4

Основни структури за управление на изчислителния процес

В тази глава ще разгледаме управляващите оператори в езика. Ще ги наричаме само оператори.

4.1. Оператор за присвояване

Това е един от най-важните оператори на езика. Вече многократно го използвахме, а също в глава 2 описахме неговите синтаксис и семантика. В тази глава ще го разгледаме по-подробно. Ще напомним неговите синтаксис и семантика.

Синтаксис

<променлива> = <израз>;

където

- <променлива> е идентификатор, дефиниран вече като променлива,
- <израз> е израз от тип, съвместим с типа на <променлива>.

Семантика

Намира се стойността на <израз>. Ако тя е от тип, различен от типа на <променлива>, конвертира се, ако е възможно, до него и се записва в именуваната с <променлива> памет.

Забележки:

• Ако <променлива> е от тип bool, <израз> може да бъде от тип bool или от който и да е числов тип.

• Ако <променлива> е от тип double, всички числови типове, а също типът bool, могат да са типове на <израз>.

• Ако <променлива> е от тип float, типовете float, short, unsiged short и bool, могат да са типове на <израз>. Ако <израз> е от тип int, unsigned int или double, присвояването може да се извърши със загуба на точност. Компилаторът предупреждава за това.

• Ако <променлива> е от тип int, типовете int, long int, short int и bool, могат да са типове на <израз>. В този случай ако <израз> е от тип double или float, дробната част на стойността на <израз> ще бъде отрязана и ако полученото цяло е извън множеството от стойности на типа int, ще се получи случаен резултат. Компилаторът издава предупреждение за това.

• Ако <променлива> е от тип short int, типовете short int и bool, могат да са типове на <израз>. В противен случай се извършват преобразувания, които водят до загуба на точност или даже до случайни резултати. Много компилатори не предупреждават за това.

Ще отбележим, че в рамките на дефиницията на една функция не са възможни две дефиниции на една и съща променлива, но на една и съща променлива може да ѝ бъдат присвоявани многократно различни стойности.

Пример: Не са допустими

```
…
double a = 1.5;
…
double a = a + 5.1;
…
```

но са допустими присвояванията:

```
…
double a = 1.5;
…
a = a + 34.5;
…
a = 0.5 + sin(a);
…
```

В езика C++ са въведени някои съкратени форми на оператора за присвояване. Например, заради честото използване на оператора:

a = a + 1;

той съкратено се означава с

a++;

Въведено е също и съкращението a-- на оператора a = a-1;

В същност ++ и -- са реализирани като постфиксни унарни оператори увеличаващи съответно намаляващи аргумента си с 1. Приоритетът им е един и същ с този на унарните оператори +, - и !.

Забележка: От оператора ++, за добавяне на 1, идва името на езика C++ - вариант на езика C, към който са добавени много подобрения и нови черти.

Допълнения: 1. Операторът за присвояване

<променлива> = <израз>;

съдържа оператора =, реализиран като дясноасоциативен инфиксен аритметично-логически оператор с приоритет по-нисък от този на дизюнкцията ||. Това позволява на <променлива> = <израз>; да се гледа като на израз от тип – типа на <променлива> и стойност – стойността на <израз>, ако е <израз> от типа на <променлива> или стойността на <израз>, преобразувана до типа на <променлива>, ако <променлива> и <израз> не са от един и същ тип.

Пример: Програмата

```
#include <iostream.h>
int main()
{int a;
 double  b = 3.2342;
 cout << (a = b) << "\n";
 return 0;
}
```

е допустима. Резултът от изпълнението й е 3, като компилаторът издава предупреждение за загуба на информация при преобразуването от тип double в тип int. Изразът a = b е от тип int и има стойност 3. Ограждането му в скоби е необходимо заради по-ниския приоритет на = от този на <<.

2. Ако е в сила дефиницията:

```
int x, y, z;
```

допустим е операторът:

```
x = y = 5;
```

Тъй като = е дясноасоциативен, отначало променливата y се свързва с 5, което е и стойността на израза y = 5. След това x се свързва с 5, което пък е стойността на целия израз.

Някои компилатори издават предупреждение при тази употреба на оператора =. Затова не препоръчваме да се използва = като аритметичен оператор.

Задачи върху оператора за присвояване

Задача 12. Нека са дадени дефинициите

```
double x, y, z;
int m, n, p;
```

Кои от следните редици от символи:

```
а) -x = y;        б) x = -y;        в) m + n = p;
г) p = x + y;     д) z = x - y      е) z = m + n;
ж) sin(0) = 0;    з) x  n + sin(z)  к) 4 = sin(p + 5)
```

са оператори за присвояване, кои не са и защо?

В случаите а), в), ж) и к) редиците не са оператори за присвояване, тъй като се прави опит на израз да се присвои израз. В случай г) редицата от символи е оператор за присвояване, но тъй като на цяла променлива се присвоява стойността на реален израз, компилаторът ще направи предупреждение за загуба на точност, а в случай з) е пропуснат знакът за присвояване. В случаите б), д) и е) редиците са оператори за присвояване.

Задача 13. Да се напише програма, която въвежда стойности на реалните променливи a и b, след което разменя и извежда стойностите им (Например, ако a = 5.6, a b = -3.4, след изпълнението на програмата a да става -3.4, a b да получава стойността 5.6).

Програма Zad13.cpp решава задачата.

```
//  Program Zad13.cpp
#include <iostream.h>
```

```
int main()
{cout << "a= ";
 double a;
 cin >> a;
 cout << "b= ";
 double b;
 cin >> b;
 double x; // помощна променлива
 x = a;
 a = b;
 b = x;
 cout << "a= " << a << "\n";
 cout << "b =" << b << "\n";
 return 0;
}
```

В тази програма променливите a и b съдържат входа и изхода от работата на програмата. Използвана е помощна (работна) променлива x, която съхранява първоначалната стойност на променливата a.

Задача 14. Да се напише програма, която въвежда положително трицифрено число и извежда на отделни редове цифрите на стотиците, на десетиците и на единиците на числото.

Програма Zad14.cpp решава задачата.

```
// Program Zad14.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{ cout << "a - three-digit, integer and positive? ";
  int a ;
  cin >> a;
  short s, d, e;
  s = a/100;
  d = a/10%10;
  e = a%10;
  cout << setw(10) << "stotici: " << setw(5) << s << "\n";
```

```
  cout << setw(10) << "desetici:" << setw(5) << d << "\n";
  cout << setw(10) << "edinici: " << setw(5) << e << "\n";
  return 0;
 }
```

Задача 15. На цялата променлива b да се присвои първата цифра на дробната част на положителното реално число x (Например ако x = 52.467, b = 4).

Програма Zad15.cpp решава задачата

```
// Program Zad15.cpp
#include <iostream.h>
#include <math.h>
int main()
{ cout << "x>0? ";
  double x;
  cin >> x;
  int i = floor(x*10);
  int b = i%10;
  cout << x << "\n";
  cout << b << "\n";
  return 0;
}
```

Задача 16. Да се напише програма, която извежда 1, ако в записа на положителното четирицифрено число a, всички цифри са различни и 0 - в противен случай.

Програма Zad16.cpp решава задачата.

```
// Program Zad16.cpp
#include <iostream.h>
int main()
{ cout << "a - four-digit, integer and positive? ";
  int a ;
  cin >> a;
  short h, s, d, e;
```

```
    h = a/1000;
    s = a/100%10;
    d = a/10%10;
    e = a%10;
    cout << (h != s && h != d && h != e &&
             s != d && s != e && d != e)
         << "\n";
    return 0;
}
```

4.2. Празен оператор

Това е най-простия оператор на C++. Описанието му е дадено на Фиг. 4.1.

Празен оператор

Синтаксис

;

Операторът не съдържа никакви символи. Завършва със знака ;.

Семантика

Не извършва никакви действия. Използва се когато синтаксисът на някакъв оператор изисква присъствието на поне един оператор, а логиката на програмата не изисква такъв.

Фиг. 4.1 Празен оператор

Забележка: Излишни празни оператори не предизвикват грешка при компилация. Например, редицата от оператори

```
a = 150;;;
b = 50;;;;
c = a + b;;
```

се състои от: оператора за присвояване a = 150;, 2 празни оператора, оператора за присвояване b = 50;, 3 празни оператора, оператора за присвояване c = a + b; и 1 празен оператор и е напълно допустим програмен фрагмент.

Други примери ще дадем по-късно.

4.3. Блок

Често синтаксисът на някакъв оператор на езика изисква използването на един оператор, а логиката на задачата – редица от оператори. В този случай се налага оформянето на блок (Фиг. 4.2).

Блок

Синтаксис

{<оператор$_1$>
 <оператор$_2$>
  . . .
 <оператор$_n$>
}

Семантика

Обединява няколко оператора в един, наречен блок. Може да бъде поставен навсякъде, където по синтаксис стои оператор.

Дефинициите в блока, се отнасят само за него, т.е. не могат да се използват извън него.

Фиг. 4.2 Блок

Пример: Операторът

```
{cout << “a= “;
 double a;
 cin >> a;
 cout << “b= “;
 double b;
 cin >> b;
 double c = (a+b)/2;
 cout << “average{a, b} = “ << c << “\n”;
}
```

е блок. Опитът за използване на променливите a, b и c след блока, предизвиква грешка.

Препоръка: Двойката фигурни скоби, отварящи и затварящи блока да се поставят една под друга.

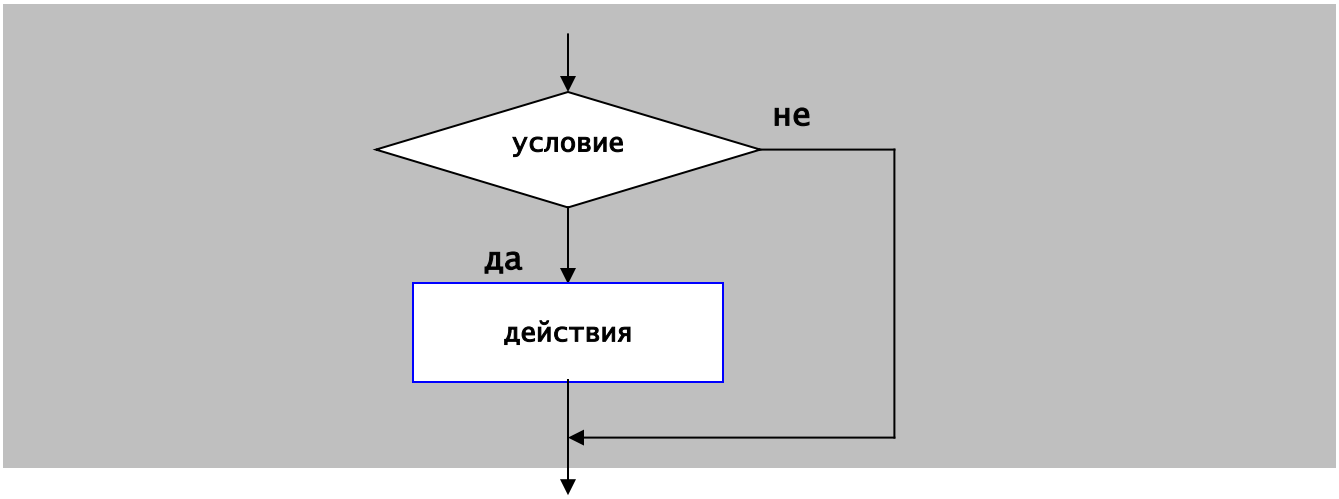
Забележка: За разлика от другите оператори, блокът не завършва със знака ;.

4.4. Условни оператори

Чрез тези оператори се реализират разклоняващи се изчислителни процеси. Оператор, който дава възможност да се изпълни (или не) един или друг оператор в зависимост от някакво условие, се нарича **условен**. Ще разгледаме следните условни оператори: if, if/else и switch.

4.4.1. Условен оператор if

Чрез този условен оператор се реализира разклоняващ се изчислителен процес от вид, илюстриран на Фиг. 4.3.



Фиг. 4.3 Разклоняващ се изчислетелен процес

Ако указано условие е в сила, изпълняват се определени действия, а ако не – тези действия се прескачат. И в двата случая след това се изпълняват общи действия.

Условието се задава чрез някакъв булев израз, а действията – чрез оператор. Фиг. 4.4 описва подробно синтаксиса и семантиката на този оператор.

