



Computer Science Department
Lab Manual - 02

Course Code: CL-217	Course : Object Oriented Programming Lab
---------------------	--

CONTENTS:

- 1) Predefined Functions
- 2) User-Defined Functions
- 3) Function Parameters.
 - a) Value Parameters
 - b) Reference Parameters
- 4) Scope of an Identifier.

1. PREDEFINED FUNCTIONS

Functions in C++ are similar to that of in Algebra. For Example, every function has a name and depending on the value specified by the user. It does some computation and gives an output (if any).

Some examples for the built in functions are given below.

Function	Header File	Purpose	Parameter(s) Type	Result
<code>abs (x)</code>	<code><cmath></code>	Returns the absolute value of its argument: <code>abs (-7) = 7</code>	<code>int</code> <code>(double)</code>	<code>int</code> <code>(double)</code>
<code>ceil (x)</code>	<code><cmath></code>	Returns the smallest whole number that is not less than <code>x</code> : <code>ceil (56.34) = 57.0</code>	<code>double</code>	<code>double</code>
<code>cos (x)</code>	<code><cmath></code>	Returns the cosine of angle: <code>x</code> : <code>cos (0.0) = 1.0</code>	<code>double</code> <code>(radians)</code>	<code>double</code>
<code>exp (x)</code>	<code><cmath></code>	Returns e^x , where $e = 2.718$: <code>exp (1.0) = 2.71828</code>	<code>double</code>	<code>double</code>
<code>fabs (x)</code>	<code><cmath></code>	Returns the absolute value of its argument: <code>fabs (-5.67) = 5.67</code>	<code>double</code>	<code>double</code>

Example Code for Predefined Functions:

```
//How to use predefined functions.
#include <iostream>
#include <cmath>
#include
<cctype>using
namespace std;
intmain ()
{ intx; double u, v;u
    = 4.2; //Line 1 v
    = 3.0; //Line 2
    cout<< "Line 3: " << u << " to the power of "
    << v << " = " << pow (u, v) <<endl; //Line 3
    cout<< "Line 4: 5.0 to the power of 4 = " <<
    pow (5.0, 4) <<endl; //Line 4 u = u + pow
    (3.0, 3); //Line 5
    cout<< "Line 6: u = " << u <<endl; //Line 6 x
    = -15; //Line 7
    cout<< "Line 8: Absolute value of " << x
    << " = " << abs(x) <<endl; //Line 8
    return 0;
}
```

Sample Run:

Line 3: 4.2 to the power of 3 = 74.088

Line 4: 5.0 to the power of 4 = 625

Line 6: u = 31.2

Line 8: Absolute value of -15 = 15

2.USER DEFINED FUNCTIONS

User defined functions are classified into two categories.

- **Value-Returning Functions:** These functions return a value of a specific type using *return* statement.
- **Void Functions:** These functions *do not* use a *return* statement to return a value.

VALUE-RETURNING FUNCTIONS:

Value-returning functions can be utilized in one of three ways:

- Save the value for further calculation.
- Use the value in some calculation.
- Print the value.

These function can be called in following scenarios.

- In an assignment statement.
- As a parameter in a function call. - In an output statement.

