**程序填空题5-1**

Birth(int year,int month, int day) :\_year(year),\_month(month),\_day(day)

Student(string name, int id, int year, int month, int day) :birth(year,month,day)

Birth

Student

**填空题5-2**

const Birth& another

\_year(another.\_year),\_month(another.\_month),\_day(another.\_day)

string name, int id

birth(1900,1,1)

const Student& another

birth(another.birth)

another.\_name

another.\_id

**填空题5-3**

~Birth()

~Student()

10002

10003

10005

10001

10004

**填空题5-4**

:m\_A(1)

void ShowPerson() const

void ShowPerson()

重载1

1

0

重载2

**填空题5-5**

static int

static void func()

int Person::m\_A

p1.m\_A

Person::m\_A

p3.func();

Person::func();

**填空题5-6**

friend class goodGay;

friend void goodGayGay::visit();

friend void printt(Building \* building);

**函数题6-3**

#include <iostream>

using namespace std;

class Complex {

private:

int real, imag;

public:

Complex(int r = 0, int i = 0) : real(r), imag(i) {}

Complex(const Complex& c) : real(c.real), imag(c.imag) {}

void Display() {

if(imag>=0) cout << real << "+" << imag << "i" << endl;

if(imag<0) cout << real << imag << "i" << endl;

}

friend Complex AddComplex(const Complex& c1, const Complex& c2) {

Complex res;

res.real = c1.real + c2.real;

res.imag = c1.imag + c2.imag;

return res;

}

};

**函数题6-4**

Array(int n) { // 构造函数

size = n;

data = new int[size]();

}

Array(const Array& other) { // 拷贝构造函数

size = other.size;

data = new int[size]();

for (int i = 0; i < size; i++) {

data[i] = other.data[i];

}

}

Array& operator=(const Array& other) { // 赋值运算符重载函数

if (this == &other) return \*this;

delete[] data;

size = other.size;

data = new int[size]();

for (int i = 0; i < size; i++) {

data[i] = other.data[i];

}

return \*this;

}

~Array() { // 析构函数

delete[] data;

}

**编程题7-2（7-1略）**

#include<iostream>

using namespace std;

class MyQueue {

public:

MyQueue() :head(new node) {}

MyQueue(const MyQueue& copy);

MyQueue(MyQueue&& copy)noexcept;

MyQueue &operator=(MyQueue &&copy)noexcept;

MyQueue &operator=(const MyQueue &copy);

void push(int x);

int pop();

int size();

int count = 0;

private:

struct node {

node() :next(NULL) {}

int data;

node \*next;

};

node \*head;

};

void MyQueue::push(int x) {

node\* p = new node;

p->data = x;

node\* q = head;

while (q->next != NULL) {

q = q->next;

}

q->next = p;

p->next = NULL;

count++;

}

int MyQueue::pop() {

if (head->next != NULL) {

node \*q = head->next;

int x = q->data;

head->next = q->next;

delete q;

count--;

return x;

}

else return -1;

}

int MyQueue::size() {

return count;

}

MyQueue::MyQueue(const MyQueue& copy) {

head = new node;

node \*p = head;

node \*q = copy.head->next;

while (q != NULL) {

node \*tmp = new node;

tmp->data = q->data;

p->next = tmp;

p = p->next;

q = q->next;

}

p->next = nullptr;

count = copy.count;

}

MyQueue::MyQueue(MyQueue &&copy) noexcept {

swap(head, copy.head);

swap(count, copy.count);

}

MyQueue& MyQueue::operator=(const MyQueue &copy) {

if (this != &copy) {

MyQueue tmp(copy);

swap(head, tmp.head);

swap(count, tmp.count);

}

return \*this;

}

MyQueue& MyQueue::operator=(MyQueue &&copy) noexcept {

if (this != &copy) {

swap(head, copy.head);

swap(count, copy.count);

}

return \*this;

}

int main() {

int x;

MyQueue q;

while (cin >> x) {

q.push(x);

}

MyQueue q1(q), q2 = q;

q.pop();

while (q.size() > 1) {

cout << q.pop() << " ";

}

cout << q.pop() << endl;

while (q1.size() > 1) {

cout << q1.pop() << " ";

}

cout << q1.pop() << endl;

while (q2.size() > 1) {

cout << q2.pop() << " ";

}

cout << q2.pop() << endl;

return 0;

}