Оператор if

Синтаксис

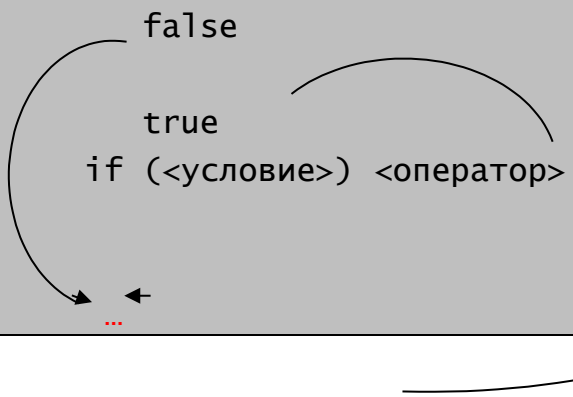
if (<условие>) <оператор>

където

- if (ако) е запазена дума;
- <условие> е булев израз;
- <оператор> е произволен оператор.

Семантика

Пресмята се стойността на булевия израз, представящ условието. Ако резултатът е true, изпълнява се <оператор>. В противен случай <оператор> не се изпълнява, т.е.



Фиг. 4.4 Условен оператор if

Забележки:

1. Булевият израз, определящ <условие>, трябва да бъде напълно определен. Огражда се в кръгли скоби.
2. Операторът след условието е точно един. Ако е необходимо няколко оператора да се изпълнят, трябва да се обединят в блок.

Задача 17. Да се напише програма, която намира най-малкото от три дадени реални числа.

Ще реализираме следните стъпки:

а) Въвеждане на стойности на реалните променливи a, b и c.

б) Инициализиране на работна реална променлива min, която ще играе и ролята на изходна променлива, със стойността на a.

в) Сравняване на b с min. Ако стойността на b е по-малка от запомнения в min текущ минимум, запомня се b в min. В противен случай, min не се променя. Така min съдържа min{a, b}.

г) Сравняване на c с min. Ако стойността на c е по-малка от запомнения в min текущ минимум, c се запомня в min. В противен случай, min не се променя. Така min съдържа min{a, b, c}.

д) Извеждане на резултата – стойността на min.

Програма Zad17.cpp реализира този алгоритъм.

```
// Program Zad17.cpp
#include <iostream.h>
int main()
{cout << "a= ";
 double a;
 cin >> a; // въвеждане на стойност на a
 cout << "b= ";
 double b;
 cin >> b; // въвеждане на стойност на b
 cout << "c= ";
 double c;
 cin >> c; // въвеждане на стойност на c
 double min = a;
 if (b < min) min = b;
 if (c < min) min = c;
 cout << "min{" << a << ", " << b << ", "
      << c << "}= " << min << "\n";
 return 0;
}
```

Ако вместо очаквано реално число, при въвеждане на стойности за променливите a, b и c, се въведе произволен низ, не представляващ число, буферът на клавиатурата, свързан със cin ще изпадне в състояние fail, а обектът cin ще има стойност false. Добрият стил за програмиране изисква в такъв случай програмата да прекъсне изпълнението си с подходящо съобщение за грешка. Програмата от задача 18 реализира този стил.

Задача 18. Да се напише програма, която намира най-малкото от три дадени реални числа. Програмата да извършва проверка за валидност на входните данни.

```
// Program Zad18.cpp
#include <iostream.h>
int main()
{cout << "a= ";
 double a;
 cin >> a;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 cout << "b= ";
 double b;
 cin >> b;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 cout << "c= ";
 double c;
 cin >> c;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 // въведени са валидни стойности за променливите a, b и c
 double min = a;
 if (b < min) min = b;
 if (c < min) min = c;
 cout << "min{" << a << ", " << b << ", "
      << c << "}= " << min << "\n";
 return 0;
}
```

Операторът return предизвиква преустановяване работата на програмата, а указаната в него стойност 1 сигнализира, че е възникнала грешка.

Задача 19. да се сортира във възходящ ред редица от три реални числа, запомнени в променливите a, b и c.

Ще реализираме следните стъпки:

а) Въвеждане на стойности за a, b и c.

б) Сравняване на стойностите на a и b. Ако е в сила релацията b<a, извършва се размяна на стойностите на a и b. В противен случай – размяната не се извършва.

в) Сравняване на стойностите на a и c. Ако е в сила релацията c<a, извършва се размяна на стойностите на a и c. В противен случай – размяната не се извършва. След тези действия, променливата a съдържа най-малката стойност на редицата.

г) Сравняване на стойностите на b и c. Ако е в сила релацията c<b, извършва се размяна на стойностите на b и c. В противен случай – размяната не се извършва.

д) Извеждане на стойностите на a, b, и c.

Програма Zad19.cpp реализира това описание.

```
// Program Zad19.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{ cout << "a= ";
  double a;
  cin >> a;
  if (!cin)
  {cout << "Error, Bad input! \n";
   return 1;
  } // въведена е валидна стойност на a
  cout << "b= ";
  double b;
  cin >> b;
  if (!cin)
  {cout << "Error, Bad input! \n";
   return 1;
  } // въведена е валидна стойност на b
```

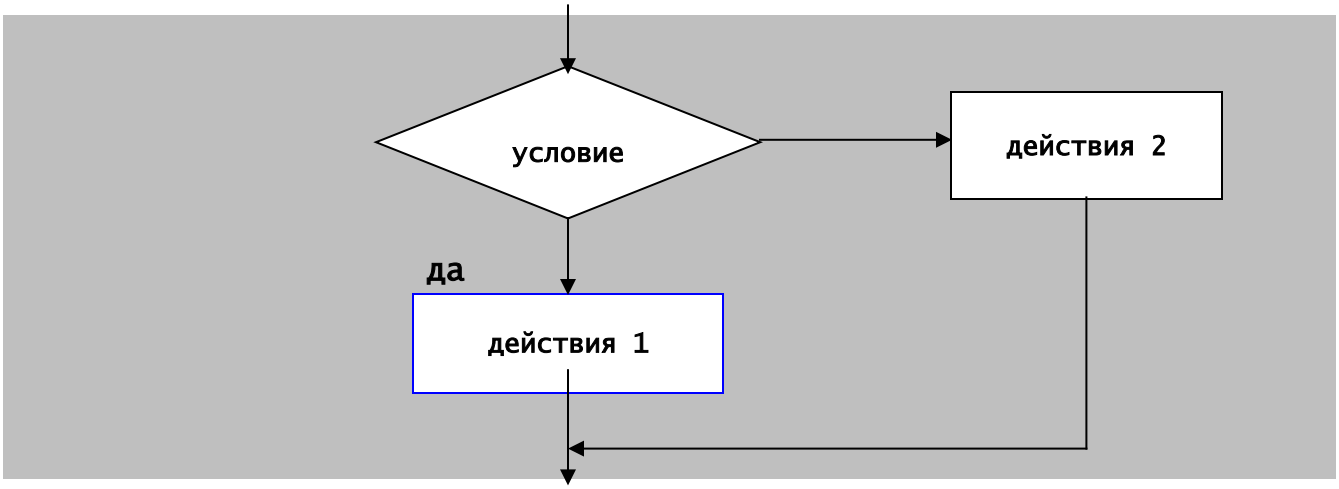
```
    cout << "c= ";
    double c;
    cin >> c;
    if (!cin)
    {cout << "Error. Bad input! \n";
     return 1;
    } // въведена е валидна стойност на c
    if (b < a) {double x = a; a = b; b = x;}
    if (c < a) {double x = c; c = a; a = x;}
    if (c < b) {double x = c; c = b; b = x;}
    cout << setprecision(2) << setiosflags(ios :: fixed);
    cout << setw(10) << a
         << setw(10) << b
         << setw(10) << c << "\n";
    return 0;
}
```

Забележка: Променливата x е видима (може да се използва) само в блоковете, където е дефинирана.

4.4.2. Оператор if/else

Операторът се използва за избор на една от две възможни алтернативи в зависимост от стойността на дадено условие.

Чрез него се реализира разклоняващ се изчислителен процес от вид, илюстриран на Фиг. 4.5.



Фиг. 4.5 Разклоняващ се изчислителен процес

Ако указаното условие е в сила, се изпълняват се едни действия, а ако не – други действия. И в двата случая след това се изпълняват общи действия.

Условието се задава чрез някакъв булев израз, а действия 1 и действия 2 – чрез оператори. Фиг. 4.6 описва подробно синтаксиса и семантиката на този оператор.

Оператор if/else

Синтаксис

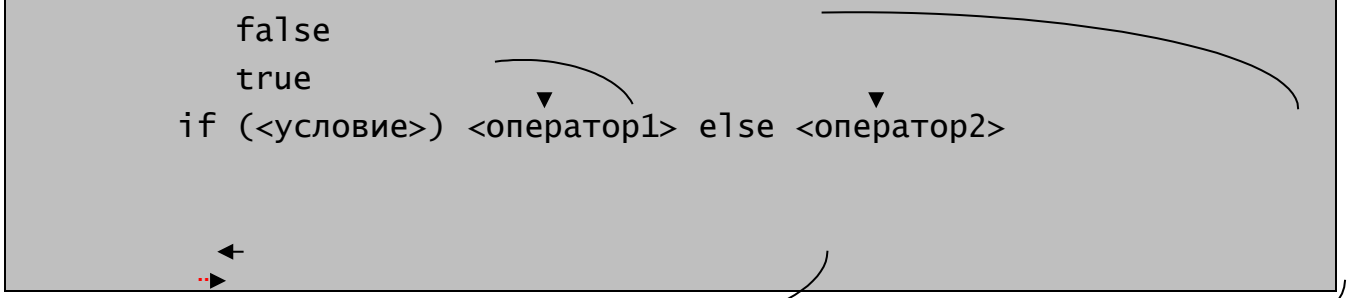
if (<условие>) <оператор1> else <оператор2>

където

- if (ако) и else (иначе) са запазени думи;
- <условие> е булев израз;
- <оператор1> и <оператор2> са произволни оператори.

Семантика

Пресмята се стойността на булевия израз, представящ условието. Ако резултатът е true, изпълнява се <оператор1>. В противен случай се изпълнява <оператор2>, т.е.



Фиг. 4.6 Оператор if/else

Забележки:

1. Булевият израз, определящ <условие>, трябва да бъде напълно определен. Задължително се огражда в кръгли скоби.
2. Операторът след условието е точно един. Ако е необходимо няколко оператора да се изпълнят, трябва да се обединят в блок.
3. Операторът след else е точно един. Ако е необходимо няколко оператора да се изпълнят, трябва да се обединят в блок.

4. Операторът if (<условие>) ; else <оператор> е еквивалентен на оператора if (!<условие>) <оператор>. В него операторът след <условие> е празния.

Задача 20. Променливата y зависи от променливата x. Зависимостта е следната:

$$y = \begin{cases} \lg(x) + 1.82, & x \geq 1 \\ x^2 + 7.x + 8.82, & x < 1. \end{cases}$$

Да се напише програма, която по дадено x намира съответната стойност на y.

Програма Zad20.cpp решава задачата.

```
// Program Zad20.cpp
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
int main()
{cout << "x= ";
 double x;
 cin >> x;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 } // въведена е валидна стойност за x
 double y;
 if (x >= 1) y = log10(x) + 1.82;
  else y = x*x + 7*x + 8.82;
  cout << setprecision(3) << setiosflags(ios :: fixed);
  cout << setw(10) << x << setw(10) << y << "\n";
  return 0;
}
```

С въвеждането на стойност на променливата x се прави проверка за валидността на въведената стойност. Ако е въведена невалидна стойност, изпълнението на програмата звършва с извеждане на

съобщението “Error! Bad Input!!”. Изпълнението на оператора if/else води до пресмятане на стойността на булевия израз x >= 1. Ако тя е true, се изпълнява операторът за присвояване y = log10(x) + 1.82;. В противен случай се изпълнява операторът за присвояване y = x*x + 7*x + 8.82;, след което се извежда резултатът.

Вложени условни оператори

В условните оператори:

if (<условие>) <оператор>

if (<условие>) <оператор1> else <оператор2>

<оператор>, <оператор1> и <оператор2> са произволни оператори, в т. число могат да бъдат условни оператори. В този случай имаме вложени условни оператори.

