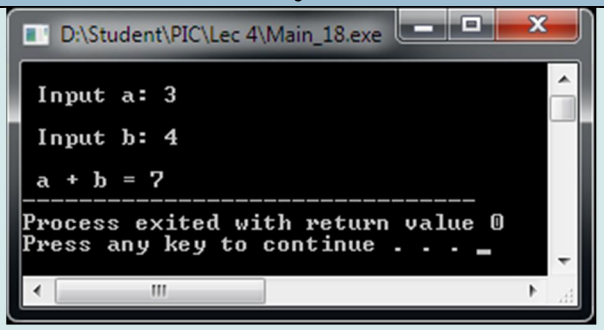
Оператори за цикли в езика C++. Реализиране на циклични алгоритми

Итерационните структури са фрагменти от алгоритмите за решаване на задачи, в които се организира циклично повторение на определен набор от инструкции, които постепенно приближават процеса до желания резултат. Този метод се използва за организация на циклични програми с неизвестен предварително брой на повторенията. Обикновено при тези процеси резултатът от предидущото действие се използва като входна информация за следващото действие (цикъл). Контролът на реализирания брой повторения се осъществява на основата на някакво условие, определящо приближаването на изчислителния процес до желания резултат. Често като условие за контрол на итерационните цикли се използва достигането на предварително зададена точност на изчисляваните величини. В програмирането съществуват няколко типа програми: линейни, разклонени и циклични.Линейни се наричат тези програми при които изпълнението на операторите се извършва последователно един след друг без да се пропуска някой.

Например:

Програмен код	Резултат

```
#include <stdio.h>>
int main(int argc, char* argv[])
{
  int a,b,c;
  printf("\n Input a: ");
  scanf("%d",&a);
  printf("\n Input b: ");
  scanf("%d",&b);
  c = a+b;
  printf("\n a + b = %d",c);
  return 0;
}
```

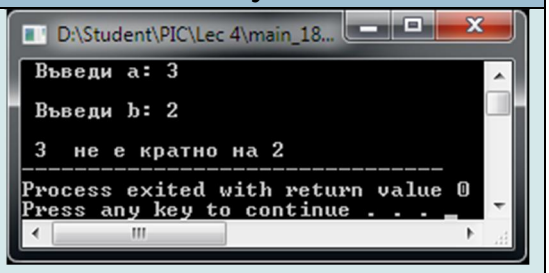


Разклонени са тези програми при които част от програмата може да не се изпълни в зависимост от това дали определено условие е изпълнено.

Например:

Програмен код	Резултат

```
#include <stdio.h>
#include <iostream>
int main(int argc, char* argv[]) {
  int a,b,c;
  setlocale(LC_ALL, "Bulgarian");
  printf("\n Въведи a: ");
  scanf("%d",&a);
  printf("\n Въведи b: ");
  scanf("%d",&b);
  c = a%b;
 if ( c)
  printf("\n %d  не е кратно на %d",a,b);
 else
   printf("\n %d е кратно на %d",a,b);
 return 0;
}
```

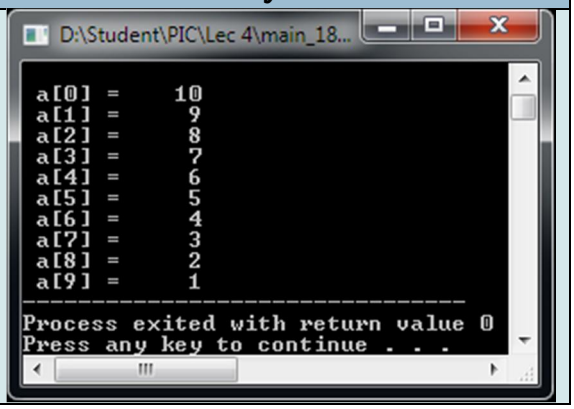


Циклични са тези програми при които част от програмата се изпълнява N пъти.

