**Lab Sections**

1. Objectives
2. Introduction
3. Declaration Syntax
4. Experiments

|  |
| --- |
|  |

Function

Templates

**Function Templates**

1. **Objectives**

**After you complete this experiment you will be able to implement templates for functions and know when it is appropriate to implement templates for functions**

1. **Introduction**

Templates allow functions to use and operate on generic data types. They provide a mechanism to parameterize a data type. Whenever you notice that two or more functions have identical code but operate on data of different types, you should consider templates if possible.

1. **Declaration Syntax**

**a. To declare a class template with one type parameter:**

template <class Type\_Parameter>

function declaration

//temple <class Type\_Parameter> is referred to as the “template prefix” which

//informs the compiler that “Type\_Parameter” is a type parameter.

**b. To declare a class template with one type parameter (alternative form):**

template <typename Type\_Parameter>

function declaration

//This declaration operates identically to the first declaration.

**c. To declare a class template with multiple type parameters:**

template <class Type\_Parameter1, class Type\_Parameter2, ..., class Type\_Parameter)

function declaration

//This declaration contains several type parameters

Consider the following **function template** declaration for a function that checks to see if two items have the same value:

template <class a\_type>

bool Is\_Equal(a\_type a, a\_type b)

{

return a == b;

}

This function contains the type parameter “a\_type” which represents a type that has not been specified.

To call the function Is\_Equal consider the following:

string x=”XXXX”, y=”YYYY”;

cout<<Is\_Equal(x,y)<<endl;

More information on templates can be found in your course textbook and on the web.

1. **Experiments**

**Step 1: In this experiment you will investigate a program that contains two functions which are perfect candidates to be implemented as a function template. Enter, save, compile and execute the following program in MSVS. Call the new project “FunctionTemplateExp” and the program “FunctionTemplate.cpp”. Answer the questions below:**

#include <iostream>

#include <string>

using namespace std;

void swap(int &a, int &b)

{

int temp = a;

a = b;

b = temp;

}

void swap(string &a, string &b)

{

string temp = a;

a = b;

b = temp;

}

void swap(char &a, char&b)

{

char temp = a;

a = b;

b = temp;

}

int main( )

{

string x = "first", y = "second";

int m = 10, n = 20;

char q = 'Q', r = 'R';

cout<<"x before swap called = "<<x<<" and y before swap called = "<<y<<endl;

swap(x,y);

cout<<"x after swap called = "<<x<<" and y after swap called = "<<y<<endl<<endl;

cout<<"m before swap called = "<<m<<" and n before swap called = "<<n<<endl;

swap(m,n);

cout<<"m after swap called = "<<m<<" and n after swap called = "<<n<<endl<<endl;

cout<<"q before swap called = "<<q<<" and r before swap called = "<<r<<endl;

swap(q,r);

cout<<"q after swap called = "<<q<<" and r after swap called = "<<r<<endl<<endl;

return 0;

}

1. Please explain the output of the program in Step 1?
2. Please examine the three different swap functions in Step 1, and state any similarities and/or differences you observe?
3. Write a function template declaration for a function with the following characteristics:
4. the function name is “Tester3”;
5. the function return type is character;
6. there are two template parameters, a\_type and b\_type;
7. there are four formal parameters: x (a\_type), y (b\_type), w (b\_type) and m (b\_type).
8. Write a function template declaration replacing the three different swap functions in Step 1: