Loops using the while statement

- Two major types of loops

  - *Count-Controlled loop*: it is executed a specified number of times

  - *Event-Controlled loop*: it is executed a number of times which is not known in advance (everything depends on what is happening in the body of the while loop)

- Count-Controlled loops

  - It uses a variable in the loop test called the *loop control variable* (or *counter*)

  - The loop control variable has to be initialized before we enter the loop

  - The loop control variable has to be inceremented every time the body of the loop is executed

```
// Computes the sum 1 + 2 + 3 + ...+ n
#include <iostream.h>

void main()
{
  int i, n,sum;

  cout << "Enter n \n";
  cin >> n;
  sum=0;
  i=1;                    //initialization of the counter
  while(i <= n)           //test
  {
    sum = sum + i;        //repeated actions
    ++i;                  //incrementation of the counter
  }
  cout << "The sum is" << sum;
}
```

counter

```
// Computes the sum of n   numbers
#include <iostream.h>

void main()
{
  int i, n,num,sum;

  cout << "Enter n \n";
  cin >> n;
  sum=0;
  i=1;
  while(i <= n)
  {
    cout << "Enter a number \n";
    cin >> num;
    sum = sum + num;
    ++i;
  }
  cout << "The sum is" << sum;
}
```

- Things to remember when you use a count-controlled loop

  * Initialize the counter so that the condition of the while statement can be tested correctly

  * Make sure that the condition of the while statement becomes *false* at some point (e.g., by incrementing the counter after every iteration)

- *Infinite* loop

  * If the coondition never becomes *false*, then the loop will be executed forever !! (e.g., if you forget to increment the counter)

- Event-controlled loops

  - *sentinel-controlled*
  - *end-of-file-controlled*
  - *flag-controlled*

- Sentinel-controlled loop

  - It is used when the program needs to read a lot of data but we **do not** know in advance how many

  - The idea is to use a special value (*sentinel* value) to signal the program that there is no more data to be processed

  - Looping continues as long as the data value read is not the sentinel

  - When the sentinel is read, execution continues at the statement immediately following the loop

```
// Computes the sum of n    numbers
#include <iostream.h>

void main()
{
 int i, n,num,sum;

 cout << "Enter n \n";
 cin >> n;
 sum=0;
 i=1;
 while(i <= n)
 {
   cout << "Enter a number \n";
   cin >> num;
   sum = sum + num;
   ++i;
 }
 cout << "The sum is" << sum;
}
```

this while loop is executed exactly n times

```
// Computes the sum of a list of numbers
#include <iostream.h>

void main()
{
  int i, n,num,sum;


  num=0;     // initialize num to something != -9999
  sum=0;
  while(num != -9999)
  {

     cout << "Enter a number \n";
     cin >> num;
     if(num != -9999)   // if not the sentinel, then process
        sum = sum + num;
  }

  cout << "The sum is" << sum;
}
```

sentinel

the number of times this while loop is executed is not known in advance

  - *Priming read*

  * It is inefficient to initialize variables with values that are never used

* *Idea:* read the first data item before you enter the while loop

```
// Computes the sum of a list of numbers
#include <iostream.h>

void main()
{
  int i, num,sum;

  num=0;    // initialize num to something != -9999
  sum=0;
  while(num != -9999)
  {
     cout << "Enter a number \n";
     cin >> num;
     if(num != -9999)
         sum = sum + num;
  }
  cout << "The sum is" << sum;
}
```

num is initialized to a value which is not used in the computation of the sum

move these statements at the end

```
// Computes the sum of a list of numbers
#include <iostream.h>

void main()
{
  int i, num,sum;

  cout << "Enter a number \n";
  cin >> num;
  sum=0;
  while(num != -9999)
  {
    sum = sum + num;
    cout << "Enter a number \n";
    cin >> num;
  }
  cout << "The sum is" << sum;
}
```

"priming read" num is initialized by reading the first data item

process the number just read and then read the next number

- Things to remember when you use sentinels

  * The value used as the sentinel cannot be one of the inputs to be processed

  * When the sentinel is read, the while loop exits *without* processing the sentinel !!

  * The user should know in advance what sentinel value is used by your program (i.e., print a message to inform the user)

- Many times, the nature of the problem dictates the value of the sentinel

```
cin.get(inChar);
while(inChar != '\n')
{
  cout << inChar;
  cin.get(inChar);
}
```

- Confusing '==' with '=' can result in an infinite loop !!!

```
cin >> dataValue;
while (dataValue = 1)
{
   .....
  cin >> dataValue;
}
```