

Лекция 5.

Структурирани абстракции за контрол

*Как в програмите задаваме
избор между варианти,
повторения и др.?*

Структурни оператори в ЕП

Структурните оператори (управляващи структури, оператори-контроли, *control statements*) — такива оператори, които в своята структура съдържат други оператори, които от своя страна се наричат **вложени оператори**



По този начин с помощта на съставните оператори могат да се конструират йерархии от оператори, съдържащи се един в друг

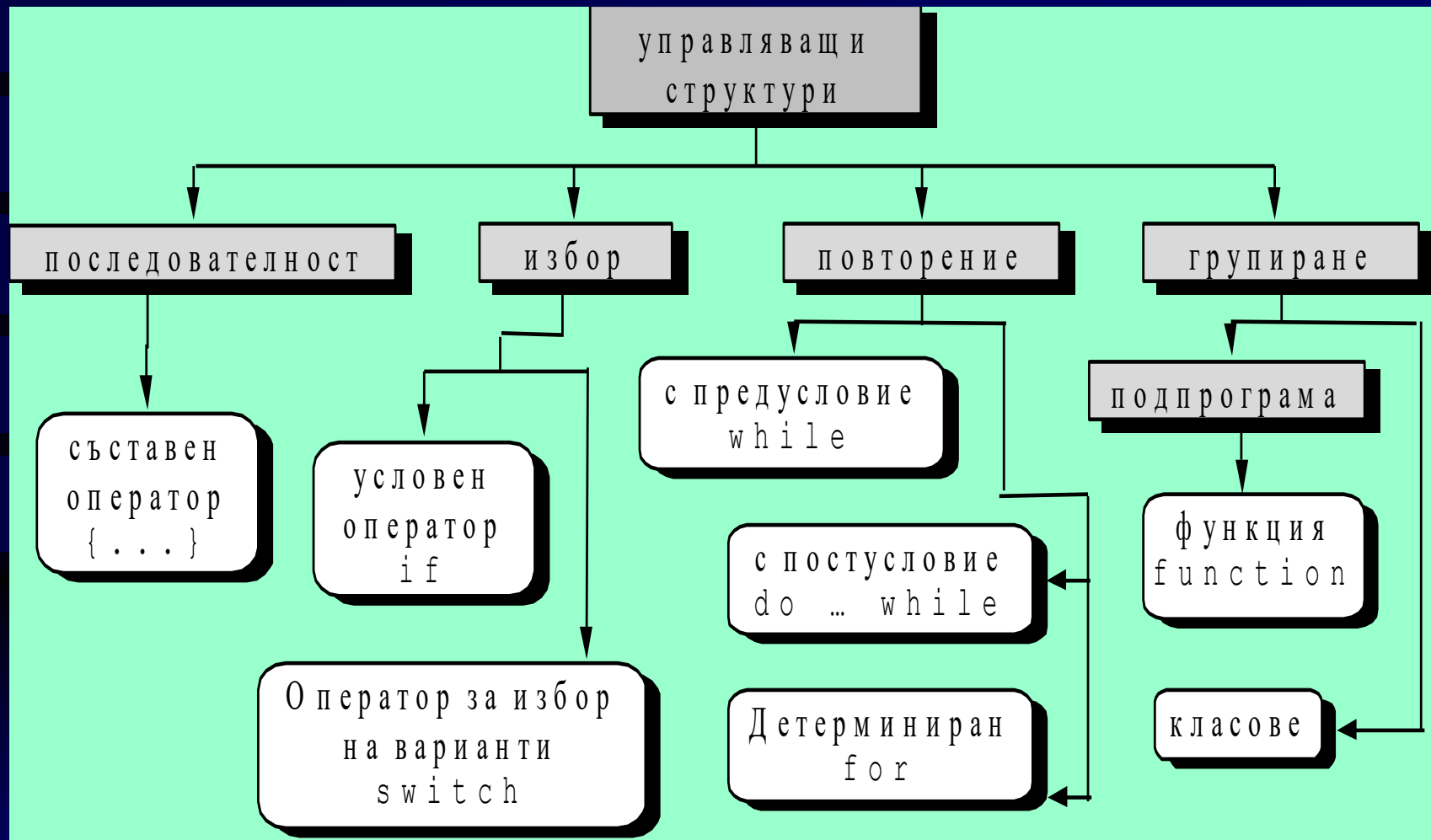


Всеки един от операторите в йерархията е разположен на определено **ниво на вложеност** - от най-външно, до най-вътрешно

