**Function:**

As an alternative, instead of performing all your assignments in **main(),** you can divide the code in smaller sections of code so that each section is meant to perform a specify task. Other sections of the program can then refer to that section for whatever it has to provide. Such a section of code is called a **function**.

To create a function, there are some rules you must follow. The fundamental formula is:

*ReturnType FunctionName*(parameters list) { }

Function Rules:

1. The first rule is that the function must be created outside of any other function. This means that you cannot create a function inside of **main()**.
2. The *ReturnType* is the kind of value that the function will produce. Some functions don't produce a (specific) result. In this case, the *ReturnType* can be substituted with the **void** keyword.
3. Like a variable, a function must have a name. The name follows the rules we have applied to variables so far. Because a function usually performs an action or a task, its name should reflect a **verb**. Examples of function names are **Show** or **Display**. In some cases, the name will be made of more than one word. You should still make sure that the name reflects an assignment.
4. The name of a function must be followed by parentheses. parameters (as many as needed): Each parameter consists of a data type specified followed by an identifier (variable name), like any regular variable declaration (for example: int x) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.
5. The parentheses of a function must be followed by an opening curly bracket "{" and a closing curly bracket "}". The section inside of the curly brackets is referred to as the body of the function. In this body, you perform the assignment of the function. For example, a function can be used to simply display a sentence.

Here is an example:

#include <iostream>

using namespace std;

//==================

int Func1 (); // Function prototype

void Func2(int i); // Function prototype

//==========================================

int main()

{

int a;

a= Func1(); // calling Func1

Func2(a); // calling Func2

return 0;

}

//=====================================

int Func1() // func1 declaration

{ int num;

cout << "enter a number";

cin>> num;

return num;

}

//=====================================

void Func2(int i) // func2 declaration

{

i = i \* i;

cout << i;

}

## Recursion Examples

void recurse()

{

recurse(); //Function calls itself

}

Consider this function. 

#include <iostream>

using namespace std;

void myMethod( int counter)

{

if(counter == 0)

return;

else

{

cout<<counter<<endl;

myMethod(--counter);

return;

}

}

int main()

{

int input;

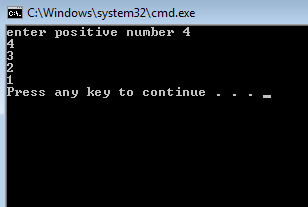
cout<<"enter positive number ";

cin>>input;

myMethod(input);

return 0;

}

This recursion is not infinite, assuming the method is passed a positive integer value. What will the output be?