History is about loops and continuums.
-Mike Bidlo

Chapter 6

Iteration and Reusability

Iterations, Loops, Nested Loops and Functions

Summary

In this chapter we will discover the true power of iterative constructs (called for loop, **while** loop and **do ... while** , **for** loop). After learning these constructs we will integrate these ideas in solving more challenging problems related to designing functions and conquer bigger problems by following the idea of reusability and extensibility.

Loops and their variants

In C++ we have three types of loop versions. It will be a nice idea to know how their syntax is and at what time one should use them.

The **while** statement

You may think of this instruction is like, if you want to do a repeated task such that if the **condition** is **true** then **statement** instruction should be executed and then the **condition** should be checked again and if the **condition** is **true**, the **statement** part should be repeated until the **condition** part gets **false**.

The syntax of this loop is the following

```
while (condition)
  statement;
```

It is possible that this **statement** part consists of multiple statements then in that scenario we will use the block statement { } in place of the statement and then in that body of **{ }** we will write all the required instructions. Here is its syntax,

```
while (condition)
{
   Statement1;
   Statement1;
}
```

There are many examples in daily life where you will be needing this type of work. E.g if you want to design a robot which should serve all the people in a restaurant with tea. Now if you want to design a code then the condition part must check that whether there is any person left in the room without any tea. Now if robots checks that there is no customer present in the restaurant then it should immediately take rest. Hence first it needs to check the condition. Now if there are few customers in the restaurant then after serving one, the

robot will check the condition again i.e. if there is any person left without tea, if so then it has to serve tea to him. Hence we will be needing this sort of loop technique while coding the robot.

Examples: Use of kbhit() and while loop.

EXAMPLE 1 prints "Hello" message until any key is pressed. Similarly **Example 2** prints whichever key you will press it also side by side prints in the message the key pressed and its ASCII value. The loops ends until the user presses ESC (which has the ASCII value 27).

EXAMPLE 1

```
1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  int main( )
5  {
6     int num = 1;
7     while (! kbhit() )
8     {
9        cout<<num<<": Hello..."<<endl;
10       num++;
11    }
12 }
13
14
```

EXAMPLE 2

```
int main( )
{
   char ch;
   cout<<endl;
   while(ch!=27)  // ASCII value of ESC is 27.
   {
      if(kbhit())
      {
         ch = getch();
         cout<<"\t"<<ch<<" Key pressed...! Its ASCII value is: "<<int(ch)<<endl;
      }
   }
}
```

Link: http://codepad.org/cd2nmd3M

Output 1

```
1: Hello...
2: Hello...
3: Hello...
4: Hello...
5: Hello...
```

Output 2

```
1 Key pressed...! Its corresponding ASCII value is: 49
6 Key pressed...! Its corresponding ASCII value is: 54
g Key pressed...! Its corresponding ASCII value is: 103
A Key pressed...! Its corresponding ASCII value is: 65
 Key pressed...! Its corresponding ASCII value is: 27
```

Description: In this example we have used **kbhit()** function defined in **conio.h** library, it returns true in case any key is pressed (and it is still in the buffer and we haven't received it yet using getch() function).

Q1: Finding the arithmetic series size (1+2+3+... + N) which accumulates to the given bound B?

Q2: Finding the harmonic series size (1/1+½+⅓+¼+⅕+.....) which accumulates to the given bound?

The **do...while** statement

The only difference between **while** and **do ... while** statement is that in **do while,** the **statement** inside **do** is executed for the first time without checking the **condition** inside the **while** parenthesis **( )**. There are many scenarios in real life where we might be needing this sort of work. E.g If you would like to do validation for some input and want to make sure that until the user enters valid input the program should keep asking from the user to enter the input again and again.

The syntax of this type of iteration is the following:

do
Statement;
while(condition);
	do
	{
	Statement 1;
	Statement 2;
	}
	while(condition);

We had seen its examples in the previous chapter for validating the date of birth and current date for age-calculator. See the syntax in the above table. In the second syntax, if you would like to perform many tasks associated with each iteration of **do … while** then you need to use the block statement **{ }** so that you can add many statements associated to the do **{ }** part of the loop. Let us do a very beautiful example related to **do ... while**.

Escaping the Well (Help the frog to free)

Let us say there is a frog living in a well, but now would like to leave the well. If the walls of the well are slippery, and the frog can jump on the wall with quantity J and it will slip with quantity S, find it will take how many steps for the frog to escape from the well, if the height of the well is H units. Keep this thing in mind that it is possible that the frog cannot escape from the well, in that case it must output that the frog is imprisoned in the well.

In implementing, the first thing we need to check is if the slipping value **S** is greater than jump value **J** then the frog cannot escape. Now if you would like to solve this problem there are two very important steps happening again and again. The distance covered by the frog is covered by **J** units in each step and it also get slip by **S** quantity. Intuitively it seems that the total number of jumps should be approximately **H/(J-S)**. As the accumulative effect of the jump is J-S on every step. But if you consider its boundary case than it is possible that on the last jump taken by the frog doesn't require the slipping, as it will already be outside the well. So here is the program which solves this problem.

```
1  int main()
2  {
3     int H=10, J=5, S=3, D=0; // H is Height, J is jump, S is slip, D is distance covered.
4     cout<< “enter H, J, S respectively : ”;
5     cin>>H>>J>>S;
6     int JumpCount=0;
7     if(J<=S && H>J)
8     {
9        cout<<"Frog is imprisoned forever...!"<<endl;
10       return 0;
11    }
12    do
13    {
14       D+=J;
15       if(D<H)
16             D -= S;
17       JumpCount++;
18    }
19    while(D<H);
20    cout<<"Total Jumps: "<<JumpCount<<endl;
21    return 0;
22 }
```

OUTPUT

```
enter H, J, S respectively : 10 5 3
Total Jumps: 4
```

```
enter H, J, S respectively : 10 3 3
Frog is imprisoned forever...!
```

If the jumping factor **J** is less than or equal to the slipping factor than in case if the jump length is lesser than height than the frog is imprisoned forever (see line 7). Otherwise , of course frog can escape now. In the code D variable measures the height covered by the frog so far (which in the beginning of the loop must be 0, as frog is at the pit of well). After each jump the value of **D** should be accumulated by adding the jumping factor **J** and subtracting the slipping factor **S** (ONLY IF after taking the jump the frog is still in the well, hence you can see the condition on Line 15). The moment the **D** distance covered becomes equal or greater than **H** the loop breaks and the **JumpCount** will have the total jumps taken by the frog.

The **for** statement

The **for** statement is used when you exactly know the range in which you would like to iterate through. The syntax of this loop is the following

// Single statement attached with the loop **for(Initialization; Condition ; ChangeFactor)** **Statement;**	//Use **{ }** when multiple statements you would like to attach **for(Initialization; Condition ; ChangeFactor)** **{** **Statement 1;** **Statement 2;** **}**

In the **for** statement there are basically three parts. The first part is **Initialization**. In this part you may declare as many variables with one data type as you want (specially those variables you would like to use within the scope of the loop). In case if you need to use those variables after that loop ends then you might need to declare the variable before that loop begins and only initialize the variables within that scope. *Note: The* **Initialization** *part executes once at the beginning of its first iteration*.

In the **condition** part you can write any condition (it could be just one condition or a composition of multiple conditions connected by logical connectors **&&** and **||** operations). In every iterations the **condition** part need to be checked, only the body part (the statement inside the loop or block of the statement) executes if **condition** is true

The last part of the **for** loop is **ChangeFactor** of the statement you may change any variable you would like (It is not mandatory that this change has to happen to the same variable which is declared in this scope). So the sequence of execution of the for loop is the following,

1. evaluate the **Initialization** expression;
2. if the value of the **condition** expression is **false**, terminate the loop and goto **Instruction 6**.
3. execute the **Statement**
4. evaluate the **ChangeFactor** expression;
5. repeat steps 2–4.
6. Instruction/Statement after loop (All the variables allocated in **Initialization** steps destroyed).

Some examples of the **for** loop are as follows,

```
for(int i=1; i <= 10; i++)
        cout<< "Hello"<<endl;
```

```
int N = 30, Sum=0;
for(int c=1; c <= N; c+=2)
        Sum+=c ;
```

```
int S = 10, E = 20, T=23;
for(int i=S; i <= E; i++)
    cout<< T*i <<endl;
```

The first example only displays the message "hello" on console 10 times, after every iteration the loop . The second example calculates sum of odd values between 1 to N (i.e. 30 in this example), the loop starts with c =1 and change by the addition of 2 which will make c equals 3 and then 5 and so on until c becomes 31 and then the loop breaks (as condition will become false). The third examples displays the the multiple of T starting from S (in this example 23*10 = 230) till E (in this example 460).

REMEMBER

If you add a semicolon after the while or for loop statement then it means that an empty instruction is associated with the loop. The the instruction or the block of instructions followed by the loop will become independent of the loop.

e.g.

```
1.  for(int i=0 ; i<10; i++)          ;
2.      cout<< "Hello";
```

The above will run 10 times with empty instruction associated with it. The message printing statement is independent of the loop. The message **"Hello"** will be printed exactly once.