Видове структурни оператори:

- **Последователност**
- **Избор** при изпълнението на една или няколко стъпки
- **Повторение** при изпълнението на една или няколко стъпки
- **Групиране** — служат за групиране на множество от оператори или данни (ще бъдат разгледани по-късно)

Оператори в езика C++. Структурни оператори



Оператори за последователност: Съставен оператор – АМ и ЗМ

Абстрактен модел

Съставният оператор обединява линейно подредено множество от оператори, които задават определена цел при обработката на данните и се изпълняват последователно по реда на написването им

Знаков модел в C++

Съставният оператор се нарича още **блок**

- **Синтаксис:**


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

- **Семантика:**

Изпълняват се последователно операторите между { и } по реда на срещането им

- **Примери:**

```
{  
    D=sqrt (D) ;   X1=(-B+D) / (2*A) ;   X2=(-B-D) / (2*A) ;  
}
```

 Тъй като в C++ декларациите се считат за оператори, то списъкът с оператори между { и } може да съдържа декларации. В резултат на това всяко име декларирано в даден блок е валидно (видимо) само в неговите рамки (е локално за блока, има **локална област на действие**)

Оператори за последователност: Оператор за последователно остойносттаване* – 3М

- **Семантика:** Операторът за последователно остойносттаване (“,”, *comma operator, sequential-evaluation*) остойносттава своите операнди последователно отляво надясно
- **Приложение:** Обикновено се използва с цел да се остойността два (или повече) изрази, там където синтактично е позволено записването само на един израз
- **Примери (функцията изисква 3 аргумента, вторият има стойност 3):**

```
// пример за оператор запетая  
FunctionOne(a, (b=1, b+2), c);
```

⚠ Не бива да се забравя, че “,” има и друга употреба – просто като разделител

```
int i = 10, b = 20, c = 30; //разделител  
i = b, c;  
cout<< i; //извежда 20 (защо)  
i = (b, c);  
cout<< i; //извежда 30 (защо)
```

Управляващи структури за избор

Управляващите структури за избор определят дали даден оператор (или последователност от оператори) трябва да бъде изпълнен или не, обикновено чрез изследване стойността на някакъв израз

- **Приложение:**

В случаи, при които това коя част от обработката ще се извърши зависи от конкретните данни (напр. ако е необходимо да се направи статистика за много хора в зависимост от техния пол, местожителство, възраст и др. В такъв случай в зависимост от данните се извършват различни изчисления)

- **Видове оператори за избор в C++:**

- **условен оператор** (оператор `if`) – изборът се прави като се изследва стойността на логически израз
- **оператори за избор на варианти** (оператор `switch...case`) – изборът е въз основа на стойността на дискретен израз

Условен оператор

Абстрактен модел

Този оператор дава възможност за условно изпълнение на някакво действие или избор между две независими действия според стойността на логически израз

Знаков модел в C++

- **Синтаксис:**

```
(1)    if (<израз>)
        <оператор1>;
        else
        <оператор2>;
```

```
(2)    if (<израз>)
        <оператор1>;
```

- **Семантика:** Ако стойността на <израз> е различна от нула, <оператор₁> се изпълнява. Ако стойността на <израз> е нула:
 - при (1) се изпълнява <оператор₂> (след `else`)
 - при (2) просто се продължава изпълнението на програмата със следващия оператор

Условен оператор – примери

(1) кратка форма

```
if((exam_mark > 4) && (assignment_mark > 4))  
    good_student++;
```

(2) със съставен оператор

```
if((exam_mark > 4) && (assignment_mark > 4)) {  
    cout << "*";  
    good_student++;  
}
```


(3) пълна форма

```
if(gender_tag == 'F')  
    females++;  
else  
    males++;
```

(4) вложени условни оператори

```
if(mark<4) {  
    S_Count++;  
    cout << "Good job!" << endl;  
}  
else  
    if(mark<5)  
        M_Count++;  
    else  
        if(mark<6)  
            O_Count++;
```


Условен оператор – особености

- синтаксисът изисква един единствен оператор, както след `if`, така и след `else`
-  операторът след `if` завършва с `”;` само ако не е съставен (виж примерите)
- при вложени оператори `if` клаузата `else` се свързва винаги с най-близкия предхождащ `if` от същото ниво на вложеност (ако искаме да избегнем действието на това правило е необходимо да използваме допълнителен съставен оператор):

```
if(mark<3) {  
    if((exam_mark == 0) && (assignment_count < 2))  
        cout << "Check for cancelled enrollment" << endl;  
}  
else  
    if(mark<4) ...
```
- в `if` не може да се използва оператора за последователно остойносттаване (`“,”`)

Условен оператор-израз*

Условният оператор-израз (*Conditional-Expression Operator*) се използва за изчисляване на точно един от два възможни изрази.

