в прямом исследовании графика функции $f(x)$. Для его построения можно использовать как специальные программные приложения* , так и простые программы, написанные средствами языка программирования высокого уровня.

Графический метод разделения корней уравнения на заданном отрезке можно реализовать и локально, с помощью электронных таблиц. Для этого достаточно построить таблицу с двумя столбцами. Первый столбец будет представлять собой разбиение отрезка на элементарные отрезки равной длины. Второй столбец будет содержать формулу, которая вычисляет значения функции $f(x)$ для соответствующих значений из первого столбца. На основании данных из столбца, содержащего значения $f(x)$, строится линейная диаграмма, которая является графиком исследуемой функции.

Пример: $f(x) = x^{\cos(2x)} + 3\sin(x)$. Исследуется наличие корней на отрезке [0,2, 10] (*рис. 3.6а, 3.6b*).

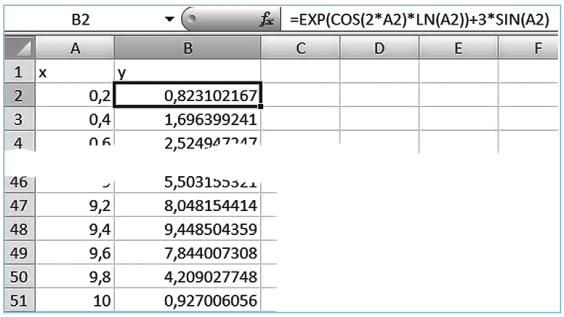


Рис. 3.6а. Функция $f(x)$ представлена таблично в электронной таблице. Столбец A содержит значения x от 0,2 до 10 с шагом 0,2. Столбец B содержит значения $f(x)$

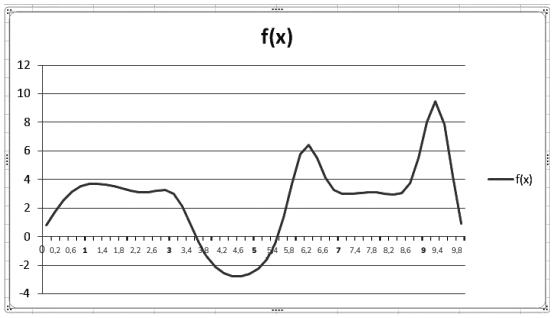


Рис. 3.6б. Функция $f(x)$ представлена графически на основании данных таблицы. Легко идентифицируются два произвольно выбранных отрезка, каждый из которых содержит ровно по одному корню уравнения $f(x) = 0$, например: [3, 5] и [5, 6]

* MatCAD, Matematica, Derive и др.

Вопросы и задания

1. Что называется решением уравнения?
2. **Объясните!** Каким условиям должна удовлетворять функция $f(x)$, для того, чтобы на заданном отрезке был хотя бы один корень уравнения $f(x)= 0$? А для того, чтобы этот корень был единственным?
3. **Практикуйтесь!** Определите аналитически количество действительных корней уравнения:
   a) $x^5 - 5x + 7 = 0$; b) $x^3 - 9x^2 + 24x - 13 = 0$.
4. **Программируйте!** Напишите программу, которая для функции $f(x)$, непрерывной на $[a, b]$, производит разбиение заданного отрезка на n равных по длине отрезков и выводит в качестве результата все отрезки, на концах которых функция имеет противоположные по знаку значения:
   a) $f(x) = x^3 - 7x^2 + 12x - 37$ на $[-10, 10]$, $n = 100$;
   b) $f(x) = \sin(3x) + 3\cos(x) - 1$ на $[-2, 2]$, $n = 100$.
5. **Примените!** Отделите наиболее подходящим способом корни уравнений:
   a) $\frac{x^4}{4} + x^3 - \frac{x^2}{2} - 3x - 8 = 0$;
   b) $2x^3 - 6x^2 - 48x + 17 = 0$;
   c) $x^4 - 14x^2 - 24x - 4 = 0$;
   d) $\frac{1}{2} - e^{-x^2} = 0$;
   e) $e^x(x^2 - 2x + 2) = 0$;
   f) $x[\sin(\ln(x)) - \cos(\ln(x))] = 0$;
   g) $\ln(x) + \sqrt{\sin(3x) + 7} - x^2 = 0$.

3.8 Метод половинного деления

Пусть дана функция $f(x)$, непрерывная на отрезке $[a, b]$, и $f(a) \times f(b) < 0$. Требуется определить на отрезке $[a, b]$ одно из решений уравнения

$$f(x) = 0. \qquad (1)$$

Свойства функции гарантируют существование хотя бы одного корня на данном отрезке.

Одним из самых простых методов определения корня уравнения $f(x) = 0$ является метод половинного деления. Метод предполагает нахождение значения c – середины отрезка $[a, c]$, затем вычисление значения $f(c)$. Если $f(c) = 0$, тогда c является точным решением уравнения.

В противном случае решение ищется на одном из отрезков $[a, c]$ или $[c, b]$. Оно принадлежит тому отрезку, на концах которого функция имеет противоположные по знаку значения (*рис. 3.7*).

Если $f(a) \times f(c) > 0$, то решение далее ищется на отрезке $[a_1, b_1]$, где a_1 получает значение c, а b_1 – значение b. В противном случае a_1 получает значение a, b_1 – значение c. Процесс деления пополам возобновляется для отрезка $[a_1, b_1]$ и повторяется до тех пор, пока не будет получено точное решение, либо (в подавляющем большинстве случаев!) отклонение вычисленного решения c_i от точного станет достаточно малым.

В процессе подряд идущих разбиений получается последовательность отрезков

$[a_0, b_0]$, $[a_1, b_1]$, $[a_2, b_2]$, ..., $[a_i, b_i]$, ...

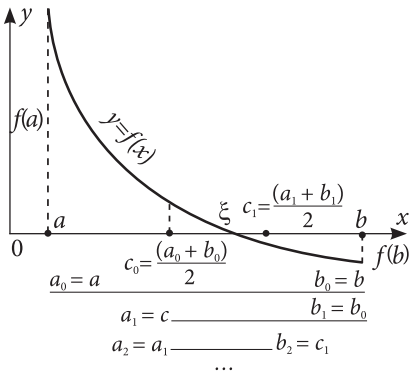


Рис. 3.7. Последовательное вычисление отрезков, содержащих решение уравнения f(x)=0

Для каждого из них имеет место отношение

$$f(a_i) \times f(b_i) < 0,\ i = 0, 1, 2, \ldots . \qquad (2)$$

Оценка погрешности. Так как точное решение ξ уравнения является точкой отрезка $[a_i, b_i]$, то следует, что разность между точным и вычисленным решениями не превосходит длину отрезка. Таким образом, локализация решения на отрезке величиной в ε гарантирует погрешность вычисления решения, не превосходящую значение ε:

$$|\xi - c_i| < \varepsilon = |b_i - a_i|.$$

Алгоритмизация метода

Исходя из математического описания метода половинного деления, можем выделить два разных случая остановки процесса вычислений решения уравнения $f(x) = 0$:

А1. ***Алгоритм вычисления решения для заданного числа последовательных делений пополам:***

Шаг 0. *Инициализация:* $i \Leftarrow 0$.

Шаг 1. Нахождение середины отрезка $c \Leftarrow \frac{a+b}{2}$.

Шаг 2. Уменьшение отрезка, содержащего решение: если $f(c) = 0$, то найдено решение $x = c$. КОНЕЦ.
В противном случае, если $f(a) \times f(c) > 0$, то $a \Leftarrow c$; $b \Leftarrow b$, иначе $a \Leftarrow a$; $b \Leftarrow c$.

Шаг 3. $i \Leftarrow i + 1$. Если $i = n$, то вычисленное решение равно $x = \frac{a+b}{2}$. КОНЕЦ.
В противном случае возвращаемся к *шагу 1*.

А2. ***Алгоритм вычисления с заданной точностью* ε:***

Шаг 1. Определение середины отрезка $c \Leftarrow \frac{a+b}{2}$.

Шаг 2. Если $f(c) = 0$, то найдено решение $x = c$. КОНЕЦ.
В противном случае, если $f(a) \times f(c) > 0$, то $a \Leftarrow c$; $b \Leftarrow b$, иначе $a \Leftarrow a$; $b \Leftarrow c$.

Шаг 3. Если $|b - a| < \varepsilon$, то вычислено решение $x = \frac{a+b}{2}$. КОНЕЦ.
В противном случае возвращаемся к *шагу 1*.

Пример 1: Определите корень уравнения $x^4 + 2x^3 - x - 1 = 0$ на отрезке $[0, 1]$ произведя 16 последовательных делений пополам.

Так как число последовательных приближений задано, а концы отрезка известны, необходимые присваивания значений переменным производятся в программе.

```
program cn05;
var  a,b,c: real;
   i,n:integer;

function f(x:real):real;
begin   f:=sqr(sqr(x))+2*x*sqr(x)-x-1;end;
begin   a:=0; b:=1; n:=16;
    for i:=1 to n do
     begin  c:=(b+a)/2;
       writeln('i=',i:3,' x=',c:10:8,' f(x)=',f(c):12:8);
       if f(c)=0 then break**
```

* В данном контексте точность ε означает погрешность вычислений, которая не превосходит значение ε.

** Инструкция безусловного перехода, прерывающая выполнение циклической инструкции, в которой она содержится.

```
      else if f(c)*f(a)>0 then a:=c else b:=c;
    end;
end.
```

Результаты:
```
i= 1 x=0.50000000 f(x)= -1.18750000
i= 2 x=0.75000000 f(x)= -0.58984375
...
i= 15 x=0.86679077 f(x)= 0.00018565
i= 16 x=0.86677551 f(x)= 0.00009238
```

```
// program cn05;
#include <iostream>
#include <iomanip>
#include <math.h>
    using namespace std;

 double a, b, c;
 int i, n;
 double f(double x)
 {
   return pow(x,4)+2*x*pow(x,2)-x-1;
 }

int main()
{
 a = 0; b = 1; n = 16;
 for (i = 1; i <= n; i++)
    {
   c = (b + a) / 2;
   cout <<  setprecision(8) <<"i=" << i << " x=" << c << "
f(x)="<<f(c) << endl;
   if (f(c) == 0) break;
   else if (f(c) * f(a) > 0 )  a = c; else b = c;
    }
 return 0;
}
```

Результаты:
```
i=1 x=0.5 f(x)=-1.1875
i=2 x=0.75 f(x)=-0.58984375
...
i=15 x=0.86679077 f(x)=0.00018565479
i=16 x=0.86677551 f(x)=9.2381174e-05
```

Пример 2: Определите корень уравнения $6\cos(x) + 8\sin(x) = 0$ на отрезке [2, 4] с точностью $\varepsilon = 0{,}00017$.

```
program cn06;
var  a,b,c,eps: real;

function f(x:real):real;
begin   f:=6*cos(x)+8*sin(x);end;
```

```
begin   a:=2; b:=4; eps:=0.00017;
    repeat
     c:=(b+a)/2;
     writeln('x=',c:10:8,' f(x)=',f(c):12:8);
     if f(c)=0 then break
     else if f(c)*f(a)>0 then a:=c else b:=c;
   until abs(b-a)<eps;
end.
```

Результаты:
```
x=3.00000000 f(x)= -4.81099492
x=2.50000000 f(x)= -0.01908454
...
x=2.49829102 f(x)= -0.00199471
x=2.49816895 f(x)= -0.00077401
```

```
// program cn06;
#include <iostream>
#include <iomanip>
#include <math.h>
    using namespace std;

 double a, b, c, eps;
 int i, n;
 double f(double x)
 {
   return 6 * cos(x) + 8 * sin(x);
 }

int main()
{
  a = 2; b = 4; eps = 0.00017;
  do
   {
  c = (b + a) / 2;
  cout <<  setprecision(8) << " x=" << c << " f(x)=" << f(c) << endl;
  if (f(c) == 0) break;
  else if (f(c) * f(a) > 0 )  a = c; else b = c;
  } while (abs(a - b) > eps);
 return 0;
}
```

Результаты:
```
x=3 f(x)=-4.8109949
x=2.5 f(x)=-0.01908454
...
x=2.498291 f(x)=-0.0019947083
x=2.4981689 f(x)=-0.00077400516
```

Вопросы и задания

1 **Объясните!** В каких случаях применяются приближенные методы решения алгебраических уравнений?
Опишите метод половинного деления. Какими преимуществами он обладает? А недостатками? Формула для оценивания погрешности, приведенная в данном параграфе, имеет вид $|\xi - c_i| < \varepsilon = |b_i - a_i|$. Выразите разность между точным и вычисленным значениями только через значения концов исходного отрезка и количество осуществляемых делений пополам.

2 Опишите алгоритм метода половинного деления.

3 **Примените!** Определите методом половинного деления решения уравнений:
1. $e^x - x^2 = 0$ на [–1, –0,5];
2. $x^3 - x - 1 = 0$ на [1, 2];
3. $x^3 + 3x^2 - 3 = 0$ на [–3, –2];
4. $x^5 - x - 2 = 0$ на [1, 2].

a) для 10, 20, 40 делений начального отрезка;
b) с точностью ε = 0,001; 0,0001; 0,00001;
c) в условиях предыдущего задания определите количество делений пополам, необходимое для достижения требуемой точности.

4 **Экспериментируйте!** Определите методом половинного деления решения на отделенных интервалах уравнений, в заданиях 3 и 5, стр. 105, для 10, 20, 30 последовательных делений пополам.

3.9 Метод хорд

Метод половинного деления, несмотря на его простоту, является малоэффективным в случаях, когда решение необходимо получить за небольшое количество итераций, но с большой точностью. В таких случаях более пригоден *метод хорд*, состоящий в разбиении начального отрезка на части, определяемые с помощью точки пересечения хорды – отрезка, соединяющего точки, соответствующие значениям функции на концах отрезка, с осью $0x$.

Пусть дана функция $f(x)$, обладающая следующими свойствами:

1. $f(x)$ непрерывна на отрезке $[a, b]$ и $f(a) \times f(c) < 0$.
2. На отрезке $[a, b]$ существуют $f'(x) \neq 0; f''(x) \neq 0$, непрерывные и сохраняющие свой знак на отрезке $[a, b]$.

Перечисленные свойства гарантируют существование единственного решения уравнения $f(x) = 0$ на $[a, b]$.

Метод хорд предполагает выбор в качестве приближения решения точки пересечения прямой, проходящей через точки $(a, f(a))$ и $(b, f(b))$ с осью $0x$.

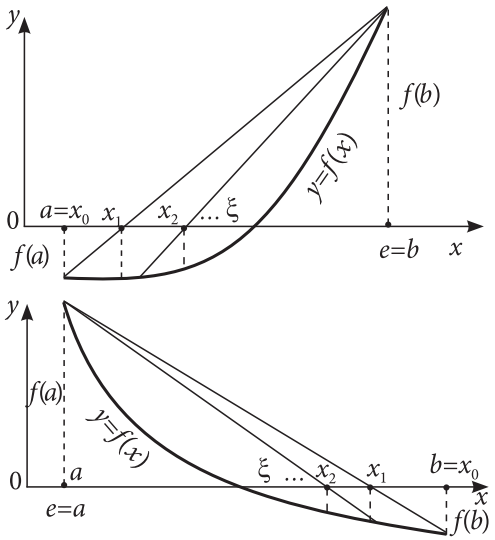


Рис. 3.8. Последовательный подход к решению уравнения методом хорд

Для реализации метода выбирается конец отрезка $[a, b]$, через который будут проведены несколько хорд (*рис. 3.8*). Выбор данного конца определяется условием:

$$f(e) \times f''(e) > 0.$$

Другой конец отрезка $[a, b]$ считается начальным приближением решения: x_0.

Через точки $(e, f(e))$ и $(x_0, f(x_0))$ проводится хорда. Определяется точка x_1, в которой хорда пересекает ось $0x$. Точка x_1 считается следующим приближением решения.

Процесс повторяется, следующая хорда проводится через точки $(e, f(e))$ и $(x_1, f(x_1))$. Таким образом, получается последовательность приближений $x_0, x_1, x_2, \ldots x_i, x_{i+1}, \ldots x_n \ldots$, пределом которой является точное решение уравнения $f(x) = 0$.

Точки e и x_0 известны. Воспользовавшись уравнением прямой, проходящей через две точки, можем определить приближение x_1 ($f(x_1) = 0$):

$$\frac{x - x_0}{e - x_0} = \frac{y - f(x_0)}{f(e) - f(x_0)},$$

откуда

$$x_1 = x_0 - \frac{f(x_0)}{f(e) - f(x_0)}(e - x_0).$$

Таким образом, зная приближение x_{i-1}, можем вычислить следующее приближение x_i с помощью рекуррентной формулы:

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f(e) - f(x_{i-1})}(e - x_{i-1}), \qquad (3)$$

$$i = 1, 2, \ldots .$$

Доказано, что последовательность значений $x_1, x_2, \ldots x_i, x_{i+1}, \ldots x_n \ldots$, вычисленных с помощью формулы (3), сходится к решению ξ уравнения $f(x) = 0$.

Погрешность метода

Из того факта, что последовательность приближений, полученных методом хорд, сходится к точному решению уравнения, можно сделать следующее заключение: погрешность вычисленного решения обратно пропорциональна количеству сделанных итераций.

Алгоритмизация метода

Применение метода хорд диктует необходимость предварительного исследования функции $f(x)$, с целью установить фиксированный конец отрезка, через который будут проведены все хорды. Число приближений n может быть задано в условии задачи либо получено из некоторых условий. В обоих случаях вычисления производятся по формуле (3). Критерием остановки будет повторение расчетов по формуле (3) n раз.

Определение фиксированного конца. Чтобы избежать необходимости вычисления $f''(x)$, поступим следующим образом: определим знак $f(x)$ в c – точке пересечения прямой, проходящей через точки $(a, f(a))$ и $(b, f(b))$, с осью $0x$. **Фиксированным будет тот конец отрезка [a, b], для которого выполняется условие: $f(e) \times f(c) < 0$.**

А1. ***Алгоритм вычисления для случая заданного числа последовательных приближений:***

Шаг 1. Определение фиксированного конца e и начального приближения x_0:

$$c \Leftarrow a - \frac{f(a)}{f(b)-f(a)}(b-a);$$

если $f(c) \times f(a) < 0$, тогда $e \Leftarrow a$, $x_0 \Leftarrow b$, иначе $e \Leftarrow b$, $x_0 \Leftarrow a$; $i \Leftarrow 0$.

Шаг 2. Вычисление x_{i+1} согласно формуле $x_{i+1} = x_i - \frac{f(x_i)}{f(e)-f(x_i)}(e-x_i)$.

Шаг 3. Если $i + 1 = n$, то вычисленное решение $x \Leftarrow x_i$. КОНЕЦ.
В противном случае, $i \Leftarrow i$+1 и возвращаемся к *шагу 2*.

Пример: Пусть дана функция $f(x)$ = ln (x sin x). Вычислить приближенное решение уравнения $f(x)$ = 0 на отрезке [0,5; 1,5] для 10 последовательных приближений с использованием метода хорд.

Для этого примера нет необходимости в предварительных математических выкладках. Так как концы отрезка и число необходимых приближений заданы, соответствующие присваивания будут проведены непосредственно в тексте программы.

```
program cn07;
var  a,b,e,c,x: real;
   n,i: integer;

function f(x:real):real;
begin   f:=ln(x*sin(x));end;
begin   a:=0.5; b:=1.5; n:=10;
    {задание фиксированного конца отрезка и начального приближения x0}
    c:=a-(f(a))/(f(b)-f(a))*(b-a);
    if f(c)*f(a)>0 then begin e:=b; x:=a; end
    else begin e:=a; x:=b; end;
     {итерационный расчет решения}
     for i:=1 to n do
       begin  x:= x-(f(x))/(f(e)-f(x))*(e-x);
     writeln(x:10:8,' ',f(x):12:8);
       end;
end.
```

Результаты:

```
i= 1 x=1.27995775 f(x)= 0.20392348
i= 2 x=1.18251377 f(x)= 0.09028687
...
i= 9 x=1.11427651 f(x)= 0.00016577
i= 10 x=1.11420523 f(x)= 0.00006678
```

```
// program cn07;
#include <iostream>
#include <iomanip>
#include <math.h>
    using namespace std;
 double a, b, c, e, x;
 int i, n;

double f(double x)
{
```

```
    return log(x * sin(x));
}
int main()
{
 a = 0.5; b = 1.5; n = 10;
 // задание фиксированного конца отрезка и начального приближения x0}
 c = a - f(a) / (f(b) - f(a)) * (b - a);
 if (f(c) * f(a) > 0) { e = b; x = a;}
 else { e = a; x = b; }
 // итерационный расчет решения}
 for (i = 1; i <= n; i++)
    {
        x = x - (f(x)) / (f(e) - f(x)) * (e-x);
        cout <<  setprecision(8) << " x=" << x << " f(x)=" << f(x) <<
endl;
    }
 return 0;
}
```

Результаты:

```
x=1.2799577 f(x)=0.20392348
x=1.1825138 f(x)=0.090286871
...
x=1.1142765 f(x)=0.00016577026
x=1.1142052 f(x)=6.6781584e-05
```

Вопросы и задания

1 **Систематизируйте свои знания!** Объясните суть метода хорд. Опишите метод хорд графически.

2 **Проанализируйте!** Как зависит фиксированный конец отрезка от знака $f''(x)$?

3 **Систематизируйте свои знания!** Опишите процесс определения фиксированного конца отрезка. Как можно избежать определения $f''(x)$?

4 **Систематизируйте свои знания!** Опишите пошаговый алгоритм метода хорд для случая заданного числа итераций.

5 **Примените!** Определите решения уравнений методом хорд для 10, 20, 30 итераций:

a) $x^3 - 0{,}2x^2 + 0{,}2x - 2{,}1 = 0$ на $[1, 2]$;

b) $5x^3 - 20x + 3 = 0$ на $[0, 1]$;

c) $e^x - x^2 = 0$ на $[-1, 0]$.

6 **Интегрируйте!** Отделите корни следующих уравнений. Решите уравнения методом хорд:

a) $\operatorname{tg}(0{,}55x + 0{,}1) - x^2 = 0$ для 5, 25 итераций;

b) $x^3 - 0{,}2x^2 + 0{,}5x + 1{,}5 = 0$ для 3, 9, 27 итераций.

7 **Проанализируйте!** Определите методом хорд на отделенных отрезках решения уравнений из упражнений 3 и 5, стр. 105, для 10, 20, 30 итераций. Сравните результаты с полученными в упражнении 4, стр. 109. Объясните расхождения.

8 **Примените!** Пусть дана функция $f(x) = x[\sin(\ln(x)) - \cos(\ln(x))]$. Определите решение уравнения $f(x) = 0$ 0 на отрезке [2, 3] с точностью ε = 0,0001, применив метод хорд.

3.10 Приближенное вычисление определенного интеграла методом прямоугольников

Одним из наиболее часто применяемых приложений численных методов является численное интегрирование. Прямые методы вычисления определенного интеграла не всегда позволяют найти аналитическое решение, часто бывает, что не известна формула интегрируемой функции. Обычно в таких случаях задано лишь некоторое число точек, для которых известно значение интегрируемой функции. В данной ситуации, вычисление интеграла возможно лишь приближенными методами (предполагается, что интегрируемая функция является непрерывной на отрезке интегрирования).

Из курса математики известно, что геометрическим смыслом определенного интеграла $\int_a^b f(x)dx$ является площадь криволинейной трапеции, определяемой осью $0x$, прямыми $x = a$, $x = b$ и графиком функции $f(x)$ на отрезке $[a, b]$ (*рис. 3.9*).

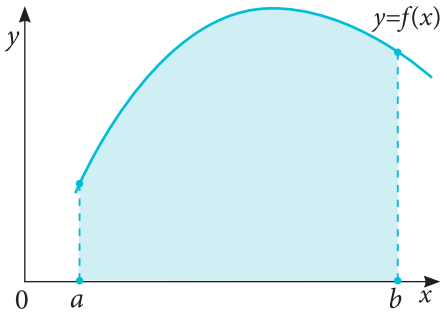


Рис. 3.9. Геометрический смысл определенного интеграла

Точное вычисление площади данной фигуры не всегда возможно. В таких случаях возможным решением могла бы послужить аппроксимация начальной фигуры множеством других фигур, площадь которых легко вычисляется. Тогда значение определенного интеграла (площадь начальной фигуры) приблизительно будет равно сумме вычисляемых площадей данных фигур.

Пусть f – функция, дифференцируемая на отрезке $[a, b]$. Требуется вычислить $\int_a^b f(x)dx$.

Для численного решения данной задачи построим разбиение отрезка $[a, b]$ с помощью последовательности точек вида $a=x_0<x_1<...<x_{n-1}<x_n<=b$. Пары последовательных точек $x_0x_1, x_1x_2, ..., x_ix_{i+1}, ..., x_{n-1}x_n$ определяют n различных отрезков, объединение которых равно $[a, b]$. Для простоты точки данного разбиения будем считать равноудаленными (*рис. 3.10*). Тогда длина h каждого элементарного отрезка $[x_i, x_{i+1}]$ определяется по формуле

$$h = \frac{|b-a|}{n}.$$

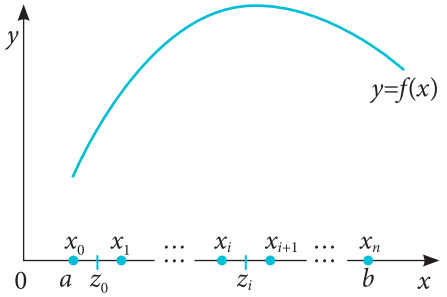


Рис. 3.10. Разбиение $[a, b]$ на элементарные отрезки. Середина элементарного отрезка $[x_i, x_{i+1}]$, точка $z_i = \frac{x_i + x_{i+1}}{2}$

Значения x_i также могут быть выражены через известные величины: $x_i = a + ih$, $i = 0, ..., n$.

Зная значения x_i и x_{i+1}, можно вычислить середину каждого элементарного отрезка $z_i = \frac{x_i + x_{i+1}}{2}$. На каждом из отрезков $[x_i, x_{i+1}]$ площадь криволиней-

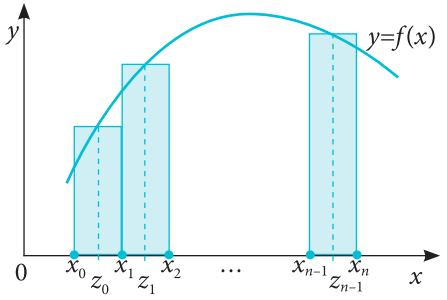


Рис. 3.11. Последовательное построение прямоугольников, апроксимирующих начальную фигуру. На каждом элементарном отрезке высота прямоугольника равна значению функции $f(x)$ *в середине отрезка*

ной трапеции аппроксимируется площадью прямоугольника S_i, со сторонами h и $f(z_i)$ (*рис. 3.11*):

$$S_i = hf(z_i) = hf(a + ih + \frac{h}{2}).$$

В данном случае вычисленное значение определенного интеграла I равно сумме площадей прямоугольников и вычисляется по формуле

$$I = h\sum_{i=0}^{n-1} f\left(a + ih + \frac{h}{2}\right),$$

где

n – число разбиений начального отрезка;

h – длина элементарного отрезка;

I – вычисленное значение интеграла.

Таким образом, вычисление интеграла сводится к вычислению значения некоторого арифметического выражения, зависящего лишь от числа разбиений отрезка интегрирования и значения функции в серединах элементарных отрезков. Метод, сводящий вычисление интеграла к вычислению суммы площадей прямоугольников, называется *методом прямоугольников.*

Погрешность метода

Вычисленное значение интеграла отличается от точного, определенного аналитически. Погрешность возникает вследствие того, что на каждом элементарном отрезке функция $f(x)$ аппроксимируется постоянной функцией g, а величина погрешности ε определяется в результате интегрирования ошибки приближения по следующей формуле:

$$\varepsilon = \left| \int_a^b f(x)dx - I \right| \le (b-a)M\frac{h}{4},$$

где M – супремум $|f''(x)|$ на $[a, b]$, I – вычисленное значение интеграла.

Из приведенной формулы следует важное заключение: погрешность метода прямоугольников пропорциональна числу разбиений отрезка интегрирования. Так, чтобы полученная в результате вычислений погрешность не превышала заданное значение ε, достаточно разбить отрезок интегрирования на n элементарных отрезков, получив значение n с помощью следующих преобразований:

$$(b-a)M\frac{h}{4} \le \varepsilon \Rightarrow \frac{(b-a)^2 M}{4n} \le \varepsilon \Rightarrow n = \left[\frac{(b-a)^2 M}{4\varepsilon}\right] + 1.$$

Алгоритмизация метода

Так как для случая вычислений с точностью, не превосходящей заданное значение ε, необходимое число разбиений можно установить априори, достаточно реализовать лишь один алгоритм – для заданного числа разбиений n:

Шаг 1. *Инициализация:* вводятся значения концов отрезка интегрирования a, b и число разбиений n.

Примечание. Для случая вычислений с точностью, не превосходящей заданное значение ε, необходимое число разбиений n считается по формуле $n = \left[\frac{(b-a)^2 M}{4\varepsilon}\right] + 1$.

Шаг 2. Вычисляется длина элементарного отрезка $h = \frac{|b-a|}{n}$. $S \Leftarrow 0$.

Шаг 3. Для всех i от 0 до n–1:

a) вычисляется значение $z_i \Leftarrow a + ih + \frac{h}{2}$;

b) вычисляется площадь элементарного прямоугольника: $S_i \Leftarrow f(z_i) \times h$;

c) вычисленная площадь добавляется к сумме предыдущих: $S \Leftarrow S + S_i$.

Шаг 4. Выводится на экран вычисленная общая площадь *S*. КОНЕЦ.

Пример 1: Требуется написать программу для вычисления интеграла $\int_1^2 \frac{x^4}{\sqrt{1+x}}\,dx$ для 20, 40, 80 и 160 разбиений отрезка интегрирования методом прямоугольников. Для каждого числа разбиений необходимо вывести вычисленное значение с шестью знаками после запятой.

