《C++ 程序设计》OJ5（参考答案）

VCG

2023 年 5 月 27 日

C++ Programming

1 TSwap

【描述】

用模板函数实现对不同类型的数据进行交换。
并使用如下主函数测试。

```
int main()
{
    int a1, a2;
    std::cin >> a1 >> a2;
    Swap(a1, a2);
    std::cout << a1 << "," << a2 << std::endl;

    double b1, b2;
    std::cin >> b1 >> b2;
    Swap(b1, b2);
    std::cout << b1 << "," << b2 << std::endl;

    char c1, c2;
    std::cin >> c1 >> c2;
    Swap(c1, c2);
    std::cout << c1 << "," << c2 << std::endl;

    return 0;
}
```

【输入描述】

输入共三行。第一行两整数，第二行两浮点数，第三行两字符

【输出描述】

输出共三行。每一行为对应输入处理后的结果，输出的两个数用逗号隔开

【输入样例】

```
2 3
1.2 2.3
a b
```

【输出样例】

```
3,2
2.3,1.2
b,a
```

【提示】

main.cpp

```cpp
#include <iostream>

template <class T>
void Swap(T & a, T & b)
{
    T temp = a;
    a = b;
    b = temp;
}

int main()
{
    int a1, a2;
    std::cin >> a1 >> a2;
    Swap(a1, a2);
    std::cout << a1 << "," << a2 << std::endl;

    double b1, b2;
    std::cin >> b1 >> b2;
    Swap(b1, b2);
    std::cout << b1 << "," << b2 << std::endl;

    char c1, c2;
    std::cin >> c1 >> c2;
    Swap<char>(c1, c2);
    std::cout << c1 << "," << c2 << std::endl;

    return 0;
}
```

2 TSort

【描述】

用模板函数实现数组的输入、排序和输出。并使用如下主函数测试你的模板

```
int main()
{
    const int LEN = 5;
    int type;
    while (std::cin >> type)
    {
        switch (type)
        {
            case 0:
            {
                int a1[LEN];
                Input<int>(a1, LEN); Sort<int>(a1, LEN); Output<int>(a1, LEN);
                break;
            }
            case 1:
            {
                char a2[LEN];
                Input(a2, LEN); Sort(a2, LEN); Output(a2, LEN);
                break;
            }
            case 2:
            {
                double a3[LEN];
                Input(a3, LEN); Sort(a3, LEN); Output(a3, LEN);
                break;
            }
        }
    }

    return 0;
}
```

