CHAPTER 6

FUNCTIONS

After completing this lesson, you will be able to:

- Explain the concept and types of functions
- Explain the advantages of using functions
- Explain the signature of functions (Name, Arguments, Return type)
- Explain

  Function prototype

  Function definition

  Function call
- Differentiate among local global and static variable
- Differentiate between formal and actual parameters
- Know the concept of local and global functions
- Use inline functions
- Pass the arguments

  by constants

  by value

  by reference
- Use default arguments
- Use return statement
- Know advantage of function overloading
- Understand the use of function overloading with

  Number of arguments

  Data type of arguments

  Return type

1.1 FUNCTIONS

Q1. What is meant by a function? Explain its purpose in C++

Answer

Function

A function is a group of statements that together perform a task

Purpose of Functions in C++

Every C++ programs has at least one function, which is main() additional functions can be defined in the program. Each function performs a specific task

Functions are one of the main building block of any programming language. Functions are used to provide modularity to a program. Creating an application using function makes it easier to understand, edit, check errors etc. function makes the work easier by writing the code once and executing it again and again as many times as required.

1.1.1 TYPES OF FUNCTIONS

Q2. Explain the types of functions in C++.

Answer

Types of function

C++ functions are categorized into the following two types

1. Library or built in functions
2. User defined functions

1. **Library or Built in Functions**

   Library or built in functions are the pre-defined functions in C++

Uses of Library or Built in Functions

Programmers can use these functions by invoking calling them directly, they do not need to write code for them

Series of Library or built in Functions

C++ provides a series of library or pre-defined functions for various commonly used operations of algebra, geometry, trigonometry, finance, graphics, clipboards etc. these functions are part of the C++ language and are completely reliable. Header file <cmath.h> must be included in the program to use library or built in functions

Examples of library or Built In Functions

Some examples of commonly used library or built in functions are:

1. Abs()
2. Div()
3. Getchar()
4. Log()
5. Pow()
6. Putchar()
7. Puts()
8. Sqrt()
9. Strcmp()
10. Time()

Program- Demonstrates the use of built in function sqrt()

The following program indicates the use of built in functions sqrt(). It requests a number and calculates its square root using sqrt() function

```
// Program to find square root of a number using sqrt) built in function

#include<iostream>

#include<math.h >          // It is required to use built in functions

Int main()

{

Double number, squareroot:
```

```
Cout<<"Enter a number";
Cin>>number;
// sqrt() is a library functions to calculate square root
Square root = sqrt(number);
Cout<<"square root of "<< number<< "="<< square root;
Return 0;
}
```

Output of the Program

Enter a number 25

Square root of 25= 5.0

Explanation

In the above program sqrt() library function is used to calculate the square root of a number. The code #include <cmath.h> in the above program is a header file. The function definition of sqrt() is present in the cmath header file. This header file contains definition of all the mathematical functions

2. **User defined functions**

C++ allows programmers to define their own functions. The functions defined by

By the programmers, according to their needs are called users defined functions. If any function is not available as built in function, users can define their own functions and use them in the same way as built in functions are used.

Working of user defined function

Consider the figure 6.1 for the working of user defined function in a C++ program

