Function – Notes

INTRODUCTION -A function is a subprogram that acts on data and often returns a value. A program written with numerous functions is easier to maintain, update and debug than one very long program. By programming in a modular (functional) fashion, several programmers can work independently on separate functions which can be assembled at a later date to create the entire project. Each function has its own name. When that name is encountered in a program, the execution of the program branches to the body of that function. When the function is finished, execution returns to the area of the program code from which it was called, and the program continues on to the next line of code.

Types of Functions

1. **Built-in Functions:** Build-in functions are part of the compiler package, such as exit(1), sqrt() etc.	2. **User-defined functions :** User-defined functions are created by the programmer as per requirements of your program.

Creating User-Defined Functions

Declare the function.

The declaration, called the FUNCTION PROTOTYPE, informs the compiler about the functions to be used in a program, the argument they take and the type of value they return.

Define the function.

The function definition tells the compiler what task the function will be performing. The function prototype and the function definition must be same on the return type, the name, and the parameters. The only difference between the function prototype and the function header is a semicolon.
The function definition consists of the function header and its body. The header is EXACTLY like the function prototype, EXCEPT that it contains NO terminating semicolon.

```
//Prototyping, defining and calling a function
#include <iostream.h>
void starline();      // prototype the function
int main()
{     starline( );     // function call
    cout<< "\t\tBjarne Stroustrup\n";
    starline( );    // function call
    return 0;
}
// function definition
void starline()
{
    int count;        // declaring a LOCAL variable
    for(count = 1; count <=65; count++)
        cout<< "*";
```

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

ARGUMENT TO A FUNCTION

Sometimes the calling function supplies some values to the called function. These are known as parameters. The variables which supply the values to a calling function called **actual parameters**. The variable which receive the value from called statement are termed **formal parameters**.

Consider the following example that evaluates the area of a circle.

```
#include<iostream.h>
void area(float);
int main()
{    float radius;
   cin>>radius;
   area(radius);
   return 0;  }
void area(float r)
{    cout<< "the area of the circle is"<<3.14*r*r<<"\n"; }
```

Here radius is called **actual parameter** and r is called **formal parameter**.

CALLING OF FUNCTION

The function can be called using either of the following methods:

1. CALL BY VALUE

In call by value method, the called function creates its own copies of original values sent to it. Any changes, that are made, occur on the function's copy of values and are not reflected back to the calling function.

2. CALL BY REFERENCE

In call be reference method, the called function accesses and works with the original values using their references. Any changes, that occur, take place on the original values are reflected back to the calling code.

Consider the following program which will swap the value of two variables.

using call by reference	**using call by value**

#include<iostream.h>
void swap(int &, int &);
int main()
{
int a=10,b=20;
swap(a,b);
	#include<iostream.h>
	void swap(int , int);
	int main()
	{
	int a=10,b=20;
	swap(a,b);

```
  cout<<a<<" "<<b;
  return 0;
}
void swap(int &c, int &d)
 { int t;
  t=c;
  c=d;
  d=t;
}
```

output:
20 10

```
  cout<<a<<" "<< b;
  return 0;
}
void swap(int c, int d)
{  int t;
  t=c;
  c=d;
  d=t;
}
```

output:
10 20

Function With Default Arguments

C++ allows to call a function without specifying all its arguments. In such cases, the function assigns a default value to a parameter which does not have a mathching arguments in the function call. Default values are specified when the function is declared. The complier knows from the prototype how many arguments a function uses for calling. Example :float result(int marks1, int marks2, int marks3=75);a subsequent function call average = result(60,70);passes the value 60 to marks1, 70 to marks2 and lets the function use default value of 75 for marks3.
The function callaverage = result(60,70,80);passes the value 80 to marks3.

Note: always give default values from right to left.

Constant Arguments

By constant arguments, it is meant that the function cannot modify these arguments. If you pass constant values to the function, then the function cannot modify the values as the values are constants.