【输入描述】

输入包含多组测试数据。
每组数据为两行，第一行整数 type(0、1、2)。第二行为相应数组的 5 个元素。

【输出描述】

对于每一组测试数据，将其排序后在一行内输出，相邻元素逗号空格分离，最后为换行。

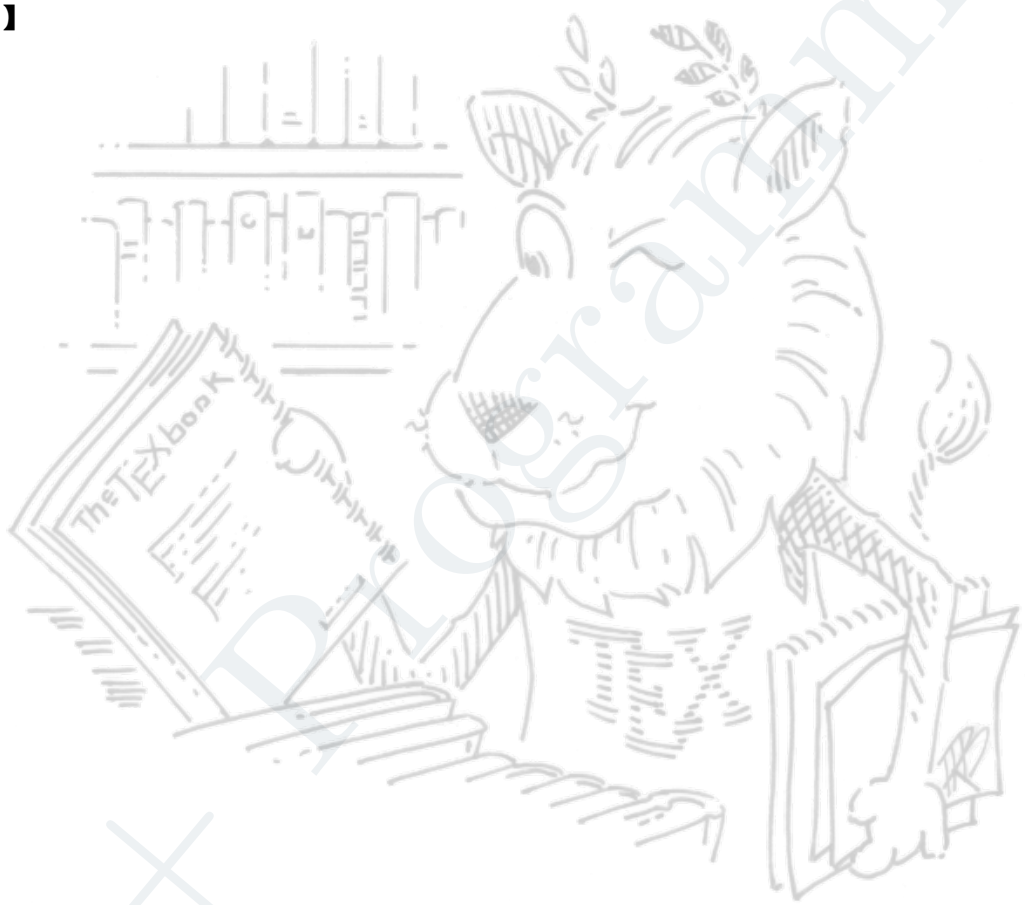
【输入样例】

```
0
3 6 1 4 5
1
A B C B A
```

【输出样例】

```
1, 3, 4, 5, 6
A, A, B, B, C
```

【提示】



C++ Programming

main.cpp

```cpp
#include <iostream>

template <class T>
void Input(T *array, int n)
{
    for (int i = 0; i < n; i++)
    {
        std::cin >> array[i];
    }
}

template <class T>
void Sort(T *array, int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (array[i] > array[j])
            {
                T temp;
                temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }
    }
}

template <class T>
void Output(const T *array, int n)
{
    for (int i = 0; i < n; i++)
    {
        std::cout << array[i];
        if(i>=n-1)
            std::cout << std::endl;
        else
            std::cout << ", ";
    }
}

int main()
{
    const int LEN = 5;
    int type;
    while (std::cin >> type)
    {
```

```
49          switch (type)
50          {
51              case 0:
52              {
53                  int a1[LEN];
54                  Input<int>(a1, LEN); Sort<int>(a1, LEN); Output<int>(a1, LEN);
55                  break;
56              }
57              case 1:
58              {
59                  char a2[LEN];
60                  Input(a2, LEN); Sort(a2, LEN); Output(a2, LEN);
61                  break;
62              }
63              case 2:
64              {
65                  double a3[LEN];
66                  Input(a3, LEN); Sort(a3, LEN); Output(a3, LEN);
67                  break;
68              }
69          }
70      }
71
72      return 0;
73  }
```

3 TSearch

【描述】

设计一个函数模板，实现在一个给定的数组中查找给定数据是否存在，如果存在则输出该数据在数组中最小的下标，如果不存在，输出-1

```
int main()
{
    int n;
    std::cin >> n;
    int *nValues  = new int[n];
    for (int i = 0; i < n; i++)
    {
        std::cin >> nValues[i];
    }
    int d;
    std::cin >> d;
    std::cout << Search(nValues, n, d) << std::endl;
    delete[] nValues;

    double f;
    std::cin >> n;
    double *dValues = new double[n];
    for (int i = 0; i < n; i++)
    {
        std::cin >> dValues[i];
    }
    std::cin >> f;
    std::cout << Search(dValues, n, f) << std::endl;
    delete[] dValues;

    std::cin >> n;
    char *cValues = new char[n];
    for (int i = 0; i < n; i++)
    {
        std::cin >> cValues[i];
    }
    char c;
    std::cin >> c;
    std::cout << Search(cValues, n, c) << std::endl;
    delete[] cValues;

    return 0;
}
```

【输入描述】

输入共三组数据，每组数据占三行。
第一组第一行整数 n1，第二行是 n1 个整数，第三行待查找整数 n

第二组第一行整数 n2，第二行是 n2 个浮点数，第三行待查找浮点数 d
第三组第一行整数 n3，第二行是 n3 个字符，第三行待查找字符 c

【输出描述】

对于每一组输入，如果查找数据存在，则输出其最小下标 (下标从 0 开始计)，否则输出-1。

【输入样例】

```
7
1 1 2 5 8 10 13
8
5
-1.0 1.1 1.2 1000.10101 8.9
3.5
4
B J F U
j
```