Например:

Програмен код	Резултат

```
#include <stdio.h>
int main(int argc, char* argv[])
{
 int a[10],i;
 for( i=0;i<10;i++)
    a[i] = 10 - i;
 for( i=0;i<10;i++)
   printf("\n a[%d] = %5d",i,a[i]);
 return 0;
}
```



Многократното изпълнение на последователност от инструкции представлява важна алгоритмическа концепция. Един от методите за организация на повторение на изпълнението на поредица от команди (който се явява итерационна структура) се нарича **цикъл**. Характерно за циклите е, че в конструкцията им се предвиждат специални инструкции, осигуряващи управлението на цикъла набор от операции, които се изпълняват многократно. Управляващите инструкциите задават началото и края на цикъла и задават структурата и типа на цикъла, а многократно изпълняваните инструкции образуват тялото на цикъла. Инструкцията за начало на цикъла обикновено се нарича заглавен оператор.

Управлението на цикличните структури обикновено съдържа три операции: **инициализация, проверка на условие за край на цикъла и модификация**. При инициализацията **се установява началното състояние**, от което започва изпълнението на цикъла. Самият процес на инициализация представлява задаване на начални стойности на някои величини, които в процеса на изпълнение на цикъла се променят. Операциите за инициализация могат да бъдат вградени в управляващите оператори на цикъла или да бъдат извършени преди началото на цикъла.

Операцията за проверка на условието за край на цикъла е много важна, тъй като **тя определя броя на повторенията на инструкциите от тялото на цикъла**. Когато такова условие не се проверява или е зададено некоректно, цикълът може да се превърне в безкраен. Модификацията е операция, чрез която се **променя състоянието на определени обекти** (стойности на величини), така че ситуацията да се приближава към изпълнението на условието за край на цикъла.

Предназначението на оператора за цикъл е да **организира многократното изпълнение на група команди, които образуват тялото на оператора**. В C/C++ се използват три различни оператора за цикъл. Всеки от тези оператори е подходящ за определени ситуации в

алгоритмите, но почти винаги е възможно да се прилага кой да е от тях за конкретен случай. Поради тази причина, програмистите обикновено предпочитат да използват един от операторите за цикъл в повечето случаи и рядко другите оператори.

Операторите за цикъл имат три основни елемента в организацията на програмния процес:

- Задаване на начални стойности на величини, участващи в управлението на цикъла. Те се наричат инициализиращи елементи и в някои случаи могат да се разполагат преди оператора за цикъл;

- Обновяване на стойностите на управляващите променливи. При всяко изпълнение на операторите от тялото на цикъла трябва да се обновява стойността на една или няколко променливи, от които зависи докога се изпълнява цикълът.

- Проверка на условие за край на цикъла. Повторението на изпълнение на операциите от тялото на цикъла се контролира от логическо условие. При всяко изпълнение на операторите от тялото трябва да се проверява това условие.

- Има три основни конструкции на цикли
- Цикъл тип **while** (докато е изпълнено условието - повтаряй тялото на цикъла);
- Цикъл тип **do...while** (повтаряй тялото на цикъла докато е изпълнено условието);
- Цикъл тип **for** (вариант на цикъла **while**).

Цикъл с предусловие (while).

Този цикъл може да се представи логически с помощта на изречението:

<Докато> <Израз> <повтаряй> <Оператор>.

Тук **Израз** е логически израз, а **Оператор** е изпълним оператор (група оператори) представляващ тялото на оператора. Схемата на оператора с предусловие има вида (фиг. 1), където с пунктирана линия е показано представянето на цикъл с предусловие.

Синтактическата форма на оператора в C/C++ има вида:

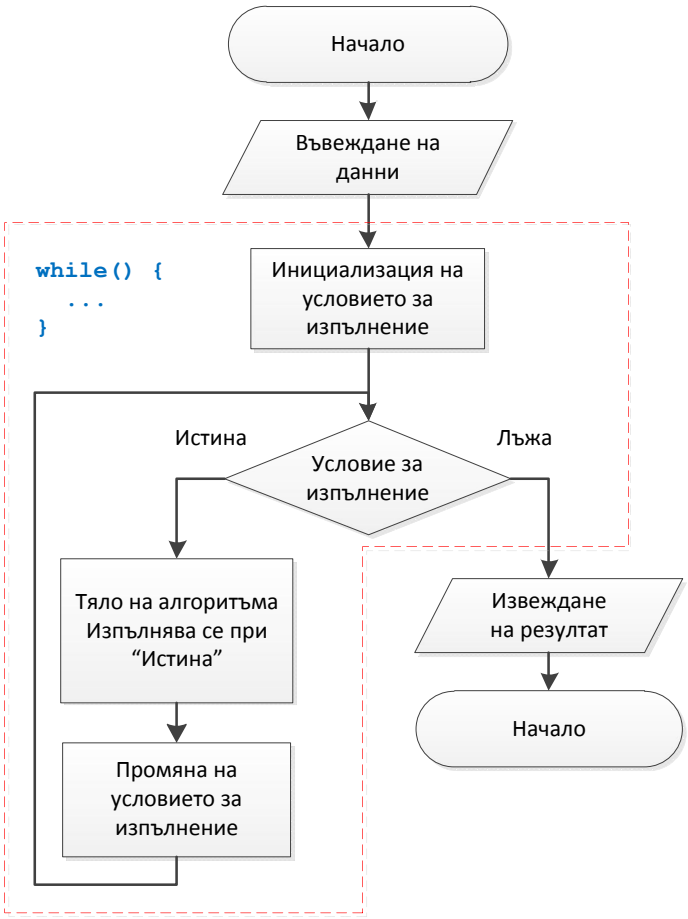
```
while(<Израз>)
{
   <Оператор>
}
```

където:

Израз - е логически израз, който дава като резултат стойност истина или лъжа (при математически изрази резултат 0 е лъжа, а за всички резултати различна от 0 е истина);

Оператор - един оператор или съставен оператор. Това е тялото на

цикъла.



Фиг.1. Цикличен алгоритъм с предусловие

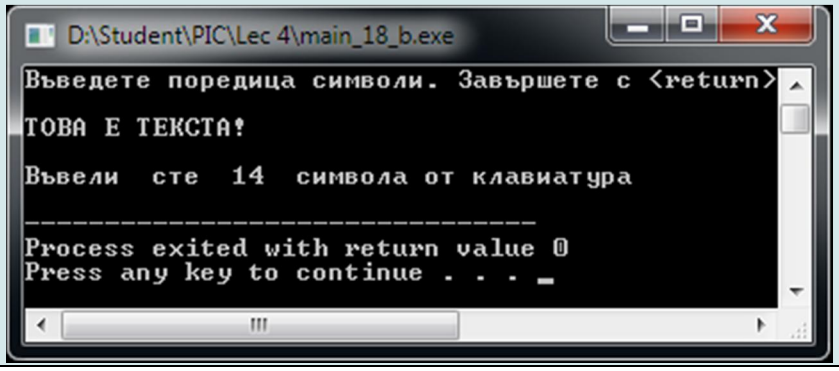
В тази конструкция на оператор за цикъл инициализиращата част се намира извън оператора за цикъл (преди оператора). Условието за проверка за продължаване на цикъла се задава с логическия израз **Израз**. Ако стойността на **Израз** е истина, се изпълнява тялото на цикъла **Оператор**, а ако е лъжа цикълът се напуска и управлението се предава на първия оператор, непосредствено следващ оператора за цикъл. След изпълнение на операторите от тялото, управлението се предава в началото на цикъла за нова проверка на условието за продължаване. Цикълът се нарича с предусловие, защото най-напред се проверява условието и ако то е истина, се изпълняват операторите от тялото на оператора. **Това означава, че е възможно тялото на цикъла да не се изпълни нито веднъж, ако още при влизането в цикъла условието е неистина.** Обновяване на стойността на управляващите променливи трябва да се предвиди в тялото на цикъла.