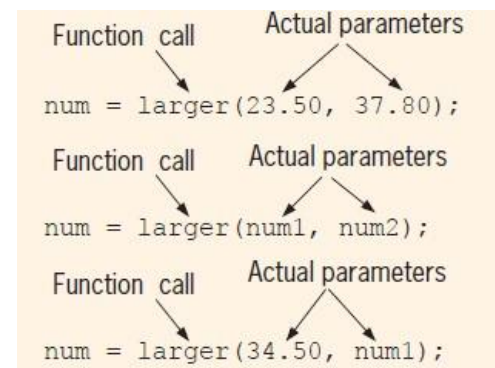
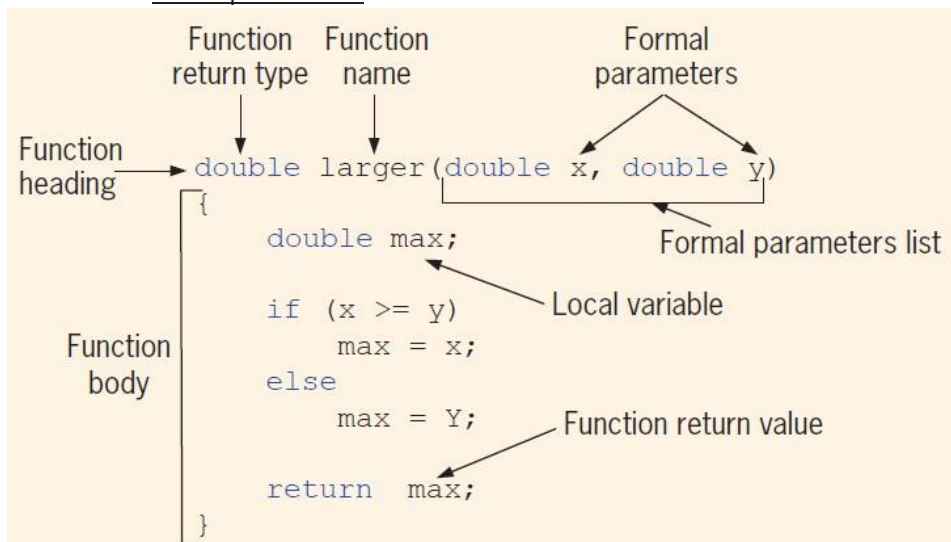
Syntax:

```
functionType functionName(formal parameter list)
{
    statements
}
```

Example Code for Value Returning Functions:

```
double larger (double x, double y)
{ doublemax; if
    (x >= y)
    max = x;
    else max =
    y; return
    max;
}
```

Components:



Usage in cout:

```
cout<< "The larger of 5 and 6 is " << larger (5, 6)
<<endl; //Line 1
cout<< "The larger of " << one << " and " << two <<
" is " << larger (one, two) <<endl; //Line 2 cout<<
"The larger of " << one << " and 29 is "
<<larger (one, 29) <<endl; //Line 3
```

```
maxNum = larger (38.45, 56.78); //Line 4
```

Usage in another Function:

```
double compareThree (double x, double y, double z)
{ return larger (x, larger (y, z));
}
```

Some Peculiarities in Functions:

A function with a returning must return a value. Consider the following function.

```
int secret (int x)
{ if (x > 5) //Line 1 return 2 *
    x; //Line 2
}
```

Now in this function when the condition of $x > 5$ is not met then the return statement will not execute. Hence resulting in no return value. In this case function may or may not return a meaningful value. It may result some strange value.

3.FUNCTION PARAMETERS:

When passing parameters to a function there are ways to do that.

1. Value Parameters (Pass by Value).
2. Reference Parameters (Pass by Reference).

VALUE PARAMETERS:

When passing value parameters in a function, the parameter is copied into the corresponding formal parameter. There is no connection between the actual and formal parameter values, this means that these parameters cannot be used to pass the result back to the calling function.

Example Code for Value Parameters:

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

void funcValueParam (int num);

int main ()
{ int number = 6; //Line 1
  cout<< "Line 2: Before calling the function "
  << "funcValueParam, number = " << number
  <<endl; //Line 2
  funcValueParam(number); //Line 3
}
```

```

        cout<< "Line 4: After calling the function "
        << "funcValueParam, number = " << number
        <<endl; //Line 4 return
        0;
    }

    void funcValueParam (intnum)
    { cout<< "Line 5: In the function funcValueParam, "
        << "before changing, num = " <<num
        <<endl; //Line 5 num
        = 15; //Line 6 Value
        Parameters | 367
        cout<< "Line 7: In the function funcValueParam, "
        << "after changing, num = " <<num
        <<endl; //Line 7
    }
}

```

Sample Run:

```

Line 2: Before calling the function funcValueParam, number = 6
Line 5: In the function funcValueParam, before changing, num = 6
Line 7: In the function funcValueParam, after changing, num = 15
Line 4: After calling the function funcValueParam, number = 6

```

REFERENCE PARAMTERES:

When a reference parameter is passed in a function, it receives the address (memory location) of the actual parameter. Reference parameters can change the value of the actual parameter.

Reference parameters are useful in following situations.

- 1- When the value of the actual parameter needs to be changed.
- 2- When you want to return more than one value from a function.
- 3- When passing the address would save memory space and time relative to copying a large amount of data.

Example code for Reference Parameters:

```

//This program reads a course score and prints the
//associated course grade.
#include <iostream>using
namespace std;

void getScore (int& score); void
printGrade (intscore); intmain
()
{ intcourseScore;
    cout<< "Line 1: Based on the course score, \n"

```

```

        << " this program computes the "
        << "course grade." <<endl; //Line 1
        getScore(courseScore); //Line 2
        printGrade(courseScore); //Line 3
        return 0;
    }
    void getScore (int& score)
    { cout<< "Line 4: Enter course score: "; //Line 4
      cin>> score; //Line 5
      cout<<endl<< "Line 6: Course score is "
      << score <<endl; //Line 6
    }
    void printGrade (int cScore)
    { cout<< "Line 7: Your grade for the course is "; //Line 7
      if (cScore>= 90) //Line 8 cout<< "A." <<endl;
        else if (cScore>= 80)
          cout<< "B." <<endl;
        else if (cScore>= 70)
          cout<< "C." <<endl;
        else if (cScore>= 60)
          cout<< "D." <<endl;
        else cout<< "F." <<endl;
    }
}

```

Sample Run: In this sample run, the user input is shaded. Line 1: Based on the course score, this program computes the course grade.

Line 4: Enter course score: 85

Line 6: Course score is 85

Line 7: Your grade for the course is B.

Example code for Value & Reference Parameters:

```

//Example 7-6: Reference and value parameters
#include
<iostream>using
namespace std;

void funOne (inta, int& b, char v); void
funTwo (int& x, inty, char& w);

intmain ()
{ intnum1, num2; char
  ch; num1 = 10;
  //Line 1 num2 = 15;
  //Line 2 ch = 'A';
  //Line 3
  cout<< "Line 4: Inside main: num1 = " << num1

```

```

    << ", num2 = " << num2 << ", and ch = "
    <<ch<<endl; //Line 4 funOne
    (num1, num2, ch); //Line 5
    cout<< "Line 6: After funOne: num1 = " << num1
    << ", num2 = " << num2 << ", and ch = "
    <<ch<<endl; //Line 6 funTwo
    (num2, 25, ch); //Line 7
    cout<< "Line 8: After funTwo: num1 = " << num1
    << ", num2 = " << num2 << ", and ch = "
    <<ch<<endl; //Line 8
    return 0;
}

void funOne (inta, int& b, char v)
{ intone; one = a; //Line
  9 a++; //Line 10 b =
  b * 2; //Line 11 v =
  'B'; //Line 12
  cout<< "Line 13: Inside funOne: a = " << a
  << ", b = " << b << ", v = " << v
  << ", and one = " << one <<endl; //Line 13
}

void funTwo (int& x, inty, char& w)
{ x++; //Line 14 y = y *
  2; //Line 15 w =
  'G'; //Line 16
  cout<< "Line 17: Inside funTwo: x = " << x
  << ", y = " << y << ", and w = " << w
  <<endl; //Line 17
}

```

Sample Run:

```

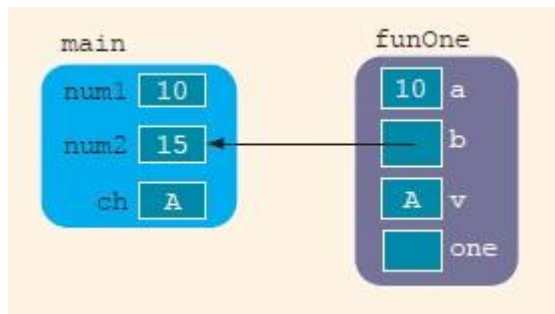
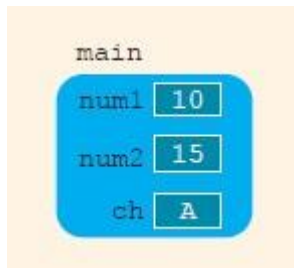
Line 4: Inside main: num1 = 10, num2 = 15, and ch = A
Line 13: Inside funOne: a = 11, b = 30, v = B, and one = 10
Line 6: After funOne: num1 = 10, num2 = 30, and ch = A
Line 17: Inside funTwo: x = 31, y = 50, and w = G
Line 8: After funTwo: num1 = 10, num2 = 31, and ch = G

```

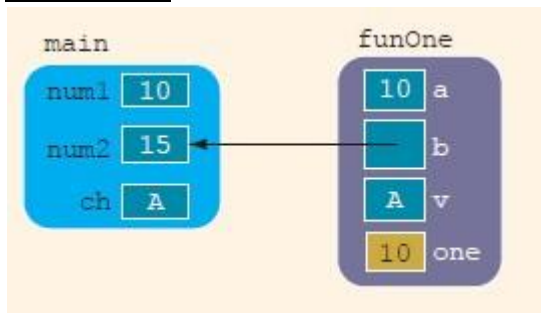
REPRESENTATION OF VARIABLES:

After Line 3:

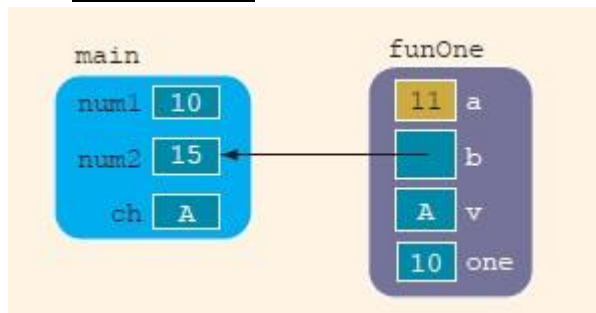
Before Line 9: (In Function funOne)



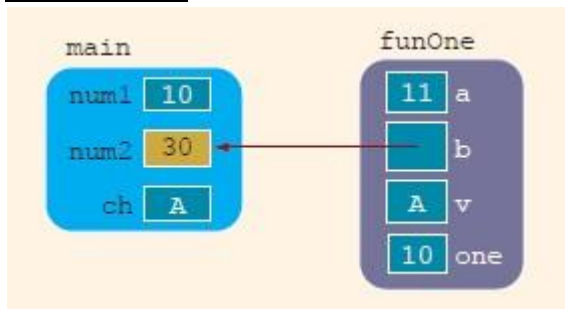
After Line 9:



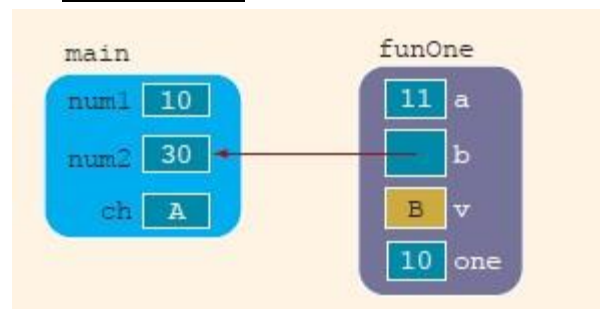
After Line 10:



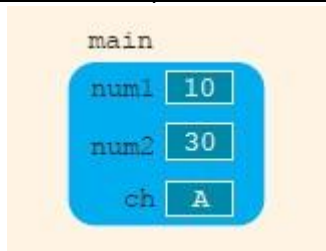
After Line 11:



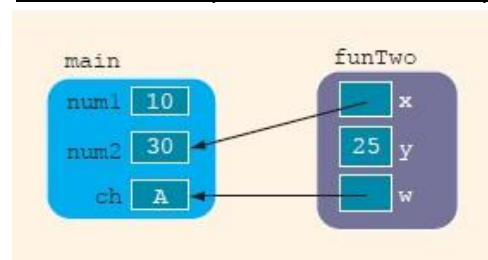
After Line 12:



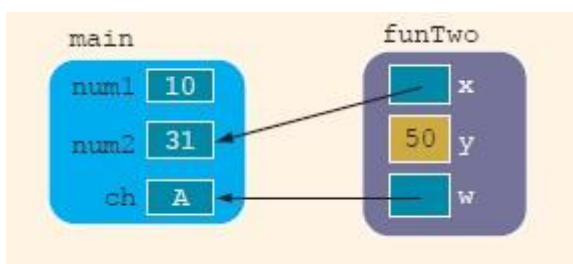
On Line 6: (When the function ends)



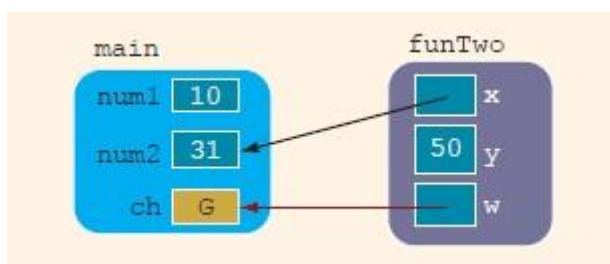
After Line 14: (In Function funTwo)



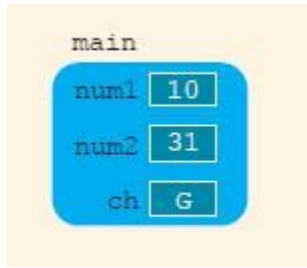
After Line 15:



After Line 16:



At Line 8 (When Function Ends):



4.SCOPE OF AN IDENTIFIER:

When talking about scope of an identifier we define it in two terms:

Local: Identifiers that are declared within a function or a block.

Global: Identifiers that are declared outside every function definition.

In general, the following rules apply when an identifier is accessed:

1. Global identifiers (such as variables) are accessible by a function or a block if:
 - a. The identifier is declared before the function definition (block),
 - b. The function name is different from the identifier,
 - c. All parameters of the function have names different than the name of the identifier, and
 - d. All local identifiers (such as local variables) have names different than the name of the identifier.
2. (Nested Block) An identifier declared within a block is accessible:
 - a. Only within the block from the point at which it is declared until the end of the block, and
 - b. By those blocks that are nested within that block if the nested block does not have an identifier with the same name as that of the outside block (the block that encloses the nested block).
3. The scope of a function name is similar to the scope of an identifier declared outside any block. That is, the scope of a function name is the same as the scope of a global variable.

Following program illustrates the rules of scope:

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

const double RATE = 10.50; intz;
double t; void one (intx, char y);
void two (inta, intb, char x); void
three (intone, double y, intz);
intmain () { intnum,
first; double x, y,
z; char name, last;
    ....
    return 0;
}
void one (intx, char y) {.....}
intw;
void two (inta, intb, char x)
{ intcount;
    ....
}
void three (intone, double y, intz)
{ charch;
    inta;
    ....
    //Block four
    { intx;
        char
        a;
        ....
    }//end Block four
    ....
}
```