【输出样例】

```
4
-1
-1
```

【提示】

使用模板函数

```
template <class T>
int Search(const T * array, int arrayLen, const T & value)
```

main.cpp

```cpp
#include <iostream>

template <class T>
int Search(const T * array, int arrayLen, const T & value)
{
    for (int i = 0; i < arrayLen; i++)
    {
        if(array[i] == value)
            return i;
    }

    return -1;
}

int main()
{
    int n;
    std::cin >> n;
    int *nValues  = new int[n];
    for (int i = 0; i < n; i++)
    {
        std::cin >> nValues[i];
    }
    int d;
    std::cin >> d;
    std::cout << Search(nValues, n, d) << std::endl;
    delete[] nValues;

    double f;
    std::cin >> n;
    double *dValues = new double[n];
    for (int i = 0; i < n; i++)
    {
        std::cin >> dValues[i];
    }
    std::cin >> f;
    std::cout << Search(dValues, n, f) << std::endl;
    delete[] dValues;

    std::cin >> n;
    char *cValues = new char[n];
    for (int i = 0; i < n; i++)
    {
        std::cin >> cValues[i];
    }
    char c;
    std::cin >> c;
    std::cout << Search(cValues, n, c) << std::endl;
```

```
49      delete[] cValues;
50
51      return 0;
52  }
```

C++ Programming

4 TVector3

【描述】

构造一个模板类（Vector），数据成员如下：

```
template<typename T>
class Vector
{
private:
    T x, y, z;
};
```

完成 Vector，并用以下函数测试

```
int main()
{
    double a, b, c;
    std::cin >> a >> b >> c;
    Vector<double> v1(a, b, c), v2(v1), v3, v4;
    double d;
    std::cin >> d;
    v4 = d * v1 + v2;

    std::cout << v4 <<std::endl;

    Vector<double>  v;
    std::cin >> v;

    int flag = (v4 == v);
    std::cout << flag << std::endl;

    return 0;
}
```

