



ПРОГРАММИРОВАНИЕ
НА ЯЗЫКЕ C

Урок №5

Програмиране на
език

C

Съдържание

1.Вградена конструкция	3
2.Практически примери.	7
Използване на вградения дебъгер Microsoft Visual Studio. 11	
3. Домашна работа.	20

1. Вградена конструкция

В предишните уроци се запознахме с конструкцията цикъл и вариантите му за реализация в езика C. Както вече сте забелязали, цикъла е един от основополагащите конструкции в програмирането. С негова помощ се решава огромно количество от задачи. Също така сте се разбрали, че в цикъла може да се вкара конструкции за логически избор, такива като `if` и `switch`. Ние няма да се спираме само на това и ще се опитаме да вкараме в цикъла друга подобна на него конструкция, тоест друг цикъл. Да разгледаме елементарен пример.

```
#include <iostream>
using namespace std;
void main ()
{
    int i=0,j;
    while(i<3){
        cout<<"\nOut!!!\n";
        j=0;
        while(j<3){
            cout<<"\nIn!!!\n";
            j++;
        }
        i++;
    }
    cout<<"\nEnd!!!\n";
}
```

Име на проекта `NestedLoop`.

Да проанализираме примера:

1. Програмата проверява условието `i<3`, тъй като 0 е по-малко от 3 условието е вярно и програмата влиза във външния цикъл.

2. На екрана се показва **Out!!!**

3. Променливата j се нулира.

4. Сега се проверява условието $j < 3$, тъй като 0 е по-малко от 3 условието е вярно и програмата влиза във вътрешния цикъл.

5. На екрана се показва **In!!!**

6. Управляващата променлива j се променя.

7. Отново се проверява условието $j < 3$, след като 1 е по-малко от 3 условието е вярно и програмата влиза във вътрешния цикъл.

8. На екрана се показва **In!!!**

9. Извършва се промяна на управляващата променлива j .

10. Отново се проверява условието $j < 3$, след като 2 е по-малко от 3 условието е вярно и програмата влиза във вътрешния цикъл.

11. На екрана се показва **In!!!**

12. Извършва се промяна на управляващата променлива j .

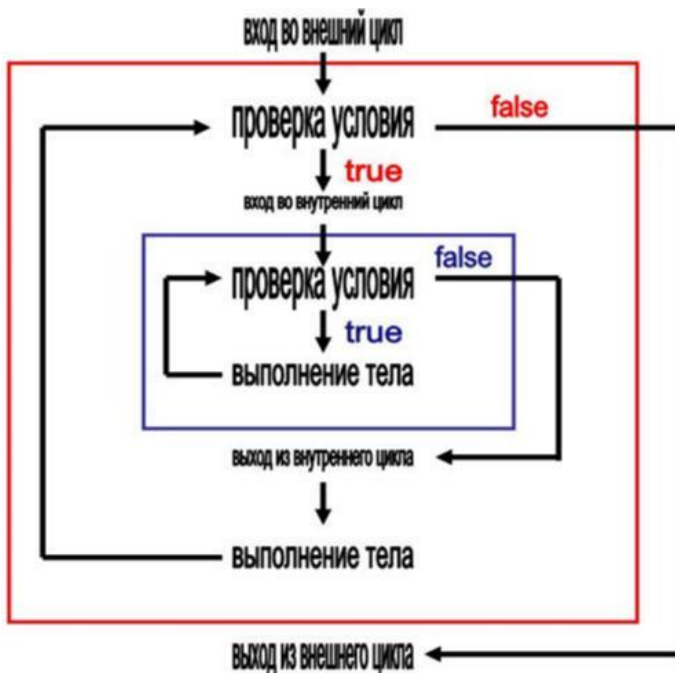
13. Отново се проверява условието $j < 3$, след като 3 не е по-малко от 3 условието е грешно и програмата излиза от вътрешния цикъл.

След това кода се връща към първи пункт 1. Всичките гореописани действия (1–13) се повтарят 3 пъти, т. е. докато i не стане равно по значение на 3. След това програмата излиза от външния цикъл и на екрана се показва **End!!!**

```
Out!!!  
In!!!  
In!!!  
In!!!  
Out!!!  
In!!!  
In!!!  
In!!!  
Out!!!  
In!!!  
In!!!  
In!!!  
End!!!  
Для продолжения нажмите любую клавишу . . .
```

Работния принцип на програмата, реализираща вложен цикъл е основан на това, че вътрешния цикъл се изпълнява изцяло на всяка стъпка от външния цикъл от начало до край. С други думи, докато програмата не излезе от вложения цикъл – изпълнението на външния няма да продължи. Долу е изобразена схема на работата на вложения цикъл.