REMEMBER

It is not necessary that the termination condition is also dependent on the same variable as of the allocated variable in the loop. Here is an example which solves finding the maximum size of the Sum of the arithmetic series (starting with 1 and with difference of 1) such that the Summation is less than a given quantity N.

e.g.

```
1.  int N = 30, Sum=0, c;
2.  for( c=1; Sum <= N; c+=1)
3.      Sum+=c ;
4.  cout<< c -1;
```

The above will run until the **Sum** variable do not exceed the value 30. When the loop will terminate (as c is allocated outside the loop hence it will not be destroyed as its scope is even beyond the loop) the c will have one more value than the required series size. Hence it outputs 1 less than c.

Loop Invariant and Program Correctness

This is an extremely useful technique to figure out the correctness of your code. It also gives you insight into how your algorithm/code is actually working. Basically it says that there is a statement or condition which is true before and after each iteration of the loop, no matter which iteration of a loop is running, so in a way it is invariant to the loop.

You remember the example we did in the Chapter 1 (problem solving section where a crazy cook was drawing two beans at a time and throwing the two outside if there are two white beans and take one black from the pile and put it in the jar, and if there is at-least one black then through that black in the pile and through the 2nd inside the jar again…….). If you look carefully, the cook is repeating a step of taking the two beans at a time and throwing beans on the pile or in the jar or picking a bean and placing it in the jar. The loop/iteration invariant there is that the number of white beans always remains Odd. And that actually gives you an insight that the last bean must have to be the white one.

While proving a loop invariant property what we usually show is that which property (usually variable holding some kind of value) will always be true before and after executing every iteration. Here is the general pattern of the loop invariant:

```
  .
  .
  .

// the Loop Invariant must be true here
while ( TEST CONDITION ) {
  // top of the loop, Invariant must be true here.
  ...
  // bottom of the loop
  // the Loop Invariant must be true here
}
// Termination + Loop Invariant = Goal
```

Let us do some examples and argue how the invariants justify the correctness and working of our algorithm/code (specially loop part of the code).

// This programs computes the summations of
// value 1 upto N
 	
1. int Summation(int N)
2. {
3. int Sum=0;
4. for(int i=1; i<=N; i++)
5. {
6. Sum+= i;
7. }
8. return Sum;
9. }
	1. int MaxSoFar;
	2. cout<<"Enter the first value: ";
	3. cin>>MaxSoFar;
	4. char choice;
	5. do
	6. {
	7. cout<<"Enter the next value: ";
	8. int num;
	9. cin>>num;
	10. if(MaxSoFar<num)
	11. MaxSoFar = num;
	12. cout<<"continue (Y/N) : ";
	13. cin>>choice;
	14. }
	15. while(choice!='N');
	16. cout<<MaxSoFar<<" is the maximum value...!!!"<<endl;

Description: The Invariant is **Sum** variable has the value Summation util **i-1**. It is extremely easy to verify that Sum holds the invariant property before the start of the loop i.e. having i-1 (1-1 = 0 value) and after one iteration i will become 2 and Sum has 1 value. Similarly after 2nd iteration i has 3 value and the Sum has 3 value (which is Sum of 1+2). Hence after the loop ends the i will N+1 and the Sum will hold all the summation up to N values.	**Description:** The invariant here is **MaxSoFar** holds the Maximum value user has entered so far. It is quite easy to see that before the loop begins **the** user has entered just one value and that is saved in **MaxSoFar**. So the invariant is valid. When the loop enters first time the user enters the next value, if the next value is greater than **MaxSoFar** (the previously stored value inside the variable), than **MaxSoFar** gets updated with the new value entered, hence the invariant is maintained. After the loop breaks (when user enters 'N') the MaxSoFar has the highest value entered by the user.

Another important thing while proving the correctness of the loop driven function is to think in your brain for once that your loop will actually terminate or not. E.g. in the above two examples it's quite easy to see that first loop only runs till N and the value of i is incrementing by 1 in each step hence the loop will break once.

Similarly in the 2nd example we can see the moment user enters the choice 'N' for continue the while conditions (checking on line 15) become false and hence the loop terminates.

Let us play with the power of loops and solve very fundamental and interesting problems. We will integrate the power of using loops and functions and design very useful custom functions which we will be using again and again wherever we will be needing them.

REMEMBER
You can perform all these different types of tasks with just one loop, but the good programming practice is that you use the construct of the loop which you think is the best for solving a particular problem.

7.2 Playing with Numbers, Sequences and Series

In this section we will visit several problems related to numbers like Range Summation, Triangle numbers, printing triangle sequences and how we will be using loops and functions to make our program much more generalised and efficient.

7.2.1 Problem: Range Addition

Write a program which adds all the natural numbers within range R1 to R2 (e.g. 100 to 1000) that are multiples of two numbers D1 and D2 but not both.

We have to write a program that add all natural numbers between the given range**.** We also have a restriction that these summed values only have those which are divisible by two given numbers but not divisible by both.

To make this program we need a function **RangeAddition** that can add all natural numbers within the required range. For calculating the range summation, it requires to have two values **R1, R2,** and also for restricting over conditions the two divisors **D1, D2** must be passed as arguments/parameters.

For designing this function the first thing we need to perform is to calculate sum of numbers within range starting from **R1** to **R2**. we have to write the code something like this.

```
int Sum = R1;
Sum += R1+1;    // Sum = Sum+R1;
Sum += R1+2;
Sum += R1+3;
.
.
.
```

How many times do we need to add?
We have to write this step until **R1 + i** exceeds **R2**. (where i is the iteration integer starting from 0).

One idea is that we can do that summation until **R1+constant** exceeds **R2** but this will be very cumbersome, if we have a big range (**R1-R2**).

To overcome this problem we take help from the loops. We will write it as:

```
int Sum=0;
for(int i=1;  R1+i<=R2;  i++)
{
        sum=sum+R1+i;
}
```

It is very easy to verify that this code will do summation from **R1** to **R2**. **You** can see that while executing the code, Whenever the loop starts with any value in **i**, **Sum** already has the summation until the **R1+i-1**.

```
1  #include <iostream>
2  using namespace std;
3  //This Function Adding Numbers between a range (R1 to R2) that are multiples two divisors D1 or D2  but not both
4  int RangeAddition(int R1, int R2, int D1, int D2);
5  int main()
6  {
7     char choice;
8     int Start,End, D1, D2,Result;
9     cout<<"Enter Range1: ";
10    cin>>Start;
11    cout<<"Enter range2: ";
12    cin>>End;
13    cout<<"Enter 2 numbers you want to get the multiples: ";
14    cin>>D1>>D2;
15    Result=RangeAddition(Start,End,D1,D2);
16    cout<<Result<<endl;
17    return 0;
18 }
19 int RangeAddition(int R1,int R2,int d1,int d2)
20 {
21    int sum=0;
22    for(int  i=0; R1+i <= R2 ;   i++)
23       if ( ( (R1+i) % d1 == 0 || (R1+i) % d2==0) && !((R1+i) % d1 == 0  && (R1+i) % d2 == 0))
24       sum=sum+(R1+i);
25    return sum;
26 }
```

Output

```
Enter Range1: 1
Enter range2: 10
Enter 2 Divisors: 1 11
The conditional Range Summation :1+2+...+10 = 55
Added values are either divisible by 1 or 11
```

Description

From Line 7 to 14 all the required parameters are populated (by taking input from the user) and on line 15 the entered value is passed to the function **RangeAddition**. For example if the two values passed are 1 and 10 then it means R1 will receive 1 and R2 will receive 10. Inside **RangeAddition** the variable is **sum** is declared with its initial value assigned to 0 on line 21. Now if you look closely the loop must add up all the range values starting from **R1, R1+1, R1+2, …..** till **R2** (i.e. **R1+i<=R2**). This is just one way to make this loop run for **R2-R1+1** times. One can easily verify that **sum** accumulates the value from start of **R1, R1+1, R1+2** …. till **R2**. And on the last line 25 the function returns the accumulated value stored in **sum**.

Exercise

You have the following three tasks to perform.

1. Verify by replacing the following loop with the previously written loop and argue by looking into watching how this loop and previous one will yield the exact same value?

1.	2.

for(int i=R1; i <= R2 ; i++)
if ((i % d1 == 0 || i % d2==0) && !(i % d1 == 0 && i % d2 == 0))
sum=sum+i;
	int times = R2-R1+1 ;
	for(int i=0;i < times ; i++)
	If (_____)
	sum=sum+ _____ ;

2. What if you calculate first the number of times your loop will run and want to compute the exact same value how will you fill the condition part and the summing part yielding the exact value?

3. Debug and see in the watches how the two programs are working.

Extending the Program to choice based option such that if after checking for one test-case the user wants to recheck some other test-case one should keep checking.

While programing when you are done, usually, with coding the solution. One needs to make sure every possibility of test-cases such that on all possible inputs cases the program must be running correctly. One possible solution to this issue is to run the program again and again and keep giving different possible inputs (for each test case) and see if the output of the programs matches with the solution.

This way is very tedious, executing again and again. Let us design a better way such that we will augment some more code around the code written in the previous example such that now after running once the program asks whether you would like to check another test-case and show the option **Y(es)/N(o)**. If the user enters **Y** then the program prompts for all the inputs again and executes the code and output the result of the second test-case. Similarly it keeps checking the test-cases until user selects the option **N**.

