

COSC1101 – Programming Fundamentals

Maham Khan

Lecture - 8

Console input – through (cin >>)

- cin is an option for console input which is provided in C++ environment.
- You are required to include the following commands in the beginning of your program:
 #include <iostream>
 using namespace std;
 iostream that represents the standard input stream.
- we can retrieve characters from cin either as formatted data using the extraction operator (>>) or as unformatted data.

Console input – through cin >>

Example

```
#include <iostream>
using namespace std;
```

```
int main()
{
    double x;
    cin >> x ;
}
```

Console output – through (cout <<)

- cout is an option for console output which is provided in C++ environment.
- You are required to include the following commands in the beginning of your program:
 #include <iostream>
 using namespace std;
 iostream that represents the standard input stream.
- we can print data using cout either as formatted data using the extraction operator (<<) or as unformatted data.

Console output – through cout <<

Example

```
#include <iostream>
using namespace std;
int main()
{
    double x, y;
    cout << "Enter two floating-point values: ";
    cin >> x ;
    cin >> y;
    cout << "The average of the two numbers is: " << (x + y)/2.0
    << endl;
    return 0;
}
```

The while loop

```
void main()
{
    int j = -4;
    while(j<=0)
    {
        cout << j << endl;
        j=j+1;
    }
}
```

Tips and Traps

- The general form of while is as shown below:
- initialize loop counter ;
- while (test loop counter using a condition)
- {
 - do this ;
 - and this ;
 - increment loop counter;
- }

The while loop

- The statements within the while loop keep on getting executed until the condition being tested remains true
- when the condition becomes false, the control passes to the first statement after the body of while loop
- In place of the condition there can be any other valid expression
- as the expression evaluates to a non-zero value the statements within the loop would get executed

The while loop

- The condition being tested may use relational or logical operators as shown in the following examples:
- `while (i <= 10)`
- `while (i >= 10 && j <= 15)`
- `while (j > 10 && (b<15 || c<20))`
- As a rule the while must test a condition that will eventually become false, otherwise the loop would be executed forever, indefinitely.

The while loop: detect error


```
void main()  
{  
  int i=1;  
  while(i<=10)  
    cout << i << endl;  
}
```

Loop increment
missing!!!

This is an indefinite loop since i remains 1 forever. The correct form would be:

The while loop

```
void main()  
{  
    int i=1;  
    while(i<=10)  
    {  
        cout << i << endl;  
        i++;  
    }  
}
```



Notice the loop
increment..

The while loop

- Loop counter can be decremented as well

```
void main()
```

```
{
```

```
    int i=10;
```

```
    while(i>0)
```

```
        cout << i << endl;
```

```
        i--;
```

```
}
```

The while loop

- Loop counter can be a float as well

```
void main()
{
    float f=10.0;
    while(f>1.5)
    {
        cout << f << endl;
        f--;
    }
}
```

The while loop

- Float can be decremented as well
- The decrement can be by any value, not necessarily 1
- What is the output of the following program

```
void main()
{
    int i=1;
    while(i<=32767)
    {
        cout << i << endl;
        i++;
    }
}
```

The while loop

- It does not print numbers from 1 to 32767
- Its an infinite loop
- It prints numbers from 1 to 32767 then
- When i becomes 32768, this value falls outside the valid integer range
- It goes to the negative side and becomes -32768
- This value satisfies the non zero condition for while loop and the loop repeats itself indefinitely

The while loop

- What is the output of the following program

```
void main()
{
    int i=1;
    while(i<=10);
    cout << i << endl;
    i++;
}
```


The while loop

- another infinite loop
- The semi colon after the while make it work like

```
while ( i <= 10 )  
;  
{  
    cout << i << endl;  
    i = i +1;  
}
```

The while loop

- Since the value of “i” is not getting incremented, the control keeps rotating within the loop forever
- To vary the loop counter
 - ++
 - --
 - +=
 - -=
 - *=
 - /=
 - %= can be used

The while loop

```
void main()  
{  
    int i=0;  
    while(i++<10);  
    cout << i << endl;  
}
```

Here, firstly the comparison of value of i with 10 is performed, and then the incrementation of i takes place.

The while loop

```
void main()  
{  
    int i=0;  
    while(++i<10)  
        cout << i << endl;  
}
```

- Here first incrementation takes place, then the comparison is performed

The odd loop

- The loops that we have used so far executed the statements within them a finite number of times
- In real life programming one comes across a situation when it is not known beforehand how many times the statements in the loop are to be executed

Do-While Loop

- Its format is:
do statement while (condition);
- Its functionality is exactly the same as the while loop, except that condition in the do-while loop is evaluated after the execution of statement instead of before, granting at least one execution of statement even if condition is never fulfilled.

```
/* Execution of a loop an unknown number of times */  
void main( )  
{  
char another ;  
int num ;  
do  
{  
cout << "Enter a number: ";  
cin >> num;  
cout << "square of " << num << " is " << num*num << endl;  
cout << "Want to enter another number y/n? ";  
cin >> another;  
}  
while ( another == 'y' );  
}
```

The odd loop

And here is the sample output...

