

Увод в програмирането

Основни структури за
управление на
изчислителния процес

Оператор за присвояване

■ 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

Оператор за присвояване ...

■ Примери:

...

```
double a = 1.5;
```

...

```
double a = a + 5.1;
```

...

```
double a = 1.5;
```

...

```
a = a + 34.5;
```

...

```
a = 0.5 + sin(a);
```

...

Оператор за присвояване ...

Допълнения

1. Операторът за присвояване е реализиран като *дясноасоциативен* инфиксен аритметично-логически оператор с приоритет по-нисък от този на дизюнкцията `||`.

`<променлива> = <израз>;`

е израз от тип – типа на `<променлива>`
и стойност – стойността на `<израз>`

Оператор за присвояване ...

Допълнения

■ *Пример:*

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

Оператор за присвояване ...

Допълнения

2. Ако е в сила дефиницията:

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

допустим е операторът:

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

3. Оператори ++ и --

- ☐ Реализирани са като префиксни и като постфиксни оператори. Единственият им аргумент е променлива.
- ☐ ++ увеличава стойността на променливата с 1
- ☐ -- намалява стойността на променливата с 1

Оператор за присвояване ...

Допълнения

- Стойността на **a++** е първоначалната (неувеличена) стойност на **a**
- Стойността на **++a** е увеличената с 1 стойност на **a**
- Аналогично за **a--** и **--a**

Оператор за присвояване ...

Допълнения

■ Пример

```
int a = 5;
cout << a << "\n";           // 5
cout << a++ << "\n";          // 5 и увеличава a
cout << a << "\n";           // 6
a = 5;
cout << a << "\n";           // 5
cout << ++a << "\n";          // увеличава с 1 и 6
cout << a << "\n";           // 6
```


Оператор за присвояване ...

Допълнения

4. Съкратени форми

$a = a + 2;$ $a += 2;$

$a = a - 2;$ $a -= 2;$

$a = a * 2;$ $a *= 2;$

По-общо:

$e1 \text{ op} = e2;$

е еквивалентно

$e1 = (e1) \text{ op } (e2);$

където op е някой от операторите

$+, -, *, /, \%, \ll, \gg, \&, |, ^$

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

Блок

■ Синтаксис

```
{<оператор1>  
  <оператор2>  
  . . .  
  <операторn>  
}
```

■ 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.

Блок ...

■ *Пример:*

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

Условни оператори

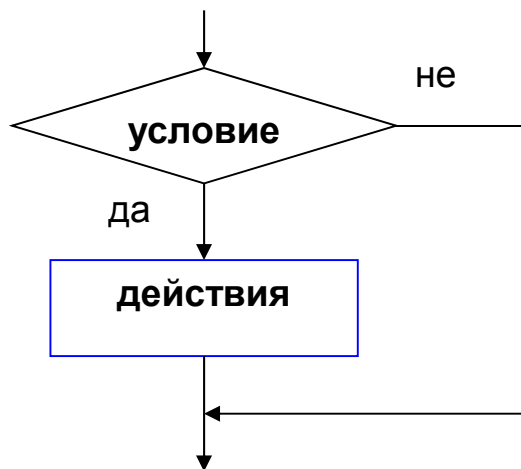
■ Условни оператори

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

Условен оператор if

- Чрез този условен оператор се реализира разклоняващ се изчислителен процес



Условен оператор if ...

■ **Синтаксис**

if (<условие>) <оператор>

където

- if 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



```
#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;
```

```
}
```


Оператор if ...

