



ПРОГРАММИРОВАНИЕ
НА ЯЗЫКЕ C

Урок №5

Вложенные циклы

Содержание

1. Вложенная конструкция	3
2. Практические примеры	7
Пример 1	7
Пример 2	9
3. Использование интегрированного отладчика Microsoft Visual Studio	12
Выполнение программы по шагам	13
Точка останова	15
«Умная» точка останова	16
4. Домашнее задание	21

1. Вложенная конструкция

В прошлых уроках вы познакомились с конструкцией под названием цикл и вариантами реализации цикла в языке C. Как вы уже успели заметить, цикл является одной из основополагающих конструкций программирования. С его помощью решается огромное количество задач. Также, вы уже столкнулись с тем, что в цикл можно вкладывать конструкции логического выбора, такие, как `if` и `switch`. Однако, не будем останавливаться на достигнутом и, попробуем вложить в цикл подобную ему конструкцию, т.е. — другой цикл. Рассмотрим простой 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";
}
```

Название проекта NestedLoop.

Проанализируем пример:

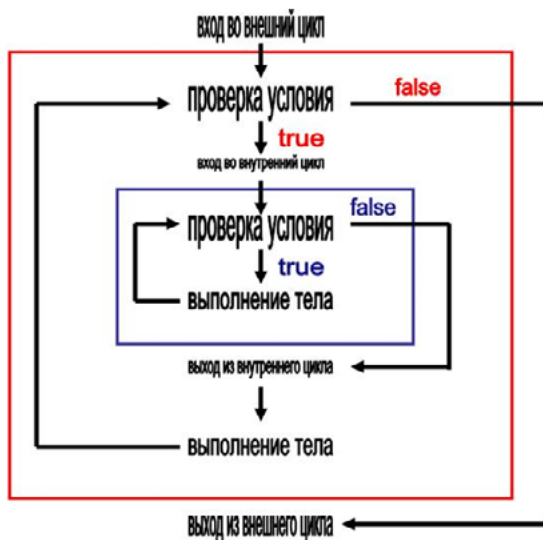
1. Программа проверяет условие $i < 3$, так как 0 меньше 3 условие является истинным и программа входит во внешний цикл.
2. Осуществляется показ на экран **Out!!!**
3. Обнуляется переменная j .
4. Теперь проверяется условие $j < 3$, так как 0 меньше 3 условие является истинным и программа входит во внутренний цикл.
5. Осуществляется показ на экран **In!!!**
6. Осуществляется изменение управляющей переменной j .
7. Снова проверяется условие $j < 3$, так как 1 меньше 3 условие является истинным и программа входит во внутренний цикл.
8. Осуществляется показ на экран **In!!!**
9. Осуществляется изменение управляющей переменной j .
10. Снова проверяется условие $j < 3$, так как 2 меньше 3 условие является истинным и программа входит во внутренний цикл.
11. Осуществляется показ на экран **In!!!**
12. Осуществляется изменение управляющей переменной j .
13. Снова проверяется условие $j < 3$, так как 3 не меньше 3 условие является ложным и программа выходит из внутреннего цикла.

Далее код возвращается к пункту 1. Все вышеописанные действия (1–13) повторятся 3 раза, т.е. до тех пор пока i не станет равно значению 3. После этого про-

грамма выйдет из выполнения внешнего цикла и на экран выведется **End!!!**

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

Принцип работы программы, реализующей вложенный цикл основан на том, что внутренний цикл полностью выполняется на каждом шаге внешнего цикла от начала до конца. Другими словами, пока программа не выйдет из вложенного цикла — выполнение внешнего не продолжится. Ниже изображена схема работы вложенных циклов.

Схема работы:

Как видите, все просто, но несмотря на это вложенные конструкции значительно упрощают реализацию большинства сложных алгоритмов. Убедитесь в этом, рассматривая следующий раздел урока, в котором мы подготовили для вас несколько примеров.

2. Практические примеры

Пример 1

Постановка задачи

Написать программу, которая выводит на экран таблицу умножения. Название проекта MultiplicationTable.

Код реализации:

```
#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";
    }
}
```

Комментарий к коду

1. Управляющие переменные внешнего и внутреннего циклов осуществляют функции множителей.
2. Управляющая переменная *i* создается и инициализируется значением 1.
3. Программа проверяет условие *i*<10, так как 1 меньше 10 условие является истинным и программа входит во внешний цикл.

4. Управляющая переменная j создается и инициализируется значением 1.
5. Программа проверяет условие $j < 10$, так как 1 меньше 10 условие является истинным и программа входит во внутренний цикл.
6. Осуществляется показ на экран произведения i на j — 1
7. Осуществляется изменение управляющей переменной j .
8. Снова проверяется условие $j < 10$, так как 2 меньше 10 условие является истинным и программа снова входит во внутренний цикл.
9. Осуществляется показ на экран произведения i на j — 2
10. Осуществляется изменение управляющей переменной j .
- ...

Действия с 5 по 7 повторяются до тех пор пока j не становится равно 10, при этом текущее значение i (1) умножается на каждое значение j (от 1 до 9 включительно), результат показывается на экран. Получается строка таблицы умножения на 1.

Затем программа выходит из внутреннего цикла и переводит экранный курсор на две строки вниз. После этого, осуществляется увеличение переменной i на единицу и снова вход во внутренний цикл. Теперь уже для вывода цепочки умножения на 2.

Таким образом, в конце концов на экране появляется вся таблица умножения.

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 продолжения нажмите любую клавишу . . .

Пример 2

Постановка задачи

Вывести на экран прямоугольник из символов 20 на 20. Название проекта Rectangle.

Код реализации