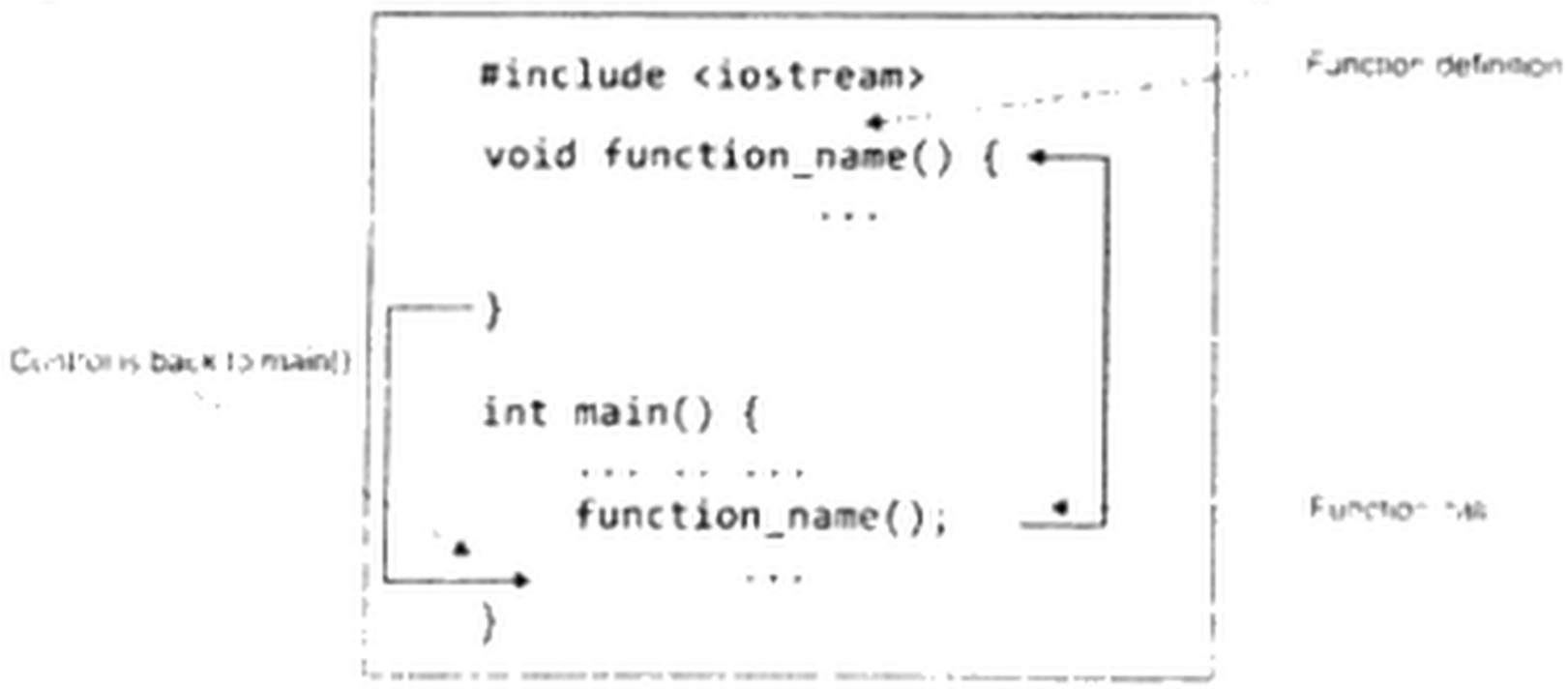

Figure 6.1 Working of User-defined function

Explanation

1. When a program begins running, the system calls the main() function, that is the system starts executing codes from main() function
2. When a function is called inside main() by its name it moves to void function_name() and all codes inside the function are executed
3. After completion of the function code, the control moves back to the main() function where the code after the call to the function_name() is executed.

Q3. Describe the method to define user defined function

Answer

Defining user defined function

A function must be defined first to use it in the program

Syntax

The general syntax for defining a user defined function is

Return-type function-name (parameters)

```
{
// function-body
}
```

Explanation

Return type

The return type suggests what the function will return. It can be int char some pointer or even a class object. There can be functions which does not return anything, they are mentioned with void.

Function name

Function name is the name of the function using the function name it is called

Program

The following program contains a function named sum with no return type and having parameters x and y with integer data type

```
Void sum(int x, int y)          // Function definition
{
Int z:
Z = x + y;
Cout<<z;
}
Int main()                      // main() program
{
Int a =10;
Int b= 20;
Sum (a, b):                     // Function sum is called/used
```

```
}
```

Here a and b are sent as arguments and x and y are parameters which will hold values of a and b to perform required operation inside function

Output of the Program

20

1.1.2 ADVANTAGES OF USING FUNCTIONS

Q4. State the advantages of using functions

Answer

Advantages of using functions

Using functions we can structure our programs in a more modularized way. The use of functions provides many advantages