Следващата програма дава възможност да се въведе поредица от символи, които се преброяват. Процесът завършва при натискане на клавиша **<return>** - символ за преминаване на нов ред **(\n)**:

Програмен код

```
#include <stdio.h>  // printf, putch, puts
#include <iostream> // setlocale
#include <conio.h>  // conio
int main(int argc, char* argv[])
{
    char ch;
    int len;
    setlocale(LC_ALL, "Bulgarian");
    len = 0;
    puts("Въведете поредица символи. Завършете с <return>\n");
    while ((ch = getch()) != '\r')
    {
        putch(ch);
        len++;
    }
    printf("\n\nВъвели  сте  %d  символа от клавиатура\n",len);
 return 0;
}
```

Резултат

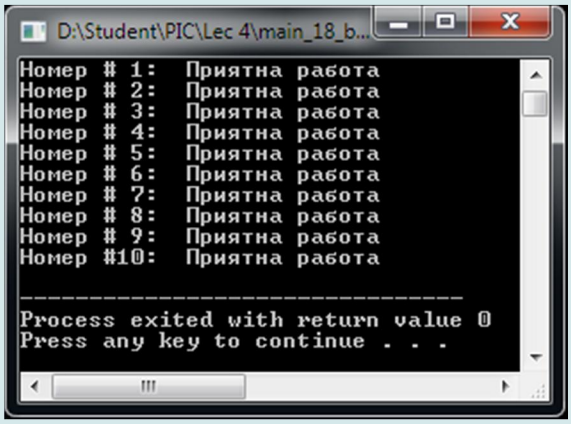


Ето пример, в който тялото на цикъла ще се изпълни точно 10 пъти:

Програмен код

```
#include <stdio.h>  // printf
#include <iostream> // setlocale
int main(int argc, char* argv[])
{
 char *msg ="Приятна работа";
 int indx;
 setlocale(LC_ALL, "Bulgarian");
 indx = 1;// Начална стойност на брояча
 while (indx<=10) {
  printf("Номер#%2d:%s\n",indx,msg);
  indx++;//Промяна стойността на брояча
 }
  return 0;
}
```

Резултат



Ето още един съкратен запис на горния цикъл:

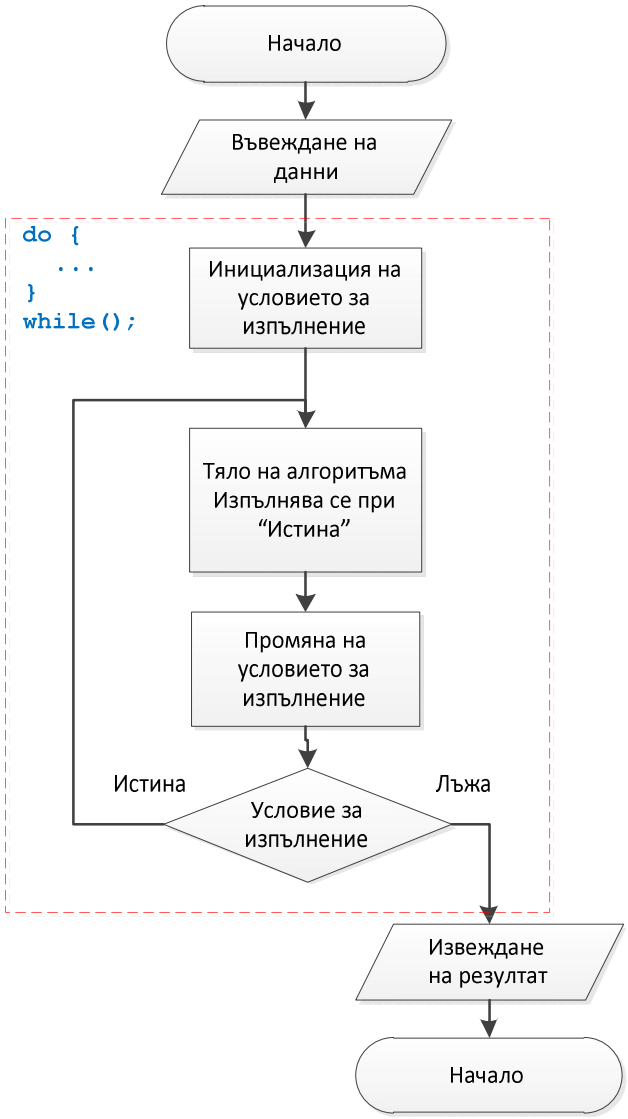
```
indx = 0;
while (indx++ < 10)
    printf("Номер #%2d:  %s\n",indx,msg);
```

Цикъл с постусловие (do...while)

Може да се представи логически с помощта на изречението:

<Повтаряй> <Оператор> <докато> <Израз>.