Length("String");

RETURN TYPE OF A FUNCTION

// Example program

```
#include <iostream.h>

int timesTwo(int num);   // function prototype
int main()
{
    int number, response;
    cout<<"Please enter a number:";
    cin>>number;
    response = timesTwo(number);  //function call
```

```
    cout<< "The answer is "<<response;
    return 0;  }
//timesTwo function
int timesTwo (int num)
{
    int answer;   //local variable
    answer = 2 * num;
    return (answer);  }
```

Returning (By) Reference

```
float & min(float &a, float &b)
{ if(a<b) return a;
else
return b;
}
```

The return type of above function is float & i.e. the function returns a reference to float type of variable. The above function returns reference to a or b rather than returning values. This means the function call can appear on the left hand side of an assignment statement. That is ,

min(x,y)= -5; is perfectly assigns -5 to the lesser of the two.

Styles of the Functions (with example)

Style 1: void functionName(void)

```
//Example program
//Screen display shown at the right
//Prototyping, defining and calling a function

#include<iostream.h>
#include<stdlib.h>

void astericks(void);   //function prototype

int main(void)
{    system("CLS");
    cout<<"Heads up, function!\n";
    astericks( );     //function call
    cout<<"Again, function!\n";
    astericks( );    //function call
    cout<<"Job well done!\n";
    return 0;    //main( ) is over - ALL STOP!!
}

//function definition
void astericks(void)
{
```

SCREEN DISPLAY

```
Heads up, function!
**********
Again, function!
**********
Job well done!
```

```
    int count;    // declaring LOCAL variable
    for(count = 1; count<=10; count++)
        cout<<"*";

    cout<<endl;
    return;   //return value is VOID, no return
}
```

Style 2: void functionName(argument(s))

```
//Example program
//Screen display shown at the right
//Passing arguments to a function

#include<iostream.h>
#include<stdlib.h>

void greeting(int x);    //function prototype

int main(void)
{
    system("CLS");
    greeting(5);   //function call- argument 5
    int number;
    do
    {          cout<<"Please enter value(1-10):\n ";
        cin>>number;      }
    while ((number < 1) || (number > 10));
    greeting(number); //argument is a variable
    return 0; }

//function definition
void greeting(int x)    // formal argument is x
{
    int i;   // declaring LOCAL variable
    for(i = 0; i < x; i++)
    {          cout<<"Hi ";        }
    cout<<endl;
    return;   //return value is VOID, no return
}
```

Screen Display

```
Hi Hi Hi Hi Hi

Please enter value(1-10):
4
Hi Hi Hi Hi
```

Style 3: non-void functionName(void)

Function calls that return values need to have a place in main for the returned value to reside, to be printed, or to be manipulated.

Always be sure to do something with a returned value! Store the return value somewhere; print the return value to the screen; use the return value in a calculation; just use it!! The following are examples of USES of function return values:

(Function calls with return locations)
x = Avg(i, j) (Function Avg takes i and j and returns a value stored in x.)

cout<< num(1,3,5); (Function num takes 1, 3 and 5 and its return value is printed)

ans=add(a,b) + add(c,d); (Function add is called twice. The return values from 2 calls are added and stored in ans.)

```
//Example program
//Screen display shown at the right
//Returning values from a function

#include<iostream.h>
#include<stdlib.h>

void greeting(int x);    //prototype-no return
int getNumber(void);   //prototype-return integer

int main(void)
{
    system("CLS");
    greeting(5);   //function call
    int response;
    response=getNumber( ); //call and store value
    greeting(response); //function call
    return 0; }

//function definition - creates a line of "Hi"s
void greeting(int x)
{
    int i;
    for(i = 0; i < x; i++)
    {        cout<<"Hi ";      }
    cout<<endl;
    return;   }

//function definition - get the user's # and return it
int getNumber(void)
{      int number;
    do
    {          cout<<"Please enter number (1-10)\n";
       cin>>number;
    }
```

Screen Display

```
Hi Hi Hi Hi Hi
Please enter number (1-10)
3
Hi Hi Hi
```

Equivalent:

return 0;

return (0);

```
    while((number < 1)||(number > 10));
    return (number);  }
```

Style 4: non-void functionName(arguments).

```
//Example program with driver and function
//Screen display shown at the right

//A minimum value function

#include<iostream.h>
#include<stdlib.h>

int min(int x, int y);    //function prototype

int main(void)     // driver program
{
    system("CLS");
    int test1 = 12, test2 = 10;  //arbitrary test values
    cout<<"The minimum of "<< test1 << " and " << test2
        << " is " << min(test1, test2) << ".\n";
     return 0;  }

//function definition
int min(int x, int y)
{    int minimum;
    if (x < y)
        minimum = x;
    else
        minimum = y;
    return (minimum);
}
```

Screen Display:

The minimum of 12 and 10 is 10.

Alternative Function Code (uses conditional operator)

```
int min(int x, int y)
{
  return((x<y)? x : y);
```

Alternative Function Code (uses multiple returns)

```
int min(int x, int y)
{
  if (x < y)
     return x;
  else
      return y;
}
```

Alternative codings for this function appear at the right. Remember, there are always many ways to accomplish a task.

Solved Answers

Q1. Differentiate between Actual Parameter and Formal Parameter. Give suitable example.

Ans. Actual Parameter

A parameter that is used in the function call to send the actual values to the function is known as actual parameter.

Formal Parameter

A parameter that is used in the function definition to receive the values from actual parameter is known as formal parameter.

Example:

```
void Square(int A)//A is formal parameter {
   cout<<2*A<<endl;
}
void main () {
   int N=4;
   Square(N);//N is actual parameter
}
```

Q2. **Differentiate between a Call by Value and Call by Reference, giving suitable examples of each.**

Ans. Call by Value
In call by value, actual parameter and formal parameter have different memory locations, so the changes done in the formal parameter are not reflected back in the actual parameter.

Call By reference
In call by reference, actual parameter and formal parameter share the same memory location, so the changes done in the formal parameter are reflected back in the actual parameter. Requires & sign in formal parameter. Example:

```
void Calc(float Sal,float &Itax)
{          //Sal - Call by value, Itax - Call by reference
          Sal=1.1*Sal;
        Itax=0.3*Sal; }
```

Q3. What do you understand by Default Parameters / arguments ?
Ans . Default Parameter
It is used to provide a default value to a parameter. If no value is sent from the actual parameter, the formal parameter automatically gets this default value. The default parameter cannot be referenced and cannot be placed before a non-default parameter.
Example:

```
void PrintLine(int N=20)  // default parameter
{
     for (int C=0;C<N;C++) cout<<"-";
}
void main () {
                              PrintLine(40) ; PrintLine () ;
}
```

Q5. What is difference between Function Prototype and Function definition ?
Ans.A **function prototype** in C++ is a declaration of a function that does not require the function body but does specify the function's name, parameter types and return type. While a **function definition** specifies what a function does, a function prototype can be thought of as specifying its interface. In the function prototype, argument names are optional, however, the type is necessary along with & or [] (if required). Example:

```
void Disp(char []);
void main () {
   Disp("Hello");
}
void Disp(char Msg[]) {
   cout<<Msg<<endl;  }
```

Q6. Differentiate between Global Variable and Local Variable.
Ans. A variable, which is declared outside all the functions in the program, is known as global variable. A global variable can be accessed and modified in any part of the program (i.e. in any function). If local variable carries identical name as global variable, to access the global variable scope resolution operator (::) is required. **Local Variable**
A variable, which is declared inside a function or a compound statement in the program, is known as local variable. A local variable can be accessed and modified in the function or the compound statement in which it is declared. Example:

```
int Num1=100,Num2=200;//Global Variables
void main () {
   int      Num2=20,Num3=30;//Local      Variables
   Num1+=10;Num2+=20;::Num2+=30;Num3+=40;
   cout<<Num1<<Num2<<::Num2<<Num3<<endl;//1
   104230240
}
```

Q7. Differentiate between Global Prototype and Local Prototype.