Той е три-аргументен

- **Синтаксис:**

`<израз> ? < израз 1> : < израз 2>;`

, където <израз> е от прост тип

- **Семантика:**

Ако стойността на <израз> е различна от нула, < израз ₁> се остойностява. Ако стойността на <израз> е нула, то се остойностява < израз ₂>. Така получената стойност е резултатът от изпълнението на условния оператор-израз

- **Примери:**

```
salary = (Employ_Status == 'G') ? 600 : 450;
```

```
maximum=(temperature > maximum) ? temperature :maximum;
```

```
cout << ((gender_tag == 'f') ? "Female : " : "Male : ");
```

Оператор за избор на варианти в езика C++

Абстрактен модел

Този оператор определя изпълнението на едно от няколко възможни действия (варианта). В езика C++ тези действия не са независими, т.е. могат да бъдат изпълнени няколко от тях или нито едно

Знаков модел в C++

- **Синтаксис:**

```
switch( <израз> )  
{  
    case <константен израз> :незад ... <оператор>незад ; break;незад  
    ...  
    default: <оператор>незад  
}
```

,където:

- <израз> трябва да е дискретен (да може да се остойността като цяло число)
- всяка case клауза асоциира (съпоставя) цяло число със съответния вариант от възможни действия
- всеки <константен израз> е от типа на <израз>

- **Семантика:** Изчислява се стойността на <израз>, след което изпълнението продължава от оператора на този вариант, за който <константен израз> е равен на изчислената стойност. В случай, че тя не се съответства на никой вариант, то се изпълнява операторът след default (ако има такъв). Изпълнението на оператора продължава до неговия край (}) или до срещането на оператора break

Оператор за избор на варианти в езика C++ – примери

(1)

```
switch( c )  
{  
    case 'A':  
        cara++;  
    case 'a':  
        lettera++;  
    default :  
        total++;  
}
```

(2)

```
switch( i )  
{  
    case -1:  
        n++;  
        break;  
    case 0 :  
        z++;  
        break;  
    case 1 :  
        p++;  
        break;  
}
```

Особености:

- Във вариантите на оператора `switch`, освен в последния не е позволено да се извършва инициализация на променливи

Управляващи конструкции за повторение (цикли)

Операторите за повторение служат за задаване на циклични процеси (цикли, итерации, *loops, iterations*)

- **Особености на циклични процеси:**

- Те осигуряват неколkokратно повторение на някакви действия, които се наричат **тяло** на цикъла
- Задължително изискване е циклите да бъдат **крайни**, т.е. броят на повторенията да е крайно число. Това изискване се осигурява от два специални компонента на циклите:
 - условие за край на цикъла – задава кога настъпва край на повторенията
 - управление на броя на повторенията – т.е. цикълът трябва да включва такова действие, което влияе върху стойността на условието за край на цикъла
- За да се осигури правилността на повторенията, в повечето случаи се налага да се извършат някои предварителни действия, които се наричат **инициализация на цикъла**

- **Основни елементи на итеративните процеси:**

- инициализация
- проверка на условието за край на повторенията
- управление на повторенията
- повтарящо се действие (тяло)