Identifier	Visibility in one	Visibility in two	Visibility in three	Visibility in Block four	Visibility in main
RATE (before main)	Y	Y	Y	Y	Y
z (before main)	Y	Y	N	N	N
t (before main)	Y	Y	Y	Y	Y
main	Y	Y	Y	Y	Y
local variables of main	N	N	N	N	Y
one (function name)	Y	Y	N	N	Y
x (one's formal parameter)	Y	N	N	N	N
y (one's formal parameter)	Y	N	N	N	N
w (before function two)	N	Y	Y	Y	N
two (function name)	Y	Y	Y	Y	Y
a (two's formal parameter)	N	Y	N	N	N
b (two's formal parameter)	N	Y	N	N	N
x (two's formal parameter)	N	Y	N	N	N
local variables of two	N	Y	N	N	N
three (function name)	Y	Y	Y	Y	Y
one (three's formal parameter)	N	N	Y	Y	N
y (three's formal parameter)	N	N	Y	Y	N
z (three's formal parameter)	N	N	Y	Y	N
ch (three's local variable)	N	N	Y	Y	N
a (three's local variable)	N	N	Y	N	N
x (Block four's local variable)	N	N	N	Y	N
a (Block four's local variable)	N	N	N	Y	N

5.STATIC & AUTOMATIC VARIABLES:

- A variable for which memory is allocated at block entry and deallocated at block exit is called an **automatic variable**.
- A variable for which memory remains allocated as long as the program executes is called a **static variable**.