При влагането е възможно да възникнат двусмислици. Ако в един условен оператор има повече запазени думи if отколкото else, възниква въпросът, за кой от операторите if се отнася съответното else. Например, нека разгледаме оператора

```
if (x >= 0) if ( x >= 5) x = 1/x; else x = -x;
```

Възможни са следните две различни тълкувания на този оператор:

а) if оператор, съдържащ if/else оператор, т.е.

```
if (x >= 0)
   if (x >= 5) x = 1/x; else x = -x;
```

При това тълкуване, ако преди изпълнението на if оператора x има стойност –5, след изпълнението му, стойността на x остава непроменена.

б) if/else оператор, с if оператор след <условие>, т.е.

```
if (x >= 0) if ( x >= 5) x = 1/x;
else x = -x;
```

При това тълкуване, ако преди изпълнението на if/else оператора x има стойност –5, след изпълнението му, стойността на x става 5.

Записът чрез съответни подравнявания, не влияе на компилатора. В езика C++ има правило, което определя начина по който се изпълняват вложени условни оператори.

Правило: Всяко else се съчетава в един условен оператор с най-близкото преди него несъчетано if. Текстът се гледа отляво надясно.

Според това правило, компилаторът на C++ ще приеме първото тълкуване за горните вложени условни оператори.

Препоръка: Условен оператор да се влага в друг условен оператор само след else. Ако се налага да се вложи след условието, вложеният условен оператор да се направи блок.

Задачи върху операторите if и if/else

Задача 21. Ако променливата a има стойност 8, определете каква стойност ще има променливата b след изпълнението на оператора

```
if (a > 4) b = 5; else
  if (a < 4) b = -5; else
     if (a == 8) b = 8; else b = 3;
```

Тъй като е в сила условието a > 4, променливата b ще получи стойността 5.

Задача 22. Стойността на y зависи от x. Зависимостта е следната:

$$y = \begin{cases} x, & x \le 2 \\ 2, & x \in (2, 3] \\ x - 1, & x > 3 \end{cases}$$

Да се напише програма, която по дадено x, намира стойността на y.

Програма Zad22.cpp решава задачата.

```
// Program Zad22.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{ cout << "x= ";
  double x;
  cin >> x;
  if (!cin)
```

```
  { cout << "Error. Bad input! \n";
    return 1;
  }
  double y;
  if (x <= 2) y = x; else
     if (x <= 3) y = 2; else y = x-1;
  cout << setprecision(3) << setiosflags(ios :: fixed);
  cout << setw(10) << x << setw(10) << y << "\n";
  return 0;
 }
```

Забележка: В програма Zad22.cpp след първото else е в сила условието x > 2. Затова не е нужно то да се проверява.

Задача 23. Да се напише програма, която въвежда три реални числа a, b и c и извежда 0, ако не съществува триъгълник със страни a, b и c. Ако такъв триъгълник съществува, да извежда 3, 2 или 1 в зависимост от това какъв е триъгълникът - равностранен, равнобедрен или разностранен съответно.

Програма Zad23.cpp решава задачата.

```
// Program Zad23.cpp
#include <iostream.h>
int main()
{cout << "a= ";
 double a;
 cin >> a;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 cout << "b= ";
 double b;
 cin >> b;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
```

```
 }
 cout << "c= ";
 double c;
 cin >> c;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 bool x = a <= 0 || b <= 0 || c <= 0 ||
        a+b <= c || a+c <= b || b+c <= a;
 if (x) cout << 0 << "\n"; else
   if (a == b && b == c) cout << 3 << "\n"; else
     if (a == b || a == c || b == c) cout << 2 << "\n"; else
        cout << 1 << "\n";
 return 0;
}
```

Булевата променлива x е помощна. Тя има стойност true, ако a, b и c не са страни на триъгълник. Получена е след намиране на отрицанието на условието a, b и c да са страни на триъгълник, т.е. на условието

a > 0 && b > 0 && c > 0 && a + b > c && a + c > b && b + c > a

като са използвани законите на де Морган.

Закони на де Морган:
!!A е еквивалентно на A
!(A || B) е еквивалентно на !A && !B
!(A && B) е еквивалентно на !A || !B

Задача 24. Да се напише програма, която на цялата променлива k присвоява номера на квадранта, в който се намира точка с координати (x, y). Точката не лежи на координатните оси, т.е. x.y ≠ 0.

Програмата Zad24.cpp решава задачата.

```
// Program Zad24.cpp
#include <iostream.h>
int main()
{cout << "x=";
```

```
double x;
cin >> x;
if (!cin)
{cout << "Error. Bad input! \n";
 return 1;
 }
cout << "y=";
double y;
cin >> y;
if (!cin)
{cout << "Error, Bad input! \n";
 return 1;
}
if (x*y == 0)
{cout << "The input is incorrect! \n";
 return 1;
}
int k;
if (x*y>0) {if (x>0) k = 1; else k= 3;}
else
 if (x>0) k = 4; else k = 2;
cout << "The point is in: " << k << "\n";
return 0;
}
```

4.4.3. Оператор switch

Често се налага да се избере за изпълнение един от множество от варианти. Пример за това дава следната задача.

Задача 25. Да се напише програма, която въвежда цифра, след което я извежда с думи.

За решаването на задачата трябва да се реализира следната неелементарна функция:

$$f = \begin{cases} zero, & i = 0 \\ one, & i = 1 \\ \ldots & \ldots \\ nine, & i = 9 \end{cases}$$

където с i е означено въведената цифра.

Последното може да стане чрез следната програма:

```
#include <iostream.h>
int main()
{cout << "i= ";
 int i;
 cin >> i;
 if (!cin)
 {cout << "Error, bad input!\n";
  return 1;
 }
 if (i < 0 || i > 9) cout << "Incorrect input! \n";
 else if (i == 0) cout << "zero \n";
 else if (i == 1) cout << "one \n";
 else if (i == 2) cout << "two \n";
 else if (i == 3) cout << "three \n";
 else if (i == 4) cout << "four \n";
 else if (i == 5) cout << "five \n";
 else if (i == 6) cout << "six \n";
 else if (i == 7) cout << "seven \n";
 else if (i == 8) cout << "eight \n";
 else if (i == 9) cout << "nine \n";
 return 0;
}
```

В нея са използвани вложени if и if/else оператори, условията на които сравняват променливата i с цифрите 0, 1, 2, …, 9.

Има по-удобна форма за реализиране на това влагане. Постига се чрез оператора за избор на вариант switch.

Програма Zad25.cpp е друго решение на задачата.

```
// Program Zad25.cpp
#include <iostream.h>
int main()
{cout << "i= ";
 int i;
 cin >> i;
 if (!cin)
```

```
 {cout << "Error, bad input!\n";
  return 1;
 }
 switch (i)
 {case 0 : cout << "zero \n"; break;
  case 1 : cout << "one \n";  break;
  case 2 : cout << "two \n";  break;
  case 3 : cout << "three \n"; break;
  case 4 : cout << "four \n"; break;
  case 5 : cout << "five \n"; break;
  case 6 : cout << "six \n"; break;
  case 7 : cout << "seven \n"; break;
  case 8 : cout << "eight \n"; break;
  case 9 : cout << "nine \n"; break;
  default: cout << “Incorrect Input! \n”;
 }
 return 0;
}
```

Операторът switch започва със запазената дума switch (ключ), следвана от, ограден в кръгли скоби, цял израз. Между фигурните скоби са изброени вариантите на оператора. Описанието им започва със запазената дума case (случай, вариант), следвана в случая от цифра, наречена етикет, двоеточие и редица от оператори.

Изпълнение на програмата

След въвеждането на стойност на променливата i се извършва проверка за коректност на въведеното. Нека въведената стойност е 7. Изпълнението на оператора switch причинява да бъде пресметната стойността на израза i – в случая 7. След това последователно сравнява тази стойност със стойностите на етикетите до намиране на етикета 7 и изпълнява редицата от оператори след него. В резултат върху екрана се извежда

seven

курсурът се премества на нов ред и се прекъсва изпълнението на оператора switch. Последното е причинено от оператора break в края на редицата от оператори за варианта с етикет 7.

Операторът switch реализира избор на вариант от множество варианти (възможности). Синтаксисът и семантиката му са дадени на Фиг. 4.7.

Оператор switch

Синтаксис

switch (<израз>)
{case <израз$_1$> : <редица_от_оператори$_1$>
 case <израз$_2$> : <редица_от_оператори$_2$>
  …
 case <израз$_{n-1}$> : <редица_от_оператори$_{n-1}$>
 [default : <редица_от_оператори$_n$>]$_{опц}$
}

където

- switch (ключ), case (случай, избор или вариант) и default (по премълчаване) са запазени думи на езика;
- <израз> е израз от допустим тип (Типовете bool, int и char са допустими, реалните типове double и float не са допустими). Ще го наричаме още switch-израз.
- <израз$_1$>, <израз$_2$>, …, <израз$_{n-1}$> са константни изрази, задължително с различни стойности.
- <редица_от_оператори$_i$> (i = 1, 2, …, n) се дефинира по следния начин:

<редица_от_оператори> ::= <празно>|
<оператор>|
<оператор><редица_от_оператори>

Семантика

Намира се стойността на switch-израза. Получената константа се сравнява последователно със стойностите на етикетите <израз$_1$>, <израз$_2$>, … При съвпадение, се изпълняват операторите на съответния вариант и операторите на всички варианти, разположени след него, до срещане на оператор break. В противен случай, ако участва default-вариант, се изпълнява редицата от оператори, която му съответства и редиците от оператори на всички варианти след него до достигане до break и в случай, че не участва такъв – не следват никакви действия от оператора switch.

Фиг. 4.7 Оператор switch

Между фигурните скоби са изброени вариантите на оператора. Всеки вариант (без евентуално един) започва със запазената дума case, следвана от израз (нарича се още case-израз или етикет), който трябва да може да се пресметне по време на компилация. Такива изрази се наричат **константни**. Те не зависят от входните данни. След константния израз се поставя знакът двоеточие, следван от редица от оператори (оператори на варианта), която може да е празна. Сред вариантите може да има един (не е задължителен), който няма case-израз и започва със запазената дума default. Той се изпълнява в случай, че никой от останалите варианти не е бил изпълнен.

Забележка: Не е задължително default – вариантът (ако го има) да е последен, но добрият стил за програмиране го изисква.

Съществува възможност програмистът да съобщи на компилатора, че желае да се изпълни само редицата от оператори на варианта с етикет, съвпадащ със стойността на switch-израза, а не и всички следващи го. Това се реализира чрез използване на оператор break в края на редицата от оператори на варианта. Този оператор предизвиква прекъсване на изпълнението на оператора switch и предаване на управлението на първия оператор след него (Фиг. 4.8.).

Операторът break принадлежи към групата на т. нар. **оператори за преход**. Тези оператори предават управлението безусловно в някаква точка на програмата.

Оператор break

Синтаксис

```
break;
```

Семантика

Прекратява изпълнеието на най-вътрешния съдържащ го оператор switch или оператор за цикъл. Изпълнението на програмата продължава от оператора, следващ (съдържащ) прекъснатия.

Фиг. 4.8 Оператор break

Програмистът съзнателно пропуска оператора break, когато за няколко различни стойности от множеството от стойности, трябва да се извършат еднакви действия.

Пример: Следният програмен фрагмент извежда дали въведена цифра е четно или нечетно число.

```
…
switch (i)
{case 1:
 case 3:
 case 5:
 case 7:
 case 9: cout << "odd number \n"; break;
 case 0:
 case 2:
 case 4:
 case 6:
 case 8: cout << "even number \n";
}
```

Забележка: Ако във вариантите на оператора switch не е използван операторът break, ще бъде изпълнена редицата от оператори на варианта, чийто case-израз съвпада със стойността на switch-израза и също всички след нея.

Използването на оператора switch има едно единствено предимство пред операторите if и if/else – прави реализацията по-ясна. Основен негов недостатък е, че може да се прилага при много специални обстоятелства, произтичащи от наложените ограничения на типа на switch-израза, а именно, той трябва да е цял, булев или символен. Освен това, използването на оператора break, затруднява доказването на важни математически свойства на програмите.

Задачи върху оператора switch

Задача 26. Да се напише програма, която по зададено реално число x намира стойността на един от следните изрази:

```
y = x - 5
y = sin(x)
y = cos(x)
y = exp(x).
```

Изборът на желания израз да става по следния начин: при въвеждане на цифрата 1 се избира първият, на 2 – вторият, на 3 – третият и на 4 – четвъртият израз.