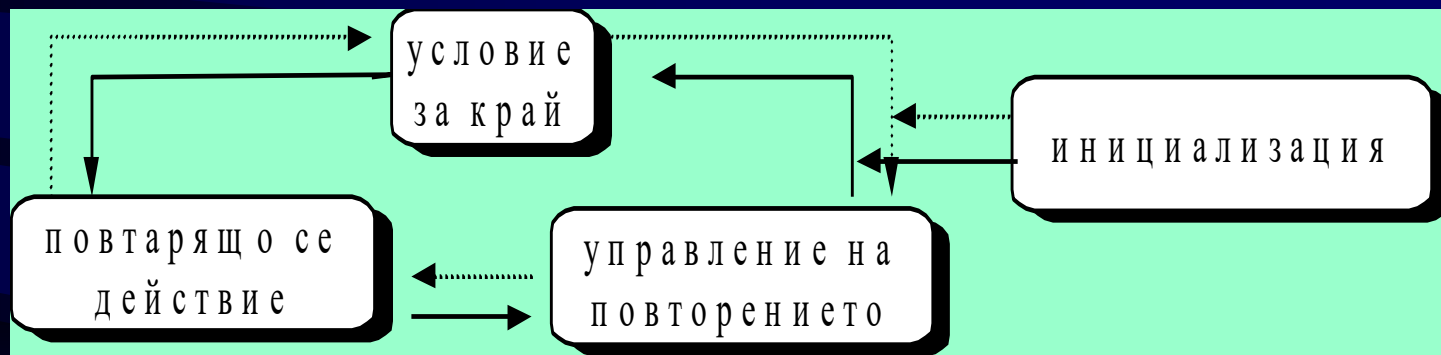
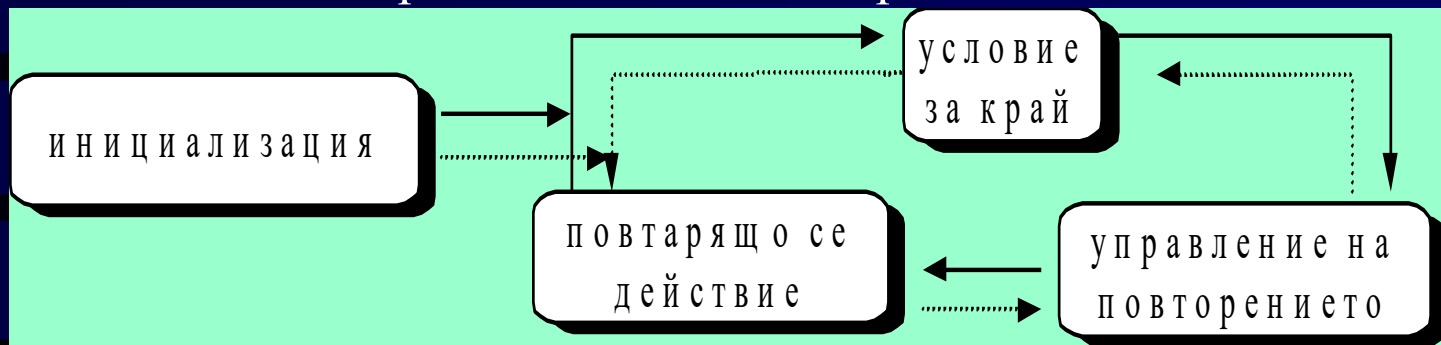
Управляващи конструкции за повторение – пример

Основни елементи на цикъла от алгоритъма на Евклид за намиране на НОД:

Цикъл	Основни елементи
Стъпка 2. Въведете и запомнете числата a и b .	Инициализация
Стъпка 3. Ако $a \neq b$, изпълнете ст. 4, в противен случай – ст. 6.	Проверка на условието за край на повторението
Стъпка 4. Ако $a > b$, то изчислете $a - b$ и го запомнете като a , в противен случай изчислете $b - a$ и го запомнете като b .	Управление на повторенията
Стъпка 5. Изпълнете ст. 3.	Повтарящо се действие

Управляващи конструкции за повторение – схема

Основните елементи на цикличните процеси могат да се срещат в различна последователност в конкретни цикли, което на следващите схеми е изобразено със стрелки по посока и обратно на часовниковата стрелка:



🔊 Повечето ЕП предоставят различни видове циклични инструкции, които най-често могат да се отнесат към следните два вида:

- оператори за цикъл, при които повторенията на тялото могат да се преброят (обикновено се нарича “for” цикъл)
- оператори за цикъл, при които тялото се изпълнява отново и отново, докато се изпълни някакво условие (обикновено се нарича “while” цикъл)

Оператори за цикъл в C++

В C++ има следните основни форми на циклични конструкции:

– **Цикли с условие:**

- с предусловие (`while`)
- със следусловие (`do ... while`)

– **Цикъл, при които повторенията на тялото могат да се преброят (`for`), т.е. както се нарича **детерминиран цикъл** (*counting, deterministic loop*), но може да се използва и с по-общо предназначение**

Оператори за повторение в C++: Цикъл с предусловие

Абстрактен модел

Абстрактният модел на тази конструкция представлява итерация, край на повторенията, на която се определя от някакво условие (логическо), а това условие **се изследва преди** всяко повторение на тялото на цикъла

