C++通关秘籍

以问题和回答的形式呈现，主要是填空题中涉及到的，以及函数题中与类相关的问题。

Dedicated to my best friend Lei.

下面的问题针对填空题

1. 如何引用？（对付填空题）

Eg1:

char a = 'C';

char& p = a;

这里 p 是一个对 char 类型的常量引用。//必须在初始化时绑定。

Eg2:

char a = 'C';

const char& p = a;

这里 p 是一个对 char 类型的常量引用。

 常量引用（const char&）允许引用非 const 类型的变量（比如 a），但通过这个引用 p 不能修改 a 的值。

 这是合法的代码。

Eg3:

1. const是什么？（对付填空题）

常量。

常指针和指针常量：

Eg:4

char a = 'C';

char\* const p;

// 这里 p 是一个**常量指针**（char\* const），表示指针本身是常量

//初始化后不能再指向别的地址。**常量指针必须在声明时初始化。**

在此解释**constant pointer** and **pointer to constant**。

int\* const ptr = &a; // 常量指针，ptr 的指向不能修改 constant pointer

\*ptr = 15; // 合法：可以修改 ptr 所指向的值

常指针必须在定义时初始化，因为它的指向不可以修改。

const int\* ptr1 = &a; // 指向常量的指针，不能修改所指向的值

// \*ptr1 = 15; // 错误：不能修改 ptr1 所指向的值

ptr1 = &b; // 合法：可以修改指针的指向

int\* const ptr2 = &a; // 常量指针，不能修改指针的指向

\*ptr2 = 15; // 合法：可以修改 ptr2 所指向的值

// ptr2 = &b; // 错误：不能修改常量指针的指向

只有constant reference

const type& ref;

type const& ref;//These two are equal

1. &是什么？

涉及到引用。申明变量时使用的“&”即是引用。（即变量的别名）

引用必须在声明时初始化。

int& a=A int&b=B 就声明了a是A的引用，b是B的引用。

Eg：

int x = 5;

int& ref = x;

ref = 10; // 修改 ref 相当于修改 x，x 的值为 10

Eg：

char a = 'C';

char& p; //这个写法错误，char& p 表示p是一个引用，应在声明时初始化。

p = a;

Eg：

char a = 'C';

const char& p;

//错误，char& p 表示p是一个常量引用，和引用一样应在声明时初始化。

p = a;

//常量引用不能通过引用修改绑定的对象，即它只允许 **读取**，不允许 **写入。**

&x，&y是指变量x，y的地址。

1. new 和 delete 的应用。（选择题）

**new 运算符**：

int\* p = new int; // 分配单个 int

int\* p = new int(10); // 分配单个 int并初始化

int\* arr = new int[10]; // 分配一个包含 10 个 int 的数组

int\* a=new int[]{1,2,3,4};

**delete 运算符**：

如果使用 new 分配单个对象，需用 delete 释放。

如果使用 new[] 分配数组，需用 delete[] 释放。

Eg：

int\* p = new int[10];

delete[] p;//这样释放，不能用delete p;

Eg：

vector<int>\* v = new vector<int>[10];

delete [] v;//这是用new分配含10个vector<int>对象的数组

Eg：

int\* a = new int[]{1, 2, 3, 4};//这种动态初始化数组的方法是不可以的

//报错：[{"不允许使用不完整的类型 \"int []\"",}]

//正确方法： int\* a = new int[4]{1, 2, 3, 4};

delete [] a;

Tip：可以用a[0],a[1]的方式访问创建的数组。

1. 指针。

\*取内容，指针本身指向相应地址。

1. 如何交换两变量的数值。（涉及到自定义函数）

Eg：

void fun(int &a, int &b)

fun(x, y);

按引用传递，直接改变变量。

Eg：

void fun(int \*a, int \*b) { int temp = \*a; \*a = \*b; \*b = temp; }

fun(&x, &y)

