Learning to Work Efficiently and Accurately
What is a function and what its parts are.
How to declare and define functions.
How to pass parameters into functions
How to return a value from a function.

What is a function?
- A function is a subprogram that can act on data and return a value.
- Every C++ program has at least one function that is `main()`.
- `main()` is called automatically when the program starts.
- `main()` might call other functions, some of which might call still others.
- Each function has its own name, and when that name is encountered, the execution of the program branches to the body of that function. When the function returns, execution resumes on the next line of the calling function.
- Well designed functions perform a specific and easily understood task.
- Complicated tasks should be broken down into multiple functions, and then each can be called in turn.
- Functions come in two varieties:
  - user-defined
  - built-in is in part of the compiler
DECLARING and DEFINING Function.

- Using functions in the program requires
  - declare the function
  - define the function
- The declaration tells the compiler the RETURN TYPE, NAME, and PARAMETERS of the function.
- The definition tells the compiler how the function works.
- No function can be called from any other function that hasn’t first been declared.
- The declaration of a function is called its PROTOTYPE.
- There are two ways to declare a function:
  - Write your prototype into a file, and then use the `#include` directive to include it in your program.
  - Write the prototype into the file in which your function is used
Function Prototypes

- The function prototype is a statement, which means it ends with semicolon [ ; ]
- It consists of the function’s:
  - Return type
  - Name
  - Parameter type, parameter name
- Example: `unsigned short int FindArea( int length, int width );`
- Function prototype does not need to contain the name of the parameters
- Example: `unsigned short int FindArea( int , int );`
- Each parameter type must be separated by commas [ , ]
- Adding parameter names makes your prototype clearer.
- Note that all functions have a return type.
- If none is explicitly stated, the return type defaults to `int`. 
Function Definition

- The definition of a function consists of:
  - Function header or declarator
  - Function's body
- The header is exactly like the function prototype, except that the parameters must be named. There is no terminating semicolon.
- The body of the function is a set of statements enclosed in braces \[ \{ \} \]
- If a function returns a value, it should ended with a return statement.
- Return statements can legally appear anywhere in the body of the function.
- Every function has return type, if one is not explicitly designated, the return type will be int.
- If a function does not return a value, its return type will be void.
- A function can take any number of parameters or no parameter at all.
- A function can written to return a value, but it not mandatory that a function return a value.
- If a function has return type of void, it can not return a value.
- Function can take any number of parameters but can return only one value.
- Variables can be passed to the function by value, by pointer or by reference.
// function program example
// A C++ program to calculate an area
#include <iostream.h>
int Area( int l, int w ); // function prototype

int length=9;
int width=4;
int X;
X = Area( length, width ); // function call

// Function definition
int Area( int l, int w )
{
    int z;
    z = l * w ;
    return z ;
}

cout << " Area is:  " << X << endl ;

// end main ( )
Local variables
- Local variables that are declared within the body of the function
- Local variables exit only locally within the function itself.
- When the function return the local variables are no longer variable.
- The parameters passed into the function are also considered local variables and can be used exactly as if they has been defined within the body of the function.
- Variables declared within the function are said to have “local scope” that means they are visible and usable only within the function which they are defined.

Global variables
- Global variables that are defined outside of any function have global scope and thus are available from any function in the program, including `main()`.
- A local variable with the same name as a global variables do not change the global variables (since different location in the memory)
- If a function has a variable with the same name as a global variable, the name refers to the local variable – not the global when used within the function.

Global variables: a word of caution
- In C++, global variables are legal but they are almost never used.
- Global variables are dangerous, because they are shared data and one function can change a global variable in a way that is invisible to another function.
Using functions as parameters to functions

- You have the functions `double()`, `triple()`, `square()` and `cube()` each of which return a value. You could write

  \[
  \text{Answer} = (\text{double}\left(\text{triple}\left(\text{square}\left(\text{cube}(\text{myvalue})\right)\right)\right)) ;
  \]

- This statement takes a variable `myvalue` and passes as an argument to the function `cube()`, whose return value is passed as an argument to the function `square()`, whose return value is in turn passed to `triple()`, and that return value is passed to `double()`. The return value of this doubled, tripled, squared and cubed number is now passed to `answer`.

Return Values

- See the following return value from the functions
  - `return 5;`
  - `return ( x > 5 );` // this will return 0 if x is not greater than 5 otherwise return 1
    // will return the value of the expression not the value of x
  - `return ( \text{Myfunction}() ) ;`

Default parameters

- Example: `long myfunction( \text{int} \text{length}, \text{int} \text{width}=25, \text{int} \text{height}=1 ) ;`
- Above example if the calling function did not include a parameter, the compiler would take the default values.
Overloading function
- C++ enables to create more than one function with the same name, this is called function overloading and the function must differ in their parameter list with different type of parameter.
- Example: `int Double( int );
  long Double( long );
  float Double( float, float );
  double Double( double, int );`
- The return type of overloading function can be the same or difference
- Function overloading is also called function polymorphism.
- Poly means many and morph means form: a polymorphic function is many-formed.

Inline function
- Uses for small size of the function
- Increases program speed
- Increases size of the program (more memory needed)
- Function declare with the keyword `inline`
- Example: `inline int Double( int );`