1. Using function a program can be defined in logical blocks. It will make the code clear and easy to understand
2. Use of function avoids typing same pieces of code multiple times. User can call a function to execute same lines of code multiple times without re-writing it
3. Individual functions can be easily treated
4. In case of any modification in the code user can modify only the function without charging the structure of the program
5. It is much easier to change or update the code in a function which needs to be done once

1.1.3 FUNCTION SIGNATURE

Q5. Name the parts of signature of a function. Give some examples

Answer

Parts of Function Signature

The signature of a function comprises of the parts given below

1. Name of the function
2. Arguments: the number, order and data types of the arguments
3. Return type of function

Example

Let us consider the add function to demonstrate function signature

```
// Function signature
Double add(int x, double y)
{
Return x+y;
}
```

The signature of the above mentioned function is

Double add (int x, double y)

The signature of this add function comprises of

1. Name of the function add
2. The data types of the arguments int, double
3. Return type of the function double

1.1.4 FUNCTION COMPONENTS

Q6. Name the components of function. Explain each component with example

Answer

Components of Functions

Each function consists of three components. These are

i. Function prototype/declaration
ii. Function definition
iii. Function call

1. **Function prototype/declaration**

Function prototype/declaration is done to tell the compiler about the existence of the function. Functions return type its name and parameter list is mentioned. A function prototype/declaration is used before the main() function. It ends with a semicolon(;)

Use of Function Prototype

Function prototype is used in C++ program only when the function is defined after the definition of main() function. If the function definition lies before the main() function then there is no need for the function prototype

Program-Demonstrates the use of function prototype

In the following program the underline portion indicates the function prototype/declaration

```
#include<iostream>
Using namespace std:
Int sum(int x, int y):        // Function prototype/declaration
Int main()
{
Int a=10:
Int b=20:
Int c= sum (a, b):
```

```
Cout <<c:
}
Int sum (int x, int y)    // Function definition
{
Return (x + y):
}
```

Output of the Program

30

2. **Function definition**

   A function definition specifies what a function does

Parts of function definition

Function definition has two parts

i. **Header**
ii. **Function body enclosed in {}**

Function header

Function header is similar to function prototype with the only difference that it has the variables name for the parameters and no semicolon (:).

Program

The following program indicates the function_definition

```
#include<iostream>
Using namespace std:
Int sum (int x, int y):          // Function prototype/declaration
Int main()
{
```

```
Int a = 10;
Int b = 20;
Int c = sum (a, b):
Cout << c;
}
Int sum (int x, int y)   // Function definition
{
Return (x + y)
}
```

Output of the Program

30

1.1.5 SCOPE OF VARIABLE IN FUNCTIONS

Q7. What do you mean by scope of variables in functions? Explain the scopes of variables.

Answer

Scope of a Variable

By the scoop of a variable we mean that in which parts of the same program the variable can be accessed.

Scope of variables

There are three scopes of variables

i. Local scope
ii. Global scope
iii. Static scope

1. **Local/Automatic variables**

Variables defined within a block of a function are called local variables. Their scope is local to the block of function only. These variables are not accessible from outside the block and thus their visibility remains only to that block

Example

In the following code segment the variables a b and c are local variables and cannot be used outside the main function

Program

Following is the example using local variables

```
#include<iostream>

Using namespace std;

Int main()

{

// Local variable declaration:

Int a , b;

Int c;

// actual initialization

A=10;

B= 20;

C= a+b;

Cout<<c;

Return 0;

}
```

Output of the Program

30

Global variables

Variables are defined at the top of a program before the main() function are called global variables these variables are accessible by all the functions of the same program. In order to declare global variables. We simply have to declare the variable outside any function or block that means directly in the body of the program

Explanation

In this example the variables a b and c are defined globally and have therefore been accessed both in main() function and addition() function

Static variables

Static variables are those variables which are preceded by the keyword static while declaring them

Example

Static abc:

Initialization of static variable

Static variables are initialized once in the program and remains in memory until the end of the program. These variables have the capability to preserve information about the last value a function returned. Static variables are local in scope to their module or function in which they are defined