Схема на работа:



Както виждате, много е лесно, но независимо от това, вложените конструкции значително опростяват реализацията на повечето сложни алгоритми. Убедете се сами като разгледате следващия раздел от урока в който ние сме ви подготвили няколко примера.

2. Практически примери

Пример 1

Постановка на задачата :

Да се напише програма, която изкарва на екрана таблицата за умножение. Име на проекта MultiplicationTable.

Код за реализация:

```
#include <iostream>
using namespace std;
void main ()
{
    for(int i=1;i<10;i++)
    {
        for(int j=0;j<10;j++)
        {
            cout<<i*j<<"\t";
        }
        cout<<"\n\n";
    }
}
```

Коментар към кода:

1. Управляващите променливи на външния и вътрешен цикъл осъществяват функцията на множител.

2. Управляващата променлива *i* се създава и инициализира със значение 1.

3. Програмата проверява условието *i*<10, след като 1 е по-малко от 10 условието е вярно и програмата отива към външния цикъл.

4. Управляващата променлива *j* се създава и инициализира със значение 1.

5. Програмата проверява условието *j*<10, след като 1 е по-малко от 10 условието е вярно и програмата влиза във вътрешния цикъл.

6. На екрана се показва произведението i на j — **1**

7. Осъществява се промяна на управляващата променлива j .

8. Отново се проверява условието $j < 10$, след като 2 е по-малко от 10 условието е вярно и програмата отново влиза във вътрешния цикъл.

9. На екрана се показва произведението на i и j — **2**

10. Осъществява се промяна на управляващата променлива j .

...

Действията от 5 до 7 се повтарят докато j не стане равен на 10, при това текущото значение на i (1) се умножава на всяко значение на j (от 1 до 9 включително), резултата се показва на екрана. Получава се ред от таблицата за умножение на 1.

След това програмата излиза от вътрешния цикъл и прехвърля курсора на екрана с два реда по-надолу. След това се осъществява увеличение на променливата i на единица и отново вход във вътрешния цикъл. А сега за изкарване на веригата за умножение по 2.

По такъв начин, в края на краищата на екрана се появява цялата таблица за умножение.

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Для продължения нажните любую клавишу . . . _

Пример 2

Постановка задачи:

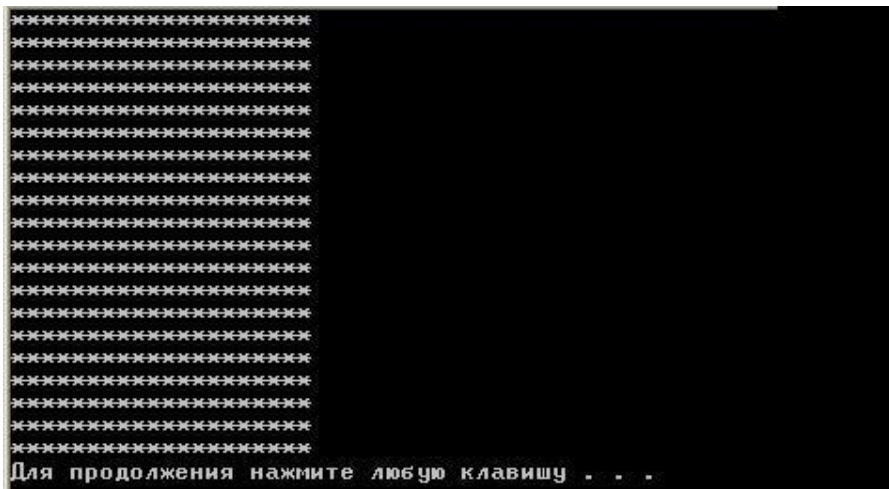
Въведете на екрана правоъгълник от символи 20 на 20. Име на проекта Rectangle.

Код за реализация:

```
#include <iostream>
using namespace std;
void main(){
    int str;
    int star_count;
    int length=20;
    str=1;
    while(str<=length)
    {
        star_count=1;
        while(star_count<=length)
        {
            cout<<"*";
            star_count++;
        }
        cout<<"\n";
        str++;
    }
}
```

Коментар към кода:

1. Управляващата променлива от външния цикъл — `str` контролира количеството редове в правоъгълника.
2. Управляващата променлива на вътрешния цикъл — `star_count` контролира количеството символи във всеки ред.
3. `length` — дължината на страна на правоъгълника
4. След изписване на всеки ред във външния цикъл се осъществява преход към следващия ред на правоъгълника.
5. Резултата е следния:



Забележка: Обърнете внимание, че независимо на това, че количеството редове съответства на количеството символи в реда – на екрана не е квадрат. Това е така защото височината и широчината на символите е различна.