Програма Zad25.cpp решава задачата.

```
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
int main()
{cout << "================================\n";
 cout << "|    y = x-5             -> 1    |\n";
 cout << "|    y = sin(x)          -> 2    |\n ";
 cout << "|    y = cos(x)          -> 3    |\n";
 cout << "|    y = exp(x)          -> 4    |\n";
 cout << "================================\n";
 cout << " 1, 2, 3 or 4? \n";
 int i;
 cin >> i;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (i == 1 || i == 2 || i == 3 || i == 4)
 {cout << "x= ";
  double x;
  cin >> x;
  if (!cin)
  {cout << "Error. Bad input! \n";
   return 1;
  }
  double y;
  switch (i)
```

```
    {case 1: y = x - 5;  break;
     case 2: y = sin(x); break;
     case 3: y = cos(x); break;
     case 4: y = exp(x); break;
    }
    cout << "y= " << y << "\n";
   }
   else
   {cout << "Error. Bad choice! \n";
    return 1;
   }
   return 0;
  }
```

4.5. Оператори за цикъл

Операторите за цикъл се използват за реализиране на циклични изчислителни процеси.

Изчислителен процес, при който оператор или група оператори се изпълняват многократно за различни стойности на техни параметри, се нарича **цикличен**.

Съществуват два вида циклични процеси:

- индуктивни и
- итеративни.

Цикличен изчислителен процес, при който броят на повторенията е известен предварително, се нарича **индуктивен цикличен процес**.

Пример: По дадени цяло число n и реално число x, да се намери сумата

$$S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}.$$

Ако S има начална стойност 1, за да се намери сумата е необходимо n пъти да се повторят следните действия:

а) конструиране на събираемо

$$\frac{x^i}{i!}, \ (i = 1, 2, \ldots, n)$$

б) добавяне на събираемото към S.

Цикличен изчислителен процес, при който броят на повторенията **не** е известен предварително, се нарича **итеративен цикличен процес**. При тези циклични процеси, броят на повторенията зависи от някакво условие.

Пример: По дадени реални числа x и $\varepsilon > 0$, да се намери сумата

$$S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

където сумирането продължава до добавяне на събираемо, абсолютната стойност на което е по-малка от ε.

Ако S има начална стойност 1, за да се намери сумата е необходимо да се повторят следните действия:

а) конструиране на събираемо

$$\frac{x^i}{i!}, (i = 1, 2, \ldots)$$

б) добавяне на събираемото към S

докато абсолютната стойност на последното добавено към сумата S събираемо стане по-малка от ε.

В този случай, броят на повторенията зависи от стойностите на x и ε.

В езика C++ има три оператора за цикъл:

- *оператор for*

Чрез него могат да се реализират произволни циклични процеси, но се използва главно за реализиране на индуктивни циклични процеси.

- *оператори while и do/while*

Използват се за реализиране на произволни циклични процеси – индуктивни и итеративни.

4.5.1. Оператор for

Използва се основно за реализиране на индуктивни изчислителни процеси. Чрез пример ще илюстрираме неговите синтаксис и семантика.

Задача 26. Да се напише програма, която по дадено естествено число n, намира факториела му.

Тъй като n! = 1.2. … .(n-1).n, следната редица от оператори го реализира:

```
int fact = 1;
fact = fact * 1;
fact = fact * 2;
…
fact = fact * (n-1);
fact = fact * n;
```

В нея операторите за присвояване са написани по следния общ шаблон:

```
fact = fact * i;
```

където i е цяла променлива, получаваща последователно стойностите 1, 2, …, (n-1), n.

Следователно, за да се намери n! трябва да се повтори изпълнението на оператора fact = fact * i, за i = 1, 2, …, n. Това може да стане с помощта на оператора for.

Програма Zad26.cpp решава задачата.

```
// Program Zad26.cpp
#include <iostream.h>
int main()
{cout << "n= ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad Input! \n";
  return 1;
 }
 if (n <= 0)
 {cout << "Incorrect Input! \n";
  return 1;
 }
```

```
    int fact = 1;
    for (int i = 1; i <= n; i++)
       fact = fact * i;
    cout << n << "! = " << fact << "\n";
    return 0;
   }
```

Операторът for е в одебелен шрифт. Той започва със запазената дума for (за). В кръгли скоби след нея е реализацията на конструкцията i = 1, 2, …, n. Тя се състои от три части: инициализация (int i = 1;), условие (i <= n) и корекция (i++), отделени с ;. Забележете, че операторът i++ не завършва с ;. След този фрагмент е записан операторът fact = fact * i;, описващ действията, които се повтарят. Нарича се **тяло на цикъла.**

Изпълнението на програмата започва с въвеждане стойност на променливата n и проверка на валидността на въведеното. Нека n = 3. След дефиницията на цялата променлива fact, в ОП за нея са отделени 4 байта, инициализирани с 1. Изпълнението на оператора for предизвиква за цялата променлива i да бъдат заделени 4 байта ОП, които да се инициализират също с 1. Следва проверка на условието i<=n и тъй като то е истина (1<=3), се изпълнява операторът fact = fact*i;, след което fact получава стойност 1. Изпълнението на оператора i++ увеличава текущата стойност на i с 1 и новата й стойност вече е 2. Отново следва проверка на условието i<=n и тъй като то е истина (2<=3), се изпълнява операторът fact = fact*i;, след което fact получава стойност 2. Изпълнението на оператора i++ увеличава текущата стойност на i с 1 и новата й стойност вече е 3. Пак следва проверка на условието i<=n и тъй като то продължава да е истина (3<=3), отново се изпълнява операторът fact = fact*i;, след което fact получава стойност 2*3, т.е. 6. Изпълнението на оператора i++ увеличава текущата стойност на i с 1 и новата й стойност вече е 4. Условието i<=n е лъжа и изпълнението на оператора for завършва.

Въпреки, че променливата i е дефинирана в оператора for, тя е “видима” (може да се използва) след изпълнението му, като стойността й е първата, за която условието i<=n не е в сила (в случая 4).

На Фиг. 4.9 е дадено детайлно описание на оператора for.

Оператор for

Синтаксис

for (<инициализация>; <условие>; <корекция>)
  <оператор>

където

- for (за) е запазена дума.
- <инициализация> е или **точно една** дефиниция с инициализация на една или повече променливи, или няколко оператора за присвояване или въвеждане, **отделени със , и не завършващи с ;.**
- <условие> е булев израз.
- <корекция> е един или няколко оператора, **незавършващи с ;**. В случай, че са няколко, отделят се със ,.
- <оператор> е точно един произволен оператор. Нарича се **тяло на цикъла.**

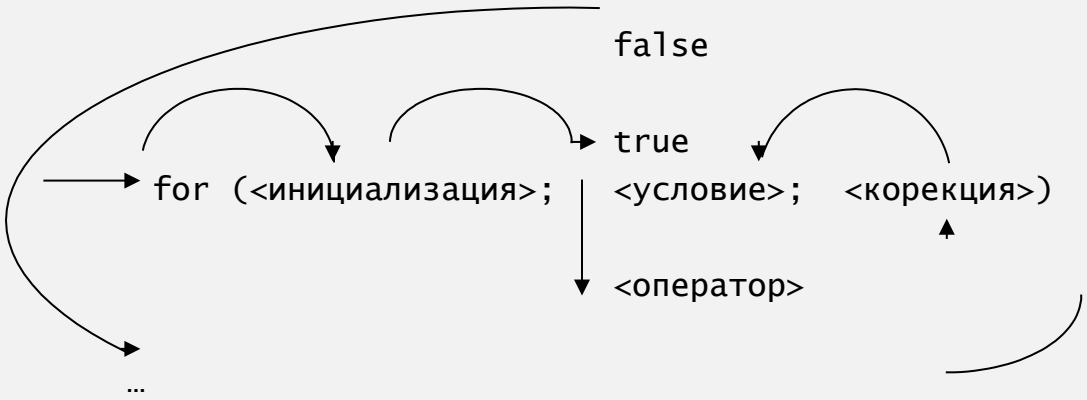
Семантика

Изпълнението започва с изпълнение на частта <инициализация>. След това се намира стойността на <условие>. Ако в резултат се е получило false, изпълнението на оператора for завършва, без тялото да се е изпълнило нито веднъж. В противен случай последователно се повтарят следните действия:

- Изпълнение на тялото на цикъла;
- Изпълнение на операторите от частта <корекция>;
- Пресмятане стойността на <условие>

докато стойността на <условие> е true.

Следната схема илюстрира изпълнението му:



Фиг. 4.9 Оператор за цикъл for

Възможно е частите <инициализция>, <условие> и <корекция> поотделно или заедно, да са празни. Разделителите (;) между тях обаче трябва да фигурират. Ако частта <условие> е празна, подразбира се true.

Забележки:

1. Тялото на оператора for е точно един оператор. Ако повече оператори трябва да се използват, се оформя блок.

2. Частта <инициализация> се изпълнява само веднъж – в началото на цикъла. Възможно е да се изнесе пред оператора for и остане празна.

Пример:

```
int i = 1;
for (; i<= n; i++)
    fact = fact* i;
```

В нея не са допустими редици от оператори и дефиниция на променливи, т.е. недопустими са:

```
for (int i, i = 4; …
```

или

```
int i;
for (i = 4, int j = 5; …
```

а също две дефиниции, например

```
for (int i=3, double a = 3.5; …
```

Нарича се така, тъй като в нея обикновено се инициализират една или повече променливи.

3. Частта <корекция> се нарича така, тъй като обикновено чрез нея се модифицират стойностите на променливите, инициализирани в частта <инициализация>. Тя може да се премести в тялото на оператора for като се оформи блок от вида {<оператор> <корекция>;},

Пример:

```
for (int i = 1; i<= n;)
   {fact = fact* i;
    i++;
   }
```

4. Ако частта <условие> е празна, подразбира се true. За да се избегне зацикляне, от тялото на цикъла при определени условия трябва да се излезе принудително, например чрез оператора break.

Пример:

```
for (int i = 1; ; i++)
    if (i > n) break;
 else fact = fact* i;
```

Това обаче е лош стил на програмиране и не го препоръчваме.

5. Следствие разширената интерпретация на true и false, частта <условие> може да бъде и аритметичен израз. Това също е лош стил на програмиране и не го препоръчваме.

Област на променливите, дефинирани в заглавната част на for

Под област на променлива се разбира мястото в програмата, където променливата може да се използва. Казва се още където тя е "видима".

Съгласно стандарта ANSI, областта на променлива, дефинирана в заглавната част на цикъла for започва от дефиницията ѝ и продължава до края на цикъла.

Това значи, че тези променливи **не** са видими след оператора for, в който са дефинирани, т.е. фрагментът

```
for (int i = 1; i <= n; i++)
{…
}
for (i = 1; i <= m; i++)
{…
}
```

е недопустим - ще предизвика синтактична грешка заради недефинирана променлива i в заглавната част на втория оператор for.

Но тъй като това е ново решение на специалистите, поддържащи езика, повечето реализации в т.число и реализацията на Visual C++ 6.0, използват стария вариант, според който *областта на променлива, дефинирана в заглавната част на цикъла for започва от дефиницията ѝ и продължава до края на оператора, в частност блока, в който се намира операторът for*. Така, за реализацията на Visual C++ 6.0, горният фрагмент е напълно допустим, а фрагментът

```
for (int i = 1; i <= n; i++)
{…
}
```

```
   for (int i = 1; i <= m; i++)
   {…
   }
```
сигнализира повторна дефиниция на променливата i.

При същата реализация, областта на променливата j от програмния фрагмент:

```
   for (int i = 1; i <= n ; i++)
      for (int j = 1; j <=m; j++)
         { …
         }
```
е края на оператора for (int i = 1; …) …, т.е.

```
   for (int i = 1; i <= n ; i++)
      for (int j = 1; j <=m; j++)
         { …
         }
   for (j = …; ….) {…}
```
ще издаде съобщение за грешка заради недефинирана променлива j.

Аналогично, фрагментът ще съобщи за недефинирана променлива k

```
   #include <iostream.h>
   int main()
   {for (int i = 1; i<=3; i++)
     {for (int j = 1; j<=3; j++)
       {for (int k = 1; k<=3; k++)
         cout << j << " " << k <<"\n";
        …
        }
      cout << i << " " << k;
      }
    return 0;
   }
```
тъй като областт на променливата k завършва в края на оператора for (int j = 1; ….