Решение: Так как число разбиений указано в условии задачи, нет необходимости в предварительной математической обработке.

```
program cn09;
const r=4;
var S, a, b, h : real;
 j,i,n:integer;
function f(x:real): real;
begin
 f:=x*x*x*x/sqrt(1+x);
end;
begin
 a:=1; b:=2; n:=10;
 for j:=1 to r do
 begin
 S:=0; n:=n*2; h:=(b-a)/n;
 for i:=0 to n-1 do
 S:=S+ h*f(a + i*h + h/2);
 writeln (' n=',n,' I=',S:0:6);
 end;
end.
```

Результаты:

```
n=20 I=3.788513
n=40 I=3.789629
n=80 I=3.789908
n=160 I=3.789977
```

```
// program cn09;
#include <iostream>
#include <iomanip>
#include <math.h>
    using namespace std;
#define r 4

 double S, a, b, h;
 int j, i, n;

double f(double x)
 {
   return pow(x, 4) / sqrt(1 + x);
 }
```

```
int main()
{
    a = 1; b = 2; n = 10;
    for (j = 1; j <= 4; j++)
    {
        S = 0; n = n * 2;
        h = (b - a) / n;
        for (i = 0; i < n; i++)
            S = S + h * f(a + i * h + h / 2);
        cout<< setprecision(8)<< "n = " << n << " I=" << S << endl;
    }
 return 0;
}
```

Результаты:

```
n = 20 I=3.7885128
n = 40 I=3.7896286
n = 80 I=3.7899076
n = 160 I=3.7899774
```

Пример 2: Требуется написать программу для приближенного вычисления интеграла $\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{\sin\ (\cos x^2)}{2} dx$ методом прямоугольников с точностью, не превосходящей заданное значение ε. Расчеты будут прерваны тогда, когда выполнится условие: $\frac{\pi}{8} h \le \varepsilon;\quad h = \frac{|b-a|}{n}$, где n – число выполненных разбиений.

Переменные *a, b,* ε задаются непосредственно в тексте программы.

Решение: Из условий задачи определяется число необходимых разбиений:

$$\frac{\pi|b-a|}{8n} \le \varepsilon;\quad \Rightarrow\quad n = \left[\frac{\pi|b-a|}{8\varepsilon}\right] + 1.$$

```
program* cn10;
var S, a, b, e, h : real;
 j,i,n:integer;
function f(x:real): real;
begin      f:=sin(cos (x*x))/2;end;
begin      a:=-pi/2; b:=pi/2; e:=0.0001;
       n:=round(pi*(abs(b-a))/(8*e))+1;
       S:=0;
       h:=(b-a)/n;
       for i:=0 to n-1 do
       S:=S+ h*f(a + i*h + h/2);
       writeln ('n=',n,' I=',S:0:6);
end.
```

Результаты:

```
n=12338 I=0.729729
```

* Здесь и в других задачах данного издания используется предопределенная в среде Turbo Pascal функция pi. Пользователи других языков программирования либо других компиляторов Pascal должны явно объявить в программе данную константу. Например: const pi=3.141.

```
// program cn10;
#include <iostream>
#include <iomanip>
#include <math.h>
    using namespace std;
#define pi 3.14159265359

 double S, a, b, e, h;
 int i, n;

double f(double x)
 {
   return sin(cos (x * x)) / 2;
 }

int main()
{
    a = -pi / 2; b = pi / 2; e = 0.0001;
    n = round(pi * (abs(b - a))/(8 * e)) + 1;
    S = 0;
    h = (b - a) / n;
    for (i = 0; i < n; i++)
        S = S + h * f(a + i * h + h / 2);
    cout << setprecision(8) << "n = " << n << " I=" << S;
 return 0;
}
```

Результаты: `n = 12338 I=0.72972935`

Вопросы и задания

1. **Аргументируйте!** Приведите доводы необходимости численного интегрирования при вычислении определенных интегралов.
2. **Объясните!** Какой процесс является основой метода прямоугольников для приближенного вычисления интеграла?
3. **Объясните!** Обоснуйте отношение между числом разбиений отрезка интегрирования и погрешностью вычислений методом прямоугольников.
4. **Проведите исследование!** Напишите программу, вычисляющую определенный интеграл методом прямоугольников, для 200, 2 000, 20 000 разбиений отрезка интегрирования. Для каждого заданного числа разбиений необходимо вывести вычисленное значение интеграла с шестью цифрами после запятой.

a) $\int_{-1}^{1}\left(\frac{1+x^2}{1+x^4}\right)dx$;

b) $\int_{1}^{9}\left(x\sqrt[3]{1-x}\right)dx$;

c) $\int_{0}^{2\pi}\frac{dx}{(2+\cos x)(3+\cos x)}$;

d) $\int_{0}^{2\pi}\frac{dx}{\sin^4 x+\cos^4 x}$;

e) $\int_{0}^{1}\frac{dx}{(x+1)\sqrt{x^2+1}}$;

f) $\int_{0}^{\ln 2}\sqrt{e^x-1}\,dx$.

Объясните расхождения в полученных для разного числа разбиений результатах. Проверьте полученные результаты с помощью онлайн-приложения для вычисления определенных интегралов: www.solvemymath.com/online_math_calculator/calculus/definite_integral.

5 **Интегрируйте!** Напишите программу для приближенного вычисления интеграла методом прямоугольников с точностью, не превосходящей заданное значение ε = 0,001; ε = = 0,0001; ε = 0,00001. Значение *M* известно априори:

a) $\int_0^3 \left(x^2+3x\right)dx$, $M = 9$; c) $\int_1^2 \left(x^3-7x\right)dx$, $M = 5$;

b) $\int_1^2 \left(x^3-2x^2+3x\right)dx$, $M = 7$; d) $\int_2^7 \left(x^2\sqrt{x-1}\right)dx$, $M = 36$.

Для каждого значения погрешности необходимо вывести вычисленное значение интеграла и число разбиений, при котором получено данное приближение.

6 **Интегрируйте!** Границы земельного участка определяются вертикальными прямыми $x = -2$ и $x = 0$, сверху – графиком функции $f(x) = x^3 - 5x + 3\cos x$, снизу – графиком функции $f(x) = x^3 - 5x + 3\cos x$. Напишите программу для вычисления площади участка, методом прямоугольников, для:

a) 10 000 разбиений; b) ε = 0,00001.

3.11 Вариации метода прямоугольников

В некоторых случаях точка, определяющая высоту прямоугольника, аппроксимирующего интеграл на элементарном участке, может не быть серединой отрезка, а одним из его концов. Тогда, если аппроксимация проводится через левые концы (*рис. 3.11*) элементарных отрезков, площадь элементарного прямоугольника $S_i = hf(x_i)$, а определенный интеграл аппроксимируется суммой:

$$I_{st} = \sum_{i=0}^{n-1} hf(x_i) = h\sum_{i=0}^{n-1} f(a+ih),$$

где $h = \frac{|b-a|}{n}$, n – число разбиений промежутка интегрирования, а применяемый метод называется *методом левых прямоугольников*.

Если аппроксимация проводится через правые концы (*рис. 3.13*) элементарных отрезков, площадь $S_i = hf(x_{i+1})$, а определенный интеграл аппроксимируется:

$$I_{dr} = \sum_{i=1}^{n} hf(x_i) = h\sum_{i=1}^{n} f(a+ih),$$

где $h = \frac{|b-a|}{n}$, n – число разбиений промежутка интегрирования, а применяемый метод называется *методом правых прямоугольников*.

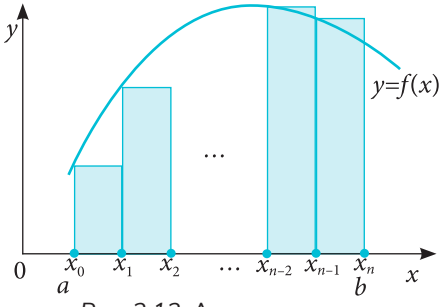


Рис. 3.12. Аппроксимация определенного интеграла левыми прямоугольниками

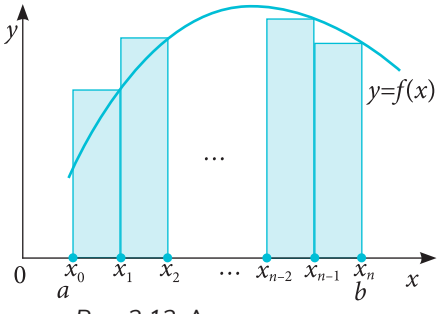


Рис. 3.13. Аппроксимация определенного интеграла правыми прямоугольниками

Погрешность вычислений для вариаций метода прямоугольников

В случае аппроксимации определенного интеграла площадями левых (правых) прямоугольников основная формула оценивания погрешности очень мало изменяется:

$$\left|\int_a^b f(x)dx - I\right| \le (b-a)\,M\,\frac{h}{2},$$

где M – супремум $|f'(x)|$ на $[a, b]$, I – вычисленное значение интеграла путем приближения левыми (правыми) прямоугольниками.

Так же, как и в случае серединных прямоугольников, число разбиений, необходимое для получения решения с точностью, не превосходящей заданное значение ε, можно получить из формулы погрешности:

$$n = \left[M\,\frac{(b-a)^2}{2\varepsilon} \right] + 1.$$

Алгоритмизация метода
(левые прямоугольники)*

Шаг 1. *Инициализация:* вводятся значения концов промежутка интегрирования a, b и число требуемых разбиений n.

Примечание. В случае вычислений интеграла с точностью, не превосходящей заданную погрешность ε, число разбиений n вычисляется с помощью формулы $n = \left[\frac{(b-a)^2 M}{2\varepsilon} \right] + 1$.

Шаг 2. Вычисляется длина элементарного отрезка $h = \frac{|b-a|}{n}$. $S \Leftarrow 0$.

Шаг 3. Для всех i от 0 до n–1:

a) вычисляются значения $x_i \Leftarrow a + ih$;
b) вычисляется площадь элементарного прямоугольника: $S_i \Leftarrow f(x_i) \times h$;
c) вычисленная площадь добавляется к суммарной площади предыдущих прямоугольников: $S \Leftarrow S + S_i$.

Шаг 4. Выводится вычисленная общая площадь S. КОНЕЦ.

Пример 3: Вычислить определенный интеграл $\int_0^{\pi} (x\sin(x))^2 dx$ методом прямоугольников (вариация левых прямоугольников) для 10, 100, 1 000 разбиений. Присваивание начальных значений осуществляется в тексте программы. Для каждого случая числа разбиений на экран выводятся вычисленное значение интеграла и число разбиений, разделенных пробелом.

```
program cn11;
const r=3;
var S, a, b, h : real;
 j,i,n:integer;
function f(x:real): real;
begin      f:=sqr((x* sin(x)));end;
begin      a:=0; b:=pi; n:=1;
       for j:=1 to r do
       begin S:=0; n:=n*10;
       h:=(b-a)/n;       for i:=0 to n-1 do
```

* Случай правых прямоугольников отличается от приведенного лишь на *3-ем шаге*. В этом случае индекс i изменяется в диапазоне: от 1 до n.

```
        S:=S+ h*f(a + i*h);
        writeln ('n=',n,' I=',S:0:6);
        end;
end.
```

Результаты:
```
n=10 I=4.381796
n=100 I=4.382315
n=1000 I=4.382315
```

```
// program cn11;
#include <iostream>
#include <iomanip>
#include <math.h>
    using namespace std;

#define r 3
#define pi 3.14159265359

 double S, a, b, h;
 int j, i, n;

double f(double x)
 {
   return pow(x * sin(x), 2);
 }

int main()
{
    a = 0; b = pi; n = 1;
    for (j = 1; j <= r; j++)
    {
        S = 0; n = n * 10;
        h = (b - a) / n;
        for (i = 0; i < n; i++)
            S = S + h * f(a + i * h);
        cout<< setprecision(8)<< "n = " << n << " I=" << S << endl;
    }
 return 0;
}
```

Результаты:
```
n = 10 I=4.3827687
n = 100 I=4.3823147
n = 1000 I=4.3823146
```

Пример 4: Требуется вычислить определенный интеграл $\int_0^1 \sin x \sin 2x \sin 3x dx$ методом прямоугольников (вариация правых прямоугольников) с точностью, не превышающей заданную погрешность ε = 0,001. Присваивание начальных значений осуществляется в тексте программы. На экран выводятся вычисленное значение интеграла и необходимое число разбиений, разделенных пробелом.

Решение: Необходимое число разбиений можно определить аналитически. Используя элементарные математические выкладки, определим значение $M = 6$:

$$n = \left[M \frac{(b-a)^2}{2\varepsilon} \right] + 1.$$

```
program cn12;
var S, a, b, e, h, M : real;
 j,i,n:integer;
function f(x:real): real;
begin    f:=sin(x)* sin(2*x)* sin(3*x);end;
begin    a:=0; b:=1; e:=0.001; M:=6;
     n:=trunc(M*(b-a)*(b-a)/(2*e))+ 1;
     S:=0; h:=(b-a)/n;
     for i:=1 to n do
     S:=S+ h*f(a + i*h);
     writeln ('n=',n,' I=',S:0:6);
end.
```

Результаты: n=3001 I=0.278729

```
// program cn12;
#include <iostream>
#include <math.h>
    using namespace std;

double S, a, b, e, h, M;
int j, i, n;

double f(double x)
 {
   return sin(x) * sin(2 * x) * sin (3 * x);
 }
int main()
{
    a = 0; b = 1; e = 0.001; M = 6;
    n = trunc(M * pow((b-a),2) / (2 * e)) + 1;
    S = 0;
    h = (b - a) / n;
```

```
    for (i = 1; i <= n; i++)
            S = S + h * f(a + i * h);
    cout<< "n = " << n << " I=" << S << endl;
 return 0;
}
```

Результаты: `n = 3001 I=0.278729`

Вопросы и задания

1 **Объясните!** В чем суть отличий между методом серединных прямоугольников и его вариациями?

2 **Объясните!** Каково отношение между числом разбиений промежутка интегрирования и погрешностью вычислений в вариациях метода прямоугольника?

3 **Проанализируйте!** Какая из вариаций метода прямоугольника является более точной? Почему?

4 **Проведите исследование!** Напишите программу, вычисляющую определенный интеграл методами серединных, левых и правых прямоугольников. Примените различное число разбиений (например, 10, 100, 1 000).

a) $\int_1^3 (x\ln x)^2 dx$; b) $\int_1^6 \left(x\sqrt{x-1}\right) dx$; c) $\int_0^1 \left(x^2\sqrt{1+3x^2}\right) dx$;

d) $\int_0^{\pi} \left(e^x \cos^2 x\right) dx$; e) $\int_0^{\pi} (x\sin x)^2 dx$; f) $\int_0^{2\pi} \frac{dx}{(2+\cos x)(3+\sin x)}$.

Объясните расхождения в полученных для разного числа разбиений результатах. Проверьте полученные результаты с помощью доступного онлайн-приложения для вычисления определенных интегралов: www.solvemymath.com/online_math_calculator/calculus/definite_ integral.

5 **Программируйте!** Напишите программу для приближенного вычисления интеграла методом прямоугольников с точностью, не превосходящей заданное значение ε = 0,005; ε = 0,0005; ε = 0,00005. Значение *M* определите аналитически:

a) $\int_1^3 \left(3x^2 + x + 2\right) dx$; b) $\int_1^2 \left(2x^3 - x^2 + 5x\right) dx$; c) $\int_{-3}^{-1} \left(2x^3 - 3x\right) dx$.

Для каждого значения погрешности необходимо вывести вычисленное значение интеграла и число разбиений, при котором получено данное приближение.

6 **Интегрируйте!** Напишите программу, определяющую, какой из следующих определенных интегралов является больше. Примените вариации метода прямоугольников:

a) $\int_0^{\pi} \left(e^{-x^2} \cos^2 x\right) dx$ или $\int_{\pi}^{2\pi} \left(e^{-x^2} \cos^2 x\right) dx$; b) $\int_0^1 \left(e^{-x}\right) dx$ или $\int_0^1 \left(e^{-x^2}\right) dx$.

IV
Раздел

Базы данных

4.1 Данные и базы данных

Элементарные данные и структуры данных

Данное – это модель представления информации, доступной для обработки некоторым процессором (человеком, программой и др.), модель, оперируя с которой можно получить новую информацию.

Элементарное данное – это сущность, неделимая по отношению к представляемой ею информации и по отношению к логическому (программе) либо физическому (ЦПУ) процессору.

С логической точки зрения, элементарное данное характеризуется:

а) *идентификатором; b) значением; c) атрибутами.*

Идентификатор некоторого данного используется для его вызова (доступа к нему).

Значение данного – это содержимое зоны памяти, в которой это данное размещено.

Область определения данного – это множество значений, которые может принимать в процессе обработки это данное.

Атрибуты данного определяют его специфические характеристики. Одним из наиболее важных атрибутов данного является его ***тип***, определяющий принадлежность к некоторому классу данных. Каждому типу данных соответствует некоторая модель внутреннего представления.

Пример 1: Объявление `var c: integer`, в языке Pascal определяет переменную, которая в любой момент работы программы идентифицирует элементарное данное *целого* типа, «занимающее» в памяти компьютера зону величиной в два байта. Областью определения типа *integer* является множество чисел {–32 768, –32 767, ..., 32 767}. В результате выполнения команды `c := 8`, в зоне, выделенной для хранения переменной, запомнится значение 8.

Пример 2: Значения простых типов языка Pascal являются элементарными данными.

Структура данных – это набор данных, между элементами которых установлены некоторые отношения, определяющие методы их идентификации и обработки. Структура данных может состоять из элементарных данных, а также, из других структур. Каждый компонент структуры данных локализуется с помощью идентификатора либо положения внутри структуры.

Пример 3: Значения типа *record* из языка Pascal являются структурами данных, компоненты которых (поля) могут вызываться по имени, то есть по идентификатору поля. Массив – это структура данных, компоненты которой (элементы массива) могут вызываться с помощью указания их положения (то есть индекса) в массиве.

Основными характеристиками некоторой структуры данных являются:

a) *Однородность.* Однородная структура состоит из компонентов одинакового типа. Так, записи (значения типа *record*) – это неоднородные структуры, а массивы – однородные;

b) *Способ доступа к компонентам*. К компонентам некоторой структуры может быть прямой либо последовательный доступ. Например, текстовые файлы являются структурами данных с последовательным доступом;

c) *Стабильность структуры.* Если в процессе обработки с помощью программы в некоторой структуре число элементов и связи между ними остаются неизменными, то такая структура является статической. В противном случае, структура называется динамической. Например, односвязные списки – это динамические структуры, а записи – статические;

d) *Время существования*. Структуры данных могут быть постоянными либо временными. Например, внешние файлы являются постоянными структурами, а массивы – временными.

Базы данных

До настоящего момента, в школьном курсе информатики, нам приходилось обрабатывать различные типы данных и структур, пользуясь средствами языка программирования, либо программными приложениями (текстовый редактор, табличный процессор и др.). Наиболее сложной обрабатываемой структурой был файл.

Известно, что данные о некоторой фирме, компании или предприятии можно сохранить в файле. Например, если речь идет о некотором лицее, то можно создать файлы *Учителя, Ученики, Классы, Предметы* и др., в которых сохранять соответственно данные об учителях, учениках, классах, учебных предметах и др.

Для хранения информации о распределении учителей по классам, понадобится еще один файл, *Учитель_Класс*, созданный на основании содержимого файлов *Учителя* и *Классы*. Предположим, что только вступивший в долность завуч создает такой файл. Если данные об учителях в файле *Учителя*, содержат ошибки, то эти ошибки повторятся и в файле *Учитель_Класс*. Очевидно, эти же ошибки появятся и в файле *Расписание_уроков*. Обнаружив ошибки, завуч будет вынужден исправлять данные во всех трех файлах. Было бы значительно проще, если при исправлении ошибок в файле *Учителя* автоматически актуализировались бы все файлы, «использовавшие» эту информацию. Это бы гарантировало ***целостность данных***.

Помимо *сложностей с актуализацией*, метод обработки данных с помощью файлов, имеет и другие недостатки:

- *Наличие одинаковых данных в нескольких независимых файлах* может привести к их дублированию, а следовательно к большому расходу памяти.
- *Сложность получения спонтанной информации*, хранящейся в нескольких независимых файлах (например, для получения списка учителей женского пола преподающих реальные предметы в 10-ом классе реального профиля, необходимо обработать, как минимум, файлы *Учителя, Классы, Предметы*).
- *Несовместимые форматы файлов* замедляют обработку информации (например, файлы, сгенерированные с помощью различных языков программирования, могут иметь различные форматы, следовательно, понадобится программное приложение, приводящее файлы к одинаковому формату).

Эти и другие проблемы можно решить путем использования структуры типа *базы данных*.

База данных – это совокупность информаций (данных), специальным образом организованных так, чтобы облегчить их хранение, обработку и извлечение.

Информация в базах данных хранится в виде записей или файлов, между которыми существуют логические связи. Кроме собственно хранения данных, база данных содержит и их описание: тип, структуру, способ организации и взаимосвязи.

Вопросы и задания

1. **Систематизируйте свои знания!** Назовите известные вам типы данных. Какие из них являются:
   a) простыми типами данных;
   b) структурными типами данных?
2. **Практикуйтесь!** Выберите элементарные данные: символ, действительное число, строка символов, массив чисел, элемент некоторого множества, целое число, значение false, файл.
3. **Объясните!** Какова область определения типа данных:
   a) `integer`;
   b) `char`;
   c) `lit_M` объявленный следующим образом `type lit_M = 'A'.. 'Z';`
   d) `boolean`?
4. **Проанализируйте!** Охарактеризуйте структуры данных (однородность, способ доступа, стабильность и время существования):
   a) множество;
   b) массив;
   c) запись;
   d) односвязный список;
   e) файл.
5. Объясните смысл понятия *база данных*.
6. **Проанализируйте!** Почему хранение и обработка информации о некоторой компании более приемлемы в виде базы данных, нежели в виде системы независимых файлов.
7. **Объясните!** Каковы отличия между базой данных и структурой данных?

4.2 Типы баз данных

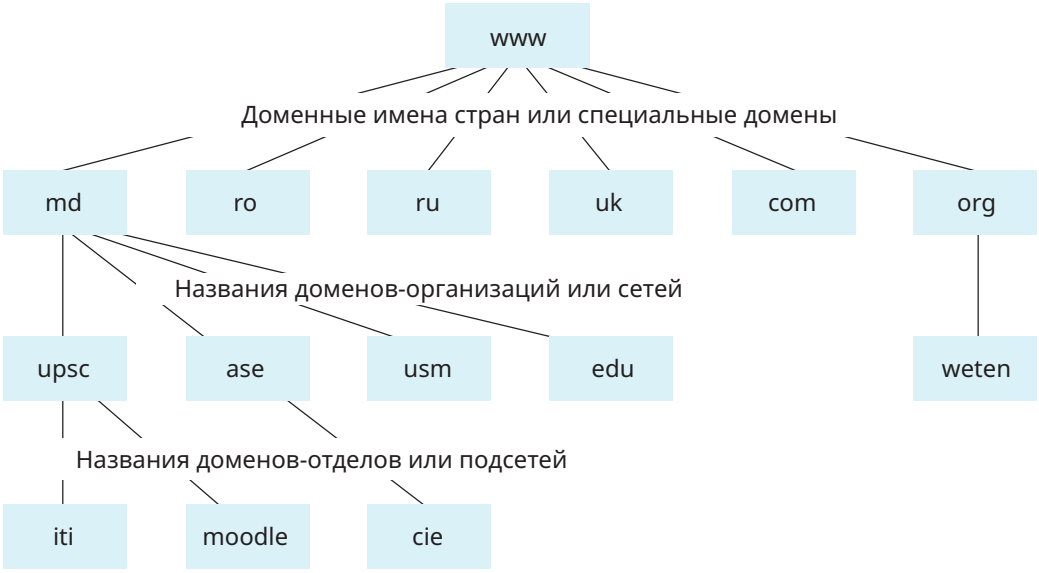
Из следующего параграфа мы узнаем, что одним из этапов разработки баз данных является создание ее *логической модели*, определяющей способ организации данных и не зависящей от физических параметров среды их хранения.

> **Логическая модель представляет собой общее описание базы данных с помощью естественного либо математического языка, диаграмм, органиграмм, графиков и других средств, понятных тем, кто будет ее разрабатывать.**

Наиболее распространенными логическими моделями являются:
a) *иерархическая* (или *древовидная*); b) *сетевая*; c) *реляционная*.
База данных называется иерархической, сетевой или реляционной в зависимости от ее логической модели.

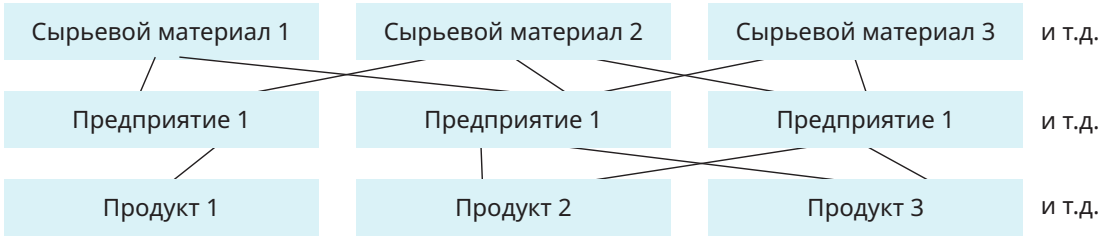
Сущности (коллекции данных) ***иерархической базы данных*** (*hierarhical database*) организованы в виде узлов, соединенных ветвями дерева. Каждый узел подчинен, максимум, одному узлу уровня, расположенного непосредственно над ним (называемого *материнским узлом*), и может быть связан с одним или несколькими узлами на иерархически нижнем уровне (называемыми *дочерними узлами*).

Пример: База данных с интернет-адресами имеет следующую иерархическую модель:



В случае ***сетевой базы данных*** (*network database*), сущности также организованы иерархически, однако одному дочернему узлу может соответствовать несколько материнских узлов.

Пример: База данных некоторой индустриальной отрасли может иметь следующую модель:



Реляционная база данных (*relational database*) представляет собой самую гибкую модель организации данных. Обход сущностей такой базы данных не является иерархическим.

Реляционная модель проста, но строго определена с математической точки зрения. Впервые такая модель была предложена в 1970 году ученым Эдгаром Франком Коддом*.

* *Эдгар Франк Кодд* (23.08.1923, остров Портленд, Англия – 18.04.2003, Уильямс Айленд, Флорида, США) – американский информатик британского происхождения. Работая для IBM, изобрел реляционную модель для управления базами данных. За вклад в развитие информатики получил в 1981 году премию Тьюринга, считающуюся *Нобелевской премией в области информатики*.

Реляционная база данных состоит из таблиц, называемых *отношениями*. Каждая таблица состоит из строк и столбцов. Строки называются *записями*, а столбцы – *полями* данных. *Заголовок таблицы* определяет ее структуру. Под *созданием таблицы*, вообще говоря, подразумевается описание ее заголовка.

На *рисунке 4.1* представлена часть схемы некоторой реляционной базы данных.

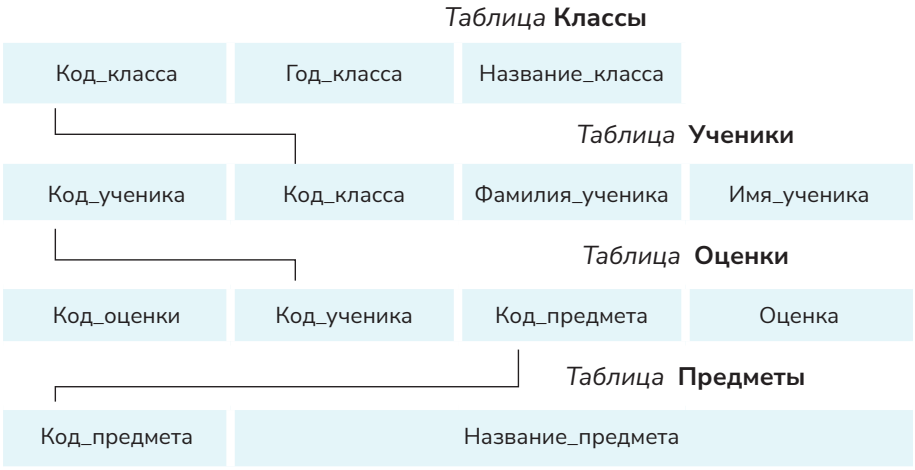


Рис. 4.1. Часть схемы реляционной базы данных

Между таблицами реляционной базы данных существуют взаимосвязи (подробнее об этих связях вы узнаете из следующих параграфов).

Замечание: Далее будем рассматривать лишь реляционные базы данных.

Вопросы и задания

1. **Систематизируйте свои знания!** Опишите структуры баз данных:
   a) иерархических;
   b) сетевых;
   c) реляционных.
2. **Примените!** Определите тип базы данных, зная, что:
   a) между любыми двумя сущностями базы существует связь;
   b) сущности базы являются узлами двоичного дерева;
   c) одна из сущностей связана со всеми остальными, и других связей не существует;
   d) вся информация базы содержится в десяти различных таблицах.
3. **Разработайте!** Представьте схематически иерархическую модель базы данных, содержащей информацию:
   a) о лицее;
   b) о продовольственном магазине;
   c) о библиотеке.
4. **Учитесь учиться!** Выполните задание 3:
   a) для сетевой модели;
   b) для реляционной модели.
5. **Примените!** Определите тип базы данных с номерами телефонов стационарной телефонной связи страны.

4.3 Создание реляционных баз данных

Основные аспекты

Система базы данных представляет собой систему организации и обработки базы данных. Эта система сформирована, собственно, из базы данных и из ансамбля программ, обеспечивающих управление и комплексную обработку данных.

Под **проектированием** или **созданием базы данных** подразумевается процесс создания системы этой базы.

Проектирование состоит, как правило, из следующих **этапов**:

1. **Этап анализа.** Анализируется область данных, формулируются требования к системе базы данных со стороны всех категорий ее пользователей, определяются операции, которыми будет управлять эта система. В результате данного анализа создается логическая модель базы данных, а также определяется структура основного и вспомогательных интерфейсов системы базы данных.
2. **Этап программирования.** Создаются логические компоненты (программы): основной интерфейс, приложения для ввода/обновления/обработки данных, приложения для запросов и для вывода информации.
3. **Запуск и эксплуатация системы базы данных.** База заполняется информацией (записями). Затем осуществляются операции по обновлению, консультации и последующей эксплуатации (а также развитию) системы базы данных.

Создание и поддержка баз данных обычно осуществляется целой группой лиц, в состав которой входят: *администраторы* (определяют начальную структуру базы данных, способ хранения информации на физическом уровне; предоставляют пользователям права доступа к базе данных, обеспечивают безопасность данных; изменяют структуру и осуществляют дальнейшее развитие базы данных), *программисты* (пишут программы на языках программирования), *дизайнеры* (осуществляют дизайн интерфейса базы данных) и др.

Замечание: Существуют современные компьютеризированные системы, называемые Системами Управления Базами Данных (СУБД), специально разработанные для создания баз данных. Подробнее о таких системах вы узнаете из следующего параграфа.

Проектирование сущностей реляционной базы данных

В предыдущем параграфе было отмечено, что сущностями реляционной базы данных являются **таблицы**, в которых хранятся *записи* (строки таблиц).

Заголовок таблицы определяет ее структуру. Под *созданием таблицы* подразумевается определение ее заголовка, то есть описание ее полей (столбцов).