Знаков модел

- **Синтаксис:**

```
while (<израз>) <оператор>;
```

, където <израз> е от целочислен тип и определя условието за край на цикъла, а <оператор> е кой да е оператор от езика и представлява **тялото** на цикъла. Ако тялото се състои от повече от един оператор, то те се обединяват в един съставен оператор

- **Семантика:**

Изчислява се стойността на <израз> и ако е true (≠0) се изпълнява <оператор>, а ако е false (=0) се прекратява изпълнението на цикъла

Цикъл с предусловие – примери и особености

- Примери:

(1)


```
x = a;  
while (fabs(x - a / x) > 0.001)  
    x = 0.5 * (x + a / x);
```


(2)


```
s = 0; i = 1;           // инициализация  
while (i <= 5)           // проверка на условието за край  
{                       // тяло  
    s = s + i;  
    i = i + 1           // управление на повторенията  
}
```

(3)

```
while (true)  
    cout<<"Безкраен цикъл";
```

 условието за край се проверява преди изпълнението на тялото на цикъла, което означава, че **тялото на цикъла while може да не се изпълни нито веднъж**

 обикновено се налага за <израз> да се извърши някаква начална инициализация преди влизане в цикъла, за да може да се изчисли неговата стойност

 управлението на повторенията е част от тялото на цикъла. Ето защо в тялото на цикъла `while` трябва да се съдържа поне един оператор, който влияе върху условието за край. В противен случай изпълнението на цикъла никога няма да завърши или ще се получи т.н. безкраен цикъл (зацикляне) – **прекрат. Ctrl+C**

Цикъл с предусловие – примерна програма

- **Условие:** Да се изчисли стойността на израз, зададен чрез последователното въвеждане на редица от реални числа разделени с един от знаците: +, -, *, /. За край на израза служи въвеждането на знака за операция =. Приоритета на операциите при изчисляването на израза се определя от реда им на въвеждане и е по намаляване (напр. $2 * 3 - 1 / 5 = 1$)
- **Проектиране:**
 - (1) **Представяне на данните:** Аргументите и резултатът на операциите са реални числа, а знакът на операциите е символ
 - (2) **Алгоритмизация:**
 - Алгоритъмът представлява цикличен процес, при който на всяка стъпка по дадени аргументи и знак на операцията се изчислява нейната стойност
 - Знакът на операцията може да бъде един от четирите +, -, *, /, то при изчисляването имаме избор между един от четири варианта. Логично следва да използваме операторът за избор на варианти
 - При първата стъпка на разглежданата итерация е необходимо да се въведат двата аргумента на операцията, а при всички останали, като първи аргумент служи резултатът от изчисление на предишната въведена операция
 - Условието за край на цикъла е (`Oper != '='`). Не бива да се забравя, че знакът "=" може да се въведе още като първа операция, при което стойността на израза е самият първи аргумент (напр. $2=2$). Това определя използването на оператора за цикъл с предусловие

Програмна реализация на примера

```
//програма изчисляваща израз
#include <iostream.h>
void main()
{
    double Res, Arg;
    char Oper;

    /*инициализация*/
    cin >> Res;    //въвеждане на първи аргумент
    cin >> Oper;    //въвеждане знак на операция
    /*цялостно въвеждане и изчисляване на изрази*/
    while (Oper != '=') {
        cin >> Arg;    //въведен е вторият аргумент
        switch (Oper) {    //изч. стойност на операция
            case '+': Res=Res+Arg;break;
            case '-': Res=Res-Arg;break;
            case '*': Res=Res*Arg;break;
            case '/': Res=Res/Arg;break;
        };
        cin >> Oper;    //въвеждане знак на операция
    };
    /*извеждане на резултата*/
    cout << Res;
```

Оператори за повторение в C++: Цикъл със следусловие

Абстрактен модел

Абстрактният модел на тази конструкция представлява итерация, край на повторенията, на която се определя от някакво условие (логическо), а това условие **се изследва след** всяко повторение на тялото на цикъла

Знаков модел

- **Синтаксис:**

`do <оператор> while (<израз>);`

, където <израз> е от целочислен тип и определя условието за **край на цикъла**, а <оператор> може да бъде кои да е оператор (може и съставен) от езика и представлява **тялото на цикъла**

