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

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

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 бъдат реализирани 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 използването му.

Задача 26. Да се напише програма, която по зададено 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 реализира повтаряне 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 скобки 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 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, 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 отново е истина (3 <= 3), 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 стойността на условието i<=n не е в сила (в случая 4).

На Фиг. 9 е дадено детайлно описание на оператора 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**.

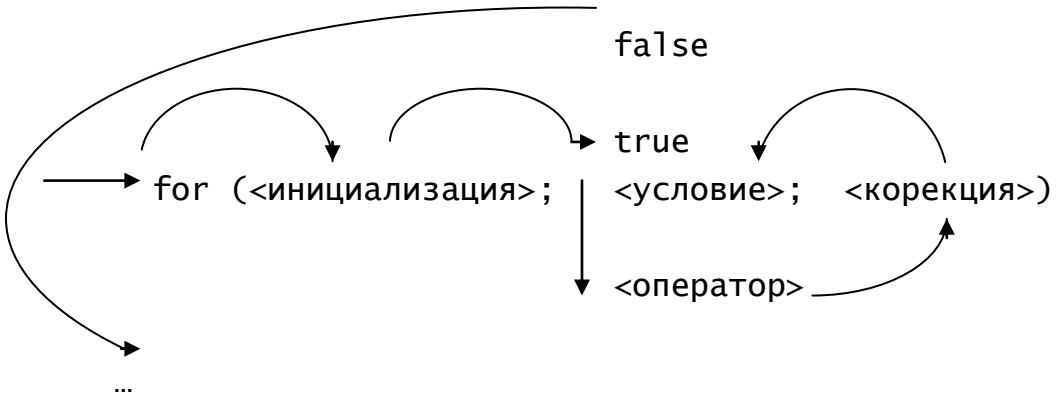
Семантика

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. 9.

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 1). 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 2).

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 3). Това обаче е лош стил на програмиране и ние 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.

Примери:

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

2.
```
for (int i = 1; i<= n;)
{fact = fact* i;
 i++;
}
```

3.
```
for (int i = 1; ; i++)
 if (i > n) break;
 else fact = fact* i;
```

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 блока, в който се намира оператора 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.

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;
}
```

Забележете, че тази програма илюстрира използването на оператора for за реализиране на итеративни циклични процеси. 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 зададени 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, $n \geq 1$, $m > 1$. Да се напише програма,която определя броя на елементите от серията числа

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

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 ($\varepsilon > 0$), прилижено 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 $|x1| < \varepsilon$, 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 $|x1| \geq \varepsilon$, повтаряне на

```
{ конструиране на ново събираемо x1;
  s = s + x1;
}
```

За реализирането на тези действия, 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 скобки 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 стойността на булевия израз fabs(x1) >= eps и тъй като тя е true, се изпълнява тялото на цикъла, след което 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 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. 10 е дадено описание на синтаксиса и семантиката на оператора.

Синтаксис

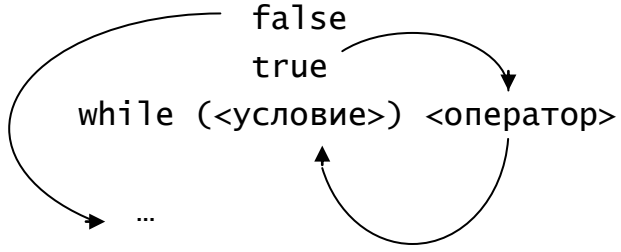
while (<условие>) <оператор>

където

- while (докато) 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. 10.

Забележки:

1. Ако е необходимо да се изпълнят могократно 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 ние 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 е задача 32.

Задачи върху оператора while

Задача 33. Да се напише програма, която по зададено естествено число, намира факториела му. За целта да се използва оператора 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 дума, която не започва със цифра.

Програма Zad35.cpp решава задачата.

```
Program Zad35.cpp
#include <iostream.h>
int main()
{int count = 0;
 int number;
 double average = 0;
 cout << "> ";
 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 го заменим с него. 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 се промени типът на average от double на int?

Задача 36. Да се напише програма, която пресмята приближено следната безкрайна сума

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

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 = x;
  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 пресмята приближено следната безкрайна сума

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

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 спиране 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 x1;
  double x2 = x;
  double s = x;
  int i = 2;
  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 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 скобки и определя условието за завършване изпълнението на цикъла.

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, има стойност 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. 11.

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 на цикъла. Огражда се в кръгли скобки.

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. 11.

Забележки:

1. Между запазените думи do и while стои точно един оператор. Ако няколко действия трябва да се извършат, 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 x1 не е дефиниран на линия:

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

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 ние 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 $m \leq n$. 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 продължава до намиране на първото число, кратно на 5, или до изчерпване на редицата без число с това 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 условие (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 е истина a%5 == 0 || i = n. Ако a%5 == 0, тъй като a е i-тият 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. Но тъй като са сканирани всички елементи от серията, наистина в нея не съществува елемент с търсеното свойство.

