SESSION 1-4: FUNCTIONS

1-4.1 Pre-Defined Functions

Compiler developers or writers usually identify certain programs that are likely to be written by those who end up using their compilers. To make it easier for their compiler users, these small programs are written by the developers and stored under different names. Each of these small programs are referred to as functions. These functions are then built into the C++ compiler. An in-built function is therefore a pre-defined function included in the C++ compiler by the developers, that is, a function which has already been written by developers and can be used by a programmer in his program without having to write a program or a code for it. Note that programmers can also write their own functions and hence the reason why we have to qualify functions written by the compiler developers as pre-defined function and the one's written by programmers as user-defined functions. Please note that compiler developers are also programmers but we want to use the term programmers for people who use compilers, etc to write their computer programs. C++ has over 150 inbuilt functions in the C++ library as such inbuilt functions are placed under different groups. Whenever a particular function is to be used in a program, the compiler must be told to read a special header file that contains definitions of the functions we want to use. This is done by using the #include directive as shown earlier. For now, we shall look at the mathematical, char (character) and string functions. Let us consider the function pow(), a function that returns a value representing the result of a number raised to a certain power. For example if we want to find x raised to the power y and store the result in z, mathematically, we will write this as $z = x^y$. In C++ we will write the same expression as $z = pow(x, y)$. The x and y are called arguments. Whenever a function is referred to in a program, we call that a functional call or function invocation, that is a function is being called to return a value based on the argument(s) given or passed. Thus, $z = pow(2,5)$ illustrates a function call to the pow function to return 2 raised to the power 5, that is 32 to be stored in z. Note that an argument can be a constant, expression or a variable. If it is a variable then it should have a value before it is passed to the function. Similarly, for an expression, it must evaluate to a value.

In each of the following tables, if a function is given as function(x) or function(x, y), it means the function requires one or two arguments respectively as input. The argument type specifies the data type of the input or the argument. For example, to use the abs() function, the argument should be of the type int and the value returned will also be of the type *int*.

MATHEMATICAL AND TRIGONOMETRIC FUNCTIONS

Most of the mathematical functions have their definitions included in the math.h header file. As such, to use any of the following mathematical and trigonometric functions one has to use the

include directive to include the header files stdlib.h and the math.h. In the following list, except for the rand, random, atoi, atoll, atof, abs and the fabs that use the stdlib.h header file, all the functions require that the header file math.h be included in the current program. The following are some of the main mathematical or trigonometric functions used in C++.

function	**meaning**	**argument type**	**value returned**	**library**
abs(x)	absolute value	int	int	cstdlib
acos(x)	cos inverse	double	double	cmath
asin(x)	sine inverse	double	double	cmath
atan(x)	tan inverse	double	double	cmath
atan2(x,y)	arc tan of x/y	double	double	cmath
atof(x)	char to double	char	int	cstdlib
atoi(x)	char to int	char	int	cstdlib
atoll(x)	char to long int	char	int	cstblib
ceil(x)	ceiling	double	double	cmath
cos(x)	cosine	double	double	cmath
cosh(x)	hyperbolic cos	double	double	cmath
exp(x)	exponentiation	double	double	cmath
fabs(x)	absolute value	double	double	cmath
floor(x)	floor	double	double	cmath
fmod(x,y)	remainder of x/y	double	double	cmath
labs(x)	absolute value	long	long	cstdlib
log(x)	logarithm	double	double	cmath
log10(x)	log to base 10	double	double	cmath
pow(x,y)	power of x to y	double	double	cmath
rand()	random no.	none	int < 32767	cstdlib
random(x)	random no.	int	int	cstdlib
sin(x)	sine	double	double	cmath
sinh(x)	hyperbolic sine	double	double	cmath
sqrt(x)	square root	double	double	cmath
srand(x)	reinitialises random number generator	unsigned int	void	cstdlib
tan(x)	tangent	double	double	cmath
tanh(x)	hyperbolic tan	double	double	cmath

CHAR FUNCTIONS

These are functions meant for characters. Most of the character functions return an integer value as result. The integer result is either 1 or 0 for true or false respectively. To use any of the following character functions, you must use the include directive to include the ctype.h header file. The following are the commonly used inbuilt character functions. They are called character functions because they accept as argument a single character.

Function	used to test if x is / convert x to	Value returned
isalum(x)	is a digit or alphabet	int
isalpha(x)	An alphabet	int
iscntrl(x)	is a control character	int
isdigit(x)	a digit	int
ispunct(x)	a punctuation	int
isspace(x)	a space	int
islower(x)	In lower case	int
isupper(x)	In upper case	int
tolower(x)	To lower case	char
toupper(x)	To upper case	char

STRING FUNCTIONS

The string functions are used on identifiers or constants that contain at least a character but the identifier should have been declared to be a string data type. Strings will be treated when we discuss arrays. To use a string function in a program, you must use the include directive to include the header file string.h. The following are the various string functions.

function	Description
strcat (x,y)	adds the string y to the end of x
strcmp(x,y)	compares the strings x and y
strcpy(x,y)	copies the string y to x so that x and y contains the same string
strlen(x)	returns the number of characters in x
strncat(x,y,n)	same as strcat except that the n is used to specify the number of characters to be appended from y.

strncmp(x,y,n)	same as strcmp except that here n is used to specify the number of characters to be compared.
strncpy(x,y,n)	same as strcpy except that the n here is used to specify the number of characters to be copied from y to x. hence x and y may not be the same after the copy operation if y has more characters than the value of n.

Other Useful functions

Function	Meaning	Argument type
rand()	Returns a random number	None
srand(x)	Returns a seeded random number	Double
time(x)	Returns time in seconds since midnight	double

Please note that whenever a rand() function is executed, it returns a random number, a number that one cannot predict. The number is always between 0 and 32767 inclusive. Using the rand function alone will always generate the same random number. Please note that the srand() function does not return a value. To ensure that the rand() function returns different random numbers you must be changing the seed value of tha srand function. Usually, the time function is called with the value 0, that is time(0), to return the single integer time of the day in seconds by reading the Computer's internal clock time. Thus, the clock function generates different numbers whenever executed. It can generate the same number if it is executed on different days and at exactly the same time.

1-4.2 User Defined Functions

A user defined function is simply a function usually written by a programmer (and in this case not the developers of the C++ compiler). The function is given a name such that it can be referenced a number of times in a main program or other functions by its name without having to code it several times. Meaningful names should be used for user defined function. Note that when a user defined function shares the same name as an inbuilt function, most compilers will not flag this as an error or give even a warning message hence you should be familiar with names that have already been used for inbuilt functions so that you do not use the same names even though you can do so. I will advise you not to use inbuilt function names as user defined function names. When a pre-defined function name is used for a user defined function, it's the definition of the user defined function that is used whenever the function is called. Functions written by a programmer and not part of the C++ predefined functions are called programmer or user defined functions. There are different types of functions that programmers can write. The general format is as follows:

resultType functionName(parameterList);

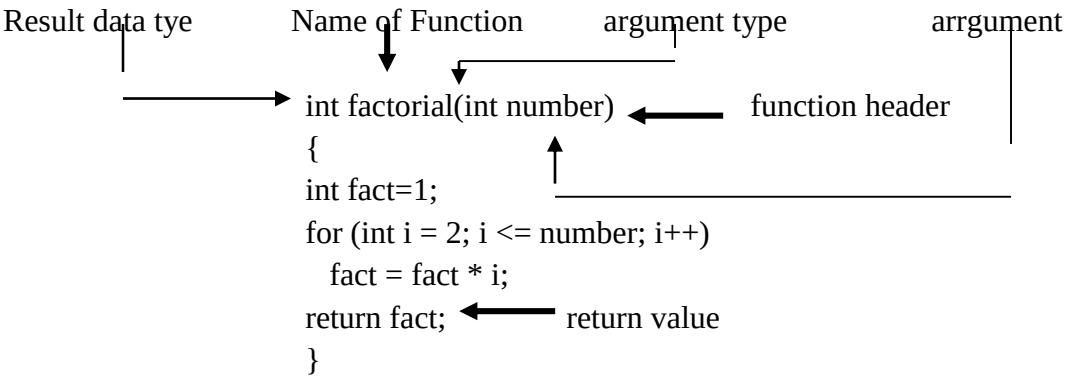
```
{
    functionalStatements;
    return value;
}
```

where *resultType* is the data type of the result to be retuned to the function that made the call. The *functionName* is any name that the programmer wishes to give to the function, however, the rules for forming identifiers apply to the names that a programmer assigns to functions. More importantly, the name must be meaningful so that the task or purpose of the function can be known without looking at the *functionalStatements*. The *parameterList* is the list of parameters that are going to be passed to the function when called. They are called formal parameters. The list can be parameters and/or arguments. The general form of the *parameterList* should be as follows:

```
dataType1 parameter1, dataType2 parameter2,…dataTypen parametern;
```

The *functionalStatements* are the various statements or instructions required by the function in order to obtain the result being expected from it. The value specifies the results to be returned by the function. Note that a function can have as many return statements as may be required and may appear anywhere in the function definition or the body of the function. Basically, a return statement is used to exit a function.

Let us consider the following function



The first line in the above function is usually referred to as function header. The name of the above function is factorial. It is expected to return only integer values whenever called. Any function calling it must pass on an integer value (number). The factorial of the number passed to the function is calculated and the return statement is used to specify the value (fact) to be returned.

USING A USER DEFINED FUNCTION IN A PROGRAM

User defined functions can be included in a program or be written to a separate file. For now we shall consider including a user defined function in a program. To do so, a function prototype and a function definition are required. The *function prototype* simply describes how the function is to be called. In addition, it tells the programmer everything he needs to know in order to write a call to the function, that is, the number of arguments that function needs, the data type of the arguments and the type of result to be returned. The function prototype should have the same syntax or format as the function header. However, the names of the variables used need not necessarily be the same but their data types must be the same. In a function prototype, one is allowed to omit the names of the identifier and state just their data types but in this course we will try to indicate the indentifiers in the function prototypes. One other difference between a function prototype and a function header is that the function header does not require a semi colon at the end whereas a terminating semicolon is required for a function prototype.

A function prototype should come before the start of the main function or main part of the program, that is before the 'int main()' statement in the program. The function prototype has two different formats as illustrated by the following:

```
resultType functionName(dataType1 parameter1, dataType2 paameter2,…..);
```

or

```
resultType functionName(dataType1,dataType2,….);
```

The *resultType* is the type of result to be returned, the *functionName* is the name of the function. The *dataType1*, *dataType2*, etc are the data types of the formal parameters. The *argument1*, *argument2*, etc are the formal parameters. In the first format the formal parameter list are provided while in the second format they are not, but we shall limit ourselves to the first format so that when necessary we can write comments on the various formal parameters. The following ar valid function prototypes:

```
int factorial (int);
int factorial(int n);
void Swap(char, char);
double sum(double y, double x);
bool CarOwner(int cars);
```

Please note that in a function header, one can equally omit the names of the identifiers, however, there will be a problem in writing the function definition and hence you must always name your identifiers in your function headers.

PLACEMENT OF FUNCTION DEFINITION

Generally, the C++ compiler will accept placing the function definition and function prototype in certain locations such as at the beginning or at the end of the main program. If a function definition

is placed before the main part of a program, then the function prototype is not needed. If however the function definition is placed at the end of the main program then both function prototype and the function definition are required. Different programmers prefer different styles but most C++ programmers prefer to use both function prototype and the function definition. As such, for this course we will adopt the style of placing all function prototypes at the start of the main part of the program and the function definitions at the end of the main program. Thus, programs will be written to have the following general structure:

```
include directives
function prototypes
int main()
{
  Statements and function calls
  return 0;
}
function definitions
```

CALLING A FUNCTION

A user defined function is called in the same way as most inbuilt functions are called. These functions can be called by the main function or other user defined functions. When arguments are given to a function to return a value, we use the term 'pass', that is arguments are passed to functions. When a function is called, it uses the arguments passed to it as inputs. The values passed are then plugged in for the formal parameters. If the arguments are variables then the value or data stored in the variables, and not the variables themselves are plugged in. The first argument is plugged in for the first formal parameter, the second argument for the second formal parameter and so forth. The function then works with the values plugged into the formal parameters by substituting all occurrences of the formal arguments with their respective passed values. Note that when a function is called with wrong order of the arguments, the results returned may not be correct and in some cases the compiler might display a wrong data type message or warning.
The syntax or format for calling a function is as follows:

```
ReceivingField = functionName(argumentList);
```

where *ReceivingField* is the name of the variable to store the result to be returned by the function being called, if any. The *ReceivingField* is omitted if a function does not return a value. The functionName specifies the name of the function being called. The *argumentList* is the list of arguments or constants to be passed on to the function. Here, one does not need to specify the data type for each argument but the data type must correspond to the data type specified in the prototype and the function header.

TYPES OF USER DEFINED FUNCTIONS

Generally, user defined functions can be classified into two types namely those that accept arguments and those that accept parameters as input. The two types of functions are similar in the way they are defined and called. The main difference between the two types is that for a function that uses arguments, it cannot modify the value of the arguments passed to it as it works with only copies of the values passed. However, for functions that receive parameters, they can change the value of the parameters they receive.