Това е всичко! Сега имате пълна информация за цикъла, неговите разновидности и принцип на работа. Но, преди да изпълните домашната си работа, трябва да се запознаете с още един раздел от урока. Този раздел може да ви помогне, не само да пишете програми, но и да анализирате тяхната работа.

3. Използване на вградения „дебъгер“ на Microsoft Visual Studio

Понятието дебъгер. Необходимостта от дебъгер. Както вече знаете, съществуват два вида програмни грешки.

Грешка при етапа на компилация – грешка на синтаксиса на езика за програмиране. Такива грешки се контролират от компилатора. Програма съдържаща такава грешка няма да се изпълни и компилатора ще покаже в кой ред от кода е произлязла грешката.

Грешка при етапа за изпълнение – грешка водеща към некоректна работа на програмата или към нейното спиране. В такъв случай трябва да се отчете, че такава грешка не се контролира от компилатора. Само в редки случаи компилатора може да изкара предупреждение за някаква некоректна инструкция, но в общи линии в такива ситуации програмиста трябва да се спасява сам.

Точно за грешки при етапа за изпълнение ще говорим сега. Често за да се намери подобна грешка е необходимо да се премине през някои от фрагментите на програмата стъпка по стъпка. Безусловно в този момент е желателно точно да се проследи какво значение в определен момент се намира в конкретни променливи. Такова пресмятане може да се проведе и на парче хартия за да се анализира програмата ред по ред. В средата за разработване Visual Studio има специално средство за организация на анализа на програмата – дебъгер. Дадения раздел от урока е посветен на основите за работа с дебъгера.

Изпълнение на програмата по стъпки.

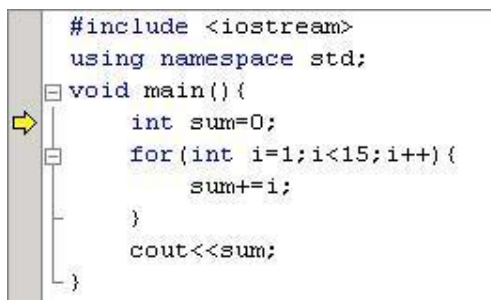
Да предположим, че се готвим да анализираме следния код:

```
#include <iostream>
using namespace std;
void main(){
    int sum=0;
    for(int i=1;i<15;i++){
        sum+=i;
    }
    cout<<sum;
}
```

За начало създайте проект и напишете този код. Компилирайте го и се убедете, че няма синтактични грешки. Сега да започваме. Натиснете на клавиатурата F10. До първия изпълним ред ще се появи жълта стрелка.

```
#include <iostream>
using namespace std;
void main(){
    int sum=0;
    for(int i=1;i<15;i++){
        sum+=i;
    }
    cout<<sum;
}
```

Именно тази стрелка показва, кой ред от кода се изпълнява сега. За да преминете към следващата стъпка от програмата натиснете отново F10. По този начин попадате на следващия ред:

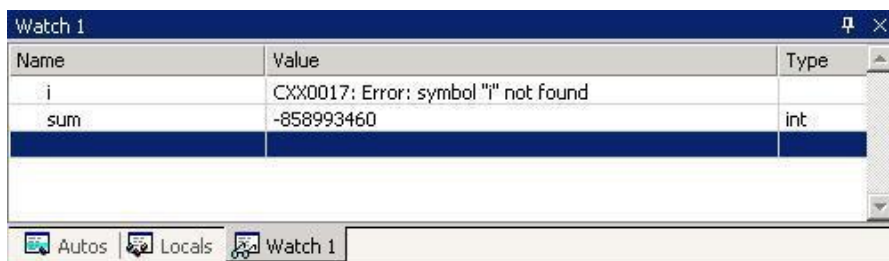


```
#include <iostream>
using namespace std;
void main(){
    int sum=0;
    for(int i=1;i<15;i++){
        sum+=i;
    }
    cout<<sum;
}
```

Обърнете внимание на това, че долу на екрана при вас се разполага набор от допълнителни прозорци за анализ на променливите:

Autos – този прозорец е предназначен за преглеждане на значенията на променливите, които съществуват в момента на изпълнение на текущия ред на кода. Да се попълни в дадения прозорец нещо от нас е невъзможно – това е автоматична функция.





