

## Chapter 5

# Divide, Conquer and Rule

## Designing Functions, Menu Based Applications and Age Calculator

---

### Summary

In this chapter we will discover the power of one of the most empowering technique in programming. That technique is called Divide, Conquer and Rule approach of programming. We will learn how to construct the complete puzzle solving by first making the small pieces like jigsaw and then connect them in such a way that the entire picture of our task is clear and conquered. This fundamental technique is extremely useful in building big and challenging tasks. It won't be wrong to say that without learning this powerful idea you can never become a great computer programmer.

---

## Functions

You must have noticed in the previous chapter that many times you did a specific same type of work and then you needed to do the similar work again and again with different inputs and minor changes. For Example you did digits summation problem, you coded that, and then in some other place you needed that summation part again. One idea is to write that same code again and again, wherever needed, but that will be very tiresome. Also scaling that application which have these kinds of code will be very very cumbersome to manage as it is possible that there was some sort of error in that piece of code and you used that code at 50 other places. Now you need to search and change all those 50 places wherever you copied or written the variant of that code. So this solution will make your code so ugly that it will be very difficult to manage as a programmer. We need to come up with some smart way.

Here is a beautiful idea - while writing codes we will break our code into abstract modular black boxes (as shown in Fig 1 as **B**). The black-box **B** (you may call it by any name) will be a piece of code which will perform some specific task on the basis of provided inputs and always returns the required output (which is the expected output from this black box **B**). The advantage of designing these black boxes will be that whenever we need to solve that specific problem which our box **B** can perform, we will call that box in the code by writing just one line (which we will describe How? Wait for a while). The great advantage with this modularity will be that during coding if we need that piece of code **B** let us say 10 times then instead of writing that set of statements 10 times again and again we will just call that piece of code by invoking **B**. Also if there is a bug (error) in that module so instead of looking for changing all the places where the variant of the code was written, we will just need to change that module and wherever that module will be called the required output will be brought in action as the repercussion of the changed code.

The module, our black box, will take some inputs, which can be one or many or non, we will call these inputs **parameters**. That black box will process those **parameters** and will generate some **result/output**.



Fig 1: The prototype of the function named as **B** with generic inputs as **Input 1**, **Input 2** and so on.

Whenever we will be needing that piece of work which our module performs we will call that Black Box by its Name and pass the values to that black box. The Black Box will internally perform that task and return the desired output back to the caller (the program which called that module). The following picture **Fig 1.2** tries to explain the details of the process of calling the module.

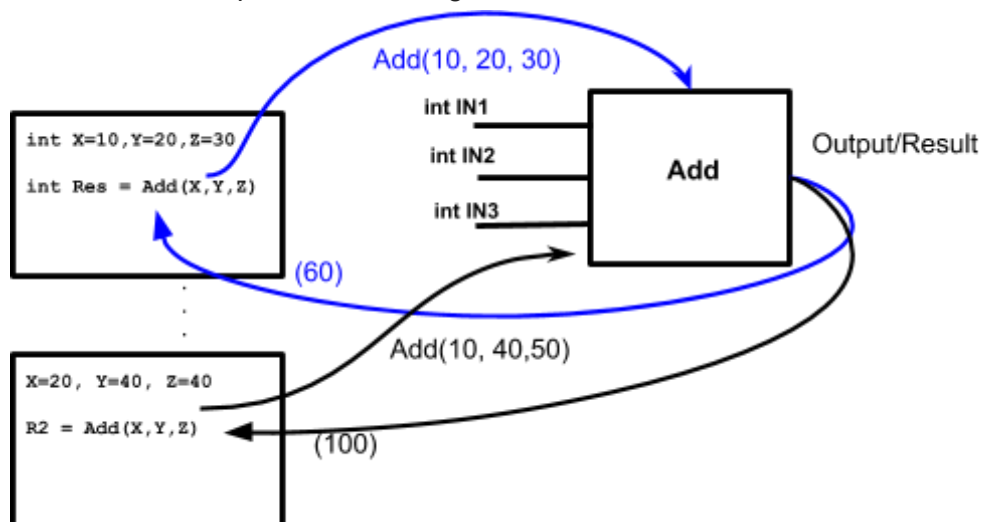


Fig 1.2: This module **Add** is taking 3 inputs and returning the summation of the 3 numbers. The module **B** is called two times at different location of the code (1st with input as (10, 20, 30) and 2nd (10, 40, 50) as inputs.

In the above example Fig 1.2, you can see that module **Add** is called with inputs as (10, 20, 30) which will be assigned in the exact same order like **IN1** will be assigned 10, **IN2** will be assigned 20 and **IN3** will be assigned 30. The module internally performs some operations and return the summation 60 (just as an example) of the 3 inputs. After few instructions passed we need to sum up again with the modified values of **X**, **Y** and **Z** respectively. Now when the new inputs are passed to module **Add** then again on the new input **(10, 40, 50)** the same code works with different inputs variables i.e. **IN1** will have 10, **IN2** will have 40, **IN3** will have 50 and return the summation 100. This is the core idea behind this modularization. We will see more examples of this high level representation of modularization and then we will implement this great technique of dividing bigger problem into smaller chunks and then module by module implement the program and scale it such that not only the code understanding will be easier but extending the code will be much elegant and abstract. Each individual module in this modularization is called **function**.

**REMEMBER**

As a programmer while using the module(function) you must never think about how the output of the module will be computed you must think of it as a black box which does the required work. This is exactly how human uses so many things in real life like when you buy the engine of the car you never intend to look into it how it is being made, you are only concerned if it will going to perform and give required power of your car. That is it.

**REMEMBER**

When you will be designing the black box i.e. your module(function). As a module designer you will never ever think about from where the module will be called from the outside world. You will only consider that given the inputs provided on the input parameters your module must performs that task correctly and generically.

**REMEMBER**

This idea of function is exactly inspired from mathematical function. Just like in mathematics every function on its every input (value from its domain) map that input to exactly one value (value from range). Exactly in a similar fashion on a certain input if you call the module multiple times then it must have to perform exactly in a similar fashion. That is why computer programmer exactly borrowed the same name for this style of coding from mathematics and calls it as function.

## Examples of Designing Functions

In this section we will have multiple examples of designing the prototype of this modularizing technique.

### What is a Prototype of a function

Prototype of a function is an abstract idea. As a programmer you must think in two modes of programming. One mode is the engineering mode of programmer who wants to use these modules to make bigger thing. And the second mode is the designer mode where the programmers are internally designing the module. Engineering mode has first very important task i.e. How to divide the bigger problem and see what possible step by step modularization of bigger program is possible. Then in a designer mode, you must understand first that your modules should have a particular task to perform on its corresponding inputs. Identifying that module name along with its inputs and output types is called prototyping of the function. Let us do several examples of prototyping of functions.

### Factorial

Factorial is a mathematical quantity which measures the number of possible arrangements. E.g. we want to compute the factorial of 5, then it is calculated as  $5 \times 4 \times 3 \times 2 \times 1 = 120$ . In general,  $n! = n \times n-1 \times n-2 \dots 1$ . From this definition, we can see that the input of this function should be an integer and output is also an integer (because integers multiplications will yield an integer). The prototype of the Factorial function will look like the following Fig 2.1.

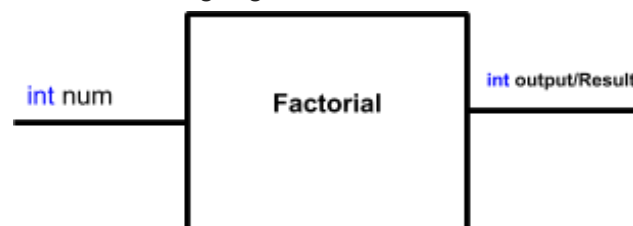


Fig 2.1: **Factorial** takes an integer input name **num** and returns an integer i.e. factorial of **num**.

## Distance

Distance is a mathematical quantity which measures distance between the two points. E.g we want to calculate distance between A(5,4) and B(10,15) then the distance(D) between these two points is ,

$$D = \sqrt{(10 - 5)(10 - 5) + (15 - 4)(15 - 4)} = 12.08$$

thus in general,

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

From this we can see that input of this function should be two points (where every point has one x coordinate and one y coordinate). Hence programmatically it must have 4 inputs integers or floats and output is a real value (which means a **float or double**, depends how precisely you want to calculate, value in term of our language C++).

In the above picture you can see two possible prototypes of the Distance function.

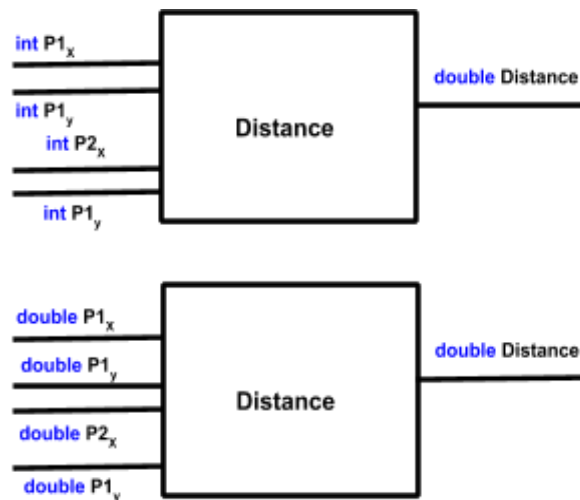


Fig 2.2: Two modular representations of Distance function one works on integers and second works on doubles

## Integers Distance Square

Integer distance square is a mathematical quantity which measures distance between two points with integer coordinates. e.g we want to calculate distance between A(5,4) and B(10,15) then the **distance square**( $D^2$ ) between these two points are  $D^2 = (10 - 5) \times (10 - 5) + (15 - 4) \times (15 - 4) = 146$ , thus in general,  $D^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$ . From this we can see that input of this function should be two integers and output is also an integer value (which means a integer value in term of our language C++). See the first module in Fig 2.3. But it is possible that the grid coordinates of the points are floating values then in that case you should have 4 doubles as inputs and output should be double. See the second module in Fig 2.3.

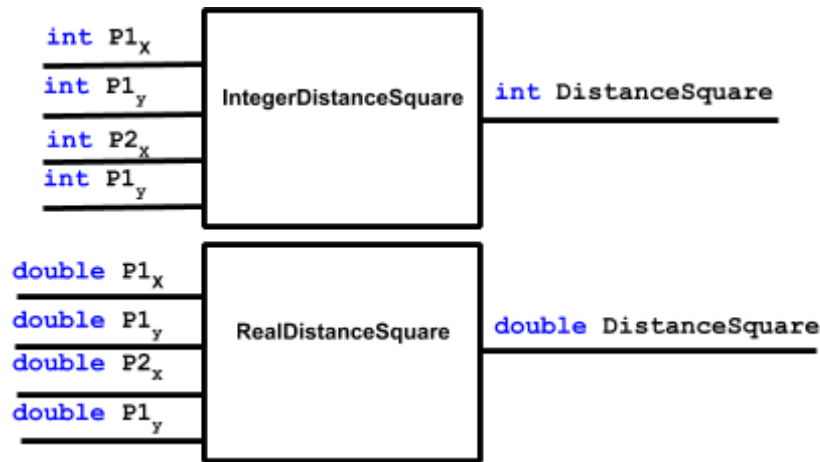


Fig 2.3: The prototype of **IntegerDistanceSquare** and **RealDistanceSquare**.

## Power

Power is a mathematical quantity which measures power of a number. E.g we want to calculate  $2^3$  then it is calculated as  $2 \times 2 \times 2 = 8$ . In general we can write it as  $N^k$  where  $k$  is a number which tells us that how many times  $N$  is to be multiplied by itself. Now For writing **Power** function we will send only two inputs as parameter and and after some calculation it will generate only one output. See Fig 2.4.

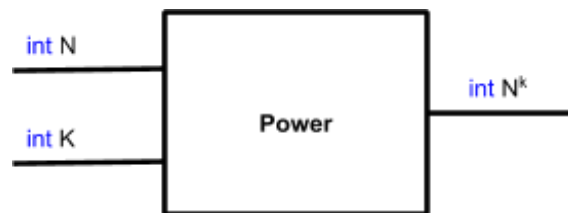


Fig 2.4: **Power** module takes two integers **N** and **K** and returns the computed integer value  $N^k$

Q<sub>1</sub> : What if the data type of **N** is **double** then what should be the datatype of the output of the Power function?

## Perfect Square Root and Integer Square Root

A perfect square root of a number **N** is a mathematical quantity **R** which if we multiply by itself  $R \times R$  then the result will be **N**. E.g.  $26^{1/2} = 5.099019513592785...$  which is a floating value. Similarly Integer Square Root is the quantity which is exactly as **R** but only its integer part. E.g **Integer\_sqrt** of 26 yields 5.

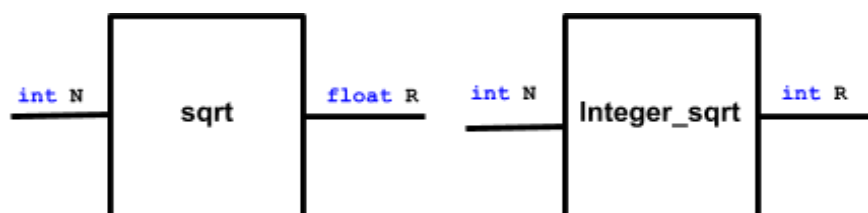


Fig 2.5: **Sqrt** takes an integer and returns the approximation of square root. **Integer\_sqrt** returns only the integer part of the square root.