【输入描述】

见样例

【输出描述】

见测试样例

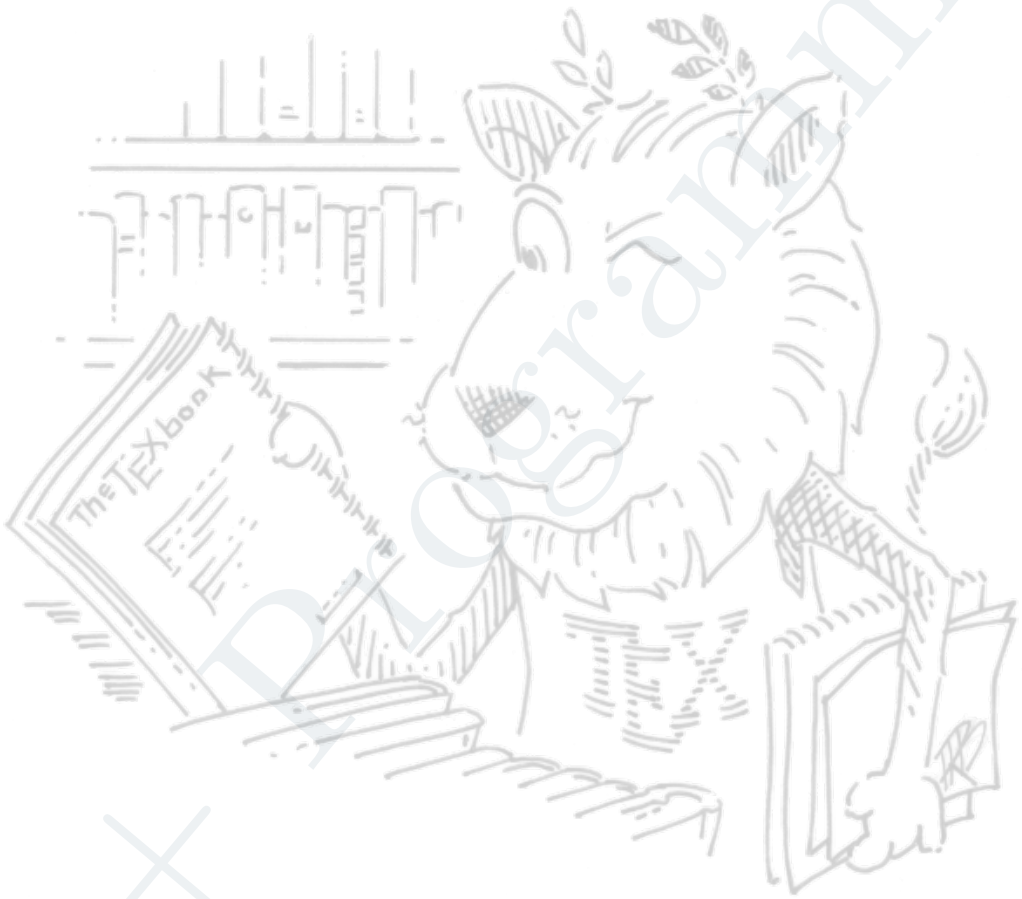
【输入样例】

```
3 4 5
2.2
9.6 12.8 16
```

【输出样例】

```
9.6 12.8 16
1
```

【提示】



C++ Programming

main.cpp

```cpp
#include <iostream>

template<typename T>
class Vector
{
public:
    Vector()
    {
        x = y = z = 0.0;
    }
    Vector(T a, T b, T c)
    {
        this->x = a;
        this->y = b;
        this->z = c;
    }

    Vector(const Vector & v)
    {
        this->x = v.x;
        this->y = v.y;
        this->z = v.z;
    }

    friend std::istream & operator >> (std::istream & in, Vector<T> & cnt)
    {
        in >> cnt.x;
        in >> cnt.y;
        in >> cnt.z;
        return in;
    }

    friend std::ostream & operator<<(std::ostream & out, const Vector<T> & cnt)
    {
        out << cnt.x << " " << cnt.y << " " << cnt.z;
        return out;
    }

    friend Vector<T> operator*(double a, const Vector<T> & cnt)
    {
        Vector<T> result;
        result.x = a*cnt.x;
        result.y = a * cnt.y;
        result.z = a * cnt.z;
        return result;
    }

    Vector<T> operator+(const Vector<T> & cnt)
```

```
49      {
50          Vector<T> result;
51          result.x = this->x + cnt.x;
52          result.y = this->y + cnt.y;
53          result.z = this->z + cnt.z;
54          return result;
55      }
56
57      int operator==(const Vector<T> & cnt)
58      {
59          double epsison = 0.000001;
60          if (abs(x - cnt.x) > epsison)return 0;
61          if (abs(y - cnt.y) > epsison)return 0;
62          if (abs(z - cnt.z) > epsison)return 0;
63
64          return 1;
65      }
66  private:
67      T x, y, z;
68  };
69
70  int main()
71  {
72      double a, b, c;
73      std::cin >> a >> b >> c;
74      Vector<double> v1(a, b, c), v2(v1), v3, v4;
75      double d;
76      std::cin >> d;
77      v4 = d * v1 + v2;
78
79      std::cout << v4 <<std::endl;
80
81      Vector<double>  v;
82      std::cin >> v;
83
84      int flag = (v4 == v);
85      std::cout << flag << std::endl;
86
87      return 0;
88  }
```

5 TStack

【描述】

实现一个 Stack 类模板并测试这一模板.

```
template<class T, int SIZE = 20>
class Stack
{
private:
    T   array[SIZE];          //数组，用于存放栈的元素
    int top;                  //栈顶位置（数组下标）
public:
    Stack();                  //构造函数，初始化栈
    void Push(const T & );    //元素入栈
    T Pop();                  //栈顶元素出栈
    void Clear();             //将栈清空
    const T & Top() const;    //访问栈顶元素
    bool Empty() const;       //测试栈是否为空
    bool Full() const;        //测试是否栈满
    int Size();               //返回当前栈中元素个数
};

int main()
{
    Stack<int, 10> intStack;

    int n;
    cin >> n; //n<=10
    for (int i = 0; i < n; i++)
    {
        int temp;
        cin >> temp;
        intStack.Push(temp);
    }

    for (int i = 0; i < n; i++)
    {
        cout << intStack.Top() << " ";
        intStack.Pop();
    }
    cout<<endl;

    if(intStack.Empty())
        cout<<"Now, intStack is empty."<<endl;

    Stack<string,5> stringStack;
    stringStack.Push("One");
    stringStack.Push("Two");
    stringStack.Push("Three");
    stringStack.Push("Four");
```

```
46      stringStack.Push("Five");
47      cout<<"There are "<<stringStack.Size()<<" elements in stringStack."<<endl;
48      stringStack.Clear();
49      if(stringStack.Empty())
50          cout<<"Now, there are no elements in stringStack"<<endl;
51
52      return 0;
53  }
```

【输入描述】

参考样例

【输出描述】

见测试样例

【输入样例】

```
3
1
2
3
```

【输出样例】

```
3 2 1
Now, intStack is empty.
There are 5 elements in stringStack.
Now, there are no elements in stringStack
```

【提示】

main.cpp

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

template<class T, int SIZE = 20>
class Stack
{
private:
    T   array[SIZE];          //数组，用于存放栈的元素
    int top;                  //栈顶位置（数组下标）
public:
    Stack();                  //构造函数，初始化栈
    void Push(const T & );    //元素入栈
    T Pop();                  //栈顶元素出栈
    void Clear();             //将栈清空
    const T & Top() const;    //访问栈顶元素
    bool Empty() const;       //测试栈是否为空
    bool Full() const;        //测试是否栈满
    int Size();               //返回当前栈中元素个数
};

template<class T, int SIZE>
Stack<T,SIZE>::Stack() : top(-1){}

template<class T, int SIZE>
void Stack<T,SIZE>::Push(const T &item)
{
    if(!Full()) array[++top] = item;
}

template<class T, int SIZE>
T Stack<T,SIZE>::Pop()
{
    if(!Empty()) return array[top--];
    else exit(1);
}

template<class T, int SIZE>
void Stack<T,SIZE>::Clear(){ top = -1; }

template<class T, int SIZE>
const T & Stack<T,SIZE>::Top() const
{
    if(!Empty()) return array[top];
    else exit(1);
}

template<class T, int SIZE>
bool Stack<T,SIZE>::Empty() const { return top == -1; }
```

