**1.**

#include <iostream>

using namespace std;

template <typename T>

T Max(T a, T b, T c)

{

return a > b ? (c > a ? c : a) : (c > b ? c : b);

}

int main()

{

char c1 = 'A', c2 = 'X', c3 = 'Z';

short s1 = 89, s2 = 12, s3 = 5;

int i1 = 55, i2 = 88, i3 = 4;

long l1 = 11, l2 = 99, l3 = 45;

float f1 = 2.02f, f2 = 0.26f, f3 = 10.56f;

double d1 = 1023.5669, d2 = 0.23, d3 = 3.1415;

cout << "char: " << Max(c1, c2, c3) << endl;

cout << "short: " << Max(s1, s2, s3) << endl;

cout << "int: " << Max(i1, i2, i3) << endl;

cout << "long: " << Max(l1, l2, l3) << endl;

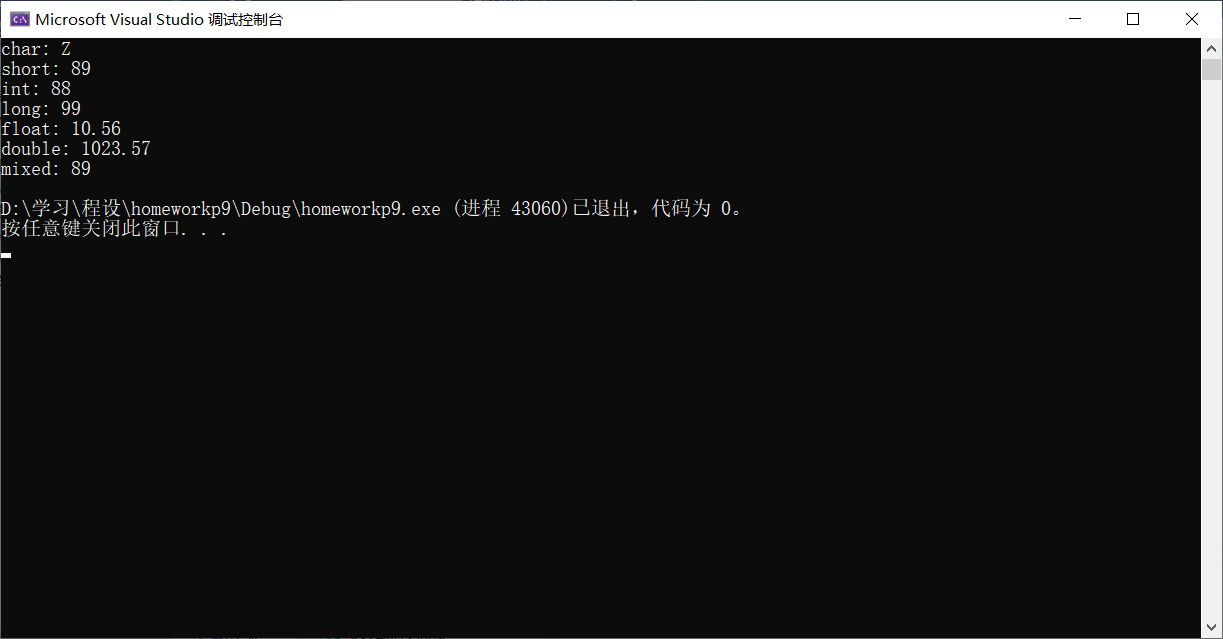
cout << "float: " << Max(f1, f2, f3) << endl;

cout << "double: " << Max(d1, d2, d3) << endl;

cout << "mixed: " << Max<double>(s1, d2, f3) << endl;

return 0;

}



**2.**

#include <iostream>

using namespace std;

template <typename T>

class Queue

{

private:

T\* Front;

int Tail;

int max;

public:

Queue(int \_size);

Queue(const Queue& \_srcque);

bool In(const T& \_elem);

T Out();

bool Empty() const { return Tail == 0; }

~Queue() { delete[] Front; };

};

template <typename T>

Queue<T>::Queue(int \_size)

{

Front = new T[max = \_size];

Tail = 0;

}

template <typename T>

Queue<T>::Queue(const Queue& \_srcque)

{

Front = new T[max = \_srcque.max];

Tail = \_srcque.Tail;

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

{

Front[i] = \_srcque.Front[i];

}

}

template <typename T>

bool Queue<T>::In(const T& \_elem)

{

if (Tail == max)

{

cout << "Overflow!" << endl;

return false;

}

else

{

Front[Tail++] = \_elem;

return true;

}

}

template <typename T>

T Queue<T>::Out()

{

if (Empty())

{

cout << "Empty queue!" << endl;

return Front[0];

}

else

{

T \_res = Front[0];

--Tail;

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

{

Front[i] = Front[i + 1];

}

return \_res;

}

}

int main()

{

Queue<int> q\_int(10);

Queue<double> q\_double(10);

cout << "int: " << endl;

for (int i = 0; i < 10; ++i)

q\_int.In(i);

q\_int.In(55);

while (!q\_int.Empty()) cout << q\_int.Out() << " ";

cout << endl;

q\_int.Out();

cout << "====================" << endl;

cout << "double: " << endl;

for (double i = 0.5; i < 47.0; i += 5.0)

q\_double.In(i);

q\_double.In(55.5);