## Comparison between the numbers

- Minimum of the three numbers

In this, we will mathematically compare the three number passed to inputs ( $N_1, N_2, N_3$ ) and figure out which of the three is the minimum - e.g. if we have three numbers 9, 7 and 3, then the Min is 3. So, the prototype of the function should have 3 values as parameters namely *int*  $N_1$ , *int*  $N_2$ , *int*  $N_3$  and the the output should also be an integer *int* **Min**.

- Maximum of the three numbers

In this, we will mathematically compare the three number ( $N_1, N_2, N_3$ ) and figure out which of the three is the maximum. Let us say if we have three numbers 9, 7 and 3, then the **Max** is 9.

So, the prototype of the function should have 3 arguments as parameters namely *int*  $N_1$ , *int*  $N_2$ , *int*  $N_3$  and the the output should also be an integer *int* **Max**.

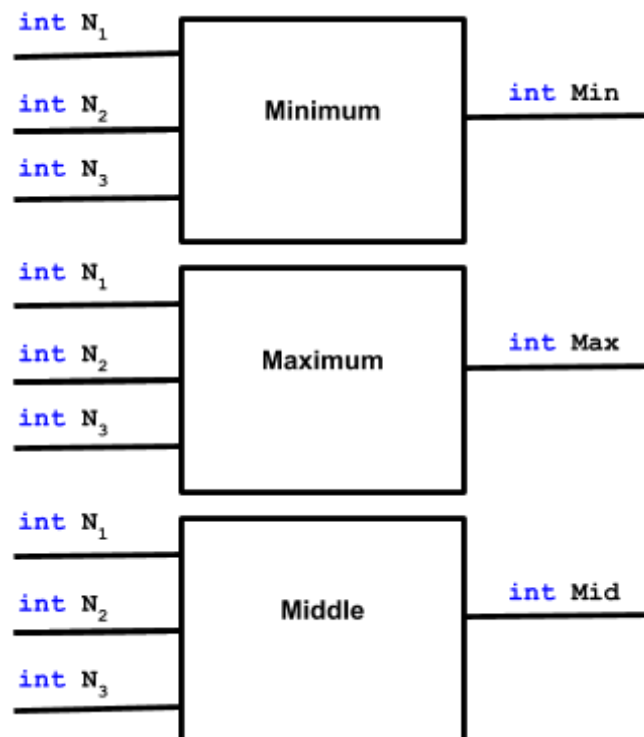


Fig 2.6: Three prototype of functions taking three inputs each and returning minimum, maximum and middle element of the three of its inputs.

- Middle of the three numbers

In this, we will mathematically compare the three number ( $N_1, N_2, N_3$ ) and figure out which of the three is the middle. Let us say if we have three numbers 9, 7 and 3, then the Middle is 7. So, the prototype of the function should have 3 arguments as parameters namely *int*  $N_1$ , *int*  $N_2$ , *int*  $N_3$  and the the output should also be an integer *int* **Mid**.

## LetterType ( IsCapital or IsSmaller or Non)

In this function we just have to tell that the given alphabet is capital, smaller or not. Here we take an assumption that this function will return an integer (1, 2, 3) which means if you pass a capital letter to this function then it return 1, or if you pass smaller letter it return 2, or if you pass parameter which is neither capital nor smaller then it return 3 (1, 2, 3 are integers that's why the return type of this function is integer). E.g if we pass alphabet 'A' in **Symbol**, (which is a capital letter) as a parameter then the output of this function will be 1 or if we pass 'a' in **Symbol**, (which is a small letter) as a parameter then the output of this function will be 2 or if we pass a symbol like '\*' which is not the alphabet then it should return 3.

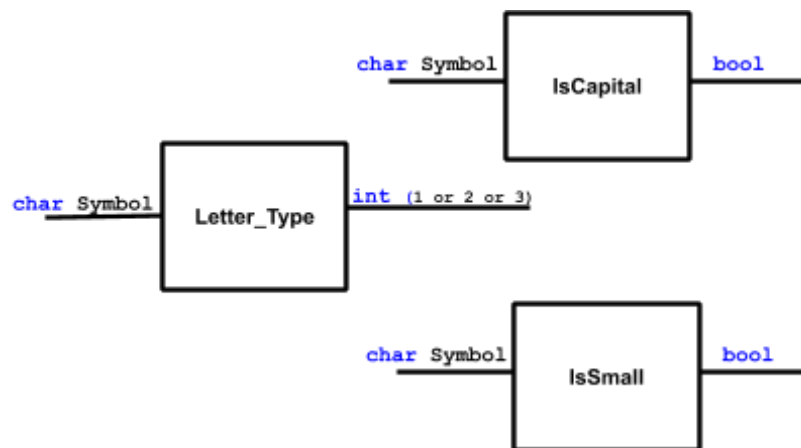


Fig 2.7: The function **Letter\_Type** will be passed with a character symbol and it will return **1** or **2** or **3**.

## Multiple of Each Other

In this function we have to tell whether the given numbers are multiple of each other or not. It has 4 cases i.e. if  $N_1$  divides  $N_2$  or  $N_2$  divides  $N_1$  or Both divide each other (the case of equal) and non is the divisor of other. We need to see that our function must return the identity of the above 4 cases. Let us make a convention that if  $N_1$  divides  $N_2$  we will return 1, if  $N_2$  divides  $N_1$  then it should return 2 if both divide each other then it should return 3 else it should return 4.

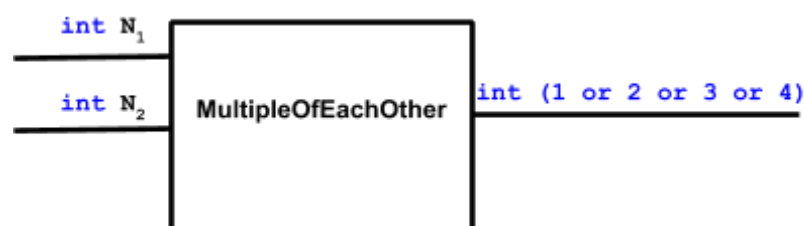


Fig 8: **IsMultiple** function will return **1** or **2** or **3** or **4** depending on which one is the multiple of other

**EXERCISE:** Design the module inputs and out for the following problem set, make their block diagram and properly write their inputs and outputs data types and names

Q1: Given three points (where each point has an x,y coordinates) the module should tell whether the inputs points represent Equilateral, Right Angled, Isosceles Right Angle, Isosceles, or Scalene Triangle?

Q2: Given four points the module should tell whether the inputs points represent Square, Rectangle, Parallelogram or Rhombus or just a general quadrilateral.

Q3: Given five points where 4 points represent the points of the rectangle and 5th points is just a location of prisoner. Design module which should represent that whether the point of location lies inside the rectangle or outside or on the rectangle.

Q4: Given three sections names (A, B, C) and their averages (out of 100 marks) design a module which should output the section's name which got the highest aggregate.

Q5: Given three values (out of 100 marks) only, design a module which should output the highest value of all the three.

Q6: Given two numbers and an operator (a character from any of the following symbols { '\*', '%', '/', '+', '-' }) design a module which should return the answer of the operator applied on the given two inputs.

Q7: Given a theta in radian compute the  $\sin \theta$ ,  $\cos \theta$ ,  $\tan \theta$ . What should be the design of the following modules? Hint: All three will have the same prototype.

Q8: Design a module which on a given real value compute the rounding off up to k digits. For example if we pass these two values 2.356900 and 3 than it should return 2.357.

## How to Implement the function and its prototype in C++

In C++, generally functions' prototype are defined in the following format,

**RETURN\_TYPE FUNCTION\_NAME(DATA\_TYPE Parameter1, DATA\_TYPE Parameter2,.....);**

First we need to tell the **RETURN\_TYPE** of the function i.e the data type of the returning value (the name abstract implementation in the above section). After that in parenthesis ( ) we are required to tell the data types of every parameter passed as inputs to function. While writing the prototype the name of the parameters is optional (though recommended). There should be comma after writing each parameter's data type (and its name if you write it).

### The Examples of prototypes

Here are few examples of prototype writing in C++.

1. `int Add(int V1, int V2);`
2. `int Factorial(int V);`
3. `double Distance(int P1x, int P1y, int P2x, int P2y);`
4. `int Power(int N, int k);`
5. `int TriangleType(int P1x, int P1y, int, int, int, int);`
6. `double RoundingOff(double Value, int K);`



You can see in example 5 that we only wrote the names of first two parameters and for rest of the 4 values we only told the data types of the inputs.

/

#### **REMEMBER**

If you write your main function first and there you want to call a specific function. Then that function needs to be defined either before the main function or at-least its prototype had to defined before the main function and then you can write your function below the main function. It is recommended that you use prototype first and then define your function below the main function. If we use the function first without writing its prototype then the compiler could not specify the function and give a error that the function is undefined (compile needs to know first before starting main that which functions are we using in our program so compiler specify the memory for the program).

#### **REMEMBER**

Internally Compiler only need to know the prototype, ignoring the variables names, because it only needs to check whether when the function is called the data inputs passed to function exactly matches with the type requirement of the function or not. If it doesn't it either warns or give error at compile time.

Now let us learn how to design and implement the function.

```
RETURN_TYPE FUNCTION_NAME(TYPE Parameter1, TYPE Parameter2,.....)  
{  
    // Function definition code is here...!  
}
```

Now, To design the overall module of the function, in c++ code, we will first tell the return type of the function, means what is the datatype of the output (which this function will return). This datatype can be integer, character or bool etc. After the return type we will write the function name and then the parameter list (the parameters list should be in the parentheses); for describing the parameter list, we will first write the datatype of the first parameter then the name of the first parameter then comma ',' then the type of second parameter then its name and then again comma ',' and so on but after the last parameter there will be no comma (',') in the parenthesis. Note that now writing the parameter name is a must because you will be using this variable inside your implementation of the function.

#### **Example of writing function and its prototype and Code**

Here is an example which ask from user to enter a year and check whether that year is a leap year or not.

```
1 #include <iostream>
2 using namespace std;
3 bool LeapYearCheck(int year); //prototype of function LeapYearCheck
4 int main()
5 {
6     int year;
7     cout << "enter year: ";
8     cin >> year;
9     if(LeapYearCheck(year))
10    {
11        cout << "It is a leap year." << endl;
12    }
13    else
14    {
15        cout << "It is not leap year." << endl;
16    }
17    return 0;
18 }
19 bool LeapYearCheck(int year) //function
20 {
21     if(year % 4 == 0 &&( year % 100 != 0 || year % 400 == 0))
22     {
23         return true;
24     }
25     return false;
26 }
```

### Output

```
enter year: 2000
It is a leap year.
```

The leap year according to Gregorian calendar is the following:

*“Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400. For example, the years 1700, 1800, and 1900 were not leap years, but the years 1600 and 2000 were.”*

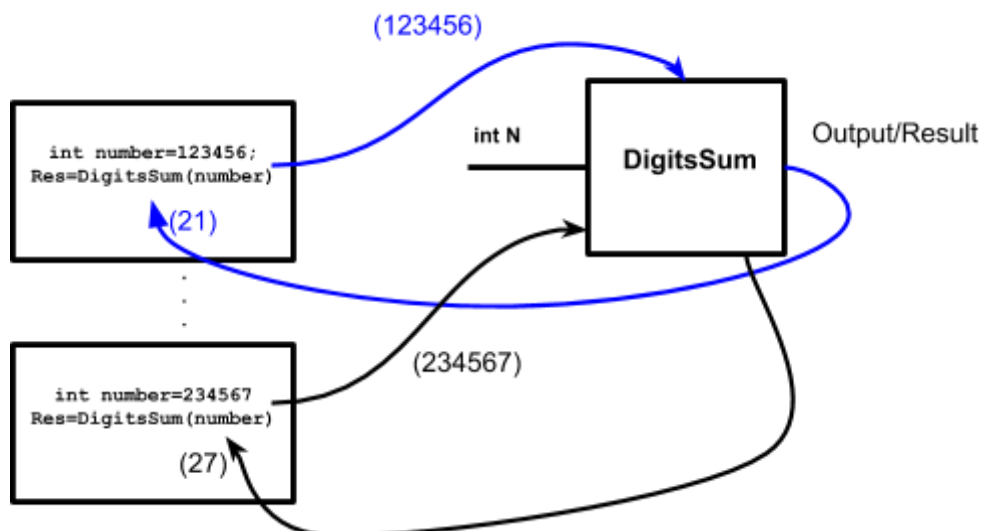
In this implementation you can see on Line 3, we have made a function prototype before the main function. Inside main on Line 9 you can see that function is called with the entered year from the user. While defining the function **LeapYearCheck(int year)** on line 19, you can notice that there is no semicolon after the this statement which means the block (the code within { } ) is all associated with this function. Inside this function the value 2000 (whatever user will enter and passed to this function) will be received and it checks the conditions and return either **true** or **false**. On Line 9 the value returned by the **LeapYearCheck** will be replaced at the exact location where the function was called - hence the if statement will become **if(true)** or **if(false)** depending upon what **LeapYearCheck** returns. And the following code executes according to **true** or **false** condition.