- **Семантика:**

1. Изпълнява се <оператор>
2. Изчислява се стойността на <израз>. Ако тя е `true` (`≠0`), се повтарят отново стъпки 1 и 2, а ако е `false` (`=0`) се прекратява изпълнението на цикъла

Цикъл със следусловие – примери и особености

- **Примери:**

```
(1)
x = 10;           //инициализация
do {             //тяло
    y = x * x + 1;
    x--;          //управление на повторенията
} while ( x > 0 ); //проверка на условието за край
```

```
(2)
do {
    cout << "Въведи месец (от 1 до 12) → " ;
    cin >> Month;
} while ( (Month < 1) || (Month > 12) );
```

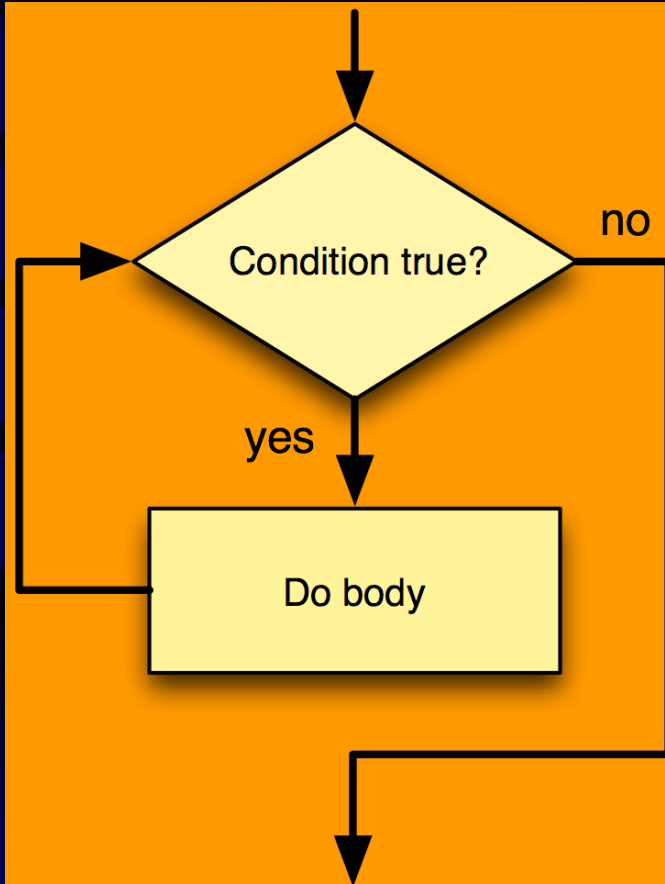
- **Особености:**

- **тялото на цикъла се изпълнява поне веднъж**

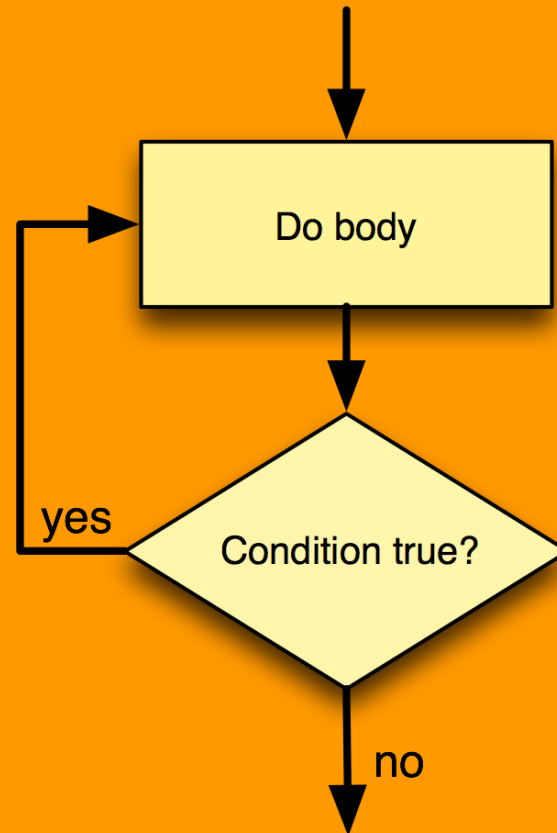
- управлението на повторенията е част от тялото на цикъла. Ето защо в тялото на цикъла трябва да се съдържа поне един оператор, който влияе върху условието за край. В противен случай изпълнението на цикъла никога няма да завърши или ще се получи т.н. безкраен цикъл (зацикляне)