<Израз> е логически израз, а **<Оператор>** е изпълним оператор представляващ тялото на оператора. Алгоритмичната схема на оператора е показана на фиг. 2.



Фиг. 2. Цикличен алгоритъм със следусловие

Синтактическата форма на оператора в C/C++ има вида:

```
do {
    <Оператор>
} while (<Израз>);
```

където:

Израз - е израз, който дава като резултат стойност 0 или различна от 0;

Оператор - един оператор или съставен оператор.Това е тялото на цикъла.

Действието на оператора се изпълнява в следната последователност:

- Тялото на цикъла се изпълнява.
- Изчислява се **Израз**.
- Ако резултатът е стойност, различна от нула ("**истина**"), цикълът се изпълнява отново.
- Изпълнението на цикъла се прекратява, когато **Израз** получи стойност "**неистина**".

Както се вижда от схемата, проверката на условието за продължаване на цикъла се прави след изпълнение на операторите от тялото на цикъла.

Разликата между цикъл тип **while** и цикъл тип **do...while** е, че при **do...while** тялото на цикъла се изпълнява винаги поне един път, независимо от стойността на **Израз**.

И тук инициализиращата част се намира преди оператора за цикъл, а обновяване на стойността на управляващите променливи се извършва в тялото на цикъла.

Пример:

```
do {
    printf("Въведете стойност");
    scanf("%d",&a); }
 while (a<low  || a>high);
```

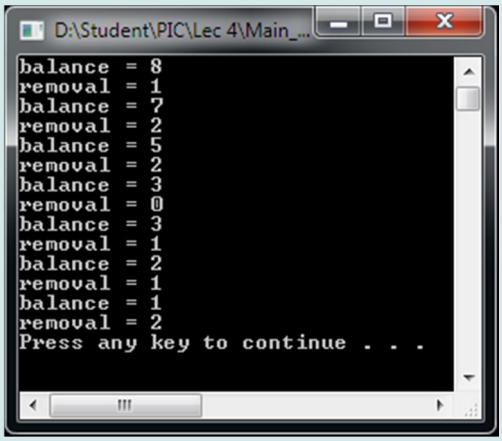
В примера се въвежда едно цяло число. Ако въведената стойност е извън диапазона **[low,high]**.

Следващият пример ще демонстрира използването на **do..while** оператора в реална програма

Пример:

Програмен код	Резултат

```
#include <iostream> // cout
#include <stdlib.h> // srand
#include <ctime>    // time
using namespace std;
int main(int argc, char* argv[]) {
 srand(time(0));
 int balance = 8;// начална стойност на брояча
  do // Начало на цикъла do while
  {
    cout << "balance = " << balance
         << endl; //Извежда баланса
// Генерира число в границите от 0 до 3
    int removal = rand() % 3;
    cout << "removal = " << removal
         << endl; // извежда числото
    // промяна условието за изход
    balance -= removal;
 }  while ( balance > 0 ); // край на цикъла do while
  return 0;
}
```



Цикъл с управляваща променлива (цикъл for)

В много случаи броят на изпълненията на операторите от тялото на цикъл е известен предварително и тогава може да се използва цикъл, управляван с променлива. Той може да се представи с логическата фраза:

за Id = St1 докато Условие2 прави Действие3

където **Id** е идентификатор управляващ цикъла (индекс на цикъла), който представлява проста променлива; **St1** е израз чиято стойност е начално значение на **Id**; **Условие2** - израз указващ условието за край и **Действие3** - оператор (оператори), който представлява тялото на цикъла.