Program-Demonstrates the use of static variables

Consider the following example to demonstrate the use of static variables

```
// static variables program

#include<conio.h>
```

```
#include<iostream.h>
Void demo();          // function prototype
Void demo();
{
Auto int var1 = 0;      // automatic/local variable
Static int var 2 = 0;     // static variable
Cout << "Automatic/local variable = "<<var1<<
        "," << "static variable = "<< var2<< end1;
++ var1;
++ var2;
}
Int main()            // start of main() page
{
Int i= 0;
While (i<#)
{
Demo();
I++;
}
Getch();
Return 0;
}
```

Output of the program

Automatic/Local variable = 0, Static variable = 0

Automatic/Local variable = 0, static variable =1

Automatic/local variable = 0, static variable =2

Explanation

Here the automatic variable var 1 losses its value when the control goes out of the function body but the static variable var 2 keep its last value

1.1.6 PARAMETERS

Q8. What is meant by parameter in C++? Explain.

Answer

Parameters

In function prototype and function call the variables and values (written in the parenthesis) are called parameters

Types of parameters

There are two types of parameters

i. Formal parameters
ii. Actual parameters

i. **Formal Parameters**

Formal parameters are those parameters which appear in function declaration/header and also in function prototype. These are also called arguments

Example

```
Void foo(int x),        // prototype – x is a formal parameter
```

```
Vid foo (int x)          // declaration – x is a formal parameter
{
}
```

Actual Parameters

Actual parameters are those parameters which appear in function cells. These are also called arguments

Example

Consider the following function cell

```
Foo (6):    // 6 is the argument passed to parameter x
Foo (y +1):   // the value of (y+1) is the argument passed to parameter x
```

The actual parameter can be fixed values variables holding values or expressions resulting in some values

1.1.7 LOCAL AND GLOBAL FUNCTIONS

Q9. Why local and global functions are used? Explain these concepts with programs

Answer

Local and Global Functions

The terms local and global functions specify the scopes of functions within programs

Types of Functions based on their scope

Based on the scope of the functions. Functions are categorized into two types

1. **Local functions**
2. **Global function**

Local functions

Those function which are defined inside the body of another function are called local function. Normally we use built in function inside the body of main() functions to perform our activities. Such declarations are termed as local

Program

Consider the following example

```
// local functions
#include<iostream.h>
#include<conio.h>
#include<math.h>
Int main()
{
Clrscr();
Int n;
Cin>>n;
Cout<<"absolute value of "<<n<<"=" <<abs(n);
Getch();
Return 0;
}
```

Output of the Program

```
-15
Absolute value of -15 = 15
```

Explanation

In the above program function Clrscr(), abs() and getch() are the local functions because they lie inside the main() function

Global Functions

A function declared outside any function is called global function. A global function can be accessed from any part of the program. Normally, user defined functions are considered as global function because usually they are defined before the main() function and are thus accessible to every part of the program

Program

Consider the following simple example

```
#include<iostream.h>

#include<conio.h>

Void print()

{

Cout << "this is global functions";

}

Int main()

{

Print ();

Cout << end1;

Print();

Getch();

Return 0;

}
```

Output of the Program

This is global function

This is global function

Explanation

In the above program, the function print() is defined at the start of the program and is thus global to every part of the program (main() function inside the main() program. It can be accessed easily.

1.1.8 INLINE FUNCTION

Q10. Why do we need inline function? Describe its working, disadvantage and format. Explain the use of inline function with the help of program

Answer

Need of Inline Function

Using function calls in program, a lot of CPU time is wasted in passing control form the calling program (main() function) to the called function and returning control back to the calling program. This limitation can be overcome by the use of inline function

Working of Inline function

In Inline function the function return type is preceded by the inline keyword which requests the compiler to treat the function as an inline and do not jump again and again to the calling function (main() and back to it in the case of inline function when the compiler compiles the code, all inline functions are expanded In place that is the function call is replaced with a copy of the contents of the function itself which removes the function call overhead.

Disadvantages of inline function

The disadvantages of the inline function is that it can make the compiled code quite larger especially if the inline function is long and or there are many calls to the inline function.

