CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE142: OBJECT ORIENTED PROGRAMMING WITH C++

December – April 2019

Chapter – 5

Functions

Devang Patel Institute of Advance Technology and Research

DEPSTAR

Objectives

- To be able to recognize when the use of a function would be appropriate in a program.
- To know the syntax for declaring and defining a function.
- To know how to return a value from a function.
- To know how to write and use functions that do not return values (sometimes called procedures).
- To know how to use parameters with function calls.
- To know under what condition parameters must be passed by reference.
- To know under what condition parameters should be passed by constant reference.
- To be able to write functions that accept parameters as values, by reference and by constant reference.

Contents

- Return types in main( )
- Function Prototyping
- Call by Reference & Call by Value
- Return by Reference
- Inline Functions
- Default Arguments
- Constant Arguments
- Function Overloading

Introduction

- Dividing a program into functions.
  - a major principle of top-down, structured programming.
- To reduce the size of the program.
- Code re-use.
- Like C++ operators, a C++ function can be overloaded to make it perform different tasks depending on the arguments passed to it.

Continue…

```
void show( );    /* Function declaration */
void main( )
{
    ----
    show( );      /* Function call */
    ----
}
void show( )     /* Function definition */
{
    ----
    ----          /* Function body */
}
```

The main( ) Function

- The main( ) returns a value of type int to the operating system by default.
- The functions that have a return value should use the return statement for termination.
- Use void main( ), if the function is not returning any value.

Function Prototyping

- The prototype describes the function interface to the compiler by giving details such as:
  - The number and type of arguments
  - The type of return values.
- It is a template
- When the function is called, the compiler uses the template to ensure that proper arguments are passed, and the return value is treated correctly.

Continue…

- Function prototype is a declaration statement in the calling program.

type function-name (argument-list) ;

- The argument-list contains the types and names of arguments that must be passed to the function.

Continue…

- Each argument variable must be declared independently inside the parentheses.

```
float avg ( int x, int y) ;    // correct
float avg ( int x, y) ;        // illegal
```

- In a function declaration, the names of the arguments are dummy variables and therefore they are optional.

Continue…

float avg (int , int) ;

The variable names in the prototype just act as placeholders and, therefore, if names are used, they do not have to match the names used in the *function call* or *function definition*.

```
void display( );          // function with an
void display(void);       // empty argument list.
```

Call by Value

A function call passes arguments by value.

- The called function creates a new set of variables and copies the values of arguments into them.

- The function does not have access to the actual variables in the calling program and can only work on the copies of values.

Call by Value

Example :

```
using namespace std; // function declaration
void swap(int x, int y);
int main () {
// local variable declaration:
int a = 100;
int b = 200;
cout << "Before swap, value of a :" << a <<
    endl;
```

Call by Value

```
cout << "Before swap, value of b :" << b << endl;
//calling a function to swap the values.
 swap(a, b);
cout << "After swap, value of a :" << a << endl;
cout << "After swap, value of b :" << b << endl;
return 0;
}
```

Call by Reference

When we pass arguments by reference, the formal arguments in the called function become aliases to the actual arguments in the calling function.

This means that when the function is working with its own arguments, it is actually working on the original data.

Call by Reference

Example :

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

// function declaration
void swap(int &x, int &y);

int main ()
{
   // local variable declaration:
   int a = 100;
   int b = 200;

   cout << "Before swap, value of a :" << a << endl;
   cout << "Before swap, value of b :" << b << endl;

   /* calling a function to swap the values using variable reference.*/
   swap(a, b);

   cout << "After swap, value of a :" << a << endl;
   cout << "After swap, value of b :" << b << endl;

   return 0;
}
```

Call by Reference

Output :

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100
```

Return by Reference

A function can return a reference.

```
int & max (int &x, int &y)
{
        if (x > y)
                return x;
        else
                return y;
}
```

Return by Reference

Example :

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

// Global variable
int num;

// Function declaration
int& test();

int main()
{
      test() = 5;

      cout << num;

      return 0;
}

int& test()
{
      return num;
}
```

Output :

5

In program above, the return type of function **test()** is **int&**. Hence, this function returns a reference of the variable num.

Inline Functions

- Inline function is a function which when invoked requests the compiler to replace the calling statement with its body.
- A keyword **inline** is added before the function name to make it inline.
- It is an optimization technique used by the compilers as it saves time in switching between the functions otherwise.
- Member functions of a class are inline by default even if the keyword **inline** is not used.
- Syntax :

```
inline return_type function_name ([argument list])
    {
    body of function
    }
```

Inline Functions

- Example :

