HO CHI MINH UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY
SOFTWARE ENGINEERING DEPARTMENT
ADVANCED PROGRAM IN COMPUTER SCIENCE
COURSE: **PROGRAMMING SYSTEMS**
LECTURER: Dr. ĐINH BÁ TIẾN

WEEK 08

TEMPLATE – EXCEPTION

- TRƯƠNG PHƯỚC LỘC
- HỒ TUẤN THANH

HCMC, November 24, 2016

TABLE OF CONTENTS

1 Template

1.1 Function Template

```
template<class T>
T findMax(T x, T y);
```

```
template<class T>
T findMax(T x, T y)
{
    if(x>y)
        return x;
    return y;
}
```

```
void main()
{
    int a1=10;
    int b1=20;
    cout<<findMax(a1,b1)<<endl;

    double a2=10.1;
    double b2=-2.3;
    cout<<findMax(a2,b2)<<endl;
}
```

1.1.1 Problem 01: Dividing into separate files

File Func.h:

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

template<class T>
T findMax(T x, T y);
```

File Func.cpp:

```
#include "Func.h"

template<class T>
T findMax(T x, T y)
{
    if(x>y)
        return x;
    return y;
}
```

File main.cpp:

```
#include "Func.h"

void main()
{
    int a1=10;
    int b1=20;
    cout<<findMax(a1,b1)<<endl;

    double a2=10.1;
    double b2=-2.3;
    cout<<findMax(a2,b2)<<endl;
}
```

You've the following errors:

Error List

3 Errors | 0 Warnings | 0 Messages

	Description
1	error LNK2019: unresolved external symbol "double __cdecl findMax<double>(double,double)" (??$findMax@N@@YANNN@Z) referenced in function _main
2	error LNK2019: unresolved external symbol "int __cdecl findMax<int>(int,int)" (??$findMax@H@@YAHHH@Z) referenced in function _main
3	fatal error LNK1120: 2 unresolved externals

A solution:

```
#include "Func.cpp"

void main()
{
    int a1=10;
    int b1=20;
    cout<<findMax(a1,b1)<<endl;

    double a2=10.1;
    double b2=-2.3;
    cout<<findMax(a2,b2)<<endl;
}
```

A better solution:

- At the end of file Func.cpp:

```
template
int findMax(int x, int y);

template
double findMax(double x, double y);
```

1.1.2 Problem 02: User-defined types

Suppose that, you defined class Fraction:

```
#pragma once

class Fraction
{
private:
    int nu;
    int de;
public:
    Fraction(void);
    ~Fraction(void);
};
```

File main.cpp:

```
void main()
{
    int a1=10;
    int b1=20;
    cout<<findMax(a1,b1)<<endl;

    double a2=10.1;
    double b2=-2.3;
    cout<<findMax(a2,b2)<<endl;

    Fraction f1;
    Fraction f2;
    cout<<findMax(f1,f2)<<endl;
}
```

You've got some compile-time errors because:

- You've not defined operator > of class Fraction:

```
template<class T>
T findMax(T x, T y)
{
    if(x>y)
        return x;
    return y;
}
```

- You've not defined operator << of class Fraction:

```
void main()
{
    int a1=10;
    int b1=20;
    cout<<findMax(a1,b1)<<endl;

    double a2=10.1;
    double b2=-2.3;
    cout<<findMax(a2,b2)<<endl;

    Fraction f1;
    Fraction f2;
    cout<<findMax(f1,f2)<<endl;
}
```

Solution: Define operator > and operator <<:

```
int operator>(Fraction f);
friend ostream& operator<<(ostream& os, Fraction f);
```

```
int Fraction::operator>(Fraction f)
{
    int delta=nu*f.de-de*f.nu;
    if(delta>0)
        return 1;
    return 0;
}
ostream& operator<<(ostream& os, Fraction f)
{
    os<<f.nu<<"/"<<f.de;
    return os;
}
```

```
template
Fraction findMax(Fraction x, Fraction y);
```

```
void main()
{
    int a1=10;
    int b1=20;
    cout<<findMax(a1,b1)<<endl;

    double a2=10.1;
    double b2=-2.3;
    cout<<findMax(a2,b2)<<endl;