按指针传递，向函数传入两变量的地址。

1. 结构体。

struct Book

{

string name; // 书名

double price; // 价格

}; //类似这样定义结构体

可以直接定义结构体变量，如 Book b。也可以动态分配内存，如 Book \*p = new Book。

**访问结构体成员**：1.**点运算符 (.)**：用于直接访问结构体变量的成员。2.**箭头运算符 (->)**：用于通过指针访问结构体的成员。

结构体初始化：

Eg：

Book b{"C++",20};//用列表初始化

Eg：

Book \*b=new Book[2];

b[0].name="C++";//成员访问使用 .，因为 b[0] 和 b[1] 是结构体对象，而不是指针。

b[0].price=20; // b[0] 等价于 \*(b + 0)，即通过指针解引用得到的Book 对象。

Eg：

Book \*p = new Book();//Book \*p = new Book;也可以

p->name = "C++";

p->price = 20;

1. 拷贝构造函数。（什么时候会被用到）

Eg：调用默认拷贝构造函数，实行浅拷贝。（没有指针就没有问题）

MyClass a;

MyClass b = a;

Eg：

MyClass a;

MyClass b(a);

void f(MyClass obj)

{

...

}

MyClass a;

f(a);

函数参数 obj 是按值传递，这会将对象 a 拷贝一份传入函数 f 的参数，会调用拷贝构造函数。

下面的问题针对函数题：

在类外面写类的实现，记得加：：

Eg：Teacher：：Teacher //::是作用域运算符

1. sort函数：记得#include <algorithm> （同步到博客）

假设我们有一个数组，需要对它的元素排序。你可以写一个冒泡排序。但也可以直接用sort函数。（sort函数默认从小到大排序）

Eg：

int arr[] = {5, 2, 9, 1, 5, 6}; // 定义一个数组

int n = sizeof(arr) / sizeof(arr[0]); // 计算数组的大小

// 使用 std::sort 对数组进行排序

std::sort(arr, arr + n); // 从 arr 的起始位置到结束位置排序

Eg：

vector<int> vec = {3, 7, 2, 8, 1, 4}; // 定义一个 vector

// 使用 std::sort 对 vector 进行排序

sort(vec.begin(), vec.end()); // vec.begin() 是起始位置，vec.end() 是结束位置

#include <iostream>

#include <algorithm> // std::sort

// 自定义比较函数

bool compare(int a, int b) {return a > b; }

// 返回 true 表示 a 应该排在 b 前面（降序）

int main() {

int arr[] = {5, 2, 9, 1, 5, 6};

// 使用自定义比较函数进行排序

std::sort(arr, arr + n, compare);

std::cout << "降序排序后的数组: ";

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

{std::cout << arr[i] << " ";}

return 0;

}

1. string的访问

string a,b;//a,b是string类的对象

我们可以使用a.size获取字符串a的长度。

a.[4]访问a字符串中第5个字符。

a.size()

1. getline

string input;

std::getline(std::cin, input);

// std::getline(std::cin, input, ','); // 使用getline，并以逗号 ',' 为分隔符

1. 模板。分为函数模板和类模板。（变量的自动推导问题）

类模板不支持类型自动推导，函数模板一定程度上支持。

template<typename T1, typename T2>

class MyClass{

private:

T1 x;

T2 y;

public:

MyClass(T1 \_x, T2 \_y):x(\_x),y(\_y){}

};

1. this->

类的成员函数中，this指针指向调用该函数的对象（类的实例）。

使用 this-> 来区分成员变量和函数参数

class MyClass {

private:

int value;

public:

MyClass(int v) : value(v) {}

}

//在类的外面写类的实现，注意类的名称在哪个位置

void **MyClass::**setValue(int value) {

// 使用 this-> 来区分成员变量和函数参数

this->value = value; // this->value 是成员变量，value 是函数参数

}

1. 运算符重载（同步到博客）

+-的重载：

Vec2 Vec2::operator+(const Vec2&b)

{

return Vec2(this->u+b.u,this->v+b.v);//注意this->怎么用

}

bool operator!=(const Vec2&a,const Vec2&b)

<< >>的重载：

ostream&operator<<(ostream&os,const Vec2&c)

{

os<<"u="<<c.u<<","<<"v="<<c.v;

return os;//注意什么地方使用os

}

1. 继承时的注意事项。

析构函数和同名函数设置成虚函数。

基类不存在默认构造函数的问题：

子类的构造函数“:” 接初始化列表，主要用于初始化基类构造函数和子类的成员变量。

在 C++ 中，派生类的构造函数在执行时会先调用基类的构造函数。如果基类没有默认构造函数，则派生类必须显式调用基类的其他构造函数。

菱形继承中间设置虚基类：

（虚继承）

1. 初始化列表（同步到博客）

class MyClass {

int x;

public:

MyClass(int val) : x(val) { } // 初始化列表

};//在初始化列表中，成员变量 x 在构造函数体执行之前被直接初始化为 val 的值。

#include<iostream>

#include<string>

using namespace std;

class Teacher{

private:

string name;

int age;

public:

Teacher(string name,int age);

string getName() const;

int getAge() const ;

void setName(string name);

void setAge(int age);

};

/\* 请在这里填写答案 \*/

Teacher::Teacher(string name,int age)

{

this->name = name;

this->age = age;

}

string Teacher::getName() const

{

return name;

}

int Teacher::getAge() const

{

return age;

}

void Teacher::setName(string name)

{

this->name = name;

}

void Teacher::setAge(int age)

{

this->age = age;

}

/\* 请在这里填写答案 \*/

int main(){

Teacher a("Wang",20);

cout<<"name:"<<a.getName()<<endl;

cout<<"age:"<<a.getAge()<<endl;

a.setName("Zhang");

a.setAge(30);

cout<<"name:"<<a.getName()<<endl;

cout<<"age:"<<a.getAge()<<endl;

return 0;

}

#include<iostream>

using namespace std;

class Vec2{

private:

double u;

double v;

public:

Vec2(double u=0,double v=0);

double getU() const;

double getV() const;

Vec2 operator+(const Vec2&b);

friend Vec2 operator-(const Vec2&a,const Vec2&b);

bool operator==(const Vec2&b) const;

friend bool operator!=(const Vec2&a,const Vec2&b);

friend ostream&operator<<(ostream&os,const Vec2&c);

friend istream&operator>>(istream&is,Vec2&c);

};

double Vec2::getU() const

{

return u;

}

double Vec2::getV() const

{

return v;

}

Vec2 operator-(const Vec2&a,const Vec2&b){

return Vec2(a.u-b.u,a.v-b.v);

}

bool Vec2::operator==(const Vec2&b) const{

return u==b.u&&v==b.v;

}

/\* 请在这里填写答案 \*/

Vec2::Vec2(double u,double v)

{

this->u = u;

this->v = v;

}

Vec2 Vec2::operator+(const Vec2&b)

{

//代码

return Vec2(this->u+b.u,this->v+b.v);

}

bool operator!=(const Vec2&a,const Vec2&b)

{

bool check = true;

if(a.u==b.u)

{

if(a.v==b.v)

check = false;

}

return check;

}

ostream&operator<<(ostream&os,const Vec2&c)//match cout

{

os<<"u="<<c.u<<","<<"v="<<c.v;

return os;

}

istream&operator>>(istream&is,Vec2&c)//match cin

{

//代码

is>>c.u>>c.v;

return is;

}

/\* 请在这里填写答案 \*/

int main(){

Vec2 a;

cin>>a;

cout<<"a: "<<a<<endl;

Vec2 b(3,4);

Vec2 c=a+b;

cout<<"c: "<<c<<endl;

Vec2 d=a-b;

cout<<"d: "<<d<<endl;

cout<<"a==a: "<<(a==a)<<endl;

cout<<"a!=a: "<<(a!=a)<<endl;

return 0;

}

#include<iostream>

#include<string>

using namespace std;

/\* 请在这里填写答案 \*/

class Fruit

{

protected:

int weight;

public:

Fruit(int a)

{

weight=a;

cout<<"Fruit Constructor"<<endl;

}

virtual ~Fruit(void)

{

cout<<"Fruit Destructor"<<endl;

}

virtual void display(void)

{

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

}

};

class Banana:public Fruit

{

private:

string origin;

public:

Banana(string a,int b):Fruit(b)

{

origin = a;

cout<<"Banana Constructor"<<endl;

}

~Banana(void)

{

cout<<"Banana Destructor"<<endl;

}

void display(void)

{

cout<<"origin="<<origin<<",weight="<<weight<<endl;

}

};

/\* 请在这里填写答案 \*/

int main(){

Fruit \*pf=new Banana("Chongqing",10);;

pf->display();

delete pf;

return 0;

}