This can be done by using a **do{...} while** loop. Around the previously written code in the main we will put a block statement **{ }** and around it we will write **do** and **while**. Now for the condition part we need to make a **char option** variable before the **do { }** block. At the end of the last instruction of the **do { }** block we will prompt **cin>>option** from user whether "You would like to continue Y(es)/N(o):". Inside the condition part of while we will check if **option== 'Y'** then do part will execute again. Any character other Y will be considered as the termination condition of the loop. See the augmented code now. Note that option variable needs to be declared outside the **do { }** scope, if you will make the **char option** variable inside the **do { }** block then in the while condition **option** variable will be destroyed and compiler will generate an undeclared identifier error.

```
1  #include <iostream>
2  using namespace std;
3  //This Function Adding Numbers between Range_1 and Range_2 that are multiples of range 1 and range2
4  int RangeAddition(int R1, int R2, int N1, int N2);
5  int main()
6  {
7     do
8     {
9
10     char option;
11    int Start,End, D1, D2,Result;
12    cout<<"Enter Range1: ";
13    cin>>Start;
14    cout<<"Enter range2: ";
15    cin>>End;
16    cout<<"Enter 2 numbers you want to get the multiples: ";
17    cin>>D1>>D2;
18    Result=RangeAddition(Start,End,N1,N2);
19    cout<<Result<<endl;
20    cout<<"do you want to continue..?(Y/N)"<<endl;
21    cin>>option;
22    }
23    while(option=='Y' || option=='y');
24    return 0;
25 }
26 .
27 .
   .
```

Output

```
Enter Range1: 1
Enter range2: 10
Enter 2 Divisors: 1 11
The conditional Range Summation :1+2+...+10 = 55
Added values are either divisible by 1 or 11


Do you want to continue... Y(es)/N(o) : Y
Enter Range1: 500
Enter range2: 3000
Enter 2 Divisors: 3 5
The conditional Range Summation :500+501+...+3000 = 1750497
Added values are either divisible by 3 or 5


Do you want to continue... Y(es)/N(o) : N
```

Description

On line 7, you can see the **char option** variable. From line 8 we have added a **do { }** block containing all the previous written code. Just before the blocks ends on line 19-20 the program prompts for an option whether the user wants to continue for checking other test cases. Line 22 contains the condition for checking that.

Program 7.2.2: Printing N^{th} Triangle Number.

Write a Function which takes as parameter N and returns N^{th} Triangle Number.

The Triangle number is a sequence in which we all add the natural numbers up to N e.g The 5^{th} Triangle Number is 1+2+3+4+5=15. In this program we need a function that all the previous upto N.

```
1  #include <iostream>
2  using namespace std;
3  //This function is Calculating the Triangle Number.
4  int TriangleNo(int N);
5  int main()
6  {
7     char option;
8     do
9     {
10    int N=0,Result;
11    cout<<"Which triangle Number Do you Want: ";
12    cin>>N;
13    Result=TriangleNo(N);
14    cout<<Result<<endl;
15    cout<<"Do you Want to continue...(Y/N)";
16    cin>>option;
17    }
18    while(option=='y' || option=='Y');
19    return 0;
20 }
21 int TriangleNo(int N)
22 {
23    int TN=0;
24    for ( int c=1; c<=N; c++ )
25       TN=TN+c;
26    return TN;
27 }
```

```
Which triangle Number Do you Want: 10
10'th Triangle # is: 55


Do you Want to continue...(Yes/No): Y

Which triangle Number Do you Want: 20
20'th Triangle # is: 210


Do you Want to continue...(Yes/No): N
```

Description

In this program we are calculating the Triangle Number. As you can see that at Line 10 we have initialized a variable **N.** At Line 11 and 12 we are taking input from the user. After this step we have called a function **TriangleNo.** In this Function we are again using **for loop** because we have a limit N. We know that at N point we have to stop so this is why we are using for loop.

Program 7.2.3: Print Triangle Number Sequence

Write a program (using previous example) which prints the triangle numbers sequence up to M (asked from the user).

In this program we have to print all the triangle numbers upto **N**. Where N is the number that is entered by the user. In this Program we are just modifying the above problem.

```
1  #include<iostream>
2  using namespace std;
3  //This Function is Calculating Triangle Numbers.
4  int TriangleNo(int n);
5  //This Function is printing Triangle Numbers.
6  void PrintTriangleSequence(int m);
7  int main()
8  {
9     int m;
10    cout<<"How Many Triangle Numbers u want to print: ";
11    cin>>m;
12    PrintTriangle(m);
13    cout<<endl;
14 }
15 .
16 .   // The code of  TriangleNo(int)  function
17 .
18 void PrintTriangleSequence(int m)
19 {
20    for(int i=1;i<=m;i++)
21    {
22       cout<<TriangleNo(i);
23       if( i != m )
24       {
25          cout<<",";
26       }
27    }
28 }
```

Output:

```
How Many Triangle Numbers u want to print: 20
1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210
```

Link: http://codepad.org/tRWkCZBf

Description:

In this program first we have initialized a variable **m** at Line 9. At line 10 and 11 we are taking input from the user.After that we have called a function named **PrintTriangle**.

7.3 Reusability and functions(Factorials, Permutations, Combinations)

Factorial

In mathematics, **factorial** is the product of all the positive integers less than or equal to n. It is denoted by **n!** For example

N! = Nx(N-1)x(N-2) x … x 1
e.g. 5! = 5 x 4 x 3 x 2 x 1 = 120

Permutation(Ordered Arrangements)

An arrangement (or ordering) of a set of objects is called a **permutation**. In a permutation, the **order** that we arrange the objects in **is important.** The number of permutations of **n** distinct objects taken **r** at a time, denoted by **nPr** where repetitions are not allowed, is given by

nPr = n(n−1)(n−2)…(n−r+1) = n!/(n-r)!

Combination(Unordered Selections)

A **combination** is a way of selecting several things out of a larger group, where order does not matter. The number of ways (or **combinations**) in which **r** objects can be selected from a set of **n** objects, where repetition is not allowed, is denoted by:

nCr = nPr / r! = n!/r!(n-r)!

Program 7.3.1: Factorial of Number

Write a function that returns the factorial of a number.

This program requires one major task and that is factorial of number, so we need a function **Factorial** that return the factorial of the user entered number. For example if the user enters 5 then factorial of 5 is 120 so the function must return 120.

Inside the **Factorial** function for computing the factorial of the received value we can use **for** loop. The reason is we know that multiplication must be accumulated through iteration within the range from 1 to the received value.

```
1  //This Program display the factorial of number
2  #include <iostream>
3  using namespace std;
4  //This function return the factorial of given number
5  int Factorial(int N); //prototype
6  int main()
7  {
8     int num;
9     cout<<"Enter a number for finding the factorial:  ";
10    cin>>num;
11    int res=Factorial(num);
12    cout<<"Factorial of "<<num<< " is = "<<res<<endl;
13    return 0;
14 }
15 int Factorial(int N)  //function
16 {
17    int result = 1;
18    for(int a=1; a<=N; a++)
19    {
20       result = result * a ;
21    }
22    return result ;
23 }
```

Link: http://codepad.org/KqmedMFI

Output

```
Enter a number for finding the factorial:  5
Factorial of 5 is = 120
```

Description

This program outputs the factorial of user entered number, at line 10 the program ask from user to enter any positive number which is passed to function **Factorial(int N)** at line 11 which return the factorial of **N.** At line 5 the prototype of function is defined and at line 15 function is defined. As you see inside the function we make an integer **result** and initialized it with 1 and after that **for** loop is used, iteration of loop starts from 1 and ends at **N**, inside the **for** loop we simply multiply integer **result** with integer **a** and then assigned the R-expression(right expression) **result*a** to **result** again and again until loop finish its iteration.

Program 7.3.2: Permutation of two Numbers

Write a function which will take two inputs and find nPr (number of permutations).

This program requires two major task first one is factorial of number and second is permutation because in permutation we required factorial function, so we need two functions, first function returns the factorial of number and that number is used for permutation and second function is for permutation. For example the factorial of 5 is 120 so we required factorial function to find permutation by using formula as described in permutation topic.

Code

```
1  //this program takes 2 numbers from user and display its permutation
2  #include <iostream>
3  using namespace std;
4  //this function return the factorial of first number
5  int Factorial(int N);
6  //this function return the permutation of numbers
7  int PermutationsCount(int N, int R);
8  int main()
9  {
10    int N, R;
11    cout<<"For Permutations enter (N, R) :  ";
12    cin>>N>>R;
13    int res = PermutationsCount(N, R);
14    cout << N<< "P" <<R <<"="<<res<<endl;
15    return 0;
16 }
17 .
18 .                  // Factorial function already implemented
19 .
20 int PermutationsCount(int N, int R)    //function nPr = N! / (n-r)!
21 {
22    int numerator=Factorial(N);
23    int denominator= Factorial(N-R);
24    return numerator/denominator;          // in short it could be written as  return
25 Factorial(N)/Factorial(N-R);
26 }
```

Output

```
For Permutations enter (N, R) :  5 3
5P3=60
```

Description

This program output the permutation of entered numbers by user. As you see we make two functions in this program at line 5 we make prototype of function **int Factorial(int N)** which returns the factorial of number. At line 20 we make another function **int PermutationsCount(int N, int R)** which returns the permutation of two numbers which is entered by user, in this function integer variable **numerator** is declared and initialized with **Factorial(N)**, at line 23 we make another variable named **denominator** and initialized it with **Factorial(N-R)** of factorial function, when the user entered both numbers first number is passed to **Factorial** function from **PermutationsCount** function at line 22 and subtraction of **N-R** is passed to **Factorial** function at line 23 and assigned it to **denominator**. At last at line 24 division(/) of **numerator** and **denominator** is returned which is further received in variable **res** in **int main()** at line 13 and output that **res** at line 14.

