Assignment 9

191220022 丁一凡

一、 概念简答题

1.

1. 函数模板在使用时，必须要进行实例化
2. 错误，有时编译程序无法根据调用时的实参类型来确定所调用的模板函数，需要在程序中显式实例化
3. 错误，只有当一个类的成员类型可变时，才是类模板
4. 正确

2.

类模板中的静态成员仅属于实例化后的类（模板类），不同类模板实例之间不共享类模板中的静态成员

3.

在C++中，源文件（模块）是分别编译的，如果在一个源文件中定义和实现了一个模板，但在该源文件中未使用到该模板的某个实例，则在相应的目标文件中，编译程序不会生产该模板相应实例的代码

二、 代码编程题

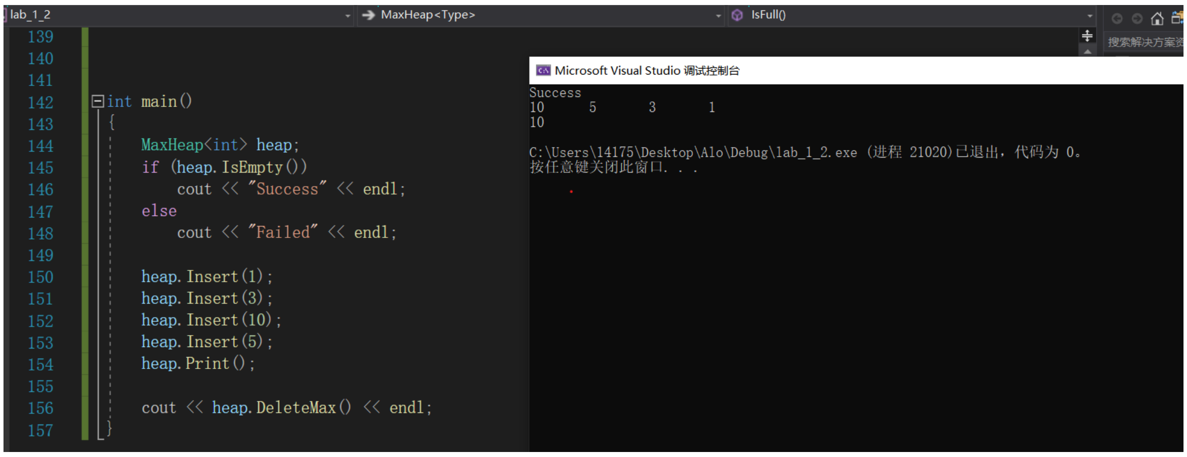
1.

```
template<class Type>
class MaxHeap {
private:
    Type* Data;
    int Size; //当前大小
    int Capacity; //总容量
    void shiftUp(int start);    //调整堆的私有函数
    void shiftDown(int start, int m);   //调整堆的私有函数
public:
    MaxHeap(); //默认构造函数，容量为10
    MaxHeap(int Capacity);
    ~MaxHeap();
    bool Insert(Type element); //插入一个元素
    Type DeleteMax(); //找出最大的元素返回，并进行删除
    bool IsFull(); //是否为满
    bool IsEmpty(); //是否为空
```

```cpp
    void Print(); //打印
};

template <class Type>
MaxHeap <Type>::MaxHeap()
{
    Capacity = 10;
    Data = new Type[Capacity];
    Size = 0;
}

template <class Type>
MaxHeap <Type>::MaxHeap(int Capacity)
{
    this->Capacity = Capacity;
    Data = new Type[Capacity];
    Size = 0;
}

template <class Type>
MaxHeap <Type>::~MaxHeap()
{
    delete[]Data;
}

template <class Type>
bool MaxHeap <Type>::Insert(Type element)
{
    if (Size == Capacity)
    {
        cerr << "Heap Full!!!" << endl;
        return false;
    }
    Data[Size] = element;
    shiftUp(Size);
    Size++;
    return true;
}

template <class Type>
void MaxHeap <Type>::shiftUp(int start)
{
    int j = start, i = (j - 1) / 2;
    Type temp = Data[j];
    while (j > 0)
    {
        if (Data[i] >= temp)
            break;
        else
        {
            Data[j] = Data[i];
            j = i;
            i = (i - 1) / 2;
        }
    }
    Data[j] = temp;
}
```

```cpp
template <class Type>
Type MaxHeap <Type>::DeleteMax()
{
    if (Size == 0)
    {
        cerr << "Heap Empty!!!" << endl;
        exit(-1);
    }
    Type result = Data[0];
    Data[0] = Data[Size - 1];
    Size--;
    shiftDown(0, Size - 1);
    return result;
}

template <class Type>
void MaxHeap <Type>::shiftDown(int start, int m)
{
    int i = start, j = 2 * i + 1;
    Type temp = Data[i];
    while (j <= m)
    {
        if (j < m && Data[j] < Data[j + 1])
            j++;
        if (temp >= Data[j])
            break;
        else
        {
            Data[i] = Data[j];
            i = j;
            j = 2 * j + 1;
        }
    }
    Data[i] = temp;
}

template <class Type>
bool MaxHeap <Type>::IsEmpty()
{
    return Size == 0;
}

template <class Type>
bool MaxHeap <Type>::IsFull()
{
    return Size == Capacity;
}

template <class Type>
void MaxHeap <Type>::Print()
{
    for (int i = 0; i < Size; i++)
    {
        cout << Data[i] << "    ";
    }
    cout << endl;
}
```