Синтактическата форма на оператора в език C/C++ има вида:

```
for([<инициализация>];[<условие за край>];[<инкремент>])
      <Оператор>
```

където:

<оператор> е един оператор или съставен оператор;

<инициализация> обикновено е израз, който присвоява начална стойност на променливата, която играе ролята на брояч на цикъла;

<условие за край> - е проверка на условие за продължаване на цикъла;

<инкремент> е израз, който реализира промяна на стойността на брояча;

Този формат е просто частен случай на цикъл **while**. Той е

еквивалентен на следния запис:

```
<инициализация>; // инициализира брояча
while (<условие за край>)// проверка за  достигане краят на цикъла
 {
      <оператор>; // изпълними оператори
      <инкремент>; // модифицира брояча на цикъла
}
```

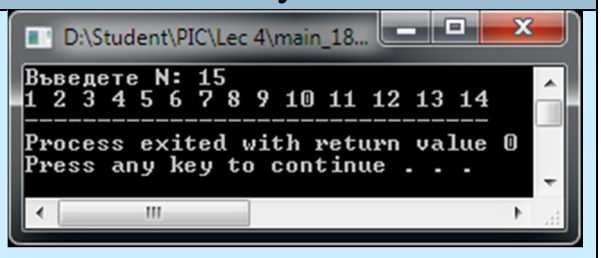
Квадратните скоби в синтаксиса показват, че всеки от изразите след **for** може да се пропусне, но точката и запетаята трябва да се запазят. Ако се пропусне **[<условие за край>],** стойността му се приема за 1 ("истина") и цикълът става безкраен

Допустими типове за управляваща променлива на цикъла са всички числени типове, булевите променливи (стойности 0 и 1) и символните типове (като номера от кодовата таблица), както и някои други типове, които ще бъдат въведени по-късно.

Ето един пример за използване на цикъл с управление с променлива. В случая, първоначално се въвежда стойността на променливата **N**, която задава броя на числата, които ще се изведат на екрана. За управляваща променлива се използва целочислената променлива **i**, която приема начална стойност 1 и стойността ѝ се увеличава в процеса на изпълнение до **N**.

Инициализиращият елемент се намира в заглавието на цикъла (**i=1**), а проверка на условието за край на цикъла се извършва автоматично чрез сравняване на стойността на управляващата променлива с крайната стойност (**N**). Ако въведената стойност за променливата **N** е по-малка от началната стойност (в случая 1), операторите от тялото на цикъла няма да се изпълнят нито веднъж.

Програмен код	Резултат

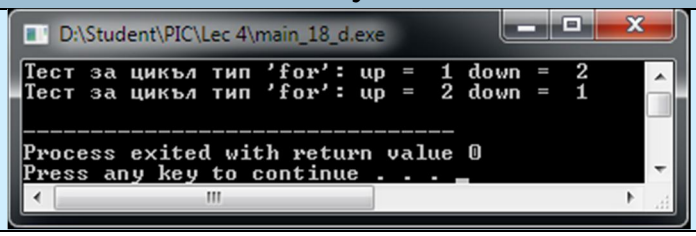
`#include <stdio.h>  // printf`
`#include <iostream> // setlocale`
`int main(int argc, char* argv[])`
`{`
`  int i,N;`
`  setlocale(LC_ALL, "Bulgarian");`
`  printf("Въведете N: ");`
`  scanf("%d",&N);`
`  for (i = 1; i < N ; i++)`
`  printf ("%d ",i);`
`  return 0;`
`}`
	

Ето пример за цикъл тип **for**:

Програмен код

```
#include <stdio.h>  // printf
#include <iostream> // setlocale
int main(int argc, char* argv[])
{
    int up, down, S1 = 1, S2 = 2;
    char *msg= " Тест за цикъл тип 'for'";
    setlocale(LC_ALL, "Bulgarian");
    for (up = S1, down = S2; up<=S2; up++, down--)
    printf ("%s: up = %2d down = %2d\n",msg,up,down);
}
```

Резултат



В разгледания формат на оператора за цикъл при всяко изпълнение на операторите от тялото на цикъла стойността на **up** се увеличава автоматично с 1 и цикълът продължава докато стойността на управляващата променлива не стане равна на крайната стойност **S2**. Естествено изискване при тази организация на цикличната процедура е стойността на **S1** да е по-малка от **S2**. Управляващата променлива заема всички стойности в интервала от **S1** до **S2**.