```
#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++;  
        }  
    }
```

Комментарий к коду

1. Управляющая переменная внешнего цикла — `str` контролирует количество строк в прямоугольнике.
2. Управляющая переменная внутреннего цикла — `star_count` контролирует количество символов в каждой строке.
3. `length` — длина стороны прямоугольника
4. После отрисовки каждой строки, во внешнем цикле осуществляется переход на следующую строчку прямоугольника.
5. Результат таков:

Для продолжения нажмите любую клавишу . . .

Примечание: Обратите внимания, что несмотря на то, что количество строк соответствует количеству символов в строке — на экране не квадрат! Это связано с тем, что высота и ширина символа разные.

Вот и всё! Теперь у вас имеется полная информация о циклах, их разновидностях и принципах работы. Но, прежде чем выполнять домашнее задание, следует ознакомиться с еще одним разделом урока. Этот раздел поможет вам не только писать программы, но и анализировать их работу.

3. Использование интегрированного отладчика Microsoft Visual Studio

Понятие отладки. Необходимость использования отладчика. Как вы уже знаете — существует два вида ошибок программы.

Ошибка на этапе компиляции — ошибка синтаксиса языка программирования. Такие ошибки или опечатки контролируются компилятором. Программа, содержащая такую ошибку просто не запустится на выполнение и компилятор укажет в какой строке кода произошла ошибка.

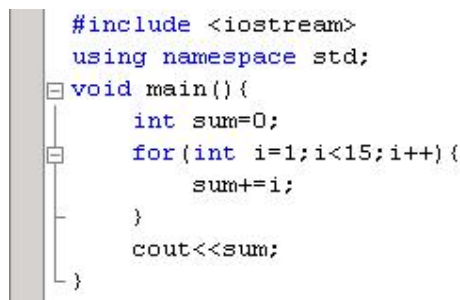
Ошибка на этапе выполнения — ошибка, приводящая к некорректной работе программы, либо к полной остановке последней. При этом следует учитывать, что такая ошибка компилятором не контролируется. Лишь в редких случаях компилятор может выдать предупреждение о какой-то некорректной инструкции, но в общем и целом в таких ситуациях программисту приходится выпутываться самому.

Именно об ошибках на этапе выполнения и пойдет речь. Зачастую, чтобы обнаружить подобную ошибку необходимо пройти некий фрагмент программы по шагам, так как если бы программа выполнялась. Безусловно при этом желательно четко просчитать какое значение в определенный момент времени находится в конкретных переменных. Можно конечно же произвести такой под-

счет на листе бумаги построчно анализируя программу, однако в среде разработки Visual Studio есть специальное средство для организации анализа программы — отладчик. Данный раздел урока посвящен основам работы с отладчиком.

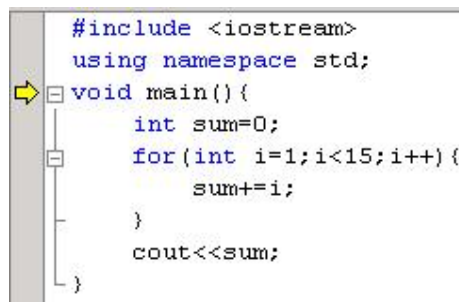
Выполнение программы по шагам

Предположим, что мы собираемся проанализировать следующий код:



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