...

```
cout << "x= ";
```

```
double x;
```

```
cin >> x;
```

```
if (!cin)
```

```
{ cout << "Error. Bad input! \n";
```

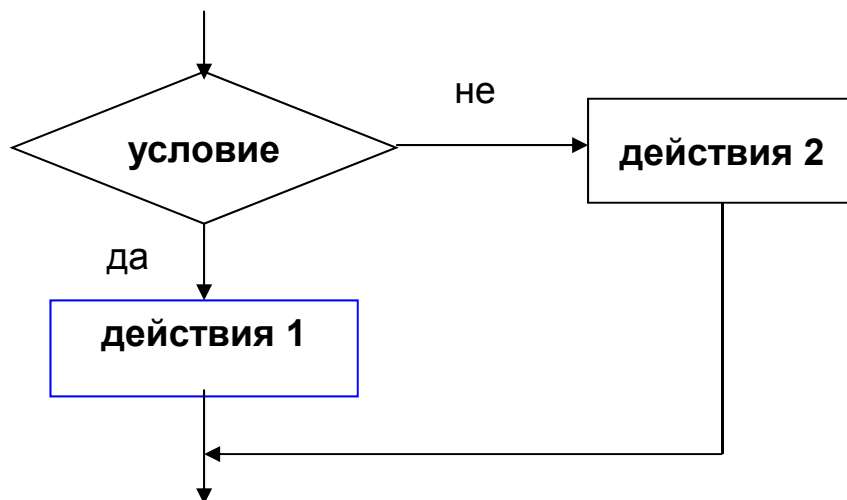
```
    return 1;
```

```
} // въведена е валидна стойност за x
```

...

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 вида



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>

Оператор if/else ...

```
double a, b;
```

```
...
```

```
double min;
```

```
if (a < b)
```

```
    min = a;
```

```
else
```

```
    min = b;
```

```
cout << "min = " << min << "\n";
```

```
...
```

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*

Вложени условни оператори ...

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

a)

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

b)

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

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

Вложени условни оператори ...

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

```
#include <iostream.h>
int main()
{ cout << "i= ";
  int i;
  cin >> i;
  if (!cin)
  { cout << "Error, bad input!\n";
    return 1;
  }
}
```

Вложени условни оператори ...

```
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;
}
```


Оператор switch

...

```
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";
}
```

...

Оператор switch ...

- Операторът switch реализира избор на вариант от множество варианти (възможности)

- *Синтаксис*

switch (<израз>)

{case <израз₁> : <редица_от_оператори₁>

case <израз₂> : <редица_от_оператори₂>

...

case <израз_{n-1}> : <редица_от_оператори_{n-1}>

[default : <редица_от_оператори_n>]

}

Оператор switch ...

Семантика

- Намира се стойността на switch-израза.
- Получената константа се сравнява последователно със стойностите на етикетите <израз1>, <израз2>, ... 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

Оператор break

- Операторът break принадлежи към групата на т. нар. **оператори за преход**. Тези оператори предават управлението безусловно в някаква точка на програмата
- *Синтаксис*
break;
- *Семантика*
Прекратява изпълнението на най-вътрешния съдържащ го оператор switch или оператор за цикъл. Изпълнението на програмата продължава от оператора, следващ (съдържащ) прекъснатия

Оператор break ...

...

```
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";
}
```

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.

Оператори за цикъл ...

индуктивен цикличен процес

- 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

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

Оператори за цикъл ...

индуктивен цикличен процес

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

а) конструиране на събираемо

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

б) добавяне на събираемото към S .

Оператори за цикъл ...

итеративен цикличен процес

- 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

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

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

Оператори за цикъл ...

итеративен цикличен процес

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

а) конструиране на събираемо

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

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 .

Оператори за цикъл ...

- 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

Оператори за цикъл ...

Оператор *for*

- Използва се основно за реализиране на индуктивни изчислителни процеси.

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

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

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

Оператори за цикъл ...

Оператор *for*

- общ шаблон:

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

за $i = 1, 2, \dots, n$

- Решение:

```
#include <iostream.h>
```

```
int main()
```

```
{ cout << "n= ";
```

```
  int n;
```

```
  cin >> n;
```

```
  if (!cin)
```

```
  {   cout << "Error. Bad Input! \n";
```

```
      return 1;
```

```
  }
```

Оператори за цикъл ...