Препоръка: При вложени оператори for да не се дефинират променливи в заглавните части на операторите.

Задачи върху оператора for

Задача 27. За какво може да бъде използвана следната програма?

```
// Program Zad27.cpp
#include <iostream.h>
int main()
{cout << "n= ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n <= 0)
 {cout << "Incorrect input! \n";
  return 1;
 }
 int f = 1;
 for (int i = 1; i <= n; cin >> i)
 f = f * i;
 cout << f << "\n";
 return 0;
}
```

Ще отбележим, че тази програма илюстрира използването на оператора for за реализиране на итеративни циклични процеси (заради оператора cin >> i). Това обаче се счита за лош стил за програмиране.

Препоръка: Използвайте оператора for **само** за реализиране на индуктивни циклични процеси. Освен това, ако for има вида:

```
for (i = start; i < (или i <= ) end; i = i + increment)
{ …
}
```

не променяйте i, start, end и increment в тялото на цикъла. Това е лош стил за програмиране. Ако цикличният процес, който трябва да реализирате, не се вмества в тази схема, не използвайте оператора for.

Задача 28. Да се напише програма, която по дадени x – реално и n – естествено число, пресмята сумата

$$s = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}.$$

Програма Zad28.cpp решава задачата.

```
// Program Zad28.cpp
#include <iostream.h>
int main()
{cout << "x= ";
 double x;
 cin >> x;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 cout << "n= ";
 short n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n <= 0)
 {cout << "Incorrect input! \n";
  return 1;
 }
 double x1 = 1;
 double s = 1;
 for (int i = 1; i <= n; i++)
 {x1 = x1 * x / i;
  s = s + x1;
 }
 cout << "s= " << s << "\n";
```

```
 return 0;
}
```

Задача 29. Нека n и m са дадени естествени числа, n ≥ 1, m > 1. Да се напише програма,която определя броя на елементите от серията числа

$i^3 + 7.i^2 + n^3$, i = 1, 2, ..., n.

които са кратни на m.

Програма Zad29.cpp решава задачата.

```
// Program Zad29.cpp
#include <iostream.h>
int main()
{cout << "n= ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n < 1)
 {cout << "Incorrect input! \n";
  return 1;
 }
 cout << "m= ";
 int m;
 cin >> m;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (m <= 1)
 {cout << "Incorrect input! \n";
  return 1;
 }
```

```
 int br = 0;
 for (int i = 1; i <= n; i++)
 if ((i*i*i + 7*i*i + n*n*n) % 7 == 0) br++;
 cout << "br= " << br << "\n";
 return 0;
}
```

Задача 30. Дадено е естественото число n, n ≥ 1. Да се напише програма, която намира най-голямото число от серията числа:

$$i^2.\cos(n + \frac{i}{n}),\ i = 1, 2, \ldots, n.$$

Програма Zad30.cpp решава задачата.

```
// Program Zad30.cpp
#include <iostream.h>
#include <math.h>
int main()
{cout << "n= ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n < 1)
 {cout << "Incorrect input! \n";
  return 1;
 }
 double max = cos(n+1/n);
 for (int i = 2; i <= n; i++)
 {double p = i*i*cos(n+i/n);
  if (p > max) max = p;
 }
 cout << "max= " << max << "\n";
 return 0;
}
```

Задача 31. Да се напише програма, която извежда върху екрана таблицата от стойностите на функциите sin x и cos x в интервала [0, 1].

Програма Zad31.cpp решава задачата.

```
// Program Zad31.cpp
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
int main()
{cout << setprecision(5) << setiosflags(ios :: fixed);
 for (double x = 0; x <= 1; x = x + 0.1)
   cout << setw(10) << x << setw(10) << sin(x)
        << setw(10) << cos(x) << "\n";
 return 0;
}
```

4.5.2. Оператор while

Чрез този оператор може да се реализира произволен цикличен процес. С пример ще илюстрираме използването му.

Задача 32. Да се напише програма, която по дадени реални числа x и ε ($\varepsilon > 0$), приближено пресмята сумата

$$S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Сумирането да продължи докато абсолютната стойност на последното добавено събираемо стане по-малка от ε.

В тази задача броят на повторенията предварително не е известен, а зависи от условието |x1|<ε, където с x1 е означено произволно събираемо. За решаването й е необходимо да се премине през следните стъпки:

- Въвеждане на стойности на x и ε.
- Инициализация x1 = 1; s = 1.
- Докато е в сила условието |x1| ≥ ε, повтаряне на действията
  {конструиране на ново събираемо x1;
   s = s + x1;
  }
- Извеждане на S.

За реализирането на този алгоритъм, ще използваме оператора за цикъл while.

Програма Zad32.cpp решава задачата.

```
// Program Zad32.cpp
#include <iostream.h>
#include <math.h>
int main()
{cout << "x= ";
 double x;
 cin >> x;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 cout << "eps= ";
 double eps;
 cin >> eps;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (eps <= 0)
 {cout << "Incorrect input! \n";
  return 1;
```

```
 }
 double x1 = 1;
 double s = 1;
 int i = 1;
 while (fabs(x1) >= eps)
 {x1 = x1 * x / i;
  s = s + x1;
  i++;
 }
 cout << "s=" << s << "\n";
 return 0;
}
```

Операторът while в нея е в одебелен шрифт. Той започва със запазената дума while (докато), след която оградено в кръгли скоби е условието fabs(x1) >= eps и завършва с оператора

```
{x1 = x1 * x / i;
  s = s + x1;
  i++;
 }
```

представляващ тялото на цикъла. Този оператор може да се прочете по следния начин: “докато е в сила условието fabs(x1) >= eps, повтаряй следното …”.

Ще проследим изпълнението на програмата за x = 1 и eps = 0.5.

След изпълнението на операторите за въвеждане и дефинициите на s, x1 и i, състоянието на паметта е следното:

x	eps	x1	s	i
1.0	0.5	1.0	1.0	1

Изпълнението на оператора за цикъл започва с пресмятане на стойността на булевия израз fabs(x1) >= eps и тъй като тя е true, се изпълнява тялото на цикъла. В резултат имаме:

x	eps	x1	s	i
1.0	0.5	1.0	2.0	2

S е натрупало първите два члена на сумата. Отново се намира стойността на булевия израз fabs(x1) >= eps - пак true и се изпълнява тялото на цикъла. В резултат се получава:

x	eps	x1	s	i
1.0	0.5	0.5	2.5	3

Сега вече S е натрупало първите три члена на сумата. Стойността на булевия израз продължава да бъде true, заради което следва поредно изпълнение на тялото. Имаме:

x	eps	x1	s	i
1.0	0.5	0.166667	2.66667	4

Отново се намира стойността на условието. Тя вече е false, което причинява завършване на изпълнението на цикъла while.

Следователно, изпълнението на оператора за цикъл while продължава докато е изпълнено условието след запазената дума while и завършва веднага когато за текущите стойности на неговите параметри то не е в сила.

Задаването на по-голяма точност (по-малка стойност на eps), ще доведе до пресмятане на сумата с по-голяма точност.

На Фиг. 4.10 е дадено описание на синтаксиса и семантиката на оператора.

Оператор while

Синтаксис

while (<условие>) <оператор>

където

- while (докато) е запазена дума;
- <условие> е булев израз;
- <оператор> е произволен оператор. Той описва действията, които се повтарят и се нарича тяло на цикъла.

Семантика

Пресмята се стойността на <условие>. Ако тя е false, изпълнението на оператора while завършва без да се е изпълнило тялото му нито

веднъж. В противен случай, изпълнението на <оператор> и пресмятането на стойността на <условие> се повтарят докато <условие> е true. В първия момент, когато <условие> стане false, изпълнението на while завършва. Изпълнението на while може графично да се илюстрира по следния начин:

Фиг. 4.10 Оператор while

Забележки:

1. Ако е необходимо да се изпълнят многократно няколко оператора, те трябва да се оформят като блок.

2. Следствие разширената интерпретация на true и false, частта <условие> може да бъде и

аритметичен израз. Това обаче е лош стил на програмиране и не го препоръчваме.

3. Тъй като първото действие, свързано с изпълнението на оператора while, е проверката на условието му, операторът се нарича още оператор за цикъл с пред-условие.

4. Операторът

for (<инициализация>; <условие>; <корекция>)
<оператор>
е еквивалентен на
<инициализация>
while (<условие>)
{<оператор>
<корекция>;
}

Пример за това дава решението на задача 33.

Задачи върху оператора while

Задача 33. Да се напише програма, която намира факториела на дадено естествено число. За целта да се използва операторът while.

Програма Zad33.cpp решава задачата.

```
// Program Zad33.cpp
#include <iostream.h>
```

```
int main()
{cout << "n= ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad Input! \n";
  return 1;
 }
 if (n <= 0)
 {cout << "Incorrect Input! \n";
  return 1;
 }
 int fact = 1;
 int i = 1;
 while (i <= n)
 {fact = fact * i;
  i++;
 }
 cout << n << "! = " << fact << "\n";
 return 0;
}
```

Задача 34. **Да се напише програма, която въвежда от клавиатурата редица от цели числа и намира средноаритметичното им. Въвеждането да продължава до въвеждане на 0.**

Програма Zad34.cpp решава задачата.

```
// Program Zad34.cpp
#include <iostream.h>
int main()
```

```
{int count = 0;
double average = 0;
cout << "> ";
int number;
cin >> number;
while (number != 0)
{count++;
average = average + number;
cout << "> ";
cin >> number;
}
if (count != 0) average = average/count;
cout << "average= " << average << "\n";
return 0;
}
```

Забележете отсъствието на проверка за валидност на входните данни. Както вече е известно, това е лош стил за програмиране. Освен това, изборът на числото 0 за край на входните данни, също не винаги е подходящ. Zad35.cpp е подобрение на горното решение.

Задача 35. Да се напише програма, която въвежда от клавиатурата редица от цели числа и

намира средноаритметичното им. Въвеждането да продължи до въвеждане на дума, при която cin попада в състояние fail.

Тъй като редицата е числова, за неин край може да служи която и да е редица от знаци, която не започва с цифра.

Програма Zad35.cpp решава задачата.

```
// Program Zad35.cpp
#include <iostream.h>
int main()
{int count = 0;
double average = 0;
cout << "> ";
int number;
cin >> number;
while (cin)
{count++;
average = average + number;
cout << "> ";
cin >> number;
}
if (count != 0) average = average/count;
```

```
cout << "average= " << average << "\n";
return 0;
}
```

Тъй като cin е стойност на израза cin >> number;, който два пъти е използван в програмата, ще заменим cin със cin >> number в условието на оператора while. Получаваме програмата:

```
#include <iostream.h>
int main()
{int count = 0;
int number;
double average = 0;
cout << "> ";
while (cin >> number)
{count++;
average = average + number;
cout << "> ";
}
if (count != 0) average = average/count;
cout << "average= " << average << "\n";
return 0;
}
```

Забележка: **Условието cin >> number на оператора while не завършва с ;.**

Въпрос: **Какво ще е поведението на програми Zad34.cpp и Zad35.cpp ако типът на average от double се промени на int?**

Задача 36. **Да се напише програма, която**

$$S = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

пресмята приближено безкрайната сума:

за произволно реално число x от интервала [-1, 1]. Сумирането да продължи докато последното добавено събираемо по модул стане по-малко от ε.

Програма Zad36.cpp решава задачата.

```
// Program Zad36.cpp
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
int main()
{cout << "x= ";
```

```
double x;
cin >> x;
if (!cin)
{cout << "Error. Bad input! \n";
return 1;
}
if (x < -1 || x > 1)
{cout << "Incorrect Input! \n";
return 1;
}
cout << "eps= ";
double eps;
cin >> eps;
if (!cin)
{cout << "Error. Bad input! \n";
return 1;
}
if (eps <= 0)
{cout << "Incorrect input! \n";
return 1;
}
double x1 = x;
double s = x1;
int i = 2;
```

```
while (fabs(x1) >= eps)
{x1 = -x1 * x * x / (i*(i+1));
s = s + x1;
i = i + 2;
}
cout << setprecision(6) << setiosflags(ios
:: fixed);
cout << "s= " << setw(10) << s << "\n";
return 0;
}
```

Нека сега решим същата задача, но с друго условие за край.