Пример: Рассмотрим базу данных *Liceu*, в которой накоплена информация о некотором лицее. Таблица *Elevi* этой базы данных хранит информацию об учениках этого лицея:

Таблица **Elevi**

Cod_elev	Nume_elev	Pren_elev	Cod_clasa	Data_elev	Foto_elev	Telefon	Gen_elev
e001	Bacinschi	Sabina	c01	28.09.2007	Package	29-82-54	F
e002	Belobrov	Andreea	c01	18.10.2007	Package	44-26-48	F
e006	Chilimciuc	Dumitru	c01	09.08.2007	Package	26-13-16	M

При описании каждого поля таблицы необходимо указать его *имя* и *тип*.

a) *Имя поля* должно быть уникальным в данной таблице. Данные, сохраняемые в поле, однородны;

b) Тип поля определяет тип его значений (числовой, строка символов, большой текст, календарное число, логический, изображение и др.).

Так, имя первого поля таблицы *Elevi* – *Cod_elev*, тип – *Строка символов*, а у пятого поля – имя *Data_elev*, тип – *Календарное число*.

Каждое значение из поля *Cod_elev* таблицы *Elevi* соответствует единственной записи.

Поле, значения которого идентифицируют каждую запись, называется ***первичным ключом***. Очевидно, поле может быть первичным ключом, только если его значение не пусто и ни разу не повторяется.

Рекомендуется, чтобы в каждой таблице базы данных было поле-первичный ключ. Иногда роль первичного ключа могут играть сразу несколько полей (если вместе они однозначно идентифицируют строки таблицы).

Поле таблицы называется ***внешним ключом***, если оно определено как первичный ключ в другой таблице. Так, в таблице *Elevi*, поле *Cod_elev* является первичным ключом, а *Cod_clasa* – вторичным.

Обычно каждая таблица реляционной базы данных находится в некотором отношении по крайней мере с еще одной таблицей этой базы. Существует **три типа отношений между таблицами**:

a) *один-к-одному*; b) *один-ко-многим*; c) *многие-ко-многим*.

В отношении типа **один-к-одному** (обозначается 1:1), каждой записи одной таблицы соответствует одна запись из другой таблицы, и наоборот. Этот тип отношений возникает, когда таблица с большим количеством полей разбивается на две таблицы. В этом случае в первичных ключах обеих таблиц значения совпадают.

Например, между таблицами *Elevi* и *Adrese_elevi* задано отношение типа *один-к-одному*. Обратите внимание на существование одинакового поля *Cod_elev*, первичного ключа в обеих таблицах, с идентичными данными.

Таблица **Adrese_Elevi**

Cod_elev	Loc_elev	Strada_elev	Nr_casa_elev	Ap_elev
e001	Chişinău	Mihail Sadoveanu	40	23
e002	Stăuceni	Viei	9	

Схематически этот тип отношения представлен на *рисунке 4.2*.

В случае отношения ***один-ко-многим*** (обозначается 1:∞ либо 1:M) каждой записи из одной таблицы могут соответствовать несколько записей из другой таблицы, а каждой записи из второй таблицы соответствует максимум одна запись из первой. Для того, чтобы установить отношение такого типа, первая таблица должна иметь поле-первичный ключ, а вторая таблица – внешний ключ типа, совместимого с типом первичного ключа первой таблицы.

Например, таблица *Clase* находится в отношении *один-ко-многим* с таблицей *Elevi*, потому что в одном и том же классе есть несколько учеников, а каждый ученик «принадлежит» лишь одному классу.

Таблица **Clase**

Cod_clasa	Anul_de_studii	Nume_clasa	Cod_profil
c01	10	A	p1
c02	10	B	p1
c03	10	C	p2
c04	10	D	p2

Первичный ключ *Cod_clasa* таблицы *Clase* «совместим» со внешним ключом *Cod_clasa* таблицы *Elevi*. Схематически этот тип отношения представлен на *рисунке 4.3*.

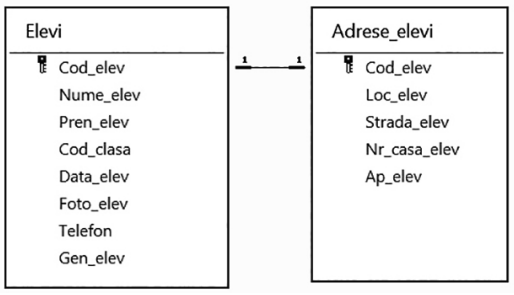


Рис. 4.2. Отношение 1:1

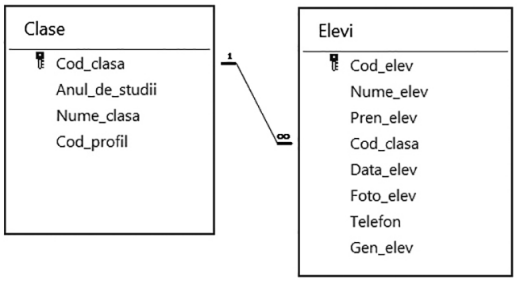


Рис. 4.3. Отношение 1:M

В отношении ***многие-ко-многим*** (обозначается ∞:∞ или M:M) каждой записи из одной таблицы могут соответствовать несколько записей из другой таблицы, и наоборот. Например, таблицы *Clase* и *Discipline* находятся в отношении *многие-ко-многим*, так как в каждом классе преподаются несколько предметов и каждый предмет преподается в нескольких классах.

Таблица **Discipline**

Cod_disciplina	Nume_disciplina
d01	Matematica
d02	Informatica
d03	Chimia
d04	Fizica

Таблица **Prof_dis_clasa**

Cod_repart	Cod_clasa	Cod_dis	Cod_prof
r001	c01	d01	prof01
r002	c01	d02	prof01
r003	c01	d03	prof02
r004	c01	d05	prof04

Отношение *многие-ко-многим* между двумя таблицами может быть установлено с помощью двух отношений типа *один-ко-многим*. Каждая из двух таблиц связывается отношением *один-ко-многим* с третьей таблицей.

Например, отношение *многие-ко-многим* между таблицами *Clase* и *Discipline* реализуется с помощью таблицы *Prof_dis_clasa* (*рис. 4.4*).

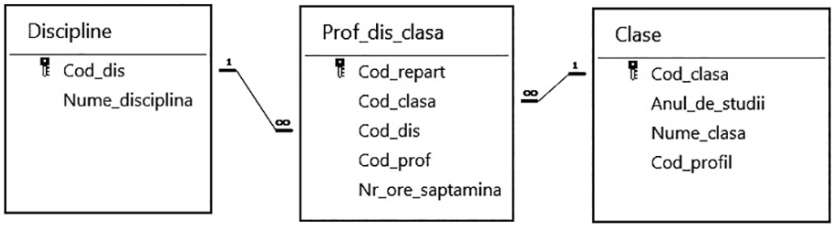


Рис. 4.4. Установление отношения M:M через отношения 1:M и M:1

Замечание: Первая запись из таблицы *Prof_dis_clasa* содержит информацию о том, что в классе c01 (то есть, в 10-ом А классе) предмет d01 (то есть *Matematica*) преподается учителем prof01 (данные об этом преподавателе хранятся в таблице *Profesori*).

Принципы проектирования

Проектируя реляционную базу данных, программисты определяют таблицы таким образом, чтобы соблюдались некоторые принципы (например, чтобы они были приведены к так называемым *нормальным формам*).

- Один из этих принципов требует, чтобы данные в полях таблицы были **элементарными данными**. Из этих соображений адреса учеников и учителей хранятся в нескольких полях. Очевидно, обработка записей была бы затруднена, если бы весь адрес был записан в одном поле.
- Для того, чтобы обновление записей было оптимальным (например, более быстрым), необходимо исключить из таблиц повторяющиеся поля (кроме ключевых полей, с помощью которых реализованы отношения между таблицами). Добиваются этого путем **разложения** соответствующей таблицы на несколько таблиц.

Например, данные об учителях, преподающих определенные предметы в определенных классах, накапливаются в четырех таблицах: *Profesori*, *Discipline*, *Clase* и *Prof_dis_clase*. Благодаря тому что между данными таблицами существуют связи, для замены во всех записях фамилии учителя достаточно изменить его в таблице *Profesori*.

Вопросы и задания

1. **Систематизируйте свои знания!** Назовите этапы создания базы данных.
2. **Объясните!** Из каких сущностей состоит реляционная база данных?
3. **Объясните!** Объясните принципы проектирования реляционной базы данных.
4. **Проанализируйте!** Чем отличается отношение M:M от отношения 1:M?
5. **Практикуйтесь!** Изучите таблицу *Elevi* базы данных *Liceu* и определите тип значений каждого поля.
6. **Примените!** Выберите первичный ключ для таблицы:
   a) *Класс_12* с полями *Порядковый_номер*, *Фамилия_ученика*, *Имя_ученика*, *Телефон_ученика*;
   b) *Сотрудники* с полями *Фамилия_сотрудника*, *Имя_сотрудника*, *Возраст_сотрудника*, *Пол_сотрудника*, *Номер_удостоверения_личности_сотрудника*;
   c) *Стоянка_авто* с полями *Марка_ авто*, *Регистрационный_номер_авто*, *Фото_авто*, *Фамилия_владельца_авто*;
   d) *Страны* с полями *Название_страны*, *Площадь_поверхности_страны*, *Столица_страны*, *Количество_жителей_страны*.
7. **Проанализируйте!** Какой тип отношений можно установить между таблицами:
   a) *Вина* и *Производители вин*;
   b) *Книги* и *Авторы*;
   c) *Стихи* и *Авторы*;
   d) *Специальности* и *Университеты*?
8. **Примените!** Спроектируйте базу данных, содержащую информацию:
   a) из телефонного справочника;
   b) о книгах персональной библиотеки;
   c) об автомобилях;
   d) о великих личностях нашей страны.

4.4 Системы управления базами данных

Основные понятия о системах управления базами данных

Система управления базами данных (СУБД) – это набор программ, позволяющих создание баз данных, ввод и обновление информации, обеспечение контроля доступа к сохраняемым в базе данным, управление данными, а также разработку приложений, использующих информацию из баз данных. СУБД выполняет следующие функции:

a) *определяет базу данных*, в том смысле, что описывает типы и структуры данных, отношения между ними и способы доступа к информации базы данных;
b) *обновляет базу данных*, то есть позволяет добавлять, редактировать и удалять записи;
c) *исполняет запросы к базе данных*, в результате которых данные упорядочиваются либо фильтруются, согласно определенным требованиям, сформулированным пользователями;
d) *создает новые данные*, получаемые путем расчетов, а также путем подведения итогов;
e) *создает отчеты*, простые и сложные, в виде таблиц, графиков, диаграмм и др.;
f) *обеспечивает обслуживание базы данных*, предполагающее исправление базы в случае появления неисправностей, сжатие (или дефрагментацию), создание резервных копий как самих данных, так и всех объектов из базы данных;
g) *обеспечивает безопасность базы данных*, защиту от неавторизованного доступа, а также устанавливает права полного либо частичного доступа.

- Программное приложение СУБД состоит из:

a) *Словаря данных* (*Data Dictionary*), содержащего описание структуры данных, используемых в базе данных;
b) *Языка запросов* (*Query Language*), обеспечивающего доступ к информации базы данных. Наиболее распространенным языком запросов (используемым в СУБД) является язык SQL (*Structured Query Language*).

- Аппаратные средства СУБД могут состоять:

a) *из одного компьютера* или
b) *из сети компьютеров*, в которой на главном компьютере (сервере) находятся программные компоненты СУБД, администрирующие базу данных и проверяющие права доступа к данным, а на остальных компьютерах – программы, предназначенные для пользователей.

Наиболее распространенными СУБД являются Microsoft Office Access, Oracle, MySQL, PostgreSQL, SQLite, Microsoft SQL Server и др.

Система управления базами данных Microsoft Office Access

Microsoft Office Access (далее просто Access) – это система управления реляционными базами данных, работающая в среде Windows. Его первая версия появилась в 1993-ем году. С его помощью можно создавать сложные базы данных, так как для этого имеются все необходимые инструменты. По сравнению с другими СУБД, Access прост и удобен в применении. Он предоставляет программистам и простым пользователям целый ряд вспомогательных программ, автоматизирующих разные этапы работы при создании запросов, формуляров, отчетов и других продуктов баз данных.

Запустить на выполнение систему Access можно несколькими способами:

– выполнить двойной щелчок, поместив указатель мыши на ссылку Access на рабочем столе Windows;

– выбрать опцию *Microsoft Office Access* из стартового меню;

– выполнить двойной щелчок, поместив указатель мыши на любую пиктограмму файла с расширением .mdb.

Запустив на выполнение Access, можем **создать базу данных**. Для этого выберем *File → New → Blanc database...*

Появится окно *Blank database.* В текстовом поле *File name* пишем имя создаваемой базы данных. Кнопка, которая сопровождает поле справа, позволяет выбрать каталог, в котором мы будем хранить эту базу данных. Нажимаем кнопку *Create*.

В окне приложения Access появляется **окно базы данных** с ее именем.

Интерфейсы различных версий Access, несмотря на отличия, содержат следующие основные элементы:

a) заголовок окна Access;
b) панель меню (*File, Home, Create, External Data, Database Tools, Help*) окна Access;
c) панель инструментов Access;
d) окно базы данных с закладками для создания и управления объектами классов Access (*Tables* – таблицы, *Queries* – запросы, *Forms* – формуляры, *Reports* – отчеты, *Pages* – страницы, *Macros* – макрокоманды, *Modules* – модули, содержащие операторы языка *Visual Basic for Applications*). Объекты базы данных становятся доступными с помощью соответствующих пиктограмм, расположенных на панели *Objects* окна базы данных.

Далее объясним, как создается и управляется в среде Access некоторая база данных. С этой целью создадим и обработаем базу данных *Liceu*, хранящую информацию об учениках и учителях некоторого лицея.

Структура базы данных *Liceu*

База данных *Liceu* состоит из 8 связанных между собой таблиц (*рис. 4.5*).

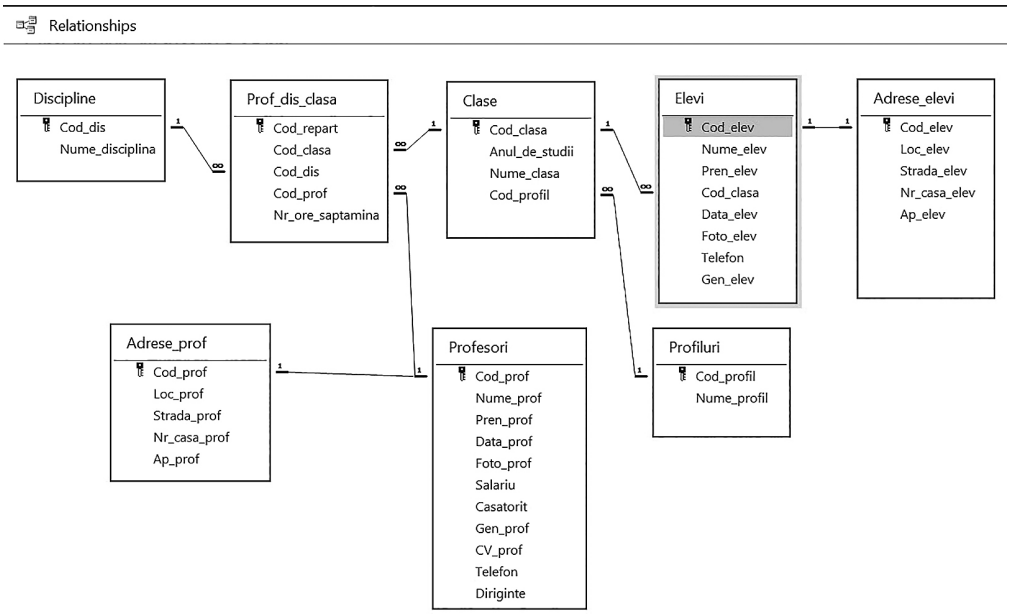


Рис. 4.5. Таблицы базы данных *Liceu*

Представим структуру этих 8 таблиц. Обратите внимание на то, что здесь мы видим лишь несколько записей каждой из *таблиц 1–8*.

Таблица 1. **Profiluri**

Cod_profil	Nume_profil
p1	Real
p2	Umanist

Таблица 2. **Discipline**

Cod_dis	Nume_disciplina
d01	Matematica
d02	Informatica

Таблица 3. **Profesori**

Cod_ prof	Nume_ prof	Pren_ prof	Data_prof	Foto_ prof	Sala-riu	CV_prof	Casa-torit	Gen_ prof	Telefon	Diriginte
prof01	Albu	Ion	01.03.1968	Pack-age	12880	Grad...	Yes	M	64-41-72	c02
prof03	Dodon	Ana	09.07.1958	Pack-age	8040	S-a năs...	No	F	51-65-70	c01

Таблица 4. **Adrese_prof**

Cod_prof	Loc_prof	Strada_prof	Nr_casa_prof	Ap_prof
prof01	Cricova	Vinului	89	
prof02	Chişinău	Grenoble	81	77

Таблица 5. **Clase**

Cod_clasa	Anul_de_studii	Nume_clasa	Cod_profil
c01	10	A	p1
c02	10	B	p1

Таблица 6. **Elevi**

Cod_	Elev	Pren_ elev	Cod_clasa	Data_elev	Foto_ elev	Telefon	Gen_ elev
e001	Bacinschi	Sabina	c01	28.09.2007	Package	29-82-54	F
e002	Belobrov	Andreea	c01	18.10.2007	Package	44-26-48	F

Таблица 7. **Adrese_elevi**

Cod_elev	Loc_elev	Strada_elev	Nr_casa_elev	Ap_elev
e001	Chişinău	Mihail Sadoveanu	40	23
e002	Stăuceni	Viei	9	

Таблица 8. **Prof_dis_clasa**

Cod_repart	Cod_clasa	Cod_dis	Cod_prof
r001	c01	d01	prof01
r002	c01	d02	prof01

Замечание: Таблицы базы данных можно скачать по адресу www.ctice.gov.md

Вопросы и задания

1. **Систематизируйте свои знания!** Опишите структуру системы управления базами данных.
2. **Объясните!** Опишите функции систем управления базами данных.
3. **Объясните!** Как можно запустить на выполнение СУБД Access?
4. **Примените!** Проанализируйте интерфейс СУБД Access и дайте характеристику классов объектов Access.
5. **Примените!** Исходя из содержимого таблиц базы данных *Liceu* установите:
   a) профиль класса, в котором учится Sabina Bacinschi;
   b) фамилию учителя, преподающего информатику в 10-ом А классе;
   c) название предмета, преподаваемого учителем Ion Albu;
   d) адрес классного руководителя 10-го А класса.

4.5 Создание таблиц

Как уже было отмечено, сущностями базы данных Access являются таблицы. С их помощью создаются другие объекты базы данных: запросы, формуляры, отчеты и др. Другими словами, реляционная база не может существовать без таблиц.

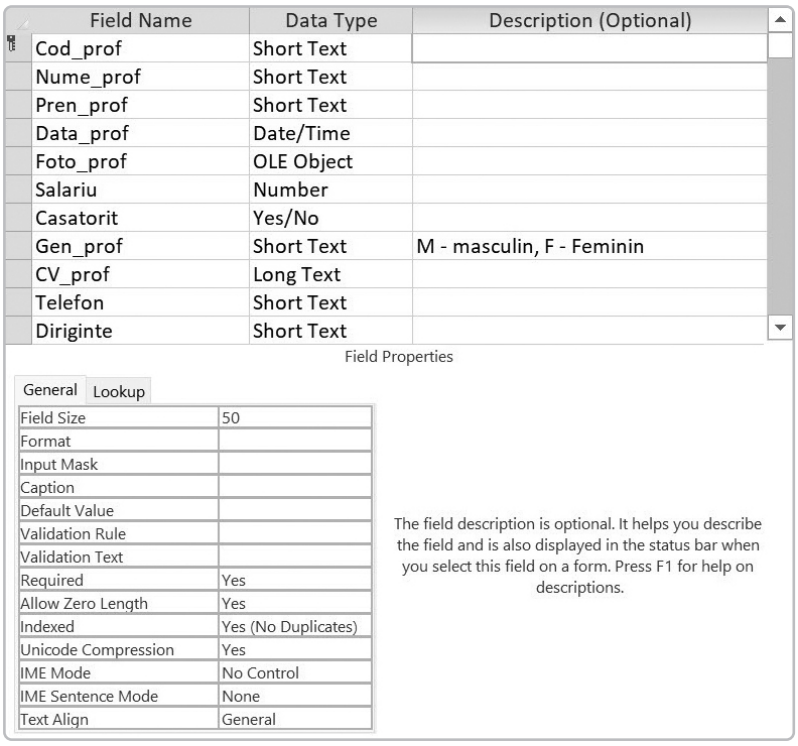
Проектирование структуры таблицы

Под **структурой таблицы** подразумевается информация, описывающая поля таблицы: их число, тип и свойства каждого поля, поля-первичные ключи и др.

Имя новой таблицы по умолчанию – *Table 1*. Ее можно переименовать с помощью команды *Rename* из контекстного меню значка таблицы.

Инструмент *View* в меню *File* позволяет создавать, редактировать или просматривать таблицу в двух режимах: представление *Design view* (режим проектирования) и представление *Datasheet View* (по умолчанию, режим электронной таблицы).

1. Выберите режим просмотра *Design view*.



Field Name	Data Type	Description (Optional)
Cod_prof	Short Text	
Nume_prof	Short Text	
Pren_prof	Short Text	
Data_prof	Date/Time	
Foto_prof	OLE Object	
Salariu	Number	
Casatorit	Yes/No	
Gen_prof	Short Text	M - masculin, F - Feminin
CV_prof	Long Text	
Telefon	Short Text	
Diriginte	Short Text	

Field Properties

General | Lookup

Field Size	50
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Allow Zero Length	Yes
Indexed	Yes (No Duplicates)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None
Text Align	General

The field description is optional. It helps you describe the field and is also displayed in the status bar when you select this field on a form. Press F1 for help on descriptions.

Рис. 4.6. Диалоговое окно Таблица *Profesori*

2. Появится диалоговое окно Table с именем по умолчанию *Table1* (*рис. 4.6*). Оно состоит из двух областей:

a) область описания структуры документа;

b) область описания свойств поля, выбранного в первой зоне (*Field Properties*).

Область описания структуры документа разбита на три столбца:

– *Field Name* (идентификатор поля);
– *Data Type* (тип поля, то есть тип его значений);
– *Description* (описание поля).

Следовательно, для каждого создаваемого поля, необходимо уточнить его имя, тип данных и описать его назначение.

Идентификатор поля может содержать разные символы, за исключением символов „.” , „!”, „[”, „]”, пробела в начале и невидимых символов (например, возврат каретки).

Длина идентификатора поля не должна превышать 64 символа.

Типы полей допустимые в Access представлены в следующей таблице:

Название типа	Описание значений типа
Short Text	Строка буквенно-цифровых символов. До 255 символов
Long Text (sau Memo)	Большие тексты. До 64 КБайт
Number	Числа

Название типа	Описание значений типа
Large Number	Большие числа
Date/time	Календарные даты
Currency	Денежные значения
AutoNumber	Натуральные числа 1, 2, 3, ... автоматически генерируемые в заданном порядке при добавлении новой записи
Yes/No	Значения Yes или No
OLE Object	Изображения или звуки
Hiperlink	Веб-адреса
Attachement	Файл
Calculated	Вычисляемое поле
Lookup Wizard	Значения из таблицы или из списка значений

Описание поля (не является обязательным) может содержать некоторые пояснения относительно поля. Свойства полей будут описаны позже.

3. После определения полей **устанавливаем первичный ключ** для таблицы. Для этого выбираем нужное поле (значения которого не повторяются), затем выбираем опцию *Primary Key* из контекстного меню поля (либо нажимаем на соответствующую командную кнопку [🔑] на панели инструментов Access). В таблице на *рисунке 4.6*, поле *Cod_prof* было выбрано в качестве первичного ключа. Если мы не выберем первичный ключ для таблицы, то система Access предложит сделать это, сразу же как вы сохраните таблицу.

Если предложение принято пользователем, то в качестве ключевого будет выбрано первое поле типа *AutoNumber*, либо создано такое поле (с именем *ID* по умолчанию).

4. **Сохраняем таблицу**, выбрав команду *Save* из меню *File*. Появляется окно *Save As,* в котором записываем название таблицы.

Задание: Проанализируйте идентификаторы и типы полей таблицы на *рисунке 4.6*. Создайте похожую таблицу *Elevi* базы данных *Liceu*, описанной в предыдущей главе.

Свойства полей таблицы

Свойства полей – это характеристики, устанавливающие дополнительный контроль над тем, в каком виде данные поля хранятся, вводятся или выводятся. Свойства зависят от типа поля и уточняются в нижней части окна таблицы (*рис. 4.6*).

1. Свойство **Field Size** определяет формат величины данных поля и существует только для текстовых и числовых полей.
   - Для типа *Short Text* допустимыми являются значения от 0 до 255, таким образом устанавливается длина строки символов, которую можно сохранить в поле. По умолчанию эта длина равна 50.
   - Для типа *Number* можно выбрать одно из значений *Byte*, *Integer*, *Long integer* (значение по умолчанию), *Single*, *Double*, *Replication ID*, *Decimal*.

2. Свойство **Format** с помощью шаблона уточняет способ вывода данных поля. Доступно для всех типов полей, кроме OLE Object.

– В случае типов *Text* и *Memo*, шаблон можно создать с помощью символов:

Символ	Описание
@	Текстовый символ или пробел.
&	Текстовый символ не обязателен.
<	Все символы в нижнем регистре.
>	Все символы в верхнем регистре.
–	Выводит символ –.

Пример: Шаблон @@-@@-@@> выведет вместо текста *abcdef* текст *AB-CD-EF*.

– Для типов *Number* и *Currency* можно выбрать одно из значений *General number*, *Currency*, *Euro*, *Fixed*, *Standard*, *Percent* или *Scientific*.

– Формат типа *Date/Time* может быть *General Date*, *Long Date*, *Medium Date*, *Short Date*, *Long Time*, *Medium Time*, *Short Time*. Уточним, что если год записан только двумя цифрами, то для значений из интервала [00, 29] Access понимает года 2000–2029, а для значений из интервала [30, 99] – года 1930–1999.

Пример: Формат *Long Date* выведет дату 28.11.89 так: 28 ноября 1989.

– Тип *Yes/No* допускает один из форматов *Yes/No*, *On/Off*, *True/False*. В последнем случае в поле можно вписывать соответственно значения 1 либо 0.

3. Свойство **Input Mask** используется для создания шаблона ограничения символов (называется иногда *маской ввода*), которые можно вводить в поле. Для создания шаблонов допустимы следующие символы:

Символ	Описание
0	Цифра (не допускается перед ней знак + или –). Ввод обязателен.
9	Цифра (допускается перед ней знак + или –). Необязательный ввод.
#	Цифра (допускается перед ней знак + или –) либо пробел (при сохранении удаляется). Необязательный ввод.
L	Буква. Ввод обязятелен.
?	Буква. Необязательный ввод.
A	Буква или цифра. Ввод обязателен.
a	Буква или цифра. Необязательный ввод.
&	Любой символ или пробел. Ввод обязателен.

Символ	Описание
C	Любой символ или пробел. Необязательный ввод.
. , : ; - /	Разделители для календарных дат или для классов числа (единицы, тысячи, миллионы, миллиарды и т.д.). Разделители по умолчанию определятся установками в окне *Regional Settings* (можно открыть из панели *Control Panel* операционной системы *Windows*).
<	Преобразует буквы справа в строчные.
>	Преобразует буквы справа в прописные.
!	Делает обязательным ввод справа налево.
\	Выводит лишь символ, следующий после \ (например, шаблон \M выводит букву M).
"Şir de caractere"	Выводит лишь строку символов (без кавычек).
Password	Вместо введенных символов будут выводиться символы *.

Примеры:

a) Шаблон >L<L0\S допускает строки из 4 символов, первый из которых – заглавная буква, второй – строчная, третий – цифра, а последний – буква S.

b) Для записи телефонных номеров только в виде (+373 22) XX-XX-XX можно использовать шаблон "(+373 22)" 00\-00\-00.

Для создания шаблонов можно использовать программу *Input Mask Wizard* (запускается щелчком по кнопке ... из кассеты свойства *Input Mask*), предоставляющую 10 полезных форматов масок ввода.

4. Свойство **Caption** задает текст, который будет появляться в заголовке при использовании данного поля в запросах, формулярах, отчетах. Если это свойство остается незаполненным, то Access использует в качестве имени идентификатор поля.

5. Свойство **Default Value** устанавливает значение поля по умолчанию. Это значение появляется автоматически при добавлении новой записи.

6. С помощью свойства **Validation Rule** можно сформулировать условия на значение данных при их вводе в поле. Условия на значение – это выражения, записанные с помощью операндов и функций Access или VBA (*Visual Basic for Applications*). Например, условие >= 100 позволяет пользователю ввести в поле числового типа только числа, большие либо равные 100.

7. В свойстве **Validation Text** записывается текст, который появится в предупреждающем окне, если введенное в поле значение не удовлетворяет условию проверки, указанному в *Validation Rule*.

8. Свойство **Required** может принимать лишь значения *Yes* и *No*. Значение *Yes* обязывает пользователя заполнить данное поле. Нет смысла заполнять это свойство для поля, имеющего роль первичного ключа (так как первичный ключ не может быть пустым), либо если задано условие проверки *Is Not Null* (не является пустым).

9. В свойстве **Allow Zero Length** тоже можно использовать лишь значения *Yes* или *No*.

Это свойство есть только у полей типа *Text* и *Memo*. При значении *Yes* в поле может быть значение длины 0, то есть пустая строка, даже если свойство *Required* имеет значение *Yes*.

10. Свойство **Indexed** позволяет (для индекса *Yes* (*Duplicates Ok*)) или запрещает (для индекса *Yes* (*No Duplicates*)) повторение значений в поле. Существующий индекс может быть удален, если из раскрывающегося списка свойства выбрать значение *No*. Для первичного ключа индекс *Yes* (*No Duplicates*) появится автоматически (*рис. 4.6*) и будет недоступен для изменения.

11. Свойство **New Values** применяется только для полей типа *Auto Number*. Для значения *Increment* Access будет генерировать новые значения поля путем прибавления 1 к большему существующему значению. Если же зададим свойству *New Values* значение *Random*, тогда поле будет заполняться случайными значениями.

Вопросы и задания

1 **Примените!** Проанализируйте таблицу *Profesori* базы данных *Liceu* и опишите ее структуру.

2 **Объясните!** Какие типы данных можно хранить в базе данных, созданной в Access?

3 **Объясните!** Каким должен быть тип поля, чтобы в нем можно было хранить фотографии? Веб-адреса? Чью-либо биографию?

4 **Примените!** Что необходимо сделать, чтобы ввод новой записи был невозможен, если пользователь не заполнил ее два последних поля?

5 **Проанализируйте!** Некоторое поле, не являющееся первичным ключом, не позволяет вводить повторяющиеся значения. По какой причине?