Watch — Е предназначен точно за тези случаи, когато е необходимо сами да изберем променлива за преглед. Просто изписвате в полето Name името на променливата и тя се отразява независимо от изпълнимия код.

Сега натиснете просто F10, преминете през кода и погледнете, как ще се променят данните в прозорците.

Забележка: Ако искате да спрете дебъгера преди да приключи анализа на кода натиснете Shift+F5

Забележка: Сигурно сте забелязали, че дебъгера започва анализа си от първия ред на програмата. Ако искате да пуснете дебъгера от определен ред на програмата – поставете курсора в необходимия ред и натиснете Ctrl+F10

Точка на прекъсване.

Да разгледаме ситуация в която ни е необходимо да изпълним част от кода и да спрем на определено място, пуснете дебъгера. За това се използва така наречената точка на прекъсване.

Наберете следния код — поставете курсора в реда `count<<i;` и натиснете клавиш F9. До реда ще се появи червена точка, това е и точката за прекъсване.

```
#include <iostream>
using namespace std;
void main() {
    cout<<"Begin\n";
    for(int i=1;i<10;i++){
        cout<<i;
    }
    cout<<"End\n";
}
```

Сега натиснете F5, програмата ще се пусне, изпълнява се до този момент, където е поставена точката за прекъсване и ще премине в режим дебъгер.

```
#include <iostream>
using namespace std;
void main() {
    cout<<"Begin\n";
    for(int i=1;i<10;i++){
        cout<<i;
    }
    cout<<"End\n";
}
```

Обърнете внимание на състоянието на конзолата(прозореца на програмата). Тук се отразява всичко, което е успяло да произлезе:

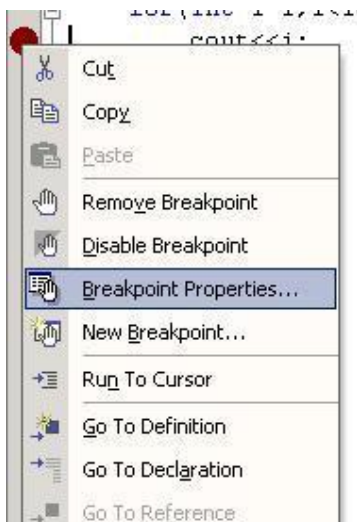
```
Begin
_
```

След това следва обикновената работа на дебъгера. Преместете жълтата стрелка с помощта на F10 и следете какво се случва с променливите. Освен това и поглеждате в прозореца на конзолата, всичките промени, които протичат ще се отразяват и там.

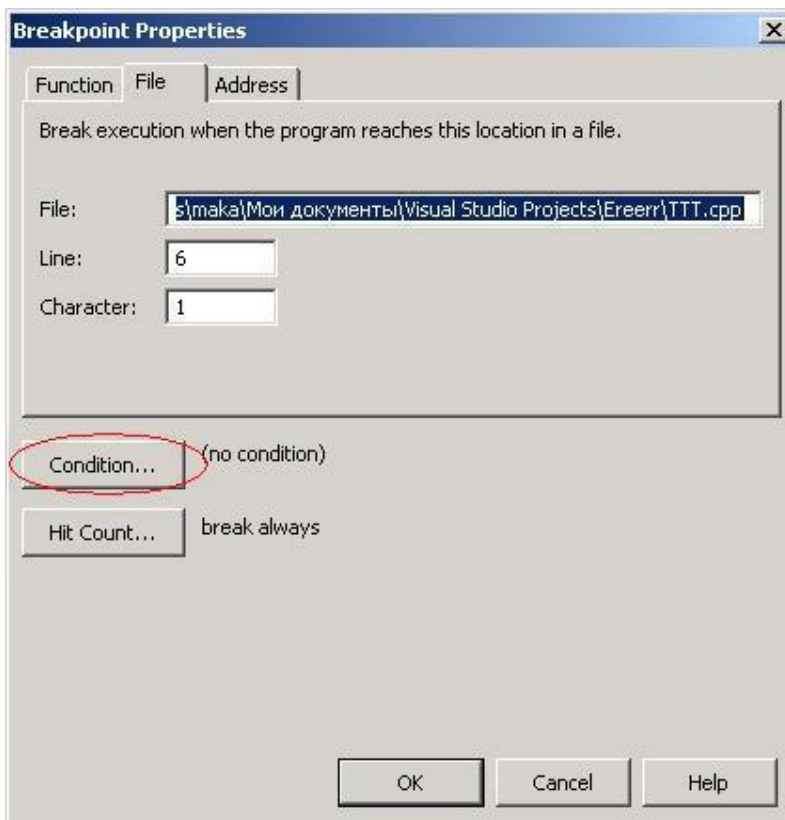
«Умна» точка на прекъсване.

