

Цикли – for, while, do...while

I. For loop

Забележка. $i += 2$ е синтаксис, еквивалентен на $i = i + 2$.

Аналогично за всички аритм. операции.

Итерация означава едно завъртане на цикъла.

инициализатор; i се нарича итератор;
казва как се казва променливата,
която ще брой колко пъти ни се е
завъртял цикълът

казва как да ни се променя променливата
на всяко завъртане на цикъла;
 $i++$ ($++i$, $i+=1$) означава, че се увеличава с 1;
 $i += 2$ -> увеличава се с 2;
 $i -= 3$ -> намалява се с 3;

```
for(int i = 0; i < 10; i++) {
```

условие;
докога ни се изпълнява
цикълът; числото, което
стои вдясно на знака < (или
>, или >= и т.н.) е броят
пъти, които ще ни се
изпълни цикълът, преди да
спре;

ако увеличаваме ($i++$) -> "инкрементира"
ако намаляваме ($i--$) -> "декрементира"

тяло на цикъла;
тялото се изпълнява
всеки път, когато се
завърта цикълът (когато
почне отначало цикълът)

...

```
}
```

Най-често за да направим n итерации използваме `for(int i = 0; i < n; i++)`, вместо `for(int i = 1; i <= n; i++)`. И при двете се правят n итерации, просто при първото броим от 0, а при второто - от 1. Причината да ползваме първото, а не второто, е особеност на броенето при индексите на масивите.

II. While loop

Синтаксис:

- `while(<условие>){`
 <тяло>;
}, където:

<условие> е булев израз, в който участва променлива от тялото на цикъла;

<тяло> е последователност от оператори; в <тяло> задължително се съдържа някаква промяна на променливата, по която итериране (или на условието, за което следим дали е изпълнено); иначе влизаме в т.нар. "безкраен цикъл";

```
int n = 3;  
while (n >= 0) {  
    cout << n * n << endl;  
    --n;  
}
```

Семантика:

- На всяка итерация се проверява дали условието е изпълнено.
- Ако условието е изпълнено, се изпълнява още веднъж <тяло>, където се променя стойността на променливата, която участва в <условие>.
- Ако условието не е изпълнено, <тяло> не се изпълнява и се излиза от цикъла.

III. Do...while loop

Синтаксис:

- do {
 <тяло>;
} while (<условие>;) където:
 - <условие> е булев израз, в който участва променлива от тялото на цикъла;
 - <тяло> е последователност от оператори; в <тяло> задължително се съдържа някаква промяна на променливата, по която итерираме (или на условието, за което следим дали е изпълнено); иначе влизаме в т.нар. "безкраен цикъл";

```
int n = 1;  
do {  
    n--; промяна на променливата от условието  
    cout << n;  
} while (n > 1; условие
```

Забележете, че след (<условие>) пишем „;“!

Семантика:

- Влиза в цикъла и изпълнява тялото.
- В тялото се променя променливата от условието.
- След като се изпълни тялото, се проверява дали е изпълнено <условие>.
- Ако условието е изпълнено, влиза в нова итерация на цикъла, т.е. тялото се изпълнява още веднъж.
- Ако условието не е изпълнено, <тяло> не се изпълнява повече и се излиза от цикъла.

IV. Разлика между while и do...while

При while първо прави проверката и ако се изпълнена, чак тогава изпълнява тялото. При do...while изпълнява първо тялото и накрая прави проверката.

Следователно тялото на do...while се изпълнява **винаги** поне веднъж!

Пр.:

```
int i = 1;  
do {  
    i--;  
    cout << i;  
} while (i > 1); //0, въпреки че условието не е изпълнено  
  
int i = 1;  
while (i > 1) {  
    i--;  
    cout << i;  
} //не изкарва нищо, защото условието не е изпълнено и въобще не влиза в тялото
```



V. Кога кой цикъл да използваме?

Уговорка: for и while са напълно взаимно заменяеми и навсякъде, където може да се използва единият, може да се ползва и другият. Do...while е малко по-различен, защото при него проверката на условието се случва след изпълнение на тялото.

- For цикъл е най-удобен, когато **очакваме точно колко на брой итерации** ще имаме, напр. искаме да въртим от 0 до n.
Защо for, вместо while? – защото ако използваме while, ще трябва ръчно вътре в самия цикъл да си следим за инкрементацията на променливата.