6 **Объясните!** Каким образом можно выяснить, есть ли в базе данных *Liceu* информация об учениках с одинаковыми датами рождения?

7 **Примените!** Какому году принадлежит дата:

a) 01.01.01; b) 30.12.30; c) 17.12.89; d) 15.04.28?

8 **Разработайте!** Создайте таблицу Access, содержащую информацию о музыкальной коллекции. Включите в нее поля для хранения имени исполнителя, студии звукозаписи, года появления, формата (диск, кассета, CD и др.) и рейтинга (например, от 5 до 10).

9 **Разработайте!** Создайте таблицу Access, содержащую информацию о коллекции кулинарных рецептов. Включите в нее поля для хранения названия блюда, их тип (первые блюда, гарниры, жаркое, пирожные, десерты и др.), время приготовления, происхождение (молдавские, французские, японские и т.д.).

10 **Примените!** Создайте шаблон, обязывающий пользователя вводить в некоторое поле только целые числа:

a) из интервала [10..99]; b) трехзначные; c) отрицательные, из двух цифр.

11 **Примените!** Создайте шаблон, позволяющий вводить в некоторое поле номера удостоверения личности граждан Молдовы. Эти номера начинаются с заглавной буквы, за которой следуют 8 цифр.

4.6 Установка связей между таблицами

Мы уже знаем, что между двумя таблицами может существовать одна из связей: 1 : 1, 1 : M, M : M.

Система Access использует для вывода и специальное окно *Relationships* для создания связей.

Рассмотрим на примере создание связи между двумя таблицами.

Установим связь 1 : M между таблицами *Clase* и *Elevi* базы данных *Liceu*.

1. Щелкнем дважды на инструмент *Relationships* из меню *Database Tools*. Появится меню *Relationships Design* и окно *Relationships,* в котором представлены таблицы, для которых устанавливаются связи.
2. Щелкнем на кнопку *Add Table* из меню *Relationships Design*. Появится окно *Show Table* (*рис. 4.7*), из которого поочередно выберите таблицы *Elevi* и *Clase*, подтверждая всякий раз выбор нажатием кнопки *Add*. В окне *Relationships* появляются идентификаторы полей выбранных таблиц (*рис. 4.8*).

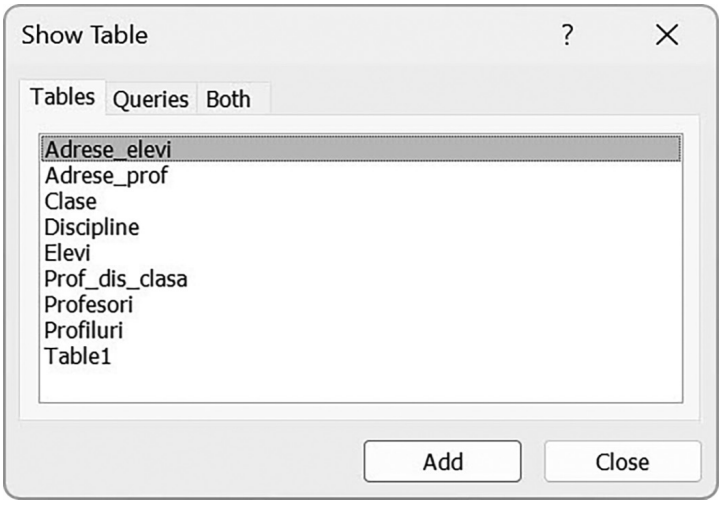


Рис. 4.7. Окно *Show Table* (Показать таблицу)

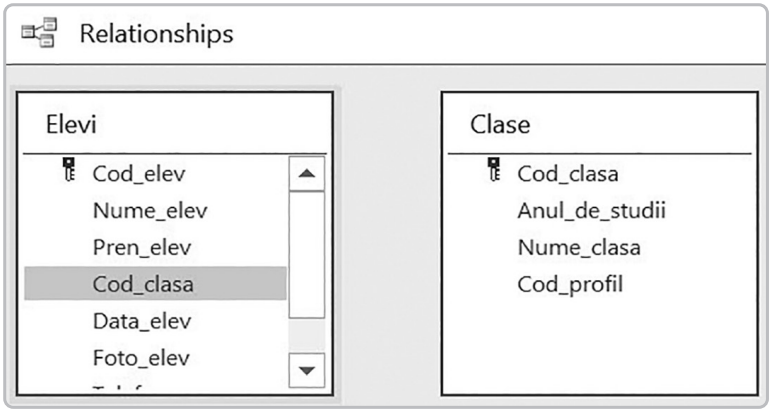


Рис. 4.8. Диалоговое окно *Relationships* (Связи)

3. Выделим поле *Cod_clasa* таблицы *Elevi*, затем, держа нажатой левую кнопку мыши, потянем его к полю *Cod_clasa* таблицы *Clase*. Появится окно *Edit Relationships*, в котором автоматически определился тип связи 1 : M (*Relationship Type: One-To-Many*). Можно также активизировать следующие характеристики связи (*рис. 4.9*):

a) *Обеспечение целостности данных (Enforce Referential Integrity);*

b) *Каскадное обновление связанных полей (Cascade Update Related Fields);*

c) *Каскадное удаление связанных полей (Cascade Delete Related Records).*

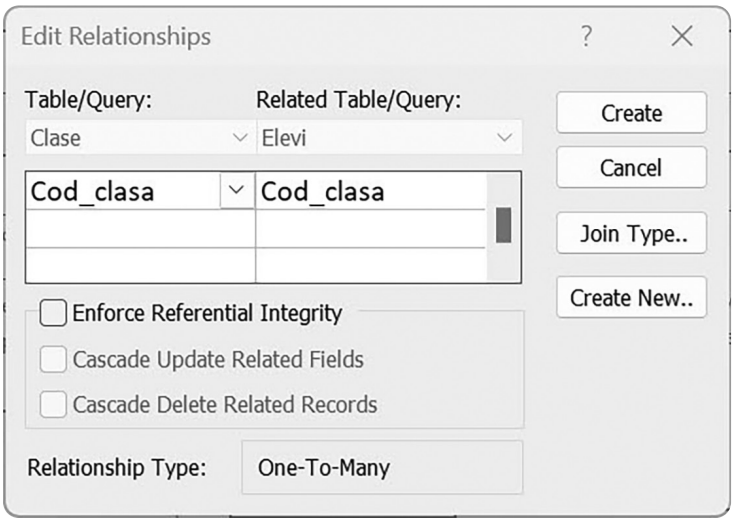


Рис. 4.9. Диалоговое окно *Edit Relationships*
(Редактирование связей)

Если активизирован флажок ***Обеспечение целостности данных***, то внешнее поле подчиненной таблицы сможет содержать лишь значения из первичного поля главной таблицы. Так, в поле *Cod_clasa* таблицы *Elevi* можно будет вписывать только «коды» классов, введенные в таблицу *Clase*.

Каскадное обновление связанных полей означает, что при изменеии некоторого значения *V* в первичном ключе главной таблицы автоматически изменятся все значения *V* во внешнем поле подчиненной таблицы. Например, если в таблице *Clase* заменим c01 – «код» 10-го A класса – на c001, то каждое значение c01 поля *Cod_clasa* таблицы *Elevi* заменится на c001.

Если активизирован флажок ***Каскадное удаление связанных полей***, то при удалении некоторой записи *X* из главной таблицы, в подчиненной таблице будут автоматически удалены все записи, содержащие в поле внешнего ключа значение поля первичного ключа записи *X*. Например, если удалим из таблицы *Clase* последнюю запись (с «кодом» c12, соответствующую 12-ому D классу), то из таблицы *Elevi* будут «удалены» все ученики 12-го D класса.

Вопросы и задания

1. Опишите алгоритм установления в Access связи между двумя таблицами.
2. **Систематизируйте свои знания!** Что означает *целостность данных* в связанных таблицах?
3. **Примените!** Школьные предметы принадлежат следующим куррикулярным областям:
   a) язык и общение;
   b) социально-гуманитарное образование (история, география, гражданское воспитание);
   c) математика и естествознание (математика, физика, химия, биология, информатика);
   d) технология;
   e) спорт (физическое воспитание).

   Откройте базу данных *Liceu*. Создайте и заполните необходимой информацией таблицу *Arii curriculare*, затем установите связь между этой таблицей и таблицей *Discipline*.
4. **Объясните!** Какова роль характеристики связи *Cascade Update Related Fields*?
5. **Объясните!** Для чего используется характеристика связи *Cascade Delete Related Records*?

4.7 Изменение таблиц

Ввод и редактирование данных

Для того чтобы ввести или отредактировать данные в таблице, ее необходимо открыть в режиме *Datasheet View* (это можно сделать инструментом *View* из меню *File*).

Смена режимов *Datasheet View* (режим редактирования, *рис. 4.10*) и *Design View* (режим проектирования) может осуществляться с помощью кнопки *View* из меню *File*.

Elevi

Cod_elev	Nume_ele	Pren_elev	Cod_clasa	Data_elev	Foto_elev	Telefon
e217	Pintilei	Pavel	c08	03.05.2006		022-47-69-44
e218	Pogujanschii	Alexandru	c08	03.11.2006		022-59-58-19

Рис. 4.10. Таблица Elevi отображается в режиме редактирования.

При редактировании содержания записи слева от нее появляется **указатель записи (УЗ)**, вид которого зависит от состояния записи (смотри следующую *таблицу*).

УЗ	Состояние записи
	Выбрана текущая запись.
*	Новая запись, в которую могут быть введены данные.
	Пользователь редактирует запись, однако изменения еще не сохранены.
	Запись блокирована другим пользователем и не может быть изменена (в многопользовательской среде – среде, в которой возможна одновременная работа нескольких пользователей в базе данных).

Для быстрого управления записями можно использовать инструменты из нижней части окна таблицы: 

, имеющие соответственно следующие функции (слева направо):

– выбор первой записи;
– выбор записи, предшествующей текущей;
– вывод порядкового номера текущей записи или выбор записи с порядковым номером, указанным в кассете;
– выбор записи, следующей за текущей;
– выбор последней записи;
– добавление новой записи.

Ввод и редактирование записей позволяют применять методы, свойственные работе с текстом. Например, данные можно копировать в *Clipboard* и удалять клавишами *Backspace* и *Delete*.

Одно нажатие клавиши *Esc* аннулирует все изменения в текущем поле, а двойное нажатие – в текущей записи.

Не обязательно заполнять информацией все поля записи, за исключением поля – первичного ключа (который не может быть пустым) и тех, для которых установлено значение *Yes* свойства *Required*.

Поле можно выбрать, щелкнув на его *заголовок* (ячейка, в которой выведено имя поля), а запись – щелкнув на ее *заголовок* (ячейка, в которой выводится указатель записи). Если, после выбора поля (записи), удерживать кнопку мыши нажатой и потянуть ее указатель к следующему полю (записи), то окажутся выбранными сразу два поля (записи).

Отметим, что **над данными, или записями, можно осуществлять следующие операции**:

– *добавление новой записи* перед текущей (команда *New Record* из контекстного меню заголовка записи или из меню *Insert* основной панели меню приложения Access);
– *удаление текущей записи* (команда *Delete Record* из контекстного меню заголовка записи или из меню *Edit* основной панели меню приложения Access);
– *заполнение полей*;
– *копирование содержимого некоторой ячейки* (команды *Copy* и *Paste* из контекстного меню ячейки);
– *копирование записи* (команды *Copy* и *Paste* из контекстного меню заголовка записи или из меню *Edit* основной панели меню приложения Access).

***Замечания*:**

1. При выполнении последней операции пользователь обязан изменить содержание поля первичного ключа.
2. Некоторые из указанных выше операций могут быть выполнены и с помощью кнопок на панели инструментов Access.
3. Для быстрого выполнения операций можно использовать и комбинации клавиш.
4. Все изменения автоматически сохраняются в записи в момент перехода к другой записи или при закрытии таблицы.
5. При вводе данных первыми заполняются главные таблицы, а затем подчиненные.

Изменение внешнего вида таблицы

Система Access выводит данные таблицы, соблюдая некоторые установки по умолчанию. Например, записи таблицы выводятся в возрастающем порядке значений поля первичного ключа, а поля следуют в том порядке, в котором были описаны при создании таблицы.

Пользователь может изменить способ представления информации из таблицы. Точнее, допустимы следующие **действия по изменению внешнего вида таблицы**:

a) *изменение порядка вывода полей;*
b) *изменение порядка вывода записей;*
c) *изменение высоты записи и ширины поля;*
d) *сокрытие столбцов;*
e) *выборка записей.*

Изменение порядка вывода полей осуществляется путем их перемещения. Чтобы переместить одно или несколько последовательных полей, выделим эти поля, затем, удерживая нажатой левую кнопку мыши, поместим ее индикатор на поле, перед которым хотим разместить выделенные поля.

Чтобы записи появились в порядке возрастания (убывания) значений некоторого поля, выделим это поле, затем щелкнем на кнопку  на панели инструментов (соответственно, кнопку ). Похожим образом осуществляется упорядочивание записей по нескольким полям. Отметим, что в этом случае упорядочивание будет осуществляться в порядке слева направо, то есть для поля справа записи будут упорядочены только, если в поле слева у них совпадают значения.

Действия по изменению высоты записей и ширины полей похожи на изменение высоты строк и ширины столбцов в редакторе текста или табличном процессоре.

Чтобы спрятать столбец, выделяем его, затем выбираем команду *Hide Fields* из его контекстного меню. Для того чтобы вернуть спрятанные поля, выбираем команду *Unhide Columns* из того же меню. Появляется диалоговое окно *Unhide Columns*, в котором отметим флажками нужные столбцы.

Выборка записей, то есть отбор тех записей, которые удовлетворяют некоторым условиям, осуществляется путем создания **фильтра**. Для этого выберем столбец, щелкнув на его заголовок, затем – на кнопку *Filter* из меню *Home*. В соседнем появившемся окне установим условия выборки (подобно фильтрации данных в *Excel*).

Удалить фильтр можно выбрав опцию *Clear Filter* из окна фильтра. Для это выбираем столбец (щелкнув по его заголовку), затем щелкнем на кнопку *Filter* из главного меню. В появившемся окне задаем критерии выборки (аналогично фильтрации данных в *Excel*).

***Замечания*:**

1. Изменения внешнего вида таблицы не сохраняются автоматически при закрытии таблицы. Пользователь может сделать это, нажав на кнопку *Save* на панели инструментов или с помощью команды *Save* из меню *Edit*.
2. Изменения внешнего вида таблицы не изменяют ее структуру.

Изменение структуры таблицы

Нормально, что в процессе проектирования (либо даже в процессе эксплуатации) базы данных возникает необходимость изменения структуры некоторых таблиц.

Внимание! Эти изменения могут нарушить целостность информации в базе данных. Например, уменьшение размерности текстового поля может привести к обрезанию его значений, а удаление поля-первичного ключа приведет к удалению записей из подчи-

ненной таблицы. Вообще-то, в зависимости от характеристик связей, система Access может не позволить некоторые изменения структуры таблицы.

В таких случаях изменения поля-первичного ключа или внешнего поля станет возможным только после удаления связи между таблицами, которую необходимо затем восстановить. Для изменения структуры таблицы ее необходимо открыть в режиме конструктора *Design View* (можно выбрать инструментом *View* из меню *File*).

В данном режиме допустимы следующие действия:

a) *добавление нового поля;*
b) *удаление некоторого поля;*
c) *изменение идентификатора поля;*
d) *изменение свойств полей;*
e) *установка первичного ключа;*
f) *устранение первичного ключа.*

Действия по изменению структуры таблицы подобны действиям по ее созданию.

Характеристика *Lookup* полей таблицы

Характеристика *Lookup* позволяет заменить поле таблицы некоторым раскрывающимся списком. В этом случае пользователь сможет заполнять данное поле, выбирая нужное значение из списка допустимых. Содержание списка можно загрузить из поля-первичного ключа таблицы, связанной с текущей, либо можно создать во время установки характеристики *Lookup*.

Воспользуемся специальной программой – мастером подстановок – для составления раскрывающегося списка для поля *Cod_clasa* таблицы *Elevi*.

1. Откроем таблицу *Elevi* в режиме конструктора. Выберем опцию *Lookup Wizard* из раскрывающегося списка возможных типов поля *Cod_clasa*.
2. Появится первое диалоговое окно *Lookup Wizard*. Выберем первую опцию, определив тем самым, что значения создаваемого списка будут взяты из связанной таблицы. Выбор второй опции означает, что элементы списка будут созданы на следующих этапах.
3. В последующих двух окнах сначала выберем таблицу *Clase*, затем поля *Cod_clasa, An_de_studii* и *Nume_clasa*. Значения выбранных полей появятся в созданном списке.
4. В четвертом окне настраивается ширина столбцов списка и задается его имя.

Вопросы и задания

1 **Систематизируйте свои знания!** Какие операции можно осуществлять с данными таблицы?

2 **Проанализируйте!** Какие изменения может повлечь за собой изменение структуры таблицы?

3 **Примените!** Откройте базу данных *Liceu*. Добавьте новое поле *Adresa_Web* в таблицу *Adrese_elevi*. Заполните это поле данными для первых 10 записей.

4 **Примените!** Откройте таблицу *Elevi* в режиме редактирования:

a) добавьте в таблицу 5 записей;
b) удалите предпоследнюю запись;
c) скопируйте данные первой записи в поля последней записи.

5 **Примените!** Откройте таблицу *Profesori* в режиме редактирования:
   a) выведите данные таблицы в следующем порядке: фамилия, имя, пол, телефон, дата рождения учителей;
   b) спрячьте поля *Salariu*, *CV_prof* и *Foto_prof*;
   c) верните для просмотра спрятанные поля.

6 **Практикуйтесь!** Выведите в алфавитном порядке список учеников базы данных *Liceu*.

7 **Практикуйтесь!** Выведите фамилии и имена учеников в убывающем порядке их возраста.

8 **Практикуйтесь!** Выведите список учителей базы данных *Liceu* в порядке убывания их заработной платы.

4.8 Выражения Access

Анализируя свойство полей *Validation Rule*, было отмечено, что условия проверки значений данных, вводимых в поле, записываются с помощью выражений Access. Далее вы увидите, что выражения используются и при формировании запросов на выборку данных, и при создании отчетов.

Выражение Access — это объявление, содержащее по крайней мере сочетание одного оператора и операнда: константы, идентификатора или функции.

При вычислении выражения возвращается одно значение. **Идентификатор** Access – это имя некоторого объекта базы данных. Таблица, поле, запрос, формуляр, отчет и сама база являются объектами. Как правило, идентификаторы записываются в квадратные скобки **[** и **]**.

Идентификатор некоторого «подобъекта» формируется из имени класса объектов и имени «подобъекта», разделенных точкой или восклицательным знаком. Таким образом, каждому объекту базы данных соответствует единственный идентификатор.

Иногда, всё же, в качестве идентификатора может быть использовано имя объекта, если это не создает неоднозначных прочтений.

Примеры идентификаторов: [Elevi].[Nume_elev], [Profesori].[Salariu], [Cod_elev].

Операторы Access

Рассмотрим 6 категорий операторов Access.

- **Арифметические операторы** (+, -, *, /, \, Mod, ^) применяются над числовыми значениями.

Пример: 15 \ 6 возвращает 2, а 15 Mod 6 возвращает 3, так как 15 = 6 · 2 + 3.

- **Операторы сравнения** сравнивают значения двух операндов и возвращают одно из логических значений *True* или *False*. В Access используются такие же операторы сравнения, как и в языке программирования Pascal: **<**, **<=**, **=**, **>=**, **>**, **<>**.
- **Логические операторы** Access применяются над логическими операндами и тоже совпадают с таковыми из Pascal: **And**, **Or**, **Not**, **Xor.**
- **Оператор присваивания =** присваивает значение некоторому объекту, переменной или константе.

- **Оператор слияния +** соединяет две символьные строки в одну.
- **Другие операторы.** Приведенные в следующей таблице операторы не входят ни в одну из перечисленных ранее категорий. Выражения, их содержащие, возвращают одно из значений *True* (или –1), *False* (или 0).

Оператор	Описание
Is	Применяется над значением Null (пустое значение) и проверяет является ли значение пустым или нет.
Like	Определяет, записана ли строка символов в соответствии с заданным в Like шаблоном. Шаблон записывается в кавычках " и ". При его составлении допускаются заменяющие символы (**?** для одного символа, **#** для цифры, ***** для любого числа символов, в том числе и их отсутствия) и списки значений. Список значений задается в квадратных скобках **[** и **]**. Если перед значением записан символ !, то принадлежащими списку считаются все значения, кроме тех, перед которыми записан **!**.
In	Определяет принадлежность некоторого значения заданному списку. Значения списка разделяются символом **;**.
Between	Определяет принадлежность некоторого числового значения заданному интервалу.

Примеры:

1. Выражение [*Elevi*].[*Telefon*] *Is Null* возвращает значение *True* только в случае, если в поле *Telefon* таблицы *Elevi* в текущей записи ничего не записано.
2. *Like* "*B*ov*" задает все строки символов, начинающиеся с буквы B и заканчивающиеся комбинацией букв ov. Следовательно, выражение "*Belousov*" *Like* "*B*ov*" возвращает значение *True*.
3. *Like* "[CK]#?" задает строки из 3-х символов: первый – одна из букв C или K, второй – некоторая цифра, третий – любой символ.
4. *Like* "*[5*ad–g*]" задает строки, заканчивающиеся цифрой 5 или одной из букв *a, d, e, f, g*.
5. *Like* "*[!ae]" задает строки, которые не заканчиваются символами *a* или *e*.
6. Выражение "*Duminica*" *In* ("*Luni*"; "*Marti*"; "*Miercuri*"; "*Joi*"; "*Vineri*") возвращает значение *False*, а выражение 4 *In* (2; 4; 7; 8) – значение *True*.
7. Выражение *Between* 2 *And* 10 эквивалентно выражению >=2 *And* <=10.

Замечание: При записи условий на значение в кассете свойства *Validation Rule* в выражении не записывается первый операнд, так как им считается идентификатор соответствующего поля. Так, правило проверки *In* ("Luni"; "Marti"; "Miercuri"; "Joi"; "Vineri") позволит пользователю вписать в соответствующее поле лишь одно из слов *Luni, Marti, Miercuri, Joi, Vineri.*

Функции Access

Функция возвращает с помощью своего имени некоторое значение. Access предоставляет более 100 встроенных функций для обработки различных типов данных: числовых, строк символов, календарных дат и др. Наиболее часто используемые функции приведены в следующей таблице:

Функции для обработки календарных дат

Функция	Возвращаемый результат
Date()	Текущая дата
DateAdd(*T*; *N*; *D*)	Календарная дата, получаемая добавлением к дате *D* или вычитанием из нее *N* (когда *N* отрицательно) календарных единиц типа *T*, где *T* может быть "yyyy", "q", "m", "ww", "d", "h", что, соответсвенно, означает лет, триместров, месяцев, недель, дней, часов
DateDiff(*T*; D_1; D_2)	Разность между датами D_1 и D_2, выраженная в единицах типа *T*
Time()	Текущее время
Now()	Текущие дата и время
Year(*D*)	Год, записанный 4 цифрами, соответствующий дате *D*
Month(*D*)	Порядковый номер месяца в году, соответствующий дате *D*
Day(*D*)	Порядковый номер дня в месяце, соответствующий дате *D*
WeekDay(*D*)	Порядковый номер дня в неделе, соответствующий дате *D* (1 – воскресенье, 2 – понедельник и т.д.)
Hour(*D*)	Час (целое число от 0 до 23), соответствующий календарному значению *D*

Функции для обработки текста

Функция	Возвращаемый результат
Asc(*C*)	Код ANSI символа *C*
Chr(*N*)	Символ, код ANSI которого равен *N*
InStr(S_1; S_2)	Позиция, начиная с которой строка S_2 содержится в строке S_1
Mid(*S*; N_1; N_2)	Подстрока строки *S* длиной N_2, начиная с позиции N_1. Параметр N_2 может отсутствовать, что означает получение подстроки из *S* путем удаления первых N_1–1 символов *S*
LCase(*S*)	Строка символов, полученная из строки *S* путем преобразования заглавных букв в строчные

Функция	Возвращаемый результат
UCase(*S*)	Строка символов, полученная из строки *S* путем преобразования строчных букв в прописные
Len(*S*)	Число символов в строке *S*
LTrim(*S*)	Строка символов, полученная из строки *S* путем удаления пробелов, имеющихся в начале
RTrim(*S*)	Строка символов, полученная из строки *S* путем удаления пробелов, имеющихся в конце
Trim(*S*)	Строка символов, полученная из строки *S* путем удаления пробелов, имеющихся в начале и в конце
Left(*S*; *N*)	Строка символов, полученная из первых *N* символов строки *S*
Right(*S*; *N*)	Строка символов, полученная из последних *N* символов строки *S*
Str(*X*)	Строка символов, полученная из знаков числового значения *X*, записанных в том же порядке
Val(*S*)	Число, полученное из символов строки *S* (если строка имеет подходящий формат)

Математические функции

Функция	Возвращаемый результат
Abs(*X*)	Модуль числа *X*
Atn(*X*)	Арктангенс (в радианах) числа *X*
Avg(*C*)	Среднее арифметическое значений поля *C*
Count(*C*)	Количество ненулевых значений поля *C*
Max(*C*)	Максимальное значение поля *C*
Cos(*X*)	Косинус числа *X*
Exp(*X*)	Значение e^X
Int(*X*)	Целая часть числа *X*
Log(*X*)	Десятичный логарифм числа *X*
Rnd()	Случайное число из интервала от 0 до 1

Функция	Возвращаемый результат
Sgn(*X*)	0, если число *X* положительно, –1, если *X* отрицательно
Sin(*X*)	Синус числа *X*
Sqr(*X*)	Квадратный корень числа *X*
Tan(*X*)	Тангенс числа *X*

Замечания:

1. Параметрами функций могут быть идентификаторы полей (записанные в скобках **[** и **]**).
2. Если параметром функции является календарная константа, то ее записывают заключенной между символами " и ".

Примеры:

1. *DateAdd*("*d*";–50; *Date*()) возвращает календарную дату, которая была 50 дней назад.
2. *Weekday*("27.09.1993") возвращает значение 2, так как 28 сентября 1993 был понедельник. (Понедельник считается вторым днем недели.)
3. *InStr*("Informatica"; "forma") возвращает 3.
4. *LCase*("INFORMATICA") возвращает текст "informatica".
5. *LTrim*("forma") возвращает текст "forma".
6. *Sgn*(– 20) возвращает значение –1.

Вопросы и задания

1. **Объясните!** Для чего в Access используются выражения?
2. **Примените!** Откройте базу данных *Liceu* и запишите условие на значение для поля *Nr_ore_saptamina* таблицы *Prof_dis_clasa*, которое позволяло бы пользователю вводить в это поле только целые положительные числа.
3. **Практикуйтесь!** Какое значение возвратит выражение:
   a) *Mid*("Propoziţie"; 3);
   b) *Mid*("Calculator"; 1; 3);
   c) *Int*(*Rnd*()*50);
   d) *Month*("15.11.1990");
   e) *Left*("Tractor"; 5);
   f) 5 *In* ("4"; "5"; "6"; "7"; "8");
   g) *Val* ("25") – 25;
   h) "R" *Like* "[TR]*";
   i) "R" + *Str*(*Date*())?

4 **Примените!** Напишите выражение, возвращающее:

a) среднюю заработную плату учителей из таблицы *Profesori*;

b) календарную дату, которая будет через 5 недель после текущей даты;

c) порядковый номер дня года для текущей даты;

d) порядковый номер дня недели для 1 января 2000;

e) количество дней между датами 5 марта 2005 и 5 декабря 2005.

5 **Учитесь учиться!** Напишите для поля *Gen_prof* таблицы *Profesori* условие на значение, которое позволит записывать в данное поле лишь значения *M* или *F*.

4.9 Основные понятия о запросах

Системы управления базами данных созданы для хранения и автоматической обработки данных. Даже в случае баз данных с несколькими сотнями записей, поиск вручную информации, удовлетворяющей определенным условиям, является достаточно сложным процессом. На практике существуют базы данных, содержащие миллионы записей. Например, некоторые операторы мобильной телефонной связи Республики Молдова имеют более 1 000 000 абонентов! В парламентских выборах в стране участвуют более 1,5 миллионов избирателей. Следовательно, при возможности электронного голосования необходимо обработать более 1,5 миллионов записей базы данных!

Поиск некоторых данных может привести к необходимости просмотра нескольких таблиц. Например, чтобы узнать, какие предметы изучает некоторый ученик из базы данных *Liceu*, необходимо проанализировать таблицы *Elevi*, *Clase*, *Prof_dis_clasa* и *Discipline*. Убедимся в этом на практике!

- Проанализируйте базу данных *Liceu* и определите, какие предметы изучает ученик *Dan Moraru*.

Предложенное задание – лишь маленький аргумент в пользу необходимости автоматической обработки информации базы данных.

Для быстрого выбора из одной или нескольких таблиц, наборов данных, удовлетворяющих определенным условиям, а также для ускорения процесса обновления данных, системы управления базами данных используют *запросы* – заявки на поиск и/или действие в соответствии с заданными условиями.

Запросы – это объекты системы управления базами данных, которые представляют собой заявки на поиск, обработку и/или на изменение данных базы.

Отметим, что при создании запросов, кроме записей из таблиц, в качестве источника информации могут использоваться и результаты других запросов, созданных ранее.

Например, для получения списка преподавателей с максимальной заработной платой создадим два запроса. Один будет находить максимальное значение *Max* в поле *Salariu*, а второй – сделает выборку записей из таблицы *Profesori*, у которых в поле *Salariu* значение совпадает с величиной *Max* (то есть – результатом первого запроса).

Чаще всего запросы используются для:

– просмотра некоторого подмножества записей таблицы, без того, чтобы открывать эту таблицу;

– вывода в виде одной таблицы данных из нескольких таблиц;
– обновления данных таблиц (изменение, добавление, удаление данных);
– осуществления вычислений над значениями полей и получения новой информации;
– создания итогов, средних значений и др.

В зависимости от типа действия и от результатов, *запросы* классифицируются так:

a) *запросы на выборку;*
b) *запросы на удаление некоторых записей;*
c) *запросы на изменение некоторых записей;*
d) *запросы с вычисляемыми полями;*
e) *запросы с группировкой данных и итогами;*
f) *перекрестные запросы.*

Запросы на выборку – это запросы, сформулированные на основании логических условий. Они выбирают подмножество данных из одной или нескольких связанных таблиц. Например, поиск учеников, родившихся до 10 января 1992, вывод списка учеников 10-го B класса – это запросы на выборку.

Запросы на удаление некоторых записей представляют собой запросы на удаление из таблицы всех записей, удовлетворяющих некоторым логическим условиям. Например, запрос на удаление из таблицы *Elevi* базы данных *Liceu* информации об учениках 12-го класса (в связи с окончанием лицея) представляет собой запрос такого типа.