Для начала создайте проект и наберите этот код. Скомпилируйте его и убедитесь, что нет синтаксических ошибок. Теперь приступим. Нажмите на клавиатуре функциональную клавишу F10. Рядом с первой выполняемой строкой кода у вас на экране появится желтая стрелка.



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

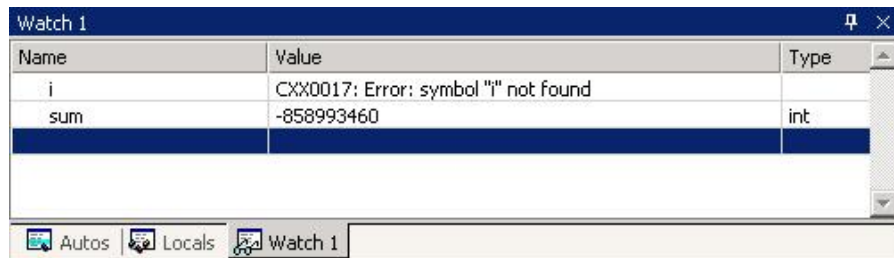
Именно эта стрелка указывает, какая строка кода сейчас «выполняется». Для передвижения на следующий шаг программы снова нажмите F10. И вы попадете в следующую строку:

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

Обратите внимание на то, что внизу экрана у вас располагается набор вкладок для анализа переменных:



Autos — эта вкладка предназначена для просмотра значений переменных, которые существуют в момент выполнения текущей строки кода. Вписать на данной вкладке что-то от себя нельзя — это автоматическая функция.



Watch — предназначена, как раз для тех случаев, когда необходимо самому выбрать переменную для просмотра. Вы просто вписываете в поле Name название переменной и она отображается независимо от выполняемого кода.

Теперь просто нажимая F10, «проойдитесь» по коду и посмотрите, как будут изменяться данные во вкладках.

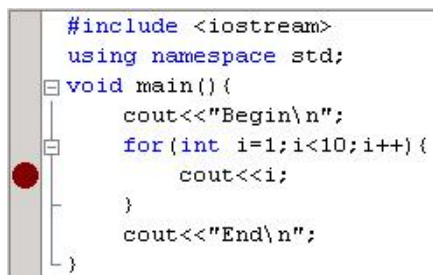
Примечание: Если вы хотите остановить отладчик раньше, чем завершиться анализ кода нажмите сочетание клавиш *Shift+F5*

Примечание: Вы, наверное, заметили, что отладчик начинает анализ с первой строки программы. Если хотите запустить отладчик с определенной строки программы — установите курсор в необходимую строку и нажмите сочетание клавиш *Ctrl+F10*

Точка останова

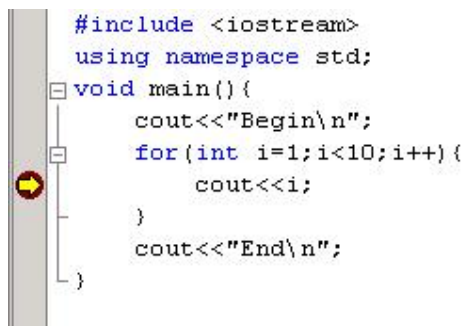
Рассмотрим ситуацию, когда нам необходимо выполнить отрезок кода и остановившись в определенном месте, запустить отладчик. Для этого используется так называемая — точка останова.

Наберите следующий код — установите курсор в строку `cout<<i;` и нажмите клавишу F9. Рядом со строкой появилась красная точка это и есть точка останова.



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

Теперь нажмите F5, программа запустится, выполнится до того момента, где установлена точка останова и перейдет в режим отладчика.



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

The image shows a code editor with a C++ program. A yellow arrow breakpoint is set on the line `for(int i=1;i<10;i++){`. The code prints "Begin", then enters a loop that prints numbers 1 through 9, and finally prints "End".

Обратите внимание на состояние консоли (окна программы). Здесь отображается все, что успело произойти:



Begin

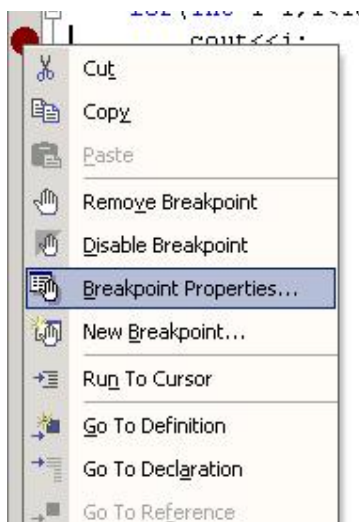