Оператор *for*

```
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;
}
```

Оператори за цикъл ...

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 един или няколко оператора, **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**.

Оператори за цикъл ...

Оператор *for*

■ 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.

Оператори за цикъл ...

Оператор *for*

■ 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.

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

Оператори за цикъл ...

Оператор *for*

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

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>;

Оператори за цикъл ...

Оператор *for*

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

4. Възможно е и тялото да е празно:

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

Оператори за цикъл ...

Оператор *for*

5. 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.

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

6. Област на променливите, дефинирани в заглавната част на for

Оператори за цикъл ...

Оператор *while*

- Чрез този оператор може да се реализира произволен цикличен процес
- *Пример:* По дадени реални числа x и $\varepsilon > 0$, да се намери сумата

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

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

Оператори за цикъл ...

Оператор *while*

- В тази задача броят на повторенията предварително не е известен, а зависи от условието $|x_1| < \varepsilon$, където с x_1 е означено произволно събираемо. Стъпки:
 - Въвеждане на стойности на x и ε .
 - Инициализация $x_1 = 1; s = 1$.
 - Докато е в сила условието $|x_1| \geq \varepsilon$, повтаряне на действията
{конструиране на ново събираемо x_1 ;
 $s = s + x_1$;
}
 - Извеждане на S .

Оператори за цикъл ...

Оператор *while*

```
#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;
   }
```

Оператори за цикъл ...

Оператор *while*

```
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;
}
```


Оператори за цикъл ...

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**.

Оператори за цикъл ...

Оператор *while*

■ 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

Оператори за цикъл ...

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.
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**.

Оператори за цикъл ...

Оператор *while*

4. Операторът

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

е еквивалентен на

```
{ <инициализация>  
while (<условие>)  
{<оператор>  
  <корекция>;  
} }
```

Оператори за цикъл ...

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.

Оператори за цикъл ...

Оператор *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);
```

Оператори за цикъл ...

Оператор *do/while*

3. Следствие разширената интерпретация на true и false, частта <условие> може да е аритметичен израз.
4. Операторът do/while завършва с ;.

```
int n = 0;
do
{
    cin >> n;
    if (!cin)
    { cout << "Error!\n";
      return 1;
    }
    ...
}
while (n != 0);
```


Оператори за цикъл ...

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:*

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

Оператори за цикъл ...

Област на променлива

- Общото правило за дефиниране на променлива е, дефиницията ѝ 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.

Оператори за цикъл ...

Област на променлива

```
int main()  
{  
  ...  
  double a;  _____  
  ...  
  for ( ... )  
  {  
    ...  
    double b;  _____  
    ...  
    for ( ... )  
    {  
      ...  
      int c;  _____  
      ...  
    }  
  }  
  return 0;  
}
```

The diagram illustrates the scope of variables in the provided C code. It uses horizontal lines and arrows to show the lifetime of each variable:

- Variable `a`:** A horizontal line starts at the declaration `double a;` and extends to the right, ending with an arrow pointing to the closing brace of the `main` function. This indicates that `a` is in scope for the entire duration of the `main` function.
- Variable `b`:** A horizontal line starts at the declaration `double b;` and extends to the right, ending with an arrow pointing to the closing brace of the outer `for` loop. This indicates that `b` is in scope for the entire duration of the outer loop.
- Variable `c`:** A horizontal line starts at the declaration `int c;` and extends to the right, ending with an arrow pointing to the closing brace of the inner `for` loop. This indicates that `c` is in scope only for the duration of the inner loop.

Оператори за цикъл ...

Област на променлива

- 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**.
- 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**.

Оператори за цикъл ...

Област на променлива

```
int main()
```

```
{ ...
```

```
  double i; _____
```

```
  ...
```

```
  for ( ... )
```

```
  { ...
```

```
    int i; _____
```

```
    ...
```

```
  } ←
```

```
  ...
```

```
}
```

```
←
```