FUNCTIONS THAT ACCEPT ARGUMENTS

So far, we have discussed functions that accept arguments. Functions that accept arguments cannot modify the content of the arguments passed to it even if such variables are declared as global variables. Such variables can be modified within the function but it does not affect the value of the variable outside the function. They can only modify variables declared within the function. Functions called this way are said to have **call-by-value** formal parameters which means a local copy of each argument is made in the function to be treated just like any other local variables.
Consider the following program

```
#include<iostream>
using namespace std;
int swap(int x, int y) ;   ←  function prototype
int main()
{
 int x,y,m;
 cout<< 'Please Enter two integers\n';
 cin>> x >> y;
 cout<< '\nThe value of x  before function call is ' << x;
 m = swap(x,y);
 cout<<'The value of x after function call is ' << x;
cout<<'The value returned by function is ' << z;
return 0;
}

int swap (int x, int y)        ←  function header
{
  int z;
    z = y;
    y = x;                     function definition
    x = z;
 return z;
}
```

In the above program, the function **swap** by its definition is to interchange the value stored in x and y, and then return the value stored in z which is actually the new value of x. However, when the above program is run, the output to the screen will be as follows:

```
Please Enter two integers
20 40
The value of x before function call is 20
The value of x after function call is 20
The value returned by function is 40
```

As can be seen from the results, one would have expected x and z to have the same value looking at the statement just before the return statement of the swap function, that is x = z. However, the value of x after the function call is the same as before the function call. This means that storing the value of y in x within the function did not change the value stored in x before the function was called. If we had displayed the new value of x within the function, then 40 would have been displayed.

FUNCTIONS THAT ACCEPT PARAMETERS

Functions that accept parameters are those that can modify the values passed to them. They have the same general structure as those that accept arguments. The basic difference in the two types of functions is that for functions that accept parameters, the parameters should have the ampersand symbol (&) to the end of its data type as shown in the program below. Note that a function can be defined to accept both arguments and parameters but can only modify the parameters if necessary. Calling a function with parameters is usually referred to as **call-by-reference**. When such a function is called, the corresponding variable argument (and not its value) will be substituted for the formal parameter. Any change made to the formal parameter in the body of the function will also be made to the argument variable when the function is called. Let us consider the swap function again and this time using parameters instead of arguments.

```
#include<iostream>
using namespace std;
int swap(int& x, int& y); // x and y are defined as arguments
int main()
{
 int x,y,m;
 cout<< 'Please Enter two integers\n';
 cin>> x >> y;
 cout<< '\nThe value of x before function call is ' << x;
 m = swap(x,y);
 cout<<'The value of x after function call is ' << x;
```

```
cout<<'The value returned by function is ' << z;
return 0;
}
int swap (int& x, int& y)
{
  int z;
   z = y;
   y = x;
   x = z;
 return z;
}
```

When the above program is run, the output to the screen will be as follows:

```
Please Enter two integers
20 40
The value of x before function call is 20
The value of x after function call is 40
The value returned by function is 40
```

It can be seen from the above output that by using the ampersand sign (&), the function was able to change the value stored in x before the function was called.

USER DEFINED FUNCTIONS AND DEFAULT ARGUMENTS

It is possible to give a formal parameter a default argument. The essence of this is to save time writing the default value at each invocation of a function. If a function uses a number of parameters, then when one parameter is given a default value all parameters following the one that has been given a default value must also be given default values. Generally, only trailing parameters of a function can have default values. Let us consider the following function headers:

(i) void myFunction(int a, int b=6, int d);
(ii) void myFunction(int a, int b=6, int d=5);
(iii) void myFunction(int a=1, b=2, int c=88);
(iv) void myFunction(int a=1, int b, int e);

In the above examples, (ii) and (iii) are legal.
Example (i) is illegal as one of the trailing parameters, d has no default value.
Example (iv) is also illegal as trailing parameters b and e have no default values.
Consider the following program:

```
#include<iostream>
using namespace std;
void ItsMe(int a, int b=10);

int main()
```

```
{
        ItsMe(10);
        return 0;
}

void ItsMe(int x, int y)
{
        cout <<(x+y);
}
```

The output of the above program is 20. Since only one parameter is provided (see bolded line), the 10 is assigned to x when ItsMe is called. The y assumes its default value of 10. If the bolded line is replaced with **ItsMe(10,19)** then the output will be 29, as such a default value is used only when a value is not specified for it when the function is called. Here, y will have the value 19 instead of the default value 10. Note that even though one could assign default variables in a function header, it is better to assign the default value in the function prototype rather than the function hearder.

FUNCTIONS THAT DO NOT RETURN A VALUE

We have so far seen functions that return a value when called. There is a third type of function that does not return any value. To tell the compiler that a function will not be returning a value when called, the data type of the result to be returned by the function is indicated as void. Thus, any function in a program that will not return a value should have void as the first word in the function prototype and the function heading. The way in which void functions are defined are similar to the way functions that return a value are defined. The void functions are best for menus, accepting input, displaying output and so forth. Since a void function does not return a value there should be no return statement in the function definition, or if a return statement is used then there should be no expression or value after it. To call a function that does not return a value the syntax is as follows:

```
functionName(arguments);
```

where *functionName* is the name of the function. The *arguments* are the parameters to be passed to the function separated by commas.

Let us consider the following program that uses void functions.

```
#include<iostream>
using namespace std;
void getInput(int& num);
void DisplayFactors(int number);

int main()
```

```
{
  int x;
  void getInput(x);
   void DisplayFactors(x);
return 0;
}
void getInput(int& num)
{
  cout <<  'Please enter an integer';
  cin >> num;
  return;
}


void DisplayFactors(int number)
{
  int count;
  for(count = 0, count <= number / 2; count++)
    if (number% count == 0)
          cout << count  <<'  ';
  return;
}
```

The above program uses two void functions. The first void function, getInput is used to accept a number from the keyboard. Note that this function uses a parameter hence can modify the value of the variable that is passed to it. The second void function, DisplayFactors, determines the factors of the argument passed to it and then display the factors as they are determined.

Even though the return statement at the end of a void function definition is not very important, the return statement can play very important role in a void function. Since we have already seen the factorial function, let us consider it as an example to explain some importance of the return statement in a void function. We will code our example as follows:

```
void factorial(int n)
{
 if(n<0)
  {
   cout <<"Illegal call to factorial function....";
   return;
  }
 else
 {
  int fact=1;
  for(int i=2;i<=n;i++)
```

```
    fact*=i;
  }
}
```

In the above example, if we omit the return statement, it can cause our program to halt or result in a wrong output. The if statement tests for the value of n against zero. If the value is less than 0, then the factorial cannot be calculated and the return statement forces the function to be exited after displaying a message indicating that the call to the factorial function is illegal.

OVERLOADING FUNCTION NAMES

Overloading a function name means given two or more definitions to a function name. This means it is possible to reuse names that have good intuitive appeal in a variety of situations. Overloading simply means having two or more functions with the same name in the same program. If a function name is overloaded then the function definitions must have different number of formal parameters or some formal parameters of different data types. When such a function is called, the compiler uses the definition whose number of formal parameters, data types of the formal parameters and the data type of the result match the arguments in the function call.

Let us consider the following function prototypes:

```
int myPowerFunction (int, int);
int myPowerFunction (double, double);
int myPowerFunction (double);
```

The above shows three different functions even though they share a common name or one can say the function myPowerFunction is overloaded with three different function argument lists. During program execution only one of the three functions will be executed at a time when a call is made to the function and that will depend on the number and the data type of the parameters passed to the function. For example, if only one argument of data type real is passed then only the third function definition will be used. However, if two arguments are passed, then depending on whether they are integers or real, either the first or the second function definition will be used. The first function definition is used only when two integer arguments are passed while the second will be used only when two real arguments are passed. Note that the first and the second functions only differ in the data type of the arguments. The first and second differ from the third by the number of arguments.

Function overloading is also called polymorphism. The word *poly* simply means many and the word *morph* means form, thus polymorphism means many forms. Function polymorphism refers to the ability to "overload" a function with more than one function definition by simply changing

the number or type of the parameters. The following program demonstrates function polymorphism.

Please note the following about function overloading:

- There is no requirement that when functions are overloaded they should perform the same or similar task. Thus, it is allowed for two or more functions to have the same name and performing completely different functions. Even though we are allowed to do this, it is not a good programming practise.
- Function overloading should be done such that there will be no ambiguity when one of the functions is to be called. Using the same function overloading with no difference in either the result data type, the number of arguments or the data type of the arguments can result can create ambiguity.
- Please note that if the difference between overloaded functions lie in the type of function call, that is call-by-value and call-by-reference, it can also creat ambiguity and hould be avoided.
- If two overloaded functions are written to differ only in terms of the data type of the result to be returned, it will result in a compilation error. Consider the following functions:

```
int Average(int, int);
double Average(int, int);
```

If a function call such as *cout<<Average (10, 5)* is made, there is going to be ambiguity as the compiler will not know which of the function definitions to use. Usually, the compiler prevents such errors from occurring by indicating during compilation time that there is an error.

LOCAL AND GLOBAL VARIABLES

Depending on where a variable is declared, it may or may not be used elsewhere in the same program. The places where a variable can be used is said to be the scope of the variable. By scope, there are two different types of variables namely *local* and gloIbal variables. Local variables are variables declared within the body of a user defined function or the main function. They are local only to the function or main function within which they are declared, that is the compiler will display an error message indicating that a particular variable or identifier is undefined if the variable is used in another function without being declared in that function. Global variables are those variables that can be used by either the main function and/or user-define functions. They are usually declared before the *int main()* of the main program. Global variables are best used when

more than one function is to use them. Variables declared within the body of the main function (that is after init main()) are local variables to the main function.

Note that the same variable name can be declared in the main function as a global variable and within a function as a local. When this happens, the two variables will be treated as being different by the compiler. In functions that do not have the same variable being declared within, references to such variables will assume their current global values. However, those declared within a function will assume their local values rather than their global values hence care should be taken when a global variable is re-declared in a function. If possible, this should be avoided.

It must be noted that the value of a global variable can be modified by any function within the main function. Since global variables make it difficult to understand large program, it is advisable not to use them if possible.
Consider the following program

```
#include<iostream>
using namespace std;
function_prototypes;
int num1, num2;
int main()
{
  program_statements;
  return 0;
 }
function_definitions;
```

In the above pseudo program code, the variables num1 and num2 are global variables and they can be used in all the functions including the main function.

1-4.3 Recursive Functions

Functions have the ability to call themselves if necessary. This is called recursion which can be either direct or indirect. A direct recursion means a function calls itself within the body of the function while for an indirect recursion, a function calls another function which in turn calls the function that called it. Thus, if a function A calls a function B and the function B calls the function A, then the function A has an indirect recursive call. Recursion may appear a bit complex to most people but there are times where a recursive function is much preferred to a non recursive function. For example to solve the problem of Tower of Hanoi, it is best to use a recursive function rather than a non recursive one. We will soon solve the problem of the Tower of Hanoi as an example of recursion.

What happens when a function calls itself is that, a new copy of the function is made and run. One important thing to note about recursion is that local variables in the new copy of the function are

independent of the previous or the original function. We will now consider some problems using recursion.

Example 1: Write a function to generate Fibonacci series given the first two terms. You are required to make use of recursion or recursive function. For example, if the first two terms of a series is 0 and 1, the next (3^{rd}) term is obtained by adding the two preceding terms (1^{st} and 2^{nd}), etc. Thus the third term of the series will be 1 and the fourth will be 2, etc. Generally, the Fibonacci series is defined as follows:

$$f_i = f_{i-2} - f_{i-1} \text{ for i=3, 4, ..., n}$$

Solution

This program assumes that the first two terms are 1 and 1.

```
#include <iostream>
using namespace std;
int fib(int n);
int main()
{

     int term, number;
     cout << "Enter term [>2] of number to find: ";
     cin >> term;
     cout << "\n\n";
     number = fib(term);
    cout <<"The required number is " <<number <<endl;
     return 0;
}// end of the main function
   int fib (int n) // start of the fibo function
   {
    if  (n < 3 )
          return (1);
     else
      {
          cout << "Calling fib(" << n-1 << ") and fib(" << n-2 <<" ).\n\n";
          return( fib(n-1) + fib(n-2));
       }
} // end of the fibo function
```

When you run the above program and enter the value 5 say, as the term required, the main function calls the fibo function and passes to it the number 5. When fibo starts executing, fibo(5) causes the function to call itself as it is supposed to return fibo(4) and fibo(3). At this stage fibo(4) and fibo(3) do not have values hence fibo(4) is first evaluated by calling for fibo(3) and fib(2). Fibo(3) calls for fibo(2) and fibo(1). Since for fibo(1) and fibo(2), the value of n is less than 3, the return value is 1 for each (see line 4 of fibo). The values of fibo(1) and fibo(2) are then substituted for fibo(3)

call (fibo(3)=1+1=2). Since fibo(2) and fibo(3) are also known at this stage they are substituted for fibo(4) call (fibo(4)=2+1=3). Finally, fibo(3) and fibo(4) are substituted for fibo(5) (fibo(5)=3+2) and value 5 is displayed as the fifth term. The following output will be displayed to the screen when the program is run with the value 5.

Program run:

```
Enter term [>2] of number to find: 5
Calling fib(3) and fib(4).
Calling fib(2) and fib(3).
Calling fib(1) and fib(2).
Calling fib(1) and fib(2).

The required number is 5
Press any key to continue
```

Question: The above program assumes the first and the second terms of the Fibonacci series to be one (1). Rewrite or modify the above program so that it can allow a user to enter the first two terms as input and to display the required term.