```
49
50  template<class T, int SIZE>
51  bool Stack<T,SIZE>::Full() const { return top == SIZE-1; }
52
53  template<class T, int SIZE>
54  int Stack<T,SIZE>::Size() { return top+1; }
55
56  int main()
57  {
58      Stack<int, 10> intStack;
59
60      int n;
61      cin >> n; //n<=10
62      for (int i = 0; i < n; i++)
63      {
64          int temp;
65          cin >> temp;
66          intStack.Push(temp);
67      }
68
69      for (int i = 0; i < n; i++)
70      {
71          cout << intStack.Top() << " ";
72          intStack.Pop();
73      }
74      cout<<endl;
75
76      if(intStack.Empty())
77          cout<<"Now, intStack is empty."<<endl;
78
79      Stack<string,5> stringStack;
80      stringStack.Push("One");
81      stringStack.Push("Two");
82      stringStack.Push("Three");
83      stringStack.Push("Four");
84      stringStack.Push("Five");
85      cout<<"There are "<<stringStack.Size()<<" elements in stringStack."<<endl;
86      stringStack.Clear();
87      if(stringStack.Empty())
88          cout<<"Now, there are no elements in stringStack"<<endl;
89
90      return 0;
91  }
```

6 TList

【描述】

设计一个单向链表的类模板，类模板的说明如下：

```
template <class T>
class List{
private:
    T data;
    List * next;
    List * tail;        //指向最后一个结点
    List * head;        //指向头结点
public:
    List():next(NULL)                       //构造头结点
    {
        head = tail = this;
    }
    List(T newnode):data(newnode),next(NULL)     //构造新结点
    {}
    void Append(T node);                    //往后面添加结点
    bool Insert(T node, T posnode);    //在结点posnode第一次出现的后面插入新结点，插入成功返回true，否则false
    void DeleteNode(T node);            //删除结点,注意可能有多个相同的结点需要删除
    void DeleteList();                      //删除整个链表
    void DisplayList();                     //显示链表
};
```

你的任务是实现这个类模板中的成员函数，然后使用如下所示的 main() 函数测试你实现的类模板。

```
int main()
{
    List<int> list1;
    list1.Append(1);
    list1.DeleteNode(1);
    list1.Append(2);
    list1.Append(3);
    list1.Append(4);
    list1.Insert(10,2);
    list1.Append(5);
    list1.Append(3);
    list1.Append(3);
    list1.DisplayList();
    list1.DeleteNode(3);
    list1.DisplayList();
    list1.DeleteList();
    list1.DisplayList();

    List<char> list2;
    list2.Append('A');
    list2.Append('B');
    list2.Append('C');
```

```
23      list2.Append('D');
24      list2.Insert('E','B');
25      list2.Insert('F','D');
26      list2.Append('G');
27      list2.Append('G');
28      list2.Append('G');
29      list2.DisplayList();
30      list2.DeleteNode('G');
31      list2.DisplayList();
32      list2.DeleteList();
33      list2.DisplayList();
34
35      return 0;
36  }
```

【输入描述】

无

【输出描述】

见测试样例

【输入样例】

无

【输出样例】