Запросы на изменение некоторых записей изменяют значения некоторого поля таблицы по одному и тому же алгоритму. Например, увеличение на 50% значений поля *Salariu* таблицы *Profesori* может быть осуществлено с помощью запроса такого типа.

Иногда возникает необходимость вывести некоторую информацию в новые поля. Например, отдельно подсчитать возраст учащихся. Для этой цели воспользуемся ***запросом с вычисляемым полем***.

Запросы с группировкой данных и итогами применяются для суммирования данных поля, подсчета средних значений, нахождения наибольшего либо наименьшего значения и др. Например, подсчет общего числа часов за неделю, реализованных в каждом классе базы данных *Liceu*, можно осуществить запросом такого типа.

Перекрестные запросы предназначены для компактного представления информации в виде таблицы. Например, информация о количестве часов в неделю по каждому предмету, в каждом классе может быть выведена с помощью перекрестного запроса в виде следующей таблицы:

Anul_de_studii	Nume_clasa	Biologia	Chimia	...
10	A	2	3	...
10	D	1	1	...
11	A	3	2	...
12	D	1	1	...
...	...	...	...	...

Так как результаты запросов зависят от информации, содержащейся в таблицах, изменения, производимые в таблицах, автоматически влекут за собой изменения в запросах (очевидно, после их повторного выполнения).

В свою очередь, в случае запросов a) – d), изменение пользователем данных в запросе может привести к изменениям в таблицах. Из этих соображений результаты запросов a) – d) называют **динамическим набором данных**.

Динамический набор данных не запоминается. Он существует только в момент выполнения запроса.

Для создания нового запроса:

1. **Выполним щелчок на кнопку** *Query Design* () из меню *Create*. Автоматически появится меню *Query Design*.
2. Появляется окно запроса и диалоговое окно *Show Table* (*рис. 4.11*).

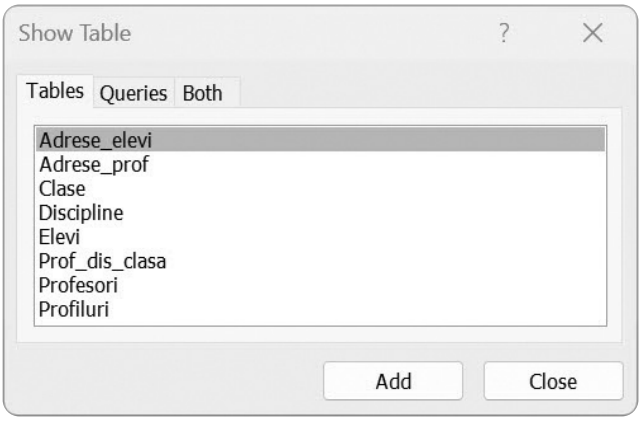


Рис. 4.11. Диалоговое окно *Show Table*
(Показать таблицу)

Выберем поочередно необходимые таблицы, нажимая всякий раз на кнопку *Add*. Поля выбранных таблиц появляются в окне запроса вместе с графическим представлением связей между таблицами (*рис. 4.12*).

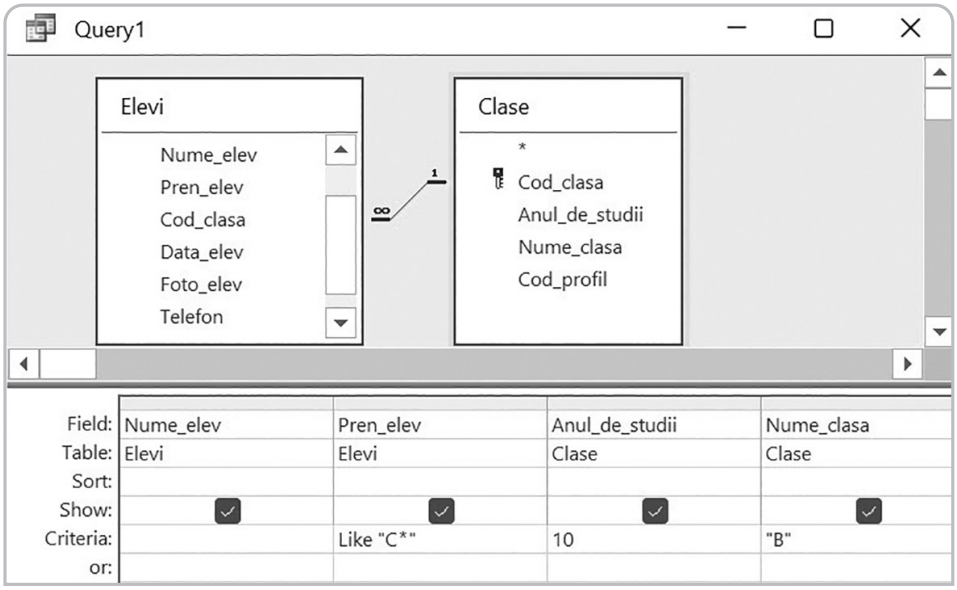


Рис. 4.12. Окно создания запроса на выборку

3. Окно запроса разбито на две зоны. В верхней зоне представлены названия полей таблиц. В нижней зоне выводится **формуляр QBE***, в котором пользователь может использовать примеры частичных объявлений либо выражения для создания запроса. Выбираем поля, которые понадобятся для написания условий выборки и те, значения которых будут выводиться. Добавить поле можно, щелкнув дважды на его идентификатор, либо выделив и перетащив его с помощью мыши из верхней части окна в формуляр. Порядок полей, выведенных в формуляр, может быть изменен похожим образом, которым мы меняли порядок полей в таблице, открытой в режиме *Datasheet View*.
4. Для каждого поля, добавленного в формуляр QBE, помимо его имени и названия таблицы, из которой оно взято (выведенные в стоках *Field* и *Table*), можно дополнительно уточнить:
   a) способ сортировки записей по данному полю (строка *Sort*), выбирая одно из значений *Ascending* (по возрастанию) или *Descending* (по убыванию);
   b) вывод или сокрытие значений поля при выполнении запроса (строка *Show*);
   c) условие выборки, которому будут удовлетворять выводимые значения (строки *Criteria* и *or*).
5. Можем **вывести данные**, указанные в запросе, до того, как сохраним этот запрос, воспользовавшись командой *Datasheet View* из меню *View*.
6. **Сохраним запрос** (кнопка *Save* или команда с таким же именем из меню *File*).

Пример: На *рисунке 4.12* представлен в режиме конструктора запрос на выборку списка учеников 10-го B класса, имя которых начинается с буквы C. На *рисунке 4.13* показан этот список еще до сохранения запроса.

Query1

Nume_ele	Pren_elev	Anul_de_stu	Nume_clas
Bujor	Călin	10	B
Cozariuc	Cătălina	10	B
Mihalachi	Cristina	10	B

Record: 1 of 3 No Filter Search

Рис. 4.13. Запрос в процессе создания, показанный в режиме *Datasheet View*

Вопросы и задания

1. **Объясните!** С какой целью используются запросы?
2. **Систематизируйте свои знания!** Охарактеризуйте основные типы запросов.
3. **Объясните!** Опишите алгоритм создания запроса.
4. **Объясните!** Почему результаты некоторых типов запросов называются *динамическими наборами данных*?

* Характеристика QBE (Query by Example) первоначально была создана для того, чтобы пользователи баз данных, без необходимости знать язык программирования, могли находить и выводить фрагменты данных. Со временем приложение QBE стало предпочитаемым инструментом для написания запросов. Большинство СУБД разработали собственные приложения QBE, позволяющие формулировать широкий спектр запросов.

5 **Примените!** Определите тип следующих запросов:

a) определение числа преподавателей мужского и женского пола;
b) определение средней заработной платы учителей базы данных *Liceu*;
c) вывод списка девушек из 11-ых классов;
d) определение числа часов, выполняемых еженедельно каждым учителем;
e) поиск учителей женского пола, преподающих в 10-ых классах.

6 **Разработайте!** Проанализируйте базу данных *Liceu* и сформулируйте по два запроса:

a) на выборку;
b) на удаление некоторых записей;
c) на изменение некоторых записей;
d) с вычисляемыми полями;
e) с группировкой и итогами;
f) перекрестных.

4.10 Запросы на выборку

Так как запросы на выборку используются чаще всего Access устанавливает именно этот тип, по умолчанию, для создаваемого нового запроса. Позже вы увидите, что пользователь должен провести дополнительно некоторые действия, чтобы изменить тип запроса.

Итак, чтобы создать запрос на выборку, выберем таблицы и нужные поля, воспользовавшись описанным в предыдущей теме алгоритмом.

Условия выборки

Очень важным моментом в процессе создания запроса на выборку является написание **условия выборки**.

Если условие формулируется для одного поля, то логическое выражение, контролирующее вывод данных, вписывается в ячейке *Criteria* для данного поля. Отметим, что оператор *Like* добавляется автоматически самим приложением Access, если в качестве условий выборки используются шаблоны, ограничивающие данные. В частности, в ячейку *Criteria* можно вписывать константы типа, совместимого с типом значений соответствующего поля.

Пример: Для вывода списка учеников по имени Ion, достаточно вписать в ячейку *Criteria* поля *Pren_elev* (из таблицы *Elevi*) строку символов "Ion".

Так как условия выборки являются логическими выражениями Access, для их записи можно использовать функции и операторы, изученные в предыдущем параграфе, в том числе и *логические*.

Кроме того, формуляр QBE предоставляет помощь в написании составных условий, состоящих из нескольких условий и операторов AND и/или OR. Так:

a) для одного поля можно определить несколько условий выборки, связанных между собой оператором OR: первое условие записывается в строке *Criteria*, все остальные – ниже, по одному в каждой ячейке;
b) два или несколько условий из строки *Criteria* считаются связанными оператором AND.

Примеры:

1. Если в формуляре запроса, из предыдущего примера, ниже значения "Ion" (в ячейке *or*) записать "Vasile", то запрос выдаст список учеников, имя которых Ion или Vasile. В случае добавления в ячейку *Criteria* поля *Nume_elev* условия "B*", запрос выдаст список учеников имя которых Ion, а фамилия начинается с буквы "B" и всех учеников по имени Vasile.

2. Запрос на *рисунке 4.14* выдаст список учеников 11-го В класса, родившихся в январе.
3. Запрос на *рисунке 4.15* выдаст список учеников 10–11 классов реального профиля. Такой же результат будет получен, если запишем в строке *Criteria* выражение 10 OR 11.
4. Запрос на *рисунке 4.16* выдаст список учителей с месячной заработной платой больше 9 000 леев и меньше либо равной 10 000 леев.

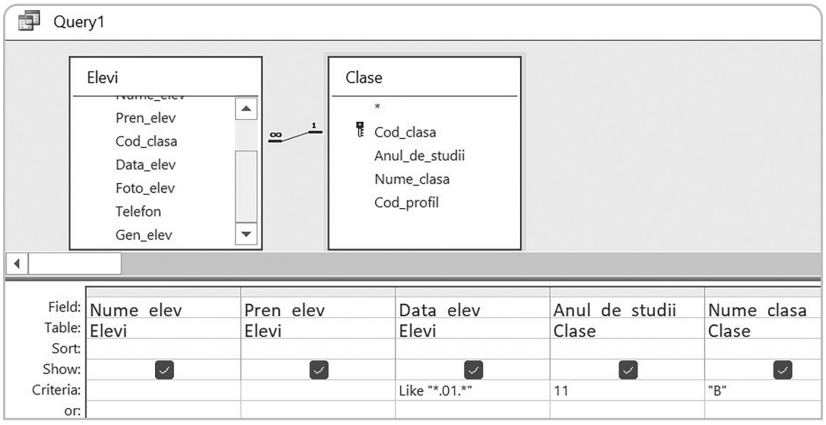


Рис. 4.14. Запрос на выборку списка учеников 11-го В класса, родившихся в январе

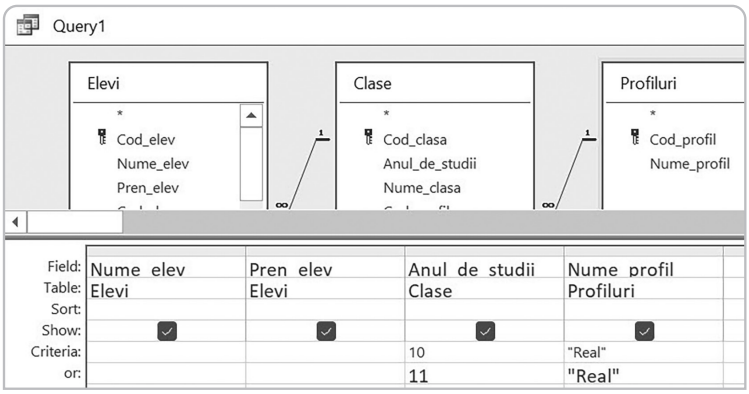


Рис. 4.15. Запрос на выборку списка учеников 10–11 классов реального профиля

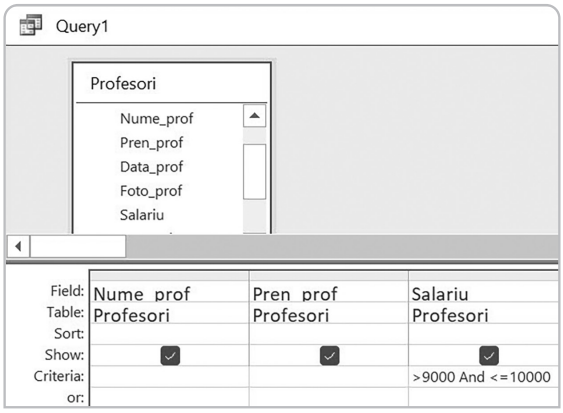


Рис. 4.16. Запрос на выборку списка учителей с месячной заработной платой больше 9 000 леев и меньше либо равной 10 000 леев

Запросы с параметрами

Очевидно, что невозможно описать все условия выборки, которые могут понадобиться пользователям. Более того, некоторые запросы могут отличаться друг от друга только значениями из формуляра QBE. Например, запросы, выводящие список предметов, изучаемых в 10-ом B классе, соответственно в 11-ом B отличаются только значениями поля *Anul_de_studii*.

В таких случаях можно создать один запрос, в котором, вместо конкретного значения *Criteria* соответствующего поля, вписать параметр.

Строка символов, заключенная в квадратные скобки [и], записанная в некоторой ячейке строки *Criteria,* воспринимается системой Access как **параметр**.

Для каждого параметра во время выполнения запроса сначала появляется диалоговое окно, в которое пользователь вписывает значение параметра (некоторое значение поля, для которого создан параметр), затем выводятся записи, у которых значения из поля с параметром совпадают со значением, указанным пользователем.

Как правило, строка символов, определяющая параметр, является некоторым текстом, подсказывающим пользователю, какое значение он должен ввести в диалоговое окно.

По умолчанию параметр считается типа *Text*. Чтобы изменить тип параметра, необходимо выбрать командную кнопку *Parameters* из меню *Query Design*. Появится окно *Query Parameters*, в котором вписываются все параметры и их типы.

Пример: На *рисунке 4.17* представлено окно некоторого запроса на выборку с тремя параметрами: первый – для поля *Anul_de_studii*, второй – для поля *Nume_clasa*, а третий (типа *Number*) – для поля *Nr_ore_saptamina*. С помощью данного запроса можно вывести список предметов, изучаемых учениками класса, указанного пользователем, с числом часов в неделю, тоже, указанным пользователем.

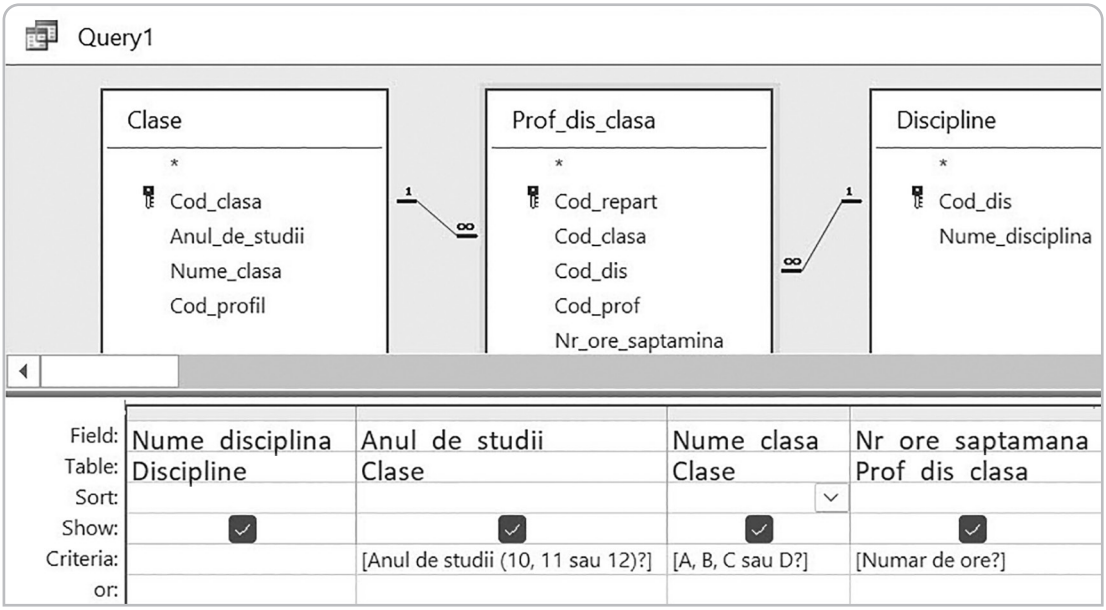


Рис. 4.17. Запрос на выборку с тремя параметрами

При выполнении запроса на *рисунке 4.17* поочередно появятся три диалоговых окна, в которые пользователь сначала введет год обучения (10, 11 или12), затем название класса (A, B, C или D) и соответственно число часов в неделю.

Вопросы и задания

1. **Систематизируйте свои знания!** Из чего состоят запросы на выборку?
2. **Объясните!** Какова роль условия выборки в запросе?
3. **Систематизируйте свои знания!** С какой целью используют параметры в запросе?
4. **Примените!** Откройте базу данных *Liceu*. Создайте запрос на выборку, который выведет список:
   a) девушек;
   b) классных руководителей;
   c) учителей, родившихся до 12 февраля 1970;
   d) учителей, родившихся в марте;
   e) мальчиков, в порядке возрастания их возраста;
   f) учеников, у которых не указан номер телефона;
   g) учителей мужского пола, не достигших возраста 50 лет;
   h) учеников, родившихся летом;
   i) учителей, номер телефона которых начинается с 48;
   j) учеников, не живущих в Кишиневе (Chişinău);
   k) учителей, чье имя начинается с буквы A или буквы E;
   l) учеников, не изучающих философию (Filosofia);
   m) учеников, изучающих математику (Matematica) 5 часов в неделю.
5. **Примените!** Откройте базу данных *Liceu*. Создайте запрос на выборку с параметром, который выведет список:
   a) предметов, изучаемых учеником, указанным пользователем;
   b) учителей, преподающих в классе, указанном пользователем;
   c) учителей, живущих в местности, указанной пользователем;
   e) учителей, преподающих предмет, указанный пользователем.
6. **Учитесь учиться!** Сформулируйте и создайте 5 запросов на выборку для базы данных *Liceu*.
7. **Учитесь учиться!** Для базы данных *Liceu* сформулируйте и создайте 5 запросов на выборку с параметром.

4.11 Запросы на изменение

Запросы на изменение применяются для создания новых таблиц на основе информации из таблиц, уже существующих в базе данных, а также для осуществления изменений в таблицах базы. В окне базы данных таким запросам соответствует иконка с восклицательным знаком.

Внимание! Запросы на изменение (за исключением создающих таблицы) изменяют содержание таблиц.

Запросы на создание таблиц

Запросы на выборку выводят информацию из таблиц в момент выполнения запроса. Результат такого запроса не сохраняется (в виде таблицы). Для этой цели имеется возможность преобразовать запрос в такой, который сможет сохранить результат в виде новой таблицы, то есть создаст из динамического набора новую таблицу.

Создадим запрос, генерирующий таблицу с данными об учениках 10-ых классов.

1. Сначала создадим запрос на выборку необходимых данных (*рис. 4.18*).
2. Из меню *Query Design* выберем *Make-Table*. Появится окно *Make Table*, где уточняется название новой таблицы (например, *Clasele_10*) и имя базы данных, в которой будет храниться новая таблица. По умолчанию эта таблица сохранится в текущей базе данных.
3. Записываем имя таблицы, нажимаем на кнопку OK и затем сохраняем запрос.
4. Для получения новой таблицы *Clasele_10* выполним этот запрос. Появляется окно с предупреждением. Подтверждаем намерение создания новой таблицы, нажав кнопку *Yes*.

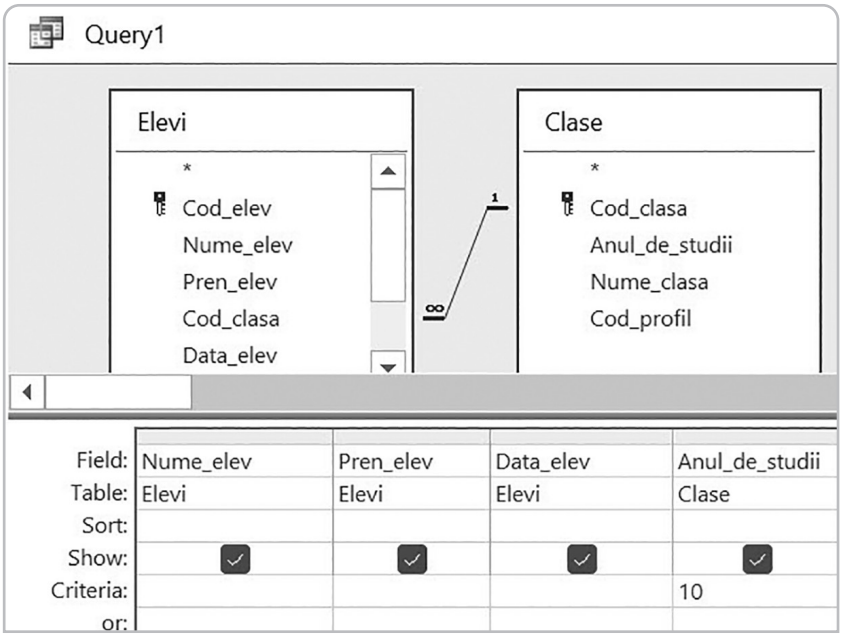


Рис. 4.18. Запрос на выборку данных об учениках 10-ых классов

Запросы на удаление записей

Часто возникает необходимость удаления некоторых записей из таблиц. Например, в случае базы данных *Liceu*, допустим, что необходимо, в связи с окончанием лицея, удалить данные об учениках и ученицах 12-ых классов.

1. Так как между таблицами *Clase* и *Elevi* существует связь **один ко многим**, с характеристикой *Cascade Delete Related Records* (Каскадное удаление связанных записей), достаточно удалить записи в таблице *Clase*, имеющие значение поля *Anul_de_studii*, равное 12. Обратное неверно, поскольку таблица *Elevi* подчинена таблице *Clase*.
2. Разрабатываем запрос, отображающий список классов. В меню *Query Design* (Конструктор запросов) выбираем *Delete* (Удалить). В форме QBE вместо строк *Show* (Показать) и *Sort* (Сортировать) появляется строка *Delete* (Удалить). В ячейке *Criteria* в поле *Anul_de_studii* пишем 12 (*рис. 4.19*).

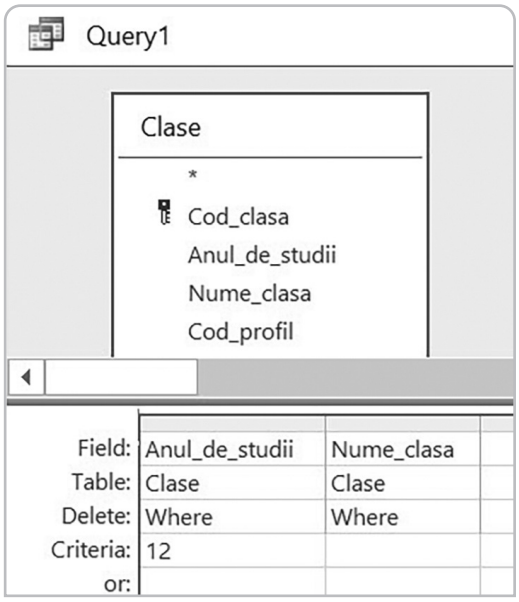


Рис. 4.19. Запрос на удаление записей (*Delete Query*)

3. Сохраним запрос. Удаление записей (*Delete Query*) осуществится после выполнения запроса.

Внимание! Так как удаленные записи не могут быть восстановлены, рекомендуется, до того как запрос на выборку будет преобразован в запрос на удаление, вывести его результаты и проверить их правильность.

Запросы на обновление записей

Если записи некоторого поля можно преобразовать по одному и тому же алгоритму, то для автоматизации этих преобразований используется запрос на обновление записей.

Сформулируем запрос, который увеличит на 50% значения из поля *Salariu* таблицы *Profesori*.

1. Создаем запрос на выборку, выводящий значения поля *Salariu*.

2. Из меню *Query Design* выбираем *Update*. В формуляре QBE вместо строк *Sort* и Show появляется строка *Update To*. Запишем выражение [Salariu]*1,5 в ячейке на пересечении строки *Update To* и поля *Salariu* (*рис. 4.20*).

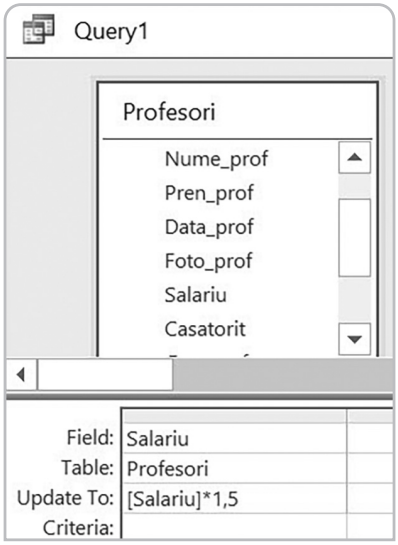


Рис. 4.20. Запрос на обновление записей (*Update Query*)

3. Сохраним запрос. Изменение записей произойдет после выполнения запроса.

Внимание! Запросы на обновление выполняются один раз. При повторном выполнении записи изменяются вновь. Так, если выполним дважды последний запрос, то заработная плата вместо 50% «вырастет» на 125%.

Запросы на добавление новых записей в существующие таблицы

Иногда возникает необходимость добавлять новые записи в одну таблицу из другой. Например, существует таблица *Elevi1*, содержащая данные об учениках, родившихся в мае. Мы желаем добавить в эту таблицу данные об учениках таблицы *Elevi*, родившихся в июне. Сделаем это так:

1. Создаем запрос на выборку, выводящий список учеников, родившихся в июне.

2. Из меню *Query Design* выбираем *Append*. Появляется окно *Append*. Из раскрывающегося списка *Table Name* выбираем *Elevi1*.

3. Сохраним, а затем выполним запрос.

Вопросы и задания

1 **Объясните!** С какой целью используют запросы на изменение?

2 **Систематизируйте свои знания!** Опишите алгоритм создания запроса:

a) генерирующего таблицу;
b) на обновление некоторых записей;
c) на удаление некоторых записей;
d) на добавление новых записей в существующие таблицы.

3 **Разработайте!** Проанализируйте базу данных *Liceu*. Разработайте запрос на создание таблицы:

a) T1 с данными о преподавателях женского пола;
b) T2 с данными об учениках, родившихся в мае;
c) T3 с данными о классных руководителях;
d) T4 с данными о преподавателях математики и химии;
e) T5 с данными об учениках, не живущих в Кишиневе.

4 **Разработайте!** Проанализируйте базу данных *Liceu*. Создайте запрос на удаление из таблицы:

a) T1 данных об учителях, рожденных летом;
b) T2 данных об учениках 11-го класса;
c) T3 данных о классных руководителях классов реального профиля;
d) T4 данных об учителях, не преподающих химию;
e) T5 данных об учениках, живущих в Крикова (Cricova).

5 **Учитесь учиться!** Проанализируйте базу данных *Liceu*. Создайте запрос, который:

a) увеличит заработную плату учителей на 500 леев;
b) уменьшит заработную плату на 300 леев учителям, преподающим только один предмет;
c) увеличит заработную плату на 25% учителям, рожденным до 01.01.1960.

6 **Учитесь учиться!** Проанализируйте базу данных *Liceu*. Создайте запрос, который добавит в таблицу:

a) T1 данные об учителях мужского пола, преподающих математику;
b) T2 данные об учениках, рожденных летом;
c) T3 данные об учителях, не являющихся классными руководителями, живущих в Кишиневе;
d) T4 данные об учителях, преподающих иностранный язык;
e) T5 данные об учениках 10-го класса, живущих в Кишиневе.

4.12 Итоговые запросы

Запросы с вычисляемыми полями

В предыдущих главах было отмечено, что при проектировании баз данных в таблицы не включаются поля со значениями, которые можно вычислить с помощью уже существующих полей.

По этим соображениям, например, в таблицу *Elevi* не включается поле *Varsta*. Значения такого поля зависят от содержания поля *Data_elev*. Создадим запрос, выводящий в новом поле возраст учеников базы данных *Liceu*. Этот запрос не затронет содержимое таблицы *Elevi*.

1. Создадим запрос на выборку на основании таблицы *Elevi*. В первые два столбца формуляра QBE включим поля *Nume_elev*, *Pren_elev*, а в третьем столбце, в месте для имени столбца, запишем выражение *Varsta*: *DateDiff*(“*yyyy*”;[*Data_elev*];*Date*()). Заметим, что *Varsta* – это имя нового поля, а функция *DateDiff*(*T*; *D1*; *D2*) возвращает разность, выраженную в календарных единицах *T* между датами D_1 и D_2 (см. тему *Функции Access*). Обратите внимание на то, что:
   – T равен “yyyy”, значит выражает года;
   – D_1 равен [Data_elev], то есть дата рождения ученика;
   – D_2 равен Date(), то есть текущая дата.
2. Сохраним запрос. Поле *Varsta* является динамическим набором. Оно существует пока выводятся результаты запроса.

Запросы с группировкой данных и итогами

Для получения результатов, основывающихся на записях одной или нескольких таблиц, воспользуемся запросами с *группировкой данных и итогами*.

Определим запрос, выводящий количество учеников каждого класса базы данных *Liceu*.

1. Создадим запрос на выборку на основе таблиц *Clase* и *Elevi*, включив поля *Anul_de_studii*, *Nume_clasa* и *Cod_elev*.
2. Щелкнем на кнопку *Totals* (Σ Totals) на панели инструментов. В формуляре QBE появляется строка *Total* (*рис. 4.21*). Заполняем ячейки этой строки:
   – в первых двух столбцах, в раскрывающихся списках, выберем значение *Group By* (так как группируем данные по году обучения и по названию класса),
   – в третьем столбце выберем функцию *Count* (так как подсчитываем количество значений поля *Cod_elev*).