## Let Us Play with Functions in C++

Let us revisit most of the problems we covered in the last chapter. We'll simplify those implementations into our function oriented approach. You will see, how fascinatingly, those weary implementations will become very simple and elegant.

### Program 1.1 Digits Summation

As this program requires three major tasks one is to get number from the user and then after having that number we must find its digits by digit summation and then we have to print the result on the screen. It would be a nice idea if we make the part of code doing digits summation only as a function. As this task, we might be needing again and again. Hence we will make a separate function **DigitsSum** which should take as parameter an integer(of 6 digits) and returns the summation of all its digits. In the following program we will show you how to make that function and then use that function in main to perform the required task.



```
1 // This program will print the sum of six digits
2 #include <iostream>
3 using namespace std;
4 //This function will take a number as input for example 123456 and return 21
5 int DigitsSum(int); //prototype
6 int main()
7 {
8     int number;
9     int result = 0;
10    cout << "enter 6 digit number: ";
11    cin >> number;
12    result=DigitsSum(number);
13    cout << "sum of 6 digit number is: " << result << endl;
14 }
15 int DigitsSum(int number)
16 {
17     int sum=0;
18     sum = sum + number % 10;
19     number = number / 10;
20     sum = sum + number % 10;
21     number = number / 10;
22     sum = sum + number % 10;
23     number = number / 10;
24     sum = sum + number % 10;
25     number = number / 10;
26     sum = sum + number % 10;
27     number = number / 10;
28     sum = sum + number % 10;
29     return sum;
30 }
```

### Output

```
enter 6 digit number: 123456
sum of 6 digit number is: 21
```

### Description

Inside the main you can see that a number was initialized by taking the input from the user (let us say that input number was 123456). On line 12 the function **DigitsSum** is called. Note that while calling the function **DigitsSum** you only write in parenthesis number which means the value of the number i.e. 123456 will be passed to **DigitsSum**. You can imagine that while executing this program the computer will shift the control to the function **DigitsSum**.

Now In this particular function an integer value will be received in a variable named **number**. While designing (writing this particular function's body) we will not think about how and when this function will be called. All we need to care is that a 6 digit value will be passed to **number**. Now this function has to do the summation of all 6 digits of the value it has received. Here if you see the code from lines 17 to 28 a local variable **sum** is declared and every digit of the value present in the variable **number** is separated (just like we did in the previous chapter) and accumulated in **sum**. On the last instruction of the function, on Line 29, **return sum** means that the value of the variable **sum** should be returned to wherever the function was called from. After the execution of the last instruction the control goes back to wherever it was called from.

Now the control is back to **main** on **Line-12**. The value returned by **DigitsSum** will replace the **rvalue** (right side of the = statement where the function was called e.g. in this case the instruction will become **result = 21**, this happens in the background by the compiler) of the assignment statement.

By the way in that particular example we have assigned the returned value to **result**. It may be that the returned value could be used in some other way, like displaying on console or multiplying the returned value with some other value or it may be passed to another function - there are lots of possibilities. All you need to remember is that wherever the function will be called the returned value will replace that function call.

Finally on line 13 the result of the 6 digits calculated will be output on console screen.

### REMEMBER

**Never pass any variable which is just for local use in the function. For Example passing a sum variable to DigitsSum will loose the abstraction of the function.**

### Extending Digits Summations

Now what if you want that user requires that you should change your program such that it should prompt twice the inputs from user and displays independently that what is the digits summations of the input numbers. Notice That now you will change the main function and add 4 more lines (27-30) which will be the exact replica of lines 25 to 28 (in the 1.1 Example). The newly added code is just calling that function **DigitsSum**, according to the entered second number.

## Problem 1.2: Multiple Numbers Digits Summation

```

.
.
.
6  int main()
7  {
8      int number;
9      int sum = 0;
10     cout << "enter 6 digit number: ";
11     cin >> number;
12     sum=DigitsSum(number);
13     cout << "sum of 6 digit number is: " << sum << endl;
14     cout << "enter 6 digit number: ";
15     cin >> number;
16     sum=DigitsSum(number);
17     cout << "sum of 6 digit number is: " << sum << endl;
18     return 0;
19 }
.
.
.

```

### EXERCISE

Generalize the above program to such that the program must ask the user first a number K and then ask K times the number and displays its digits summations?

## Program 2: Determine Character type

If you look into the steps to solve the problem, the first step should be after taking the input symbol from the user identifying the symbol whether it is capital, small alphabets or any other character other than alphabets. The second step should be after identifying the symbol type printing the proper message which type of character was entered by the user. Hence for the first step we will make a function **DetermineCharType**. The second step is to make **DisplayCharType** function which should properly display the message on the basis of input **Type** passed. Now in **main**, our task is to connect the two functions with proper arguments. Here is the implementation detail how we will fulfil this task.

```
1  // This program tells either the letter is Capital Letter or Small or non//
2  #include <iostream>
3  using namespace std;
4  /*This function will get a symbol from user and return that whether the symbol is capital(1)/Small(2) letter or
5  non-alphabet(3). */
6  int DetermineCharType(char Sym);
7  //This function will just print the result depending on int(Type)
8  void PrintCharType(int T);
9  int main()
10 {
11     char symbol;
12     cout << "enter character: ";
13     cin >> symbol;
14     int Type = DetermineCharType(symbol);
15     PrintCharType(Type);
16     cout << "enter another character: ";
17     cin >> symbol;
18     Type = DetermineCharType(symbol);
19     PrintCharType(Type);
20     return 0;
21 }
22 void PrintCharType(int T)
23 {
24     switch(T)
25     {
26         case 1:
27             cout << "Capital letter" << endl; break;
28         case 2:
29             cout << "Small letter" << endl; break;
30         case 3:
31             cout << "Non" << endl; break;
32     }
33 }
34 int DetermineCharType(char Sym)
35 {
36     if(Sym >= 'A' && Sym <= 'Z')
37         return 1;
38     else if(Sym >= 'a' && Sym <= 'z')
39         return 2;
40     else
41         return 3;
42 }
```

### Output

```
enter character: g  
Small letter
```

```
enter character: )  
Non
```

### Description

Now, as you see the `int main`, at Line 14 we have initialized a symbol by taking it from the user in a variable `char symbol`, (let us say the entered symbol is 'g') and at Line 15 we have called function **DetermineCharType** which receive a symbol in `Sym`. In the function as we have set some conditions on Line 36-41 and the function returns 1(Line 37) if the symbol is capital, and if the symbol is small it returns 2(Line 39), and if the symbol is neither capital nor small then it returns 3(Line 41). The function returns a number from where it has been called of. In this particular example it has been called from the main (with 'g' passed as parameter hence `Sym` will have 'g'), so after returning, the control is shifted to the main at Line 15 and the returned number (in that particular example 1) will be stored in the variable `Type`. Now as we also have to print the result on the screen, Hence for that on Line15 we have called a function **PrintCharType** and passed a number which we have received from the function **DetermineCharType** as a parameter to the printing function. In this function we have used a switch statement and apply switch cases on Line 26, 28 and 30. Now in a sequence every case will check until one becomes true and program will `cout` (print) the message which is in the switch case block, on the console and the control will again transfer from where it call(main). Now all the control is again shifted to the main at Line 16. From Line 16 to 19 again the same process will proceed as on Lines 12 to 15. And after that the program will terminate.

### Program 3: Check for Multiplicity

In our main flow of the program after taking the inputs from the user we need to not only check for multiplicity for the two entered numbers and but also we need to display the result of the multiplicity (which number is multiple of other). For solving this problem we will first make a function **MultipleOfEachOther** which will take two numbers as input and return an integer(i.e 1, 2 or 3) if the first value is divisible by 2nd (in that case it should return 1) or if the 2nd number is divisible by the 1st value (return 2) otherwise in case of non multiple of the other (it should return 3) as output.

The second most important thing is depending on the value returned by the **MultipleOfEachOther** we must design a function which should display a proper message on the two values whether `v1` is multiple of `v2` or `v2` is multiple of `v1` or non is the multiple of the other. For integrating the two function what we will do is to make actually one function **DisplayMultiplicity** which will perform two tasks together first it will receive a flag from **MultipleOfEachOther** and then depending upon the returned value it will display which number is the multiple of other or non.

```

1  // This program tell the numbers is multiples of each other or not
2  #include <iostream>
3  using namespace std;
4  //This function will get two values and return that the 1st value is multiple of the 2nd or vice versa or none .
5  int MultipleOfEachother(int ,int);
6  //This function calls MultipleOfEachother and displays the message for the multiplicity accordingly.
7  void DisplayMultiplicity(int, int); // This is a prototype.
8  int main()
9  {
10     int value1;
11     int value2;
12     cout << "Enter two numbers: ";
13     cin >> value1 >> value2;
14     DisplayMultiplicity(value1, value2);
15     return 0;
16 }
17 void DisplayMultiplicity(int v1, int v2)
18 {
19     switch(MultipleOfEachother(v1,v2))
20     {
21         case 1:
22             cout << v1 << " is the multiple of " << v2 << " i.e. " << v2 << "*" << v1/v2 << "=" << v1 << endl;
23             break;
24         case 2:
25             cout << v2 << " is the multiple of " << v1 << " i.e. " << v1 << "*" << v2/v1 << "=" << v2 << endl;
26             break;
27         case 3:
28             cout << "Non" << endl;
29             break;
30     }
31 }
32 // returns 1 if v1%v2==0 or returns 2 if v2%v1==0 else returns 3
33 int MultipleOfEachother(int v1,int v2)
34 {
35     if(v1%v2 == 0)
36         return 1;
37     else if(v2%v1 == 0)
38         return 2;
39     else
40         return 3;
41 }

```

### Output

```

Enter two numbers: 24 3
24 is the multiple of 3 i.e. 3*8=24

```

```

Enter two numbers: 17 51
51 is the multiple of 17 i.e. 17*3=17

```

```

Enter two numbers: 15 12
Non is multiple of other...!

```

Link: <http://codepad.org/HiCdsKCO>



## Description

After taking input of two number from the user we have called the function **DisplayMultiplicity** with two entered values (let us say 24 and 3, assigned to v1 and v2 respectively). In the switch statement on line 29 and at the place of switch choice we have called the function **MultipleOfEachOther**. This function will return an integer depending on the values passed (24 and 3 respectively) and corresponding case will be executed (in this example case 1 will be executed and it will output  $3 \times 8 = 24$ ).

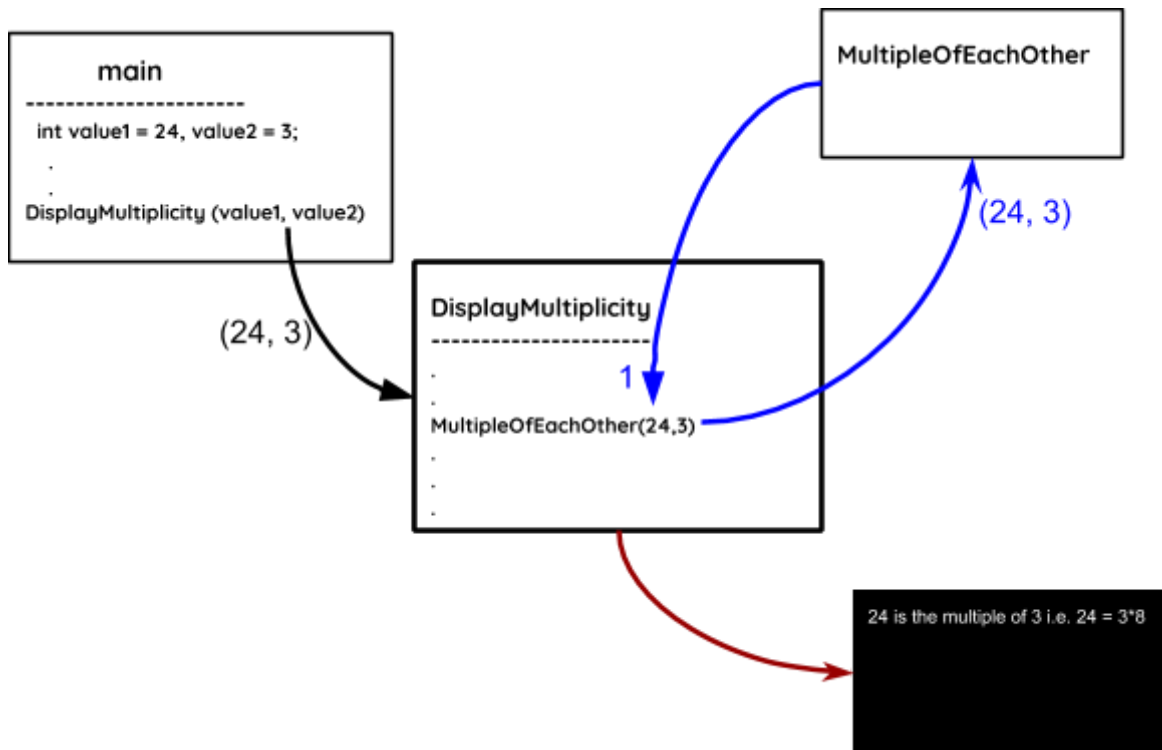


Fig 3: **main** calls **DisplayMultiplicity(24, 3)** and then **DisplayMultiplicity** calls **MultipleOfEachOther(24, 3)** and displays, depending on the value returned , in switch which one is the multiple of other.

