If we want to read in 3 numbers and find the total, the program can be written like this:

```
 0 //Read in 3 numbers: find total, version 1
 1 #include<iostream.h>
 2 void main()
 3 { int num1, num2, num3, total;
 4   cout<<"Enter a number:";
 5   cin>>num1;
 6   cout<<"Enter a number:";
 7   cin>>num2;
 8   cout<<"Enter a number:";
 9   cin>>num3;
10   total=num1+num2+num3;
11   cout<<"Total="<<total<<"\n";
12 } //main
```

The same program can also be written as shown below:

```
 0 //Read in 3 numbers: find total, version 2
 1 #include<iostream.h>
 2 void main()
 3 { int num, total=0;
 4   cout<<"Enter a number:";
 5   cin>>num;
 6   total=total+num;
 7   cout<<"Enter a number:";
 8   cin>>num;
 9   total=total+num;
10   cout<<"Enter a number:";
11   cin>>num;
12   total=total+num;
13   cout<<"Total="<<total<<"\n";
14 } //main
```

Notice that this version only uses 2 variables. The total is found by keeping a running total instead of adding all of the numbers together in one statement.

Moreover, lines 5-7 are exactly the same as 8-10, and 11-13. If we wanted to read in 10 numbers, we would only have to repeat lines 5-7 several more times. For 100 numbers the program starts to get rather unwieldy. There has to be a better way!

The better way is to use a loop. There are 3 different kinds of loops: for, while, and do.

for

A **for** loop is extremely powerful. The format is shown below:

```
for (________; ____________________; ____________) _______________;
```

initialize list | Boolean expression to continue | execute at end list | body of loop

A **for** loop is a good choice when we know how many times we want the code to repeat. The program to find the total of 3 numbers is shown below.

```
 0 //Read in 3 numbers: find total using a loop
 1 #include<iostream.h>
 2 void main()
 3 { int num, total=0;
 4   int n;  //used to 'drive' the loop
 5   for(n=0; n<3; n++)
 6   {  cout<<"Enter a number:";
 7      cin>>num;
 8      total=total+num;
 9   } //end of for loop
10   cout<<"Total="<<total<<"\n";
11 } //main
```

If you work through the code carefully, you will notice that the value of n is 0, 1, and 2 inside the loop. When n becomes 3, the loop ends. When the loop ends, the next statement after the loop is executed.

Another very important point to note is that the body of the loop is either the single statement that follows the for clause, or all of the statements enclosed in curly braces{ }: lines 6 to 8. If you leave off the curly braces line 6 will execute 3 times, then you will have an opportunity to input just one number. Try it!

Another advantage of this loop is that we can ask the user how many numbers they want to add, then loop that many times instead of the constant 3.

```
 0 //Read in numbers: find total using a loop
 1 #include<iostream.h>
 2 void main()
 3 { int num, total=0;
 4   int numbers;
 5   int n;  //used to 'drive' the loop
 6   cout<<"How many numbers are there to be added:";
 7   cin>>numbers;
 8   for(n=0; n<numbers; n++)
 9   {  cout<<"Enter a number:";
10      cin>>num;
11      total=total+num;
12   } //end of for loop
13   cout<<"Total="<<total<<"\n";
14 } //main
```

We will look at several examples of for loops to give you an idea of how versatile this command can be.

Example: The program below prints 10 9 8 7 6 5 4 3 2 1 Blast OFF! Notice that we used n—to count backwards. Blast off is printed just once after the loop ends. If we switched lines 6 and 7 (put Blast off inside the loop), it would print 10 blast off, 9 blast off, etc.

```
0 //Blast off!
1 #include<iostream.h>
2 void main()
3 { int n;  //used to 'drive' the loop
4   for(n=10; n>=0; n--)  //count BACKWARDS
5   { cout<<n<<"\n";
6   } //end of for loop
7   cout<<"Blast Off!\n";
8 } //main
```

Explanation:

4: An initial value of 10 is given to num when the loop starts; the loop continues as long as num is greater than zero. After executing the body of the loop, 1 is subtracted from num, and the test is performed again.

5: There is only one statement in the body of the loop, so no braces are necessary. Most loops have more than one statement and have the body of the loop enclosed in braces. This could have been written as:

```
0 //Blast off!
1 #include<iostream.h>
2 void main()
3 { int n;  //used to 'drive' the loop
4   for(n=10; n>=0; n--)  //count BACKWARDS
5     cout<<n<<"\n";
6   cout<<"Blast Off!\n";
7 } //main
```

If you think the code is clearer with the curly braces, by all means put them in. Note that indenting line 5 helps make it clearer to humans that the body of the loop is line 5, but makes ***no*** difference to the compiler.

Example: Instead of counting by 1 or –1, this program counts by 5. Notice that the loop continues as long as n is less than 50. The loop ends when n is 50, and 50 does **not** print.

```
0 //Count  5, 10, 15, 20, 25, 30, 35, 40, 45
1 #include<iostream.h>
2 void main()
3 { int n;  //used to 'drive' the loop
4   for(n=5; n<50; n+=5)  //count by 5
5     cout<<n<<"\n";
6 } //main
```

n=5 is a shortcut for n=n+5. Either one will product the same results.

Example: The program below prints powers of 2:

```
0 //Print 1 2 4 8 16 32 64 128 256
1 #include<iostream.h>
2 void main()
3 { int n;  //used to 'drive' the loop
4   for (n = 1; n<500; n*=2)
5   {  cout<<n<<"\n";
6   } //for loop
7 } //main
```

The statement n*=2 is the same as n=n*2;

Example: The program below prints by .5:

```
0 //Print 0 0.5  1  1.5  2  2.5  3 3.5  4  4.5
1 #include<iostream.h>
2 void main()
3 { double n;  //used to 'drive' the loop
4   for (n = 0; n<5.0; n=n+0.5)
5   {  cout<<n<<"\n";
6   } //for loop
7 } //main
```

Example: The program below prints by table of fractions:

```
0 //Print table of fractions
1 #include<iostream.h>
2 void main()
3 { int n;  //used to 'drive' the loop
4   cout<<"1/N\tDecimal\n";  // \t is a tab
5   for (n = 1; n<10; n++)
6   {  cout<<"1/"<<n<<"\t"<<1.0/n<<"\n";
7   } //for loop
8 } //main
```

Notice that line 4 prints a heading before the loop starts. \t is the tab character.

Output:

```
1/n  Decimal
1/1  1
1/2  0.5
1/3  0.333333
1/4  0.25
1/5  0.2
1/6  0.166667
1/7  0.142857
1/8  0.125
1/9  0.111111
```

The variable used to drive the loop is not limited to int and double. The example below prints the alphabet.

```
 0 //Print the alphabet
 1 #include<iostream.h>
 2 void main()
 3 { char letter;  //used to 'drive' the loop
 4   for (letter='A'; letter<='Z'; letter++)
 5   {  cout<<letter; //no \n, alphabet will be one line
 6   } //for loop
 7   cout<<"\n";  //go to a new line after printing alphabet
 8 } //main
```

Explanation: letter++ increments to the next letter. ++ can be used with int and char but not with double.

Experiment: Write a program to print the alphabet backwards.

Example: The next example shows that the body of the loop can be empty. The program converts an integer to binary. The user enters an integer: use 56 for example. The first loop multiplies place by 2 until place is greater than 56. The second loop works back down each place to print the binary digits.

```
 0 //Convert integer to binary
 1 #include<iostream.h>
 2 void main()
 3 { int num, place;
 4   cout<<"Enter a number:";
 5   cin>>num;
 6   for (place=1;place<=num;place*=2); //find max. place
 7   for (place/=2; place>=1;place/=2)  //print digits
 8   { cout<<num/place;
 9     num=num%place;
10   }  //for loop
11   cout<<"\n";
12 } //main
```

This can be a little confusing, especially if you are not too familiar with binary numbers and the % - modulus operator is unfamiliar. However, it does show how powerful a for loop can be.

Execution	**place**	**num**	**printout** (*with no line feed*)
user enters 13		13	
first loop initialize	1		
*place=place*2*	2		
“	4		
“	8		
“, *loop ends with this*	16		
Execution	**place**	**num**	**printout** (*with no line feed*)
second loop initialize	8	13	*place > num so divide by 2*
print num/place			**1**
num=num%place		5	
place=place/2	4		
print num/place			**1**
num=num%place		1	
place=place/2	2		
print num/place			**0**
num=num%place		1	
place=place/2	1		
print num/place			**1**
num=num%place		0	
place=place/2	0		
loop ends			