Току що сме пуснали анализ на програмата от определено място. Обаче, ще забележим, че дебъгера е проработил веднага след като е започнало изпълнението на цикъла, т.е. на първата итерация. Това е неудобно в случай, ако итерациите са много и на вас ви се налага да пропуснете няколко от тях. С други думи искате да започнете анализа от 5 итерация на цикъла. Да се реши този проблем е лесно – да се направи точката на прекъсване „умна“.

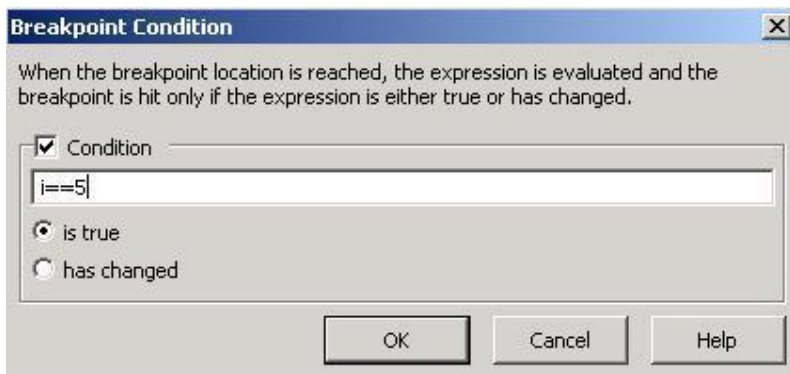
За това на самата точка натиснете десен бутон на мишката и в откритото меню изберете Breakpoint Properties.



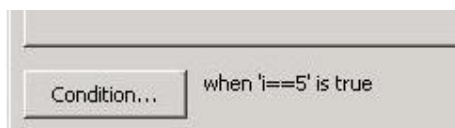
Пред вас се е появил прозорец. File, Line и Character — това е файл, ред и позицията в която е установена точката на прекъсване. Нас ни интересува копчето Condition, до което е написано — (no condition). Натискаме го.



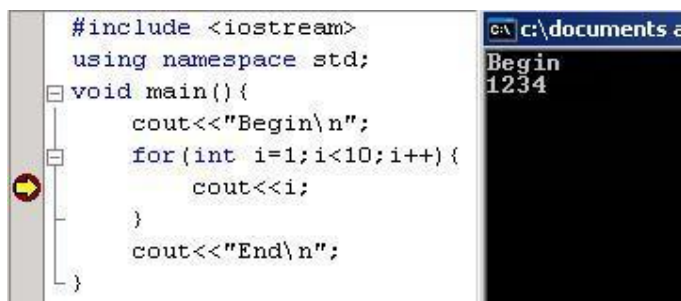
Появява се прозорец, в който трябва да се напише условието с което да се пусне дебъгера. Нашето условие $i==5$. Избираме `is true` — тоест да се спре когато условието стане вярно. Натискаме OK.



Сега до копчето има друг текст. В главния прозорец също така натискаме ОК.



Когато всичко е готово, натискаме F5 и гледаме какво ще се случи. Както виждате, всичко е успешно – дебъгера се е пуснал в нужния за нас момент.



Забележка: Да се премахне точката за прекъсване може да се направи с натискане на F9 в реда в който точката се намира.

Днес не разказахме за всичките възможности на дебъгера. Ние ще засегнем тази тема след време пак когато ще изучаваме функции. А сега, настоятелно препоръчваме да се по упражнявате с работа с дебъгера, например, с всички примери от урока и с домашната работа.

Успех!

4. Домашна работа

1. Да се създаде програма, която да изкарва на екрана прости числа в диапазона от 2 до 1000. (Числото се нарича просто само ако се дели на 1 и на себе си без остатък; при което числата 1 и 2 не се смятат за прости).

2. Да се напише програма, която да изкарва на екрана следната фигура:

```
*****
*               *
*               *
*               *
*               *
*               *
*               *
*****
```

Широчината и височината на фигурата се определят от потребителя с клавиатура.

3. С помощта на цикъл да се покаже на екрана календар на текущия месец.