Задача 37. Да се напише програма, която

$$s = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

пресмята приближено безкрайната сума: за произволно реално число x от интервала [-1, 1]. Сумирането да продължи докато абсолютната стойност на разликата на последните две добавени събираеми стане по-малка от ε.

В този случай е необходимо да се конструират и добавят първите две събираеми и чак тогава

да се провери условието |x1 – x2| < ε. Ако то е в сила трябва да се съобщи сумата, а в противен случай да се продължи с конструирането и добавянето на следващото събираемо.

Програма zad37.cpp решава задачата.

```
// Program Zad37.cpp
#include <iostream.h>
#include <math.h>
int main()
{cout << "x= ";
double x;
cin >> x;
if (!cin)
{cout << "Error. Bad input! \n";
return 1;
}
if (x < -1 || x > 1)
{cout << "Incorrect Input! \n";
return 1;
}
cout << "eps= ";
double eps;
cin >> eps;
```

```
if (!cin)
{cout << "Error. Bad input! \n";
return 1;
}
if (eps <= 0)
{cout << "Incorrect input! \n";
return 1;
}
double x1 = x;
double x2 = -x*x*x/6.0;
double s = x1 + x2;
int i = 4;
while (fabs(x1-x2) >= eps)
{x1 = x2;
x2 = -x1*x*x/(i*(i+1));
s = s+x2;
i = i+2;
}
cout << "s= " << s << "\n";
return 0;
}
```

Това решение не е много добро. Тъй като условието за завършване на изпълнението на

оператора while съдържа две последователни събираеми, преди използването на оператора while, те трябва да бъдат конструирани, а след това поддържани в тялото на цикъла. По-добро решение може да се получи като се използва операторът do/while – оператор за цикъл с пост-условие.

4.5.3. Оператор do/while

Използва се за реализиране на произволни циклични процеси. Ще го илюстрираме чрез пример, след което ще опишем неговите синтаксис и семантика. За целта ще използваме задача 37.

Програма Zad37_1.cpp реализира тази задача, като използва оператора do/while.

```
// Program Zad37_1.cpp
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
int main()
{cout << "x= ";
```

```
double x;
cin >> x;
if (!cin)
{cout << "Error. Bad input! \n";
return 1;
}
if (x < -1 || x > 1)
{cout << "Incorrect Input! \n";
return 1;
}
cout << "eps= ";
double eps;
cin >> eps;
if (!cin)
{cout << "Error. Bad input! \n";
return 1;
}
if (eps <= 0)
{cout << "Incorrect input! \n";
return 1;
}
double x2 = x;
double s = x;
int i = 2;
```

```
double x1;
   do
   {x1 = x2;
x2 = -x1*x*x/(i*(i+1));
   s = s + x2;
   i = i + 2;
} while (fabs(x1-x2) >= eps);
cout << setprecision(5) <<
setiosflags(ios :: fixed);
cout << "s= " << setw(10) << s << "\n";
return 0;
}
```

Операторът do/while в нея е в одебелен шрифт. Започва със запазената дума do (прави, повтаряй следното), следва произволен оператор (в случая блок), който определя действията, които се повтарят и затова се нарича тяло на цикъла. Запазената дума while (докато) отделя тялото на оператора от булевия израз fabs(x1-x2) >= eps. Последният е ограден в кръгли скоби и определя условието за завършване изпълнението на цикъла.

Операторът do/while завършва с ;.

Ще проследим изпълнението на програмата за x = 1 и eps = 0.5.

След изпълнението на операторите за въвеждане и дефинициите на s, x1, x2 и i, състоянието на паметта е следното:

x	eps	x1	x2	s	i
1.0	0.5	-	1.0	1.0	2

Изпълнението на оператора за цикъл започва с изпълнение на тялото на цикъла – блока

```
{x1 = x2;
x2 = -x1*x*x/(i*(i+1));
s = s+x2;
i = i+2;
}
```

след което се получава:

x	eps	x1	x2	s	i
1.0	0.5	1.0	-0.16667	0.83333	4

Пресмята се стойността на булевия израз fabs(x1-x2) >= eps и тъй като тя е true, повторно се изпълняват операторите от блока, съставящ тялото. В резултат имаме:

x	eps	x1	x2	s	i
1.0	0.5	-0.16667	0.00833	0.84167	6

Сега вече стойността на булевия израз, определящ условието за завършване, е false. Изпълнението на оператора за цикъл завършва. С извеждането на стойността на сумата s завършва и изпълнението на програмата.

Забелязваме, че в тази програма, настройката на променливите x1 и x2 става в тялото на цикъла do/while, а не извън него. Това се обуславя от факта, че тялото на този вид цикъл поне веднъж ще се изпълни.

Описанието на синтаксиса и семантиката на оператора do/while е илюстрирано на Фиг. 4.11.

Оператор do/while

Синтаксис

do

<оператор>

while (<условие>);

където

- do (прави, повтаряй докато …) и while (докато) са запазени думи на езика.

- <оператор> е точно един оператор. Той описва действията, които се повтарят и се нарича тяло на цикъла.

- <условие> е булев израз. Нарича се условие за завършване изпълнението на цикъла. Огражда се в кръгли скоби.

Семантика

Изпълнява се тялото на цикъла, след което се пресмята стойността на <условие>. Ако то е false, изпълнението на оператора do/while завършва. В противен случай се повтарят действията: изпълнение на тялото и пресмятане стойността на <условие>, докато стойността на <условие> е true. Веднага, след като стойността му стане false, изпълнението на оператора завършва.

Фиг. 4.11 Оператор за цикъл do/while

Забележки:

1. Между запазените думи do и while стои точно един оператор. Ако няколко действия трябва да се опишат, оформя се блок.
2. *Дефинициите* в тялото, не са видими в <условие>. Например, не е допустим фрагментът:

```
double x2 = x;
double s = x;
```

```
int i = 2;
do
{double x1 = x2;
x2 = -x1 * x * x / (i*(i+1));
s = s + x2;
i = i + 2;
} while (fabs(x1-x2) >= eps);
```

Компилаторът ще съобщи, че променливата x1 от линия

```
} while (fabs(x1-x2) >= eps);
```

не е дефинирана.

Следователно, всички променливи в <условие> трябва да са дефинирани извън оператора do/while.

3. Следствие разширената интерпретация на true и false, частта <условие> може да е аритметичен израз. Това е лош стил за програмиране и не го препоръчваме.

4. Операторът do/while завършва с ;.

Задачи върху оператора do/while

Задача 38. Да се напише програма, която намира произведението на целите числа от m до

n, където m и n са дадени естествени числа и m ≤ n. За целта да се използва операторът do/while.

Програма Zad38.cpp решава задачата.

```
// Program Zad38.cpp
#include <iostream.h>
int main()
{cout << "m= ";
int m;
cin >> m;
if (!cin)
{cout << "Error. Bad Input! \n";
return 1;
}
if (m <= 0)
{cout << "Incorrect Input! \n";
return 1;
}
cout << "n= ";
int n;
cin >> n;
if (!cin)
{cout << "Error. Bad Input! \n";
```

```
            return 1;
                }
            if (n <= 0)
        {cout << "Incorrect Input! \n";
            return 1;
                }
            if (m > n)
        {cout << "Incorrect Input! \n";
            return 1;
                }
            int prod = 1;
            int i = m;
                do
            {prod = prod * i;
                i++;
            } while (i <= n);
        cout << prod << "\n";
            return 0;
                }
```

Тъй като е в сила релацията m ≤ n, произведението ще съдържа поне един елемент от редицата от цели числа {m, m+1, m+2, …, n}. Това прави възможно използването на оператора do/while.

Задача 39. **Да се напише програма, в резултат от изпълнението на която се изяснява, има ли сред числата от серията: $i^3 - 3.i + n^3$, i = 1, 2, …, n, число, кратно на 5. Ако има, да се изведе true, иначе – false.**

Решението на тази задача изисква последователно да се конструират елементите от серията числа. Това да продължава до намиране на първото число, кратно на 5, или до изчерпване на редицата без число с указаното свойство да е намерено.

Програма Zad39_1.cpp е едно решение на задачата.

```
// Program Zad39_1.cpp
#include <iostream.h>
int main()
{cout << "n= ";
int n;
cin >> n;
if (!cin)
{cout << "Error. Bad Input! \n";
```

```
return 1;
}
if (n <= 0)
{cout << "Incorrect Input! \n";
return 1;
}
int i = 0;
int a;
do
{i++;
a = i*i*i - 3*i + n*n*n;
} while (a%5 != 0 && i < n);
if (a%5 == 0) cout << "true\n";
else cout << "false\n";
return 0;
}
```

Възникват два въпроса:

1) Ако условието (a%5 != 0 && i < n) стане лъжа, тъй като се излиза от цикъла, правилно ли следващият оператор определя резултата?

От законите на де Морган следва, че след излизане от цикъла е в сила a%5 == 0 || i == n. Ако a%5 == 0, тъй като a е i-тия елемент на серията и i ≤ n, наистина в серията съществува

елемент с исканото свойство. Ако a%5 не е 0, следва че i == n ще е в сила, т.е. a е n – тият елемент и за него свойството не е в сила. Тъй като са сканирани всички елементи от серията, в нея наистина не съществува елемент с указаното свойство.

2) Ще се стигне ли до състояние, при което горното условие наистина ще е лъжа, т.е. ще завършили изпълнението на оператора за цикъл do/while?

Условието (a%5 != 0 && i < n) ще е лъжа, ако a%5 == 0 || i == n е в сила и се достига или когато в серията има елемент с търсеното свойство, или е сканирана цялата серия и i указва последния й елемент. Ако в серията няма елемент с исканото свойство, тъй като i е инициализирано с 0 и се увеличава с 1 на всяка стъпка от изпълнението на оператора do/while, в един момент ще стане вярно условието i = n, т.е. цикълът ще завърши изпълнението си.

Друго решение дава програмата Zad39_2.cpp. В нея е пропуснат фрагментът, въвеждащ стойност

на променливата n, тъй като е същия като в програма Zad39_1.cpp.

```
// Program Zad39_2.cpp
#include <iostream.h>
int main()
{ …
int i = 1;
int a;
do
{a = i*i*i - 3*i + n*n*n;
i++;
} while (a%5 != 0 && i <= n);
if (a%5 == 0) cout << "true\n";
else cout << "false\n";
return 0;
}
```

Лошото при това решение, че в тялото на цикъла има разминаване на елемента от серията, запомнен в a, и поредния му номер.

Задача 40. Да се напише програма, която въвежда естествено число и установява, дали цифрата 5 участва в записа на числото.

Програма Zad40.cpp решава задачата.

```
// Program Zad40.cpp
#include <iostream.h>
int main()
{cout << "n= ";
int n;
cin >> n;
if (!cin)
{cout << "Error. Bad Input! \n";
return 1;
}
if (n <= 0 )
{cout << "Incorrect Input! \n";
return 1;
}
int d;
do
{d = n % 10;
n = n / 10;
} while (d != 5 && n != 0);
if (d == 5) cout << "true\n";
else cout << "false\n";
return 0;
```

}

В тялото на цикъла последователно се намират цифрата на единиците на числото n и числото без цифрата на единиците си. Това продължава докато поредната цифра е различна от 5 и останалото, след задраскването на цифрата на единиците, число е различно от 0.

Задача 41. Нека a е неотрицателно реално число. Да се напише програма, която приближено пресмята квадратен корен от a по метода на Нютон.

Упътване: (метод на Нютон) Дефинира се

$$x_0 = 1$$

$$x_{i+1} = \frac{1}{2}(x_i + \frac{a}{x_i}), i = 0, 1, 2, ...$$

редица от реални числа x_0, x_1, x_2, x_3, … по следния начин:

Сумирането продължава докато абсолютната стойност на разликата на последните два констуирани елемента на редицата стане по-малка от ε, $\varepsilon>0$, е дадено достатъчно малко реално число.