```
2 10 3 4 5 3 3
2 10 4 5

A B E C D F G G G
A B E C D F
```

【提示】

(1) Append 函数是在链表尾部（即 tail 指向的结点）后面增加一个新节点，因此 tail 指针有变化。
(2) 在 Insert 函数中，需要先找到 posnode，因此要定义一个指针指向 posnode；然后新建一个结点（该结点的数据是 node)；如果 posnode 是最后一个结点，插入新节点还需要修改 tail 指针，否则直插入新节点不需要修改 tail 指针。
(3) 在 DeleteNode 函数中，需要两个临时指针 find,pre，其中 find 指向 pre 的 next，最终 find 指向删除的结点。在循环中，如果需要删除的是最后一个节点，需要修改 tail 指针，删除 find，然后 find=NULL；如果不是最后一个结点，把 find 指向的结点从链表中分离出来，删除 find，find 指向 pre 的 next；如果没有找到删除的结点，则继续找下一个结点：pre =find；find=find->next
(4) 在 DeleteList 函数中，从头循环将要删除的结点从链表中分离出来，头指针相应修改，然后删除（delete）分离的结点。最后将 tail 和 head 指向同一个，next 指针为 NULL。

(5) 在 DisplayList 函数中，从头到尾显示结点。

C++ Programming

main.cpp

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

template <class T>
class List{
private:
    T data;
    List * next;
    List * tail;        //指向最后一个结点
    List * head;        //指向头结点
public:
    List():next(NULL)                       //构造头结点
    {
        head = tail = this;
    }
    List(T newnode):data(newnode),next(NULL)     //构造新结点
    {}
    void Append(T node);                    //往后面添加结点
    bool Insert(T node, T posnode);    //在结点posnode第一次出现的后面插入新结点，插入成功返回true，否则false
    void DeleteNode(T node);                //删除结点,注意可能有多个相同的结点需要删除
    void DeleteList();                      //删除整个链表
    void DisplayList();                     //显示链表
};


template <class T>
void List<T>::Append(T node)
{
    tail->next = new List(node);
    tail = tail->next;
}

template <class T>
bool List<T>::Insert(T node, T posnode)
{
    for(List * find = head->next; find != NULL; find = find->next)
        if(find->data == posnode)          //找到了插入的位置
        {
            List * temp = new List(node);
            if(find->next == NULL)          //posnode是最后一个结点，此时需要修改tail
            {
                temp->next = NULL;
                find->next = tail = temp;
                return true;
            }else{                                  //不是最后一个结点，只是中间结点,不需要修改tail
```

```
46                  temp->next = find->next;
47                  find->next = temp;
48                  return true;
49              }
50          }
51      return false;
52  }
53
54  template <class T>
55  void List<T>::DeleteNode(T node)
56  {
57      List * find = head->next, * pre = head;
58      while(find != NULL)
59      {
60          if(find->data == node)
61          {
62              if(find->next == NULL)          //要删除的是最后一个结点，需要修改tail
63              {
64                  pre->next = NULL;
65                  tail = pre;
66                  delete find;
67                  find = NULL;
68              }else{                          //要删除的不是最后一个结点
69                  pre->next = find->next;
70                  delete find;
71                  find = pre->next;
72              }
73          }else{                              //不是需要删除的结点
74              pre = find;
75              find = find->next;
76          }
77      }
78  }
79
80  template <class T>
81  void List<T>::DeleteList()
82  {
83      List * current = head->next;
84      List * temp;
85      while(current != NULL)
86      {
87          temp = current;
88          head->next = current->next;
89          current = current->next;
90          delete temp;
91      }
92      tail = head;
93      head->next = NULL;
94  }
95
```

```
96  template <class T>
97  void List<T>::DisplayList()
98  {
99      for(List * temp = head->next; temp != NULL; temp = temp->next)
100         cout<<temp->data<<" ";
101     cout<<endl;
102 }
103
104 int main()
105 {
106     List<int> list1;
107     list1.Append(1);
108     list1.DeleteNode(1);
109     list1.Append(2);
110     list1.Append(3);
111     list1.Append(4);
112     list1.Insert(10,2);
113     list1.Append(5);
114     list1.Append(3);
115     list1.Append(3);
116     list1.DisplayList();
117     list1.DeleteNode(3);
118     list1.DisplayList();
119     list1.DeleteList();
120     list1.DisplayList();
121
122     List<char> list2;
123     list2.Append('A');
124     list2.Append('B');
125     list2.Append('C');
126     list2.Append('D');
127     list2.Insert('E','B');
128     list2.Insert('F','D');
129     list2.Append('G');
130     list2.Append('G');
131     list2.Append('G');
132     list2.DisplayList();
133     list2.DeleteNode('G');
134     list2.DisplayList();
135     list2.DeleteList();
136     list2.DisplayList();
137
138     return 0;
139 }
```

7 TComplex

【描述】

复数模板类 Complex 包含实部和虚部，复数的模为实部和虚部的平方和再开根号。完成这个模板类，并通过测试函数

```
int main()
{
    int ir, ii;
    cin >> ir >> ii;
    Complex<int> ci1(ir, ii);
    Complex<int> ci2(2, 3);
    Complex<int> ci = ci1 + ci2;
    cout << ci.Mag() << endl;

    double dr, di;
    cin >> dr >> di;
    Complex<double> cd(dr, di);
    cout << setiosflags(ios::fixed) << setprecision(2);
    cout << cd.Mag() << endl;

    return 0;
}
```

【输入描述】

实部虚部
实部虚部

【输出描述】

见测试样例

【输入样例】

```
1 1
1.5 1.5
```

【输出样例】

```
5
2.12
```

【提示】

函数模板，类模板，基本操作符重载，cin/cout 重载，友元

main.cpp

```cpp
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;

template<class T>
class Complex
{
private:
    T real, image;
public:
    Complex(T real, T image)
    {
        this->real = real;
        this->image = image;
    }

    T Mag()const
    {
        double res = real*real + image*image;
        return T(sqrt(res));
    }

    Complex<T> operator+(const Complex<T> c)
    {
        return Complex<T>(real+c.real, image+c.image);
    }
};

int main()
{
    int ir, ii;
    cin >> ir >> ii;
    Complex<int> ci1(ir, ii);
    Complex<int> ci2(2, 3);
    Complex<int> ci = ci1 + ci2;
    cout << ci.Mag() << endl;

    double dr, di;
    cin >> dr >> di;
    Complex<double> cd(dr, di);
    cout << setiosflags(ios::fixed) << setprecision(2);
    cout << cd.Mag() << endl;

    return 0;
}
```