测试用例及运行截图：



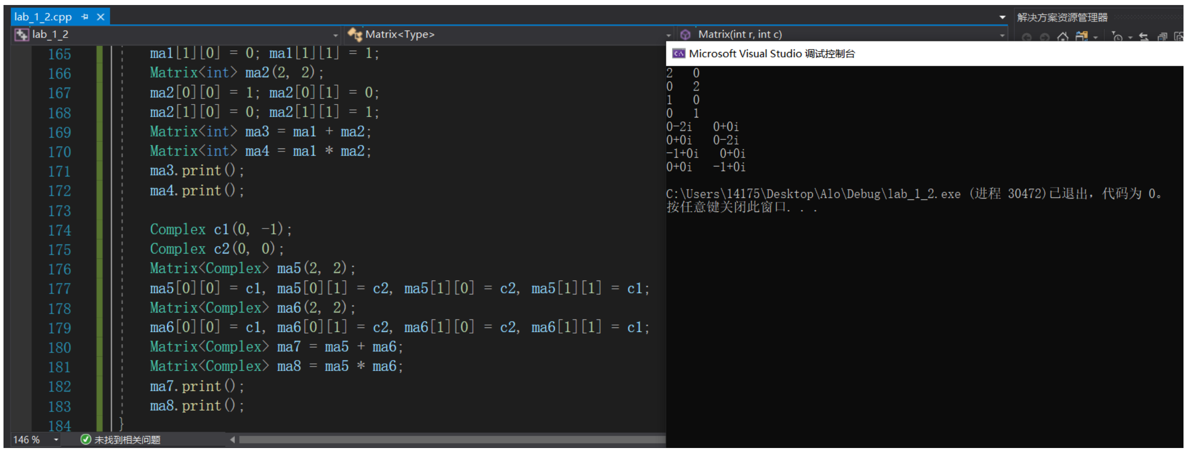
```
int main()
{
    MaxHeap<int> heap;
    if (heap.IsEmpty())
        cout << "Success" << endl;
    else
        cout << "Failed" << endl;

    heap.Insert(1);
    heap.Insert(3);
    heap.Insert(10);
    heap.Insert(5);
    heap.Print();

    cout << heap.DeleteMax() << endl;
}
```

```
Success
10      5       3       1
10

C:\Users\14175\Desktop\Alo\Debug\lab_1_2.exe (进程 21020)已退出，代码为 0。
按任意键关闭此窗口. . .
```

