**National University of Computer and Emerging Sciences**

Logo

Description automatically generated with medium confidence

**Lab Manual 12**

**Object Oriented Programming**

|  |  |
| --- | --- |
| Course Instructor | Miss Abeeda |
| Lab Instructor (s) | Siddiqua Nayyer  Dilawar Shabbir |
| Section | A |
| Semester | Spring 2021 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

# Objectives:

Topics Covered in this Lab:

* Template Functions
* Template Functions and Overloading
* Template Specialization

**Function templates** are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one variable type or class without repeating the code for each type. This is achieved through template parameters. A template parameter is a special kind of parameter that can be used to pass a type as parameter. These function templates can use these parameters as if they were regular types. The format for declaring function templates with type parameters is:

**template <class identifier> function\_declaration;**

While defining a function template the body of the function definition is preceded by a statement **template <class identifier>.** The identifier can then be used as the data type of the parameters, the return type of the function, the data type of local variables and/or the data types of parameters.

**Exercise 0: Template Practice**

Consider the definition of the following function template:

template <class type>

type surprise(type x, type y)

{

return x + y;

}

Understand the code. What is the output of the following statements? Make a .cpp file to execute this code and add the output as comments in this file.

1. cout << surprise(5, 7) << endl;
2. string str1 = "Sunny";

string str2 = " Day";

cout << surprise(str1, str2) << endl;

**Exercise 1: Template Functions**

***Help: http://www.cplusplus.com/doc/oldtutorial/templates/***

Write Template function for performing arithmetic operation of type int, float, double, long. Main for this function is given below.

void main()

{

int a, b; // this can be float, int or double too

char op;

cout << "Enter first operand ";

cin >> a;

cout << "Enter second operand ";

cin >> b;

cout << "Enter operation ";

cin >> op; // op can be +, -, \* or /

if (op == '\*' || op == '+' || op == '-' || op == '/')

{

performOperation(a, b, op);

}

else

{

cout << "Wrong operation";

}

}

**Exercise 2: Template Function and Overloading**

1. Write two function templates GetMax and GetMin that take two arguments and return the maximum and minimum of the two respectively.
2. Then paste the following code in your source file and run the program. The program should run peacefully.

int main ()

{

int i=5, j=6, k;

long l=10, m=5, n;

k=GetMax<int>(i,j);

n=GetMin<long>(l,m);

cout << k << endl;

cout << n << endl;

return 0;

}

1. Now remove the <int> and <long> from the code above and execute again. Does the program still work?
2. Now replace the main function above with the main given below. You will need to change the code (declaration and definition) for GetMin and GetMax so that the following code works without an error.

int main ()

{

char i=’Z’;

int j=6, k;

long l=10, m=5, n;

k=GetMax<int,long>(i,m);

n=GetMin<int,char>(j,l);

cout << k << endl;

cout << n << endl;

return 0;

}

1. Now remove the <int,long> and <int,char> from this new main and re-run the program, is there any trouble with this version?
2. Now overload these template functions so that the maximum and minimum of 3 numbers could be found.

**Exercise 3: Template Specialization**

If we want to define a different implementation for a template when a specific type is passed as template parameter, we can declare a specialization of that template.

template <>

char\* maximum <char\*>(char\* x, char \* y)

{

if (strcmp(x,y)==1)

return x;

else

return y;

}

Consider a template function **increment**, that receives a variable (it can be int, double, float etc) and increase the value of that variable by 1.

Now Write a template specialization for char \* variables (character arrays) that convert all letters of character arrays to upper case.

Hint:

Lowercase characters ASCII range from 97 to 122. If the character is found to be in this range then the program converts that character into an uppercase character. ASCII of ‘A’ and ‘a’ differs by 32.