张涵之 191220154 第三次作业

概念题

1. 请说出C++中同类对象共享数据的两种方式，并比较它们的优缺点。

a) 采用全局变量：共享的数据与对象之间缺乏显式联系，不通过对象也能访问，不安全；

b) 采用静态数据成员：共享的数据只能通过对象或类访问，安全。

2. 下面对静态数据成员的描述中，正确的是 ( D )

3. 已知类A是类B的友元，类B是类C的友元，则 ( D )

4. 简述C++中的迪米特法则(Law Of Demeter)，遵循迪米特法则设计的模块具有哪些优点？

一个类的成员函数除了访问自身类结构的直接子结构（本类的数据成员）外，不能以任何方式依赖于任何其它类的结构，只应向某个有限集合中的对象发送消息。

迪米特法则能够降低类之间的耦合，很容易使得系统的功能模块相互之间独立。

编程题

1. 数组类Array和矩阵类Matrix：实例化两个Matrix对象并按行输出：

#include <iostream>

using namespace std;

class Array {

int\* array;

friend class Matrix;

public:

Array(int n);

~Array();

};

Array::Array(int n) {

array = new int[n];

}

Array::~Array() {

delete[]array;

}

class Matrix {

int row;

int col;

Array\* arr;

public:

Matrix(int r, int c);

void input(int\* a, int length);

Matrix\* inverse();

void print();

};

Matrix::Matrix(int r, int c) {

row = r;

col = c;

arr = new Array(row \* col);

}

void Matrix::input(int\* a, int length) {

int i;

if (row \* col > length) {

for (i = 0; i < length; i++)

arr->array[i] = a[i];

for (i = length; i < row \* col; i++)

arr->array[i] = 0;

}

else {

for (i = 0; i < row \* col; i++)

arr->array[i] = a[i];

}

}

Matrix\* Matrix::inverse() {

Matrix\* inversed\_m = new Matrix(row, col);

for (int i = 0; i < row \* col; i++)

inversed\_m->arr->array[i] = - arr->array[i];

return inversed\_m;

}

void Matrix::print() {

int i, j;

for (i = 0; i < row; i++) {

cout << '[';

for (j = 0; j < col - 1; j++)

cout << arr->array[i \* col + j] << ", ";

cout << arr->array[(i + 1) \* col - 1] << ']' << endl;

}

}

int main()

{

int r, l;

cin >> r >> l;

Matrix m(r, l);

int\* a, length;

cin >> length;

a = new int[length];

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

cin >> a[i];

m.input(a, length);

m.print();

Matrix\* reversed = m.inverse();

reversed->print();

return 0;

}

2. 字符串类String：两个字符串拼接赋值给第三个，输出拼接的字符串及其总长度：

#include <iostream>

using namespace std;

class String {

char\* s;

friend void string\_join(String a, String b, String& c);

public:

String();

String(char\* c);

int string\_length();

void print\_string();

};

String::String() {

s = new char[100];

for (int i = 0; i < 100; i++)

s[i] = '\0';

}

String::String(char\* c) {

int i;

s = new char[100];

for (i = 0; c[i] != '\0'; i++)

s[i] = c[i];

s[i] = '\0';

}

int String::string\_length() {

int length = 0;

for (; s[length] != '\0'; length++);

return length;

}

void string\_join(String a, String b, String& c) {

int i, j;

for (i = 0; a.s[i] != '\0'; i++)

c.s[i] = a.s[i];

for (j = 0; b.s[j] != '\0'; j++) {

c.s[i] = b.s[j];

i += 1;

}

c.s[i] = '\0';

}

void String::print\_string() {

cout << s << endl;

}

int main()

{

char\* temp1 = new char[50];

char\* temp2 = new char[50];

cin >> temp1 >> temp2;

String SA(temp1), SB(temp2);

String SC;

string\_join(SA, SB, SC);

SA.print\_string();

SB.print\_string();

SC.print\_string();

cout << SC.string\_length() << endl;

return 0;

}

定义类Single使得程序中只能创建一个该类的对象：

#include <iostream>

using namespace std;

class Single{

Single() {}

static bool exist;

static Single\* the\_one;

public:

static Single\* create\_the\_one(){

if (exist == false) {

the\_one = new Single;

exist = true;

}

return the\_one;

}

};

bool Single::exist = false;

Single\* Single::the\_one = NULL;

int main()

{

Single\* one = Single::create\_the\_one();

Single\* two = Single::create\_the\_one();

cout << one << " " << two << endl;

return 0;

}