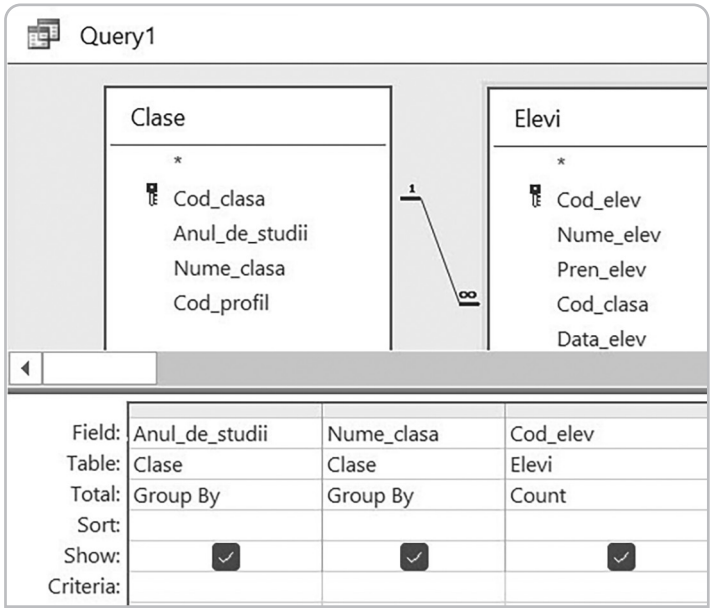


Рис. 2.21. Запрос с группировкой данных и итогами

3. Сохраним запрос. Результат запроса представлен на *рисунке 4.22*.

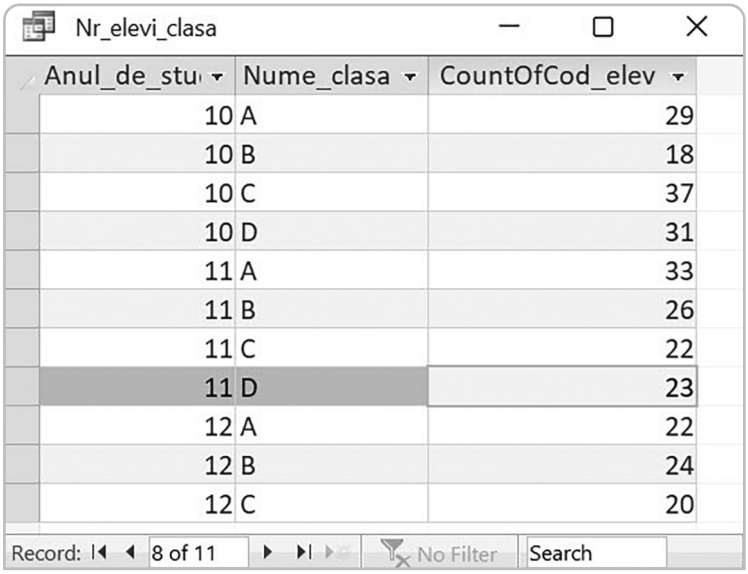


Рис. 4.22. Динамический набор данных, сгенерированный запросом с группировкой данных и итогами

Замечания:

1. Раскрывающиеся списки ячеек строки *Total* предоставляют различные *глобальные функции* (функции, применимые для групп ячеек с данными) для получения итогов: *Sum*, *Max*, *Min*, *Avg*, *First* и др.

2. В запросах с группировкой данных и итогами тоже можно использовать условия выборки. Например, если для поля *Cod_elev* предыдущего запроса написать условие >25, тогда запрос будет выводить только информацию о классах с числом учеников больше 25.

Перекрестные запросы

Перекрестные запросы – это запросы с итогами, в которых пользователь определяет вид таблицы для вывода результатов. Такой тип запроса рекомендуется, если необходимо подведение большого числа итогов. При создании перекрестных запросов необходимо иметь ввиду следующие ограничения:

a) названия строк результирующей таблицы могут быть значениями из одного или нескольких полей;

b) названия столбцов результирующей таблицы могут быть значениями только одного поля;

c) значения остальных ячеек результирующей таблицы являются результатами вычислений с помощью некоторой глобальной функции;

d) записи результирующей таблицы не могут быть отсортированы по вычисляемым полям.

Составим запрос, который выводит количество часов, предусмотренных еженедельно для каждого предмета в каждом классе базы данных *Liceu*.

1. Создадим запрос на выборку на основании таблиц *Clase*, *Discipline* и *Prof_dis_clasa*, включающий поля *Anul_de_studii*, *Nume_clasa*, *Nume_disciplina* и *Nr_ore_saptamina*.

2. Из меню *Query Design* выберем *Crosstab*. В формуляре QBE появляются строки *Total* и *Crosstab* (*рис. 4.23*).

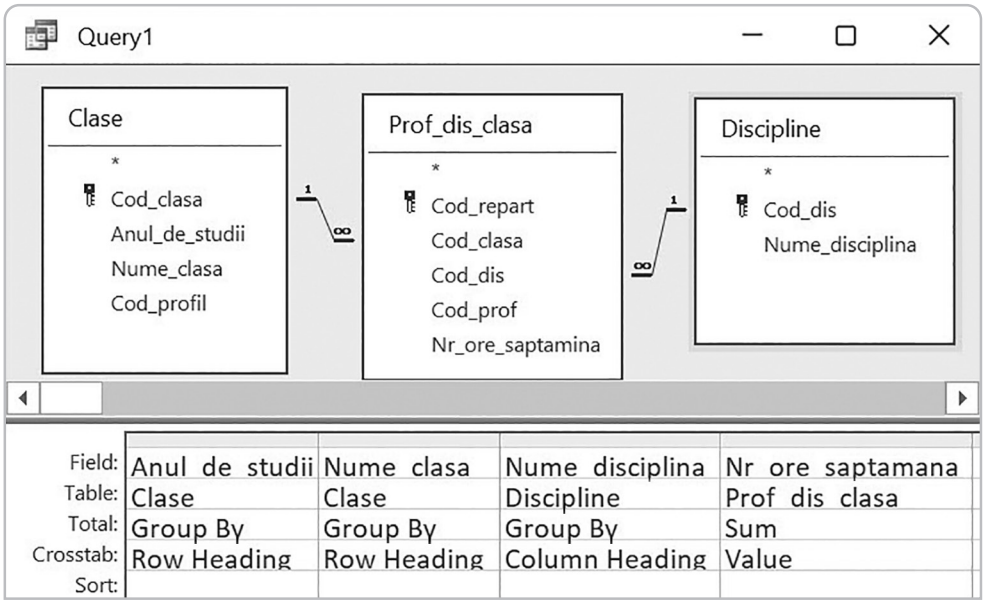


Рис. 4.23. Перекрестный запрос

3. Заполним ячейки строк *Total* и *Crosstab* как на *рисунке 4.23*. Таким образом, значения полей *Anul_de_studii* и *Nume_clasa* будут названиями строк результирующей таблицы, значения поля *Nume_disciplina* – названиями столбцов результирующей таблицы, а значения поля *Nr_ore_saptamina* будут просуммированы и заполнят все остальные ячейки результирующей таблицы (*рис. 4.24*).

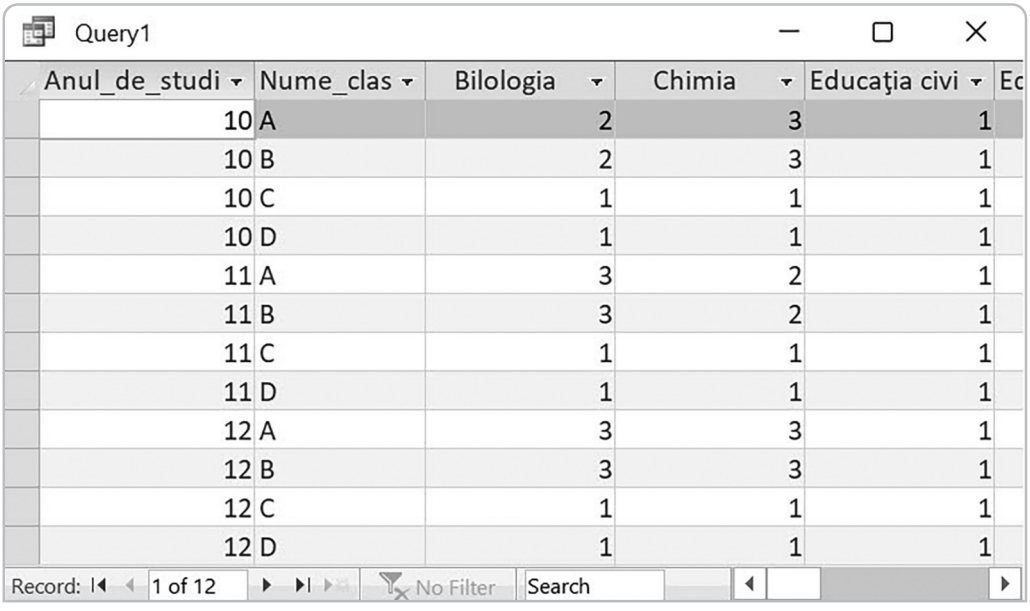


Рис. 4.24. Динамический набор данных, сгенерированный перекрестным запросом

Вопросы и задания

1 **Объясните!** Что собой представляют запросы:
a) на вычисление полей;
b) с группировкой данных и итогами;
c) перекрестные?

2 **Систематизируйте свои знания!** Опишите алгоритм создания запроса:
a) на вычисление полей;
b) с группировкой данных и итогами;
c) перекрестного.

3 **Примените!** Проанализируйте базу данных *Liceu*. Определите с помощью запроса:
a) возраст учеников, выраженный в днях;
b) число преподавателей мужского и женского пола;
c) количество классов каждого из профилей real и umanist;
d) среднюю месячную заработную плату учителей;
e) число учеников из каждой местности;
f) число часов, преподаваемое каждым учителем в течение недели;
g) число часов, преподаваемое в каждом классе за неделю;
h) значение минимальной заработной платы;
i) учителей с максимальной заработной платой;
j) количество учителей, имеющих заработную плату ниже средней.

4 **Учитесь учиться!** Проанализируйте базу данных *Liceu*. Сформулируйте и реализуйте по 2 запроса:
a) на вычисление полей;
b) с группировкой данных и итогами;
c) перекрестных.

4.13 Формуляры

Формуляры – это объекты базы данных системы Access, предусмотренные специально для создания удобного пользовательского интерфейса, обеспечивающего ввод, редактирование, вывод записей из таблиц и запросов.

Формуляры увеличивают скорость обработки данных и уменьшают вероятность ошибок. Кроме того, они позволяют выводить данные в более привлекательном формате, нежели в режиме *Datasheet View*.

С помощью формуляров можно осуществлять проверку данных, вводимых в данную базу из других баз данных, могут создаваться *субформуляры* (формуляры, входящие в другие формуляры), *кассеты поиска* (для быстрого доступа к записям), *списки опций* и др. Компоненты некоторого формуляра называются **элементами управления** или **объектами управления**.

Создание формуляра с помощью программы-мастера

Создадим формуляр, позволяющий редактировать данные в таблице *Elevi* базы данных *Liceu*.

1. В меню *Create* находятся четыре командные кнопки:

– *Form* (создание формуляра для редактирования записей выбранной таблицы. Он активен, только если выбрана таблица);

– *Form Design* (создание формуляра в режиме конструктора);

– *Blank Form* (создание пустого формуляра, т.е. без кнопок управления, полей и т.п.);

– *Form Wizard* (создание формы с помощью программы-мастера).

Мы выбираем последнюю опцию (*Form Wizard*).

2. Появится окно *Form Wizard* (*рис. 4.25*). Из комбинированного списка *Table/Queries* выделим значение *Table*: *Elevi*. Тогда в списке *Available Fields* появятся идентификаторы таблицы *Elevi*.

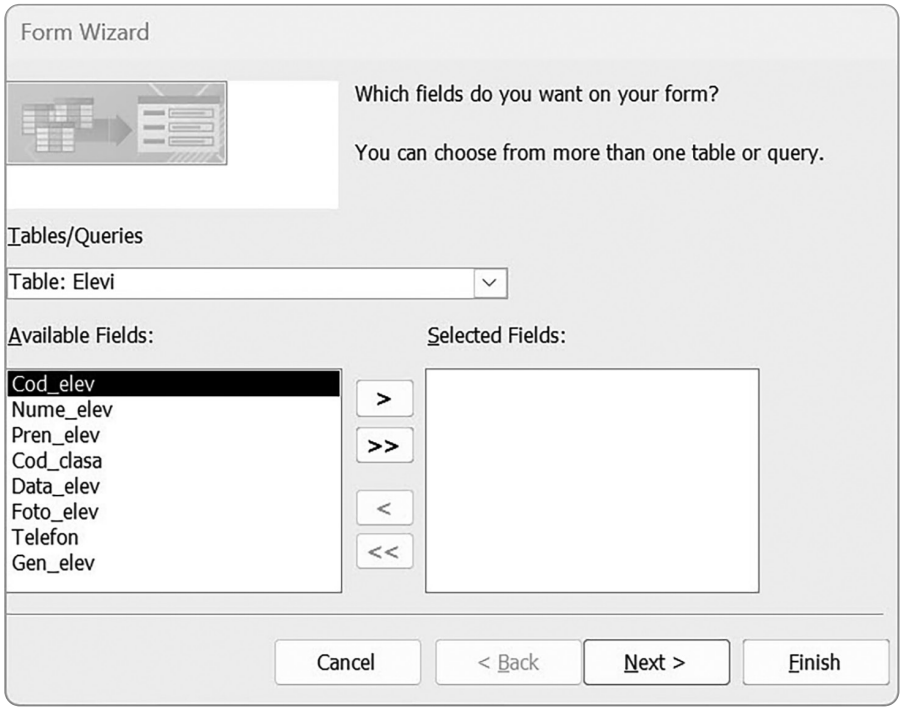


Рис. 4.25. Диалоговое окно *Form Wizard* (создание формы с помощью вспомогательной программы)

3. Выберем поля, которые будут включены в формуляр, подтверждая выбор кнопкой [>]. Для включения всех полей одновременно нажмем кнопку [>>] . Идентификаторы выбранных полей переходят в список *Selected Fields*. Кнопки [<] и [<<] используют для возврата полей, выбранных ошибочно. После выбора всех полей нажимаем кнопку *Next*.

4. Следующее окно *Form Wizard* предоставляет возможность выбора способа вывода полей формуляра (*рис. 4.26*). Выберем опцию *Columnar* (выбранные на предыдущем шаге поля будут выводиться одно под другим в виде столбца).

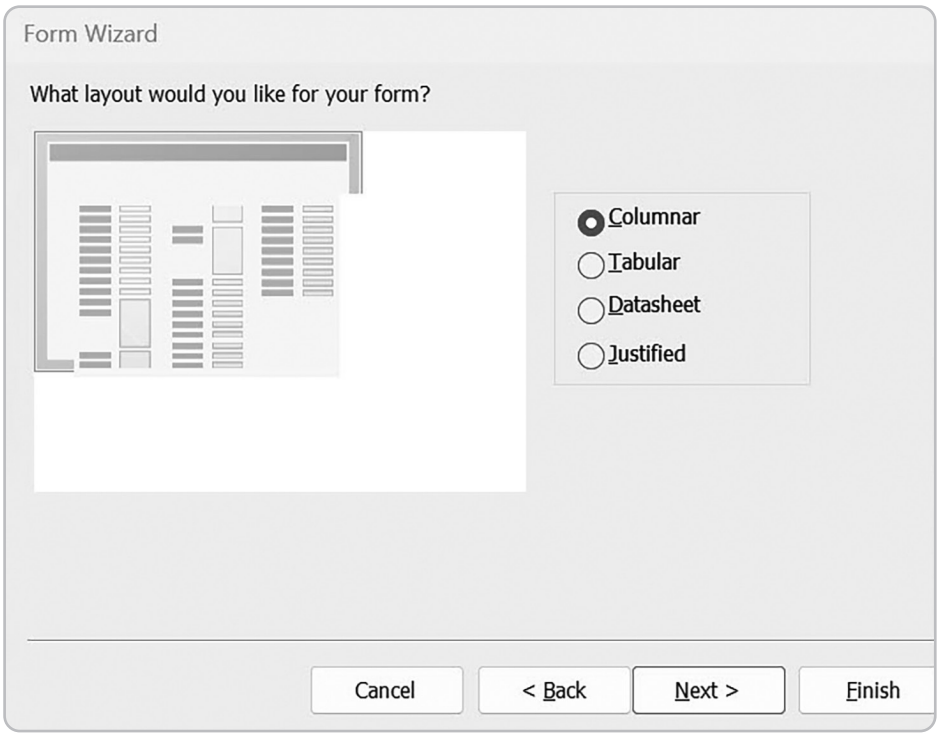


Рис. 4.26. Выбор способа вывода полей

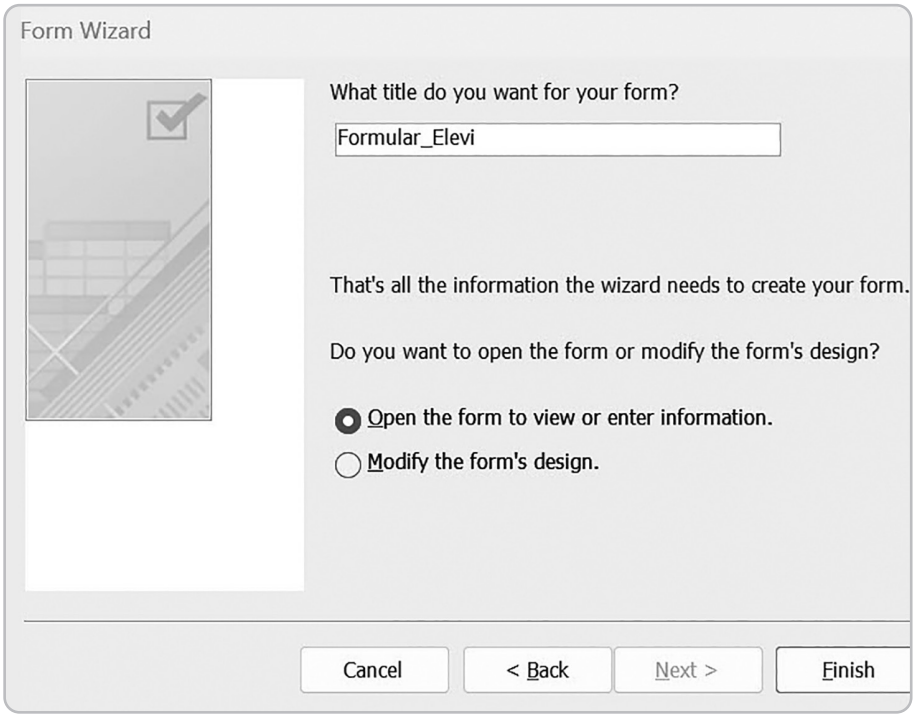


Рис. 4.27. Установка имени формуляра

5. В последнем окне зададим название формуляра (*Formular_Elevi)*. Если выберем опцию *Modify the Form's design* (*рис. 4.27*), то после нажатия кнопки *Finish* формуляр откроется в режиме конструктора. В противном случае, формуляр откроется в режиме редактирования данных.

6. Откроем формуляр в режиме просмотра и редактирования данных (*рис. 4.28*).

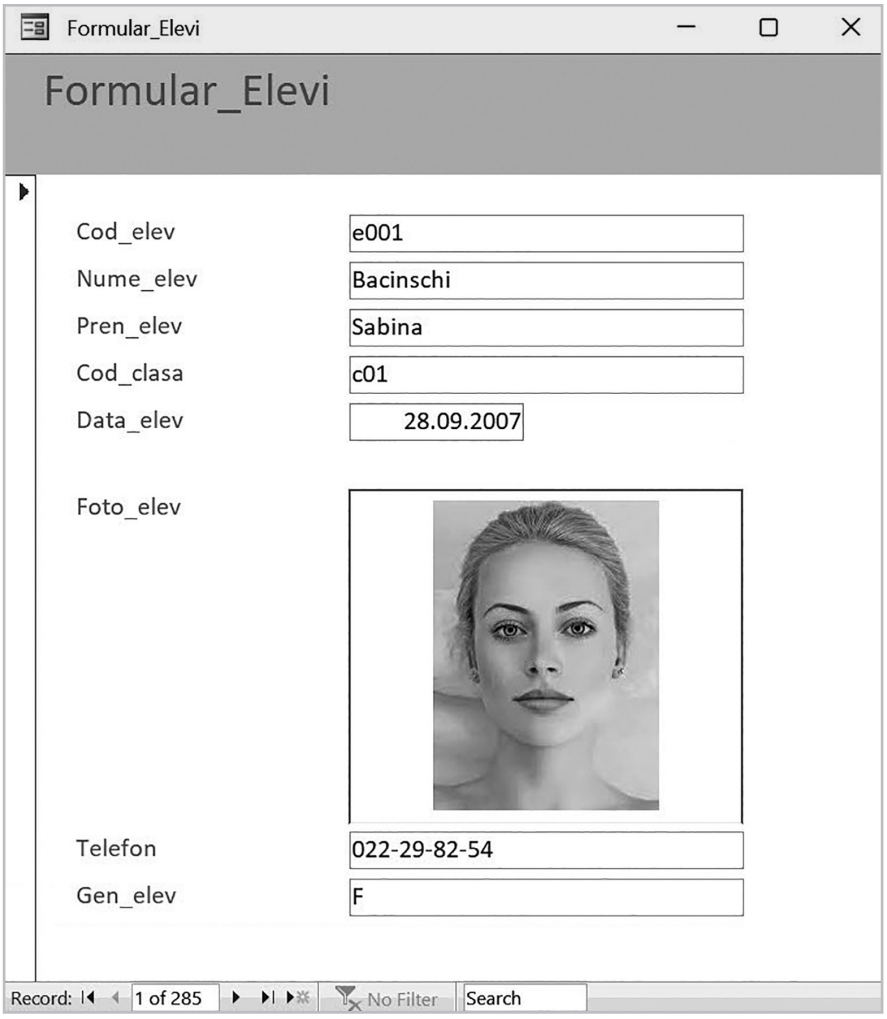


Рис. 4.28. Формуляр в режиме просмотра

Кнопки в нижней части формуляра используются для пролистывания записей. Так, кнопки , выводят первую и соответственно последнюю запись. Для просмотра предыдущей или последующей записей нажмите кнопку или соответственно .

При добавлении новых данных нажмите кнопку либо выберите *New Record* из меню *Insert*.

Замечания:

1. Формуляр может быть создан на основе большего числа исходных объектов (таблиц и запросов).
2. Изменение данных в формуляре приводит к их изменению в исходных объектах.

3. Если хотим вводить данные с помощью формуляра, он обязательно должен содержать первичный ключ и поля с установленным значением *Yes* для свойства *Required*. Если эти поля не заполнить, то новая запись не будет добавлена.

Создание и изменение формуляров в режиме проектирования

Режим проектирования – это самый исчерпывающий способ создания либо изменения формуляров, потому что он позволяет отредактировать любой элемент управления в формуляре.

1. Чтобы создать формуляр в режиме конструктора, дважды щелкнем кнопку *Form Design* в меню *Create*.
2. Появляется окно создания (редактирования) формы и меню *Form Design* с инструментами для редактирования формуляра (*рис. 4.29*).
3. В общем случае поверхность формуляра состоит из нескольких разделов (*рис. 4.29*). Изменение размера каждого раздела осуществляется с помощью мыши.

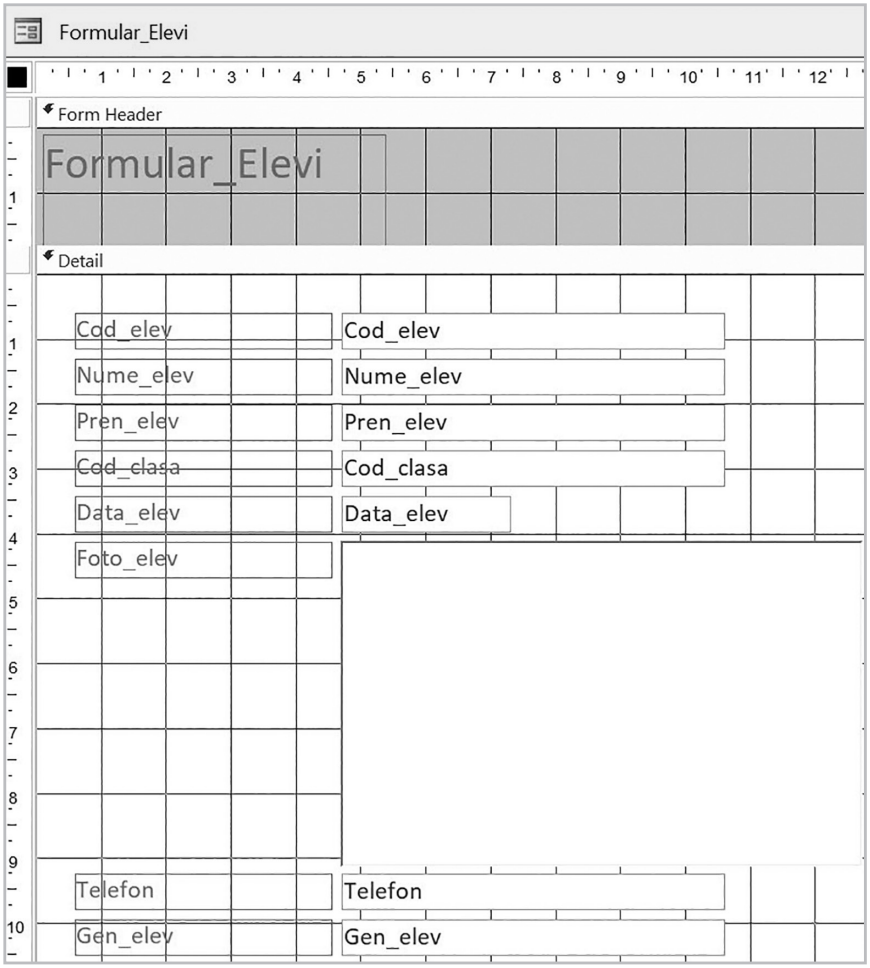


Рис. 4.29. Формуляр в режиме проектирования

Нижняя горизонтальная линия и вертикальная линия справа определяют его нижнюю и соответственно правую границы.

– Раздел **Detail** появляется автоматически при создании или редактировании нового формуляра. В нем выводятся данные из источников информации для формуляра.
– Разделы **Form Header** и **Form Footer** представляют собой верхний и нижний колонтитулы формуляра, предназначенные для вывода некоторой полезной информации. Эта информация остается неизменной при смене записей и может, к примеру, содержать рекомендации по применению формуляра, итоговые данные, данные об авторах, текущее время, дату создания формуляра и др.

Разделы *Form Header* (верхний колонтитул страницы) и *Form Footer* (нижний колонтитул страницы) не выводятся по умолчанию, а лишь при подключении опции *Header/Footer* из меню *Form Design*.

4. Добавление полей в формуляры осуществляется нажатием кнопки *Add Existing Fields* (из меню *Form Design*) по ранее указанному алгоритму (появляется окно *Field List*).

Группа инструментов *Controls* (*рис. 4.30*) в меню *Form Design* предоставляет возможность выбора и размещение элементов управления на поверхности формы.

Рис. 4.30. Группа инструментов *Controls*

Проанализируем некоторые инструменты этой группы:


Select Objects не используется для помещения элемента управления, а только для выбора (если он активирован) уже существующих в формуляре элементов.

Text Box создает текстовое окно, в котором пользователь сможет изменять текст.

Label создает текстовое окно, в которое можно вывести статический текст (который пользователь не сможет изменить).

Button создает командную кнопку, с помощью которой может быть выполнена некоторая команда.


Tab Control создает вкладки в формуляре, на каждой из которых могут размещаться свои элементы управления. С его помощью можно реализовать переключение между формулярами.


Link создает гиперссылку.


Navigation control создает подчиненный формуляр для навигации.

Option Group создает прямоугольную кассету, в которую можно поместить переключатели или флажки. В случае переключателей пользователь сможет выбирать одновременно только одну из предложенных опций.


Insert Page Breack создает разрыв страницы при печати, определяя начало новой страницы.


Combo Box создает поле со списком, сформированное из текстового поля, в которое пользователь может вписать значение или выбрать из раскрывающегося списка.


Line создает отрезки прямых, используемые для разграничения разделов и дизайна (цвет и толщину линий можно менять).


Togle Button создает выключатель – кнопку для активации или деактивации некоторого состояния.


List Box создает открытый список, из которого можно выбрать одно значение.


Rectangle создает прямоугольники, используемые для дизайна.


Check Box создает флажок для подтверждения или запрета некоторого состояния.


Unbound Object Frame отображает свободный объект OLE (например, созданный с помощью Microsoft Draw), который не является значением поля типа OLE некоторой записи.


Attachment позволяет прикреплять вложения (документы, презентации, изображения и т.д.).


Option Button создает кнопку (также называемую радиокнопкой), используемую для активация или деактивация состояния (или опции). Обычно она используется в группе с несколькими кнопками этого типа для организации вариантов исключения.


Subform/Subreport создает подчиненный формуляр (субформуляр) или подчиненный отчет.

Bound Object Frame отображает объект OLE – содержимое некоторого поля типа OLE (например, рисунок).


Image отображает статическое изображение (которое, будучи помещенным в формуляр, не может быть изменено).


Web Browser Control создает область для отображения содержимого веб-страницы непосредственно в формуляре.


Chart создает диаграмму.

- С точки зрения функциональности различают следующие **категории элементов управления**:
  a) *связанные элементы управления;*
  b) *вычисляемые элементы управления;*
  c) *независимые элементы управления*.

Связанные элементы управления служат для вывода и редактирования данных из полей таблиц и запросов, на основе которых был построен формуляр. Связанный элемент управления состоит из поля (в которое выводятся данные) и метки, ассоциируе-

мой с этим полем (в которой выводится пояснительный текст, например, имя поля). По умолчанию, метка связанного поля совпадает с идентификатором поля, значения которого выводятся в поле данных этого элемента управления (*рис. 4.31*). В режиме *Datasheet View* в поле, изображенного на *рисунке 4.31* формуляра, созданного ранее (*рис. 4.29*), будет отображаться номер телефона.

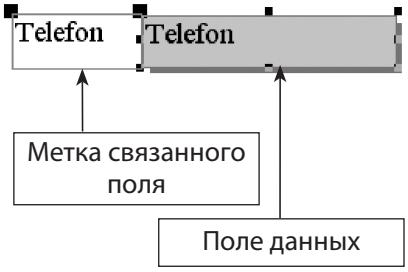


Рис. 4.31. Связанный элемент управления

Можно менять размеры и перемещать методом *Drag & Drop* компоненты элементов управления.