## Program 4: Highest Aggregate

As this program requires two major tasks one is to get roll number and 5 marks of courses from the user two times and then after having that numbers we must check which roll number got highest aggregate and then we have to print the result on the screen. It would be a nice idea if we make the part of code doing summation only as a function. As this task, we might be needing again and again. Hence we will make a separate function **UpdateMax** which should take as parameter an integer(of 6 digits). In the following program we will show you how to make that function and then use that function in main to perform the required task.

```

1  // This program shows which roll number got highest aggregate
2  #include <iostream>
3  using namespace std;
4  // This program takes roll number and marks of 5 courses as input 2 times
5  int Sum(int c1, int c2, int c3, int c4, int c5); //prototype
6  bool UpdateMax(int highest_Yet, int c1, int c2, int c3, int c4, int c5); //prototype
7  int main()
8  {
9      int r_no, max_r_no;
10     int c1, c2, c3, c4, c5;
11     int max_total = 0;
12     cout << "1st student Roll# and marks of five courses: " << endl;
13     cin >> r_no >> c1 >> c2 >> c3 >> c4 >> c5;
14     max_total = Sum(c1, c2, c3, c4, c5);
15     max_r_no = r_no;
16     cout << "2nd student Roll# and marks of five courses: " << endl;
17     cin >> r_no >> c1 >> c2 >> c3 >> c4 >> c5;
18     if(UpdateMax(max_total, c1, c2, c3, c4, c5))
19     {
20         max_total = Sum(c1, c2, c3, c4, c5);
21         max_r_no = r_no;
22     }
23     /*
24     Write the code here for the rest of the Roll Nos and their courses marks
25     one by one one....!!!
26     */
27     cout<<max_r_no<<" has the highest marks "<<max_total<< endl;
28     return 0;
29 }
30 int Sum(int c1, int c2, int c3, int c4, int c5) //function
31 {
32     return c1+c2+c3+c4+c5;
33 }
34 bool UpdateMax(int highest_Yet, int c1, int c2, int c3, int c4, int c5) //function
35 {
36     if(highest_Yet<Sum(c1,c2, c3, c4, c5))
37     {
38         return true;
39     }
40     else
41     {
42         return false;
43     }
44 }

```

### Output

```

1st student Roll# and marks of five courses:
1391 80 70 60 14 88
2nd student Roll# and marks of five courses:
1376 70 80 80 88 89
1376 has the highest marks 407

```

### Description

In this code first user enter the 1st student data(roll number and marks of 5 courses)and calculate the sum of the marks using the **Sum** function given on the line 30 and assign it to **max\_total** and first roll number is assigned to the **max\_r\_no** number then user enter second student roll number and it's 5 subjects marks then we calculate it's marks summation and check whose summation is greater by using **UpdateMax** function on line 34 and then according to **UpdateMax** function we set our **max\_r\_no** and the **sum** after that we gave output on the screen.

Link: <http://codepad.org/wuTzx9o9>

### Program 5: Rock-Paper-Scissor using functions

In Rock Paper Scissor we need to determine which player is the winner and we need to display on console screen who is the winner. After taking inputs from the two players (their choices), the very first task we need to perform is figuring out who won the game. And the second important task is to display who is the winner of the game. Essentially we need to perform these two task.

For this purpose we will design two functions **DetermineWinner** and **Display\_Winner**. **DetermineWinner** gets two inputs which are symbol entered by both players and it returns an integer value 1(in case player1 wins) or 2(when player2 wins) or 3(when the game is drawn) or 4(invalid inputs). In **Display\_Winner** function we will pass an integer as input 1(player1 wins) or 2(player2 wins) or 3(game is draw) or 4(invalid inputs) and on the basis of this input we display who is winner of the game. For example if player1 enter 'r' and player2 enter 's' then **DetermineWinner** will return 1 and this value will pass to **DisplayWinner** it will display player1 wins and if both players enter same input then **DetermineWinner** will return 3 and this value will pass to **DisplayWinner** will display game is draw and if any of two player enter an invalid symbol then **DetermineWinner** will return 4 and **DisplayWinner** will display invalid input.

In **int main()** After getting the moves (their chosen symbols) from the users, these symbols will be pass to **DetermineWinner** for finding the result of the game. **DetermineWinner** function will return an integer which we will assign to some variable and that value will be passed to **DisplayWinner** function which will display the message of the Winner of the game to console.

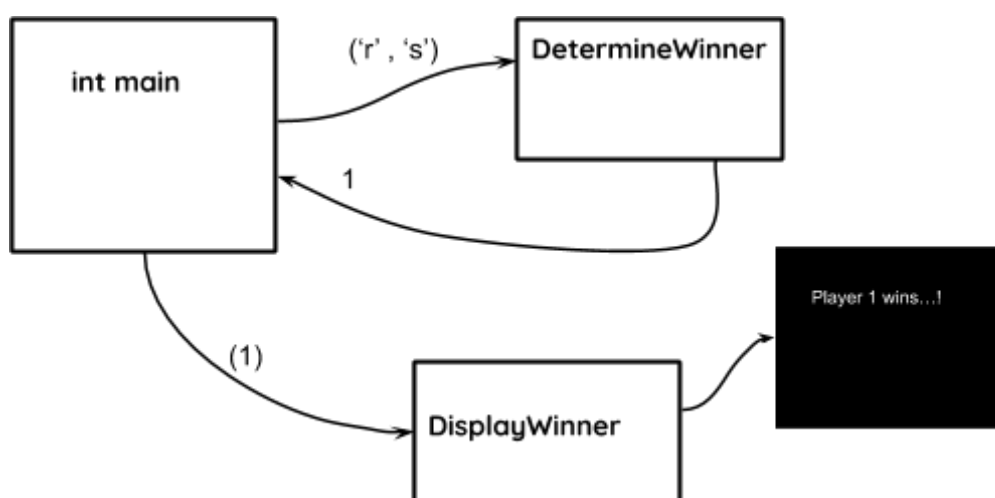


Fig 5: main calls **DetermineWinner** and then calls **DisplayWinner** with an ID (1 or 2) of the user who won the value 3 means the game is draw.

```

1  #include<iostream>
2  #include<conio.h>
3  using namespace std;
4  //this function returns which player is winner or game is tie
5  int DetermineWinner(char player1_symbol, char player2_symbol);
6  //This function Only displays who wins the game, the player id is passed to this function.
7  void DisplayWinner(int who_wins);
8  int main()
9  {
10     int who_wins;
11     char player1_symbol;
12     char player2_symbol;
13     cout << "\tR for rock, P for paper, or S for scissors:" << endl;
14     cout << "\t Player 1 vs Player 2: ";
15     player1_symbol = getch();
16     cout << "\t\t",
17     player2_symbol = getch();
18     cout << "\t\t" << endl;
19     who_wins = DetermineWinner(player1_symbol, player2_symbol);
20     DisplayWinner(who_wins);
21     return 0;
22 }
23 int DetermineWinner(char player1_symbol, char player2_symbol)
24 {
25     if ((player1_symbol == 's' && player2_symbol == 'p') || (player1_symbol == 'r' &&
26         player2_symbol == 's') || (player1_symbol == 'p' && player2_symbol == 'r'))
27     {
28         return 1;
29     }
30     else if ((player1_symbol == 'p' && player2_symbol == 's') || (player1_symbol == 's' &&
31         player2_symbol == 'r') || (player1_symbol == 'r' && player2_symbol == 'p'))
32     {
33         return 2;
34     }
35     else if ((player1_symbol == 'r' && player2_symbol == 'r')
36         || (player1_symbol == 'p' && player2_symbol == 'p')
37         || (player1_symbol == 's' && player2_symbol == 's'))
38     {
39         return 3;
40     }
41     else
42     {
43         return 4;
44     }
45 }
46 void DisplayWinner(int who_wins)
47 {
48     cout<<"\t\t";
49     switch (who_wins)
50     {
51         case 1:
52             cout << "player 1 wins" << endl;
53             break;
54         case 2:
55             cout << "player 2 wins" << endl;
56             break;
57         case 3:
58             cout << "game draws" << endl;
59             break;

```

```
60         case 4:
61             cout << "invalid character input" << endl;
62             break;
63     }
64 }
```

### output

```
R for rock, P for paper, or S for scissors:
Player 1 vs Player 2: **
player 2 wins
```

Link: <http://codepad.org/A0H3n2Ko>

### Description

In `int main()` from line 10 to line 20, first we get input from user, note here we have used a function `getch()` define in library `conio.h` (the difference between `cin` and `getch()` is using `cin` the entered value is also displayed on console and keep displaying until user enters ENTER key, while in `getch()` on user pressing any key the control comes back to the program without displaying anything on console and the function call is replaced by the entered value, here you can see that is why we have displayed a "\*" symbol just to make sure that player 1 should know that his turn is over and the next player should enter his value), used to get input from user. Note that if we had used `cin` here the 2nd player could easily cheat by selecting the character which makes him wins easily. These inputs are passed to **DetermineWinner** which from line 23 to line 45 computes who is the winner and return an integer back to `int main()` at line 19 and saves to an integer `who_wins` and at line 20 this `who_wins` is passed to **DisplayWinner** which is from line 46 to 64 and it uses switch conditions to display who is winner depending on integer passed to it. As this function do not returns any value so after displaying it control is returned back to `int main()` and at line 72 `return 0` is used to close the program.

#### TO REMEMBER

The variable name of the function parameters may or may not be the same as from where the function is called. As whenever the function is called with parameters passed by value the only thing passed to the parameters is the value and then the control of execution is transferred to the function call. Compiler itself manages this abstraction.

#### TO REMEMBER

The Local Variables declared and used within a function call gets destroyed as soon as the control leaves that function and returns to the callee position.

## Annotating our Game to a series of matches

Now we would like to change the above code to such that the two players can play game multiple times like a series of matches. And after the end of the series player of the series should be announced i.e. Who wins maximum number of time. For this purpose we need to make some changes in above code such that we can play again and again. For this purpose we need to learn a new construct **do-while** loop .

## Do-While Loop

The syntax of a **do...while** loop in **C++** is:

Syntax	Example
<pre>do {     statement(s); } while( condition );</pre>	<pre>char choice; do {     cout&lt;&lt; "hello"&lt;&lt;endl;     cin&gt;&gt;choice; } while( choice== 'y' );</pre>

This loop works in the fashion that first the work is done and then usually the condition is checked after doing the work inside the block { } of **do**. If the condition is true then the program part inside the do block will be done again. This work will be repeated until the condition is false. You can see in the example above that first the "hello" message will be printed regardless of checking the condition and then if the user enters 'y' then the message "hello" will be printed again otherwise the loop will break.

---

We can divide this big problem into smaller steps. Firstly we must concentrate on making this game as a series game i.e. both players can play game multiple time like a series until they want. For this we need a choice variable which is used to ask if players want to continue the series or stop the series, and also a **do-while** loop, which operates on this choice variable. Following is the modified **int main ()** of this code.

```

. .
. .
. .
. .
10 int main()
11 {
12     int who_wins;
13     char player1_symbol;
14     char player2_symbol;
15     char choice;
16     do
17     {
18         cout << "\tR for rock, P for paper, or S for scissors:" << endl;
19         cout << "\t Player 1 vs Player 2: ";
20         player1_symbol = getch();
21         cout << " ";
22         player2_symbol = getch();
23         cout << " " << endl;
24         who_wins = DetermineWinner(player1_symbol, player2_symbol);
25         Display_Winner(who_wins);
26         cout << "press (c)ontinue or (s)top \n";
27         cin >> choice;
28     }
29     while(choice=='c');
30     return 0;
31 }
. .
. .

```

Link: <http://codepad.org/h2eH6sYU>

```

(R)ock, (P)aper, or (S)cissors:
**
player 2 wins
press (c)ontinue or (s)top
c
(R)ock, (P)aper, or (S)cissors:
**
player 1 wins
press (c)ontinue or (s)top
c
(R)ock, (P)aper, or (S)cissors:
**
game draws
press (c)ontinue or (s)top
s

```

### Description

In above code we have introduced a new variable **char choice** (on line 15) which is used to ask user if they want to continue series of game or they want to stop it. From line 16 to line 29 we have used a **do-while** loop which operates till user wants to continue the series by entering 'c'. In these lines same code is placed which is used in very first code of this problem. At line 26 and 27 the program asks players whether they want to continue the game or they want to stop the series if the user enters the choice 'c' then the program continues otherwise the program stops and program terminates.