- операторът се използва за организиране на циклични процеси, при които условието за край може да се провери след изпълнението на повтарящото се действие (например за проверка коректността на входните данни)

Различия в семантиката на while и do ...while



while flowchart



do/while flowchart

Оператори за повторение в C++: Детерминиран ЦИКЪЛ

Абстрактен модел

Абстрактният модел на тази конструкция **представлява итерация, повторенията, на която могат да се преброят**

Знаков модел

- **Синтаксис:**

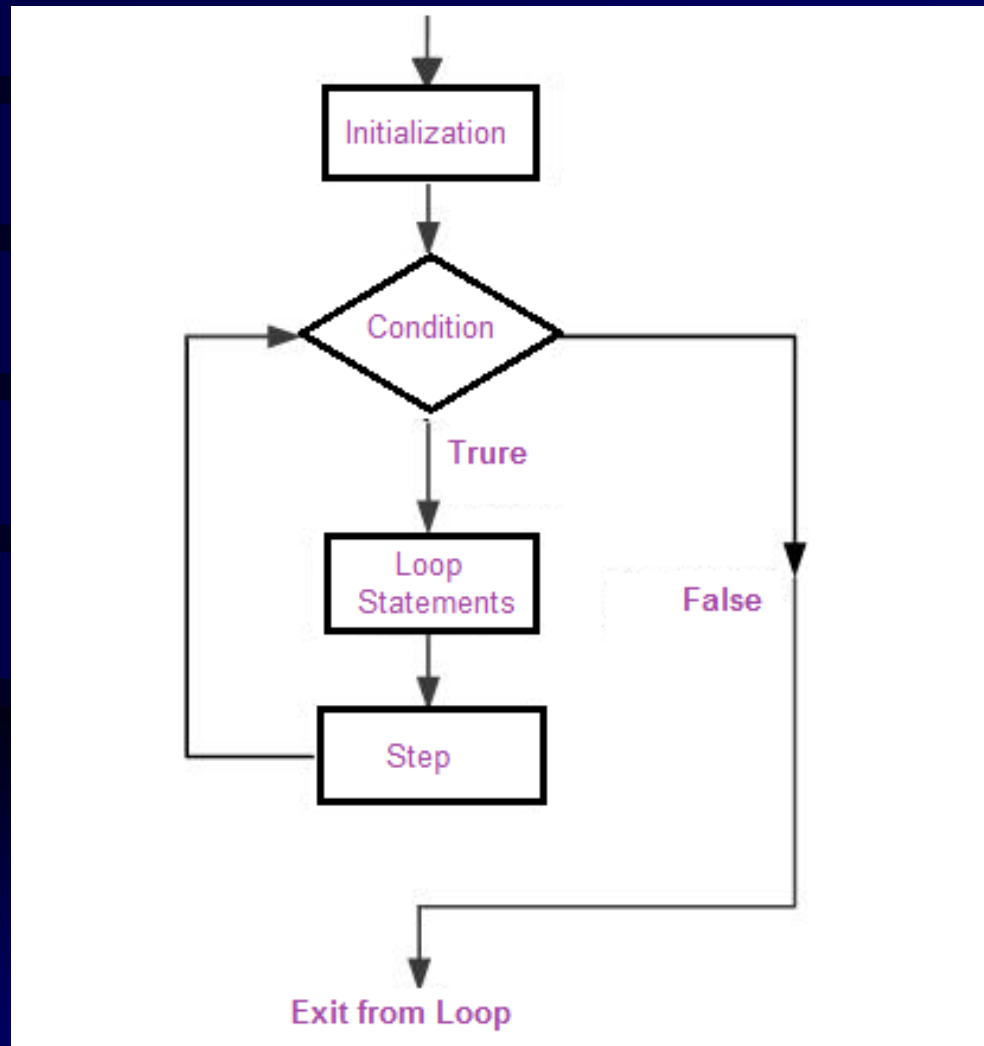
```
for (<инициализиращ_оператор>незад.; <израз1>незад.; <израз2>незад.) <оператор>;
```

- <инициализиращ_оператор> обикновено инициализира променлива брояч наричаща се **управляваща променлива на цикъла**. Той може да съдържа последователност от оператори и декларации разделени със запетая
- <израз₁> е целочислен израз, който обикновено се използва за проверка **на** условието за край на цикъла
- <израз₂> съдържа код, който обикновено променя текущата стойност на управляващата променлива (управление на повторението)
- <оператор> може да бъде кой да е оператор (може съставен) от езика и представлява тялото на цикъла