Добавить связанный элемент управления можно, нажав кнопку *Add Existing Fields* в меню *Form Design* или вставив элемент *Text Box* и указав имя необходимого поля в кассете *Unbound*.

Несмотря на свое название, текстовое поле может выводить, кроме текстов, числовые, календарные и другие данные.

По умолчанию, Access выравнивает текст по левой границе поля, а числа – по правой.

Любой связанный элемент управления можно детально редактировать, вызвав опцию *Properties* из контекстного меню элемента. При выборе этой опции появляется окно *Property Sheet*, в котором отображаются все свойства выбранного из раскрывающегося списка элемента (*рис. 4.33*).

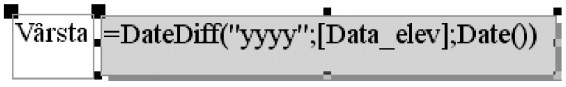


Рис. 4.32. Элемент контроля Vârsta elevului

Вычисляемые элементы управления применяются для вывода значений выражений и могут обрабатываться точно так же как и связанные элементы управления. Добавить такой элемент на поверхность формуляра можно с помощью инструмента *Text Box*, только в текстовом поле *Unbound* записывается выражение (после знака =), значение которого появится при просмотре в этом поле.

Например, элемент управления на *рисунке 4.32*, будучи добавленным в ранее созданный формуляр, будет выводить возраст учеников.

Независимые элементы управления не связаны с информацией в таблицах или запросах, а значит, не меняются при переходе от одной записи к другой. Применяются для создания дизайнерских эффектов.

- **Свойства элемента управления** определяют различные его характеристики: цвет, размеры, источник данных, позицию на поверхности формуляра, реакцию на определенные действия пользователя и др.

Изменить свойства элемента управления можно в специальном окне, появляющемся при двойном нажатии на кнопку мыши, выбрав ею нужный элемент (*рис. 4.33*). Все свойства сгруппированы в четыре категории (закладки): *Format*, *Data*, *Event*, *Other*. Группа *All* включает все свойства, упорядоченные в алфавитном порядке.

- Группа **Format** включает свойства, определяющие внешний вид объекта (размеры, цвет, формат и др.).
- Группа **Data** включает свойства относительно источника записей, управляемых данным элементом.
- Группа **Event** включает список событий (действия со стороны пользователя или приложения), на которые может реагировать элемент.
- Группа **Other** включает все остальные свойства. Они относятся к окнам операционной системы Windows.

• Формуляр может быть распечатан, как и любой другой объект базы данных, выбором опции *Print* из меню *File*.

Для того чтобы просмотреть формуляр в том виде, в каком он будет распечатан, воспользуйтесь кнопкой *Preview* на панели инструментов или выберите опцию *Preview* из меню *File*.

Рис. 4.33. Свойства текстовой кассеты *Nume_prof*

Подчиненный формуляр

Подчиненный формуляр (субформуляр) – это формуляр, включенный в другой формуляр. Включить другой формуляр в данный можно с помощью программы-мастера или путем самостоятельного проектирования. Мы рассмотрим только случай применения мастера, который является доступным, если активирован инструмент *Use Control Wizard* в кассете *Controls*.

Пусть дан формуляр *Clase*, созданный для вывода и редактирования данных о классах базы данных *Liceu*. Включим в него подчиненный формуляр, в котором будут отображаться данные об учениках выбранного в основном формуляре класса.

1. Откроем формуляр *Clase* в режиме *Design*. Активируем сначала кнопку *Control Wizard*, затем кнопку *Subform/Subreport* из кассеты *Controls* и щелкнем мышью на поверхности формуляра, в месте, где появится подчиненный формуляр. Появилось окно *SubForm Wizard,* в котором выделим таблицу *Elevi*.
2. В следующем диалоговом окне выделим поля таблицы *Elevi*, а в третьем окне подтвердим название поля связи (*Cod_clasa*) между формуляром и субформуляром.
3. В последнем окне *SubfForm Wizard* напишем имя субформуляра (*Lista_elevi*). Перейдя в режим просмотра *Form View*, увидим список учеников класса, выбранного в основном формуляре (*рис. 4.34*).

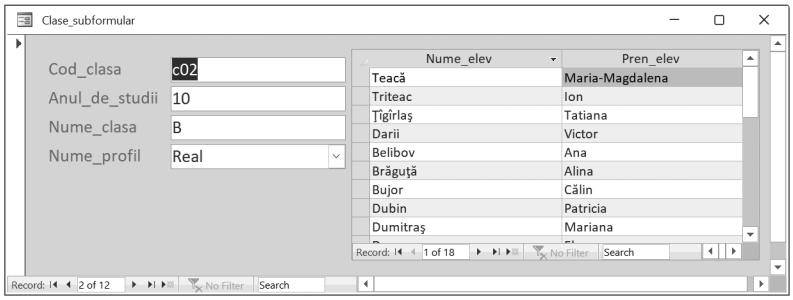


Рис. 4.34. Формуляр с подчиненным формуляром

Замечание: Субформуляр *Lista_elevi* автоматически сохранится на диске при сохранении основного формуляра.

Вопросы и задания

1. **Объясните!** Для каких целей применяются формуляры?
2. **Систематизируйте свои знания!** Какие объекты базы данных могут быть источниками данных для формуляров?
3. **Проанализируйте!** Охарактеризуйте зоны формуляра в режиме конструктора.
4. **Объясните!** Какова роль элементов управления некоторого формуляра?
5. **Систематизируйте свои знания!** Опишите категории элементов управления.
6. **Объясните!** Каким образом можно изменить свойства некоторого элемента управления?
7. **Примените!** Проанализируйте базу данных *Liceu*. Создайте с помощью программы-мастера формуляр для редактирования данных:
   a) об учителях базы данных, включая их адреса; b) о предметах базы данных.
8. **Практикуйтесь!** Добавьте в созданный ранее формуляр текстовое поле, в которое выводятся текущие дата и время.

9 **Практикуйтесь!** Добавьте в созданный ранее формуляр текстовое поле, в которое выводится число дней, оставшихся до конца: a) текущего года; b) текущего учебного года.

10 **Учитесь учиться!** Проанализируйте базу данных *Liceu*. Создайте формуляр, выводящий данные:

a) о каждом классе, содержащий подчиненный формуляр со списком учителей, преподающих в классе, выбранном в основном формуляре.

b) о каждом классе, содержащий подчиненный формуляр со списком предметов, изучаемых в классе, выбранном в основном формуляре.

c) данные о каждом учителе базы, содержащий подчиненный формуляр со списком предметов, преподаваемых учителем, выбранным в основном формуляре.

4.14 Отчеты

Отчеты представляют собой конечный продукт базы данных. В них комбинируются данные из таблиц, запросов и формуляров для печати либо для записи на диск в формате, легко воспринимаемом для чтения. В отличие от формуляров, отчеты:

a) специально предусмотрены для печати;

b) не могут менять данные в таблицах или запросах, на базе которых они созданы;

c) не выводят данные в форме таблицы (нет режима типа *Datasheet View*).

Как и в случае формуляров, отчеты могут создаваться в режиме программы-мастера и в режиме конструктора.

Создание отчета с помощью программы-мастера

1. В меню *Create* есть четыре кнопки для создания отчетов:
- *Report* (создание отчета, соответствующего выбранной таблице или запросу. Активна только если выбрана таблица или запрос);
- *Report Design* (создание отчета в режиме конструктора);
- *Blank Report* (создание пустого отчета, в который вы будете добавлять информацию);
- *Report Wizard* (создание отчета с помощью программы–мастера).

Дважды щелкаем по четвертому варианту.

2. Появится окно *Report Wizard* (*рис. 4.35*). В раскрывающемся списке *Tables/Queries* выберем по очереди таблицы *Clase*, *Elevi* и запрос *Varsta*, а из кассеты *Available Fields* (Доступные поля) выбираем поля *Anul_de_sudii*, *Nume_clasa* (из таблицы *Clase*), *Nume_elev*, *Pren_elev* (из таблицы *Elevi*) и *Varsta* (из запроса *Varsta*).

3. В следующем окне мастера отчетов выберем поля (сначала *Anul_de_studii*, затем *Nume_clasa*), после чего записи будут сгруппированы (*рис. 4.36*).

4. В третьем окне Report Wizard указываем поля, по которым будут сортироваться записи (*рис. 4.37*).

Используя кнопку *Summary Options*, можем указать для каждого числового поля один или несколько вариантов расчета: сумма, среднее арифметическое, минимальное или максимальное значение. Для расчета среднего возраста учащихся (для каждого класса и всего лицея) выбираем опцию *Avg* (*рис. 4.38*).

Включение опции *Summary Only* будет отображать только средний возраст в целом по лицею.

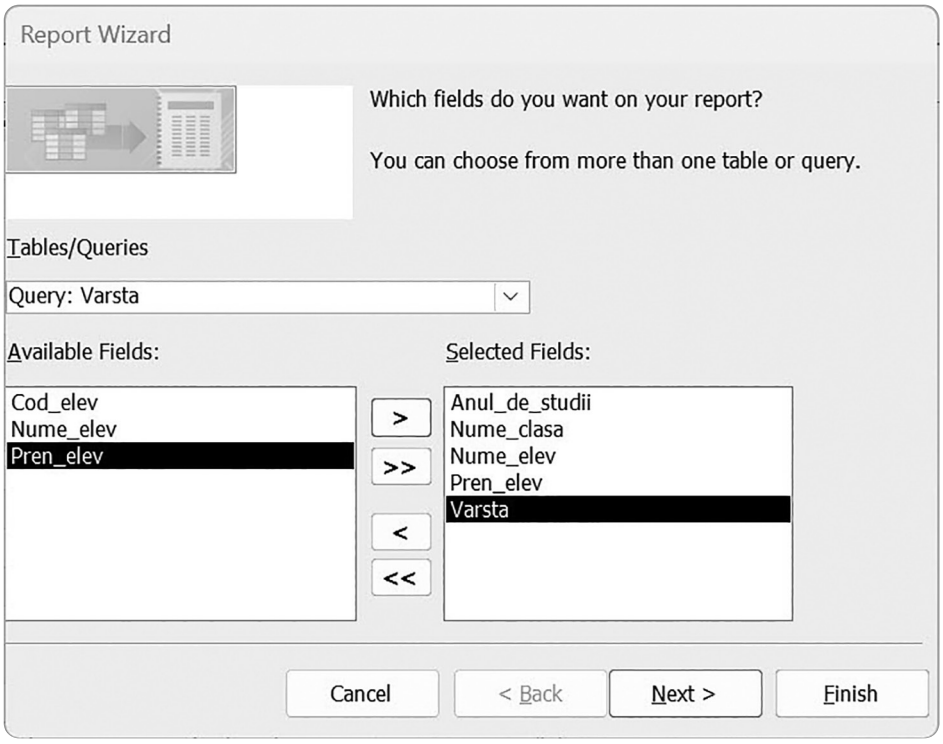


Рис. 4.35. Диалоговое окно *Report Wizard* (Ассистент отчетов)

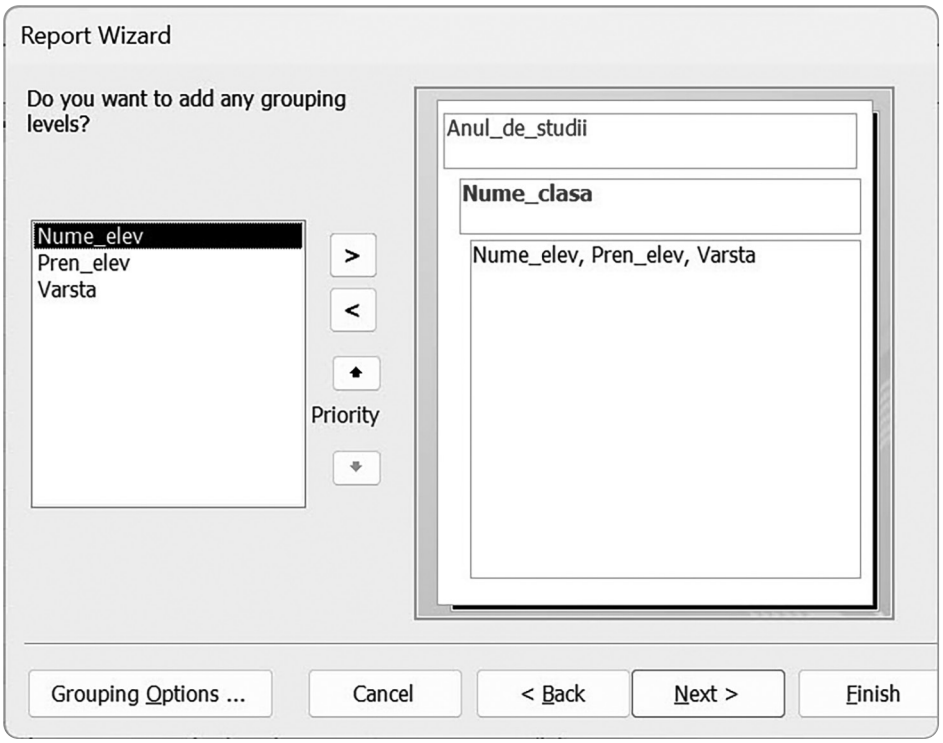


Рис. 4.36. Выбор полей для группировки записей

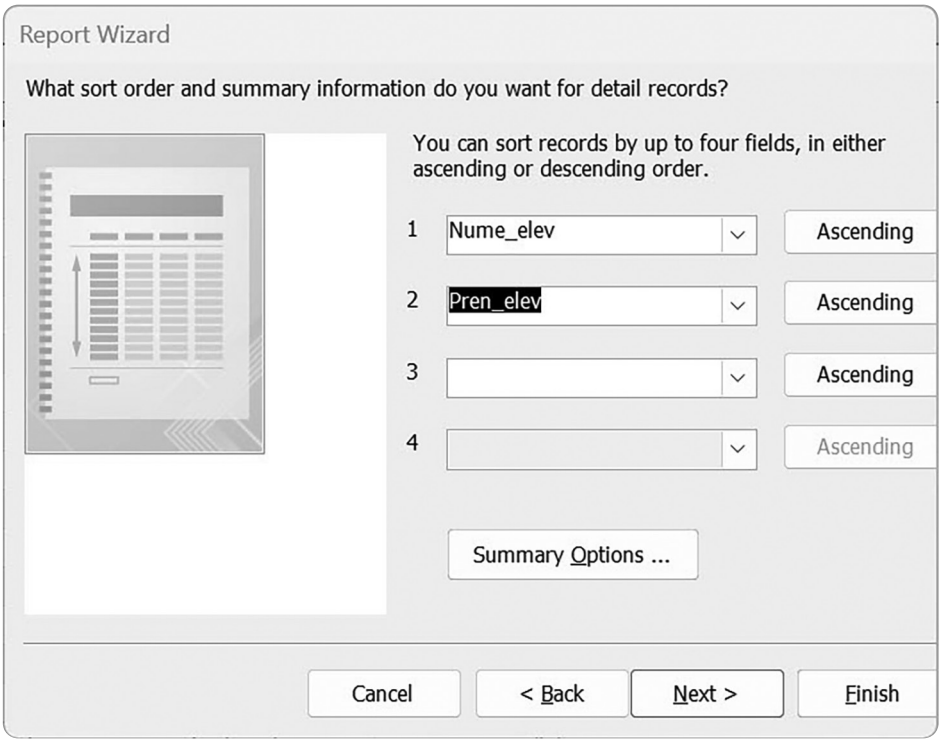


Рис. 4.37. Выбор полей для сортировки записей

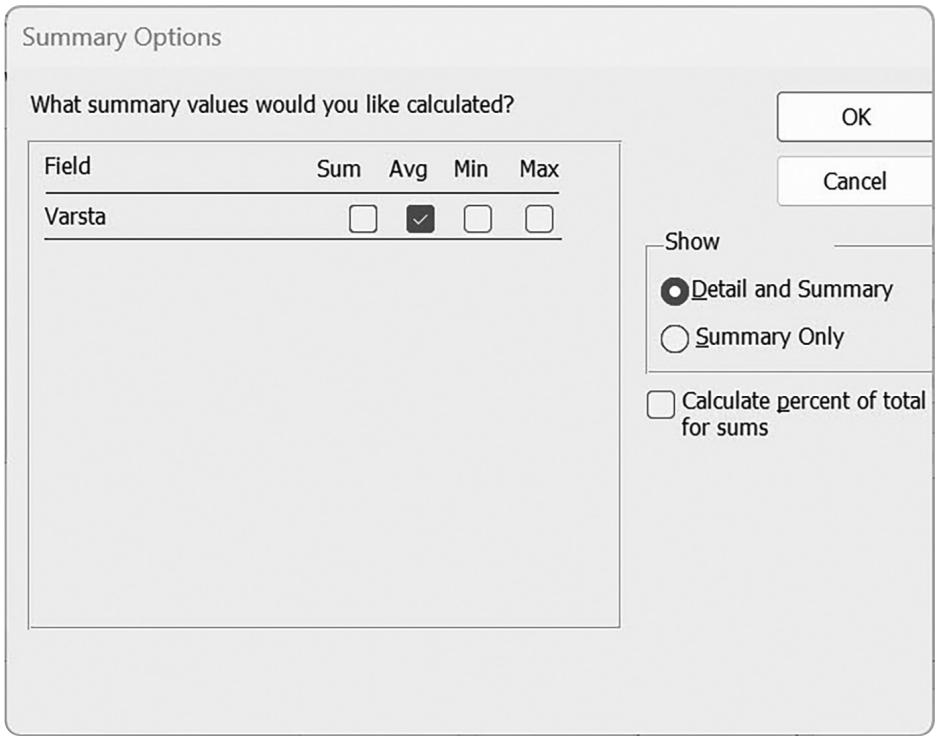


Рис. 4.38. Выбор опций для расчетов

5. В следующих трех окнах *Report Wizard* указываем способ размещения записей на странице, стиль и соответственно заголовок отчета. На *рисунке 4.39* представлена часть созданного отчета.

Для печати отчета, нажмем на кнопку *Print* на панели инструментов.

Замечание: Создание подотчетов осуществляется подобно созданию подчиненных формуляров.

Anul_de_studii	Nume_clasa	Nume_elev	Pren_elev	Varsta
10				
	A			
		Bacinschi	Sabina	17
		Belobrov	Andreea	17
		Brîncă	Carmen	17
		Bulgac	Ion	17

Рис. 4.39. Отчет *Возраст учеников*

Изменение отчетов в режиме *Design View*

Режим *Design View* для создания и изменения отчетов похож на режим *Design View* формуляров. Как и формуляры, отчеты состоят из пяти основных зон: *Report Header*, *Page Header*, *Detail*, *Page Footer* и *Report Foter*. В отчете могут также быть и зоны для групп данных. Например, на *рисунке 4.40* представлен отчет, создание которого было описано ранее, открытый в режиме конструктора. Обратите внимание на то, что, кроме 5 основных зон, появились зоны *Anul_de_studii Header*, *Nume_clasa Header*, *Anul_de_studii Footer* şi *Nume_clasa Footer*.

Действия с элементами управления отчета осуществляются подобно таковым из формуляра.

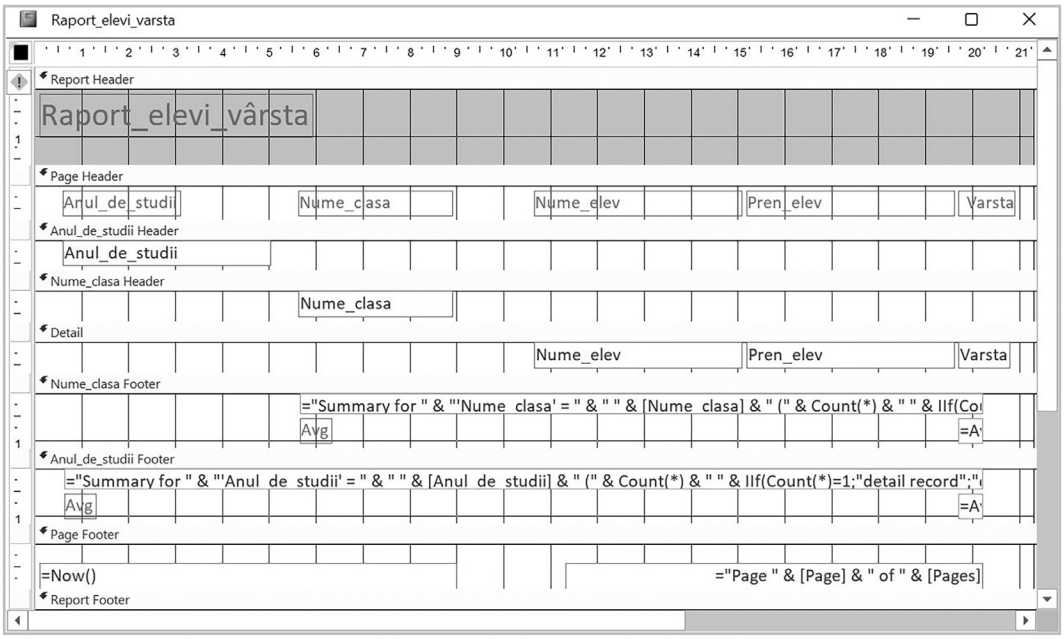


Рис. 4.40. Отчет *Возраст учеников*, отраженный в режиме *Design View* (режиме проектирования)

Создание диаграмм

Известно, что графическое представление данных воспринимается легче, особенно, если сравниваются числовые значения.

В системе управления базами данных диаграммы считаются специальным типом отчета. Можем создать диаграмму на базе таблицы или запроса.

Пример:

Создадим диаграмму, представляющую распределение по классам учеников базы данных *Liceu*, основываясь на данных запроса *Nr_elevi_clasa*.

1. Выберем запрос *Nr_elevi_clasa* (Количество учеников в классе). Нажимаем кнопку *Form Design* в меню *Create*. Появится меню *Form Design*. Убедимся, что опция *Use Control Wizard* в группе инструментов *Controls* включена. Выбираем инструмент *Chart* (Диаграмма) из той же группы инструментов, затем щелкаем в область формуляра, в которой хотим, чтобы отображалась диаграмма.
2. Появляется первое окно *Chart Wizard* (Мастер диаграмм), в котором выбираем поля *Anul_de_studii*, *Nume_clasa* и *CountOfCod_elev*.
3. В следующем окне *Chart Wizard* указываем тип диаграммы: *Column Chart* (Столбчатая диаграмма).
4. В третьем окне *Chart Wizard* (*рис. 4.41*) устанавливаем источник категорий данных (*Anul_de_studii*), источник ряда данных (*CountOfStudent_Code*) и легенду (*Nume_clasa*).

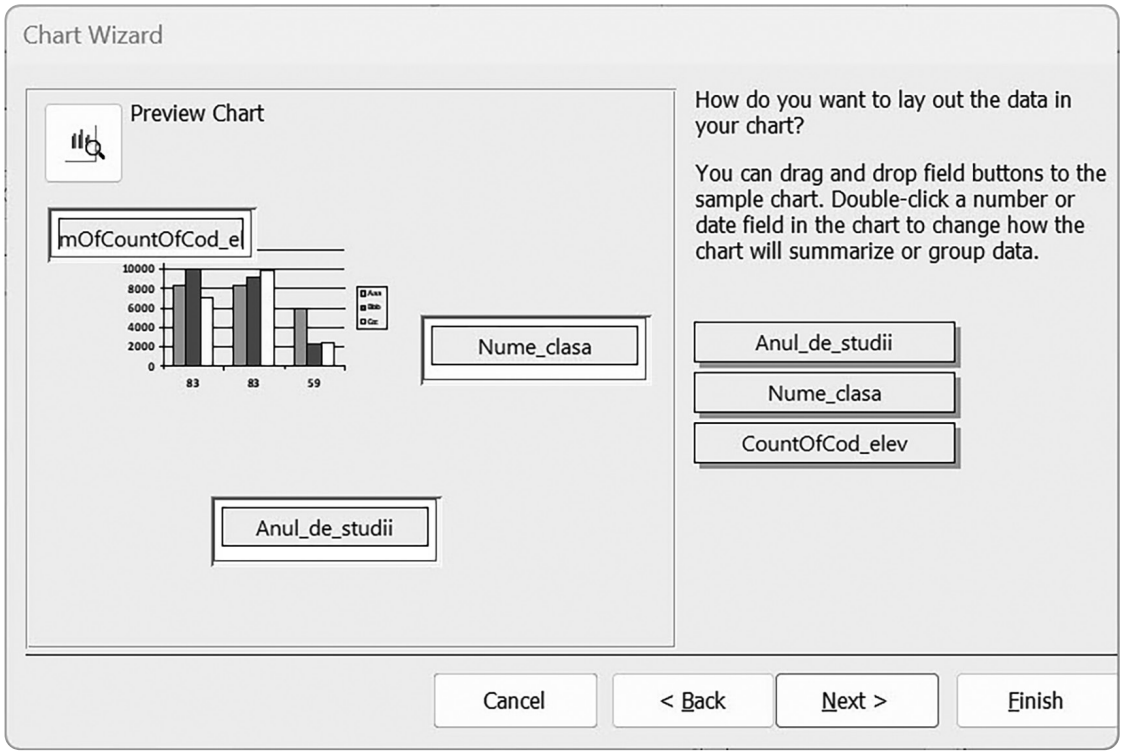


Рис. 4.41. Диалоговое окно *Chart Wizard* (Ассистент диаграмм)

5. В последнем окне *Chart Wizard* задаем имя диаграммы и уточняем, будет или нет выводиться легенда. Получим диаграмму как на *рисунке 4.42*

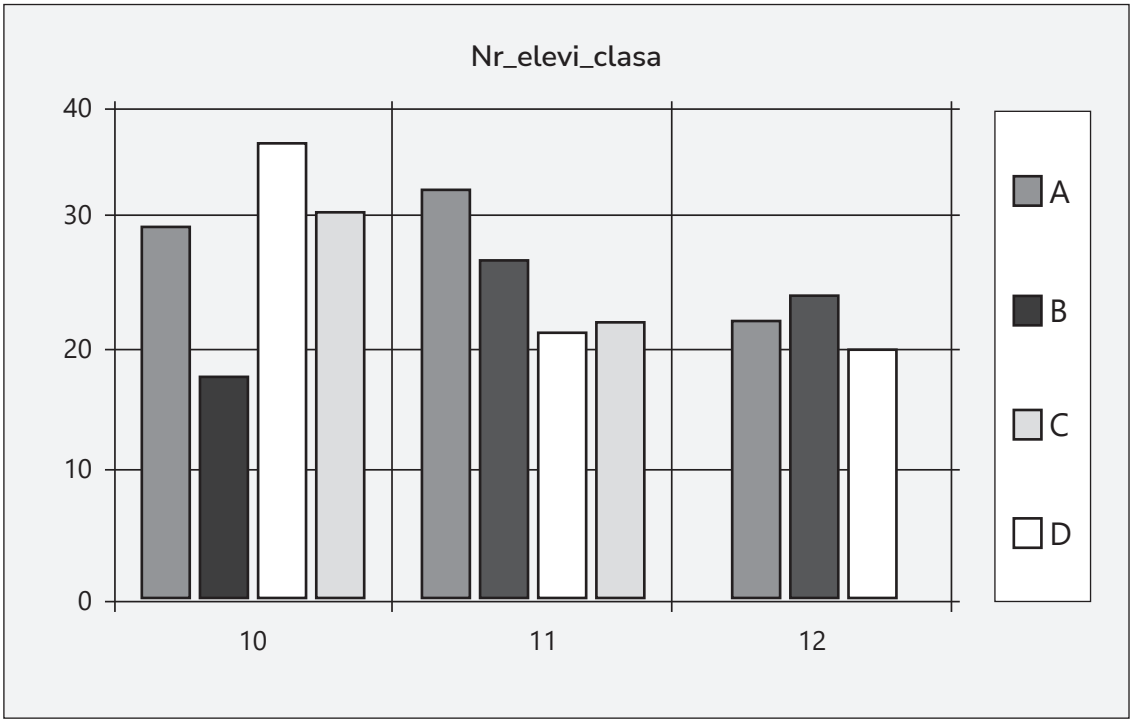


Рис. 4.42. Диаграмма, показывающая распределение учеников базы данных *Liceu* по классам

Вопросы и задания

1. **Систематизируйте свои знания!** Для каких целей применяются отчеты?
2. **Объясните!** Какие объекты базы данных могут служить источниками данных для отчетов?
3. **Анализируйте!** Какова роль диаграмм в базе данных?
4. **Примените!** Проанализируйте базу данных *Liceu*. Создайте отчет, в который выводится:
   a) среднее арифметическое часов за неделю для каждого года обучения;
   b) среднее арифметическое часов за неделю для каждого профиля;
   c) общее количество часов в неделю для каждого года обучения и в целом по лицею.
5. **Примените!** Проанализируйте базу данных *Liceu*. Создайте диаграмму, представляющую графически количество:
   a) учащихся каждого профиля;
   b) учащихся из каждой местности проживания;
   c) учителей каждого пола;
   d) часов за неделю, преподаваемых в каждом классе.
6. **Учитесь учиться!** Проанализируйте базу данных *Liceu*. Создайте диаграмму, представляющую графически количество учеников разного возраста.

4.15 Администрирование баз данных

Для того чтобы база данных нормально функционировала, она должна постоянно развиваться и поддерживаться. Обычно эти операции осуществляет администратор базы данных. В этом параграфе рассмотрим некоторые действия, необходимые для поддержки функциональности баз данных.

Сжатие и восстановление баз данных

- Со временем, в результате добавлений, обновлений, изменений различных объектов базы, *растет размер файла* базы данных и *уменьшается производительность*, то есть быстрота доступа и обработки данных.

Эти последствия объясняются не только ростом объема данных, но и:

- наличием в базе некоторых временных объектов (спрятанных от пользователя), даже если система больше в них не нуждается;
- присутствием в файле базы свободных пространств, получившихся в результате удаления некоторых объектов базы данных.

В таких случаях необходимо сжатие базы данных – операция по дефрагментации файла базы, то есть по удалению внутренних свободных пространств.

- Иногда возникают повреждения файла базы данных. Произойти это может по внешним причинам (например, отключение источника питания, поломки в сети) или в случае одновременного доступа к файлу базы большого числа пользователей.

В этом случае база нуждается в *восстановлении*.

Для **сжатия и восстановления базы данных**:

1. Закроем базу данных, оставив открытым приложение Access.

2. Выполним щелчок на *Compact and Repair Database* из меню *Database Tools*.

3. Появляется диалоговое окно *Database to Compact From*, в котором уточним название базы данных, которая будет сжата и восстановлена. Подтвердим действие нажатием кнопки *Compact*.