За да се реализира намаляващ цикъл в предходния пример е необходимо израза определящ условието за край (**up<=S2**) е необходимо да се замени с израза (**down >= S1**).

Нарастващ цикъл	Намаляващ цикъл

`cout <<"N = "; cin >> N;`
`for( int indx =1; indx <= N; indx++)`
`  cout << indx << endl;`
	`cout <<"N = "; cin >> N;`
	`for( int indx =N; indx >= 1; indx-- )`
	`  cout << indx << endl;`

Вложени цикли

Вложените цикли представляват конструкция от няколко цикъла един в друг. Цикълът, в който се влага друг цикъл, се нарича външен цикъл, а този, който се влага, се нарича вътрешен. Най-вътрешния цикъл се изпълнява най-много пъти. В примерната конструкция по долу представлява вложен цикъл.

Пример:

```
  for( int i=0;i<10;i++)
    for(int j= 10; j>0; j--)
        cout << " i = " << i << "   j = " << j << endl;
```

След инициализация на първия **for** цикъл (i=0) ще започне да се изпълнява втория. Ще се инициализира променливата му (j= 10), ще се провери условието (j>0) и ще се изпълнят изразите в него

(cout << " i = " << i << " j = " << j << endl;), след което ще се обнови променливата (j--) и ще продължи изпълнението на този цикъл, докато условието му не върне false. В този случай ще се върне в първия **for** цикъл, ще се извърши обновяване на неговата променлива (i++), ще се провери условието (i<10) и отново ще бъде изпълнен целия втори цикъл.

Тук има вложение на два цикъла, като вътрешният цикъл (индекс j) се изпълнява за всяка стойност на променливата на външния цикъл. В случая тялото на вътрешния цикъл ще се изпълни 90 пъти. Може да се организира многократно влагане на цикли един в друг.

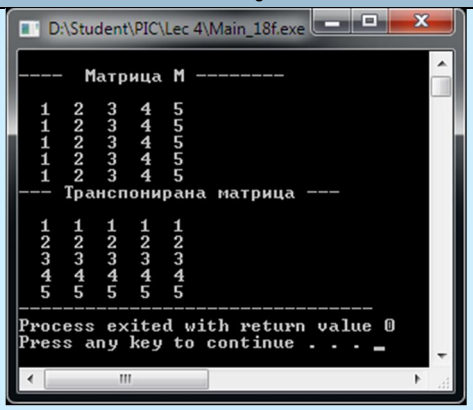
Обикновено for цикъла се използват за манипулация на двумерни масиви.

Пример:

Този пример демонстрира действието на вложените цикли в алгоритъм за транспониране на матрици (в програмните езици матриците се представят с масиви)

Програмен код	Резултат

```
#include <stdio.h>  // printf
#include <iostream>>
int main(int argc, char* argv[])
{
  int M[5][5] = {{ 1,2,3,4,5},
                 { 1,2,3,4,5},
                 { 1,2,3,4,5},
                 { 1,2,3,4,5},
                 { 1,2,3,4,5}};
  int i,j,k,N=5;
  setlocale(LC_ALL, "Bulgarian");
 printf("\n-----  Матрица M -----\n");
  for( i = 0; i < N ; i ++) {
   printf("\n");
   for( j = 0; j < N ; j ++) {
      printf("%3d",M[i][j]);
   }
  }
    // Транспониране на матрицата
  for( i = 0; i < N ; i ++) {
   for( j = i+1; j < N ; j ++) {
       k = M[i][j];
       M[i][j]= M[j][i];
       M[j][i] = k;
   }
  }
    // Извеждане на матрицата
 printf("\n Транспонирана матрица\n");
  for( i = 0; i < N ; i ++) {
   printf("\n");
   for( j = 0; j < N ; j ++) {
       printf("%3d",M[i][j]);
   }
  }
}
```



Вложените цикли, използвани необмислено, могат да влошат производителността на програмата.