Enter a number: 5

square of 5 is 25

Want to enter another number y/n? y

Enter a number: 7

square of 7 is 49

Want to enter another number y/n? n

In this program the do-while loop would keep getting executed till the user continues to enter y.

This loop ensures that statements within it are executed at least once even if n is supplied first time itself.

Alternative way

the same functionality if required, can also be accomplished using for and while loops

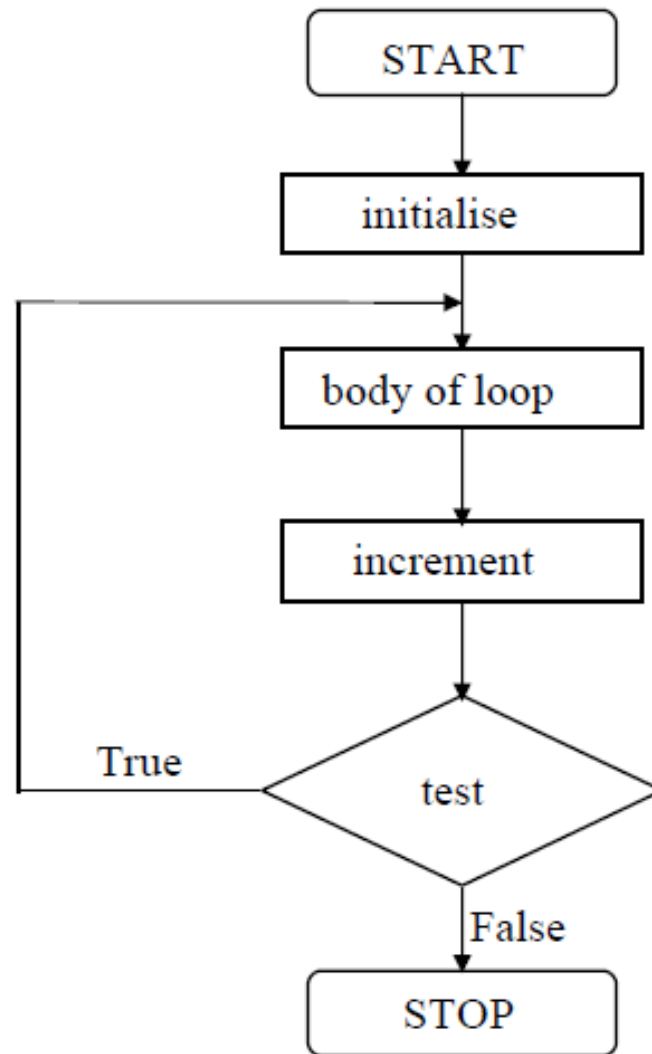
`/* odd loop using a for loop */`

```
void main( )
{
    char another = 'y' ; int num ;
    for ( ; another == 'y' ; )
    {
        cout << "Enter a number: ";
        cin >> num;
        cout << "square of " << num << " is " << num*num << endl;
        cout << "Want to enter another number y/n? ";
        cin >> another;
    }
}
```

Alternative way

```
/* odd loop using a while loop */
main( )
{
    char another = 'y' ; int num ;
    ✓ while ( another == 'y' )
    {
        cout << "Enter a number: ";
        cin >> num;
        cout << "square of " << num << " is " << num*num << endl;
        cout << "Want to enter another number y/n? ";
        cin >> another;
    }
}
```

- There is a minor difference between the working of while and do-while loops.
- This difference is the place where the condition is tested.
- The while tests the condition before executing any of the statements within the while loop
- As against this, the do-while tests the condition after having executed the statements within the loop



The *break* Statement

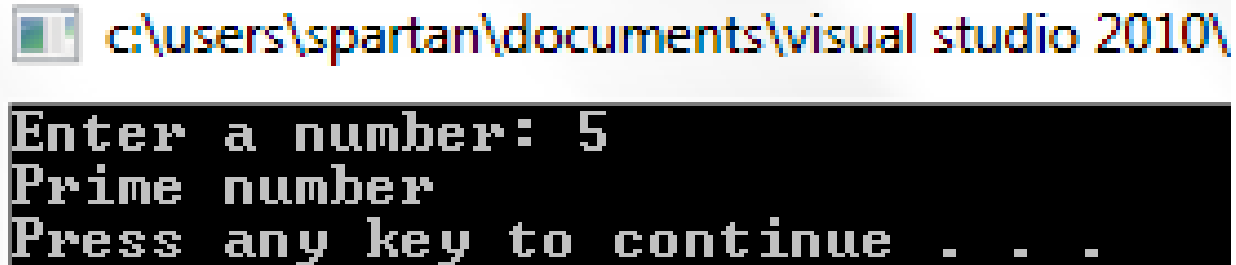
- We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test
- The keyword `break` allows us to do this
- When `break` is encountered inside any loop, control automatically passes to the first statement after the loop. A `break` is usually associated with an `if`

