**Templates And Exceptions**

**LAB #** **10**

**Fall 2019**

**CSE208L Object Oriented Programming Lab**

Submitted by: **Shah Raza**

Registration No. : **18PWCSE1658**

Class Section: **B**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to:

**Engr. Sumayyea Salahuddin**

January 6, 2020

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

**Objectives of the Lab:**

Objectives of the lab are to:

# Develop generic function using function template.

# Develop generic class using class template.

* Understand various types of exceptions and exception handling mechanism.
* Use of try-catch blocks to handle exceptions.

# Activity # 01

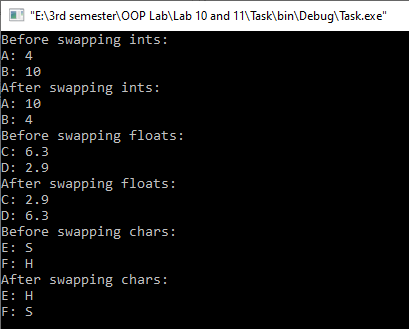
**Title:**

Create a template for Swap.

**In C++**

**Source code: Output:**

#include <iostream>

using namespace std;

template<typename T>

void Swap(T &a, T &b)

{

T temp= a;

a=b;

b=temp;

}

int main()

{

int a=4,b=10;

float c=6.3,d=2.9;

char e='S', f='H';

cout<<"Before swapping ints: \n";

cout<<"A: "<<a<<"\nB: "<<b<<endl;

Swap<int>(a,b);

cout<<"After swapping ints: \n";

cout<<"A: "<<a<<"\nB: "<<b<<endl;

cout<<"Before swapping floats: \n";

cout<<"C: "<<c<<"\nD: "<<d<<endl;

Swap<float>(c,d);

cout<<"After swapping floats: \n";

cout<<"C: "<<c<<"\nD: "<<d<<endl;

cout<<"Before swapping chars: \n";

cout<<"E: "<<e<<"\nF: "<<f<<endl;

Swap<char>(e,f);

cout<<"After swapping chars: \n";

cout<<"E: "<<e<<"\nF: "<<f<<endl;

return 0;

}

# Activity # 02

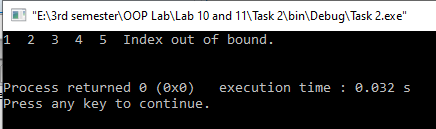
**Title:**

Create a class called SmartArray that can store data of any type and generates exception when it overflows.

**In C++**

**Source code: Output:**

#include <iostream>



using namespace std;

template<class T>

class SmartArray

{

private:

T \*Array;

int SIZE;

public:

SmartArray()

{

SIZE=5;

Array= new T[SIZE];

}

SmartArray(int Size)

{

SIZE=Size;

Array=new T[SIZE];

}

T &Access(int index)

{

if(index<0 || index>=SIZE)

throw "Index out of bound.\n";

else

return Array[index];

}

~SmartArray()

{

delete Array;

}

};

int main()

{

SmartArray<int> Arr;

SmartArray<char> CArr(6);

try

{

for (int i=0;i<5;i++)

Arr.Access(i)=i+1;

for (int i=0;i<6;i++)

cout<<Arr.Access(i)<<" ";

}

catch(const char \*msg)

{

cerr<<msg<<endl;

}

return 0;

}

# Activity # 03

**Title:**

Create a class Rectangle. The class has attributes length and width, each of which defaults to 1. Provide methods that calculate the perimeter and the area of the rectangle. Provide set and get methods for both length and width. The set methods should verify that length and width are each floating-point numbers greater than or equal to 0.0 and less than 20.0. Write a program to test class Rectangle.

**In C++**

**Source code:**

#include <iostream>

using namespace std;

class Rectangle

{

private:

float length,width;

public:

Rectangle():length(1),width(1){}

float Perimeter(){return 2\*(length+width);}

float Area(){return length\*width;}

float GetL(){return length;}

float GetW(){return width;}

void SetL(float l)

{

if(l>=0.0 && l<=20.0)

length=l;

else

throw"The length should be in the range 0-20.";

}

void SetW(float w)

{

if(w>=0.0 && w<=20.0)

width=w;

else

throw"The width should be in the range 0-20.";

}

};

int main()

{

Rectangle r;

try

{

r.SetL(12.4);

r.SetW(20.8);

}

catch(const char \*msg)

{

cerr<<msg<<endl;

}

cout<<"Length: "<<r.GetL()<<endl;

cout<<"Width: "<<r.GetW()<<endl;

cout<<"Area: "<<r.Area()<<endl;

cout<<"Perimeter: "<<r.Perimeter();

return 0;

}

**Output:**