Ans. If a function prototype is placed in the body of another function, it is local prototype and the function is locally available to the function which declares it.

If a function prototype is placed outside all the functions,It is global prototype and the function is globally available to all the functions.

```
void line(int n=5);  // global prototype
void main( )
{
void box( ); // local prototype
.
.
. }
```

Explain the Output

#include<iostream.h>
void Indirect(int Temp=20)
{
for(int I=10;I<=Temp;I+=5)
cout<<I<<",";
cout<<endl;
}
void Direct(int &Num)
{
Num+=10;
Indirect(Num);
}
void main( )
{
int Number=20;
Direct(Number);
Indirect( );
cout<<.Number =.<<Number<<endl;
}
Answer:
10,15,20,25,30
10,15,20
Number = 30
	#include<iostream.h>
	int Execute(int M)
	{
	if(M%3==0)
	return M*3;
	else
	return M+10;
	}
	void Output(int B=2)
	{
	for(int T=0;T<B;T++)
	cout<<Execute(T)<<"*";
	cout<<endl;
	}
	void main( )
	{ Output(4);
	Output( );
	Output(3); }
	Answer:
	0*11*12*9*
	0*11*
	0*11*12*

Errors Correction:	
#include<iostream.h> void main( ) {First = 10, Second = 20; Jumpto(First;Second);	#include<iostream.h> **void Jumpto(int N1, int N2 = 20);** void main( ) {**int First = 10, Second = 20;**

Jumpto(Second); } void Jumpto(int N1, int N2 = 20) { N1=N1+N2; count<<N1>>N2; }	**Jumpto(First,Second);** Jumpto(Second); } void Jumpto(int N1, int N2 = 20) { N1=N1+N2; **cout<<N1<<N2;** }

#include<iostream.h>
const int Multiple 3;
void main( )
{ value = 15;
for(int Counter = 1;Counter = <5;Counter
++, Value -= 2)
if(Value%Multiple = = 0)
cout<<Value * Multiple;
cout<<endl;
else cout<<Value + Multiple <<endl; }
	#include<iostream.h>
	#include<iomanip.h>
	const int Multiple= 3;
	void main( )
	{ **int value = 15;**
	for(int Counter = 1;**Counter <=5**;Counter
	++, Value -= 2)
	if(Value%Multiple = = 0)
	{cout<<Value * Multiple;
	cout<<endl;}
	else cout<<Value + Multiple <<endl; }

#include [iostream.h]
void main
{
Present=25, past=35;
Assign(present ; past);
Assign(past);
}
void Assign (int default1 ,default2=30);
{ default1=default1+ default2;
cout<<default1**>>** default2;}
	#include **<iostream.h>**
	void Assign (int default1 ,default2=30);
	void **main()**
	{ **int Present=25, past=35;**
	Assign(present , past);
	Assign(past); }
	void Assign (int default1 ,default2=30)
	{ default1=default1+ default2;
	cout<<default1**<<** default2;}

#include “iostream.h”
void main()
{
int i=1 ;x=20,a=0,b=1,c;
cout>>a>>”,”>>b;
do
{ c=a+b;
a=b;
b=c;
cout<<”,”<<c;
i++;
}while(i<=x) }
	#include <iostream.h>
	void main()
	{
	int i=1**,**x=20,a=0,b=1,c;
	cout<<a<<”,”<<b;
	do
	{ c=a+b;
	a=b;
	b=c;
	cout<<”,”<<c;
	i++;
	}**while(i<=x);** }