2.

```
template <class Type>
class Matrix
{
    vector<vector<Type>> p_data;
    int row, col;
public:
    Matrix(int r, int c)
    {
        row = r;
        col = c;

        for (int i = 0; i < row; i++)
        {
            vector<Type> temp(col);//, ((void*)0));
            /*for (int j = 0; j < col; j++)
            {
                Type t;
                temp.push_back(t);
            }*/
            p_data.push_back(temp);
        }
    }
    ~Matrix()
    {
        p_data.clear();
    }
    vector<Type>& operator[] (int i)
    {
        return p_data[i];
    }
    Matrix& operator = (const Matrix& m)
    {
        if (&m == this)
            return *this;
        row = m.row, col = m.col;
        p_data.clear();
        for (int i = 0; i < row; i++)
        {
            vector<Type> temp(col, 0);
            /*for (int j = 0; j < col; j++)
            {
                Type t = new Type;
                temp.push_back(t);
            }*/
            p_data.push_back(temp);
        }
        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                p_data[i][j] = m.p_data[i][j];
            }
        }
    }
    bool operator ==(const Matrix& m) const
    {
        if (row != m.row || col != m.col)
            return false;
        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                if (p_data[i][j] != m.p_data[i][j])
                    return false;
            }
        }
        return true;
    }
    Matrix operator + (Matrix& m)
    {
        Matrix temp(row, col);
        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                temp[i][j] = p_data[i][j] + m.p_data[i][j];
            }
        }
        return temp;
    }
    Matrix operator * ( Matrix& m)
    {
        Matrix temp(row, m.col);
        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < m.col; j++)
            {
                for (int k = 0; k < col; k++)
                {
                    temp.p_data[i][j] = p_data[i][k] * m.p_data[k][j] +
temp.p_data[i][j];
                }
            }
        }
        return temp;
    }
    void print() const
    {
        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                cout << p_data[i][j] << "    ";
            }
            cout << endl;
        }
    }
};

class Complex
{
    int real, imaginary;
public:
    Complex() { real = imaginary = 0; }
    Complex(int r, int i) { real = r; imaginary = i; }
    Complex operator +(const Complex& c)
    {
        Complex result;
        result.real = real + c.real;
        result.imaginary = imaginary + c.imaginary;
        return result;
    }
    Complex operator *(const Complex& c)
    {
        Complex result;
        result.real = real * c.real - imaginary * c.imaginary;
        result.imaginary = imaginary * c.real + real * c.imaginary;
        return result;
    }
    Complex& operator =(const Complex& c)
    {
        if (&c == this)
            return *this;
        this->imaginary = c.imaginary;
        this->real = c.real;
        return *this;
    }
    friend ostream& operator << (ostream& os, const Complex& c);
};

ostream& operator << (ostream& os, const Complex& c)
{
    cout << c.real;
    if (c.imaginary < 0)
    {
        cout << c.imaginary << "i";
    }
    else
    {
        cout << "+" << c.imaginary << "i";
    }
    return os;
}
```

测试用例及运行结果截图：



```cpp
        ma1[1][0] = 0; ma1[1][1] = 1;
        Matrix<int> ma2(2, 2);
        ma2[0][0] = 1; ma2[0][1] = 0;
        ma2[1][0] = 0; ma2[1][1] = 1;
        Matrix<int> ma3 = ma1 + ma2;
        Matrix<int> ma4 = ma1 * ma2;
        ma3.print();
        ma4.print();

        Complex c1(0, -1);
        Complex c2(0, 0);
        Matrix<Complex> ma5(2, 2);
        ma5[0][0] = c1, ma5[0][1] = c2, ma5[1][0] = c2, ma5[1][1] = c1;
        Matrix<Complex> ma6(2, 2);
        ma6[0][0] = c1, ma6[0][1] = c2, ma6[1][0] = c2, ma6[1][1] = c1;
        Matrix<Complex> ma7 = ma5 + ma6;
        Matrix<Complex> ma8 = ma5 * ma6;
        ma7.print();
        ma8.print();
    }
```

```
2   0
0   2
1   0
0   1
0-2i   0+0i
0+0i   0-2i
-1+0i   0+0i
0+0i   -1+0i

C:\Users\14175\Desktop\Alo\Debug\lab_1_2.exe (进程 30472)已退出，代码为 0。
按任意键关闭此窗口. . .
```