Format for the Declaration of Inline Function

The format for the declaration of inline function is given in the following segment

```
// inline function declaration
Inline type name (arguments ....)
{
Statements:
}
```

Calling an inline function is simple and is just like the call to any other function. In the call, the inline keyword is not needed to be included

Program- Demonstrates the use of inline function

The following program demonstrates the use of inline function in a program

```
// use of inline function to find minimum out of two integers
#include<iostream.h>
#include<conio.h>
Inline int main(int a, int b)
{
Return a > b > b a;
}
Int main()
{
Cout<<"Minimum out of 13 and 32 is:      "<<min(13, 32);
```

```
Cout<<"\nMinimum out of 34 and 65 is:  "<<min(34, 65);
Getch();
Return 0;
}
```

Output of the Program

```
Minimum out of 13 and 32 is: 13
Minimum out of 34 and 65 is: 34
```

1.2 PASSING ARGUMENTS AND RETURNING VALUES

Q11. State the use of passing arguments and returning values in C++ programs

Answer

When we want to execute functions we need to pass arguments (values) to them. The result (values) produced within the function body is then returned back to the calling program. The following section describes different methods used for passing arguments (values) to functions

1.2.1 PASSING ARGUMENTS

Q12. Describe the most commonly used methods of passing arguments to functions. Explain each method with the help of program

Answer

Methods of Passing Arguments to Functions

The following are the most commonly used methods of passing arguments to functions

1. Passing arguments by constants
2. Passing arguments by values
3. Passing arguments by reference

1. **Passing arguments by constants**

   While calling a function arguments are passed to the calling function. In passing arguments by constants the actual constant values (numeric and character) are passed instead of passing the variables holding these constants.

Program 1

Consider the following program

```
// Passing Integer constant
#include<iostream.h>
#include<conio.h>
Void show (int x)
{
Cout << "x= "<<x<<End1;
}
Int main()
{
Show (60);
Getch();          // function call
Return 0;
}
```

Output of the program

X = 60

Explanation

In the above program, the function call show(60) passes the argument (60) to the function

Program 2

Consider another program defining a function **showGender()** that takes a character constant, 'F' or 'M' as argument from the calling function

```
// Passing character constants
#include<iostream.h>
#include<conio.h>
Void showGender (char ch)
{
Cout <<"The gender marker you passed is : "<<ch<< end1;
}
Int main ()
{
ShowGender ('F');   // Function Call
Getch ();
Return 0;
}
```

Output of the Program

The gender marker you passed is: F

2. Passing arguments by values

By default arguments in C++ are passed by value. When arguments are passed by value a copy of the argument is passed to the function

Program

Consider the following program to demonstrate the use of passing arguments by value

```
//passing parameters by value example 2
#include<iostream.h>
#include<conio.h>
Void foo (int y)
{
Cout<<"y in foo () = "<<y<<End1;
Y=6;
Cout<<y in foo () = <<y<<end1;
}                              // y is destroyed here
Int main ()
{
Int x = 5;
Cout<<"x in main () before call = "<<x<<end1;
Foo(x);
Cout<<"x in main () after call= "<<x<<end1;
Getch ();
Return 0;
}
```

Output of the Program

X in main () before call = 5

Y in foo () = 5

Y in foo () =

X in main () after call = 5

Explanation

In this program the original value of 'x' is not changed before and after calling the function foo () although it changes the values within its body

3. **Passing arguments by reference**

   In pass by reference, the reference to the function parameters is passed as arguments rather than variables. Using pass by reference, the value of arguments can be changed within the function

In passing arguments by reference, the original variables should precede by the ampersand sign (&)

Program

Consider the following example

```
// Passing parameters by reference example 1

#include<iostream.h>

#include<conio.h>

Void add one (int & y

{

Y++;                    // changing values in function

}
```

```
Int main ()
{
Int x = 55:
Cout <<"x in main () before call = "<<x<<End1;
Addone (x)):
Cout <<"x in main () after call= "<<x<<End1;
Getch ():
Return 0:
}
```

Output of the Program

X in main () before call = 55

X in main () after call = 56

Values returned by Function

Sometimes we need a function to return multiple values. However, functions can only have one return value. One way to return multiple values is the use of the method of passing arguments by reference

Change the Value of the Arguments

Passing arguments by this method allow programmers to change the value of the arguments, which is sometimes useful. Similarly, it is a fast approach to passing and processing values in a function and also has the facility of returning multiple values from a function

1.2.2 DEFAULY ARGUMENTS

Q13. Why do we need default arguments? State its syntax and explain this concept with program

Answer

Arguments

When declaring a function, we can specify a default value for each of the last parameters. This values will be used if the corresponding argument is left blank when calling to the function. To do this we simply have to use the assignment operator and a value for the arguments in the function declaration.