8 TMyQueue

【描述】

设计一个 MyQueue 类模板，类模板说明如下：

```
template <typename T> class MyQueue;//前置声明
template <typename T> std::ostream& operator<<(std::ostream&, const MyQueue<T>&);

template <typename T>
class QueueItem
{
public:
    QueueItem(const T &t) :item(t), next(0)
    {}
private:
    T item; //value stored in this element
    QueueItem *next; // pointer to next element in the MyQueue

    friend class MyQueue<T>;//友元类
    //通过友元函数重载<<运算符模板函数，要写上<<后的<Type>
    friend ostream & operator<< <T> (ostream & os, const MyQueue<T> & q);
};

template <typename T>
class MyQueue
{
public:
    MyQueue() : head(0), tail(0) {} // Empty MyQueue

    MyQueue(const MyQueue &Q) //拷贝构造函数
        :head(0), tail(0)
    { CopyElements(Q); };

    ~MyQueue() { Destroy(); }

    MyQueue & operator=(const MyQueue &);

    // return element from head of MyQueue
    T & Front() { return head->item; }
    const T & Front() const { return head->item; }
    void Push(const T &); //add element to back of MyQueue
    void Pop(); // remove element from head of MyQueue
    bool Empty() const { return head == 0; }
    void Display() const;
private:
    QueueItem<T> *head;
    QueueItem<T> *tail;
    void Destroy(); //delete all the elements
    void CopyElements(const MyQueue &);

```

```
46      //设置友元函数
47      friend ostream & operator<< <T> (ostream & os, const MyQueue<T> & q);
48  };
```

实现这个类模板中的成员函数，然后使用如下所示的 main() 函数测试这一类模板。

```
1   int main()
2   {
3       MyQueue<int> qi;
4       qi.Push(1);
5       qi.Push(2);
6       qi.Push(3);
7       qi.Push(4);
8       qi.Push(5);
9       qi.Pop();
10      qi.Display();
11      cout<<"\n";
12      cout<<qi;
13      cout<<endl;
14
15      MyQueue<int> qi2(qi);
16      qi2.Display();
17      cout<<endl;
18
19      MyQueue<int> qi3;
20      qi3 = qi;
21      cout<<qi3;
22
23      return 0;
24  }
```