Link: http://codepad.org/X3fx6Wif

Program 7.3.3: Combinations of two Numbers

Write a function which will take two inputs and find nCr(number of combinations)

After taking two numbers from user, this program requires two major task first one is factorial of number and second is permutation because in permutation we required factorial function and then combination function is required in which we use factorial and permutation function. When the user enter 2 numbers we find factorial of first number and by using both numbers we find permutation and at the end combination is being solved by using these two functions.

Code

```
1  //this program takes 2 numbers from user and display its combinations
2  #include <iostream>
3  using namespace std;
4  //this function return the factorial of first number
5  int Factorial(int num);
6  //this function return the permutation of numbers
7  int PermutationsCount(int N, int R);
8  //this function return the combination of numbers
9  int CombinationsCount(int N, int R);
10 int main()
11 {
12    int N, R;
13    cout<<"For Combinations enter (N, R):  ";
14    cin>>N>>R;
15    int res = CombinationsCount(N, R);
16    cout <<N<<"C"<<R<<"="<<res<<endl;
17    return 0;
18 }
19 .
20 .    // Factorial function already implemented
21 .    // Permutations function already implemented
22 .
23 int CombinationsCount(int N, int R)    // nCr = nPr / r!
24 {
25    int denominator=Factorial(R);
26    int numerator=PermutationsCount(N, R);
27    return numerator/denominator;              // in short it can written as PermutationsCount(N,R) /
28 Factorial(R)
29 }
```

Link: http://codepad.org/AjSYWOZN

Output

```
For Combinations enter (N, R):  5 3
5C3=10
```

Description

7.4 Evens/Odds, Min/Max

Evens/Odds

All the Numbers that are divided by 2 are called even numbers. The Number which are not divisible of two are called Odd Numbers.

Problem 7.4.1 Frequency of Even and Odd Numbers

Write a program that takes inputs until user enters -1 and your program tell the frequency of even and odd nos.

In this Problem we have to write a program that will enter numbers until the -1 entered and tell how many of them from total are even and how many of them are odd. For it we will check whether the number entered is even or odd if we will add 1 in even count else we will add in odd count.

```
1  #include <iostream>//This program  will enter Some numbers from user until user enter
2  #include<Windows.h>//and tell how many entered numbers are even and how many numbers
3  using namespace std;//are odd
4  bool IsEven(int Number);//It will return true if number enter is even else it will return false
5  bool IsMinusOne(int Number);//It will return true if number enter is -1 else false
6  void Print(int Evens,int Odds,int Total);//it will print output on screen
7  void Message();//It will display input message on the screen
8  void Delay(int number);//It will cause delay
9  int main()
10 {
11    int Total=0,Evens=0,Odds=0,Number=0;
12    Message();
13    cin>>Number;
14    while(!IsMinusOne(Number))
15    {
16       if(IsEven(Number))
17       {
18          Evesn++;
19       }
20       else
21       {
22          Odds++;
23       }
24       Total++;
25       Print(Evens,Odds,Total);
26       Delay(10);
27       system("cls");
28       Message();
29       cin>>Number;
30    }
31    Print(Evens,Odds,Total);
32    return 0;
33 }
34 bool IsEven(int Number)
35 {
36    if(Number%2==0)
37    {
38       return true;
39    }
40    return false;
41 }
42 bool IsMinusOne(int Number)
43 {
44    if(Number==-1)
45    {
46       return true;
47    }
48    return false;
49 }
50 void Print(int Evens,int Odds,int Total)
51 {
52    cout<<"total Number Entered="<<Total<<endl;
53    cout<<"total Even Number Entered="<<Evens<<endl;
54    cout<<"total Odd Number Entered="<<Odds<<endl;
55 }
56 void Message()
57 {
```

```
58      cout<<"enter -1 to finish program"<<endl;
59      cout<<"enter Number YOu want to check"<<endl;
60  }
61  void Delay(int number)
62  {
63      for(int i=0; i<number*90000000; i++);
64  }
```

Output During Running

```
Enter -1 to finish program
Enter number you want to check: 2
Total numbers entered: 1
Total even numbers entered: 1
Total odd numbers entered: 0
```

Final Output

```
Enter -1 to finish program
Enter number you want to check: -1
Total numbers entered: 4
Total even numbers entered: 2
Total odd numbers entered: 2
```

Description

In this program our main start from the line number 11 firstly we initialize all the integers we needed. Then we Display our Message using the **Message Function** after displaying message we enter the input in **int Number** after the user enter the number we start our while loop in our while loop Braces instead of condition we called a function which is written on line number (42-49) in which there is if the number entered is -1 which is our program ending condition so it will return false else it will return true and on which iteration it will enter -1 our **while loop** end and after that our program main output is displayed on the screen.In **while loop** first we check that the number is even or odd for this we use the function **Is Even** in which there is an if condition which has condition that if the number % 2 is equal to zero then this
Function will return true else it will return false. So in the while loop if it return true we increment 1 in even else in odd. After that we increase one in total and then we use a **function delay** in which their is a single empty for loop which will run until Number Passed*90000000 we use it to stop the program for some time and time can be controlled by the number passed to it as parameter. Then we clear our screen using system cls. And at the end we again display the message and take the input of user in the number it go soo on until user enter -1.

Link

http://codepad.org/vJr7ZkSA

Problem 7.4.2: Maximum and Minimum of Numbers

Write a program that takes inputs until user enter -1 and your program tells the maximum and minimum no.

In this problem we have to write a program that will enter numbers from user until user enter -1 and tell which number is minimum and which number is maximum in the user entered number show the output on the console screen.So firstly we Make two Variables and assign the first number to max and min variables and then check the other enter number with these numbers and replace if max or min.

Code

```
#include <iostream>//It will enter numbers from user until -1 entered and tell which number is
#include<windows.h>//Maximum and which number is minimum
using namespace std;
bool MaxUpdate(int Maximum,int Number);//It will return true whether number is max or not
bool MinUpdate(int Minimum,int Number);//It will return true whether number is min or not
bool IsMinusOne(int Number);//It will return true if number entered is -1 else false
void Print(int Max,int Min,int Total);//It will Print Output on the screen
void Message();//It will show Input Message on the screen
void Delay(int number);//It will cause delay
int main()
{
   int Number=0,Max=0,Min=0,Total=0;
   Message();
   cin>>Number;
   Max=Min=Number;
   while(! IsMinusOne(Number))
   {
      Total++;
      if(MaxUpdate(Max,Number))
      {
         Max=Number;
      }
      if(MinUpdate(Min,Number))
      {
         Min=Number;
      }
      Print(Max,Min,Total);
      Delay(10);
      system("CLS");
      Message();
      cin>>Number;
   }
   if(Total!=0)
   Print(Max,Min,Total);
   return 0;
}
bool MaxUpdate(int Maximum,int Number)
{
   if(Maximum<=Number)
   {
      return true;
   }
   return false;
}
bool MinUpdate(int Minimum,int Number)
{
   if(Minimum>=Number)
   {
      return true;
   }
   return false;
}
bool IsMinusOne(int Number)
{
   if(Number==-1)
   {
      return true;
```

```
58      }
59      return false;
60  }
61  void Print(int Max,int Min,int Total)
62  {
63      cout<<"Total Numbers Entered="<<Total<<endl;
64      cout<<"Max Number ="<<Max<<endl;
65      cout<<"Min Number ="<<Min<<endl;
66  }
67  void Message()
68  {
69      cout<<"enter -1 to finish program"<<endl;
70      cout<<"enter Number YOu want to check"<<endl;
71  }
72  void Delay(int number)
73  {
74      for(int i=0; i<number*90000000; i++);
75  }
```

Output During Run

```
enter -1 to finish program
enter Number YOu want to check
2
Total Numbers Entered=1
Max Number =2
Min Number =2
```

Final Output

```
enter -1 to finish program
enter Number YOu want to check
-1
Total Numbers Entered=5
Max Number =50
Min Number =1
```

Description

This Code has the same sequence as in the problem **(7.4.1).** But as according to the problem we have to change our code a little bit Firstly we assign the first entered number to **max** and **min** as first enter number at that time is **max** and **min** we check it with all other entered numbers and change max and min according to checks and as for checks instead of **IsEven** we use two functions **MaxUpdate** and **MinUpdate**. IN **Max Update** we just return true or false **if(max<=Number)** then true else false and vice versa in the min update and the second change is in the while loop instead of adding or incrementing the value is assigned to max or min according to the conditions as you see there are two conditions first if the max update return true then max is equal to number and if min update return true then number is assigned to min. At the end in the same way it is printed on the console screen.