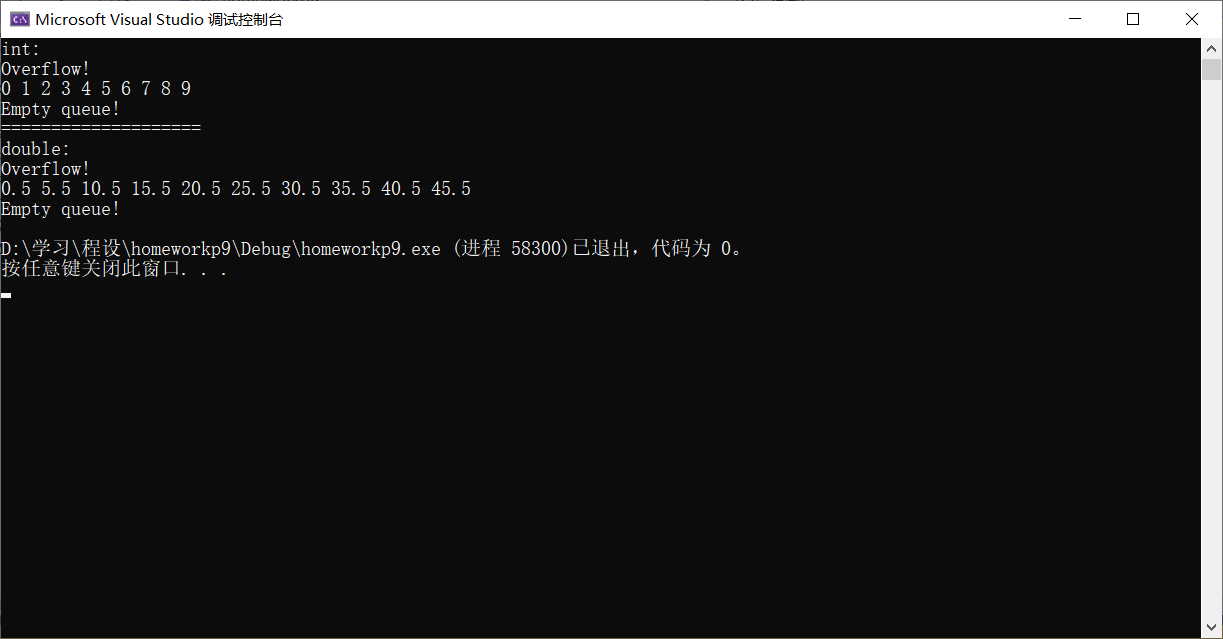
while (!q\_double.Empty()) cout << q\_double.Out() << " ";

cout << endl;

q\_double.Out();

return 0;

}



**3\*.**

#include <iostream>

using namespace std;

template <typename \_Elem>

class List

{

public:

struct node

{

\_Elem data;

node\* next;

};

List() : head(NULL) {} //构造函数

void Insert(const \_Elem& newElem); //在链表尾插入某值

void Delete(const \_Elem& delElem); //删除等于某值的一个结点

void Print() const; //打印所有值

void Clear(); //清空链表

~List() { Clear(); } //析构函数，调用Clear()

private:

node\* head;

};

template <typename \_Elem>

void List<\_Elem>::Insert(const \_Elem& newElem)

{

if (head == NULL)

{

head = new node;

head->data = newElem;

head->next = NULL;

return;

}

node\* tmp = head;

while (tmp->next) tmp = tmp->next;

tmp = tmp->next = new node;

tmp->data = newElem;

tmp->next = NULL;

}

template <typename \_Elem>

void List<\_Elem>::Delete(const \_Elem& delElem)

{

if (head == NULL) return;

if (head->data == delElem)

{

node\* tmp = head;

head = head->next;

delete tmp;

return;

}

node\* tmp = head;

while (tmp->next && tmp->next->data != delElem) tmp = tmp->next;

if (tmp->next)

{

node\* aft = tmp->next->next;

delete tmp->next;

tmp->next = aft;

}

}

template <typename \_ELem>

void List<\_ELem>::Print() const

{

if (head == NULL) return;

node\* tmp = head;

do

{

cout << tmp->data << " ";

} while (tmp = tmp->next);

cout << endl;

}

template <typename \_Elem>

void List<\_Elem>::Clear()

{

if (head == NULL) return;

node\* tmp = head;

do

{

tmp = tmp->next;

delete head;

} while (head = tmp);

head = NULL;

}

int main()

{

List<int> l;

List<char> lc;

for (int i = 0; i < 20; ++i)

l.Insert(i);

l.Print();

cout << "====================" << endl;

l.Delete(15);

l.Delete(15);

l.Delete(0);

l.Delete(19);

l.Delete(40);

l.Print();

cout << "====================" << endl;

l.Clear();

l.Delete(9);

l.Insert(4);

l.Delete(5);

l.Print();

cout << "====================" << endl << "====================" << endl;

for (char c = 'a'; c < 'z' + 1; ++c)

{

lc.Insert(c);

}

lc.Print();

cout << "====================" << endl;

lc.Delete('p');

lc.Insert('T');

lc.Insert('H');

lc.Insert('U');

lc.Print();

return 0;

}

