TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PURWANCHAL CAMPUS

DHARAN

LABSHEET 9 : [TEMPLATES AND EXCEPTION]

BY

Rabin Poudel [PUR079BCT056]

DEPARTMENT OF COMPUTER & ELECTRONICS ENGINEERING

PURWANCHAL CAMPUS

DHARAN, NEPAL

JULY, 2024

0.1 Question 1 :

Create a function called sum ( ) that returns the sum of the elements of an array. Make this function into a template so it will work with any numerical type. Write a main ( ) program that applies this function to data of various type.

Code:

```
#include <iostream>
#define SUCCESS 0
using namespace std;

template < typename T>
T sum(T array[],int n)
{
   T s= 0;
   for(int i = 0 ; i < n; i++)
   {
      s+=array[i];
   }
   return s;
}
int main()
{
   int num[] = {4,5,6};
   float fnum[] = {4.0,3.0,5.5};
   cout << sum(num,3) << endl;
   cout << sum(fnum,3) << endl;

   return SUCCESS;
}
```

0.2 Question 2:

Write a class template for queue class. Assume the programmer using the queue won't make mistakes, like exceeding the capacity of the queue, or trying to remove an item when the queue is empty. Define several queues of different data types and insert and remove data from them.

Code:

```
#include <iostream>
#define SUCCESS 0
using namespace std;

template <typename T>
class Queue
{
 private:
   T data[100];
   int pos;
 public:
   Queue()
   {
      pos = 0;
      for(int i = 0; i< 100; i++)
         data[i]=0;
   }
   void add(T d)
   {
      data[pos] = d;
      pos++;

   }
   T get()
   {
      T   d = data[0];
      for(int i = 0 ; i < pos; i++)
      {
         data[i] = data[i+1];
```

```
        }
        pos−−;
        return d;
    }
};
int main()
{
    Queue<int> intlist;
    intlist.add(3);
    intlist.add(4);
    cout << intlist.get() << endl;
    cout << intlist.get() << endl;
    Queue<float> flist;
    flist.add(3.0);
    flist.add(4.9);
    cout << flist.get() << endl;
    cout << flist.get() << endl;
    return SUCCESS;
}
```

0.3 Question 3:

Modify the stack class given in the previous lab to add the exception when user tries to add item while the stack is full and when user tries to add item while the stack is empty. Throw exception in both of the cases and handle these exceptions.

Code:

```
#include<iostream>
#include<cstring>
#define SUCCESS 0;
using namespace std;
#define SIZE 2
class exc{
  public:
    string info;
};
template<class T>
class Stack
{
  private:
    T st[SIZE];
    int top;
  public:
    Stack();
    void push(T var);
    T pop();
};

template<class T>
Stack<T>::Stack()
{
    top=−1;
}

template<class T>
void Stack<T>::push(T var)
{
    try
    {
        if(top >= (SIZE−1))
        {
            top = SIZE − 1;
            exc   e;
            e.info="Stack is full";
            throw   e;
        }
        else
        {
            st[++top]=var;
        }
    }
    catch(exc e)
    {
        cerr << e.info << endl;
    }
}

template<class t>
t Stack<t>::pop()
{
    try {
        if (top < 0)
        {
            exc e;
            e.info ="stack is empty";
            throw e;
        }
        else
        {
            return st[top−−];
        }
    }
    catch(exc e)
    {
        cerr << e.info << endl;
    }
```

```
}

int main()
{
   Stack<float> s1;

   s1.push(111.1F);
   s1.push(222.2F);
   s1.push(333.3F);

   cout<<"1 : "<<s1.pop()<<endl;
   cout<<"2 : "<<s1.pop()<<endl;
   cout<<"3 : "<<s1.pop()<<endl;

   Stack<long> s2;

   s2.push(123123123L);
   s2.push(234234234L);
   s2.push(345345345L);

   cout<<"1 : "<<s2.pop()<<endl;
   cout<<"2 : "<<s2.pop()<<endl;
   cout<<"3 : "<<s2.pop()<<endl;

   return SUCCESS;
}
```

0.4 Question 4:

Write any program that demonstrates the use of multiple catch handling, re-throwing an exception, and catching all exception.

Code:

```
#include <iostream>
#define SUCCESS 0
using namespace std;
class DIVZERO{};
class DIVMINUS{};
int main()
{
   int a, b;
   float ans;
   try {
      cout << "a";
      cin >> a;
      cout << "b";
      cin >> b;

      try {
         if(b < 0)
            throw DIVMINUS();
         if(b == 0)
            throw DIVZERO();
         ans = a/b;
      }
      catch (DIVZERO)
      {
         cerr << "rethrowing DIVZERO exception" << endl;
         throw;
      }
      catch (DIVMINUS)
      {
         cerr << "divison by minus in not allowed"<< endl;
      }
   } catch (...) {
      cerr << "caught exception";
   }
   cout << ans;
   return SUCCESS;
}
```