Link: http://codepad.org/MxoTSkKv

Problem 7.4.3 Combine above given two problems

In this problem we have to combine the above two problems of the max min and even odd the input format will be same.

```
bool Max1(int Maximum,int Number);//IT will return true if number is bigger then max
bool IsEven(int Number);//It will Return true if number is even
bool Min1(int Minimum,int Number);//it wil return true if number is smaller then minimum
bool IsMinusOne(int Number);//It will return true if number is -1
void Print(int Max,int Min,int Total,int Evens,int Odds);//It will print output on the screen
void Message();//It will print input message on the screen
void Delay(int number);//It will cause some delay according to passed number.
int main()
{
    int Total=0,Evens=0,Odds=0,Number=0;
    int Max=0,Min=0;
    Message();
    cin>>Number;
    Min=Max=Number;
    while(!IsMinusOne(Number))
    {
       Total++;
       if(IsEvenOrOdd(Number))
       {
          Evens++;
       }
       else
       {
          Odds++;
       }
       if(Max1(Max,Number))
       {
          Max=Number;
       }
       if(Min1(Min,Number))
       {
          Min=Number;
       }
      Print(Max,Min,Total,Evens,Odds);
      Delay(10);
      system("CLS");
      Message();
      cin>>Number;
    }
    if(Total!=0)
      Print(Max,Min,Total,Evens,Odds);
    return 0;
}
.
.       // Previous Implementations
.

void Print(int Max,int Min,int Total,int Even,int Odd)
{
   cout<<"Total Numbers Entered="<<Total<<endl;
   cout<<"Max Number ="<<Max<<endl;
   cout<<"Min Number ="<<Min<<endl;
    cout<<"total Even Number Entered="<<Even<<endl;
   cout<<"total Odd Number Entered="<<Odd<<endl;
}
```

Output

```
enter -1 to finish program
enter Number YOu want to check
-1
Total Numbers Entered=7
Max Number =95
Min Number =1
total Even Number Entered=3
total Odd Number Entered=4
```

Description

This code is just combination of the problem number**(7.4.1 & 7.4.2)** In it all the code and functions are same but there is a slight change in the inner part of **while loop and print function.** In **print function** we added some parameters and show complete output on the screen and in the inner part of **while loop** all the if conditions used in **(7.4.1 & 7.4.2)** and our code is completed.

Link

http://codepad.org/CA8a18Qc

7.5 GCD, LCM, Fraction and its Relatives

7.5.1 GCD

In mathematics, the **greatest common divisor** (**GCD**) of two or more integers, is the largest positive integer that divides each of the integers. For example, the **GCD** of 8,16 and 12 is 4. Sometimes it is also called HCF (Highest common factor).

7.5.1.1 Problem: Finding the GCD of two numbers.

In this problem we need to determine the biggest number which divides two number. By keeping in mind the reusability aspect of programming we will make a function **Gcd** which will take two integers **n1, n2** and determines the highest number which divides both the numbers. Our idea of solving this problem is that first we will find the smallest **s** of the two numbers **n1,n2**. Now we can easily argue that the GCD will be one of the number between **1** to **s**. So we have two option i.e. First, iterate from 1 to s and keep remembering if the any number between the range divides both n1 and n2. The last number which divides both n1 and n2 will be our GCD. Second option is if we iterate from s to 1 and return the first number which divides both n1 and n2.

Code:

```
1  #include <iostream>
2  using namespace std;
3  int Smaller(int n1, int n2);
4  int Gcd(int n1,int n2);
5  int main()
6  {
7     int n1,n2;
8    cout<<" Enter Two Numbers";
9    cin>>n1>>n2;
10   int res=Gcd(n1,n2);
11   cout<<res<<endl;
12 }
13 int Smaller(int n1, int n2)
14 {
15    if(n1<n2)
16       return n1;
17    else
18       return n2;
19 }
20 int Gcd(int n1,int n2)
21 {
22    int result=0;
23    int limit=Smaller(n1,n2);
24    for(int d=1;d<=limit;d++)
25    {
26       if(n1%d==0&&n2%d==0)
27       {
28          result=d;
29       }
30    }
31     return result;
32 }
```

```
.
.
.
.
.
.
.
.
.
.
.
.
.
.
int Gcd2(int n1,int n2)
{
   int result=0;
   int limit=Smaller(n1,n2);
   for(int d=limit;d>=1;d++)
   {
      if( n1%d ==0 && n2%d==0)
      {
         result=d; break;
      }
   }
    return result;
}
```

Output:

```
 Enter Two Numbers : 12 8
thier GCD is : 4
```

Link: http://codepad.org/fHQrqGUv

Description:

In above code at line 7 we initialize two variables **n1, n2** and at line 8 and 9 we get input from user. At line 10 we call **Gcd** which returns an integer which is stored in **res**. In **Gcd** we get smaller of two number by calling **Smaller** which returns the smaller between two numbers, which is ending point of our **For-loop** in **Gcd.** At line 11 we display this **res** on console. In above codes we use **For-loop** because we know the starting and ending point of our iteration. In **Gcd** we uses **For-loop** whose starting value is **1(int d=1)** and it iterates till **limit(d<=limit)**, which in itself checks every value from **1** to **limit** which divides and their answer is zero and if this condition fulfils it saves that value in **result** and it returns the last/biggest value which divides both of the numbers. On the other hand in **Gcd2** we starts iteration in reverse order i.e. from **limit** to **1** and returns the very first value which divides both of the numbers. It is more efficient way to determine GCD of two numbers.

7.5.1.2 Problem: Write a program that takes a number n from the user and then takes n inputs from user and find their GCD, Reuse the i) part.

Here in this problem we need to calculate GCD of **n** numbers. So the idea is to reuse the already solved problem of finding GCD of two numbers which we have already implemented using function **GCD(int n1, int n2).**

The idea for extending that previous solution is inspired by this idea that is e.g we can calculate GCD of three numbers by calculating first GCD of the two numbers and storing it in some variable and then taking GCD of previously calculated GCD and the third number entered by the user.

While focusing on this problem, we notice that we need to get number from user many times. Firstly we get numbers from user **n1** and **NextNum** and store their GCD in a variable. Now we ask another number from user in **n2(**as we won't need this variable further in our program**)** and calculating GCD of this number and the firstly calculated GCD and store this in another variable. Here we can also use **G** to store result because it is not used in program furthermore. This code is as followed.

```
1   int n=0;
2    cout<<"Enter N : ";
3    cin>>n;
4    int n1,NextNum,G;
5    cout<<"Enter N1 : ";
6    cin>>n1;
7    cin>>NextNum;
8
9    G= Gcd(n1, NextNum);
10
11   cin>>NextNum;
12   G= Gcd(G, NextNum);
13
14   cin>>NextNum;
15   G= Gcd(G NextNum);
16
17   cin>>NextNum;
18   G= Gcd(G, NextNum);
19
20   cout<<n1<<" Is the GCD"<<endl;
```

Link: http://codepad.org/zxcHuiZH

Now we can see from above that a highlighted part of code is repeated. We can use a loop here to make it a generic.

Code:

```
1  #include <iostream>
2  using namespace std;
3  int Smaller(int n1, int n2);
4  int Gcd(int n1,int n2);
5  int main()
6  {
7     int n=0;
8     cout<<"Enter N : ";
9     cin>>n;
10    int n1, NextNum, G;
11    cout<<"Enter N1 : ";
12    cin>>n1;
13    G = n1;
14    for(int i=2;i<=n;i++)
15    {
16       cout<<"Enter Next Number : ";
17       cin>>NextNum;
18       G=Gcd(G,NextNum);
19    }
20    cout<<n1<<" Is the GCD"<<endl;
21 }
```

Output:

```
Enter N : 5
Enter N1 : 12
Enter N2 : 8
Enter N2 : 16
Enter N2 : 22
Enter N2 : 18
2 Is the GCD
```

Link: http://codepad.org/Qwdn9nm3

Description

In above code at line 7 we make a variable **n(**which stores how many numbers user will enter**)** and at line 8 and 9 we get from user how many numbers he/she will enter. At line 10 we make two new variables **n1, NextNum(**which get inputs from user**).** At line 11 and 12 we get a number from user and from line 13 to 18 we use a **for-loop** as we know we want loop till **n**. Our **for-loop** is controlled by **i**, where **i** is started from from **i=2** because we have already saved one variable in **n1** and in **G**. In this loop we get another number from user and simultaneously we calculate GCD of both numbers and store this new GCD in **G.** So basically our loop saves GCD of **i-1** numbers in **n1**. And at line 19 we display the last GCD calculated on console which is stored in **G.**

7.5.2 L.C.M

A common multiple is a number that is a multiple of two or more numbers. The common multiples of 3 and 4 are 0, 12, 24, ... and so on. The **least common multiple** (**LCM**) of two numbers is the smallest non-zero number that is a multiple of both hence LCM of 3 and 4 is 12, similarly LCM of 12 and 18 is 36.

LCM, GCD and Multiplication of Two Numbers
LCM, GCD and the product of the two numbers N1, N2 are associated with a following equality

$$N1 \times N2 = LCM(N1, N2) \times GCD(N1, N2)$$

Using the above equality relation we will find the LCM of the the two numbers.

Problem: find LCM of two numbers.

In this code we need to have a function **LCM** of two numbers. This function will further call **Gcd** to determine largest divisor of both numbers and dividing by the multiplication of the two numbers by GCD will yield LCM of the two numbers.

Code:

```
1  int Gcd(int n1,int n2);
2  int Lcm(int n1,int n2);
3  int main()
4  {
5      int n1,n2;
6     cout<<"Enter Two Number";
7     cin>>n1>>n2;
8     int res=Lcm(n1,n2);
9     cout<<res<<endl;
10 }
11 .
12 .       //Here we will have all the function which we had already made in the previous examples
13 .
14
15 int Lcm(int n1,int n2)
16 {
17    return n1*n2/Gcd(n1,n2);
18 }
```

Output:

```
Enter Two Number : 12 8
24
```

Link: http://codepad.org/fJrNBRvk

Description

In above code at line 8 we make variables **n1, n2.** At line 9 and 10 we get both numbers from user. At line 11 we call **Lcm** and its returned integer is stored in **result.** At line 12 we display this **result** on console.