Далее следует обычная работа отладчика. Перемещайте желтую стрелку с помощью F10 и следите, что происходит с переменными. Кроме того, заглядывайте в окно консоли, все изменения происходящие в коде будут отображаться и там.

«Умная» точка останова

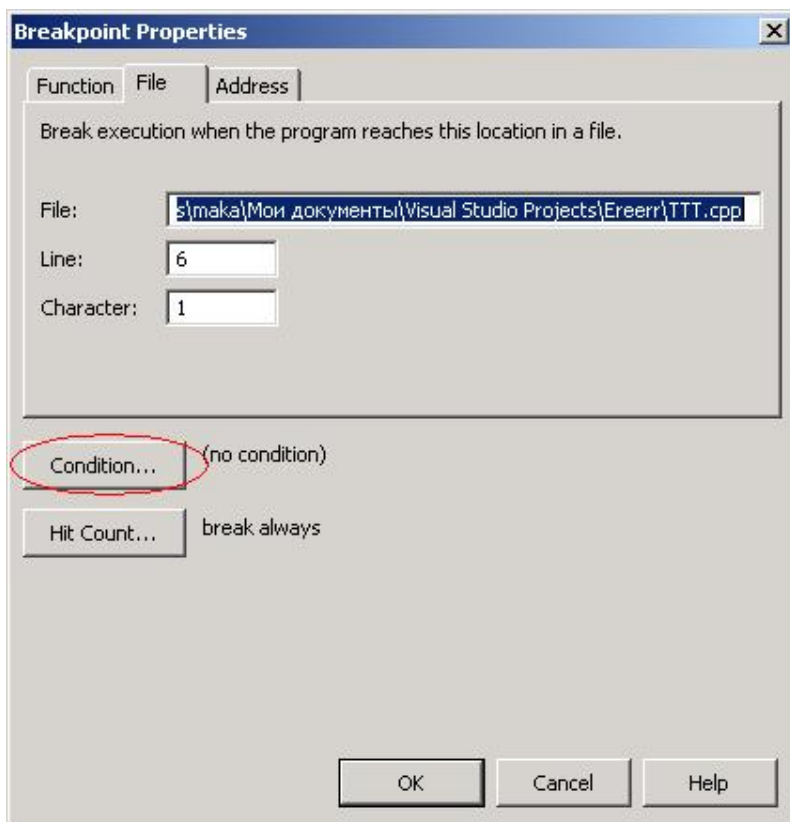
Только что мы запустили анализ программы с определенного места. Однако, заметим, что отладчик сработал сразу, как только началось выполнение тела цикла,

т. е. на первой итерации. Это неудобно, в случае, если итераций большое количество, и несколько из них вам надо пропустить. Другими словами — вы хотите начать анализ, например, с 5 итерации цикла. Решить проблему просто — сделать точку останова «умной».

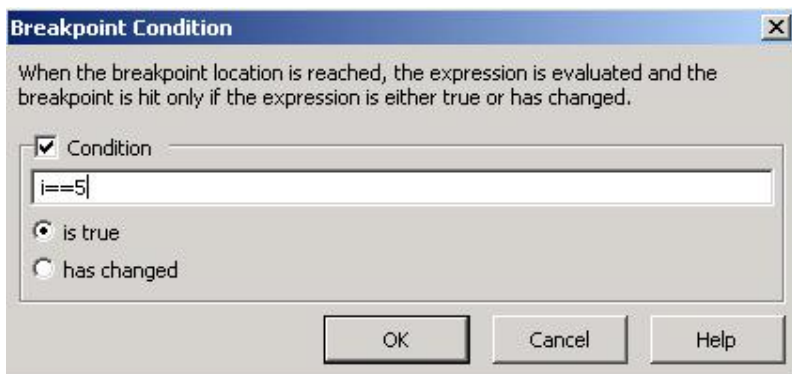
Для этого на самой точке щелкаете правой кнопкой мыши и в открывшемся меню выбираете Breakpoint Properties.



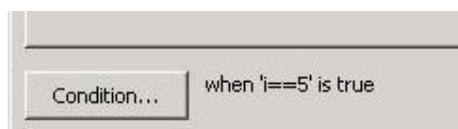
Перед вами появилось окошко. File, Line и Character — это файл, строка и позиция в которой установлена точка останова. Нас интересует кнопка Condition, рядом с которой написано — (no condition). Нажимаем её.



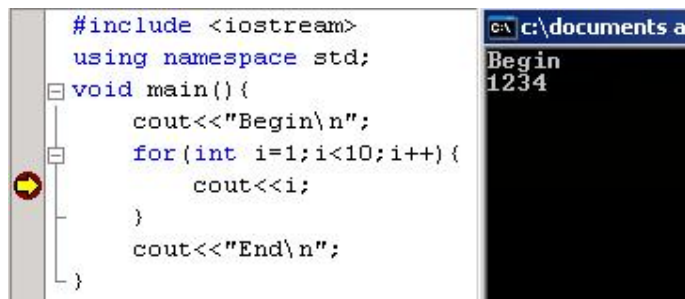
Появилось окно, в котором нам необходимо вписать условие при котором запустится отладчик. Наше условие `i==5`. Выбираем `is true` — то есть остановиться, когда условие станет истинным. Нажимаем ОК.



Теперь рядом с кнопкой другая надпись. В главном окне также нажимаем ОК.



Когда все готово, нажимаем F5 и смотрим, что произойдет. Как видите, все удачно — отладчик запустился в нужный нам момент.



Примечание: Снять точку останова можно простым нажатием F9, в той строке, где точка располагается.

Сегодня мы рассказали не обо всех возможностях отладчика. Мы еще затронем эту тему в будущем при изучении функций. А, сейчас, настоятельно рекомендуем вам — потренироваться работе с отладчиком, например, на всех примерах урока и на своем домашнем задании. Удачи!

4. Домашнее задание

1. Создать программу, которая выводит на экран простые числа в диапазоне от 2 до 1000. (Число называется простым, если оно делится только на 1 и на само себя без остатка; причем числа 1 и 2 простыми не считаются).
2. Написать программу, которая выводит на экран — следующую фигуру:

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

Ширина и высота фигуры задаются пользователем с клавиатуры.

3. При помощи цикла показать на экран календарь текущего месяца.



Урок №5

Вложенные циклы

© Компьютерная Академия «Шаг»
www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объёме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объём и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.