```
#include <iostream>
#include <conio.h>
using namespace std;
inline void print(int x)
{
    cout<<x<< " ";
}
int main() {
    int i, N;
    cout<<"C++ Program to print first N natural numbers"<<endl<<endl;
```

Inline Functions

```
        cout<<"Enter total number of natural numbers:"<<endl; cin>>N;
        for(i=1;i<=N;i++)
        {
                print(i);
        }
        return 0;
}
```

Output :

```
C++ Program to print first N natural numbers

Enter total number of natural numbers:
10
1 2 3 4 5 6 7 8 9 10
```

Function Overloading

- In C++ programming, two functions can have same name if number and/or type of arguments passed are different.

- These functions having different number or type (or both) of parameters are known as overloaded functions.

For example:

```
int test() { }
int test(int a) { }
float test(double a) { }
int test(int a, double b) { }
```

Function Overloading Continue…

- Here, all 4 functions are overloaded functions because argument(s) passed to these functions are different.
- Notice that, the return type of all these 4 functions are not same. Overloaded functions may or may not have different return type but it should have different argument(s).
- The number and type of arguments passed to these two functions are same even though the return type is different. Hence, the compiler will throw error.

```
int test(int a) { }

double test(int b){ }
```

Default Argument

- In C++ programming, you can provide default values for function parameters.
- The idea behind default argument is simple. If a function is called by passing argument/s, those arguments are used by the function.
- But if the argument/s are not passed while invoking a function then, the default values are used.
- Default value/s are passed to argument/s in the function prototype.

Default Argument Continue...

Case1: No argument Passed

```
void temp (int = 10, float = 8.8);

int main( ) {
    temp( );
}

void temp(int i, float f ) {
... ... ...
}
```

Case2: First argument Passed

```
void temp (int = 10, float = 8.8);

int main( ) {
    temp(6);
}

void temp(int i, float f ) {
... ... ...
}
```

Case3: All arguments Passed

```
void temp (int = 10, float = 8.8);

int main( ) {
    temp(6, -2.3 );
}

void temp(int i, float f ) {
... ... ...
}
```

Case4: Second argument Passed

```
void temp (int = 10, float = 8.8);

int main( ) {
    temp( 3.4);
}

void temp(int i, float f ) {
... ... ...
}
```

i = 3, f = 8.8
Because, only the second argument cannot be passed
The parameter will be passed as the first argument.

Figure: Working of Default Argument in C++

Default Argument Continue…

- **Example :**

```
#include<iostream>
using namespace std;
// A function with default arguments, it can be called with
// 2 arguments or 3 arguments or 4 arguments.
int sum(int x, int y, int z=0, int w=0)
{
   return (x + y + z + w);
}
/* Drier program to test above function*/
int main()
{
   cout << sum(10, 15) << endl;
   cout << sum(10, 15, 25) << endl;
   cout << sum(10, 15, 25, 30) << endl;
   return 0;
}
```

Output:

```
25
50
80
```

Default Argument Continue…

- **Common mistakes when using Default argument**

1. void add(int a, int b = 3, int c, int d = 4);

The above function will not compile. You cannot miss a default argument in between two arguments. In this case, c should also be assigned a default value.

2. void add(int a, int b = 3, int c, int d);

The above function will not compile as well. You must provide default values for each argument after b. In this case, c and d should also be assigned default values.
If you want a single default argument, make sure the argument is the last one. void

add(int a, int b, int c, int d = 4);

3. No matter how you use default arguments, a function should always be written so that it serves only one purpose.

Review Questions

Write True or False :

1. A function can return more than one value. –**False**
2. A function needs function definition to perform a specific task. – **True**
3. If a function returns no value, the return type must be declared as void. – **True**
4. In a function, The return statement is not required if the return type is anything other than void. – **False**
5. A local variable declared in a function is not usable out side that function. - **True**

Review Questions

Write a function for following Problem statement :

1. Function to print prime numbers between 100 to 200
2. Function to print Fibonacci series
3. Function to print factorial of number

Presentation Prepared By:

Ms. Krishna Patel

Contact us:

dweepnagarg.ce@charusat.ac.in
parthgoel.ce@charusat.ac.in
hardikjayswal.it@charusat.ac.in
dipakramoliya.ce@charusat.ac.in
krishnapatel.ce@charusat.ac.in
khushipatel.ce@charusat.ac.in

Subject Teachers:

Ms. Dweepna Garg
Subject Coordinator

Mr. Parth Goel
https://parthgoelblog.wordpress.com

Mr. Hardik Jayswal

Ms. Khushi Patel