Default value

If a value for that parameter is not passed when the function is called, the default value is used but if a value is specified this default value is ignored and the passed value is used instead

Syntax

To demonstrate the concept of default arguments, consider the following general syntax

Type function_name (parameer1=value)):

Here in this line of code the type is any valid data type function_name is the name of the function and parameter1 is the parameter having a default value named value assigned to it in the prototype

Example

In the example given below parameter 'b' has a default value 2 assigned to it in the declaration. Now if one value is passed in the function call then the default value of b will be takn as the value of parameter 'b'

Int divide (int a , int b=2)

Program

Consider the following program to demonstrate the concept of default arguments

```
//default arguments program
#include <iostream.h>
#include<conio.h>
Int divide (int a, int b=2)
{
Int r;
R=a/b;
Return ®;
}
Int main ()
{
Cout << "Result by providing one argument = "<<divide (12);
/* function call with fewer arguments */
Cout << end1;
Cout << "Result by providing both the arguments = "<<divide (20, 4);
Getch ()
Return 0;
}
```

Output of the Program

Result by providing one argument = 6

Result by providing both the argument = 5

Explanation

As we can see in the body of the program there are two calls to function divide() in the first one

Divide (12):

We have only specified one argument but the function divide () gets the second value from the default argument int b=12 in the function prototype and thus result in 6

In the second call

Divide (20, 4):

There are two parameters so the default value for b (int b=2) is ignored and 'b' takes the value passed as argument that is 4 making the result returned equal to 5(20/4).

1.2.3 RETURN STATEMENT

Q14. State the use and syntax of return statement. Explain with program

Answer

Return Statement

If the return type of a function is any valid data type such as int, long, float, double, long double or char then return statement should be used in the function body to return the result to the calling program (main () function)

Syntax

Consider the following function

```
// the use of return statement in the cube () function

Int cube (int x)

{
```

```
Int c,
C = x*x*x:
Return c:
}
```

If a function returns a value by the use of return statement then the call to the function should be called from a statement or an expression

Example 2

```
// function call in the case of using return statement in a function
Cout << cube (x):     // first method
Int y=cube (x):       // second method
```

1.3 FUNCTION OVERLOADING

Q15. Define function overloading. Explain this concept by giving a program code

Answer

Function Overloading

In programming languages, two or more variables functions with the same name cannot be used in a single program or block of code. Function overloading is a feature of C++ that allows to create multiple functions with the same name so long as they have different number or types of parameters.

Program

Consider the following example having two functions with the same name 'multiply' that operate on integers and floating point numbers

```
// use of overloaded function
```

```
#include<iostream>
#include<conio.h>
Int multiply (int a, int b)
{
Return (a*b);
}
Float multiply (float a, float b)
{
Return (a*b);
}
Int main ()
{
Cout << "Output of integers"<< multiply (4, 30)<< end1;
Cout<<"Output of floats " << multiply (3.5, 4.5);
Getch ();
Return 0;
}
```

Output of the Program

Output of integers = 120

Output of floats = 15.75

Explanation

In this case we have defined two functions with the same name multiply that accept two arguments. One accepts two arguments of type int and the other

of type float. The compiler looks at the arguments and calls its respective function. Here multiply (int a, int b) is called for

Multiply (4, 30) because he arguments match with the data types of the formal parameters.