4. Появляется диалоговое окно *Compact Database Into*, в которое впишем имя, под которым сохранится сжатая и восстановленная база данных. Подтвердим действие нажатием кнопки *Save*.

Создание резервных копий

Access может восстановить поврежденную базу данных полностью или частично. В последнем случае потерянные данные можно восстановить из резервной копии базы данных.

Следовательно, администратор базы должен периодически создавать такие копии.

Это действие можно осуществить традиционным образом, скопировав файл базы в другую папку.

Обеспечение безопасности данных

В базе данных могут содержаться конфиденциальные данные, и в то же время доступ к ней по сети (возможно, и через Интернет) могут иметь многие пользователи. В этом случае предпринимаются действия по защите от несанкционированного доступа

к данным. Более того, различные пользователи могут иметь различные права доступа. Например, некоторые пользователи могут иметь право вводить информацию, другие – только просматривать определенную ее часть.

С этой целью каждому пользователю или группам пользователей предоставляются:

– уникальное имя для доступа;

– пароль;

– права доступа или владельца.

Самый простой способ защиты базы данных от несанкционированного доступа – это установление пароля, без которого базу невозможно открыть.

Для **создания пароля**:

1. Закроем базу данных, оставив открытым приложение Access.

2. Выполним *File* → *Open*. Выделим файл базы данных, а затем опцию *Open Exclusive* из раскрывающегося списка *Open*.

3. Выполним *File* → *Info* → *Encript with Password*. Появляется диалоговое окно *Set Database Password,* в котором устанавливаем и подтверждаем пароль.

Замечания:

1. Существуют и другие действия по администрированию баз данных с целью защиты данных. Например, данные базы можно **шифровать**, чтобы их невозможно было просматривать без **дешифрования**, с помощью текстовых редакторов или специальных программ.
2. Создание групп, паролей и прав доступа осуществляется из каскадного меню *Security* меню *Tools.*

Создаем, исследуем и упражняемся

На рисунке (*рис. 4.43*) представлены реляционные связи между таблицами базы данных «Спортивный клуб» (поле *Platit* логического типа).

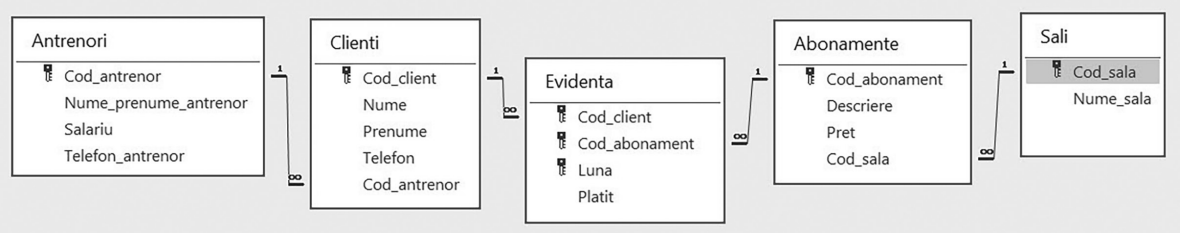


Рис. 4.43. Реляционные связи между таблицами базы данных «Спортивный клуб»

1. Создайте базу данных со структурой, показанной на рисунке, и заполните ее записями.

2. Объясните, почему поля **Cod_client**, **Cod_abonament** и **Luna** таблицы **Evidenta** формируют вместе первичный ключ этой таблицы.

3. Разработайте запрос, который вычисляет:

a) список клиентов, имя которых начинается с буквы A или C;

b) список тренеров, имена которых состоят не менее чем из 5 букв;

c) среднюю заработную плату тренеров;

d) список тренеров с заработной платой выше средней;
e) список кодов абонементов с максимальной ценой;
f) количество клиентов у каждого тренера;
g) имена тренеров с наибольшим количеством клиентов;
h) общая сумма денег, уплаченная каждым клиентом;
i) имена клиентов, заплативших наибольшую сумму (в общей сложности);
j) помещения, наиболее часто посещаемые клиентами.

4. Разработайте формуляры ввода и редактирования данных для каждой из таблиц базы данных «Спортивный клуб».

Создаем в команде

1. Разработайте реляционную модель базы данных с информацией:

a) о дневнике ученика;
b) о европейских странах;
c) об автомобилях;
d) о названиях книг;
e) о веб-страницах.

2. Создайте спроектированные таблицы базы данных.

3. Установите первичные ключи и связи между таблицами базы данных.

4. Заполните каждую таблицу не менее чем пятью записями.

5. Разработайте:

a) 4 запроса на выборку, один из которых будет с параметром;
b) 3 запроса на изменение записей;
c) 2 запроса на группировку и итоги;
d) перекрестный запрос;
e) запрос на исключение некоторых записей.

6. Создайте два формуляра для вставки/изменения записей в таблицах базы данных.

7. Создайте не менее двух отчетов с информацией из базы данных.

8. Создайте диаграмму, которая будет отражать данные из отчетов.

Раздел V

Модули по выбору

Чтобы в полной мере реализовать призвание каждого из вас, учебная программа по информатике предлагает вам возможность изучить модуль по вашему выбору:

– Расширенная обработка информации в базах данных.

– Экспериментальные методы в гуманитарных науках.

– Веб-программирование.

– Динамические структуры данных в PASCAL или C/C++.

Материалы соответствующих модулей рекомендуется изучать с использованием *метода перевернутого класса*. Напоминаем, что данный метод основан на самостоятельной работе учеников, при этом преподаватель выступает в роли консультанта и проводника. Чтобы облегчить каждому ученику выбор желаемого модуля, в этой главе содержится их краткое описание. Сами модули можно скачать с официального сайта Центра информационно-коммуникационных технологий в образовании (www.ctice.gov.md), раздел *Ресурсы*.

5.1 Расширенная обработка информации из баз данных в виде списков

1. Сортировка и выборка записей

В этом параграфе объясняется, как *сортировать и выбирать записи* базы данных, организованной в виде списка. Такие базы данных изучались в 9 классе. Для закрепления изученного материала и закрепления навыков сортировки и выбора записей из таких баз данных рекомендуем выполнить представленные в данном параграфе практические работы на компьютере.

2. Сводка данных

Изучив этот параграф, вы узнаете, как получать *сводки* или, другими словами, *итоги* по записям базы данных в виде списков. В частности, вы разовьете навыки группировки данных, получения вложенных структур группировки, итогов и промежуточных итогов и т.д. исходя из информации в базах данных в виде списков.

В случае приложений управления базами данных обобщение данных выполняется с помощью запросов на группировку и итоговых запросов, которые применяются для суммирования данных из полей, получения средних значений, минимальных или максимальных значений и т.д

3. Сводные таблицы

Очень часто для получения итоговых данных по базе данных в виде списков используются *сводные таблицы*, иногда их также называют *таблицами pivot*. Эти таблицы особенно полезны в случаях, когда списки данных содержат много, сотни или даже тысячи записей. Сводные таблицы аналогичны результатам, полученным с помощью перекрестных запросов (которые можно создавать в приложениях управления базами данных). Изучив этот параграф, вы узнаете об алгоритме создания сводных таблиц на основе списка данных. Отметим, что эти списки должны соответствовать следующим требованиям:

- не содержать пустых имен полей, то есть каждый столбец должен иметь имя;
- не иметь итоговых строк.

5.2 Экспериментальные методы в гуманитарных науках

1. Экспериментальная методология гуманитарных наук

В этом параграфе дается обзор *объектов изучения социологии*:

- человеческие группы и коллективы, например, сельское население, городское население; малые группы (ученики, преподаватели, спортсмены, экологи, подростки, молодежь, взрослые, пожилые люди и т.д.);
- социальные явления и процессы, например, общественное мнение, вероисповедание, традиции, образование, политика, коммуникации;
- социальные институты, такие как семья, школа, парламентаризм, государство, собственность и т.д.

Ниже приведены методы, приемы и процедуры планирования и проведения социологических исследований:

- наблюдение;
- эксперимент;
- анализ документов;
- индивидуальное или групповое интервью;
- опрос общественного мнения;
- ситуационный анализ (кейс-стади, *case study*).

Также в этом пункте указаны основные ИТ-инструменты, предназначенные для обработки социологических данных, собранных в рамках соответствующего исследования.

2. Переменные в социологических экспериментах

В этом параграфе объясняются переменные, которые появляются в социологическом эксперименте, например:

- мотивация учеников к обучению;
- благополучие учащихся в школе;
- квалификация педагогических кадров;
- наличие современных физических, химических, биологических лабораторий;
- мнения учащихся относительно изучения отдельных школьных предметов;
- результаты обучения, продемонстрированные учащимися.

Особое внимание уделяется вероятностным связям между переменными, которые выявляются в социологических исследованиях.

Внимание! Для обучения, развития и закрепления практических навыков использования цифровых средств в гуманитарных науках мы рекомендуем учащимся опираться в своем обучении на индивидуальную или групповую разработку одного из следующих проектов:

1) Проведение опроса среди обучающихся в учебном заведении по вопросам:
   a) уровень удовлетворенности качеством учебников;
   b) уровень удовлетворенности физическими условиями в школе;
   c) степень участия учащихся во внеклассной деятельности;
   d) отношение учащихся к возможным случаям списывания;
   e) деятельность ученического совета образовательного учреждения;
   f) профессии, которые выберут учащиеся после окончания школы;
   g) мнения учеников о действиях, которые следует предпринять для предотвращения случаев издевательства, насилия в школе (*bulling*), дискриминации, сексизма, преступности среди несовершеннолетних и т.д.;
   h) мнения учеников о действиях, которые следует предпринять для продвижения здорового образа жизни, профилактики ранней беременности, профилактики заболеваний, передающихся половым путем.
2) Проведение опроса среди молодежи в населенном пункте, где расположено учебное заведение относительно следующих проблем:
   a) качество услуг, предназначенных для молодежи, на данной территории;
   b) возможное место строительства досугового центра;
   c) возможное место строительства спортивного центра;
   d) намерения молодых людей внести финансовый вклад или способствовать через неоплачиваемый труд в развитие местности;
   e) мнения молодежи о действиях, которые следует предпринять для предотвращения случаев чрезмерного употребления алкоголя или запрещенных веществ.
3) Проведение опроса среди граждан по месту нахождения образовательного учреждения относительно:
   a) качества жизни;
   b) политических возможностей;
   c) отношения к некоторым событиям, имеющим международный резонанс;
   d) качества дорог;
   e) состояния окружающей среды;
   f) деятельности органов местного публичного управления и т.д.

Разработка проектов начнется в рамках изучения второго параграфа данного модуля и будет проводиться шаг за шагом в процессе изучения каждого из следующих параграфов.

3. Контрольные группы и переменные-паразиты

Изучив этот параграф, вы узнаете значение основных терминов, используемых в социологических исследованиях:

- контрольная группа;
- синхронный опыт;
- диахронический опыт;
- контрольная группа с артефактом;
- переменная-паразит.

Вы также будете тренироваться и развивать свои навыки определения контрольных групп и типа эксперимента в зависимости от специфики изучаемого социального явления.

4. Экспериментальные проекты и выбор субъектов

Изучив этот параграф, вы познакомитесь с типами экспериментальных проектов (типами экспериментальных дизайнов) – однофакторными и многофакторными – а также разовьете свои навыки в выборе предметов в соответствии со спецификой исследуемого социального явления.

5. Математическое описание первичной информации

Материалы настоящего параграфа составляют математическую основу количественного социологического исследования и обеспечивают связь между гуманитарными и реальными науками, предоставляя социологу релевантные, объективные и правдивые методы измерения интенсивности социальных явлений. После изучения этого параграфа, вы сможете сделать осознанный выбор типа шкалы измерения (номинальной, порядковой, интервальной, относительной) в зависимости от специфики изучаемого социального явления, и разовьете свои навыки проектирования этих шкал.

6. Сводка и численное описание данных

Целью этого параграфа является тренировка и развитие навыков обобщения и численной записи данных, используемых в количественных социологических исследованиях. Важно понимать, как использование числовых данных способствует получению основанных на фактических данных выводов относительно конкретного социального явления. В этом параграфе для каждого типа шкалы описываются индексы центральной тенденции и индексы дисперсии и на основе нескольких примеров иллюстрируется способ интерпретации их числовых значений.

7. Популяции и выборки

Изучив этот параграф, вы освоите методы отбора проб:

- эмпирический;
- квот;
- типовых единиц;
- вероятностный;
- случайного выбора;
- стратификации.

8. Анализ данных с использованием электронных таблиц

Целью данного параграфа является формирование и развитие навыков использования расширенных возможностей электронных таблиц для анализа данных в области гуманитарных наук:

- сбор данных;
- проверка данных;
- систематизация данных;
- группировка данных;
- расчет индексов центральной тенденции;
- построение гистограмм;
- интерпретация полученных результатов.

9. Программные продукты для социальных наук

Как правило, специализированные приложения, предназначенные для планирования и проведения социологических исследований и соответственно обработки собранных данных, являются коммерческими продуктами. Следовательно, их использование влечет за собой определенные расходы либо за их установку на школьных или ученических компьютерах, либо для приобретения подписки для их использования через Интернет. В то же время существуют и бесплатные программные продукты, но предлагаемые ими инструменты более скромные. Таким образом, ученики под руководством преподавателя выберут программный продукт, который будут использовать в процессе преподавания-обучения-оценивания, и, применяя систему помощи, изучат соответствующий инструмент.

Внимание! Разработанные учениками проекты будут представлены на сессиях коллегиальной оценки. Эти сессии будут включать электронные презентации со скриншотами, подтверждающими точность социологического исследования, выводы, полученные в результате этого исследования, ответы на вопросы и комментарии, высказанные присутствующими на этих сессиях.

5.3 Веб-программирование

1. Способы использования кодов JavaScript в HTML-документах

В этом параграфе содержится базовая информация о языке программирования JavaScript (используемом при разработке различных типов интерактивных веб-приложений) и о том, как использовать коды JavaScript в гипертекстовых документах.

Рассматриваются три случая:

- вставка кода JavaScript непосредственно в HTML-документ;
- импорт кодов JavaScript из отдельного файла (содержащего только код JavaScript);
- использование кодов JavaScript для указания атрибутов тегов HTML.

2. Простые данные, выражения и функции

В этом параграфе объясняется базовый синтаксис выражений JavaScript. Выражение JavaScript представляет собой допустимую комбинацию операторов и операндов. Изучаются следующие частные случаи выражений:

a) константы (числовые, включая константы объектов Math; символьные строки; логические; нулевая константа);
b) переменные;
c) массивы и их элементы;
d) вызовы функций (возвращаемого типа, в том числе математических функций – методов объекта Math);
e) арифметические выражения;
f) логические выражения;
g) правильные сочетания выражений a) – f) с помощью операторов и скобок.

В этом параграфе представлены различные типы операторов JavaScript:
- арифметические;
- присваивания;
- приращения;
- декремента;
- сравнения;
- логические.

3. Операции ввода-вывода

В этом параграфе вы изучите некоторые способы организации процесса ввода и чтения данных (или извлечения результатов) для веб-приложений:
- методом *write* объекта *document*;
- через текстовые поля;
- с помощью диалоговых окон.

4. Структуры данных: массивы; строки символов

Изучив этот параграф, вы узнаете, что в JavaScript массивы, а следовательно, и строки, являются *объектами* — структурами, состоящими из *свойств* (данных объекта) и *методов* (функций объекта).

Вы изучите методы обработки массивов (объединение массивов, удаление элементов, добавление элементов, изменение позиций элементов, упорядочивание элементов, извлечение последовательности из массива и т.д.) и строковые операции (определение длины строки, конкатенация строк, вызов символа по его индексу, поиск ASCII-кода символа, поиск символов по их коду, поиск позиции подстроки в строке, извлечение подстроки из строки, сравнение строк, разделение подстрок, преобразование букв строки, форматирование строк, преобразование строки в число и т.д.).

5. Контроль действий. События

В этом параграфе вы познакомитесь с понятием *событие* (в контексте программирования). События используются для того, чтобы пользователь мог взаимодействовать с веб-приложениями: нажатием клавиши, командной кнопки или кнопки мыши, перемещением мыши, наведением курсора на элемент веб-страницы, выбором поля формы, доступом к гиперссылкам и т.д.

6. Структуры управления

Структуры управления являются базовой особенностью языков программирования. В этом параграфе вы узнаете, как с помощью синтаксиса JavaScript можно создавать следующие структуры управления:

– условное выражение;
– структуры выбора с оператором *if ... else*;
– селектор;
– цикл с предусловием;
– цикл с постусловием;
– цикл с параметром;
– цикл *for... in*;
– цикл с параметрами (цикл *for*).

Кроме того, все вышеописанные параграфы модуля содержат:
– вопросы для самооценки (призваны способствовать осознанию и закреплению изученного);
– различные задания на интеграцию и применение новых знаний;
– более сложные задания, направленные на развитие базовых навыков написания кода JavaScript и разработки веб-приложений для решения реальных задач, часто встречающихся в повседневной жизни.

5.4 Динамические структуры данных в PASCAL

1. Динамические переменные. Тип ссылки

В этом параграфе содержится основная информация о классификации переменных компьютерной программы на статические и динамические. В нем объясняется, как ссылаться на динамические переменные и специальные типы данных, используемые для этой цели. При изучении этого параграфа важно понимать разницу между способами выделения и освобождения памяти для статических и динамических переменных, тренировать и развивать свои навыки создания и уничтожения динамических переменных.

2. Структуры данных

Изучив этот параграф, вы узнаете, как классифицируются структуры данных, используемые в компьютерных программах:
– неявные структуры и явные структуры;
– статические конструкции и динамические конструкции;
– однородные структуры и неоднородные структуры;
– рекурсивные структуры.

Этот параграф представляет собой краткое введение в структуры данных и направлен на понимание роли *структурной информации* в описании взаимосвязей, существующих между компонентами структурированных данных.

3. Односвязные списки

Изучив этот параграф, вы познакомитесь с составными частями динамических структур данных: полем данных и полем связи, научитесь строить односвязные списки с использованием метода итерации и метода рекурсии.

4. Обработка односвязных списков

Изучение этого параграфа потребует повышенного внимания, поскольку операции, часто используемые в случае односвязных списков, встречаются и при обработке других динамических структур данных:

- просмотр списка и обработка полезной информации, связанной с каждой ячейкой;
- поиск определенного элемента, идентифицированного по его значению;
- включение (вставка) элемента в определенное место списка;
- исключение (удаление) элемента из списка и т.д

Графические представления односвязных списков и выполняемых над ними операций, включенные в этот параграф, помогут вам интуитивно понять суть операций обработки данных, хранящихся в динамических структурах данных, и изменения самих структур.

5. Стеки

Под *стеком* мы понимаем односвязный список, обладающий тем свойством, что операции вставки и извлечения элементов выполняются только на одном его конце. Хотя стеки представляют собой частные случаи односвязных списков, они имеют большое практическое значение, поскольку используются как при разработке системных программ (управление памятью, реализация вызовов подпрограмм, рекурсия и т.д.), так и при разработке прикладных программ (анализ синтаксиса, моделирование экономических процессов, электронный документооборот и т.д.).

При изучении этого параграфа ученики сосредоточатся на использовании стеков для решения задач, часто встречающихся в повседневной работе экономистов, инженеров, программистов.

6. Очереди

Под очередью (по-английски *queue*) мы подразумеваем односвязный список, в котором все вставки выполняются на одном конце, а извлечения – на другом. Как и стеки, очереди являются частным случаем односвязных списков и используются, в частности, для моделирования процессов обслуживания клиентов, управления движением транспортных средств и планирования работ.

7. Бинарные деревья

По сравнению с однонаправленными списками двоичные деревья представляют собой более сложные структуры, и их изучение требует полного владения как итеративными, так и рекурсивными методами. Поэтому перед изучением этого параграфа ученикам следует освежить свои знания, уделив особое внимание специфике рекурсивных определений и способам управления памятью в случае рекурсивных вызовов.

Графические представления, включенные в этот параграф, помогут учащимся лучше понять, как организовать данные в форме двоичных деревьев, а также значение следующих терминов:

- узел, корневой узел, конечный узел, неконечный узел;
- потомок, левый потомок, правый потомок;
- поддерево, левое поддерево, правое поддерево;
- уровень узла, высота дерева.

8. Обход бинарных деревьев

Обход бинарных деревьев подразумевает посещение их узлов в определенном порядке. Изучив этот параграф, вы сможете разрабатывать подпрограммы для обхода бинарных деревьев в:

- прямом порядке КЛП (корень → левое поддерево → правое поддерево);
- центрированном порядке ЛКП (левое поддерево → корень → правое поддерево);
- обратном порядке ЛПК (левое поддерево → правое поддерево → корень).

Алгоритмы обхода двоичного дерева имеют широкое практическое применение, особенно при оценке выражений, написанных на языках программирования высокого уровня.

Рекомендуемые виды учебной деятельности

Для обучения и развития практических навыков использования динамических структур данных мы рекомендуем:

Задания:

- интуитивное знакомство (через рисунок) с методами динамического распределения памяти;
- обоснование необходимости использования динамических структур данных;
- подчеркивание различий между неявными и явными структурами данных, между однородными и неоднородными структурами данных, между статическими и динамическими структурами данных;
- выбор задач, решение которых требует использования структур данных, предложенных в исследовании;
- создание, использование и уничтожение динамических переменных;
- разработка программ, использующих динамические переменные;
- объяснение того, как выделять оперативную память при использовании статических и динамических переменных;
- хранение и обработка данных с использованием списков, стеков, очередей и двоичных деревьев.

Исследование практических примеров:

- поиск информации в списках, очередях, стеках и двоичных деревьях;
- обход списков, стеков, очередей и двоичных деревьев;
- вставка и удаление данных из списков, стеков, очередей и двоичных деревьев;
- области использования динамических структур данных.

Проекты:

- обработка списков кандидатов на поступление в лицей;
- обработка списков, сформированных из отдельных слов текста;
- моделирование процесса въезда-выезда вагонов в/из железнодорожного депо;
- обработка списков сотрудников предприятия;
- синтаксический анализ арифметических выражений;
- моделирование очереди ожидания самолетов, запрашивающих посадку в аэропорту;
- создание и обработка бинарных деревьев, представляющих участников спортивных турниров «на выбывание»;
- вычисление арифметических выражений, представленных двоичными деревьями.

Программы, исследования и проекты, разработанные учащимися, будут представлены на сессиях взаимной оценки. Эти занятия будут включать в себя электронные презентации со скриншотами, подтверждающими синтаксическую и логическую корректность разработанных программ, результаты, отображаемые этими программами, практические компьютерные демонстрации.

5.5 Динамические структуры данных в C/C++

1. Указатели и ссылки

Обычно доступ к значениям переменной осуществляется путем указания ее имени. Однако в программах, имеющих реальное практическое значение, возникает необходимость использовать не только текущие значения переменных, объявленных в программе, но и адреса ячеек памяти, назначенные этим соответствующим переменным.

Для обработки адресов памяти языки C/C++ предлагают программисту специальные типы данных, называемые *указателями*. Набор значений переменной-указателя состоит из адресов памяти. Напоминаем, что в английском языке слово *pointer* означает индикатор, указатель.

Использование переменных-указателей особенно полезно в случаях, когда программист хочет обрабатывать в программах на языке C/C++ не только текущие значения определенных переменных, но и связи, существующие между этими переменными.

2. Структуры данных

В программах на языке C/C++ структуры данных содержат не только значения определенных переменных, но и отношения (связи), существующие между ними. Простейшей структурой данных является массив, состоящий из компонентов одного типа. Отношения между компонентами массива представлены их упорядочением и нумерацией (строками и столбцами в случае двумерных массивов).

В случае более сложных структур данных соответствующие компоненты могут быть самых разных типов, а связи между ними обозначаются с помощью указателей. Более того, компоненты такой структуры данных могут создаваться и удаляться во время выполнения программ, в которых они были объявлены. Структуры данных, компоненты которых также содержат и указатели называются *динамическими структурами данных.*

3. Односвязные списки

Каждый компонент этой структуры данных содержит два поля: поле данных, которое необходимо обработать, и поле адреса. Очевидно, что поле адреса имеет тип *указателя*. Этот указатель указывает на следующий компонент в списке, и в случае последнего компонента списка он имеет значение NULL.

Изучив этот параграф, вы сформируете и разовьете свои навыки обработки односторонних списков:

- инициализация списка;
- прохождение списка;
- включение компонентов в список;
- удаление компонентов из списка.

4. Динамически распределяемые очередь и стек

Очереди и стеки являются частными случаями списков. В отличие от обычных списков, в которых вставка и удаление элементов может осуществляться в любом месте списка, в случае очередей и стеков соответствующие операции выполняются только на концах: в начале и/или конце списка.

В случае очередей новые компоненты вставляются в один конец списка, а удаляются – в противоположном. По этим причинам очереди также называют структурами данных типа *First In – First Out* (первым пришел – первым ушел). В случае стеков новые компоненты вставляются и извлекаются только с одного из концов. По этим причинам стеки также называют структурами данных типа *Last In – First Out* (последним пришел – первым ушел).

5. Двоичные деревья поиска

Компоненты бинарных деревьев формируются из трех полей:

- поле данных, подлежащее обработке;
- поле, содержащее адрес левого поддерева;
- поле, содержащее адрес правого поддерева.

Если одно из полей адреса содержит значение NULL, соответствующее поддерево отсутствует. Если оба поля адреса содержат значения NULL, то соответствующий компонент является конечным, имеющим аллегорическое название «лист».

Изучая этот параграф, вы сформируете и разовьете свои навыки обработки двоичных деревьев:

- инициализация дерева;
- вставка и удаление компонентов;
- обход деревьев в прямом, центрированном и обратном порядке.

Рекомендуемые учебные мероприятия

Для обучения и развития практических навыков использования динамических структур данных мы рекомендуем учащимся учебные задания, перечисленные в конце предыдущего параграфа.

VI
Глава

Дополнительные материалы

Содержание этой главы предназначено для углубленного изучения следующих тем информатики:

Методы программирования:	Метод возврата. Метод *Разделяй и властвуй*.
Моделирование и численные методы:	Метод Ньютона (Newton). Метод трапеций.
Динамические структуры данных:	Тип данных *pointer* (PASCAL).

Соответствующие темы, предусмотренные учебным планом, имеют статус факультативных и предлагаются учащимся, которые после окончания лицея намерены продолжить обучение в области информатики и информационных технологий.

Как и в случае с факультативными модулями, темы этой главы рекомендуется изучать с использованием метода *перевернутого класса*. Напоминаем, что данный метод основан на самостоятельной работе учеников, при этом преподаватель выступает в роли консультанта и наставника.

Чтобы облегчить каждому ученику выбор желаемых тем, ниже представлено их краткое описание. Все дополнительные материалы можно скачать с официального сайта Центра информационно-коммуникационных технологий в образовании (www.ctice.gov.md), раздел *Resurse*.

1. Метод возврата (по-английски *backtracking*)

Идея этого метода программирования очень проста:

- решение задачи представляется вектором (массивом);
- компоненты вектора выбираются пошагово из упорядоченных наборов, по одному набору для каждого из компонентов;
- элемент из текущего набора включается в качестве компонента в конструируемый вектор только в том случае, если он удовлетворяет определенным условиям продолжения;
- если таких элементов больше нет в текущем наборе, выполняется возврат к предыдущему варианту выбора.

Отметим, что именно возврат к предыдущему варианту выбора и дал название изучаемому методу.

Хотя метод возврата не гарантирует получения оптимального решения, он широко применяется при разработке программ для графической обработки, планирования маршрутов, распределения ресурсов и т.д.

2. Метод *Разделяй и властвуй*

Метод «*Разделяй и властвуй*» (лат. *divide et impera*) — общий метод разработки алгоритмов, который включает в себя:

- многократное разделение большой задачи на две или более подзадач одного и того же типа, но разного размера;
- решение подзадач напрямую, если их размер позволяет это, или разбиение их на другие подзадачи еще меньшего размера;
- объединение решений решенных подзадач для получения решения исходной задачи.

Как правило, этот метод очень часто используется для бинарного поиска, сортировки элементов, оценки сложности и планирования работ. Программы, разработанные на основе метода *Разделяй и властвуй*, просты, а время выполнения относительно невелико.

3. Метод Ньютона

Метод Ньютона, как и метод хорд, применяется для решения алгебраических и трансцендентных уравнений. Этот метод быстрее метода хорд, но его можно применять только для некоторых уравнений типа $f(x) = 0$.

Таким образом, на интервале $[a, b]$, на котором ищется соответствующее решение, функция $f(x)$ должна соответствовать следующим условиям:

– быть непрерывной;

– величины $f(a)$ и $f(b)$ должны иметь разные знаки;

– $f(x)$ имеет первую и вторую производные, обе непрерывные, с постоянным знаком.

Рекуррентная формула для вычисления приближений решения содержит как функцию f(x), так и ее первую производную.

Интуитивно рассматриваемый метод заключается в последовательном проведении касательных к графику функции $f(x)$ и определении точек пересечения соответствующих касательных с осью $0x$.

Итеративное построение касательных повторяется до тех пор, пока расстояние между двумя последовательными точками пересечения не станет меньше точности, с которой необходимо вычислить решение уравнения.

4. Метод трапеций

Метод трапеций – это метод численного интегрирования, используемый для приближенного вычисления значения определенного интеграла. Метод основан на делении интервала интегрирования на более мелкие подынтервалы, обычно равной длины, и аппроксимации площади под графиком функции $f(x)$ путем вычисления площадей трапеций, вписанных под ее график.

Интуитивно метод трапеций состоит в следующем:

- интервал интегрирования $[a, b]$ разбивается на равные подынтервалы;
- для каждого подынтервала строится трапеция, образованная осью $0x$, прямыми, перпендикулярными оси $0x$, проходящими через концы соответствующего подынтервала, и линейной аппроксимацией функции $f(x)$ на этом подынтервале;
- суммирование площадей построенных трапеций.

Поскольку трапециевидное приближение точнее прямоугольного, метод трапеций требует меньше вычислений, чем прямоугольный метод. Следовательно, при одинаковой точности вычисления приближенных значений определенных интегралов метод трапеций оказывается более быстрым.

5. Тип данных *pointer* (указатель)

Эта тема поможет понять, как программисты могут эффективно управлять памятью компьютера, имея полный контроль над созданием, уничтожением и обработкой динамических переменных, а также выделением и освобождением оперативной памяти, используемой такими переменными.

Тип данных *pointer*, иногда под другими названиями, встречается практически во всех современных языках программирования, являясь часто используемым инструментом при разработке программ, предназначенных для обработки сложных структур данных.

Формируя, развивая и закрепляя навыки работы с указателями, учащиеся получают прочную основу для изучения современных языков программирования, предназначенных для разработки системных программных продуктов, мультимедийных приложений и компьютерных игр.

Tiparul executat la
Casa Editorial-Poligrafică „Bons Offices”
str. Feredeului, 4/6