Program listing:

```
#include <iostream>
#include <cstdlib>
using namespace std;
int fib(int n);
int one,two;
int main()
{
     int term, number;
     cout<<"Please enter the first two terms of the series ";
     cin>>one >>two;
     cout << "Enter term [>2] of number to find: ";
     cin >> term;
     cout << "\n\n";
     number = fib(term);
    cout <<"The required number is " <<number <<endl;
     system("PAUSE");
     return 0;
}// end of the main function

   int fib (int n) // start of the fibo function
   {
    if  (n ==1 )
```

```
        return (one);
    else if (n==2)
      return(two);
     else
      {
         cout << "Calling fib(" << n-1 << ") and fib(" << n-2 << ").\n\n";
         return( fib(n-1) + fib(n-2));
       }
  } // end of the fibo function
```

Program run

```
Please enter the first two terms of the series 2 3
Enter term [>2] of number to find: 5


Calling fib(3) and fib(4).
Calling fib(2) and fib(3).
Calling fib(1) and fib(2).
Calling fib(1) and fib(2).
The required number is 13
Press any key to continue . . .
```

You can check whether or not the above is correct by listing all the intermediate steps up to the fifth term and that is

2 3 5 8 13

Example 2:

Write a program making use of recursion to find the factorial of a given positive integer.

Solution

```
#include<iostream>
using namespace std;
int factorial(int n);
int main()
{
        unsigned int n;
        cin>>n;
        int m=factorial(n);
        cout <<m <<endl;
        return 0;
```

```
}

int factorial(int n)
{
        int fact=1;
        if (n==0)
                return(1);
        else
                fact= n*factorial(n-1);
                return fact;
}
```

Question

Write a recursive C++ program that can accept two positive integers as inputs. The first integer being a number to be converted to a different base and the second being the base required. Your program should accept only bases between 2 and 16 inclusive. Your output should be the first integer in base of the second integer. Hint: you are not to use array of numbers in your program.

Program listing

```
#include<iostream>
using namespace std;
void bases(int number, int base);
int main()
{
 int base,n;
 do
 {
  cout<<"Please enter a VALID number to convert ";
  cin>>n;
 } while(n<0);

 do
 {
   cout<<"\t Enter the base required [2-16] inclusive: ";
   cin>>base;
 } while (base<2||base>16);
 cout<<"The number " <<n <<" in base " <<base <<" is ";
 bases(n,base);
 cout <<endl;
 system("pause");
 return 0;
}
void bases(int n, int b)
```

```
{
    char digits[]={"0123456789ABCDEF"};
    if(n>b)
       bases(n/b,b);
    cout<<digits[n%b];
}
```

Program run

```
Please enter a VALID number to convert 2748
      Enter the base required [2-16] inclusive: 16
The number 2748 in base 16 is ABC
Press any key to continue . . .
```

1-4.4 Inline Functions

Whenever a function has a very short definition, the definition of the function can be specified in the function prototype and one will not have to define the function again at the end of the main function. We will limit our inline functions to functions that are about three lines long though strictly speaking there is no limit to the number of lines that an inline function can have in its definition. Inline functions are identified by the word 'inline' and they have the general format as follows:

```
inline returnDataType functionName(arguments)
{
  Function definition
}
```

where returnDataType is the type of result to be returned by the inline function. Void should be used if it is expected not to return any value. The functionName is the name of the inline function, that is the name by which the inline function will be called. The arguments are the parameters to be passed to the inline function when called. The function definition is the statements that form the body of the inline function. One advantage of inline functions is that they run much faster even though they take up more storage. They take up more storage because with an inline function, each function call in a program is replaced by a compiled version of the function definition. Note that for ordinary user defined functions, a compiled version is not used to replace each function call.

Question: Newton's method is a one way of finding the location of the zero or the root of a mathematical function, f(x) say. The method works by initially guessing the x location of the zero. The function value at the current x is computed to get a better estimate of the zero. This process is repeated until the difference between two successive function values is at most 0.0000001. Each estimate of xnew is computed from xold as follows:

$$x_{new} = x_{old} - \frac{f(x_{old})}{f'(x_{old})}$$

Where $f'(x_{old})$ is the derivative of the function $f(x_{old})$. Given that $f(x) = 5x^2 + 4x - 72$ write a C++ program that will accept as input an initial guess value, x_{old} and then find the zero of the above function. You are required to use inline functions. Your output should be the root of the function and the number of iterations performed to obtain the root. We shall revisit this question again this time without using inline functions.

Program listing

```
#include<iostream>
#include<cstdlib>
#include<cmath>
using namespace std;
inline double f(double x) {return(2.0*pow(x,5)+4*x-72);}
inline double fdev(double x) {return(10.0*pow(x,4)+4);}
int main()
{
        int cnt=0;
        double xnew,xold;
        cout <<"Please enter initial guess ";
        cin>>xold;
        xnew=xold;

        do
        {       xold=xnew;
                xnew=xold -f(xold)/fdev(xold);
                cnt++;
        } while(fabs(xold-xnew)>pow(10,-7));
        cout<<"The zero of the function is " <<xnew <<endl;
        cout <<"The number of iterations is " <<cnt <<endl;
    system("pause");
        return 0;
}
```

Question: One method of finding the integral of a given function is the rectangular method with altitutes chosen at the midpoints of the subinterval. The given function is apparently divided into smaller rectangles, the area of each rectangle is then calculated and finally the sum of all the small areas multiplied by Δx. For example, given the function $f(x) = x^3 + 2$ to be integrated over the interval 0 and 2, we can write this as follows:

$area = \sum_{x=0}^{2} f(x)\Delta x$ for x=x+ Δx for the subintervals. First, you are to calculate the Δx which is given by $\Delta x = (b-a)/n$ where *n* is the subintervals and a and b are the lower and upper limits of the interval end points.

Program listing:

```
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;
inline double f(double x) {return (pow(x,3)+2);}
int main()
{
 double lower, upper, n;
 double x,deltaX, area=0;
 cout <<"Please enter the lower and upper limits of function ";
 cin >>lower >>upper;
 cout <<"Please enter the number of subintervals ";
 cin>>n;
 deltaX=(upper-lower)/n;
 x=lower+deltaX/2;
 for(int i=0;i<n;i++)
 {
   area+=f(x);
   x+=deltaX;
  }
 area*=deltaX;
 cout <<"\nThe approximate value of the integral is " <<area;
 cout <<endl;
 system("PAUSE");
 return 0;
}
```

Program run 1:

```
Please enter the lower and upper limits of function 0 3
Please enter the number of subintervals 10
The approximate value of the integral is 26.1487
Press any key to continue . . .
```

Program run 2:

```
Please enter the lower and upper limits of function 0 3
Please enter the number of subintervals 100
```

```
The approximate value of the integral is 26.249
Press any key to continue . . .
```

Note that with the above function and inputs, the exact value should have been 26.25. However, because we are using approximation method to get the integral value of the function, it is expected that there will be some small difference in the actual and the approximated value. Note that in most cases the larger the number of the subintervals the better is the approximated value hence if you intend to use the above program in any program then you must ensure that you choose a subinterval as large as possible so as to get a more accurate result. This is evident in the two program runs, as the one using 100 subinterval gave a value much closer to the actual value as compared to using 10 as the number of subintervals.

Question: The trapezium rule or formula is a better approximation for calculating the area under a given curve. It also uses smaller rectangles in computing the total area. The sum of the areas, totalArea, of the trapezoids is given by

$$totalArea = \frac{\Delta x}{2}[\, f(x_0) + 2f(x_1) + 2f(x_2) + ...2f(x_{n-1}) + f(x_n)$$

or

$$totalArea = \Delta x\left[\frac{f(lower) + f(upper)}{2} + \sum_{i}^{n-1} f(x_i)\right]$$

where the lower and the upper are the lower and the upper limits of the given function and n is the number of strips (trapezoids) to be used. Write a program for the trapezium rule. Test your program with the function $f(x) = x^3 + 2$.

Program listing:

```
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;
inline double f(double x) {return (pow(x,3)+2.0);}
int main()
{
 double lower, upper, n;
 double x,deltaX, area;
 cout <<"Please enter the lower and upper limits of function ";
 cin >>lower >>upper;
 cout <<"Please enter the number of subintervals ";
 cin>>n;
 area=(f(upper)+f(lower))/2.0;
 deltaX=(upper-lower)/n;
 x=lower+deltaX;
```

```
for(int i=1;i<n;i++)
{
  area+=f(x);
  x+=deltaX;
}
 area*=deltaX;
cout <<"\nThe approximate value of the integral is " <<area;
cout <<endl;
system("PAUSE");
return 0;
}
```

Program run:

```
Please enter the lower and upper limits of function 0 3
Please enter the number of subintervals 100

The approximate value of the integral is 26.252
Press any key to continue . . .
```

Alternatively, you can code as follows:

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

inline double f(double x) {return (pow(x,3)+2);}
int main()
{
 int l,u; //l and u are for the upper and lower limits respectively
 int strips;
 double x,sum, dx;
 cout <<"Please enter the lower and upper limits of function ";
 cin >>l >>u;
 cout <<"Please enter the number of strips required ";
 cin>>strips;
 sum=0.5*(f(l)+f(u));
 dx=(double(u)-l)/strips;
 for(x=l+dx;x<=u-dx;x+=dx)
 {
   sum=sum+f(x);
 }
 sum=sum*dx;
 cout <<"The integral value of the given function is " <<sum <<endl;
```

```
 system("PAUSE");
 return 0;
}
```

Program run

```
Please enter the lower and upper limits of function 0 3
Please enter the number of strips required 100
The integral value of the given function is 26.252
Press any key to continue . . .
```

Question: Another method of estimating the area under a given curve is by using Simpson's rule. The Simpson's rule gives a better estimate than the rectangle method and the trapezoidal method. This method works best if the number of subintervals is even, thus with Simpson's rule, the interval [lower, upper] is divided into an even number n of subintervals, each having a length of Δx, and the totalArea is given by

$$totalArea = \frac{\Delta x}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + ... + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right]$$

Without using any decision making statement such as the if or the switch statement, write a program using the Simpson's rule for finding the area under a curve. Use the function $f(x) = x^3 + 2$ to test your program.

Program listing:

```
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;
inline double f(double x) {return (pow(x,3)+2.0);}
int main()
{
 double lower, upper, n;
 double x,deltaX, area;
 int coeff=4;
 cout <<"Please enter the lower and upper limits of function ";
 cin >>lower >>upper;
 cout <<"Please enter the number of subintervals ";
 cin>>n;
 area=(f(upper)+f(lower));
 deltaX=(upper-lower)/n;
 x=lower+deltaX;
 for(int i=1;i<n;i++)
 {
```

```
  area+=coeff*f(x);
  x+=deltaX;
  coeff=6-coeff;
 }
 area*=deltaX/3;;
 cout <<"\nThe approximate value of the integral is " <<area;
 cout <<endl;
 system("PAUSE");
 return 0;
}
```

Program run:

```
Please enter the lower and upper limits of function 0 3
Please enter the number of subintervals 10
The approximate value of the integral is 26.25
Press any key to continue . . .
```

As can be seen from the outputs of the three methods of finding area under a curve, the Simpson's rule is the best followed by the Trapezoidal method and then the rectangle method. Even with 10 subintervals, the Simpson's rule gives a better estimate of the area than using the trapezoidal method with 100 subintervals.

Question

For the above three questions on finding the area under a curve, a fixed function $f(x) = 5x + 10x - 72$ has been used meaning that the programs can be used to find the approximated integral value of only this function. You are therefore required at this stage to modify your solutions such that each of the above three programs can accept as input any valid polynomial function and output the approximated integral value of a specified range using a given number of strips.

Self Assessment 1-4

1. Convert each of the following mathematic expressions to a C++ arithmetic expression making use of appropriate user defined function where necessary.

(a) $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ where a, b and c are constants

(b)

$$f = a \quad + \sum_{n=1}^{n} \left(a \quad \cos \frac{n\pi x}{L} + b \quad \sin \frac{n\pi x}{L} \right)$$

Where a, n, b, L and x are constant.

2. what will sqrt(pow(3,4)) return?

3. Without using an if statement but rather a pre-defined function write a C++ statement that can assign the largest of *a, b, c* and *d* to large where *a, b c, d* and *large* are identifiers.

4. Without using any pre-defined function or the modulus operator (%) write your own function that can return the remainder of two numbers when the first number is divided by the second number.

SESSION 2-4: ARRAYS

2-4.1 One Dimensional Arrays

So far we have looked at variables that hold or can store only a single value at a time. It is possible to use a data structure which can hold many values, all of the same data type. Such a structure is referred to as an array. An array is a series of storage locations to hold data. Each storage location is about the same size for the particular data type the array is intended to store. The use of arrays therefore allows for a variable to be assigned more than one value at a time such that each value of the variable is referenced to by a subscript. Supposing there are 1000 values representing students' ages. It would be very tedious to assign each student's age a unique symbolic or variable name. The best way of handling this is to assign all the ages a common name and then use a subscript to refer to any age (element) at any particular time.

Arrays are declared using a square bracket and the syntax for declaring arrays is as follows:

```
dataType arrayName[size];
```

where *dataType* specifies the type of data values to be stored in the array. The *arrayName* is the name of the array. The *size* is an integer value or an expression that specifies the number of storage locations required to store all the elements of the array. Unlike other programming languages such as Pascal in which the subscript can start from any value, in C++ the subscript or the index always starts with zero. As such, any value less than zero or greater than size-1 cannot be used to refer to

the array elements as this will result in subscript out of range error message. Normally, with most programs that you might write, the size of arrays may not be known in advance or when even the size is known, it may change for each run of a program with different inputs. One way of solving this is to declare an array as large as possible such that it can handle any number of values likely to be stored. Since the exact size of an array may not be known in advance, there is the need to keep track of the number of values stored in an array. In such cases, the arrays tend to be partially filled. A typical situation may be when one intends to write his/her user defined function for converting a number in base 10 to say base 2. Since the program can be used for different numbers in base 10 and each number is likely to have different number of digits when converted to base 2, there will be the need to declare a very large array size so that as many base 10 numbers as possible can be used by the same program to convert them to base 2. A typical array declaration is as follows:

```
int marks[10];
```

The above will create the array marks to have 10 separate memory locations numbered from 0 to 9. Each of the 10 locations will hold a single integer value.

To store a value to a specific location of an array using an assignment statement the syntax is

```
arrayName[subscript]=value;
```

using an input statement will be

```
cin >>arrayName[subscript];
```

and using output statement will be

```
cout <<arrayName[subscript];
```

where *arrayName* is the name of the array. The *subscript* may be a variable that currently stores a value in the range of 0 to size-1 inclusive. For example, to assign the value 30 to the fifth location of the array named marks will be:

```
marks[4] = 30;
```

To access the value stored in a specified location of an array, the syntax is as follows:

```
arrayName[location];
```

where *arrayName* is the name of the array in question and the location specifies the index or the position of the element in the array. For example, the statement cout<<marks[5]; means display to the screen the 6^{th} value stored in the array marks.

Question

Write a program that can allow a user to first enter a set of n numbers to be stored in an array. The program should then ask a user to enter two integers say A and B. The A is required to be inserted in the array of numbers at position B. Checks must be made so that only valid numbers can be entered for B whenever the program is run. The output should be the new array.

Program listing

```
#include<iostream>
```

```
using namespace std;
void getInput();
void InsertItem();
void DisplayOutput(int m,char x[30]);
//10000 below is an arbitrary number, num is number to be inserted
double list[10000],num;
int pos,n; // pos will be the point of insertion
int main()
{
 getInput();
 DisplayOutput(n, "\nThe list BEFORE insertion is.....\n");
 InsertItem();
 DisplayOutput(n+1, "\n\nThe list AFTER insertion is.....\n");
 cout <<endl;
 system("pause");
 return 0;
}

void getInput()
{
    cout <<"\nPlease enter number of numbers in the set: ";
    cin >>n;
    cout<<"\n\n";
    for(int i=0;i<n;i++)
    {
       cout <<"Please enter number " <<i+1 <<" in the list ";
       cin>>list[i];
    }
   cout <<"Please enter number to be inserted and its position ";
   cin >>num >>pos;
   while (pos<0||pos>n)
   {
     cout <<"Position cannot be negative or greater than " <<n;
     cin >>pos;
   }
   system("cls");
}

void InsertItem()
{
   for(int i=n;i>=pos-1;i--)
      list[i+1]=list[i];
   list[pos-1]=num;
}
```

```
void DisplayOutput(int m, char x[30])
{
   cout <<x <<"\t";
   for(int i=0;i<m;i++)
    cout <<list[i] <<"   ";
   return;
}
```

Program run

```
Please enter number of numbers in the set: 6

Please enter number 1 in the list 12
Please enter number 2 in the list 10
Please enter number 3 in the list 100
Please enter number 4 in the list 34
Please enter number 5 in the list 67
Please enter number 6 in the list 90
Please enter number to be inserted and its position 90 4


The list BEFORE insertion is.....
        12   10   100   34   67   90

The list AFTER insertion is.....
        12   10   100   90   34   67   90
Press any key to continue . . .
```

Question

Employees of JB and SONS Consultants Limited are paid at an hourly rate of 25,000.00 cedis per hour for regular hours and one and one half times per hour for overtime hours in a week. Any hour worked over forty hours per week is overtime. The following national tax **sliding** scale is then applied to determine the amount of tax to be paid by an employee.

Gross pay	Tax Rate (%)
first 125,000	0
Next 125,000	5
Next 1,500,000	10
Next 2,750,000	15
Next 1,250,000	20

Excess over 5,000,000	30

In addition, 6% of an employee's gross pay is withheld for Social Security or GUSSS contribution, 3% is withheld as constituency tax and 2% is withheld by the employer as welfare contribution. If an employee has more than three dependants then an amount of 5,000.00 cedis is paid for each dependent in excess of 3 towards National Health Insurance Scheme. The company has no more than 100 employees. You are required to write a program to compute a worker's gross pay, the deductions and his/her net pay. Your program should allow for a number of staff details to be entered for the necessary computations. Your program should also prompt after each staff's computation whether there are more records to be entered.

Solution:
For this problem we will read employee's details, compute the necessary items and store them in an array before writing out the output. We will assume that for each employee, we have his/her name, hours worked in a week and the number of dependants, and we shall use the variables Name, HoursWorked and Dependants respectively for them. In addition, we will use the variable names GrossPay, Tax, SSC, ConsTax, Welfare, NHIS and NetPay for the gross pay, Income Tax, Social Security Contribution, Constituency Tax, Welfare contribution, National Health Insurance Scheme and net pay respectively.

The table below shows the minimum amount of tax one will have to pay if his/her gross pay is in the range given under Gross pay, plus the excess of the gross pay over the minimum amount of the range less 1 times the corresponding tax rate. For example, any staff whose Gross pay is between 250,001 to 1,750,000 inclusive will have to pay a miminum of 6,250 cedis plus 10% of the excess over 250,000 cedis.

Gross pay	Tax Rate (%)	Minimum tax to pay if Gross pay is in the range of amount to the left in.
250,001-1,750,000	10	6,250
1,750,001-2,750,000	15	156,250
2,750,001-5,000,000	20	306,250
Over 5,000,000	30	756,250

Program listing

```
#include<iostream>
#include<fstream>
#include<cctype>
```

```
#include<string>
using namespace std;
void getInputCompute();
void DisplayOutput();
const int size=100;
int count=0;
char Name[size][30];
int HoursWorked[size], Dependants[size];
double GrossPay[size], Tax[size], SSC[size], ConsTax[size];
double Welfare[size],NetPay[size], NHIS[size];
ofstream fcout;
int main()
{
  fcout.open("output.dat",ios::app);
  getInputCompute();
  DisplayOutput();
  return 0;
}
void getInputCompute()
{
        int s;

        double scale[]={6250,156250,306250,756250};
        int rates[]={5,10,15,20,30};
        double limit[]={125000,250000,1750000,2750000,5000000};
        while(s!=9)
        {   char fn[20],ln[20];
           cout <<"\n\nPlease enter the following information ";
                cout <<"\nfirst name and Surname of staff     :";
                cin>>fn; cin>>ln;
                strcat(fn,"  ");
                strcpy(Name[count],strcat(fn,ln));
                cout <<"Hours worked      :";cin>>HoursWorked[count];
                cout <<"No. of Dependants :";cin>>Dependants[count];
                if (HoursWorked[count]> 40)
                  GrossPay[count]=40.0*25000+30000*(HoursWorked[count]-40);
                else
                        GrossPay[count]=25000*HoursWorked[count];
                SSC[count]=0.06*GrossPay[count];
                ConsTax[count]=0.03*GrossPay[count];
                Welfare[count]=0.02*GrossPay[count];
                if (Dependants[count]>3)
                        NHIS[count]=5000*(Dependants[count]-3);
                else
```

```
                    NHIS[count]=0;

                if(GrossPay[count]>limit[4])//gross pay >5,000,000
                    Tax[count]=scale[3]+rates[4]*(GrossPay[count]-limit[4])/100;
                else if(GrossPay[count]>limit[3])//gross pay >2,750,000
                    Tax[count]=scale[2]+rates[3]*(GrossPay[count]-limit[3])/100;
                else if(GrossPay[count]>limit[2])//Gross pay >1,750,000
                    Tax[count]=scale[1]+rates[2]*(GrossPay[count]-limit[2])/100;
                else if(GrossPay[count]>limit[1])//Gross pay >250,000
                    Tax[count]=scale[0]+rates[1]*(GrossPay[count]-limit[1])/100;
                else if(GrossPay[count]>limit[0])//Gross pay >125,000
                    Tax[count]=rates[0]*(GrossPay[count]-limit[0])/100;
                else
                    Tax[count]=0;
NetPay[count]=GrossPay[count]-(Tax[count]+SSC[count]+ConsTax[count]);
NetPay[count]=NetPay[count]-(Welfare[count]+NHIS[count]);
 count++;
cout <<"Please enter 9 to end, any other character to enter data ";
 cin>>s;
}
}
void DisplayOutput()
{       fcout<<"    Name\tGross Pay \tTax\t\tSSC\t\tConsTax\t\tWelfare\t\tNHIS";
   fcout<<"\t       Net Pay" <<endl;
        for(int i=0;i<count;i++)
        {   fcout.setf(ios::fixed);
                fcout.width(10);fcout.precision(2);
                fcout <<Name[i] <<"\t" <<GrossPay[i] <<"\t";
                fcout <<Tax[i] <<"\t" <<SSC[i];
                fcout <<"\t" <<ConsTax[i] <<"\t" <<Welfare[i] <<"\t" <<NHIS[i];
                fcout <<"\t\t" <<NetPay[i] <<endl;
        }
        fcout.close();
}
```

Input data:

First Name	Last Name	Hours worked	Dependants
Issac	Arhin	90	2
James	Arthur	60	4
Joyce	Ansah	50	3
Kwame	Annam	45	1
Ekua	Sophia	20	5
Esi	Kofua	35	4

Maame	Obu	42	0
Kofi	Yeh	65	4
Kingsley	Doe	38	6
Edward	Acquah	72	2

Program output

When you run the above program, your output format will be slightly better than presented here as values have been squeezed so as to fit on lines.

Name	Gross Pay	Tax	SSC	ConsTax	Welfare	NHIS	Net Pay
Isaac Arhin	2500000.00	268750.00	150000.00	75000.00	50000.00	0.00	1956250.00
James Arthur	1600000.00	141250.00	96000.00	48000.00	32000.00	5000.00	1277750.00
Joyce Ansah	1300000.00	111250.00	78000.00	39000.00	26000.00	0.00	1045750.00
Kwame Annam	1150000.00	96250.00	69000.00	34500.00	23000.00	0.00	927250.00
Ekua Sophia	500000.00	31250.00	30000.00	15000.00	10000.00	10000.00	403750.00
Esi Kofua	875000.00	68750.00	52500.00	26250.00	17500.00	5000.00	705000.00
Maame Obu	1060000.00	87250.00	63600.00	31800.00	21200.00	0.00	856150.00
Kofi Yeh	1750000.00	156250.00	105000.00	52500.00	35000.00	5000.00	1396250.00
Kingsley Doe	950000.00	76250.00	57000.00	28500.00	19000.00	15000.00	754250.00
Edward Acquah	1960000.00	187750.00	117600.00	58800.00	39200.00	0.00	1556650.00

Question: In Computer Science International College, letter grades are assigned to numeric scores or marks by using the following grading on the curve scheme. In this scheme, letter grades are determined as follows:

Exams Mark (x)	Letter Grade
$x \geq m + 1.5\sigma$	A
$m + 0.5\sigma \leq x < m + 1.5\sigma$	B
$m - 0.5\sigma \leq x < m + 0.5\sigma$	C
$m - 1.5\sigma \leq x < m - 0.5\sigma$	D
$x < m - 1.5\sigma$	F

where m and σ are the mean and standard deviation respectively of the exams marks of all students in a particular class. You are required to write a program to accept as input the exams marks of all students in a class and then output the marks and their corresponding letter grades. The output should be in decreasing magnitude of exams marks.

Program listing

```
#include <iostream>
#include <cstdlib>
# include <cmath>
using namespace std;
double mean(int x[], int n);//computes average mark
double stdev(int x[], int n);//computes standard deviation
void getInput(int x[],int& n);//for getting input marks
void DisplayOutput(int x[],int n);//To display output marks and grade
```

```
void SortMarks(int x[],int n);//To arrange output marks on merit
int main()
{
    int x[10],n=0;
    getInput(x,n);//get input marks
    SortMarks(x,n);//sort the marks in order
    DisplayOutput(x,n);//compute letter grades and output
    system("PAUSE");
    return 0;
}

double mean(int x[], int n)
{
  double sum=0;
  for(int i=0;i<n;i++)
     sum=sum+x[i];
 return (sum/n);
}

double stdev(int x[], int n)
{
  double sum=0;
  double xbar=mean(x,n);
  for(int i=0;i<n;i++)
    sum=sum+pow((x[i]-xbar),2);
  return sqrt (sum/n);
}

void getInput(int x[],int& n)
{
 cout<<" Please enter the number of students    " ;
 cin>> n;
 for(int i=0;i<n;i++)
 {
    cout << "Enter Mark[" <<i+1 <<"]  ";
    cin>> x[i];
  }
}

void SortMarks(int x[], int n)
{
 for(int i=0;i<n-1;i++)
  for(int j=0;j<n-i-1;j++)
    if(x[j]<x[j+1])
```

```
   {
    int temp=x[j+1];
    x[j+1]=x[j];
    x[j]=temp;
  }
}

void DisplayOutput(int mark[],int n)
{
    double xbar,std;
    xbar=mean(mark,n);
    std=stdev(mark,n);
    cout <<"\n\t\tExams Mark\t\tGrade\n";
    for(int i=0;i<n;i++)
    {
      cout <<"\t\t   " <<mark[i] <<"\t\t";

      if (mark[i]<xbar-1.5*std)
          cout <<"\t  F" ;
      else if ((mark[i]>=xbar-1.5*std)&& (mark[i]<xbar-0.5*std))
          cout <<"\t  D" ;
      else if ((mark[i]>=xbar-0.5*std)&& (mark[i]<xbar+0.5*std))
          cout <<"\t  C" ;
      else if ((mark[i]>=xbar+0.5*std)&& (mark[i]<xbar+1.5*std))
          cout <<"\t  B" ;
      else
          cout <<"\t  A" ;
      cout <<endl;
    }
     cout <<endl;
}
```

Sample run

```
Please enter the number of students    12
Enter Mark[1]  90
Enter Mark [2]  65
Enter Mark [3]  87
Enter Mark [4]  75
Enter Mark [5]  63
Enter Mark [6]  88
Enter Mark [7]  45
Enter Mark [8]  39
```

```
Enter Mark [9]  77
Enter Mark [10]  81
Enter Mark [11]  70
Enter Mark [12]  65

        Exams Mark        Grade
          90              B
          88              B
          87              B
          81              B
          77              C
          75              C
          70              C
          65              C
          65              C
          63              C
          45              F
          39              F

Press any key to continue . . .
```

Question

Write a C++ program that can accept two positive integers as inputs. The first integer being a number to be converted to a different base and the second being the base required. Your program should accept only bases between 2 and 16 inclusive. Your output should be the first integer in base of the second integer.

Program listing

```
#include<iostream>
using namespace std;
void bases(int number, int base);
int main()
{
 int base,n;
 do
 {
  cout<<"Please enter a VALID number to convert ";
  cin>>n;
 } while(n<0);

 do
```

```
 {
   cout<<"\t Enter the base required [2-16] inclusive: ";
   cin>>base;
 } while (base<2||base>16);
 cout<<"The number " <<n <<" in base " <<base <<" is ";
 bases(n,base);
 cout <<endl;
 system("pause");
 return 0;
}
void bases(int n, int b)
{
    char digits[]={"0123456789ABCDEF"};
    int dig[100],i=0;
    while(n>0)
    {
       dig[i]=n%b;
       n=n/b;
       i++;
    }
    for(int j=i-1;j>=0;j--)
     cout <<digits[dig[j]];
 }
```

Program runs

```
Please enter a number to convert 27480
        Enter the base required [2-16] inclusive: 16
The number 27480 in base 16 is 6B58
Press any key to continue . . .