7.5.3 Reduced fraction

To reduce a **fraction** to **lowest terms** (also called its simplest form), divide both the numerator and denominator by the GCD. For **example**, 2/3 is in **lowest** form, but 4/6 is not in **lowest** form (the GCD of 4 and 6 is 2) and 4/6 can be expressed as 2/3.

reduced(n1,n2)=n1/GCD(n1,n2) , n2/GCD(n1,n2)

Problem: calculate and display reduced fraction.

In this problem after getting two numbers from user we need to determine GCD of these two numbers and then we divide these two numbers by this calculated GCD to get reduced form. So here we need function **ReducedFraction** which will get two integers and after finding their GCD **g** it will divide these numbers by this calculated **g** and display this result on console.

Code:

```
1  #include <iostream>
2  using namespace std;
3  int Smaller(int n1, int n2);
4  int Gcd(int n1,int n2);
5  void ReducedFraction(int n,int d);
6  int main()
7  {
8     int n,d;
9     cout<<"Enter Two Number : ";
10    cin>>n>>d;
11    ReducedFraction(n,d);
12 }
13 .
14 .    //Here we will have all the function which we had already made in the previous examples
15 .
16 void ReducedFraction(int n,int d)
17 {
18    int g=Gcd(n,d);
19    cout<<"reduced fraction is : "<<n/g<<"/"<<d/g<<endl;
20 }
```

Output:

```
Enter Two Number : 4 6
reduced fraction is : 2/3
```

Link: http://codepad.org/uz17qPyj

Description

In above code at line 8 we define two variables **'n,d'**. At line 9 and 10 we we get these two variables from user and at line 11 we call **ReducedFraction** which finds reduced fractions and display it on console.

7.6 Perfect Squares, Integer Square-root

Perfect square

Problem 7.6.1: Perfect Square or Not

Write a program that keeps on taking input from user, until the user enters -1 and tell whether the input number is a complete square or not(It should use a function that returns true if a function is complete square else returns false).

Perfect square means that a if any number multiplies with its own then give another number is a perfect square of that number. We can check by multiplying number by its own from 1 to half of the entered perfect square. Why half because if the entered number is too large then we multiply the numbers one by one that will be very lengthy and also perfect square is always smaller than the half of number.

Code:

```
1  //this program tells us that the entered number is perfect square or not until user enters -1.
2  #include <iostream>
3  using namespace std;
4  //this function tells us that either the number is perfect square or not.
5  bool IsPerfectSquare(int);
6
7  int main()
8  {
9     int num=0;
10    cout << "Enter a number to know is a perfect square : ";
11    cin>>num;
12    for(int cnt=0;num!=-1;cnt++)
13    {
14       if(IsPerfectSquare(num))
15       {
16          cout<<" The number is a perfect square. \n";
17       }
18       else
19       {
20          cout<<" The number is not a perfect square. \n";
21       }
22    cout << "Enter a number to know is a perfect square : ";
23    cin>>num;
24    }
25    return 0;
26 }
27 bool IsPerfectSquare(int num)
28 {
29    for(int cnt=0;cnt<num/2;cnt++)
30    {
31       if(cnt*cnt==num)
32       {
33          return true;
34       }
35    }
```

```
36      return false;
37  }
```

Output

```
Enter a number to know is a perfect square : 25
 The number is a perfect square.
Enter a number to know is a perfect square : 16
 The number is a perfect square.
Enter a number to know is a perfect square : 23
 The number is not a perfect square.
Enter a number to know is a perfect square : -1
```

Link: http://codepad.org/plHPrahG

Description

_____In this problem first we input a number to operate(weather is a perfect square) on, until user enters -1 this condition is written in the for loop on line 12. Then on line 14 we called our function and according to the returned value we print the message accordingly either the number is perfect square or not, and then we ask the number to input again and then our loop will check again. Thus our loop iterate until user enters -1.
Now, in our function **IsPerfectSquare** we passed a number, On line 29 our loop is being started with a counter from 0 to that n number which is equal or smaller than the square of s. Now on line 31 we check that square of counter is equal to that number or not, if the number is equal to square of counter that means the number is perfect square and we return true from that function, otherwise we return false.

Integer Square-root

Problem 7.6.2:

Modify your program to such that until user enters -1 it keeps on asking Numbers and telling side by side the number is Perfect Square of which number(make a separate function to return this number).

Code

```
1  /*this program tells us that the entered number is perfect square
2  of which number, if not then the closest number until user enters -1.*/
3  #include <iostream>
4  using namespace std;
5  //this function tells us that either the number is perfect square or not.
6  bool IsPerfectSquare(int);
7  /*this function tells us that the number is perfect square
8  or which of the closest number.*/
9  int IntegerSquareRoot(int);
10 int main()
11 {
12    int num=0;
13    cout << "Enter a number (for finding its square-root)  : ";
14    cin>>num;
15    for(int cnt=0;num!=-1;cnt++)
16    {
17       if(IsPerfectSquare(num))
18       {
19          cout<<" The number is a perfect square of : "
20             <<IntegerSquareRoot(num)<<" .\n";
21       }
22       else
23       {
24          cout<<" The number is not a perfect square. It's integer square root is : "
25                <<IntegerSquareRoot(num)<<" .\n";
26       }
27       cout << "Enter a number (for finding its square-root)  :  ";
28       cin>>num;
29    }
30    return 0;
31 }
32 bool IsPerfectSquare(int N)
33 {
34    for(int s = 0; s*s  <= N ; s++)
35    {
36       if(s*s==N)
37       {
38          return true;
39       }
40    }
41    return false;
42 }
43 int IntegerSquareRoot(int N)
44 {
45    int s;
46    for(s=0; s*s  <= N;s++)
47    {
48       // This loop will end when s will be one greater than integer square root of N.
49    }
50    return s-1;
51 }
```

Link : http://codepad.org/ZrQnH7Ox

Output

```
Enter a number to know is a perfect square : 25
 The number is a perfect square of 5 .
Enter a number to know is a perfect square : 35
 The number is not a perfect square. It is integer square of 5 .
Enter a number to know is a perfect square : 37
 The number is not a perfect square. It is integer square of 6 .
Enter a number to know is a perfect square : 8
 The number is not a perfect square. It is integer square of 2 .
Enter a number to know is a perfect square : -1
```

Description

7.7 Primes and its relatives

Prime and Composite Integers

Those positive Integers (other than 1 and 0) which are divisible by 1 and itself are called prime numbers (excluding 1 here). E.g. 2, 3, 5, 7, 11, 13 , …. Is the prime number sequence. All the other positive numbers other than 0 and 1 are called composite numbers. E.g. 4, 6, 8, 9, 10, 12, 14, 15, …

Write a function which takes a positive number as input and tells whether the number is a prime number or not?

For designing the function, let us name it as **IsPrime**, first we need to define its input and output types. **IsPrime** should take as parameters an integer hence its input type must be an **int** and return type must be **bool (true/false)** because it has to tell whether the given number is prime or not. For checking the number to be prime one needs to check for all the numbers from 2 till 1 less than the received number, let's call it **N**, whether any number divides the given value (if it does we can immediately return false as the number doesn't qualify to be a prime number and we have found it's at-least one divisor). If we checked all the possible divisors between the range of **2** to **N-1** if none of the number divides N then we should return true, as this number isn't divisible by any number other than 1 and N itself (no need to check whether the number is divisible by 1 or N (as every number is divisible by itself and 1).

Here is the implementation of the **Isprime** function.

```
1  #include <iostream>
2  using namespace std;
3  bool IsPrime(int N)
4  {
5     if(N<=1)
6        return false;
7     for(int d=2; d < N; d++)
8     {
9        if(N % d == 0)
10       {
11          return false;
12       }
13    }
14    return true;
15 }
16 int main()
17 {
18       int num;
19       cout<<"Enter a number (for Primality testing): ";
20       cin>>num;
21       if(IsPrime(num))
22             cout<<num<<" is a prime number...!"<<endl;
23       else
24                              cout<<num<<"  is  a  composite
25 number...!"<<endl;
26       return 0;
   }
```

```
.
.
bool IsPrime(int N)
{
   if(N<=1)
      return false;
   for(int d=2; d < N/2; d++)
   {
      if(N % d == 0)
      {
         return false;
      }
   }
   return true;
}
.
.
.
```

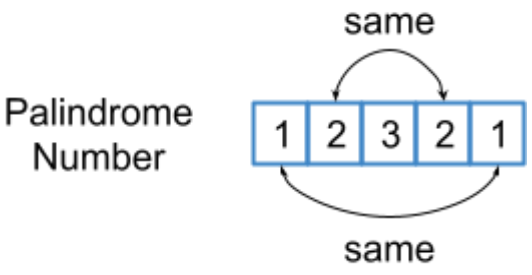
Output

```
Enter a number (for Primality testing): 21
21 is a composite number...!
```

7.8 Palindrome and Reverse and binary numbers

Palindrome

A **palindrome** is a word, phrase, number, or other sequence of characters which reads the same backward as forward, such as **12321.**



Binary Number

In mathematics and digital electronics, a **binary number** is a number expressed in the binary numeral system or base-2 numeral system which represents numeric values using two different symbols: typically **0 (zero)** and **1 (one).** For example the binary of **10** is **1010.**

Program 7.8.1: Reverse of Number

Write a function that takes an input integer N from the user and makes another variable with M reverse of the number entered and prints it.(write a function which returns M).
Input: 12345
Output: 54321
This program require one major task and that is reverse of number, so we need a function that return the reverse of user entered number. For example if the user enter 1234 then function returns 4321.