- **Семантика:**

1. Изпълнява се <инициализиращ_оператор>
2. Изчислява се <израз₁> и ако е true, то: изпълнява се <оператор>, изчислява се <израз₂> и отново се повтаря ст. 2, а ако <израз₁> се оцени като false, цикълът се прекратява

Оператори за повторение в C++: Детерминиран ЦИКЪЛ



Детерминиран цикъл – примери и особености

• Примери:

(1)

```
for (i = 0; i < 10; i++)  
    sum += i;
```

(2)

```
for (int j = 100; j > 0; j--) {  
    sum += j;  
    cin>> sum; }
```

(3)

```
for( ; ; )  
    cout<<"Безкраен цикъл";
```

(4)

```
for( int i = 0;  
i < 100;  
cout << ++i << endl );
```

• Особенности:

🔊 проверката за край на цикъла (т.е. дали управляващата променлива е достигнала крайната си стойност) се извършва преди изпълнението на тялото, което означава, че цикълът може да не се изпълни нито веднъж

🔊 променлива може да бъде декларирана в инициализиращата част на оператора `for`, като нейната област на видимост ще бъде до края на оператора или блока, в който се съдържа този `for`

🔊 <инициализиращ_оператор>, <израз₁> и <израз₂> не са задължителни (вж. пр. 3)

🔊 Въпреки че трите полета на оператора `for` са предназначени за инициализация, тестване за край и управление на повторението, това не е задължително (вж. пр. 4)

Оператори за повторение в C++ v.11

В допълнение на вече разгледаните оператори за цикъл във версия C++11 може да се използва и оператор за цикъл базиран на обхват (range-based for loop) представлява итерация, върху елементи в даден обхват (колекция, множество)

Знаков модел

- **Синтаксис:**

```
for (<декларация> : <израз>) <оператор>;
```

- < декларация > декларира променлива итератор с тип съответстващ на типа на елементите от колекцията
- <израз> е колекция от елементи

- **Семантика:**

1. Прави се оценка на колекцията (нейния обхват)
2. На всяка итерация на променливата итератор се присвоява поредния елемент от колекцията

```
vector<int> vec; vec.push_back( 10 ); vec.push_back( 20 );  
for (int i : vec )  
{  
    cout << i;  
}
```

Оператори за безусловен преход в C++:

`break` и `continue`

Операторите за безусловен преход **предизвикват незабавно предаване на управлението към конкретен оператор от програмата**

Оператор `break`

Оператор `break` се използват в комбинация с операторите за цикъл или с оператора `switch`

- **Семантика:** `break` прекратява изпълнението на цикъла (причинява изход от най-вътрешния цикъл) или на остатъка от блока `switch`. Предава управлението към следващия оператор

- **Пример:**

```
for( ; ; ) // няма условие за край
{
    if (sum >= 1000) break;
    sum += j; j++;}
cout << "Управлението се предава тук.\n";
```

Оператор `continue`

- **Семантика:** `continue` причинява следващата итерация (повторение) от цикъла да се стартира веднага, като се избегне останалата част от текущото повторение.

- **Пример:**

```
for (int j = 100; j > 0; j--) {
    if(j%2 == 0) continue;
    sum += j; }
```

Оператори за безусловен преход в C++:

`goto`

Операторът за безусловен преход `goto` може да се използва за да се организира цикъл, както и за прекратяване изпълнението на цикъл, но в по-голямата част от случаите той трябва да се избягва, защото нарушава целостта на структурата на програмата

- **Семантика:** `goto` предизвиква незабавно предаване на управлението към конкретен оператор от програмата, на който е съпоставен етикет. Операторът трябва да бъде от същата функция. Етикетът е идентификатор

- **Пример:**

```
int j = 100, sum = 0;  
LoopStart: sum += j;  
j--;  
if(j > 0) goto LoopStart;
```

Структурни оператори – други ЕП

- **Java** – няма съществена разлика нито по отношение на синтаксис нито от семантична гледна точка в сравнение с операторите на C++.
- **PHP** – има разлика в синтаксиса.

```
while (expr):  
    statement  
    ...  
endwhile;
```

- **Python** – разлика в синтаксиса.

```
if expression:  
    statement(s)  
else:  
    statement(s)
```