Please enter a number to convert 2005
        Enter the base required [2-16] inclusive: 2
The number 2005 in base 2 is 11111010101
Press any key to continue . . .
```

2-4.2 Multi Dimensional arrays

The above definition of arrays is for one dimensional arrays. They are one dimensional because the elements in the array are referenced by using a single subscript. However, it is possible to define arrays such that each element is referenced by more than one subscript. If array elements are referenced by two subscripts then we have a two dimensional array, by three subscripts we have three dimensional array etc. For 2 dimensional arrays, the elements of the arrays are stored in rows and columns. The syntax for defining a two dimensional array is as follows:

dataType arrayName[Rows][Columns];

where *dataType* is one of the C++ data types. The *arrayName* is the name of the two dimensional array. The *Rows* and the Columns specify the number of rows and columns required to store the array elements respectively. The total *number* of memory locations that will be reserved for a 2 dimensional array is given by

rows*Columns

Note that for a 2 dimensional array, the first element of the array has its row and column subscript to be zero.

The following is a typical declaration of a two dimensional array.

int weights[10][15];

The above define weights as 2 dimensional array with 10 rows and 15 columns thus giving it a total of 150 storage or memory locations for the elements of the array. Just like a 1-d array, the rows and columns of the 2-d arrays have their index values starting from 0.

HOW ARRAYS ARE STORED IN MEMORY

When an array is declared, its elements are stored in a specific order, however, you can reference the elements in any order. The locations where the elements are stored within the array are usually specified by a programmer. The following is just to give a rough idea as to the order in which the elements are stored in memory. For a one dimensional array, they are stored as follows (assuming the name of the array is PAY and the size is n where *n* is an integer).

For a higher dimensional array, the elements are stored in column major order, that is all elements in column 0 would be stored before that of column 1, etc. The method of storing elements in a three dimensional array is similar to the 2D above.

2-4.3 Arrays as Function Argument or Parameter

Arrays can be used as function arguments or parameter in the same way as other variables. However, in passing an array to a function there is nothing like an argument or a parameter, that is, there is no call-by-reference and no call-by-value. When an array is passed to a function, it is simply referred to as array argument. In passing an entire array, the array argument does not tell the function the size of the array. There is therefore the need to have another parameter of the type int that gives the size of the array. Array arguments are treated liked call-by-reference, as such if a function changes the array parameter, the change is equally made to the array arguments. The syntax for a function prototype with an array argument is as follows:

for a one dimensional

dataType functionName(ArrayType ArrayName[],…,int size);

for a two dimensional

dataType functionName(ArrayType ArrayName[][size1],…,int size2);

where *dataType* is the data type of results to be returned by the function, *ArrayType* is also the data type of the array and *ArrayName* is the name of the array. The size for a 1-D specifies the number of elements, for a 2-D, *size1* specifies the number of columns and *size2* the number of rows. The solution to the following question illustrates the use of arrays as function arguments or parameters.

Generally, when an entire array is passed to a function, the function can change any of the values of the array. Sometimes, it is important to prevent functions from changing array values. To prevent functions from changing arrays passed to them the data type of the array in the function prototype and the function header should be preceded with the reserved word const. For example:

```
void KeepUs(const int x[], int size)
```

QUESTION: In mathematics, a square matrix is said to be STOCHASTIC if it satisfies the following conditions:
i) if all its elements are non-negative numbers and
ii) if the sum of all elements in any particular row equals one.
Making use of functions, write a program to accept a 3 x 3 matrix as input and output STOCHASTIC if it satisfies the above conditions otherwise output NOT STOCHASTIC.

Program listing

```
#include <iostream>
#include <cstdlib>
using namespace std;
void SignCheck(double mat[][3]);
void SumCheck(double mat[][3]);
void InputMat(double mat[][3]);
bool sat=true;

int main()
{    double matrice[3][3];
     InputMat(matrice); //get input values
     SignCheck(matrice);//check if all values are positive
     if(sat==true)
       SumCheck(matrice);
     if(sat==true)
       cout<<"The Input Matrix is Stochastic " <<endl;
     else
       cout<<"The Input Matrix is NOT Stochastic " <<endl;
     system("PAUSE");
     return 0;
```

```
}
void SignCheck(double mat[3][3])
{
  for(int i=0;i<3;i++)
   for(int j=0;j<3;j++)
    if(mat[i][j]<0)
    {
     sat=false;
     return;
    }
}
void SumCheck(double mat[][3])
{
   for(int i=0;i<3;i++)
   {
    double sum=0;
    for(int j=0;j<3;j++)
    sum=sum+mat[i][j];
    if(sum !=1)
     {
       sat=false;
       return;
     }
   }
}

void InputMat(double mat[][3])
{
  for(int i=0;i<3;i++)
   for(int j=0;j<3;j++)
   {
    cout <<"Enter Matrix[" <<i <<"][" <<j <<"]: ";
    cin>>mat[i][j];
   }
}
```

Question: A contraceptive manufacturing company, JB and DAUGHTERS Ltd, manufactures 10 different types of contraceptives that they refer to them as contraceptive 1, contraceptive 2, etc. The company has 25 outlets usually referred to as outlet 1, outlet 2, etc that sell their drugs. At the end of each month, all the outlets send their sales transactions to the manufacturing company. The manufacturing company is now considering increasing the production of the most used contraceptive. You are required to write a C++ program that accepts as input the quantity of each

type of contraceptive sold by each of the outlets and output the following information with appropriate captions:

(1) The total number of each contraceptive sold by all outlets
(2) The total number of contraceptive sold by each outlet.
(3) The contraception with the largest sales quantity.

Solution:

This question is nothing more than writing a program to find the total of each row and total of each column of elements in an array as well as returning the largest column total. Since we have not dealt with files, the inputs are to be entered from the keyboard and output display to the screen. Let us assume we want the output to have the following format;

```
                        Contraceptives
  Outlet :        1   2    3  …   6    7    8    9  10      Total
---- ------------------------------------------------- ---------
      1  :                                                      :
      2  :                                                      :
      :                                                       :
      25 :                                                    :
 ---------------------------------------------------------------
Total
Contraceptive with the largest sales quantity of is Contraceptive n.
```

Program listing

Since the program run will be quite long if we are to use the 25 rows, we shall limit this for 10 outlets. The program can run for any number of contraceptives and any number of outlets just by changing the value of MaxRows and MaxColumns in the program. For output formatting, we shall also assume that all sales quantity per contraceptive is nor more than say 100.

```
#include<iostream>
#include<iomanip>
using namespace std;
const int MaxRows=11; // Extra row to store row totals
const int MaxColumns=11; //Extra column to store column totals
void AcceptInput(void);
void WriteColumnTotals(void);
void RowAndTotals();
double SalesArray[MaxRows][MaxColumns];
//The following will be used to store largest sale amount and the outlet number
double largestSales=0, contra=0;
```

```
int main()
{
  int i;
  AcceptInput();
  cout<<"\n\nOutlet\t\t\tContraceptive \t\t\t   Totals\n";
  cout <<"\t";
  for (i=0;i<MaxColumns-1;i++)
       cout<<setw(5) <<(i+1);
  cout<<endl;
  cout<<"---------------------------------------------------------------------\n";
  RowAndTotals();
  cout<<"\n---------------------------------------------------------------------\n";
  WriteColumnTotals();
  cout<<"\n\nThe contraceptive with the largest sale quantity of "
      <<largestSales <<" is " <<(contra+1);
  cout <<endl;
  system("pause");
  return 0;
} // end of the main function

void AcceptInput(void)
{
 int rows,columns;
 cout<<"Please enter value for Contraceptive \n";
  for (rows=0;rows<MaxRows-1; rows++)
     for(columns=0; columns<MaxColumns-1;columns++)
      {
       cout <<(rows+1) << " of outlet " <<(columns+1) <<": ";
       cin>>SalesArray[rows][columns];
      };// end of the for statments
 } // end of the procedure AcceptInput

void RowAndTotals(void)
{
 int rows,columns;
 double rowTotal;
 for (rows=0;rows<MaxRows-1;rows++)
 {
   rowTotal=0;
   cout <<"\n " <<setw(2) <<(rows+1) <<"      " ;
   for (columns=0;columns<MaxColumns-1;columns++)
     {
       rowTotal=rowTotal+SalesArray[rows][columns];
       cout<< setw(5)<< SalesArray[rows][columns];
```

```
      }
       SalesArray[rows][MaxColumns-1]=rowTotal;
      cout <<" |" <<setw(5) <<rowTotal;
 } // end of for rows
 } // end of the function ComputeTotals


void WriteColumnTotals()
{
 int rows,columns;
 double ColumnTotal;
 cout <<"\n\t";
 for (columns=0;columns<MaxColumns;columns++)
 {
   ColumnTotal=0;
   for (rows=0;rows<MaxRows-1;rows++)
     ColumnTotal=ColumnTotal+SalesArray[rows][columns];
  cout <<setw(5) <<ColumnTotal;
  SalesArray[MaxRows-1][columns]=ColumnTotal;
 }
  for (columns=0;columns<MaxColumns-1;columns++)
  if (SalesArray[MaxRows-1][columns]>largestSales)
    {
     largestSales=SalesArray[MaxRows-1][columns];
     contra=columns;
    }
 } // end of the WriteColumnTotals function
```

Program run

```
Outlet          Contraceptive                    Totals
            1     2     3     4     5     6     7     8     9    10
-------------------------------------------------------------------------------

 1          2    35    63    35    23     4     9    85    23    21 |  300
 2         12    23    41    52    63    21    20     4     3     5 |  244
 3         96    41     2    10     3     5     6    15    21    10 |  209
 4          3     5    96    65    12    10     5     9     7     8 |  220
 5         41    20     9     6     5    41    23    10    23     8 |  186
 6          9     6     5    74     2     3     1    45    52    21 |  218
```

```
7       35   96   32   14   23   25   10   30   12   10 |  287
8       45   25   21   36   35   21   24   25   10    5 |  247
9        6   74    9    6   41   21   20   10    5    6 |  198
10      41   12   35   47   12   10   25   16   88    4 |  290
--------------------------------------------------------------------------

        290  337  313  345  219  161  143  249  244   98 2399

The contraceptive with the largest sale quantity of 345 is 4
Press any key to continue . . .
```

INITIALISATION OF ARRAYS

Arrays can be initialised during declarations. If a program is to be run a couple of times with the same set of data then it will be better to initialise the array elements by the data else the user may have to enter the same data each time the program is run. If array locations are to be used as accumulators then again it will be better to initialise the array elements to zeros say. During array initialisation, for a one dimensional array, the array size may be omitted. If this is done the compiler will automatically allocate sufficient memory locations for the array elements. For a two dimensional array, the number of rows may be omitted but the number of columns must be specified. When the number of array elements are known and the number of columns are known then the number of rows can be determined by the compiler and hence why it is necessary that the number of rows be specified. The array elements are listed in a pair of brackets separated by commas. The syntax for initialising an array is as follows:

```
arrayType ArrayName[size] ={dataList};
arrayType ArrayName[] ={dataList};          } for one dimensional arrays

arrayType ArrayName[rows][columns] ={dataList};
arrayType ArrayName[][columns] ={dataList};  } for two dimensional arrays
```

where *arrayType*, *ArrayName*, *size* is the data type of array elements, the name and the size of the array respectively. The *dataList* is the values for the array initialisation separated by commas. The *rows* and *columns* are the number of rows and columns of the array. The following are all valid array initialisations:

```
int cost[] = {500, 750, 220, 320, 145, 789};
int cost[6] ={500, 750, 220, 320, 145, 789};
int cost[][3] = {500, 750, 220, 320, 145, 789};
int cost[2][3] = {500, 750, 220, 320, 145, 789};
```

If the number of columns is not specified during array initialisation for a two dimensional array the following error message will be displayed by the (Bloodshe Dev-C++) compiler:

error C2087: '<Unknown>' : missing subscript

If, instead of specifying the number of columns, a programmer chooses to specify the number of rows the following error messages will be displayed:

error C2087: '<Unknown>' : missing subscript
error C2078: too many initializers

One other condition under which the error message error C2078: too many initializers will be displayed is when the data list contains more elements than the number of memory locations for the array. Although the compiler displays an error message when the number of data items is more than the number of storage locations, it does not do so when there are fewer number of data items than the memory locations assigned to the array. For example if you initialise an array as follows the compiler will give no error message and neither will it give a warning message.