【输入描述】

无

【输出描述】

见测试样例

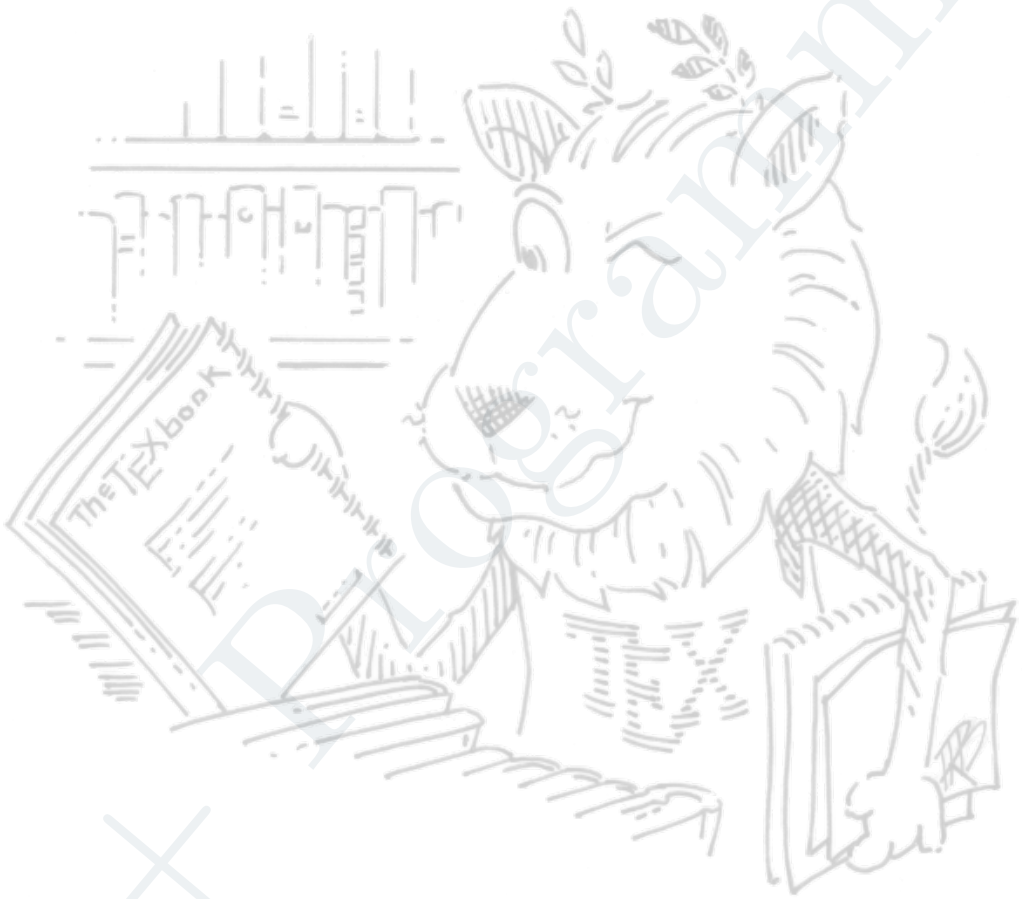
【输入样例】

无

【输出样例】

```
2 3 4 5
< 2 3 4 5 >
2 3 4 5
< 2 3 4 5 >
```

【提示】



C++ Programming

main.cpp

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

template <typename T> class MyQueue;//前置声明
template <typename T> std::ostream & operator<<(std::ostream &, const MyQueue<T> &)
    ;

template <typename T>
class QueueItem
{
public:
    QueueItem(const T &t) :item(t), next(0)
    {}
private:
    T item; //value stored in this element
    QueueItem *next; // pointer to next element in the MyQueue

    friend class MyQueue<T>;//友元类
    //通过友元函数重载<<运算符模板函数，要写上<<后的<Type>
    friend ostream & operator<< <T> (ostream & os, const MyQueue<T> & q);
};

template <typename T>
class MyQueue
{
public:
    MyQueue() : head(0), tail(0) {} // Empty MyQueue

    MyQueue(const MyQueue &Q) //拷贝构造函数
        :head(0), tail(0)
    { CopyElements(Q); };

    ~MyQueue() { Destroy(); }

    MyQueue & operator=(const MyQueue &);

    // return element from head of MyQueue
    T & Front() { return head->item; }
    const T & Front() const { return head->item; }
    void Push(const T &); //add element to back of MyQueue
    void Pop(); // remove element from head of MyQueue
    bool Empty() const { return head == 0; }
    void Display() const;
private:
    QueueItem<T> *head;
    QueueItem<T> *tail;
    void Destroy(); //delete all the elements
    void CopyElements(const MyQueue &);
```

```
48
49      //设置友元函数
50      friend ostream & operator<< <T> (ostream & os, const MyQueue<T> & q);
51  };
52
53  template<typename T>
54  void MyQueue<T>::Destroy()
55  {
56      while(!Empty())
57          Pop();
58  }
59
60  template<typename T>
61  void MyQueue<T>::Pop()
62  {
63      QueueItem<T> * p = head;
64      head = head->next;
65      delete p;
66  }
67
68  template<typename T>
69  void MyQueue<T>::Push(const T & val)
70  {
71      QueueItem<T> * pt = new QueueItem<T>(val);
72      if(Empty())
73          head = tail = pt;
74      else{
75          tail->next = pt;
76          tail = pt;
77      }
78  }
79
80  template<typename T>
81  void MyQueue<T>::CopyElements(const MyQueue<T> &orig)
82  {
83      for(QueueItem<T> * pt = orig.head; pt; pt = pt->next)
84          Push(pt->item);
85  }
86
87  template<typename T>
88  MyQueue<T> & MyQueue<T>::operator=(const MyQueue<T> & orig)
89  {
90      for(QueueItem<T> * pt = orig.head; pt; pt = pt->next)
91              Push(pt->item);
92      return *this;
93  }
94
95  template<typename T>
96  void MyQueue<T>::Display() const
97  {
```

```
98      for(QueueItem<T> * pt = head; pt; pt = pt->next)
99                  cout<<pt->item<<" ";
100 }
101
102 template<typename T>
103 ostream & operator<<(ostream & os, const MyQueue<T> & q)
104 {
105     os<<"< ";
106     QueueItem<T> * p;
107     for(p = q.head; p; p = p->next)
108         os<<p->item<<" ";
109     os<<">";
110     return os;
111 }
112
113 int main()
114 {
115     MyQueue<int> qi;
116     qi.Push(1);
117     qi.Push(2);
118     qi.Push(3);
119     qi.Push(4);
120     qi.Push(5);
121     qi.Pop();
122     qi.Display();
123     cout<<"\n";
124     cout<<qi;
125     cout<<endl;
126
127     MyQueue<int> qi2(qi);
128     qi2.Display();
129     cout<<endl;
130
131     MyQueue<int> qi3;
132     qi3 = qi;
133     cout<<qi3;
134
135     return 0;
136 }
```