### Some more extension

Now come to next part of this problem which is after players have played a complete series then show who is the winner of series. For this we need to have two variable for counting how many time both

players have won the game. Initially their values are zero but when a player wins a game, we have to increment 1 in the value to that variable. This increment is to be done in switch conditions where our previous code displays the winning player on console screen. Following is the modified `int main` of this problem.

```
1  int main()
2  {
3      int who_wins;
4      char player1_symbol;
5      char player2_symbol;
6      int player1_win_count=0;
7      int player2_win_count=0;
8      char choice;
9
10     do{
11         cout << "Enter R for rock, P for paper, or S for scissors:" << endl;
12         player1_symbol = getch();
13         cout << "****";
14         player2_symbol = getch();
15         cout << "****" << endl;
16         who_wins = DetermineWinner(player1_symbol, player2_symbol);
17         Display_Winner(who_wins);
18         if(who_wins==1)
19             {
20                 player1_win_count++;
21             }
22         if(who_wins==2)
23             {
24                 player2_win_count++;
25             }
26         cout << "press (c)ontinue or (s)top \n";
27         cin >> choice;
28     } while (choice == 'c');
29
30     if (player1_win_count>player2_win_count)
31     {
32         cout<<"Final Result \n";
33         cout << "player one wins" << endl;
34     }
35     else if (player1_win_count<player2_win_count)
36     {
37         cout<<"\nFinal Result \n";
38         cout << "player two wins" << endl;
39     }
40     else
41     {
42         cout << "game is draw..!" << endl;
43     }
44     return 0;
45 }
```



```
Enter R for rock, P for paper, or S for scissors:
**
player two wins
press (c)ontinue or (s)top

C
Enter R for rock, P for paper, or S for scissors:
**
invalid input
press (c)ontinue or (s)top
C
Enter R for rock, P for paper, or S for scissors:
**
player two wins
press (c)ontinue or (s)top
S

Final Result
player two wins

Process returned 0 (0x0)   execution time : 24.699 s
Press any key to continue.
```

Link: <http://codepad.org/Uqk9EaDq>

---

## Program 6: Digit To Word Conversion using Functions (revisited)

This program requires three steps. First it needs that it should prompt from the user the number which user wants to see as word by word. After taking input our program should do splitting of number into its digits. After doing the splitting of the number into its digits then it must has to print those digits to word. If you carefully look into the printing of the digits (it should remind you how painfully you written the code of that switch statements again and again) it is a same code written again and again corresponding to different digits. So it would be a good idea that we make a function which takes as input a digit and prints its english word translation. After making that function we will just call this function for each digit.

Hence for conquering this program we will make two modules one **Number2DigitByDigitWordDisplay** and also **Digit2Word** functions. **Number2DigitByDigitWordDisplay** will split the number into its corresponding digits and call **Digit2Word** for its each digit such that the number is displayed in English.

```
1  #include <iostream>
2  using namespace std;
3  //This Function Will Print The Word corresponding to the digit passed.
4  void PrintWord(int);
5  //This Function is Separating the digits and store them in three different variables.
6  void Number2DigitByDigitWordDisplay(int);
7  int main()
8  {
9      cout<<"Digit to number represent\n"<<endl;
10     int num;
11     cout<<"Enter three digit Number:";
12     cin>>num;
13     Number2DigitByDigitWordDisplay(num);
14     return 0;
15 }
16 void PrintWord(int Digit)
17 {
18     switch(Digit)
19     {
20     case 1:
21         cout<<"One"<<"\t";
22         break;
23     case 2:
24         cout<<"Two"<<"\t";
25         break;
26     case 3:
27         cout<<"Three"<<"\t";
28         break;
29     case 4 :
30         cout<<"Four"<<"\t";
31         break;
32     case 5:
33         cout<<"Five"<<"\t";
34         break;
35     case 6:
36         cout<<"Six"<<"\t";
37         break;
38     case 7:
39         cout<<"Seven"<<"\t";
40         break;
41     case 8:
42         cout<<"Eight"<<"\t";
43         break;
44     case 9:
45         cout<<"Nine"<<"\t";
46         break;
47     case 0:
48         cout<<"Zero"<<"\t";
49         break;
50     }
51 }
52 void Number2DigitByDigitWordDisplay(int num)
53 {
54     if(num<000 && num>999)
55     {
56         cout<<"Error enter three digit number"<<endl;
57         return;
58     }
59     int d1,d2;
```

```
60     d2=num%10;
61     num=num/10;
62     d1=num%10;
63     num=num/10;
64     PrintWord(num);
65     PrintWord(d1);
66     PrintWord(d2);
67 }
```

### Output

```
Digit to number represent
Enter three digit Number:564
Five      Six      Four
```

Link: <http://codepad.org/BGehOokn>

### Description

We needed a function **Number2DigitByDigitWordDisplay** to convert a three digit number into separate digits then that digits will be converted into words. On line 6 we wrote the prototype of this function and there we specified that what is the output and inputs of the function. Then on line 13 this function is called in the **int main()**. In main this function is called after we took a input (three digit integer) from user. It means that the workload of our whole program is on this function. Now on line 52 this function started in this function first we check either the entered number is consist of three digits or not if the entered number is not of three digit then simply the function returns

## Program 7: Calculator

As this program requires three major tasks one is to get numbers from the user and then after having that numbers we must do arithmetic operation on numbers and then we have to print the result on the screen. It would be a nice idea if we make this part of code doing arithmetic operation (+, \*, -, /, %) only as a function. As this task, we might be needing again and again. Hence we will make a separate function **calculation** (on line 16 and its prototype on line 5) which should take parameters an integer(of 2 numbers in this function) and character (one of the value from either of {+, \*, -, /, %} ) and returns the answer. In the following program we will show you how to make that function and then use that function in main to perform the required task.

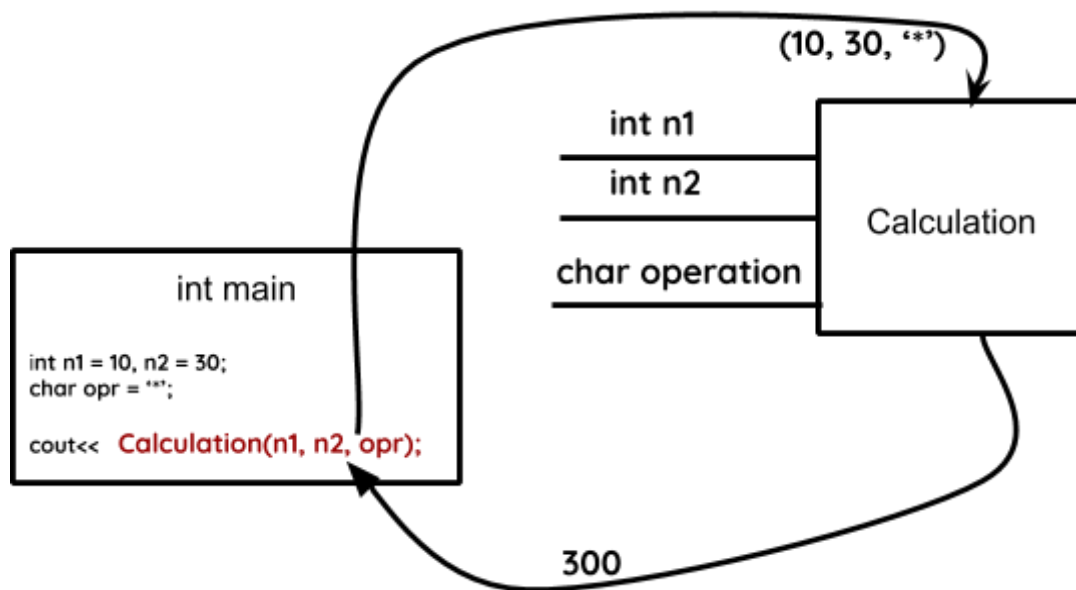


Fig 2.7: The working of the call from main to Calculation function and how it will be replaced by the returned value.

```

1  // This program do arithmetic operation on two numbers
2  #include <iostream>
3  using namespace std;
4  // In this code user enter two number and symbol to perform arithmetic operation
5  int calculation(int num1,int num2,char symbol); //prototype
6  int main()
7  {
8      int n1, n2;
9      cout<< "enter expression e.g. 12+56 : ";
10     // write here the rest of code
11     .
12     .
13     .
14     return 0;
15 }
16 int calculation(int num1,int num2,char symbol) //function
17 {
18     int ans;
19     switch(symbol)
20     {
21     case '+':

```

```

22     ans=num1+num2;
23     return ans;
24     case '-':
25         ans=num1-num2;
26         return ans;
27     case '/':
28         ans=num1/num2;
29         return ans;
30     case '%':
31         ans=num1%num2;
32         return ans;
33     case '*':
34         ans=num1*num2;
35         return ans;
36     }
37 }

```

Link: <http://codepad.org/VGcrqOio>

### Output

```

enter number 1, symbol and number 2: 12%7
12 % 7 = 5

```

```

enter number 1, symbol and number 2: 19*10
19 * 10 = 190

```

### Description

As you see in this code prototype is declared on line 5 and its complete function on line 16 to 37, in this function **switch** statement is used which can perform operation on two numbers according to input symbol(arithmetic) in this **switch** instead of **break**, **return** is used, which return the answer of two numbers. It will automatically break that code when it return something.

## Program 8: Highest Average of a Section

Now as you are provided with the **int main** code given below and as you have to figure out the section which has greater marks. For that you have to write a function for which the prototype is given below.

```

1 char maximum(char sec1,int avg1,char sec2,int avg2,char sec3,int avg3,char sec4,int avg4,char sec5,int
  avg5,char sec6,int avg6);

```

This Function is receiving the **section** names, and its **average marks**. Now you have to write a function which will return the name of the section which has highest average.

```

1  #include <iostream>
2  using namespace std;
3  // This program takes input about sections and averages and tell which section got highest average
4  char maximum(char sec1,int avg1,char sec2,int avg2,char sec3,int avg3,
5               char sec4,int avg4,char sec5,int avg5,char sec6,int avg6);
6  int main()
7  {
8      char max_section;
9      char sec1, sec2, sec3, sec4, sec5, sec6;
10     int avg1, avg2, avg3, avg4, avg5, avg6;
11     cout << "enter sections and their averages: " << endl;
12     cin >> sec1 >> avg1;
13     cin >> sec2 >> avg2;
14     cin >> sec3 >> avg3;
15     cin >> sec4 >> avg4;
16     cin >> sec5 >> avg5;
17     cin >> sec6 >> avg6;
18     max_section = maximum(sec1,avg1,sec2,avg2,sec3,avg3,sec4,avg4,sec5,avg5,sec6,avg6);
19     cout << max_section << " got the highest average." << endl;
20     return 0;
21 }
22 //write the maximum function here
23 char maximum(char sec1,int avg1,char sec2,int avg2,char sec3,int avg3,
24               char sec4,int avg4,char sec5,int avg5,char sec6,int avg6);
25 {
26     .
27     .
28     .
29     .
30     .
31     .
32     .
33     .
34     .
35     .
36     .
37     .
38     .
39     .
40     .
41     .
42     .
43     .
44     .
45     .
46     .
47     .
48     .
49     .
50     .
51     .
52     .
53     .
54     .
55     .
56     .
57     .
58     .
59     .
60     .
61     .
62     .
63     .
64     .
65     .
66     .
67     .
68     .
69     .
70     .
71     .
72     .
73     .
74     .
75     .
76     .
77     .
78     .
79     .
80     .
81     .
82     .
83     .
84     .
85     .
86     .
87     .
88     .
89     .
90     .
91     .
92     .
93     .
94     .
95     .
96     .
97     .
98     .
99     .
100    .

```

**Output**

```

enter sections and their averages:
B 90
D 80
C 60
A 99
E 91
F 80
A got the highest average.

```

**Description**

Link: <http://codepad.org/cXQjLVKo>

**Program 9: Determining Quadrilateral Type**

As we have implemented in the previous chapter that this program requires multiple functionalities to be implemented. Like it first compute the the sides lengths and the diagonals of the quadrilateral. So the first function we need to make is **ComputeDistanceSquare** function which will require 2 points (4 integers) and compute the integer value i.e. square of the distance. Once we have **ComputeDistanceSquare**

function, we need to make another function **DetermineQuadrilateralType** which on given 4 points (8 integers) determine whether the given shape is square (1), rectangle (2), rhombus(3), parallelogram(4), or any other shape. Hence it must return an integer. Note that this **DetermineQuadrilateralType** will use **ComputeDistanceSquare** for finding all the sides as well as the diagonals lengths and then comparing the computed sides and diagonals to determine quadrilateral type. For example if we enter (0,0) (2,0) (2,2) (0,2) the our code will output its a square. See Fig 9 for details.