Code

```
1  //this program shows the reverse of user entered number
2  #include <iostream>
3  using namespace std;
4  //this function returns the reverse of number
5  int Reverse(int num);  //prototype
6  int main()
7  {
8     int num;
9     cout << "enter number: ";
10    cin >> num;
11    cout << "reverse is: " << Reverse(num) << endl;
12    return 0;
13 }
14 int Reverse(int num)    //function
15 {
16    int rem, m=0;
17    while(num!=0)
18    {
19       rem=num%10;
20       m=(m*10)+rem;
21       num=num/10;
22    }
23    return m;
24 }
```

Output

```
enter number: 112245
reverse is: 542211
```

Description

In this program when user enter some number it is passed to function **int Reverse(int num)** at line 14, inside this function we make two integer variables **rem**, **m** and initialize the **m** with 0 after that **while** loop is used with condition **num!=0**, inside **while** loop we are separating the passed number by using modulus(**%**) digit by digit from least significant digit side and assigned it to variable **m**, but before assigning any value you must know that we need space to accommodate that number so that's why we multiply **m** with 10. For example user enter 123 and we have to reverse it, first we take a modulus of that number as we did at line

19 and assigned it to variable **rem**, now **rem** contain 3 if we don't multiply it with 10 the next separated number is added in it (3+2 = 5) which we don't want, we want 32. After multiply **m** with 10 (as we did at line 20) **m** becomes 30, now if we add 2 in 30 it will become 32. At last we simply divide our number(**num**) with 10 to erase the last digit because we don't want it again after dividing our actual number which is 123 becomes 12. This whole process is repeated until our actual number does not become 0. See the image below you will understand it better.

Link: http://codepad.org/KerrXi3v

Program 7.8.2: Palindrome Check

Write a program(function) that Takes An Integer N from User and tells(returns) whether N is a Palindrome or Not.

Input: N = 12321
palindrome
N = 12320
not palindrome

This program require two major tasks first one is reverse of number and second one is to check that number is palindrome or not, so we need two functions, first function take a reverse of number and second function is to check that number is palindrome or not.

Code

```
1  //this program tells the number is palindrome or not
2  #include <iostream>
3  using namespace std;
4  //this function returns the reverse of number
5  int Reverse(int num);   //prototype
6  //this function tell the number is palindrome or not
7  bool IsPalindrome(int num);   //prototype
8  int main()
9  {
10    int num;
11    cout << "enter number: ";
12    cin >> num;
13    if(IsPalindrome(num))
14    {
15       cout << "It is palindrome."<<endl;
16    }
17    else
18    {
19       cout << "It is not Palindrome."<<endl;
20    }
21    return 0;
22 }
23 int Reverse(int num)   //function
24 {
25    int rem, m=0;
26    while(num!=0)
27    {
28       rem=num%10;
29       m=(m*10)+rem;
30       num=num/10;
31    }
32    return m;
33 }
34 bool IsPalindrome(int num)   //function
35 {
36    if(num==Reverse(num))
37    {
38       return true;
39    }
40    return false;
41 }
```

Output

```
enter number: 12321
It is palindrome.
```

```
enter number: 12323
It is not Palindrome.
```

Description

In this program, when user enter any number it is passed to **Reverse** function which return the reverse of that number as we use in previous program. The important thing is that we call **Reverse** function inside another function which is of **bool** type as condition **(num==Reverse(num))** at line 36 which check if the user entered number and reversed number is same then this function return **true** else return **false**. At last we

simply check in **int main()** at line 13 if the function **IsPalindrome(int num)** returns **true** then display message that the number is palindrome otherwise it is not palindrome.
Link: http://codepad.org/RQyJo1s9

Program 7.8.3: Decimal to Binary Number

Write a program(function) that takes number in decimal and outputs its binary representation. Test it using main program.
This program require one major task and that is to convert decimal number to binary so we need function, which takes a decimal number and return its binary representation.
Code

```
1  //this program display binary representation of decimal number
2  #include <iostream>
3  using namespace std;
4  //this program convert decimal number to binary
5  int DecimalToBinary(int number); //prototype
6  int main()
7  {
8     int number;
9     cout<<"Enter a number between 0 to 1023: ";
10    cin>>number;
11    cout<<"Binary representation is : "<<DecimalToBinary(number);
12    return 0;
13 }
14 int DecimalToBinary(int number)   //function
15 {
16    int i=1,rem,result=0;
17    do
18    {
19       rem=number%2;
20       result=result+(rem*i);
21       i=i*10;
22       number=number/2;
23    }
24    while(number!=0);
25    return result;
26 }
```

Output

```
Enter a number between 0 to 1023: 10
Binary representation is : 1010
```

Description

Link: http://codepad.org/QIxsatC2

Fibonacci Number

Definition
A series of numbers in which each next upcoming number is the sum of the two preceding numbers and initial two values are constant as 0 and 1. The simplest is the series 1, 1, 2, 3, 5, 8, etc.

Problem 7.9.1

Write a function which takes as parameter N>=0 and tells(returns) Nth Fibonacci number

Code

```
#include<iostream>
using namespace std;//This Program Will tell the nth Fibonacci number
int Fibonacci(int t0,int t1,int Nth);//It will return nth Fibonacci term
int main()
{
   int Nth=0;
   cout<<"Enter Nth value: ";
   cin>>Nth;
   cout<<"Nth Fibonacci is equal to : "<<Fibonacci(Nth);
   return 0;
}
int Fibonacci(int Nth)
{

   int t0=0;
   int t1=1;
   int NthFibonacci=0;
   if(Nth==0)
   {
      return t0;
   }
   else if(Nth==1)
   {
      return t1;
   }
   else
   {
      for(int i=2;i<=Nth;i++)
      {
         NthFibonacci=t0+t1;
         t1=NthFibonacci;
         t0=NthFibonacci-t0;
      }
      return NthFibonacci;
   }
}
```

Output

```
Enter Nth value: 6
Nth Fibonacci is equal to: 8
```

Description

In this code the most essential part is the **Fibonacci function** In which firstly there are two checks that if the value entered is 0 so it will return **t0** and if the value is 1 it will return **t1**. As you see for this problem we need a **loop** as if user enter five we have to write a same calculation for the five times. So after **if statement** in **else** our **for loop** start which will go up to the **nth term** in it first we add both numbers and assign it to nth **Fibonacci** in the **loop** the value stored in the **nth fibonacci** will be the **i-1th** fibonacci term and the value stored in the **t0** and **t1** will be the two preceding values. As you see the sum of **t0** and **t1** is assigned to **nth fibonacci** and the **t0-nth** fibonacci which is equal to **t1** is assigned **t0** and **t1** is assigned the **nth fibonacci**

as when **loop** start again the preceding values will have **i-1** preceding terms. In main we only enter the value and send parameters and it and show output on the screen.

Problem 7.9.2

Write a program that takes input 'k' from user which means user is interested in k Fibonacci numbers and then ask again one by one these k Fibonacci numbers who wants to know.(Reuse part i)
e.g. Fn = Fn-1 + Fn-2, F0 = 0, F1 = 1
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55…...

Code

```
#include <iostream>
using namespace std;
int Fibonanci(int t0,int t1,int Nth);
void FibonaciPrinting(int t0,int t1,int NumberOfTerms);//It will show all output on screen.
int main()
{
   int t0=0,t1=1,nthvalue=0;
   cout<<"enter Number of values you want to know: ";
   cin>>nthvalue;
   FibonaciPrinting(t0,t1,nthvalue);
   return 0;
}
int Fibonanci(int t0,int t1,int Nth)
{
   int i_1_thFibonanci;
   if(Nth==0)
   {
      return t0;
   }
   else if(Nth==1)
   {
      return t1;
   }
   else
   {
       i_1_thFibonanci=t1;
       for(int i=2;i<=Nth;i++)
       {
          i_1_thFibonanci=t0+t1;
          t1=i_1_thFibonanci;
          t0=i_1_thFibonanci-t0;
       }
       return i_1_thFibonanci;
   }
}
void FibonaciPrinting(int t0,int t1,int NumberOfTerms)
{
    int NthValue=0;
    for(int i=0;i<NumberOfTerms;i++)
    {
       cout<<"enter Nth Fibonanci number you want to know:";
       cin>>NthValue;
       cout<<NthValue<<"="<<Fibonanci(t0,t1,NthValue)<<endl;
    }
}
```

Link

http://codepad.org/zKGz63x7

Output

```
enter Number of values you want to know: 2
enter Nth Fibonanci number you want to know:6
6=8
enter Nth Fibonanci number you want to know:5
5=5
```

Description

In this code we use the same function of Fibonacci But in it the extra use function is Fibonacci printing. First we take input how many Fibonacci you want to know next that is given to Fibonacci printing in it we have a for loop in it we enter which fibonacci you want to know and then after that fibonacci function is called which will return the nth fibonacci then after that Output is shown on screen and so on the for loop work until the user entered value and output is shown on the screen.