Преждевременно излизане от цикъл - break

Доста често се налага да се прекъсне изпълнението на даден цикъл, преди да са изпълнени зададения брой итерации. Това може да стане по два начина, задаване на стойност на индексната променлива извън горната граница за цикъла, или използването на оператора **break**.

Използването на първия случай не е удачно поради необходимостта да се знаят винаги граничните условия на цикъла. Това не винаги е възможно. **Например:**

```
void InputData()
{
 int i;
 for(;;)
  {
       i = getchar();
       if( i != 0 )
           printf("\n\t\%+5d",i);
       else
        {
          printf("\n\t\%+5d Exit ",i);
         // Излизане от цикъла
        }
  }
}
```

В горният пример изразът **for(;;)** дефинира безкраен цикъл. Излизането от цикъла практически е не възможно с използването на първия метод. Използването на оператора **break**, би ни помогнало в тази ситуация. Ето как би изглеждал горния пример:

```
void InputData()
{
 int i;
 for(;;)
  {
       i = getchar();
       if( i != 0 )
           printf("\n\t\%+5d",i);
       else
        {
          printf("\n\t\%+5d Exit ",i);
         break; // Излизане от цикъла
        }
  }
}
```

За повече информация за оператора break виж switch

Прекъсване на текуща итерация continue

Има случаи, в които не желаем да напуснем изобщо даден цикъл, а искаме само да пропуснем останалата част от операторите в него и да продължим пак от началото му. Такъв начин на действие се осъществява от оператора **continue**. Ето пример:

```
#define LIMIT 100
#define MAX 10
int main( void )
 {
   int i,j,k,score;
   int scores[LIMIT][MAX];
   for (i=0; i<LIMIT; i++)  {
    j=0;
    while (j<MAX-1)   {
       printf ("Въведете резултат #%d:",j);
       scanf("%d",&score);
       if (score<0)
         continue;
       scores[i][++j] = score;
   }
   scores[i][0] = j;
 }
  return 0;
}
```

При изпълнението на оператор **continue**, програмата пропуска останалата част от операторите в цикъла и започва от началото му. Поради това резултатите са различни от тези, получени без **continue**. Сега вътрешният цикъл не се напуска. Вместо това, въвеждането на -1 се интерпретира като грешка и управлението се предава в началото на цикъла **while**. Тъй като не се изпълнява операторът, който променя брояча, цикълът ще се изпълни отново с последната стойност на j и ще повтори въпроса.

Оператор за безусловен преход goto

Езикът C/C++ предлага и оператор **goto**, въпреки че повечето от случаите могат да се решат с останалите три оператора за предаване на управлението. Използвайте оператора внимателно и само ако се налага. Форматът му е:

```
goto етикет;
    . . .
етикет:  оператор
```

където "етикет" е идентификатор, свързан с даден оператор.

Задачи за самоподготовка

1. Напишете програма, която отпечатва на конзолата числата от 1 до N, които не се делят на 5 и 9. Числото N се въвежда от клавиатурата.
2. Напишете програма, която чете от конзолата поредица от цели числа и отпечатва най-малкото и най-голямото от тях.
3. Напишете програма, която чете от конзолата числото N и отпечатва сумата на първите N члена от редицата на Фибоначи: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, …
4. Напишете програма, която пресмята N!/K! за дадени N и K (1<K<N).
5. Напишете програма, която пресмята N!*K!/(N-K)! за дадени N и K.

Въпроси за самоконтрол

1. Какво е предназначението на цикличните програми?
2. Какво се разбира под термина "цикъл"?
3. Какви видове циклични алгоритми познавате?
4. Какво е цикъл с предусловие?
5. Какво е цикъл със следусловие?
6. Как се реализира намаляващ цикъл?
7. Каква е разликата между цикъл с предусловие и цикъл със следусловие?