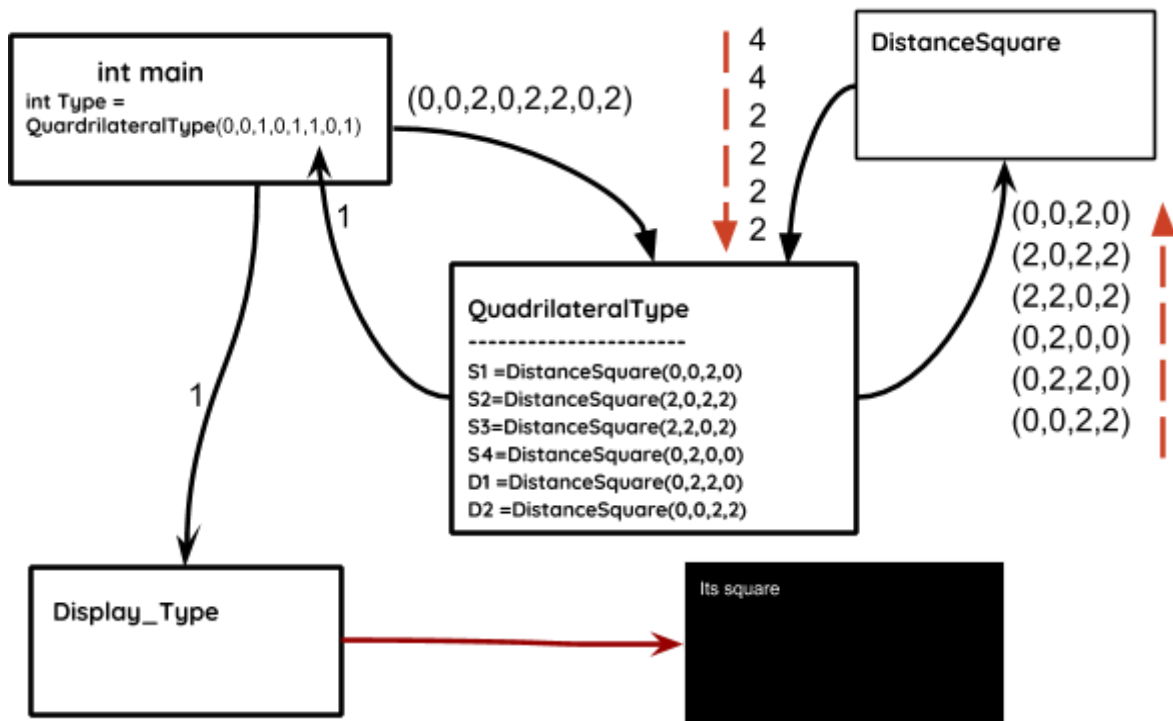


Fig 9: **QuadrilateralType** will be called from main and **DistanceSquare** is called 6 times by **QuadrilateralType** for computing the 4 sides and 2 diagonals of the rectangles.

Now let us see the implementation of the independent functions and their prototypes and how they will be called from main to implement the whole program.

```

1  #include <iostream>
2  using namespace std;
3  int DistanceSquare(int x1,int y1, int x2,int y2)
4  {
5      return (p1x-p2x)*(p1x-p2x)+(p1y-p2y)*(p1y-p2y);
6  }
7  int DetermineQuadrilateralType(int p1x,int p1y, int p2x,int p2y,int p3x,int p3y,int p4x,int p4y)
8  {
9      int s1,s2,s3,s4,dg1,dg2;
10     s1=DistanceSquare(p1x,p1y,p2x,p2y);
11     s2=DistanceSquare(p2x,p2y,p3x,p3y);
12     s3=DistanceSquare(p3x,p3y,p4x,p4y);
13     s4=DistanceSquare(p4x,p4y,p1x,p1y);
14     dg1=DistanceSquare(p1x,p1y,p3x,p3y);
15     dg2=DistanceSquare(p2x,p2y,p4x,p4y);
16     if (s1==s2 && s1==s3 && s1==s4)
17     { // Square or Rhombus
18         if (dg1==dg2)
19         {
20             return 1; // Square case
21         }
22         else
23         {
24             return 2; // Rhombus Case
25         }
26     }
27     else if (s1==s3 && s2==s4 && s1 != s2)
28     { // Rectangle or Parallelogram
29         if (dg1==dg2)
30         {
31             return 3; // Rectangle
32         }
33         else
34         {
35             return 4; // Parallelogram
36         }
37     }
38     else
39     {
40         return 5; // Any other Quadrilateral
41     }
42 }
43 void DisplayQuadrilateralType(int type)
44 {
45     switch(type)
46     {
47     case 1:
48         cout << "Its a square" << endl; break;
49     case 2:
50         cout << "Its a rhombus" << endl; break;
51     case 3:
52         cout << "Its a rectangle" << endl; break;
53     case 4:
54         cout << "Its a parallelogram" << endl; break;
55     case 5:
56         cout << "It's a quadrilateral" << endl; break;
57     }
58 }
59 int main()

```



```

60 {
61     int p1x, p1y, p2x, p2y, p3x, p3y, p4x, p4y;
62     cout << "enter points: " << endl;
63     cout << "P1: ";
64     cin >> p1x >> p1y;
65     cout << "P2: ";
66     cin >> p2x >> p2y;
67     cout << "P3: ";
68     cin >> p3x >> p3y;
69     cout << "P4: ";
70     cin >> p4x >> p4y;
71     int type = DetermineQuadrilateralType(p1x, p1y, p2x, p2y, p3x, p3y, p4x, p4y);
72     DisplayQuadrilateralType(type);
73     return 0;
74 }

```

### Output

```

enter points:
P1: 0 0
P2: 2 0
P3: 2 2
P4: 0 2
Its a square

```

Link: <http://codepad.org/SMVvhEek>

### Description

In above code from line 63-71 we get inputs from user and at line 72 we pass these coordinates to **DetermineQuadrilateralType** for example (0,0,2,0,2,2,0,2). The first thing this functions computes is all sides lengths and the diagonals of the given shape. Lines 11 to 16 are doing that specific task. Note that this length computations require exactly the same computation again and again with different points as inputs to **DistanceSquare** function. You can notice that instead of computing the exact length which requires a square root computation we have computed the distance square quantity. As we are only checking the equality of the sides and diagonals in hence even the square of the length property will also hold true. Hence line 17-43 computes that given shape lies in which possible case and acts accordingly by returning the identity of the shape. In this particular example it will **return 1** to main then this value **1** is passed to **DisplayQuadrilateralType** which will display on console screen that **"its a square"**.

### Program 10: Identifying the triangle

For this problem we need to calculate distance between points 3 times and we need to determine which shape of the triangle is given by the user, after that we need to display type of triangle is provided by the user.

So we need three functions **DistanceSquare** which gets coordinates of two points and it returns distance square between these two points. Similarly we need **DetermineTriangleType** which get coordinates of points and it will determine which triangle these points will form and also we need a **DisplayTriangleType** which display on console which type of triangle is formed.

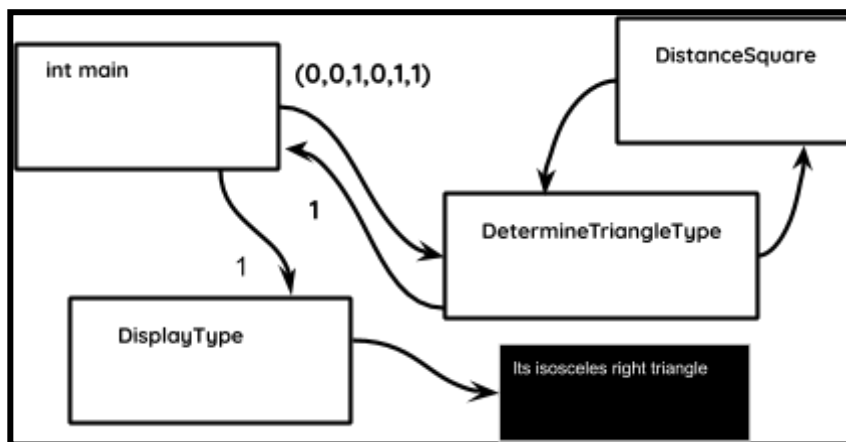


Fig 10: **main** will first call **DetermineTriangleType** and then call **DisplayType** for proper message to display on console.

```

1  int CalculateDistance(int p1x,int p1y,int p2x,int p2y)
2  {
3      .....
4  }
5  int DetermineTriangleType (int p1x,int p1y,int p2x,int p2y,int p3x,int p3y)
6  {
7      .....
8  }
9  void DisplayTriangleType(int type)
10 {
11     .....
12 }
13 int main()
14 {
15     int p1x, p1y, p2x, p2y, p3x, p3y;
16
17     cout << "enter points: " << endl;
18     cout << "P1: ";
19     cin >> p1x >> p1y;
20     cout << "P2: ";
21     cin >> p2x >> p2y;
22     cout << "P3: ";
23     cin >> p3x >> p3y;
24     int type= DetermineTriangleType(p1x, p1y, p2x, p2y, p3x, p3y);
25     DisplayTriangleType(type);
26     return 0;
27 }

```

Link

<http://codepad.org/n4ojH340>

```

P1: 0 0
P2: 1 0
P3: 1 1
Triangle is Isosceles Right Triangle.

```

**Description**

In this code from line 16 to 24 we get input from user and at line 25 we call **DetermineTriangleType** for example we enter (0,0) (1,0) (1,1) then this function will return a value which we get in **int** type then this integer is passed to **DisplayTriangleType** which will display which type of triangle it is.

**Program 11: Point Inside/Outside/On The Rectangle**

In this problem we need to calculate distance between points many times. Determine whether the point lies in rectangle or outside or on border of rectangle.

So we need three functions, **calculate\_Distance** which will get coordinates of two points i.e. 4 inputs and it will return an integer value which is distance between these points. Similarly we need **DeterminePointLocation** which will get coordinates of 4 points i.e. 8 inputs and will returns an integer value depending upon whether point is inside, outside or on border of rectangle, then this integer value is passed to **DisplayPointLocation** which displays location of point depending on integer value passed to it.

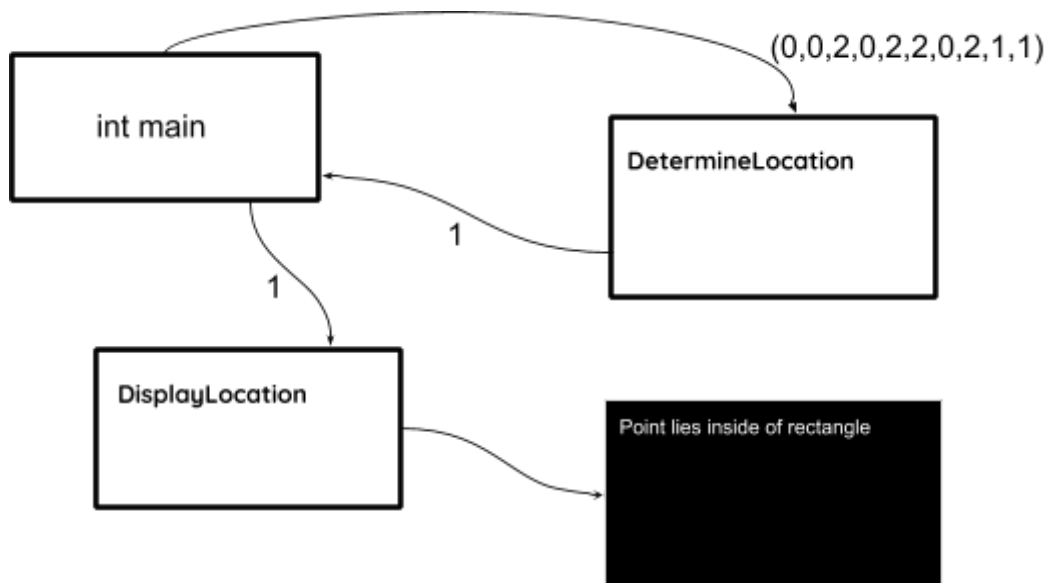


Fig 11: The flow and the way function will be called from main. The complete details of the inputs with its datatypes are deliberately left as an exercise to the user.

```

1  //This program determines the point lies within the square or not
2  int DeterminePointLocation (int p1x,int p1y,int p2x,int p2y,int p3x,int p3y,int p4x,int p4y,int px,int py)
3  {
4      if(((px<p1x) || (px<p2x) || (px>p3x) || (px>p4x)) &&
5          ((py<p1y) || (py<p2y) || (py>p3y) || (py>p4y)))
6      {
7          return 1;
8      }
9      if(((py>p1y) || (py<p2y) || (py<p3y) || (py>p4y)) &&
10         ((px>p1x) || (px>p2x) || (px<p3x) || (px<p4y)))
11     {
12         return 2;
13     }
14     else
15     {
16         return 3;
17     }
18 }
19 void DisplayPointLocation(int type)
20 {
21     if(type==1)
22     {
23         cout<<"point lies inside rectangle"<<endl;
24     }
25     else if(type==2)
26     {
27         cout<<"point lies outside rectangle"<<endl;
28     }
29     else if(type==3)
30     {
31         cout<<"point lies on the rectangle"<<endl;
32     }
33 }
34 int main()
35 {
36     char choice;
37     int p1x,p1y,p2x,p2y,p3x,p3y,p4x,p4y,px,py;
38     cout << "enter points:" << endl;
39     cout << "P1: ";
40     cin >> p1x >> p1y;
41     cout << "P2: ";
42     cin >> p2x >> p2y;
43     cout << "P3: ";
44     cin >> p3x >> p3y;
45     cout << "P4: ";
46     cin >> p4x >> p4y;
47     cout << "P: ";
48     cin >> px >> py;
49     int type= DeterminePointLocation (p1x,p1y,p2x,p2y,p3x,p3y,p4x,p4y,px,py);
50     DisplayPointLocation(type);
51     return 0;
52 }

```

```
enter points:
P1: 0 0
P2: 2 0
P3: 2 2
P4: 0 2
P: 1 1
P lies inside.
```

Link: <http://codepad.org/qVtk1OIU>

### Description

In above code from line 37 to 49 we get inputs from user and at line 50 we call **DeterminePointLocation** and pass all points to it for example (0,0) (2,0) (2,2) (0,2) and (1,1) then these points are passed to **DeterminePointLocation** then it will check whether point(1,1) lies inside(1) the shape formed by other points or outside(2) or in the border of shape(3) and then returns this value to **main**. After that we pass this value to **DisplayPointLocation** which will display the location of point on console screen.