Exercise

Problem 1

Write a program which takes as parameter T and prints all the Fibonacci numbers less than T. (Reuse part i)

Input: You want to Print Up to: 50

The Sequence Up to <50 is: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Hint

In this Problem you can use the above function of **fibonacci** just you need to design a loop which will Work until the fibonacci number return number greater than number entered.

Problem 2

Write a program which takes two parameters **Start** and **End** and prints all the Fibonacci numbers between them.(Reuse part i)

Problem 3

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

7.10 Some Problem

After doing above problems, here we have to proceed to some other problems of loops do that we can clear our concepts about loops.

Program 7.10.1: Triplets

Write a Program which finds a triplet a, b and c (3 integers) whose sum satisfies this: $a^2+b^2+c^2=1000$

In this problem we are given a condition, and by using loops we have to determine an equation which will fulfil this condition for any value. In this problem we won't need any function but we will use nested loops to solve this problem.

Nested loop
The placing of one **loop** inside the body of another **loop** is called **nesting**. When you "nest" two **loops**, the outer **loop** takes control of the number of complete repetitions of the inner **loop**. While all types of **loops** may be **nested**, the most commonly **nested loops** are **for loops**. **nested loops**.

Code

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5     int a,b,c;
6     for(a=0;a<=1000;a++)
7     {
8        for(b=0;b<=1000;b++)
9        {
10          for(c=0;c<=1000;c++)
11          {
12             if(a+b+c==1000)
13                cout<<a<<" + "<<b<<" + "<<c << " = 1000" <<endl;
14          }
15       }
16    }
17 }
```

Output

```
0 + 632 + 368 = 1000
0 + 633 + 367 = 1000
0 + 634 + 366 = 1000
0 + 635 + 365 = 1000
0 + 636 + 364 = 1000
0 + 637 + 363 = 1000
0 + 638 + 362 = 1000
0 + 639 + 361 = 1000
0 + 640 + 360 = 1000
0 + 641 + 359 = 1000
0 + 642 + 358 = 1000
0 + 643 + 357 = 1000
0 + 644 + 356 = 1000
0 + 645 + 355 = 1000
0 + 646 + 354 = 1000
0 + 647 + 353 = 1000
0 + 648 + 352 = 1000
```

Link: http://codepad.org/ugnjGmMS

Description

In above code at line 5 we declare three integers **a,b,c**. At line 6 we use a **for-loop** which is controlled by a variable **a=0** and terminates at **a=1000**. Inside this loop we declare another **for-loop** at line 8 which starts from **b=0** and terminates at **b=1000** and inside this loop we also declares another **for-loop** which starts from **c=0** and terminates from **c=1000**. Inside this loop we use an if condition which become true if sum of these loop controllers is equal to 1000 then these loops controllers are displayed on console screen.

Exercise

7.10.2 Write a Program which Prints all the triplet a, b and c which satisfies this: a+b+c = 1000.

7.10.3 Find three consecutive numbers whose multiplication is equal to the Required Number, (Your program should prompt for the number N and outputs 3-numbers those if we multiply equals to N. Also if there isn't any 3–tuple of numbers then it should say NO).

CHALLENGE SECTION

Shapes Printing

As we discussed earlier the power of functions, how we divide our problems in small pieces and make a complete picture by attaching/matching all these parts and use any small part anywhere if we want that function to perform such operation. This is called **divide** and **conquer.** You will see in a while the beauty of this.

Classically other books use nested loop to print these shapes(we will do this with classical method also) but we are going to print these generically in short simple way. You will see in while how beautifully we are going to do this.

The above shapes is our target to achieve. Now think for a while what kind of coding module we need again and again until we print these shapes in classical way.

Shape 1

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5     int height;
6     char symbol;
7     cout << "enter height: ";
8     cin >> height;
9     cout << "enter symbol: ";
10    cin >> symbol;
11    for(int ln = 1; ln <= height; ln++)
12    {
13       for(int c = 1; c <= ln; c++)
14       {
15          cout << symbol;
16       }
17       cout << endl;
18    }
19    return 0;
20 }
```

Line #	symbol(no of times)
1	1 - star('*')
2	2
3	3
.	.
.	.
ln	ln

Link: http://codepad.org/ccDtt1ya

Output

```
enter height: 5
enter symbol: *
*
**
***
****
*****
```

Description

Shape 4

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5     int height;
6     char symbol;
7     cout << "enter height: ";
8     cin >> H;
9     cout << "enter symbol: ";
10    cin >> symbol;
11    for(int ln = 1; ln <= H; ln++)
12    {
13       for(int k = H; k > ln;  k--)
14       {
15          cout<<" ";
16       }
17       for(int j = 1; j <= ln; j++)
18       {
19          cout<<symbol;
20       }
21       cout<<endl;
22    }
23    return 0;
24 }
```

Line #	space	symbol('*')
1	H-1	1
2	H-2	2
3	H-3	3
.	.	.
.	.	.
ln	H-ln	ln

Link: http://codepad.org/uQ6j9byX

Output

```
enter height: 5
enter symbol: *
    *
   **
  ***
 ****
*****
```

Description

Generic Function for Printing Stars

Code

```
#include <iostream>
using namespace std;
void PrintASymbolKtime(char sym, int k);
//this function print the symbol(sym) k times
int main()
{
   char symbol = '*';
   PrintASymbolKtime(symbol, 10);
   return 0;
}
void PrintASymbolKtime(char sym, int k)
{
   for(int a=1; a<=k; a++)
   {
      cout<<sym;
   }
}
```

Link: http://codepad.org/b0P4Z6jF

Output

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

Description

Lets see how to print different shapes using divide and conquer strategy

Shape 1

Code

```cpp
#include <iostream>
using namespace std;
void PrintASymbolKtime(char sym, int k);
void Shape1(char sym, int H);
int main()
{
   char symbol;
   int lines;
   cout << "enter symbol and number of lines: ";
   cin >> symbol >> lines;
   Shape1(symbol, lines);
   return 0;
}
.
.  //PrintASymbolKtime function here
.
void Shape1(char sym, int H)
{
   for(int ln=1; ln<=H; ln++)
   {
      PrintASymbolKtime(sym,ln);
      cout<<"\n";
   }
}
```

Line #	symbol(no of times)
1	1
2	2
3	3
.	.
.	.
ln	ln

Link: http://codepad.org/nlHcqCRS

Output

```
enter symbol and number of lines: * 15
*
**
***
****
*****
******
*******
********
*********
**********
***********
************
*************
**************
***************
```

Description

Shape 2

Code

```
1  #include <iostream>
2  using namespace std;
3  void PrintASymbolKtime(char sym, int k);
4  void Shape2(char sym, int H);
5  int main()
6  {
7     char symbol;
8     int lines;
9     cout << "enter symbol and number of lines: ";
10    cin >> symbol >> lines;
11    Shape2(symbol, lines);
12    return 0;
13 }
14 .
15 .  //PrintASymbolKtime function here
16 .
17 void Shape2(char sym, int H)
18 {
19    for(int ln=1; ln<=H; ln++)
20    {
21       PrintASymbolKtime(sym, H+1-ln);
22       cout << endl;
23    }
24 }
```

Line #	symbol(no of times)
1	15 - star('*')
2	14
3	13
.	.
.	.
ln	H + 1 - ln

Link: http://codepad.org/c1VFwSFc

Output

```
enter symbol and number of lines: * 15
***************
**************
*************
************
***********
**********
*********
********
*******
******
*****
****
***
**
*
```

Description

Shape 2

Code

```
1  #include <iostream>
2  using namespace std;
3  void PrintASymbolKtime(char sym, int k);
4  void Shape3(char sym, int H);
5  int main()
6  {
7     char symbol;
8     int lines;
9     cout << "enter symbol and number of lines: ";
10    cin >> symbol >> lines;
11    Shape3(symbol, lines);
12    return 0;
13 }
14 .
15 .  //PrintASymbolKtime function here
16 .
17 void Shape3(char sym, int H)
18 {
19    for(int ln=1; ln<=H; ln++)
20    {
21       PrintASymbolKtime(' ', H-ln);
22       PrintASymbolKtime(sym, ln);
23       cout<< "\n";
24    }
25 }
```

Line #	space	symbol('*')
1	H-1	1
2	H-2	2
3	H-3	3
.	.	.
.	.	.
ln	H-ln	ln

Link: http://codepad.org/DsvmkfkN

Output

```
enter symbol and number of lines: * 15
              *
             **
            ***
           ****
          *****
         ******
        *******
       ********
      *********
     **********
    ***********
   ************
  *************
 **************
***************
```

Description

Print Diamond

Practice Exercise

1. The following iterative sequence is defined for the set of positive integers:

   $n \rightarrow n/2$ (n is even)

   $n \rightarrow 3n + 1$ (n is odd)

   Using the rule above and starting with 13, we generate the following sequence:

   $$13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.
Which starting number, under one million, produces the longest chain?

2. The sequence of triangle numbers is generated by adding the natural numbers. So the 7th triangle number would be 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28. The first ten terms would be:

   1, 3, 6, 10, 15, 21, 28, 36, 45, 55, …

   Let us list the factors of the first seven triangle numbers:

   1: 1
   3: 1,3
   6: 1,2,3,6
   10: 1,2,5,10
   15: 1,3,5,15
   21: 1,3,7,21
   28: 1,2,4,7,14,28

   We can see that 28 is the first triangle number to have over five divisors.
   What is the value of the first triangle number to have over five hundred divisors?