2) Ще се стигне ли до състояние, при което горното условие наистина ще е лъжа?

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 програмата, в един момент ще стане вярно условието i = n, т.е. цикълът ще завърши изпълнението си.

Друго решение дава програмата Zad39_2.cpp. То не съдържа фрагментът, въвеждащ стойност на променливата n.

```
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 != 5n && 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 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 редица от реални числа x_0, x_1, x_2, x_3, … по следния начин:

$$x_0 = 1$$

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

Сумирането да продължи докато абсолютната стойност на разликата на последните два констуирани елемента на редицата е 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 << "Bad Input! \n";
  return 1;
 }
 if (a < 0)
 {cout << "Incorrect Input! \n";
  return 1;
 }
 cout << "eps= ";
 double eps;
 cin >> eps;
 if (!cin)
 {cout << "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 кой де 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”;
```

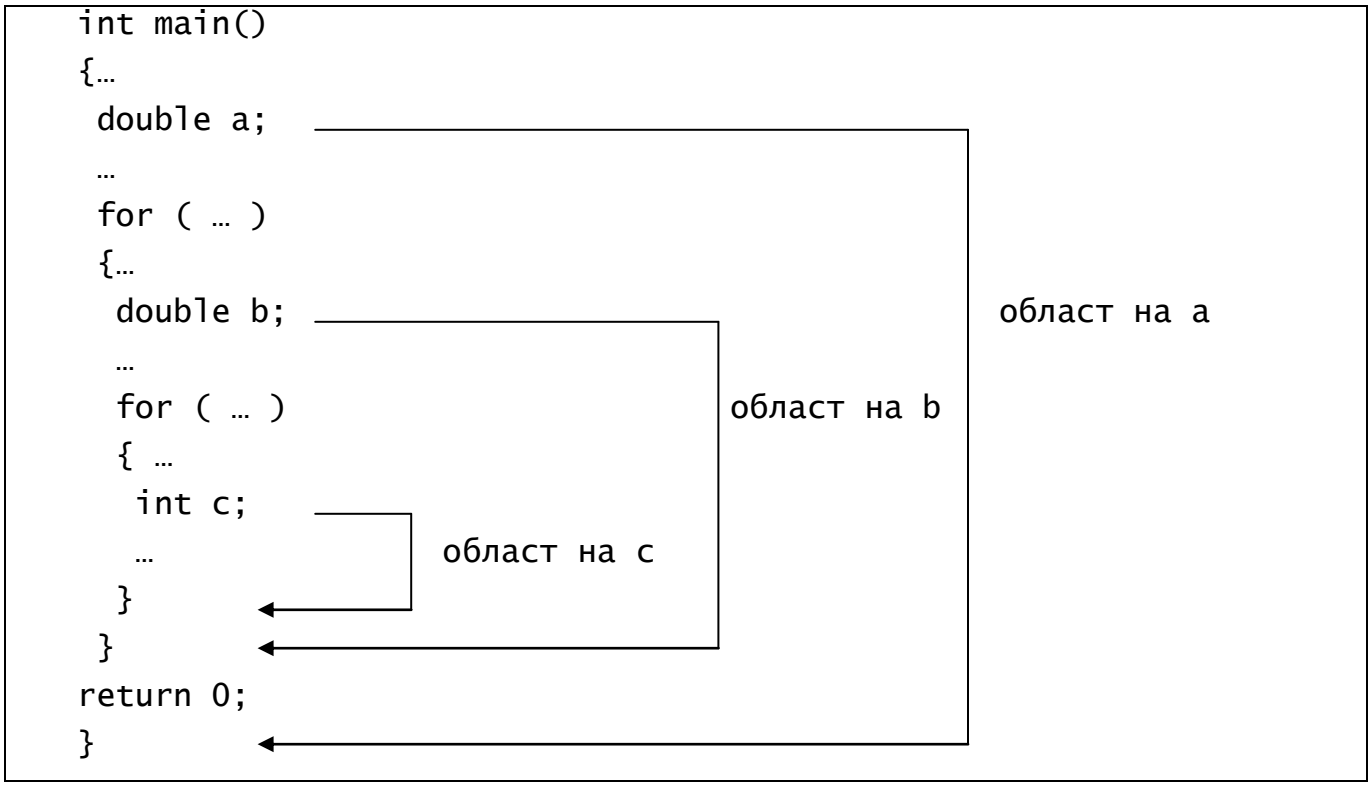
В резултат се конструират и извеждат на отделни редове всички двойки от вида (i, j), където i = 1, 2, 3 и j = 1, 2, 3, 4, 5.

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, в който променливата е дефинирана. На Фиг. 12 за променливите a, b и c са определени областите им.