## Program 12: Finding the Middle of the Three Numbers

This problem requires three major task one is to take numbers from the user and then find the second maximum or middle of the three numbers and the last task is to print the result on the screen/console.

Now as we have to find the middle number among three numbers(i.e **N1, N2, N3**). For that we will be needing two types of comparisons, greater than and also less than. This is because for a number **N1** to be the middle candidate **N1** needs to be either less than **N2** and greater than **N3** or greater than **N2** and lesser than **N3**. The similar procedure needs to be done for candidacy of **N2** to be the middle value and also **N3** should be checked for the candidacy of middle value. This should make you immediately realize that greater than and lesser than functionality must be made and also the middle candidacy check functions should be made.

Hence we will make **IsGreater** function which will take two numbers as input and tell (in **true/false**) whether the first number is greater than second. Similarly we also need **IsSmaller** function. After that we will make the function **IsMiddle** which will take 3 parameters i.e. first number to be the candidate of middle value and the two more values from which it needs to compare the its candidacy for being middle by comparing one to be greater and one to be smaller (here we will be using **IsGreater** and **IsSmaller** function) .

We will now see the implementation of these functions and its use to find the middle value out of the 3 inputs taken from user.

```
1 //This program tell us the middle number between three numbers.
2 #include<iostream>
3 using namespace std;
4 // function will take two number as input and return true if num1 is greater, and return false if num2 is greater
5 bool IsGreater(int num1,int num2);
6 // function will take two number as input and return true if num1 is smaller, and return false if num2 is greater
7 bool IsSmaller(int num1,int num2);
8 // returns true if the first number is middle element
9 bool IsMiddle(int Candidate, int num1, int num2);
10 int main()
11 {
12     int num1, num2, num3;
13     cout << "enter 3 numbers: ";
14     cin >> num1 >> num2 >> num3;
15
16     if (IsMiddle(num1, num2, num3))
17     {
18         cout << num1;
19     }
20     else if (IsMiddle(num2, num1, num3))
21     {
22         cout << num2;
23     }
24     else if (IsMiddle(num3, num1, num2))
25     {
26         cout << num3;
27     }
28     else
29     {
30         cout<< "Non...";
31     }
32     cout<< "is the middle number";
33     return 0;
34 }
35 bool IsGreater(int num1,int num2)
36 {
37     return num1>num2;
38 }
39 bool IsSmaller(int num1,int num2)
40 {
41     return num1<num2;
42 }
43 bool IsMiddle(int Candidate, int num1, int num2)
44 {
45     if ((IsGreater(Candidate,num1) && IsSmaller(Candidate,num2))
46         || (IsGreater(Candidate,num2)&& IsSmaller(Candidate,num1)))
47     {
48         return true;
49     }
50     return false;
51 }
```

**Output**

```
enter 3 numbers: 90 3 60
60 is the middle number
```

```
enter 3 numbers: 60 90 30
60 is the middle number
```

```
enter 3 numbers: 30 90 40
40 is the middle number
```

```
enter 3 numbers: 30 30 30
Non... is the middle number
```

Link: <http://codepad.org/EPPqvESh>

**Description**

In this code after taking three input from user we have called a function **IsMiddle** on line 16 to check whether **N1** is middle number or not, for that we have also pass these three numbers as a parameter to that function in the order of (**N1**, **N2**, **N3**). Now in function **IsMiddle** we have further called two function **IsGreater** and **IsSmaller** in one condition on the lines 45 and 46. And this function **IsMiddle** return/**false** depending on the condition. In Function **IsGreater** we have just passed two parameter **candidate(N1)** and the **num1(N2)** which is returning on line 37 true if the candidate is greater than the **num1**, else it return **false**. In function **IsSmaller** we have also passed two parameter **candidate(N1)** and the **num2(N3)** which is returning **true** on line 41 if candidate is smaller than **num2**. As if both the function **IsGreater** and **IsSmaller** are return **true** then the function **IsMiddle** will also return **true** on line 48, **else** if any of the two function return **false** than the **IsMiddle** function also return **false** on line 50. And control is again shifted to the main function. Now if the function return true the line 18 will execute and the **N1** will be displayed inside that if block of { }.

Otherwise the line 20 will be executed and function **IsMiddle** is called again with the candidacy of **N2** as the middle value checking and exactly the similar procedure will be checked for this case - if it **return true** than it displays the number **N2** inside **else if** block { } otherwise if it return **false** than line 24 will be executed with candidacy of **N3** as being middle. And if it **return true** then the **N3** is displayed on the screen and if it return false than line 28 will be executed and display message "Non" (because that will be only possible way if there isn't any middle value that is either two or three values are equal). After all the conditions line 32 will execute and display a message on the screen and the program will terminate.

**Program 13: Ceiling of the real value**

In this problem we want ceil number from floating number which is a complete process based on some conditions and computations. So we need a function which will accept a float type number and returns an integer type number.

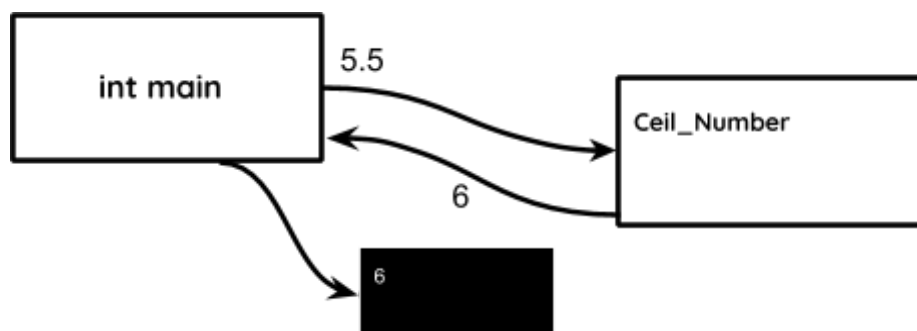


Fig 13: main calls with a floating value for example 5.5 to the function **CeilNumber** and it returns the corresponding ceiled value and that value gets output on console.

```

1 // This program display ceiling integer from floating number
2 #include <iostream>
3 using namespace std;
4 // This program convert floating number to ceiling number
5 int CeilNumber(float num); //prototype
6 int main()
7 {
8     float num;
9     cout << "Enter Number: ";
10    cin >> num;
11    int result = _____; //write the rest of code
12    // Display the message here
13    return 0;
14 }
15 int CeilNumber(float num) //function
16 {
17     int ceil=num;
18     if (num>0 && ceil!=num)
19     {
20         return ceil+1;
21     }
22     else
23     {
24         return ceil;
25     }
26 }

```

**Output**

```

Enter Number: 5.5
6

```

```

Enter Number: -5.5
-5

```

```

Enter Number: 5
5

```

Link: <http://codepad.org/mZzjLf6X>

**Description**

In this code at line 10 we get a float value from user and at line 11 we call **Ceil\_Number** with correct arguments (which is your task) and then we have to display that result on console screen. This function will make an integer type variable **ceil** at line 17 and assign given number **num** to this variable and check if given number **num** is greater than zero and both integer type variable **ceil** and **float** type number **num** are not equal then it will add one in integer type number **ceil**. your task is to call this function in **int main()** with correct arguments.



### Program 14: Floor of the real value

In this problem we want floor number which is a complete process based on some conditions and computations. So we need a function which will accept a float type number and returns an integer type number.

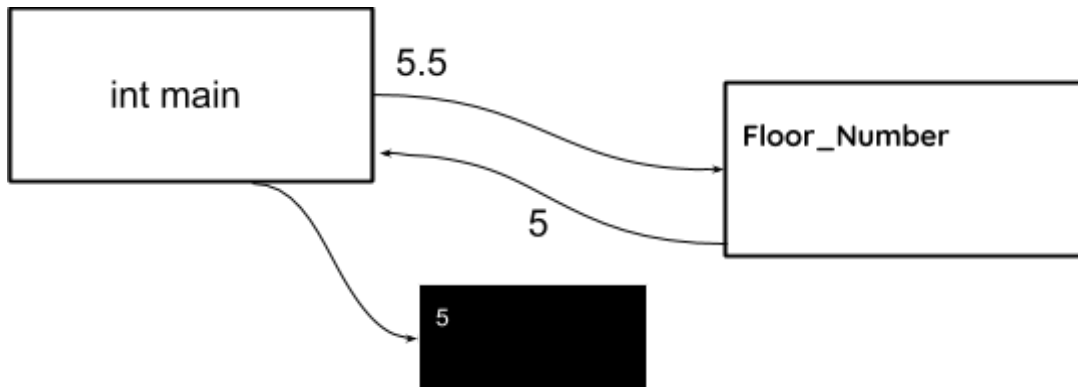


Fig 14: **main** calls with a floating value for example 5.5 to the function **FloorNumber** and it returns the corresponding ceiled value and that value gets output on console.

```

1  // This program display flooring integer from floating number
2  #include<iostream>
3  using namespace std;
4  // This program convert floating number to flooring number
5  int Floor_Number(float num);//prototype
6  int main()
7  {
8      float f;
9      cout<<"Enter Number";
10     cin>>f;
11     int result = _____;
12     // Display the message here
13 }
14 int Floor_Number(float num)//function
15 {
16     int floor=num;
17     if (num<0 && floor!=num)
18     {
19         return floor-1;
20     }
21     else
22     {
23         return floor;
24     }
25 }
  
```

#### output

```

Enter Number: 5.5
5
  
```

```
Enter Number: -5.5
-6
```

```
Enter Number: 5
5
```

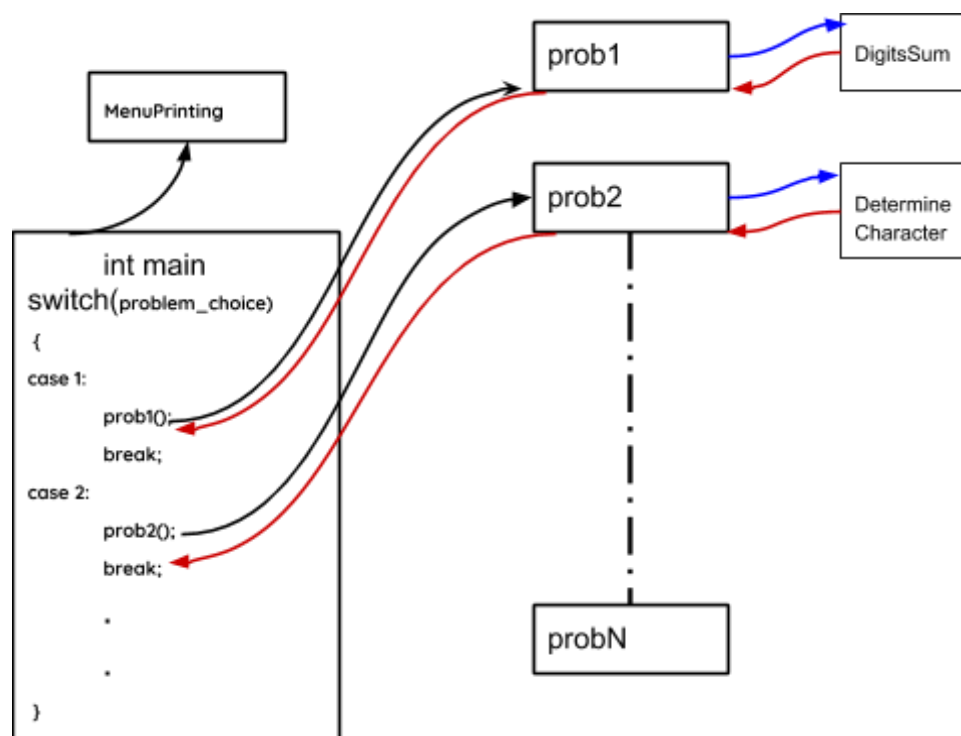
**Link:** <http://codepad.org/hUmAfjAt>

### Description

In this code at first we have take input float value from user and at line 11 we called a function **Floor\_Number** with correct arguments (which is your task) and then we have to display that result on console screen. This function will make an integer type variable **floor** at line 16 and assign given number **num** to this variable and check if given number **num** is greater than zero and both integer type variable **floor** and **float** type number **num** are not equal then it will subtract one in integer type number **floor**. your task is to call this function in **int main()** with correct arguments.