    Fraction f1;
    Fraction f2;
    cout<<findMax(f1,f2)<<endl;
}
```

1.1.3 Problem 03: Particular circumstances

Main.cpp:

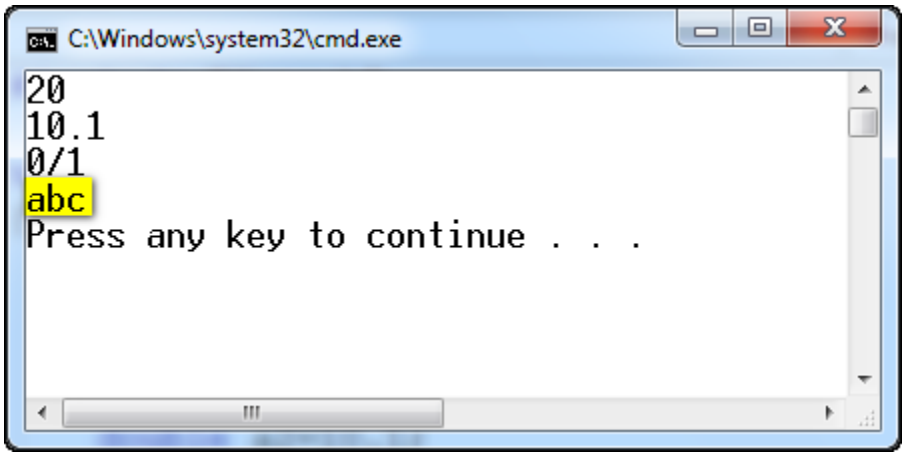
```
void main()
{
    int a1=10;
    int b1=20;
    cout<<findMax(a1,b1)<<endl;

    double a2=10.1;
    double b2=-2.3;
    cout<<findMax(a2,b2)<<endl;

    Fraction f1;
    Fraction f2;
    cout<<findMax(f1,f2)<<endl;

    char str1[]="abc";
    char str2[]="def";
    cout<<findMax(str1,str2)<<endl;
}
```

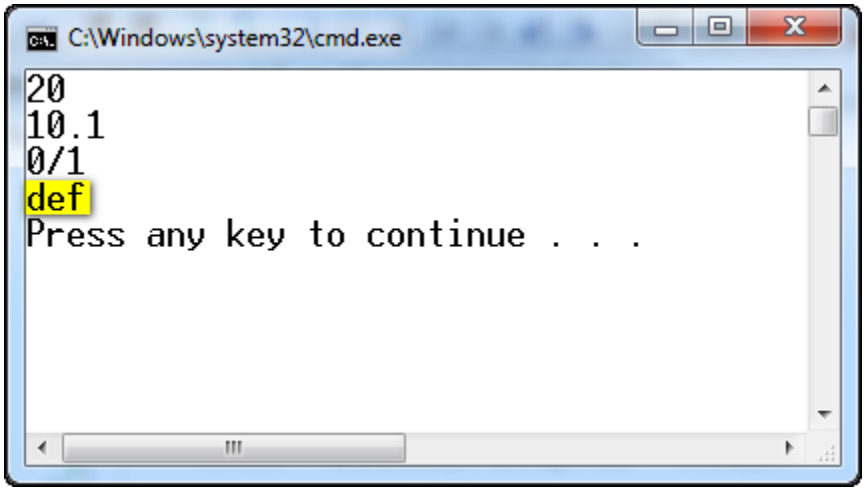
Output:



Solution:

```
template <>
char* findMax(char *x, char * y)
{
    int res=stricmp(x,y);
    if(res>0)
        return x;
    return y;
}
```

Here is the result:



1.2 Class Template

TwoEleClass.h:

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

template<class T>
class TwoEleClass
{
private:
    T x;
    T y;
public:
    TwoEleClass(void);
    ~TwoEleClass(void);
    TwoEleClass (T _x, T _y);
    T findMax();
};
```

TwoEleClass.cpp:

```
#include "TwoEleClass.h"

template<class T>
TwoEleClass<T>::TwoEleClass(void)
{
}

template<class T>
TwoEleClass<T>::~TwoEleClass(void)
{
}

template<class T>
TwoEleClass<T>::TwoEleClass (T _x, T _y)
{
    x=_x;
    y=_y;
}

template<class T>
T TwoEleClass<T>::findMax()
{
    if(x>y)
        return x;
    return y;
}

template class TwoEleClass<int>;
```

Main.cpp:

```
void main()
{
    TwoEleClass<int> t(5,6);
    cout<<t.findMax()<<endl;

}
```