Фиг. 12

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

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

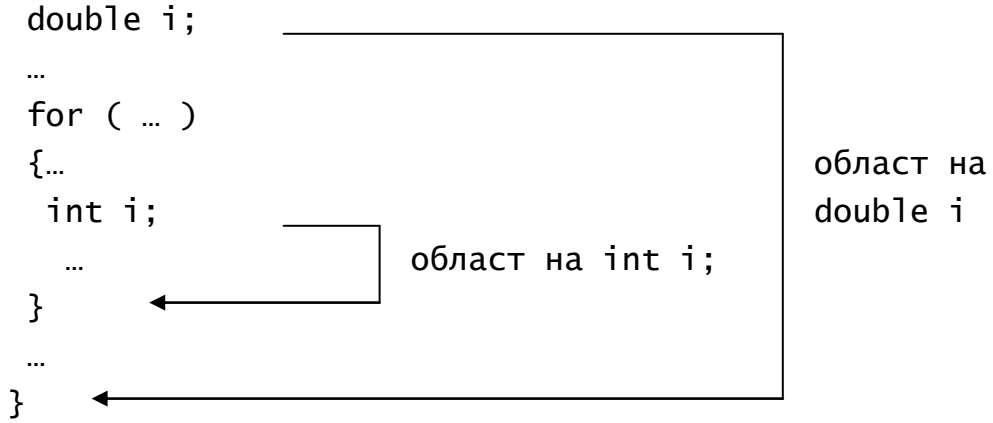
Всяка променлива е видима – може да се използва в областта си. Така b и c не могат да се използват навсякъде в тялото на main, а само с означените области.

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

Ако областите на променливите не се припокриват, 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:

```
int main()
{…
```



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 << "Error!\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. Ще пропуснем дефинициите на променливите a1, a2, a3, a4, a, p и q. Те 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 << "Error!!\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. 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 2. Да се напише програма, която намира сумата от кубовете на всички цели числа, намиращи се в интервала ($x + \ln x$, $x^2 + 2x + e^x$), където x > 1.

Задача 3. 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 4. Да се напише програма, която по дадено реално число x, намира стойността на сумата

а) $y = \sin x + \sin x^2 + \sin x^3 + ... + \sin x^n$;

б) $y = \sin x + \sin^2 x + \sin^3 x + ... + \sin^n x$;

в) $y = \sin x + \sin\sin x + \sin\sin\sin x + ... + \underbrace{\sin\sin ...\sin x}_{n \text{ пъти}}$

Задача 5. Да се напише програма, която намира

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

Задача 6. 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 7. Дадено е естественото число n (n ≥ 1). Да се напише програма, която пресмята сумата:

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

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

(Да не се използват функциите exp и log).

Задача 8. Да се напише програма, която извежда в нарастващ ред всички трицифрени естествени числа, които не съдържат еднакви цифри (/ и % да не се използват).

Задача 9. 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 10. 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 11. Да се напише програма, която извежда върху екрана следната таблица:

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

Задача 12. Дадено е естествено число n (n ≥ 1). Да се напише програма, която намира и извежда първите n елемента от серията числа

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

(Да не се използват функциите exp и log).

Задача 13. 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 14. 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 15. 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 16. Да се напише програма, която намира всички четирицифрени числа от интервала [m, n], в записа на които участва цифрата 5 (m и n са дадени естествени числа, m < n).

Задача 17. Да се напише програма, която намира всички четирицифрени числа от интервала [m, n], цифрите на които образуват намаляваща редица (m и n са дадени естествени числа, m < n).

Задача 18. Да се напише програма, която намира всички петцифрени числа от интервала [m, n], цифрите на които са различни (m и n са дадени естествени числа, m < n).

Задача 19. Дадено е естествено число n (n > 1). Да се напише програма, която намира всички прости числа от интервала [2, n].

Задача 20. Да се напише програма, която намира всички прости делители на дадено естествено число n.

Задача 21. Да се напише програма, която по дадени реални числа x, a и ε,ε>0, приближено пресмята сумата.

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

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

Задача 22. Да се напише програма, която намира броя на цифрите в десетичния запис на дадено естествено число.

Задача 23. 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 24. Да се напише програма, която проверява дали сумата от цифрите на дадено естествено число е кратна на 3.

Задача 25. Да се напише програма, която намира всички естествени числа, ненадминаващи дадено естествено число n, които при преместване на първата им цифра най-отзад, се увеличават k пъти (k е дадено естествено число, k > 1).

Задача 26. 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 27. 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 28. За дадено естествено число да се провери дали цифрите му, гледани отляво надясно, образуват монотонно растяща редица.

Задача 29. За дадено естествено число да се провери дали цифрите му са различни.

Задача 30. Да се намерят всички прости делители на дадено естествено число.

Задача 31. 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).

Задача 32. За естествените числа n и m операцията ++ се определя по следния начин: n oo m = n + m + n%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 33. За естествените числа n и m операцията ++ се определя по следния начин: n oo m = n + m + n%m. Да се напише програма, която проверява дали съществува двойкА (n, m) от естествени числа, за която е в сила релацията n oo m = m oo n (m и n са естествени числа от интервала [a, b]).

Допълнителна литература

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