Similarly, multiply (float a, float b) is called for the call multiply (3.5, 4.5) because the arguments and formal parameters match each other in types

1.3.1 ADVANTAGES OF FUNCTION OVERLOADING

Q16. State the use of advantages of function overloading

Answer

Advantages of Function Overloading

Following are the uses of function overloading

1. Using function overloading we can declare multiple functions with the same name that have slightly different purposes
2. Function overloading can significantly lower complexity of programs
3. It increases the readability of programs
4. It exhibits the behavior of polymorphism

1.3.2 USE OF FUNCTION OVERLOADING

Q17. Describe the use of function overloading

Answer

Use of Function Overloading

For the complete understanding of the concept of function overloading one should know those features of the overloaded functions which disambiguate them and make them unique in a single program.

Features of Function Overloading

These features are listed hereunder

1. Number of arguments
2. Data type of arguments
3. Return type

1. **Number of arguments**

   Functions can be overloaded if they have different numbers of parameters. More than one function with the same name but different number of parameters can be used in a single program. Int the calls to these functions the compiler disambiguates the calls by looking at the number of arguments and formal parameters in the function decelerators

Program

Consider the following example

```
//overloaded functions with different number of parameters

#include<iostream.h>

#include<conio.h>

Int Addition (int x, int y)

{

Return (X + Y);

}

Int Addition (int X, int Y, it Z)

{
```

```
Return (X + Y + Z)
}
Int main ()
{
Int a=55, b=22, c=100;
Cout <<"Output of 2 integers="<<Addition (a, b)
Cout<<end1;
Cout<<"Output of 3 integers="<< Addition (a, b, c);
Cout << end1;
Getch()
Return 0;
}
```

Output of the Program

Output of 2 integers = 77

Output of 3 integers = 177

Explanation

In this example two functions have been used with the name **Addition**. One has two parameters and the second has three parameters. In the main() function there are two calls Addition (a, b) and Addition (a, b, c) which call function int Addition (int X, int Y) and int Addition (int X, int Y, int Z) respectively by looking at the number of parameters

2. Data type of arguments

One way to achieve function overloading is to define multiple functions with the same name but different types of parameters

Program

Consider the following example

```
// overloaded function print () with different types of parameters
#include <iostream.h>
#include <conio.h>
Void print (int A)
{
Cout <<"A="<<A<<end1;
}
Void print (float B)
{
Cout<<"B="<<B<<End1;
}
Int main ()
{
Print (15;
Print (21.45);
Getch ();
Return 0;
}
```

Output of the Program

A= 15

B = 21. 45

Here two functions, print (int A) and print (float B) have been used with different types of parameters int and float. In main () function, two calls are used print (!5) and print (21.45). the compiler looks at the type of arguments and calls the corresponding function.

3. **Return type**

   The return type of a function is not considered when functions are overloaded. It means that if two or more functions with the same name have same function signatures but different return type then they are not considered as overloaded and the compiler will generate error message

Example

Consider the following case

```
Int display ();                    // this prototype generates error message
Double display ();
```

Explanation

Here the compiler generates error message of re-declaration for the second declaration at line number 2. It is because that both the declarations have same number of parameters (zero in this case) but only the return types are different which do not play any role in the overloading. If we want to do it for this these functions will need to be given different names

KEY POINTS

- A function is a self-contained program that performs a specific task
- C++ has two types of functions, built in and user defined. Built in functions are specific in their activities and cannot be used for general type of tasks
- Every C++ program comprise of one function called main (). It is the point from where the compiler starts the execution of every C++ program
- A good C++ programmer writes programs that comprise of small functions. The main () function consists of function calls
- Function are one of the main building block of C++ programs. It modularizes large programs into segments and thus increase the program readability and also provides the facility of code reusability
- Function prototype tells the compiler the name of the function, number, types and order of parameters
- A function definition is a set of instructions enclosed in a block which performs the intended task of the function
- Local/automatic variables have local scope and can only be accessed in the block in which they are declared. These variables cannot be accessed from outside the block
- Global variables have global access and its visibility is throughput the program in which they are defined
- Static variables have scope of local variables and retain its values throughout the life of the program
- The variables that appear in the function prototype and function declaration are called formal parameters.
- The variables in function cells that hold the values to be passed to the function are called actual parameters

- Inline functions are special functions with inline keyword that are used to minimize the problem of function call overhead with the expense of memory wastage