Програма Zad41.cpp решава задачата.

```
// Program Zad41.cpp
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
int main()
{cout << "a= ";
double a;
cin >> a;
if (!cin)
{cout << "Error! Bad Input! \n";
return 1;
}
if (a < 0)
{cout << "Incorrect Input! \n";
return 1;
}
cout << "eps= ";
double eps;
cin >> eps;
if (!cin)
{cout << "Error! Bad Input! \n";
return 1;
```

```
}
if (eps <= 0 || eps > 0.5)
{cout << "Incorrect Input! \n";
return 1;
}
double x0;
double x1 = 1;
do
{x0 = x1;
x1 = 0.5*(x0 + a/x0);
} while (fabs (x1-x0) >= eps);
cout << setprecision(6) <<
setiosflags(ios :: fixed);
cout << "sqrt(" << a << ")= " << setw(10)
<< x1 << "\n";
return 0;
}
```

4.5.4. Вложени оператори за цикъл

Тялото на който и да е от операторите за цикъл е произволен оператор. Възможно е да е оператор за цикъл или блок, съдържащ оператор

за цикъл. В тези случаи се говори за вложени оператори за цикъл.

Пример: Програмният фрагмент

```
for (int i = 1; i <= 3; i++)
  for (int j = 1; j <= 5; j++)
cout << “(“ << i << “, “ << j << “)\n”;
```

съдържа вложен оператор for и се изпълнява по следния начин: Променливата i получава последователно целите стойности 1, 2 и 3. За всяка от тези стойности, променливата j получава стойностите 1, 2, 3, 4 и 5 и за тях се изпълнява операторът

```
cout << “(“ << i << “, “ << j << “)\n”;
```

В резултат се конструират и извеждат на отделни редове всички двойки от вида (i, j), където i = 1, 2, 3 и j = 1, 2, 3, 4, 5. След този фрагмент обаче, променливата i е видима (може да се използва), докато променливата j е невидима (не може да се използва, освен ако не се дефинира повторно).

При влагането на цикли, а също при използването на блокове, възникват проблеми,

свързани с видимостта на дефинираните променливи.

Област на променлива

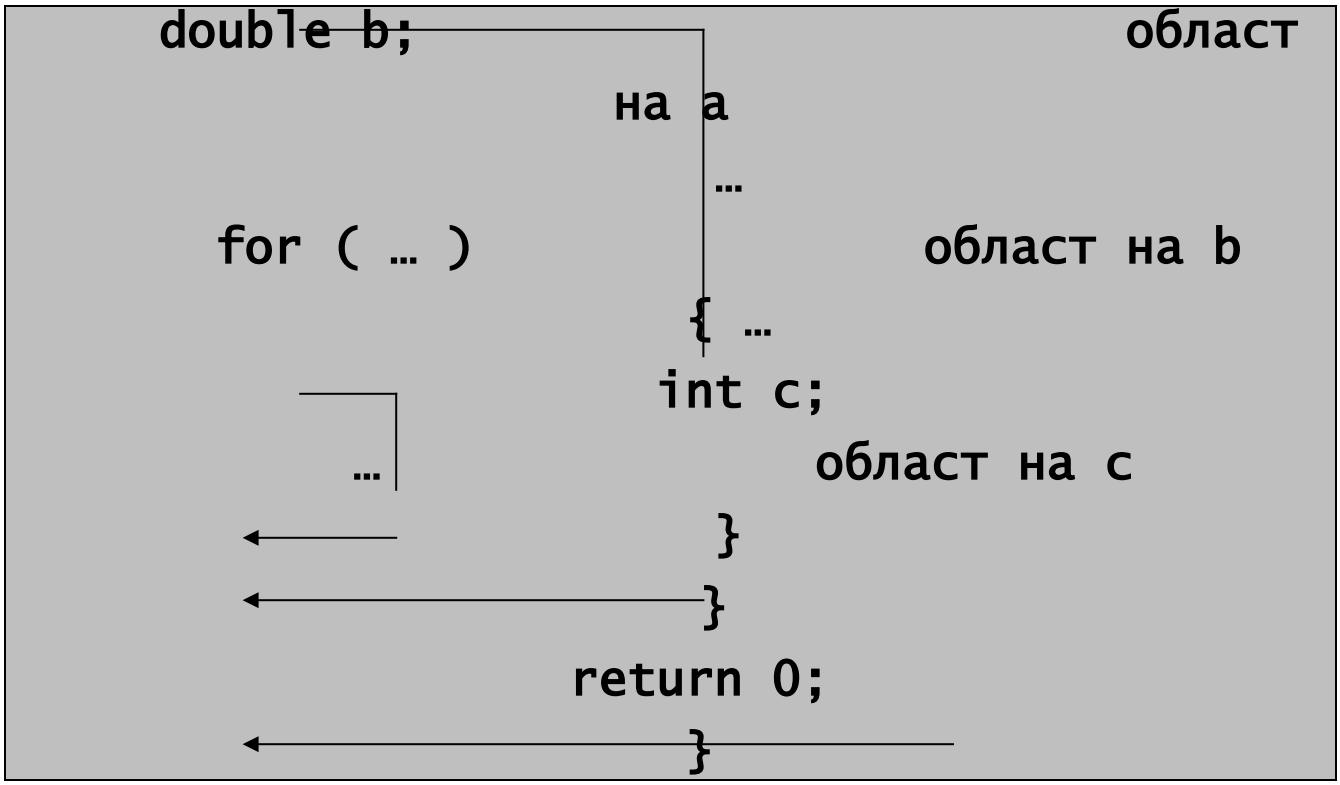
Общото правило за дефиниране на променлива е, дефиницията й да е възможно най-близко до мястото където променливата ще се използва най-напред.

Областта на една променлива започва от нейната дефиниция и продължава до края на блока (оператора), в който променливата е дефинирана.

На фиг. 4.12 за променливите a, b и c са определени областите им.

Пример за области на променливи

```
int main()
    {…
     double a;
     …
     for ( … )
     {…
```



Фиг. 4.12 Пример за области на променливи

Всяка променлива е видима – може да се използва в областта си. Променливите, от примера на Фиг. 4.12, b и c не могат да се използват навсякъде в тялото на main, а само в означените части.

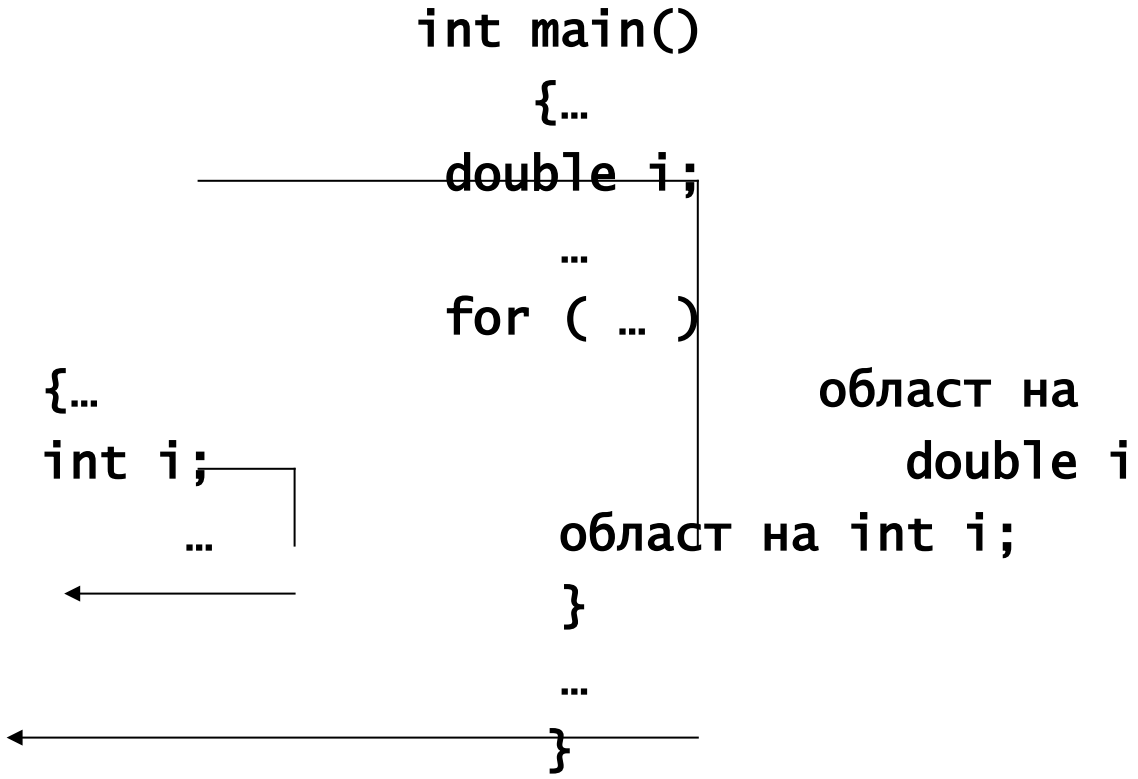
Променлива, дефинирана в някакъв блок, се нарича локална променлива за блока.

Променлива, дефинирана извън даден блок, но така, че областта й включва блока, се нарича нелокална променлива за този блок.

Възниква въпросът: ***Може ли променливи с еднакви имена да бъдат дефинирани в различни блокове на програма?***

Ако областите на променливите не се препокриват, очевидно няма проблем. Ако обаче те са вложени една в друга, пак е възможно, но е реализирано следното правило: локалната променлива "скрива" нелокалната в областта си.

Пример:



Според правилото, в тялото на оператора for е видима цялата променлива i (локална за тялото), а не нелокалната double i.

Ако това води до конфликт с желанията ви, преименувайте например локалната за тялото на for променлива int i.

Задачи върху вложени оператори за цикъл

Задача 42. **Да се напише програма, която намира всички решения на деофантовото уравнение a1.x1 + a2.x2 + a3.x3 + a4.x4 = a, където a1, a2, a3, a4 и a са дадени цели числа, а неизвестните x1, x2, x3 и x4 приемат стойности от интервала [p, q] (p и q са дадени цели числа, p<q).**

Програма Zad42.cpp решава задачата.

```
// Program Zad42.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{cout << "a1= ";
int a1;
cin >> a1;
if (!cin)
{cout << "Error. Bad Input!\n";
```

```
        return 1;
            }
        cout << "a2= ";
            int a2;
           cin >> a2;
            if (!cin)
{cout << "Error. Bad Input!\n";
            return 1;
            }
        cout << "a3= ";
            int a3;
           cin >> a3;
            if (!cin)
{cout << "Error. Bad Input! \n";
            return 1;
            }
        cout << "a4= ";
            int a4;
           cin >> a4;
            if (!cin)
{cout << "Error. Bad Input!\n";
            return 1;
            }
        cout << "a= ";
```

```
int a;
cin >> a;
if (!cin)
{cout << "Error. Bad Input!\n";
return 1;
}
cout << "p= ";
int p;
cin >> p;
if (!cin)
{cout << "Error. Bad Input!\n";
return 1;
}
cout << "q= ";
int q;
cin >> q;
if (!cin)
{cout << "Error. Bad Input!\n";
return 1;
}
if (p>=q)
{cout << "Incorrect Input!\n";
return 1;
}
```

```
for (int x1 = p; x1<= q; x1++)
 for (int x2 = p; x2 <= q; x2++)
  for (int x3 = p; x3 <= q; x3++)
   for (int x4 = p; x4 <= q; x4++)
   if (a1*x1 + a2*x2 + a3*x3 + a4*x4
            == a)
     cout << setw(5) << x1 << setw(5)
          << x2
        << setw(5) << x3 << setw(5)
     << x4 << "\n";
          return 0;
              }
```

Задача 43. Да се напише програма, която проверява дали *съществува* решение на деофантовото уравнение a1.x1 + a2.x2 + a3.x3 + a4.x4 = a в интервала [p, q], където a1, a2, a3, a4, a, p и q са дадени цели числа, p<q.