Example of Break Statement

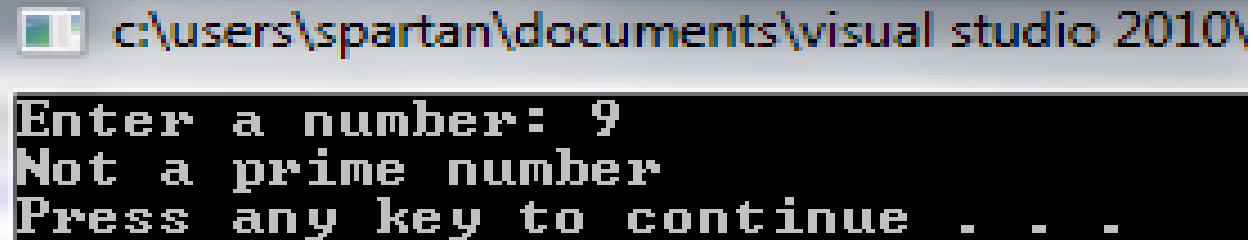
```
void main( )
{
    int num, i ;
    cout << "Enter a number: ";
    cin >> num;
    i = 2 ;
    while ( i <= num - 1 )
    {
        if ( num % i == 0 )
        {
            cout << "Not a prime number" << endl ;
            break ;
        }

        i++ ;
    }
    if ( i == num )
    cout << "Prime number" << endl;
    system ("pause");    //a way to pause the output window
}
```

Output



c:\users\spartan\documents\visual studio 2010\
Enter a number: 5
Prime number
Press any key to continue . . .



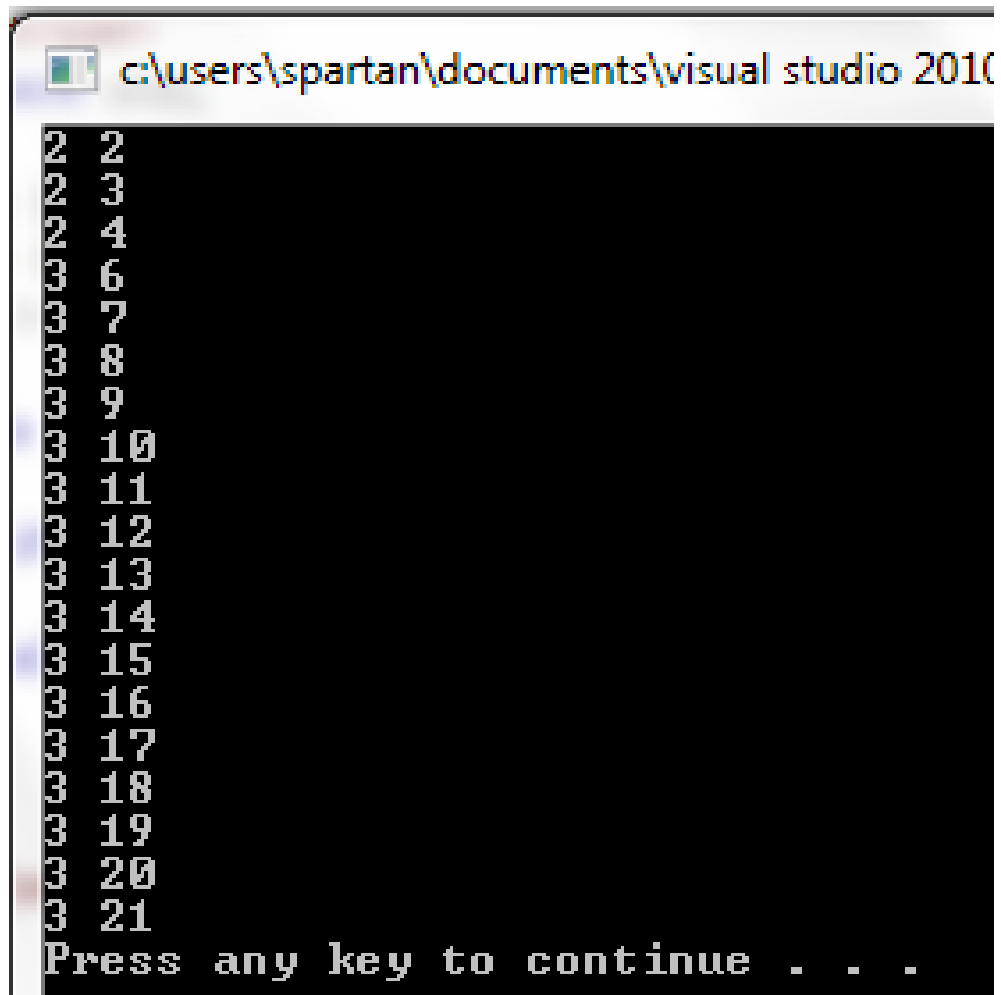
c:\users\spartan\documents\visual studio 2010\
Enter a number: 9
Not a prime number
Press any key to continue . . .