Пр.: за изчисляване на първите n члена на някаква редица, за намиране на сумата на много на брой числа, които имат някаква зависимост, за принтиране на повтаряща се последователност и др.

```
int i = 0;  
while (i < 10) {  
    cout << i;  
    i++;  
}  
  
vs.  
  
for (int i = 0; i < 10; i++) {  
    cout << i;  
}
```

Do...while цикъл е много подходящ за валидация на входа.

Защо do...while, вместо while? - ако използваме while и очакваме тепърва да се въведе стойност за променливата, която валидираме, то в условието трябва да оценим булев израз, в който не знаем стойността на променливата каква е.

Пр.: искаме в дадена програма въведеното число n да е положително, защото е дължина на отсечка. Това, което правим, е да позволим на потребителя да въвежда, докато n не изпълнява нашето условие.

```
int n;
do {
    cin >> n;
} while (n < 0);

int n;
while (n < 0) { //не се компилира; казва, че n не е инициализирана
    cin >> n;
}
```

VI. Break и continue

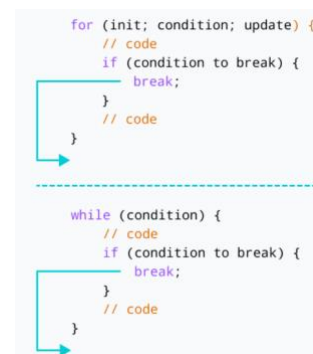
Break прекратява преждевременно изпълнението на цикъла.

- От този break нататък нищо повече не се изпълнява.
- Работи по еднакъв начин за for, while, do...while.
- Точно като при switch.

Пр.:
//a) for loop
for (int i = 0; i < 10; i++) {
 if (i == 4) {
 break;
 }
 cout << i << " ";
}
// 0 1 2 3

//b) while
int i = 0;
while(i < 5){
 if(i == 4) {
 break;
 }
 cout << i;
 i++;
}
//1234

//в) do...while
int i = 0;
do {
 i++;
 cout << i;
 if (i == 3) {
 break;
 }
} while (i < 4); //123



Continue прекратява преждевременно текущата итерация на цикъла и започва нова, ако има такава.

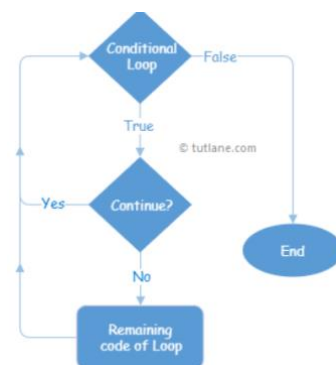
- От този continue нататък нищо повече не се изпълнява за **текущата** итерация.
- Работи по еднакъв начин за for, while, do...while.
 - Можете да си мислите за continue като за break, но само за една итерация.

Пр.:
//a) for loop
for (int i = 0; i < 10; i++) {
 if (i == 4) {
 continue;
 }
 cout << i << " ";
}
// 0 1 2 3 5 6 7 8 9

//b) while
int i = 0;
while(i < 5){
 if(i == 4) {
 continue;
 }
 cout << i;
 i++;
}
//0123

//в) do...while
int i = 0;
do {
 i++;
 if (i == 3) {
 continue;
 }
 cout << i;
} while (i < 4); //124

//г) do...while
int i = 0;
do {
 i++;
 cout << i;
 if (i == 3) {
 break;
 }
} while (i < 4); //123



VII. Префиксен и постфиксен оператор

Префикс – „представка“.

Префиксен оператор – **++i (--i)**; („++“ или „--“ е вляво на някаква променлива, **пред** променливата);

Семантика – променливата се увеличава (намалява) с 1 и после се изпълнява операцията, в която участва променливата с префиксния оператор.

Пр.:
int i = 0;
cout << ++i; //1

Постфикс – „наставка“.

Постфиксен оператор – **i++ (i--)**; („++“ или „--“ е вдясно на някаква променлива, **след** променливата);

Семантика – първо се изпълнява операцията, в която участва променливата с постфиксния оператор, после се увеличава (намалява) променливата с 1 и във всяка следваща операция тази променлива участва с увеличената (намалената) си с 1 стойност.

Пр.:
int i = 1;
cout << i++;
cout << " " << i; //1;2