Global variables are static variables by default and variables declared in a block are automatic variables. Syntax for declaring a static variable is:

```
static dataType identifier;
```

Example Code for Static Variables:

```
//Program: Static and automatic variables
#include
<iostream>using
namespace std; void
test (); intmain ()
{ intcount;
    for (count = 1; count <= 5; count++) test
        ();
    return 0;
}
void test ()
{ staticintx = 0; inty
  = 10; x = x + 2;
  y = y + 1;
  cout<< "Inside test x = " << x << " and y = "
  << y <<endl;
}
```

Sample Run:

```
Inside test x = 2 and y = 11
Inside test x = 4 and y = 11
Inside test x = 6 and y = 11
Inside test x = 8 and y = 11
Inside test x = 10 and y = 11
```

LAB TASKS:

TASK – 01:

Your city bank has revised their credit card policy and have requested you to write a C program which will identify if a user has exceeded their credit limit or not. The program should ask the following information from the user

- a) Account number
- b) Balance at the beginning of the month
- c) List of all items purchased this month along with their charges, calculate the total charges
- d) Total of all credits applied to this customer's account this month
- e) Allowed credit limit

The program should input each fact, calculate the new balance = (beginning balance + total charges – credits), and determine whether the new balance exceeds the customer's credit limit. For the customer whose credit limit is exceeded, the program should display the customer's account number, credit limit, new balance and the message "Credit limit exceeded by 123.74."

All the users have purchased at least one product. You have the liberty to ask the user how many products they want to enter.

Sample input/output:

Enter account number (-1 to end): 100

Enter beginning balance: 4568.78

How many products you would like to Enter: 5

Enter charges for item 1: 150.00

Enter charges for item 2: 200.00

Enter the charges for item 3: 100.50

Enter the charges for item 4: 150.50

Enter the charges for item 5: 500.75

Your total: 1101.75

Enter total credits: 500.00

Enter credit limit: 5000.00

Account: 100

Credit limit: 5000.00

Balance: 4179.53

Credit Limit Not Exceeded.

TASK – 02:

Salesflow is one of leading software house they are starting their recruitment process for three different following positions:

1. Associate Developer
2. Assistant Developer
3. Trainee Engineer

There is defined criteria for recruitment process, if candidate clear the test with 50 marks then he will be select for the post of trainee engineer, experience are not the requirement in it. if candidate secured 60 marks with one or more than one year of experience and 70 marks with 2 year or more than two years of experience, then he/she will be select as assistant and associate developer respectively. Write a function that will take a test marks from user and ask for experience (if its entered marks \geq 60). After that, function will show the assigned position.

TASK – 03:

Write the program that calculate the volume . Create a two separate function

1. One function with the name of "getData (int h, int a)", that will take two input from user for the variable of "h" and "a".
2. Second function "volumeCalu ()" will calculate the volume and function must be called by first function "getData ()". Only first function call from the main function.

TASK - 04:

Given two points in a Cartesian plane Write a program to calculate the distance between them and midpoint . Your program must have at least the following functions:

- a. **distance:** This function takes as its parameters four numbers that represent two points in the plane and returns the distance between them.
- b. **midpoint:** This function takes as its parameter four numbers that represent two points in the plane and returns true if the midpoint has positive values for x, y both and returns false if any of the value is negative.

TASK – 05:

During the tax season, every Friday, J&J accounting firm provides assistance to people who prepare their own tax returns. Their charges are as follows.

- a) If a person has low income (\leq 25,000) and the consulting time is less than or equal to 30 minutes, there are no charges; otherwise, the service charges are 40% of the regular hourly rate for the time over 30 minutes.

- b) For others, if the consulting time is less than or equal to 20 minutes, there are no service charges; otherwise, service charges are 70% of the regular hourly rate for the time over 20 minutes.

(For example, suppose that a person has low income and spent 1 hour and 15 minutes, and the hourly rate is \$70.00. Then the billing amount is $70.00 \times 0.40 \times (45 / 60) = \21.00 .)

Write a program that prompts the user to enter the hourly rate, the total consulting time, and whether the person has low income. The program should output the billing amount. Your program must contain a function that takes as input the hourly rate, the total consulting time, and a value indicating whether the person has low income. The function should return the billing amount. Your program may prompt the user to enter the consulting time in minutes.