# Menu Base program

When we do so many programs in one program it is known as a Menu Based Program. As you can see in the following program we are handling two programs simultaneously by user's choice. If user enters 1 it will execute first program which is Digit Submission and if he enters 2 it will execute second program which is Determine Character Type.



```
1  #include <iostream>
2  #include <windows.h>
3  #include <conio.h>
4  using namespace std;
5  int DigitsSum(int number)
6  {
7      int sum=0;
8      sum = sum + number % 10;
9      number = number / 10;
10     sum = sum + number % 10;
11     number = number / 10;
12     sum = sum + number % 10;
13     number = number / 10;
14     sum = sum + number % 10;
15     number = number / 10;
16     sum = sum + number % 10;
17     number = number / 10;
18     sum = sum + number % 10;
19     return sum;
20 }
21 int prob1()
22 {
23     int number;
24     int sum = 0;
25     cout << "enter 6 digit number: ";
26     cin >> number;
27     sum=DigitsSum(number);
28     cout << "sum of 6 digit number is: " << sum << endl;
29 }
30 int DetermineCharacterType(char character )
31 {
32     if(character >= 'A' && character <= 'Z')
33     {
34         return 1;
35     }
36     else if(character >= 'a' && character <= 'z')
37     {
38         return 2;
39     }
40     else
41     {
42         return 3;
43     }
44 }
45 int prob2()
46 {
47     char character;
48     cout << "enter character: ";
49     cin >> character;
50     switch(DetermineCharacterType(character))
51     {
52         case 1:
53             cout << "Capital letter" << endl;
54             break;
55         case 2:
56             cout << "Small letter" << endl;
57             break;
58         case 3:
59             cout << "Non" << endl;
```

```

60         break;
61     }
62     return 0;
63 }
64 void MenuPrinting()
65 {
66     cout<< "p1: 6 Digit sum\n"
67         << "p2: Determine Character Type\n";
68 }
69 int main()
70 {
71     char choice;
72     do
73     {
74         system("cls");
75         int problem_choice;
76         MenuPrinting();
77         cin >> problem_choice;
78         switch(problem_choice)
79         {
80             case 1:
81                 prob1();
82                 break;
83             case 2:
84                 prob2();
85                 break;
86         }
87         cout << "\n\n do u want to continue: (Y/y)";
88         cin >> choice;
89     }
90     while (choice=='y' || choice=='Y');
91     return 0;
92 }

```

### Description

In this code we pick up the above problems code and paste it above this code our menu based main and convert all the other problems name to prob 1/prob 2/prob 3 e.t.c. after that we make a function which will print the menu of our problems. Then we enter the problem choice from the user that which problem user want to execute. After getting user there is a switch statement used to call the main function of the corresponding problem as in case 1 prob1 is called and as we have changed the name of main of the other problems as prob1 e.t.c. so it will execute the main of the prob1 after that user enter whether after running the corresponding problem he want to continue or to terminate the program. In this way we have made our menu base.

## Age Calculator

We want to make a special calculator that can calculate the age of any person. As we know that for calculating any persons age we must have one's date of birth and also we must know the current date. We must have to be careful about the validity of the two dates. After validating the two dates we need to subtract the two dates in a very particular fashion we will discuss them in a while. Let us us first discuss how to validate the dates.

### Date Validations Checks

- The first thing that we need to watch out is that if both dates are correct or not. Like if days are greater than 0 and less than 28, 29, 30 or 31(depending which month is the date) and the month is not greater than 12 or less than 1 then it means that the date is correct.  
E.g 15-04-1999 (**Correct**)  
32-13-2001 (**Wrong**)
- The second thing that we need to watch out is that the second date(Current date) must be greater than the date of birth because we will subtract date of birth from current date. If current date is smaller than then the subtraction isn't possible.


Note: While coding we need to make sure that next step which is calculating age, will only be triggered when the above entered dates are valid. In case if the entered values are invalid then our program must take the input again until the valid input is entered.

### Calculating Age

Now after validating both dates now we would like to calculate the age of the user. Lets us suppose that we have a date 28-09-2000 as a Date of Birth and 13-02-2017 as a Current Date. As you must have learnt in your childhood Mathematics that we start subtraction from least significant digit (which is last digit) and in case the subtraction of a digits yields a -ve value then we take the carry from its immediate higher significant digit. E.g. if we would like to do the following subtraction 871 - 798:

$$\begin{array}{r}
 871 \\
 -798 \\
 \hline
 \end{array}$$

Subtraction Order



7	7	6
<del>8</del> 71	<del>8</del> 61	<del>8</del> <del>7</del> 1
-798	-798	-798
<hr/> 073	<hr/> -33	<hr/> -7
	+10	+10=
	=	3
	7	

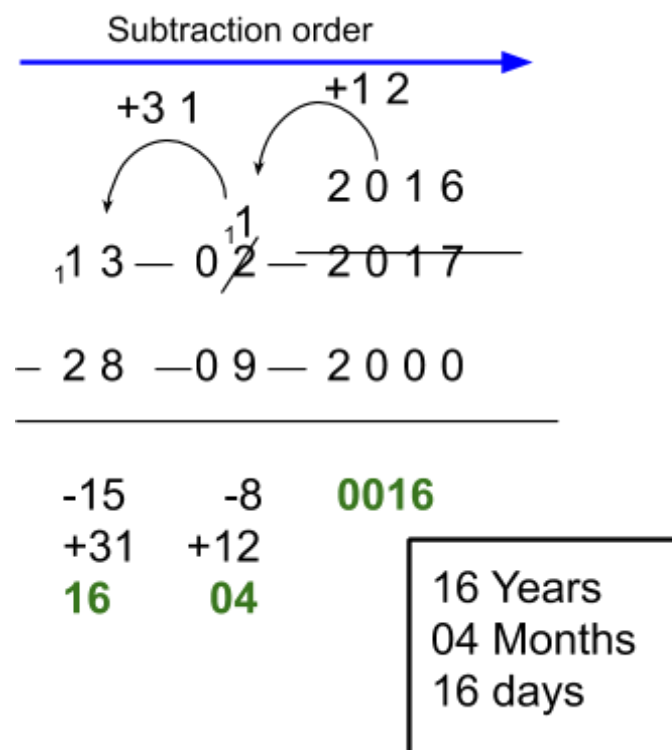
But in dates our least significant digits are Days and Most significant digits are Years. So we will start subtraction from days then we will move to months and then years.

13-02-2017

28-09-2000

As we can see we can not subtract days because 13 is less than 28 so we will take carry from months. As previous month is January and it has 31 days so we will add 31 in 13 which will be 44 now our days will be (44-28=16). Same procedure with months that we can not subtract 9 from 2 so will take carry from Year. It means we will add 12 in 2 and it will be 14. Now our months will be (13-9=4) and now the years can be subtracted easily(2016-2000=16)

So the Age is 16-years 4 Months and 16-Days



While writing the code we need to remember these things that if we want a carry from previous month and that month is January then we will add 31 in days (because January has 31 days), if it is April then we will add 30 (because April has 30 days).

Code of age calculator is given below.

**C++, pasted just now:**

```

1  #include <iostream>
2  using namespace std;
3  //This Function is Simply Checking Dates.
4  bool validate(int day, int month, int year);
5  bool IsLeapYear(int year);
6  int DaysInAMonth(int month, int year);
7  //This Function is Comparing Both Dates And Validating them.
8  bool ValidateCurrentDate(int B_Day, int B_Month, int B_Year, int C_Day, int C_Month, int C_Year);
9  //This Function is Calculating Age.
10 void agefinder(int B_Day, int B_Month, int B_Year, int C_Day, int C_Month, int C_Year);
11 int main()
12 {
13     int dob, mob, yob; //date of birth, month of birth, year of birth
14     int cd, cm, cy;    //current date, current month, current year
15     do
16     {
17         cout << "enter date of birth(day month year): ";
18         cin >> dob >> mob >> yob;
19     }
20     while (validate(dob, mob, yob) == false);
21     do
22     {
23         cout << "enter current date(day month year):: ";
24         cin >> cd >> cm >> cy;
25     }
26     while (validate(cd, cm, cy) == false || ValidateCurrentDate(dob, mob, yob, cd, cm, cy) == false);
27     agefinder(dob, mob, yob, cd, cm, cy);
28     return 0;
29 }
30 bool validate(int day, int month, int year)
31 {
32     if (day <= 0 || day>DaysInAMonth(month, year))
33         return false;
34     else
35         return true;
36 }
37 bool ValidateCurrentDate(int B_Day, int B_Month, int B_Year, int C_Day, int C_Month, int
38 C_Year)//Birth_day,Birth_Month,Birth_Year,Current_Day,Current_Month,Current_Year
39 {
40     if (C_Year>B_Year)
41         return true;
42     if (C_Year<B_Year)
43         return false;
44     if (C_Month>B_Month)
45         return true;
46     if (C_Month<B_Month)
47         return false;
48     if (C_Day >= B_Day)
49         return true;
50     else
51         return false;
52 }

```

```

53 int DaysInAMonth(int month, int year)
54 {
55     int days=0;
56     if (month == 4 || month == 6 || month == 9 || month == 11)
57     {
58         days += 30;
59     }
60     else if (month == 1 || month == 3 || month == 5 || month == 7
61             || month == 8 || month == 10 || month == 12)
62     {
63         days += 31;
64     }
65     else if (month == 2 && year % 4 == 0)
66     {
67         days += 29;
68     }
69     else days += 28;
70     return days;
71 }
72
73 void agefinder(int B_Day, int B_Month, int B_Year, int C_Day, int C_Month, int C_Year)
74 {
75     int days = C_Day - B_Day;
76     if (days<0)
77     {
78         C_Month--;
79         days += DaysInAMonth(C_Month, C_Year);
80         if (days < 0)    // This could be possible in case of 1-31 and carry is 28/29
81         {
82             C_Month--;
83             days += DaysInAMonth(C_Month, C_Year);
84         }
85     }
86     int month = C_Month - B_Month;
87     if (month<0)
88     {
89         C_Year--;
90         month += 12;
91     }
92     int year = C_Year - B_Year;
93     cout << "Age is: " << year << " years " << month << " months " << days << " days " << endl;
94 }

```

### Output

```

enter date of birth(day month year): 24 5 1997
enter current date(day month year): 26 8 2017
your age is: 20 years 3 months 2 days

```

### Description

The first thing that we need to make sure is that if both dates are correct or not. E.g the months must be greater than 0 and less than 12 etc. The **validate** function is doing the same task. We are passing



three parameters (**day, month and year**) and checking that if days are less than 0 or greater than days in a particular month than it must return false otherwise return true.

In **validateCurrentDate** function we are comparing both dates. We are passing SIX parameters to this function(**B\_Day, B\_Month, B\_Year, C\_Day, C\_Month, C\_Year**). If **C\_Year** is greater than **B\_Year**, it will return true. But if **C\_Year** is smaller than it will return false because the solution will not be possible. If both years are same then we will check the months. **C\_Month** should be greater than **B\_Month**. If **C\_Month** is smaller, the function will return false again. Now if months are same then we will move to the days. Now If **C\_Day** are smaller than **B\_Day** then the solution will not be possible because in this situation the current date is greater than the date of birth.

After validating both dates we will move to the last step which is Age Calculation. This is what we are doing in **agefinder** function. This function is taking SIX parameters (**B\_Day, B\_Month, B\_Year, C\_Day, C\_Month, C\_Year**). First we will subtract days(**C\_Day - B\_Day**), if this subtraction gives a negative answer (less than zero) then we will take a carry from **C\_Month** which will add 28, 29, 30 or 31 (Depending upon which month it is). NOTE - the carry of the number of days must be from the previous month as it unlike the decimal number representation a month is not telling that the current month has been passed, rather its the previous month which has been passed. Now when we are subtraction months(**C\_Month - B\_Month**) and it also gives a negative answer, To avoid this we will take a carry again from **C\_Year**. It will add 12 in Months and at last we will subtract Years(**C\_Year - B\_Year**).

Link: <http://codepad.org/LN5fRIIt>

## Practice Exercise

1. **AGE CALCULATOR:** Take today's date (particularly the day of the week). Calculate what was the day on 14th August 1947 (On the day Pakistan came into being).
2. **AGE CALCULATOR:** We know that there are 365 days (in a normal year other than 366 days for Leap year case). What if we convert a date into days by multiplying the number of passed months to its days and for calculating the age we subtract the converted days and calculate the age in years and months and days again? Now solve the Age Calculator again using this idea.
3. 2520 is the smallest number that can be divided by each of the numbers from 1 to 10 without any remainder. What is the smallest positive number that is evenly divisible by all of the numbers from 1 to 20? Generalize the program to such that if you are given a range  $R_1$  and  $R_2$  find the smallest number which is divisible by all the numbers between  $R_1$  and  $R_2$ .

4. The sum of the squares of the first ten natural numbers is,

$$1^2 + 2^2 + \dots + 10^2 = 385$$

*Which function should be designed to perform this task? Design and implement that function?*

The square of the sum of the first ten natural numbers is,

$$(1 + 2 + \dots + 10)^2 = 55^2 = 3025$$

*Which function should be designed to perform this task? Design and implement that function?*

Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is  $3025 - 385 = 2640$ . *Which function should be designed to perform this task? Design and implement that function?*

Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum. ***Make main function to implement the following task.***

5. A Pythagorean triplet is a set of three natural numbers,  $a < b < c$ , for which,

$$a^2 + b^2 = c^2$$

For example,  $3^2 + 4^2 = 9 + 16 = 25 = 5^2$ .

There exists exactly one Pythagorean triplet for which  $a + b + c = 1000$ .

Find the product  $abc$ .

6. The sum of the primes below 10 is  $2 + 3 + 5 + 7 = 17$ . Find the sum of all the primes below two million.