2 Exception

Class Natural Number:

```
class NaturalNumber
{
private:
    int num;
public:
    NaturalNumber(void);
    ~NaturalNumber(void);

    void SetNum(int value);

    NaturalNumber Divide(const NaturalNumber &n);

    friend istream& operator>>(istream&is, NaturalNumber &n);

    friend ostream& operator<<(ostream&os, const NaturalNumber &n);

    NaturalNumber Substract(const NaturalNumber &n);
};
```

```
NaturalNumber::NaturalNumber(void)
{
    num=0;
}
```

```
void NaturalNumber::SetNum(int value)
{
    if(value<0)
    {
        throw new NegativeNumberException;
    }
    else
    {
        num=value;
    }
}
```

```
NaturalNumber NaturalNumber::Divide(const NaturalNumber &n)
{
    NaturalNumber kq;
    if(n.num==0)
    {
        throw new DividedByZeroException;
    }
    else
    {
        kq.num=num/n.num;
    }
    return kq;
}
```

```
istream& operator>>(istream&is, NaturalNumber &n)
{
    cout<<"Enter a natural number: ";
    int x;
    is>>x;
    try
    {
        n.SetNum(x);
    }
    catch(NegativeNumberException* ex)
    {
        cout<<"ERROR: "<<ex->GetErrorString()<<endl;
        n.SetNum(0);
    }
    return is;
}
```

```
ostream& operator<<(ostream&os, const NaturalNumber &n)
{
    os<<n.num;
    return os;
}

NaturalNumber NaturalNumber::Substract(const NaturalNumber &n)
{
    NaturalNumber kq;
    if(num<n.num)
    {
        throw new Num1LessThanNum2Exception;
    }
    else
    {
        kq.num=num-n.num;
    }
    return kq;
}
```

Class DividedByZeroException:

```
class DividedByZeroException
{
public:
    DividedByZeroException(void);
    ~DividedByZeroException(void);

    char* GetErrorString();
};
```

```
char* DividedByZeroException::GetErrorString()
{
    return "Divided by zero";
}
```

Class NegativeNumberException:

```
class NegativeNumberException
{
public:
    NegativeNumberException(void);
    ~NegativeNumberException(void);

    char* GetErrorString();
};
```

```
char* NegativeNumberException::GetErrorString()
{
    return "Negative number";
}
```

Class Num1LessThanNum2Exception:

```
class Num1LessThanNum2Exception
{
public:
    Num1LessThanNum2Exception(void);
    ~Num1LessThanNum2Exception(void);

    char* GetErrorString();
};
```

```
char* Num1LessThanNum2Exception::GetErrorString()
{
    return "Number 1 < Number 2";
}
```

Main.cpp:

```
void main()
{
    try
    {
        NaturalNumber n1;
        n1.SetNum(10);

        NaturalNumber n2;
        n2.SetNum(7);

        NaturalNumber n3;
        n3=n1.Substract(n2);

        cout<<"N3 = "<<n3<<endl;

        n2.SetNum(0);
        cout<<n1.Divide(n2)<<endl;
    }
    catch(NegativeNumberException* ex1)
    {
        cout<<"ERROR: "<<ex1->GetErrorString()<<endl;
    }
    catch(Num1LessThanNum2Exception * ex2)
    {
        cout<<"ERROR: "<<ex2->GetErrorString()<<endl;
    }
    catch(DividedByZeroException * ex3)
    {
        cout<<"ERROR: "<<ex3->GetErrorString()<<endl;
    }
}
```

3 Exercises

Solution Name: your student ID.

Project Names: Exercise01, Exercise02.

File name of your submission: your student ID.rar/zip. Remove all folders Debug and file *.ncb before submitting to Moodle.

3.1 Exercise 01

Write the following functions (don't use class):

1. Input an array of elements
2. Output the array to screen
3. Calculate the sum of elements
4. Find the minimum element
5. Find the frequency of x in the array

Test the above functions with:

1. Array of integer numbers
2. Array of float numbers
3. Array of fractions
4. Array of strings

3.2 Exercise 02

Implement class TArray (dynamic allocated array) with the following methods and operators:

1. Operator <<
2. Operator >>
3. Operator []: get/set the value at position idx
4. Calculate the sum of elements
5. Find the minimum element in the array
6. Divide all elements by x

Requirements:

- Test with:
  - Array of integer numbers
  - Array of float numbers
  - Array of Fraction
- Class Template
- Throw and try catch all exceptions