```
int cost[][4]={1,2,3,4,5,6,7,8,9};
```

Note that in the above example since the number of columns has been specified to be 4, one would have thought that the number of data items will be divisible by 4 but this is not the case yet the compiler is also not giving any error message or warning. In memory the array will be stored as follows:

```
1  2  3  4
5  6  7  8
9  0  0  0
```

Again, note that the compiler automatically assigns zeros at the end of the last row of the array if the data given is not sufficient. This is true only when at least there is one data value for the last row else instead of zeros unpredictable values are used. Now, let us look at some more examples to get a good understanding of initialising arrays with insufficient data. In fact, if possible make sure that the data items are equal in number to the number of memory locations for the array.

Consider the following program where the bolded line deals with the array initialisations:

```
#include<iostream>
using namespace std;
int main()
{
    int cost[][4]={0};
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<4;j++)
            cout <<cost[i][j] <<"\t";
      cout <<endl;
```

```
        }
        return 0;
}
```

1. One would have thought that since the data items are fewer than expected, the computer will assign zeros to the remaining array locations. The compiler will only add zeros to locations on a particular row if at least one data item on that row is provided. Any row with no data provided will have unpredictable values assigned to the array locations. The output from the above program will be as follows:

```
        0           0           0           0
  1245120     4208617           1     4394608
  4394416    34603536    58851856  2147348480
```

2. if we change the bolded line to **int cost[][4]={0,0,0,0,2};** then the output will be as follows:

```
        0           0           0           0
        2           0           0           0
  1245120     4208617           1     4394608
```

Note what has happened to values on the third row. They are different from what was obtained in 1 above. This tells us to pay nore attention to array initiliations.

2-4.4 Arrays of Characters

The C++ has a data type char and as such variables that are to store non digits can be defined to be of the char data type. Such variables can store only one character at a time. It is sometimes necessary to have more than one character in a data item that needs to be stored in a variable. The only way this can normally be done in C++ is to define a variable as an array of characters. An array of characters constitutes a string.

The syntax for declaring a string variable is as follows:

```
char StringName[size];
```

where *StringName* is the name of the string. The *size* is maximum number of characters that can be stored in the string. Note that for strings an extra storage location is required to store the null character ('\0') as a delimiter. This implies if size is the string size specified then the maximum number of characters that can be stored in the string is size-1. String characters are accessed just as array of numbers by using subscripts. For example name[5] will mean the 6^{th} character stored in string name. Unlike array of numbers, you can print or display an entire array of characters (string) by specifying just the string name. For example, to print all characters stored in the string name, one can code

```
cin>>name;
```

Strings are initialised in the same way as array of numbers. As such, the syntax for initialising a string during declaration is as follows:

```
char StringName[size]="value";
```

where *StringName* is the name of the string and *size* defines the number of characters that can be stored in the string. The value is the initialisation characters (alphabets and digits). For example, we can initialise a string virus to TROJAN as follows:

```
char virus[]="TROJAN";
```

It is also correct to write

```
char virus[]={'T','R','O','J','A','N'}
```

but there is a difference between the two statements. With char viruss="TROJAN" the compiler automatically adds a null character ('\0') after the character 'N' to indicate the end of the string. With char virus[]={'T','R','O','J','A','N'} the compiler does not add the null character.

Since a string is just like any other array, subscripts can be used to refer to the string elements. For example, we can store the character T as the 5^{th} item in the string Letters as follows:

```
Letters [4] = 'T';
```

I hope you have not forgotten that array index starts from zero and hence the fifth location of an array has the subscript to be 4. Whenever you want to manipulate string characters you have to be careful not to replace the null character with a different character this is because when a string loses the null character it ceases to behave like a string. Also, many of the string functions depend on the presence of the null character to mark the end of a string value. Note that the null character is not printed or considered as part of the string.

Two dimensional strings are declared in the same way as two dimensional array of numbers. As such, the syntax for declaring a 2-d string is as follows;

```
char StringName[rows][size];
```

where the *rows* specifies the number of rows required. The *size-1* specifies the maximum number of characters that can be stored on a row. For example, to declare a string capable of storing names of 100 students where no student's name is more than 60 characters long, the string declaration may be as follows if the string variable is name:

```
char name[100][61];
```

Note that one has been added to the number of characters that can be stored on a row to cater for the null character. To print a name on any particular row, the column need not be specified. For example, to display to the screen the fourth name stored in the string name, one can code cout<<name[3];

STRING INPUT AND OUTPUT

The insertion operator, >> can be used to accept an array of characters from the keyboard. One problem with the use of the insertion operator is that it stops reading characters when it encounters a whitespace (blanks and line breaks). Consider the following program:

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
   char mess[200];
   cout <<"Please enter a string of at most 200 characters ";
   cin>>mess;
   cout<<mess;
  return 0;
}
```

If during program execution one enters the string "For what shall a man be profited if he refuses to learn C++" as input for mess, the output to the screen will be "For". You must therefore take care when using strings in a program. C++ has member functions that make it possible for spaces to be read and treated as characters. Some of these are as follows:

The member function get

The member function get is used with the insertion operator to read one character at a time and store it in a variable of type char. Since the get behaves similar to the cin, most C++ programmers treat get as a member function of the stream cin. The syntax of the get function is as follows:

```
cin.get(CharVariable);
```

where *CharVariable* is a variable of the char type. As an example, let us consider the following program:

```
#include<iostream>
using namespace std;
int main()
{
        char mess[200];
        int i=-1;
        cout <<"Please enter a line of input ";
        do
        {
           i++;
          cin.get(mess[i]);
        } while (mess[i]!='\n');
        for(int j=0;j<i;j++)
          cout<<mess[j];
  return 0;
}
```

The above program will accept any string that is entered at the keyboard and output it in the same way. It allows characters to be entered until the enter key is pressed. If the bolded lines in the above program is replaced with **cout <<mess**, the string entered will be displayed but then the compiler will add unpredictable characters to make the string equal to 200 as specified earlier in the program. As such, you must be careful how to display your string after you have declared a much larger storage locations for it. Note that instead of cout in the above program we could have used cout.put function. Thus, the cout.put and cout are very similar. The syntax for the put function is as follows:

```
cout.put(CharVariable);
```

where *CharVariable* is the variable whose content is being displayed.

The member function getline

The getline function is similar to the get function except that the getline function allows an entire line of text to be entered at a time. The general syntax is as follows:

```
cin.getline(stringVariable,size+1);
```

where the *stringVariable* is the name of the string and *size* is the maximum number of characters that can be stored in the *stringVariable*. The following program will allow a user to input a string of up to 200 characters and then display the input to the screen.

```
#include<iostream>
using namespace std;
int main()
{
        char mess[200];
        cout <<"Please enter a line of input ";
        cin.getline(mess,200);
        cout<<mess;
  return 0;
}
```

Unlike the get, the getline function does not allow the compiler to add characters to the string to make it to the size of the string. If one enters 'It is finished' as input the output will be exactly the same. Note that if the number of characters entered is more than the string size then the excess characters will be truncated from the right hand side of the string. For example, if "Opanyin Kweku Okoh" is entered as an input to a string variable, name, of size 10, then

```
cout <<name;
```

will output "Opanyin K".

The member function read

The read function is similar to the getline function. It is used to read a specified number of characters into a variable. The general syntax is as follows:

cin.read(stringVariable,size+1);

where the *stringVariable* is the name of the sting variable to store the input. The *size* is the number of characters to receive as input to the string variable. The following program illustrate the use of the read function.

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
        char mess[20];
        cout <<"Please enter a line of input: ";
        cin.read(mess,10);
        cout<<mess <<endl;
   system("pause");
 return 0;
}
```

program run:

```
Please enter a line of input: Isaac Oheneba Arthur
Isaac OhenA ☻ ⌂♥
Press any key to continue . . .
```

With the read function, program execution will continue only when the specified number of characters have been entered. For example, here one is expected to enter 10 characters for the variable mess. If the number of characters entered is less than 10, the read statement will force program execution to pause until the number of characters is at least 10. In the above program, we could see that the mess was declared to be a string of size 20 but the read statement is to accept only 10 characters into mess. What happens is that the compiler will add some characters at the end of the input string in an attempt to make up the required 20 characters as can be seen from the program run hence care must be taken when displaying content of identifiers obtained by using the read function. In fact the same thing happens when in the read function one specifies the exact string size. The main reason for the added characters is that the compiler is unable to determine the end of the string when it comes to output hence the best thing to do will be to store the null character as the last character in the input string as shown in the program below (see bolded line).

Program listing:

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

```
int main()
{
        char mess[20];
        cout <<"Please enter a line of input: ";
        cin.read(mess,10);
         mess[10]='\0';
        cout<<mess <<endl;
   system("pause");
 return 0;
}
```

Program run (using the same input as above):

```
Please enter a line of input: Isaac Oheneba Arthur
Isaac Ohen
Press any key to continue . . .
```

Question

Without using any user defined functions, write a program that accepts as input N number of names and sorts them in alphabetical order of surname. During program execution, the names will be entered in the form first name and last (surname) name. The names will be entered in lower cases but the output should have the first character in each surname and first name in capitals. Finally, for each name, the surname must be displayed first since the sorting is in order of surname.

Solution: We will use the array names to store the names that we want to sort. We shall assume up to one thousand names and that each first name and last name is no more than 20 characters. The first column of names will be used to store first name and the second column for the surname. Thus names will be declared as char names[1000][2][20]. The 1000 is the maximum number of names, the 2 is the number of columns required for storing first and surnames and the 20 is the maximum number of characters that each part of the name can contain.

Program listing

```
#include<iostream>
#include<iomanip>
#include<cctype>
using namespace std;
int main()
{
 char names[1000][2][20],name[20];
 int i,n,j;
 cout<<"Please enter the number of names required to be sorted ";
 cin>>n;
```

```
cout <<"\nEnter the names in form \tFirst Name \tLast Name\n";
for(i=0;i<n;i++)
{
  cout <<"\t\t\t" <<(i+1) <<"\t";
  cin>>names[i][0] >>names[i][1];
}
system("cls");
for(i=0;i<n-1;i++)
 for(j=0;j<n-i-1;j++)
    if(strcmp(names[j][1],names[j+1][1])>0)
    {
       strcpy(name,names[j][1]);
       strcpy(names[j][1],names[j+1][1]);
       strcpy(names[j+1][1],name);
       strcpy(name,names[j][0]);
       strcpy(names[j][0],names[j+1][0]);
       strcpy(names[j+1][0],name);
    }
cout<<"\t\tThe sorted names will be as follows:\n";
cout <<"\n\tLast Name\t\tFirst Name\n";
cout <<setw(60); cout.fill('_'); cout<<"_\n";
cout.fill(' ');
 for(i=0;i<n;i++)
 {
    names[i][0][0]=toupper(names[i][0][0]);
    names[i][1][0]=toupper(names[i][1][0]);
  cout <<"\n\t" <<setw(30) <<left <<names[i][1] <<setw(30) <<names[i][0];
 }
 cout <<endl;
 system("pause");
 return 0;
}
```

Program run

Please enter the number of names required to be sorted 10

Enter the names in form	First Name	Last Name
1	john	arthur
2	james	mensah
3	esi	kofua
4	abena	ansah
5	william	acquah
6	grace	ashong

7	angela	harrison
8	gabriel	zon
9	araba	kwansima
10	kweku	paintsil

The sorted names will be as follows:

Last Name	First Name
Acquah	William
Ansah	Abena
Arthur	John
Ashong	Grace
Harrison	Angela
Kofua	Esi
Kwansima	Araba
Mensah	James
Paintsil	Kweku
Zon	Gabriel

Press any key to continue . . .

Question

Rewrite the above program so that the names are all written in uppercases.

Program listing

```
#include<iostream>
#include<iomanip>
#include<cctype>
using namespace std;
int main()
{
 char names[1000][2][20],name[20];
 int i,n,j;
 cout<<"Please enter the number of names required to be sorted ";
 cin>>n;
 cout <<"\nEnter the names in form \tFirst Name \tLast Name\n";
 for(i=0;i<n;i++)
 {
   cout <<"\t\t\t" <<(i+1) <<"\t";
   cin>>names[i][0] >>names[i][1];
```

```
 }
 system("cls");
 // sort the names in acsending order of last names
 for(i=0;i<n-1;i++)
   for(j=0;j<n-i-1;j++)
      if(strcmp(names[j][1],names[j+1][1])>0)
      { //swap the two names when they are out of order
         strcpy(name,names[j][1]);
         strcpy(names[j][1],names[j+1][1]);
         strcpy(names[j+1][1],name);
         strcpy(name,names[j][0]);
         strcpy(names[j][0],names[j+1][0]);
         strcpy(names[j+1][0],name);
      }
cout<<"\t\tThe sorted names will be as follows:\n";
cout <<"\n\tLast Name\t\tFirst Name\n";
cout <<setw(60); cout.fill('_'); cout<<"_\n";
cout.fill(' ');
 for(i=0;i<n;i++)
 {
    int t1=strlen(names[i][1]);//length of first name
    int t2=strlen(names[i][0]);//length of last name
    for(j=0;j<t1;j++) //convert last name to capitals
      names[i][1][j]=toupper(names[i][1][j]);
    for(j=0;j<t2;j++)//convert first name to capitals
      names[i][0][j]=toupper(names[i][0][j]);
    cout <<"\n\t" <<setw(30) <<left <<names[i][1] <<setw(30) <<names[i][0];
 }
 cout <<endl;
 system("pause");
 return 0;
}
```

Program run: The program run here is using the same input list as above.

```
The sorted names will be as follows:

    Last Name                   First Name
_____________________________________________________

    ACQUAH                  WILLIAM
    ANSAH                     ABENA
    ARTHUR                  JOHN
    ASHONG                  GRACE
```

HARRISON	ANGELA
KOFUA	ESI
KWANSIMA	ARABA
MENSAH	JAMES
PAINTSIL	KWEKU
ZON	GABRIEL

Press any key to continue . . .

Questions and answers

We will at this stage look at some questions on how we can make use of what we have learnt so far in writing programs. Before looking at the solutions, please try the questions on your own first and then compare with the solution provided. Some of the questions may appear too easy when you read them but try them and you will see that you will have some difficulty in getting the answer correct or some are likely to take sometime before you get the logic.

Question 1: Making use of a function, write a program to evaluate the value of $\sin^2 x$ according to the formula

$$\sin^2 x = x^2 - \frac{2^3 x^4}{4!} + \frac{2^5 x^6}{6!} - \frac{2^7 x^8}{8!} + \ldots$$

Your program should also return the number of terms needed to evaluate $\sin^2 x$ to 1 in 1,000,000 together with the value of $\sin^2 x$ for a given x with appropriate captions. You must put in a check so that the program accepts x as input if only the absolute value of x is less than $\frac{\pi}{2}$.

Solution

To be able to solve this question you must first try to write down the general expression. This should be as follows:

$$\sin^2 x = \sum_{1}^{n} \frac{(-1)^{i+1} 2^{2i-1} x^{2i}}{(2i)!}$$

where n is the number of terms required.

Program listing

```
#include<iostream> //needed for cin and cout
#include<cmath> // needed for the pow function
using namespace std;
int fact(int);   //factorial function prototype
int main()
{
  double pi=3.14159;
  int i=1;
```

```
  double sineSqd,x,prev;
  cout<<"Please enter the value of x ";
  cin >> x;
  while (fabs(x)>=pi/2)
  {
    cout<<"Please " <<-pi/2 <<"<=x<=" <<pi/2;
    cin >> x;
  }
  sineSqd=0;
   do
  {
            prev=sineSqd;
            sineSqd=sineSqd+pow(-1,i+1)*pow(2,2*i-1)*pow(x,2*i)/fact(2*i);
            i=i+1;
  } while (fabs(prev-sineSqd)>pow(10,-6));
  cout <<"sin(" <<x <<") is " <<sineSqd;
  cout <<" and the number of terms used is " << i-1;
  cin>>x;
return 0;
}

int fact(int n)
{
 int f=1;
 for(int i=n;i>1;i--)
          f=f*i;
 return f;
}
```

If the above program is run with x = 0.22 the value displayed to the screen is 0.0476242 and the number of terms used is 4. Note that the value returned for x is in radians.

Question 2: You are required to write a C++ program that can display the numbers 0 to 99 inclusive in the following format, that is as a 10 by 10 array:

0	10	20	30	40	50	60	70	80	90
1	11	21	31	41	51	61	71	81	91
2	12	22	32	42	52	62	72	82	92
3	13	23	33	43	53	63	73	83	93
4	14	24	34	44	54	64	74	84	94
5	15	25	35	45	55	65	75	85	95
6	16	26	36	46	56	66	76	86	96
7	17	27	37	47	57	67	77	87	97
8	18	28	38	48	58	68	78	88	98

9 19 29 39 49 59 69 79 89 99

Solution:

```
#include<iostream>
using namespace std;
int main()
{
        int i,j;
        char space[4]="   ";
        for(i=0;i<10;i++)
        {
                for(j=0;j<10;j++)
                    cout<<(i+j*10) <<space;
                cout <<endl;
        }
return 0;
}
```

Question 3: Using at least one user defined function called ConvertBase, write a program that can convert a number from base 10 to any one of the bases 2, 3, 4, 5, 6, 7, 8, 9 and 16.

Solution

```
#include<iostream>
using namespace std;
void ConvertBase(int x, int b);
void DisplayOutput(int x);

int i,yy[100],xx[100],num,base; // These variables are declared as global
int main()
{
        cout <<" Please enter the number in base 10 to convert ";
        cin >>num;
        cout <<"Please enter the base to convert to ";
        cin >> base;
        while ((base<2 || base>9) && base!=16)
       {
         cout <<"Please enter valid base [0-9 or 16] ";
         cin >> base;
        }
        ConvertBase(num,base);
        DisplayOutput(xx[100]);
        return 0;
}
void ConvertBase(int x, int b)
{
        i=0;
```

```
            while (x>0)
            {
              xx[i]=x%b;
              x=x/b;
              i++;
            }
            for(int j=0;j<i;j++)
                        yy[j]=xx[i-j-1];

}

void DisplayOutput(int x)
{
 int j;
 cout<< "The number " <<num <<" in base 10 is equivalent to ";
 switch (base)
 {
 case 2: case 3: case 4: case 5: case 6:
 case 7: case 8: case 9:
            for(j=0;j<i;j++)
            cout <<yy[j];
    break;
 case 16:
            for(j=0;j<i;j++)
              switch (yy[j])
              {
                        case 10:cout <<"A";break;
                        case 11:cout <<"B";break;
                        case 12:cout <<"C";break;
                        case 13:cout <<"D";break;
                        case 14:cout <<"E";break;
                        case 15:cout <<"F";break;
                        default: cout <<yy[j];
              }
 }
 cout <<" in base " << base;
 cin>>base;
}
```

Question 4: Write a program that can accept as input two positive integers; the first being a number to convert to base 10 and the other being the base that the first number is in. Your program should display the equivalent of the number in base 10.

Solution

```
#include<iostream>
using namespace std;
int main()
{
        int n,num,b,base, sum=0;
        cout <<"Please enter the number to convert " ;
        cin >>num;
        cout <<"Please enter the base of the above number ";
        cin >> base;
        b=1;
        n=num;
        cout <<"The number " <<num << " in base " <<base;
       cout <<" is equivalent to ";
        while (n>0)
        {
                sum=sum+n%10*b;
                n=n/10;
                b=b*base;
        }
      cout << sum <<" in base " <<base;
        return 0;
}
```

Assignment: The above program does not do any validation. For example it will accept a number such as 126 when its base is entered as 2. As you know 126 cannot be in base 2. You are therefore required to modify the above program so that once the base is entered for a number, it checks to see if the number is valid. Limit yourself to numbers in base 2 to 9 inclusive. You should put in checks so that numbers entered as input are positive.

Question 5: Newton's method is a one way of finding the location of the zero or the root of a mathematical function, f(x) say. The method works by initially guessing the x location of the zero. The function value at the current x is computed to get a better estimate of the zero. This process is repeated until the difference between two successive function values is at most 0.0000001. Each estimate of xnew is computed from xold as follows:

$$x_{new} = x_{old} - \frac{f(x_{old})}{f'(x_{old})}$$

Where $f'(x_{old})$ is the derivative of the function $f(x_{old})$. Given that $f(x) = 2x^5 + 4x - 72$ write a C++ program that will accept as input an initial guess value, x_{old} and then find the zero of the above function. Do not use inline functions. Your output should be the root of the function and the number of iterations performed to obtain the root.

Program listing

```
#include<iostream>
#include<cmath>
#include <cstdlib>
using namespace std;
double f(double x);
double fdev(double x);
int main()
{
        int cnt=0;
        double xnew,xold;
        cout <<"Please enter initial guess ";
        cin>>xold;
        xnew=xold;
        do
        {
                xold=xnew;
                xnew=xold -f(xold)/fdev(xold);
                cnt++;
        } while(fabs(xold-xnew)>pow(10,-7));
        cout<<"The zero of the function is " <<xnew <<endl;
        cout <<"The number of iterations is " <<cnt <<endl;
    system("pause");
    return 0;
}


double f(double x)
{
        return (2.0*pow(x,5)+4*x-72);
}

double fdev(double x)
{
        return (10.0*pow(x,4)+4);
}
```

Irrespective of the value entered as initial guess, the zero of the function is computed as 2. However, depending on how close the absolute value of the initial guess is to the value 2, the fewer the number of iterations. This is evident in the two program runs below. The bolded numbers are the different initial guesses used for the different program run.

Program run 1

Please enter initial guess **100**

```
The zero of the function is 2
The number of iterations is 22
Press any key to continue . . .
```

Program run 2

```
Please enter initial guess 10000
The zero of the function is 2
The number of iterations is 43
Press any key to continue . . .
```

Question 6: Four thieves stole a large quantity of sardines in cartons from one supermarket at Kejetia market one night. As the operation was conducted late in the night they could not count the number of cartons they had but agreed on sharing the sardines equally early the next morning. The sardines were kept them in a small room and charged their security man to take care of them. Since the thieves did not count the sardines, each of them wanted to cheat the others by stealing some of their own sardines. So, in the middle of the night, one of the thieves woke up, entered the room where they kept their sardines. He first counted all the sardines and realised that if they were to share equally there would be one carton left so he decided to give that to the security man. He then divided the remaining cartons by four and took his share away.

For the remaining sardines, each of the other three thieves also got up in turns and repeated exactly what the first thief had done unknowingly. Early the next morning, when they gathered to share their sardines, they all saw that the quantity has reduced drastically but could not complain. They all counted the remaining sardines and realized that there would be one left after dividing it by four so they agreed on giving one carton to the security man and shared the remaining equally.

The matter was reported to the police and two days later the four thieves were arrested following a tip off. After a lengthy court hearing, the thieves were charged to pay ten million cedis each in addition to the cost of the number of sardines one had in his possession. The problem now is to estimate or calculate the total number of sardines stolen as well as the number each thief had as share.

Assume you are the only programmer with the court and you have been tasked to write a program that would produce a report showing the total number of sardines (including that of the security man) and the share of each thief since this is a case which has not been handled by the court before. The owner of the supermarket claims that the number of cartons stolen would not be less than 5000 and not more than 15,000. Your program should output all possible values between the above range.

Solution: Occasionally, you may meet lengthy questions like this but when you read carefully you will see that they are quite simple. In fact, all that the above question involves is to find all numbers between 5000 and 15000 that meet the following criteria:

(i) the number minus one is divisible by 4 and
(ii) the number is reduced by 1 and then by a quarter
(iii) and that steps (i) and (ii) can be repeated five times.

Program listing:

```
#include<iostream>
#include<iomanip>
using namespace std;
const int thiefs=4;
int main()
{
        int thieves[thiefs+1];
        int i,thief, sardines, sard;
        cout <<"\t\tREPORT ON POSSIBLE NUMBER OF STOLEN SARDINES";
        cout <<"\n\t\t\t AT THE KEJETIA MARKET";
        cout <<"\nNumber \t\t 1st THIEF\t    2nd THIEF\t     3rd THIEF";
        cout <<"\t    4th THIEF";
        cout <<"\n**********************************************";
        cout <<"**********************************\n";
        for(sardines=5000;sardines<15001;sardines++)
        {   sard=sardines;i=0;
                for(thief=0;thief<=thiefs;thief++)
//check if number of sardines is divisible by 4 after taking one out
                 if((sard-1)%thiefs==0)
                 {   i++;
                        thieves[thief]=(sard-1)/thiefs;
                        sard=sard-thieves[thief]-1;
                 }
                 else
break;
/* the break means if the quantity of sardine is not divisible by 4 after deducting one then consider
next quantity of sardines by  leaving the for thief loop.*/

/*test whether for a particular quantity of sardines all thieves were able to have their share and if
yes display the share of each thief and the quantity */
                 if(i>thiefs)
                 {
                        cout <<sardines <<"\t";
                        for(int j=0;j<thiefs;j++)
                          cout <<"\t\t" <<thieves[j]+thieves[thiefs];
```

```
                    cout <<endl;
                }
            }
            system("pause");
            return 0;
        }
```

Program Output:

```
        REPORT ON POSSIBLE NUMBER OF STOLEN SARDINES
             AT THE KEJETIA MARKET
Number      1st THIEF   2nd THIEF   3rd THIEF   4th THIEF
*****************************************************************
5117            1683        1363        1123        943
6141            2020        1636        1348        1132
7165            2357        1909        1573        1321
8189            2694        2182        1798        1510
9213            3031        2455        2023        1699
10237           3368        2728        2248        1888
11261           3705        3001        2473        2077
12285           4042        3274        2698        2266
13309           4379        3547        2923        2455
14333           4716        3820        3148        2644
```

Assignment: Modify the above program so that a user has the chance of entering the number of thieves from the keyboard. Note that you may have to take into account the output so that you can have your output in the required format and with all the needed information.

Question 7: Write a program that first accept as input two matrices of size n by n and an arithmetic operator (+, - and *) that specifies the operation to be performed on the two matrices. For example, if the operator is * then the two matrices are to be multiplied. Make use of two functions; one to get the input matrices and the other to compute the new matrix based on the required operation. Your output should be the input matrices together with the resultant matrix and the operation in a format such as follows:

```
1  1  1      2  4  4      3  5  5
2  2  1  +   2  3  1  =   4  5  2
3  4  1      4  4  0      7  8  1
```

Program listing:

```
#include <iostream>
```

```
#include <cstdlib>
#include <iomanip>
using namespace std;
void getInput(int x[][10],int y[][10]);
void Compute(int x[][10],int y[][10],int z[][10],char op);
int r,c; //r and c are no of row and columns respectively
char op;
int x[10][10],y[10][10], z[10][10];
int main()
{
     getInput(x,y);
     Compute(x,y,z,op);
     system("PAUSE");
     return 0;
}
void getInput(int x[][10],int y[][10])
{
 int i, j;
 cout <<"Enter the number of rows and columns of Matrix ";
 cin>>r >>c;
 do
 {
   cout <<"Enter matrix operation (+,-,*) ";  cin>>op;
 } while (op<42||op>45);
 for(i=0;i<r;i++)
   for(j=0;j<c;j++)
   {
    cout<<"Enter Matrix A[" <<i<<"][" <<j <<"] :";
    cin>>x[i][j];
   }
 for(i=0;i<r;i++)
   for(j=0;j<c;j++)
   {
    cout<<"Enter Matrix A[" <<i<<"][" <<j <<"] :";
    cin>>y[i][j];
   }
}

void Compute(int x[][10], int y[][10], int z[][10],char operation)
{
 int i,j,k;
 switch(operation)
 {
  case '-': for(i=0;i<r;i++)
```

```
        {
          for(j=0;j<c;j++)
            cout <<setw(7) <<x[i][j];
          cout<<" |";
          if(i==r/2) cout <<"  " <<op; else cout<<"   ";
          for(j=0;j<c;j++)
             cout<<setw(7)<<y[i][j];
          cout<<" |";
          if(i==r/2) cout <<"  ="; else cout<<"   ";
          for(j=0;j<c;j++)
            cout<<setw(7)<<(x[i][j]-y[i][j]);
         cout <<endl;
        }
        break;
case '+': for(i=0;i<r;i++)
        {
          for(j=0;j<c;j++)
            cout<<setw(7)<<x[i][j];
          cout<<" |";
          if(i==r/2) cout <<"  " <<op; else cout<<"   ";
          for(j=0;j<c;j++)
             cout<<setw(7)<<y[i][j];
          cout<<" |";
          if(i==r/2) cout <<"  ="; else cout<<"   ";
          for(j=0;j<c;j++)
            cout<<setw(7)<<(x[i][j]+y[i][j]);
         cout <<endl;
        }
        break;
case '*':
       for(i=0;i<r;i++)
         for(j=0;j<c;j++)
         {
          z[i][j]=0;
          for(int k=0;k<r;k++)
           z[i][j]=z[i][j]+x[i][k]*y[k][j];
         }
       for(i=0;i<r;i++)
         {
          for(j=0;j<c;j++)
            cout <<setw(7) <<x[i][j];
            cout<<" |";
          if(i==r/2) cout <<"  " <<op; else cout<<"   ";
          for(j=0;j<c;j++)
```

```
            cout <<setw(7)<<y[i][j];
          cout<<" |";
          if(i==r/2) cout <<"  ="; else cout<<"   ";
          for(j=0;j<c;j++)
            cout<<setw(7)<<z[i][j];
         cout <<endl;
        }
    }// end of switch statement
} // end of function
```

Program run:

```
Enter the number of rows and columns of Matrix 3 3
Enter matrix operation (+,-,*) *
Enter Matrix A[0][0] :1
Enter Matrix A[0][1] :2
Enter Matrix A[0][2] :3
Enter Matrix A[1][0] :2
Enter Matrix A[1][1] :1
Enter Matrix A[1][2] :5
Enter Matrix A[2][0] :2
Enter Matrix A[2][1] :1
Enter Matrix A[2][2] :1
Enter Matrix A[0][0] :2
Enter Matrix A[0][1] :0
Enter Matrix A[0][2] :1
Enter Matrix A[1][0] :2
Enter Matrix A[1][1] :3
Enter Matrix A[1][2] :1
Enter Matrix A[2][0] :2
Enter Matrix A[2][1] :3
Enter Matrix A[2][2] :1

    1     2     3 |       2     0     1 |       12    15     6
    2     1     5 |  *    2     3     1 |  =    16    18     8
    2     1     1 |       2     3     1 |        8     6     4
Press any key to continue . . .
```

Assignment: Modify the above program such that it will accept arrays of size *n* by *m* where *n* and *m* are the number of rows and number of columns in an array. Checks should be made that the array operation is possible. For example two matrices can be multiplied only if the number of

columns in the first matrix is the same as the number of rows in the second matrix. Where an operation cannot be performed an appropriate error message should be displayed.

Question 8: A magic square is a square matrix of size *n x n* where *n* is an odd integer value, and that each of the integers 1, 2, 3, … n^2 appears exactly once. The sum of each column and that of each row as well as that of the diagonal must be equal. For example, given a 5 x 5 magic square, the sum of each row, each column and each diagonal must be equal to 65 and 175 for a 7 x 7 magic square. The procedure for filling a magic square matrix is as follows:

Let K be 1 and then place the value of K at the centre of the first row.

Move up one row and one column to the right and place the value of K+1 there. Repeat this step until one of the following situations arise:

(i) If a move takes you to a row above the first row of the jth column then move to the bottom of the jth column and insert the value of K+1 there.

(ii) If a move takes you to a column to the right of the rightmost column of the ith row then move to the first column of the ith row and insert value of K+1 there.

(iii) If a move takes you to where the square has already been occupied or where the move will take you above first row and the right of the rightmost column, place the value of K+1 immediately below the value of K.

Write a program that can accept any odd number as matrix size and output the corresponding magic square. Use user defined functions where possible.

Program listing: We will limit this program to 1000 x 1000 but then changing the 1000 to any higher odd number will enable you to generate the magic square matrix for higher sizes. Please note that to run this program or your solution with a high matrix size, it will be advisable to write the output to a file as you can have a limited number of data values displayed to the screen per line.

```
#include <iostream>
#include <cstdlib>
#include <iomanip>
using namespace std;
void initilize(void);// to set matrix elements to zeros
void MagicSquare(void);//to fill the matrix with correct values
void DisplaySquare(void);//to display the magic square elments
int x[1000][1000],n;
int main()
{
 cout <<"Please enter N for the number of rows and columns ";
do //repeat this until the user enters an odd number
{
    cout <<"\nPlease note that N should be an odd number ";
    cin >>n;
} while (n%2==0);
```

```
 MagicSquare();
 DisplaySquare();
 cout <<endl;
 system("PAUSE");
 return 0;
}

void MagicSquare(void)
{
  int row,col,k,lrow,lcol;
   row=0; col=n/2;
  x[row][col]=1;
  for(k=2;k<=n*n;k++)
  {
   lrow=row;
  lcol=col;
   col = col+1;
   row = row-1;
   if (row<0) row=n-1;
   if (col>n-1) col=0;
   if ((x[row][col]!=0)||(col>n-1&&row<0))
      {
        row=lrow+1;
        col=lcol;
      } // end of the if ((x[row][col]!=0)||(col>n-1&&row<0)) test
   x[row][col]=k;
  } // end of the for k loop
}// end of the MagicSquare function

void initilize(void)
{
 for(int i=0;i<n;i++)
  for(int j=0;j<n;j++)
    x[i][j]=0;
}

void DisplaySquare(void)
{
   for(int i=0;i<n;i++)
   {
   for(int j=0;j<n;j++)
     cout <<setw(4) <<x[i][j];
   cout<<endl;
   }
```

```
  cout <<endl;
}
```

Program run 1:

```
Please enter N for the number of rows and columns
Please note that N should be an odd number 5
 17 24  1  8 15
 23  5  7 14 16
  4  6 13 20 22
 10 12 19 21  3
 11 18 25  2  9

Press any key to continue . . .
```

Program run 2:

```
Please enter N for the number of rows and columns
Please note that N should be an odd number 7
 30 39 48  1 10 19 28
 38 47  7  9 18 27 29
 46  6  8 17 26 35 37
  5 14 16 25 34 36 45
 13 15 24 33 42 44  4
 21 23 32 41 43  3 12
 22 31 40 49  2 11 20

Press any key to continue . . .
```

Program run 3:

```
Please enter N for the number of rows and columns
Please note that N should be an odd number 15
 122 139 156 173 190 207 224   1  18  35  52  69  86 103 120
 138 155 172 189 206 223  15  17  34  51  68  85 102 119 121
 154 171 188 205 222  14  16  33  50  67  84 101 118 135 137
 170 187 204 221  13  30  32  49  66  83 100 117 134 136 153
 186 203 220  12  29  31  48  65  82  99 116 133 150 152 169
 202 219  11  28  45  47  64  81  98 115 132 149 151 168 185
 218  10  27  44  46  63  80  97 114 131 148 165 167 184 201
   9  26  43  60  62  79  96 113 130 147 164 166 183 200 217
  25  42  59  61  78  95 112 129 146 163 180 182 199 216   8
  41  58  75  77  94 111 128 145 162 179 181 198 215   7  24
  57  74  76  93 110 127 144 161 178 195 197 214   6  23  40
```

```
 73  90  92 109 126 143 160 177 194 196 213   5  22  39  56
 89  91 108 125 142 159 176 193 210 212   4  21  38  55  72
105 107 124 141 158 175 192 209 211   3  20  37  54  71  88
106 123 140 157 174 191 208 225   2  19  36  53  70  87 104
```

Press any key to continue . . .

Question 9

Without using any existing formula, write a program that accepts as input an integer, n (n>1) representing the degree of a polynomial and output all the Binomial coefficients for degrees 0 through to n in a Pascal Triangle form. For example, if n is entered as 4 then your output should be as follows:

```
                1
            1       1
        1       2       1
    1       3       3       1
1       4       6       4       1
```

You may use the following algorithm as a guide

(i) Create an array of size of n+1 by n+1 (where n is degree of polynomial)

(ii) Store at column n+1 of row 1 the value 1

(iii) Move one row down and 1 column to the left, find the sum of the numbers to the top left and the top right of current position and store the sum at the current position. Example for 1st item of row two, we will sum items at row 1 column 4 and row 1 column 6.

(iv) Move two columns to the left and repeat step (iii).

(v) Steps (iii) and (iv) should be repeated until the required number of coefficients has been generated. The required number of coefficient for m is m.

(vi) Repeat steps (ii) to (iv) until row n is completed.

Program listing

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
  int x[100][100],n,row,col,r;
//check that the degree is at least 1
```

```
  do
  {
     cout <<"Please enter the degree of the polynomial ";
     cin>>n;
  } while (n<1);
  n=n+1;
//initiliasing array elemnts to zeros
 for (int i=0;i<n;i++)
  for(int j=0;j<n*2+1;j++)
    x[i][j]=0;
 x[0][n]=1;
//filling the array with appropriate values
 for (row=1;row<n;row++)
 {
    col=n-row;
    for(r=1;r<=row+1;r++)
    {
       x[row][col]=x[row-1][col-1]+x[row-1][col+1];
       col=col+2;
    }
 }
//diplay the pascal triangle
 for (int i=0;i<n;i++)
 {
  for(int j=0;j<n*2+1;j++)
   if(x[i][j]==0)
     cout <<setw(4) <<" ";
   else
      cout <<setw(4) <<x[i][j];
  cout <<endl;
 }
   system("pause");
   return 0;
}
```

Program run

Please enter the degree of the polynomial 9

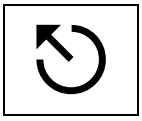
```
                    1
                 1      1
              1      2      1
           1      3      3      1
        1      4      6      4      1
     1      5     10     10      5      1
```

1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1

Self Assessment 2-4

1. Write a void function that takes one-dimensional array and the size of the array as arguments and then initialise the array elements to zeros.

Learning Track Activities



Unit Summary

1. We learnt in this Unit about functions. Functions are very important as they help us to break a given problem down into smaller modules and write them as functions and test them separately. This makes it easy to detect logical errors easily. There are a number of ways of classifying functions and notable among these are (1) the one who wrote the functions, (2) the way the functions are called and (3) whether a function returns a value when called or not.. In classifying functions as to who wrote them, they can be classified into pre-defined and programmer defined functions. We can also classify them as inline or not. The pre-defined functions are those that were written by those who developed the C++ compiler and made them part of the compiler. The programmer defined functions are those written by a programmer like you so that you can use them in your program. When functions are classified by the way they are called then we two types namely call-by-value and call-by-reference functions. Call-by-value functions can also make a copy of the value they receive and work with them. They may or may not change the value received but if they do the changes are only temporal within the function that made the change where as for a call-by-reference, when they make a change to the value the new value replaces the old value in the memory location where the original value was kept and therefore any

reference to the memory location will refer to the new value. With respect to classification by whether a function returns a value or not when called, we have functions that do return a value when called and those that do not. For those that do not, the data type of the value to be returned is indicated as void whereas if the function was to return a value the data type of the value returned will be specified as one of the C++ data types such as int, double, bool, etc.

2. In this same unit we have learnt about arrays. Arrays are similar to identifiers just that identifiers usually hold one value at a time while an array can be used in holding a number of data items of the same type at any given time. Arrays are best used for data items (especially inputs) that we intend to refer to them more than once in a program otherwise the same data will have to be entered more than one. Let us consider our standard deviation problem again. You will realise that to calculate the required standard deviation value we will have to do the following:

(i) sum all the **given numbers**

(ii) divide the sum by the total number of numbers given to obtain their mean

(iii) find the sum of the square of the difference between each of the **given number** and their mean

(iv)divide the sum in (iii) by the total number of numbers given and take the square root.

We can see clearly from the above that the given numbers are to be referenced in (i) and (iii). As such, if they are kept in arrays during (i) it will not be necessary to accept the same numbers as input again from the keyboard or be read from a file. In declaring an array, we need to have it a name and also let the computer know the number of storage locations that should be reserved for it. The array elements are referenced using a subscript.

Key terms/ New Words in Unit

1. [State key term/New Words: explain what it means]

2. [State key term/New Words: explain what it means]

3. [State key term/New Words: explain what it means]

4. [State key term/New Words: explain what it means]

Unit Assignments 4

[insert here details of self-assessment]