Програма Zad43.cpp решава задачата. В нея са пропуснати дефинициите и въвеждането на стойности на променливите a1, a2, a3, a4, a, p и q, тъй като са аналогични на тези от задача

42. Условието p<q прави подходящ оператора do/while.

```
// Program Zad43.cpp
#include <iostream.h>
int main()
{…
 if (p>=q)
{cout << "Incorrect Input!\n";
 return 1;
}
int x1 = p;
 bool b;
 do
{int x2 = p;
 do
{int x3 = p;
 do
 {int x4 = p;
 do
{b = a1*x1 + a2*x2 + a3*x3 + a4*x4 == a;
 x4++;
} while (!b && x4 <= q);
 x3++;
} while (!b && x3 <= q);
```

```
            x2++;
        } while (!b && x2 <= q);
            x1++;
    } while (!b && x1 <= q);
    if (b) cout << "yes\n";
      else cout << "no\n";
          return 0;
              }
```

Задачи

Задача 1. Явява ли се условен оператор редицата от символи:

а) `if (x < y) x = 0; else y = 0;`

б) `if (x > y) x = 0; else cin >> y;`

в) `if (x >= y) x = 0; y = 0; else cout << z;`

г) `if (x < y) ; else z = 5;`

д) `if (x < y < z) then z = z + 1;`

е) `if (x != y) z = z+1; x = x + y;`

Задача 2. Кое условие е в сила след запазената дума else на условния оператор:

а) `if (a > 1 && a < 5) b = 5; else b = 10;`

б) `if (a < 1 || a > 5) b = a; else a = b;`

в) `if (a == b || a == c || b == c) c = a + b; else c = a – b;`

Задача 3. Да се намерят грешките в следните оператори:

а)
```
if (1 < x < 2) x = x + 1; y = 0;
   else x = 0; y = y + 1;
```

б)
```
if (1 < x) && (x < 2)
   {x = x + 1;
    y = 0;
    };
```

```
else
{x = 0;
 y = y + 1;
};
```

Задача 4. Да се напише програма, която по дадено реално число x намира стойността на y, където

$$y = \begin{cases} 0, & x \le 0 \\ x^2, & x > 0. \end{cases}$$

Задача 5. Да се напише програма, която по зададени стойности на реалните променливи a, b и c намира:

а) min{a+b+c, a.b.c} + 15.2

б) max{a^2 - b^3 + c, a – 17.3 b, 3.1 a + 3.5 b – 8 c} - 17.9.

Задача 6. Да се напише програма, която увеличава по-малкото от две дадени цели неравни числа пет пъти, а по-голямото число намалява 8 пъти.

Задача 7. Да се напише програма, която въвежда четири реални числа и ги извежда във възходящ (низходящ) ред върху екрана.

Задача 8. Дадени са три числа a, b и c. Да се напише програма, в резултат от изпълнението на която, ако е в сила релацията a ≥ b ≥ c, числата се удвояват, в противен случай числата се заменят с техните абсолютни стойности.

Задача 9. Да се намери стойността на z след изпълнението на операторите

```
z = 0;
if (x > 0) if (y > 0) z = 1; else z = 2;
```

ако:

а) x = y = 1 б) x = 1, y = -1 в) x = -1, y = 1

Задача 10. Да се запише указаното действие чрез един условен оператор:

а) $d = \begin{cases} \max\{a, b\}, & x < 0 \\ \min\{a, b\}, & x \ge 0 \end{cases}$

б) d = max(a, b, c)

Задача 11. Да се напише условен оператор, който е еквивалентен на оператора за присвояване

```
x = a || b && c;
```

където всички променливи са булеви и в който не се използват логически операции (Например, операторът x = not a; е еквивалентен на оператора if (a) x = false; else x = true;).

Задача 12. Да се напише оператор за присвояване, еквивалентен на условния оператор

if (a) x = b; else x = c;

(всички променливи са булеви).

Задача 13. Да се напише програма, която по зададено число a, намира корена на уравнението f(x) = 0, където

$$f(x) = \begin{cases} 5xa^{\frac{1}{3}} + |a-1|^{\frac{1}{2}}, & a > 0 \\ e^{ax} - a^2 - 5, & a \le 0. \end{cases}$$

Задача 14. Да се напише програма, която по дадено реално число x намира стойността на y:

а) y = (...(((x + 2)x + 3)x + 4)x +... + 10)x + 11

б) y = (...(((11x + 10)x + 9)x + 8)x + ... + 2)x + 1.

Задача 15. Да се напише програма, която намира сумата от кубовете на всички цели числа, намиращи се в интервала (x + lnx, x^2 + 2x + e^x), където x > 1.

Задача 16. Дадено е естественото число n (n ≥ 1). Да се напише програма, която намира броя на тези елементи от серията числа i^3 - 7.i.n + n^3 , i = 1, 2, ..., n, които са кратни на 3 или на 7.

Задача 17. Да се напише програма, която по дадено реално число x намира стойността на сумата:

а) y = sinx + $sinx^2$ + $sinx^3$ + ... + $sinx^n$;

б) y = sinx + sin^2x + sin^3x + ... + sin^nx;

в) y = sinx + sinsinx + sinsinsinx + ... + $\underbrace{\text{sinsin ...sinx}}_{\text{n пъти}}$

Задача 18. Да се напише програма, която намира

$$\sqrt{1 + \sqrt{3 + \sqrt{5 + \ldots \sqrt{97 + \sqrt{99}}}}}$$

Задача 19. Да се напише програма, която по дадено естествено число n (n ≥ 1) намира стойността на f:

а) f = (2n)!! = 2.4.6. ... .2n;

б) f = (2n-1)!! = 1.3.5. ... .(2n-1);

в) f = n!!.

Задача 20. Дадено е естествено число n (n ≥ 1). Да се напише програма, която пресмята сумата:

а) $\left(\frac{1}{1}\right)^n + \left(\frac{1}{2}\right)^n + \ ... + \left(\frac{1}{n}\right)^n$ б) $\left(\frac{1}{1}\right)^1 + \left(\frac{1}{2}\right)^2 + \ ... + \left(\frac{1}{n}\right)^n$

в) $\left(\frac{1}{1}\right)^n + \left(\frac{1}{2}\right)^{n-1} + \ ... + \left(\frac{1}{n}\right)^1$

(Да не се използват функциите exp и log).

Задача 21. Да се напише програма, която извежда в нарастващ ред всички трицифрени естествени числа, които не съдържат еднакви цифри (/ и % да не се използват).

Задача 22. Да се напише програма, която намира и извежда броя на точките с цели координати, попадащи в кръга с радиус R (R > 0) и център - координатното начало.

Задача 23. Да се напише програма, която извежда таблицата на истинност за булевата функция f = (a and b) or not (b or c) в следния вид

a	b	c	f
true	true	true	true
true	true	false	true
	.	.	.
false	false	false	true

Задача 24. Да се напише програма, която извежда върху екрана следните таблици:

```
1                 1 2 3 4 5 6 7         1 2 3 4 4 3 2 1
1 2                 1 2 3 4 5 6           2 3 4 4 3 2
1 2 3                 1 2 3 4 5             3 4 4 3
1 2 3 4                 1 2 3 4               4 4
1 2 3 4 5                 1 2 3               3 4 4 3
1 2 3 4 5 6                 1 2             2 3 4 4 3 2
1 2 3 4 5 6 7                 1           1 2 3 4 4 3 2 1
```

Задача 25. Дадено е естествено число n (n ≥ 1). Да се напише програма, която намира и извежда първите n елемента от серията числа:

$$\left(1 + \frac{1}{i}\right)^i, \; i = 1, 2, 3, \ldots$$

(Да не се използват функциите exp и log).

Задача 26. Едно естествено число е съвършено, ако е равно на сумата от своите делители (без самото число). Например, 6 е съвършено, защото 6 = 1+2+3. Да се напише програма, която намира всички съвършени числа ненадминаващи дадено естествено число n.

Задача 27. Да се напише програма, която намира всички трицифрени числа от интервала [m, n], на които като се задраска цифрата на десетиците, намаляват цяло число пъти (m и n са дадени естествени числа, m < n).

Задача 28. Да се напише програма, която намира всички четирицифрени числа от интервала [m, n], на които като се задраска цифрата на стотиците, се делят на 11 (m и n са дадени естествени числа, m < n).

Задача 29. Да се напише програма, която намира всички цели числа от интервала [m, n], в записа на които участва цифрата 5 (m и n са дадени естествени числа, m < n).

Задача 30. Да се напише програма, която намира всички цели числа от интервала [m, n], цифрите на които образуват намаляваща редица (m и n са дадени естествени числа, m < n).

Задача 31. Да се напише програма, която намира всички цели числа от интервала [m, n], цифрите на които са различни (m и n са дадени естествени числа, m < n).

Задача 32. Дадено е естествено число n (n > 1). Да се напише програма, която намира всички прости числа от интервала [2, n].

Задача 33. Да се напише програма, която намира всички прости делители на дадено естествено число n.

Задача 34. **Да се напише програма, която по**

$$S = 1 + \frac{a.x}{1!} + \frac{a(a-1)}{2!}.x^2 + \ldots + \frac{a(a-1)\ldots(a-n+1)}{n!}x^n + \ldots$$

дадени реални числа x, a и ε (ε>0), приближено пресмята сумата.

Събирането да продължи докато бъде добавено събираемо, абсолютната стойност на което е по-малка от ε (ε > 0 е дадено достатъчно малко реално число).

Задача 35. **Да се напише програма, която намира броя на цифрите в десетичния запис на дадено естествено число.**

Задача 36. **Да се напише програма, която проверява дали дадено естествено число е щастливо, т.е. едно и също е при четене отляво надясно и отдясно наляво.**

Задача 37. **Да се напише програма, която проверява дали сумата от цифрите на дадено естествено число е кратна на 3.**

Задача 38. **Да се напише програма, която намира всички естествени числа, ненадминаващи дадено естествено число n, които при преместване на първата им (най-лявата) цифра**

най-отзад, се увеличават k пъти (k е дадено естествено число, k > 1).

Задача 39. Да се напише програма, която намира всички естествени числа от интервала [m, n], на които като се задраска k-тата цифра (отляво надясно), намаляват цяло число пъти (m, n и k са дадени естествени числа, m < n).

Задача 40. Да се напише програма, която намира всички естествени числа от интервала [m, n], на които като се задраска k-тата цифра (надясно наляво), намаляват цяло число пъти (m, n и k са дадени естествени числа, m < n).

Задача 41. Числата на Фибоначи се дефинират по следния начин:

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

Да се напише програма, която намира сумата на числата на Фибоначи от интервала [a, b] (a и b са дадени естествени числа, a<b).

Задача 42. Проверете дали са в сила релациите:

a) fib(1) + fib(2) + … + fib(n) = fib(n+2)-1;

б) $fib(1) + fib(3) + \ldots + fib(2n-1) = fib(2n)$;

в) $fib^2(1) + fib^2(2) + \ldots + fib^2(n) = fib(n).fib(n+1)$

за всяко цяло число n от интервала [1, 100].

Задача 43. Проверете дали е в сила релацията $fib(n+m) = fib(n-1).fib(m) + fib(n).fib(m+1)$ за всяка двойка цели числа (n, m) от интервала [1, 100].

Задача 44. Проверете дали са в сила свойствата:

а) Число на Фибоначи е четно тогава и само тогава, когато поредният му номер се дели на 3.

б) Число на Фибоначи се дели на 3 тогава и само тогава, когато поредният му номер се дели на 4.

в) Число на Фибоначи се дели на 4 тогава и само тогава, когато поредният му номер се дели на 6.

г) Число на Фибоначи се дели на 5 тогава и само тогава, когато поредният му номер се дели на 5.

д) Число на Фибоначи се дели на 7 тогава и само тогава, когато поредният му номер се дели на 8.

е) Число на Фибоначи се дели на 16 тогава и само тогава, когато поредният му номер се дели на 12.

За целта използвайте числата на Фибоначи в интервала [1, 1000].

Задача 45. Проверете дали ако q е прост делител на fib(n) и q е различен от естественото число p, то fib(n.p)/fib(n) не се дели на q.

Задача 46. За естествените числа n и m операцията oo се определя по следния начин: n oo m = 3n + 7m + 10n%m. Да се напише програма, която намира всички двойки (n, m) от естествени числа, за които е в сила n oo m = m oo n (m и n са естествени числа от интервала [a, b]).

Задача 47. За естествените числа n и m операцията oo се определя по следния начин: n oo m = 3n + 7m + 10n%m. Да се напише програма, която проверява дали съществува двойка (n, m) от естествени числа, за която е в сила

релацията n оо m = m оо n (m и n са естествени числа от интервала [a, b]).

Допълнителна литература

1. К. Хорстман, Принципи на програмирането със C++, София, СОФТЕХ, 2000.
2. Ст. Липман, Езикът C++ в примери, "КОЛХИДА ТРЕЙД" КООП, София, 1993.
3. **B. Stroustrup, C++ Programming Language. Third Edition, Addison – Wesley, 1997.**