- In this program the moment num \% i turns out to be zero, (i.e. num is exactly divisible by i) the message “Not a prime number” is printed and the control breaks out of the while loop.

The Nested while loop

```
void main( ) {  
    int i = 1 , j = 1 ;  
    while ( i++ <= 10 )  
    {  
        while ( j++ <= 20 )  
        {  
            if ( j == 5 )  
                break ;  
            else  
                cout << i << " " << j << endl;  
        }  
    }  
    system("pause"); // a way to pause the output window  
}  
// In this program when j equals 5, break takes the control outside  
the inner while only, since it is placed inside the inner while.
```

OUTPUT



```
c:\users\spartan\documents\visual studio 2010
2 2
2 3
2 4
3 6
3 7
3 8
3 9
3 10
3 11
3 12
3 13
3 14
3 15
3 16
3 17
3 18
3 19
3 20
3 21
Press any key to continue . . .
```

The *continue* Statement

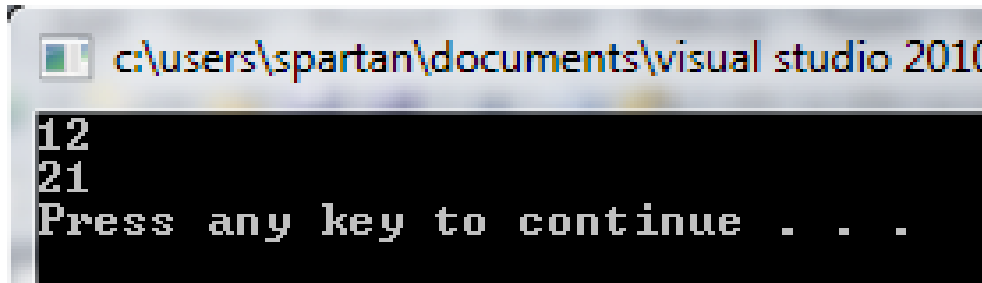
- In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed
- The keyword `continue` allows us to do this
- When `continue` is encountered inside any loop, control automatically passes to the beginning of the loop.
- A `continue` is usually associated with an `if`

```

void main( )
{
int i, j ;
    for ( i = 1 ; i <= 2 ; i++ )
    {
        for ( j = 1 ; j <= 2 ; j++ )
        {
            if ( i == j )
                continue ;
            cout << i << j << endl;
        }
    }
    system("pause"); // a way to pause the output window
}

```

OUTPUT:



```

c:\users\spartan\documents\visual studio 2010
12
21
Press any key to continue . . .

```

- Note that when the value of *i* equals that of *j*, the `continue` statement takes the control to the `for` loop (inner) bypassing rest of the statements pending execution in the `for` loop (inner).

Summary of Loops

- The three type of loops available in C are for, while, and do-while.
- A break statement takes the execution control out of the loop.
- A continue statement skips the execution of the statements after it and takes the control to the beginning of the loop.
- A do-while loop is used to ensure that the statements within the loop are executed at least once.

The Switch Statement

- we are often faced with situations where we are required to make a choice between a number of alternatives rather than only one or two.
- C provides a special control statement that allows us to handle such cases effectively; rather than using a series of if statements

Decisions Using *switch*

- The control statement that allows us to make a decision from the number of choices is called a **switch**, or more correctly a **switch-case-default**, since these three keywords go together to make up the control statement

General Syntax

```
switch ( integer expression )  
{  
  case constant 1 :  
    do this ;  
  case constant 2 :  
    do this ;  
  case constant 3 :  
    do this ;  
  default :  
    do this ;  
}
```

- The integer expression following the keyword switch is any C expression that will yield an integer value
- It could be an integer constant like 1, 2 or 3, or an expression that evaluates to an integer
- The keyword case is followed by an integer or a character constant.
- Each constant in each case must be different from all the others. The “do this” lines in the above form of switch represent any valid C statement

Example Program

```
void main( )
{
    int i = 2 ;
        switch ( i )
        {
            case 1 :
                cout << "I am in case 1 \n" ;
                break;
            case 2 :
                cout << "I am in case 2 \n" ;
                break;
            case 3 :
                cout << "I am in case 3 \n" ;
                break;
            default :
                cout << "I am in default case \n" ;
        }
}
```

Output:

I am in case 2