c++复习提纲

from Karry Ran

常见考点:

- cout 和 cin 都是对象名称，初级类名为i/ostream 抽象类名为ios <文件输入输出>
- 在c++种实现运行时多态性 <多态>
    - 动态多态 虚函数的知识点
        - 虚函数的调用规则
    - 静态多态
    - 运算符重载 <重要考点>
        - 关键字：operator
        - 重载规则：只能重载C++中已经有的运算符，不能定义新运算符
        - 不同的方式形参变量个数
        - << >> 的重载 以及强制类型转化的重载
- 有关继承和派生
    - 继承方式所带来的对属性的改变 （派生类访问权限 类外访问权限 搞明白）
    - 派生类对象可以赋值给基类对象
    - 派生类和基类构造函数析构函数调用顺序 <直接创建对象 和 创建对象指针在堆区的不同>
- 常对象
    - 通过常对象只能调用它的常成员函数
        - 常函数的调用规则
- 模板
    - 模板的书写方式
    - 模板函数
        - 模板函数和普通函数重载时的调用规则
- 静态成员函数和静态成员数据
    - 静态成员函数的初始化
- 杂点:
    - 全局变量的表示 ::变量名

```
int x = 2;
int main(void)
    int x = 3
    cout << ::x << " " << x;
            2             3
```

第一章 基础知识

1.2对c的扩充

1.2.1随时随地声明变量

1.2.2作用域限定符 "::"

- 语法 限定符 :: 成员名
- 限定符一般为类名 用于类成员访问
- 没有限定符 :: 成员名表示全局变量
- 只有成员名就不用说了

1.2.3 const

- const 为右结合

1.2.4 内联函数

- 在函数前面加 inline 就可
- <编译器> 将函数 代码嵌到每一处

```
9. 下列有关内置函数的叙述中，正确的是（ C ）。
A）内置函数在调用时发生控制转移
B）内置函数必须通过关键字inline来定义
C）内置函数是通过编译器来实现的
D）内置函数体的最后一条语句必须是return语句
```

1.2.5 函数重载

- c不允许在同一作用域内出现名称相同的函数
- c++允许 但是重载函数的(参数) <类型> <个数> <顺序>至少要有一个不一样 或者是否为const
- <返回值类型>没有所谓

1.2.6 new & delete

```
new *p = int; //无初值 只有类型
new *p = int(5); //有初值 有类型
new *p = int[5]; //数组

delete p; // 针对前两种
delete []p; // 针对最后一种
```

第二章 类和对象

2.2

2.2.1 类成员访问权限

- 默认为private
- public: 1.类外可访问 2.派生类可访问
- protected: 1.类外不可访问 2.派生类可访问
- private: 1.类外不可访问 2.派生类不可访问

2.2.1 类成员存储空间

- 只有数据成员才算在类的size里面
- static 也好 函数也罢都不算

2.3 构造函数和析构函数

2.3.1构造函数

- 可重载
- 可用初始化列表 但是继承来的父类成员必须用初始化列表

2.3.2 拷贝构造函数

- 类名(const 类名 &对象名字)
- 浅拷贝：类中不含指针型的数据成员 那么就是浅拷贝 简单的让两个类之间的数据成员复制一下就可以了
- 深拷贝：类中含有指针型的数据成员 要深拷贝

```
class Person
{
    public:
        //拷贝构造函数
        Person(const Person & p)
        {
            cout << "拷贝构造函数" << endl;
            age = p.age; // 普通数据成员简单复制一下就ok
             //如果不利用深拷贝在堆区建立新的内存 就会造成浅拷贝带来的重复释放堆区的问题
            height = new int(*p.height);
        }
        //析构函数
        ~Person()
        {
            //标准清空类构建的堆区间写法
            if(height != NULL)
            {
                delete height;
                  height = NULL;
            }
            cout << "析构函数" << endl;
         }

        int age;
        int *height;
};
```

2.3.3析构函数

- 不能重载
- 后构建先析构

2.5 对象指针

普通指针不再赘述 只要知道虚基类指针可以实现动态多态即可

2.5.1 this指针

- 只有<非静态> <成员> 函数才有 this 指针 (一定要清楚)
- 编译器会在每一个非静态成员函数中生成this指针

2.6静态成员

2.6.1 静态数据成员

- 存储空间并不在类内 而是在全局变量的地方 其本质就是被类所共享的类内全局变量
- 定义在类内： static int count;
- 初始化在类外　　方式：数据类型 类名:: count = 0；//和正常的函数定义格式是一样的

2.6.2 静态成员函数

- 静态成员函数定义在类内：static int 函数名();
- 调用可以通过类名::函数名的方式， 也可以通过对象.函数的方式
- 静态成员函数一般来说不能访问普通数据成员函数 只能访问非静态成员函数 （但是你要是非得做也可以）
- 无this指针

2.7友元关系

就是为了解决一个类的private成员无法被类外调用的问题

2.7.1 友元函数

friend 返回值类型 + 函数名字 + (形参列表);

- 友元函数并不是类的成员函数 只是一个普通的函数
  - friend .....可以放在类的任何位置
  - 调用的时候直接调用 不需要指明对象或对象指针
- 友元函数没有this指针 所以一般来讲：友元函数的形参里都有对象来点明对哪个对象进行的操作

2.7.2 友元类

- 访问该类中所有private数据成员
- 准确来说是说有的成员

2.8共享数据的保护 const

2.8.1 常引用

- void print(const int & i); i作为形参将实际参数换了名字 保证不改变实参的值

2.8.2 常对象

- 常对象在声明的时候就要初始化 因为在此之后不能做任何更改
- 常对象没有 this 指针 （在类内数据修改成员的函数对于常对象来说都不可以调用）
- 常对象只能调用常成员函数 以及类的静态成员函数
  - 这一点容易出辨析：常对象只能访问不改变成员值的函数 （×） 一些非常成员函数也不改变成员值，但是常对象只能访问常成员函数和静态成员函数。

2.8.3 常成员

- 常成员函数
  - 返回值类型 函数名（形参表） const
  - 只区别有无const修饰的时候也可以视为函数重载
- 常数据成员
  - 构造函数对常数据成员进行初始化时 <必须> 通过<初始化列表>进行 而不能通过赋值来进行
  - 对象创建之后，常数据成员的值不能在任何函数中被改写
  - 如果有多个重载构造函数都必须初始化常数据成员

第三章 类的继承与派生

设计一个派生类分为三部分->吸收基类成员、改造基类成员、新增自己的成员

继承方式（了解其核心内涵->类外访问权限，派生类访问权限）

3.3 派生类的构造函数和析构函数

3.3.1构造函数

- 对基类成员和子对象成员（组合类定义的对象）的初始化必须通过初始化列表的方式，新增成员初始化既可以用列表，也可以在构造函数体中进行。
- 派生类构造函数初始化顺序：先调用基类构造函数初始化基类数据成员，如果有子对象，随后进行子对象的初始化，最后初始化派生类新增成员。
- **当派生类有多个基类时，处于同一层次的各个基类的构造函数的调用顺序取决于基类的声明顺序**
- 如果派生类的基类也是一个派生类，则每个派生类只需要负责其直接基类的构造函数的初始化
- 派生类构造函数负责调用基类构造函数并为其提供所需的参数，因此如果基类的构造函数定义了一个或多个参数那么派生类就要有构造函数
- 对上一点：如果说基类默认构造函数，子类没有初始化成员那么我们在子类中也不需要有构造函数

3.3.2 析构函数

- 和构造顺序完全相反：先调用派生类的析构函数析构派生类对象新增部分，如果该派生类含有子对象接下来就析构子对象，最后调用基类的析构函数析构基类部分
- 一定要明确这个完全相反的概念，就是销毁的顺序和构造的顺序完全相反，不管是什么情况

3.4 多继承

3.4.1 多继承中的构造函数

- 应该按照派生类声明各个基类的先后顺序来逐一提供参数书写初始化列表

多继承中有二义性问题

- 从两个基类里继承了相同名字的函数
- 菱形继承

3.5 虚基类

目的：解决多继承中的二义性

思考 家具 -> 床类和沙发类 -> 床和沙发类 （在第一级派生的时候每个派生类都要将公共基类设为虚基类）

虚基类的初始化

- 同层次下虚基类的构造函数在非虚基类之前调用
- 若同一个层次包含多个虚基类，则按它们的声明顺序调用
- 若虚基类由非虚基类派生而来先调用虚基类的构造函数 再遵循上述原则

```
class A;
class B;
class C;
class D : public A, virtual public B, virtual public C
{
}

/*
初始化D的时候构造函数调用顺序：
- B()
- C()
- A()
- D()
*/
```

3.6 组合类

原本的派生是说 派生类B 是基类 A的一种， 含有A的属性自己又有延伸

但是现在组合类是说 A由B 构成。 B是A的一部分

3.6.2组合类中的构造函数 和 析构函数

这个地方的调用顺序 上面派生类中描述的已经很清楚了

```
class Date
{
private:
    int day;
    int year;
}
class People : public c
{
public:
```

```
    People(int a, int b, int d, int y, int x): c(a, b) : birthday(d, y) : x(x)
{};
private:
    Date birthday;
    int x;
}
```

第四章 多态性

4.1 概述

- 静态多态性：<编译程序>时绑定所要调用的函数体来实现，属于静态绑定。在编译阶段就确定通过函数名所调用的函数体。通过<函数重载>来实现
- 动态多态性：在运行程序时确定绑定对象，属于动态绑定。在编译时无法确定绑定对象，必须要到程序运行时才能关联到需要执行的函数体，到程序运行时才确定。 通过<虚函数> <继承>来实现

4.2 虚函数

- 基类中定义了虚函数，派生类中的同名函数也一定是虚函数。
- 通过 <指向**基类**对象的指针或引用> 来实现动态调用
- 虚函数必须是非静态成员函数（静态成员函数本质上不属于本成员，并且属于静态联编）
- 派生类中重新定义虚函数的时候要求完全相同，一模一样
- 内联函数也不能当作虚函数
- 构造函数不能是虚函数，但是析构函数一般都声明为虚函数

4.2.3 虚函数隐藏和覆盖：

- 覆盖就是虚函数特性
- 隐藏只不过是因为函数重载且定义域造成的无法访问罢了

4.2.4 虚函数具有传递性

- 多重继承中基类派生的所有派生类原型相同的函数都是虚函数

4.2.5 虚析构函数

- 创建一个父类指针 指向 new 出来的一个派生类对象

  如果不将父类中的析构函数设置为虚函数 那么 delete时 就不会销毁派生类对象（不会调用派生类的析构函数 只会调用父类的析构函数）

- 构造函数 拷贝函数 析构函数都要自己设计之后才能被继承

  构造函数对数据成员初始化时尽量采用初始化列表的方式 -> 速度快

4.3 抽象类

4.3.1 纯虚函数

virtual + 返回值 + 函数名(参数列表) = 0;

4.3.2 抽象类

- 含有纯虚函数的类叫做抽象类，抽象类只能用于派生新的类，不能够用来创建对象。
- 如果创建出来的新类部分纯虚函数没有得到实现，仍然是抽象类
- 可以使用虚基类指向抽象类的指针或引用，以支持运行时的多态性

4.5 运算符重载

4.5.1 运算符重载的意义

- 运算符重载实际上就是函数重载，属于静态多态性。

4.5.2 运算符重载的规则

- 运算对象至少有一个用户自定义的数据类型
- 不能违背运算符原来的语法规则 如不能改变操作数个数、不能改变符号优先级和结合性
- 只能用已经存在的 不能自创
- 不能完全失去原本的意义 +号你不能重载成减号的意思
- sizeof . * :: ?不能重载
- = () [ ] -> 必须类内重载
- << >> 必须类外重载

4.5.3 成员函数和非成员函数重载

- 成员函数运算符函数重载
  - 对象本身是一个隐含运算符的操作数 所以形参列表里面就少了1个操作数
  - **<返回值> operator 运算符号<参数>**
- 非成员运算符函数重载
  - 非成员函数的形参个数与原运算数相同
  - 使用friend表示非成员运算符重载做友元函数
- 重点掌握以下几个重载

```
//1.自增自减
int operator++(); // 不带操作数表示前置
Ran& operator++();
int operator++(int); // 带操作数表示后置
Ran operator(int);

//2. <<
ostream& operator<<(ostream &out, Person &p)
{
    out << "a = " << p.a << " b = " << p.b;
    return out;
}
//3. >>
istream& operator>>(istream &in, Person &p)
{
    in >> p.a >> p.b;
    return in;
}
//注解：输入输出运算符必须重载为非成员函数 且一般声明他是类的友元函数以便于访问类中的私有成员

//4.类型转换运算符 想让一个基本数据类型转换为类类型 不存在这样的转换 故需要自己写
```

```
operator    double()    const { return x; }
          转化后的类型名          转化的对象
```

第5章模板

5.1 函数模板和模板函数

```
template <模板类型形参列表>
<返回值类型> <函数名>（函数形参列表）
{
    函数体
}
// <模板类型参数表>中每一个类型参数前都需要添加前缀class
```

5.2 重载函数模板

- 重载函数模板必须要以<参数个数>上的差异来与一般重载函数进行区别 //要求更加严格
- 匹配顺序
    - 先匹配类型完全相同的普通重载函数
    - 再匹配重载模板函数

5.3 类模板以及模板类

```
//声明类模板
template <class Type>
class TClass
{
public:
    TClass (Type);
    void SetItem(Type);
private:
    Type item;
}

//定义函数
template <class Type>
TClass<Type> :: TClass(Type n)
{
    item = n;
}

template <class Type>
TClass <Type> :: SetItem(Type n)
{
    item = n;
}

int main()
{
    TClass<int> obj1(200); //实例化对象的时候必须要有准确的类型
}
```

5.3.2 类模板的派生

两种派生方式

- 从类模板直接派生出新的模板类
- 从类模板生成的模板类（确定类型的）派生出非模板类

```
template <class Type>
class Parent
{
}
//way 1
template <class Type>
class Son1 : public Parent <Type>

//way 2
class Son2 : public Parent <int>
```

第7 章输入输出

7.1 基础

- 用cout << 输出时时间上先将这些数据传送到程序中的输出缓冲区保存，直到缓冲区满了 遇到endl 或程序已结束，这是才将缓冲区的全部数据传送到显示器上显示出来
- I/O 库中的类被称为流类 抽象基类 ios 派生出来的一些类的也要了解
- 键盘、屏幕和内存的交互都可以通过流对象来实现

7.2 cout

- 除了使用cout以外 cout.put() 也可以实现单个字符的输出

7.3 cin

- 输入流对象.get() 可以用于输入单个字符 <cin只不过是一种输入流对象 我们可以自己命名很多种>
- 输入流对象.getline(字符指针, 字符个数)； 读取一行或确定数的字符 读到尾部也会返回eof
- 输入流对象.eof() 检验是否到达文件尾部 如果到达其为1 不到则为0

7.4 文件操作与文件流

- 文件类
    - ifstream
    - ofstream
    - fstream
- 文件的打开关闭操作
    - 打开方式

```
ios::in -> 以输入的方式打开文件 1.如果不存在则出错 2.否则打开成功 ifstream的默认打开方式 文件指针当前位置在文件的开始处
ios::out -> 以输出的方式打开文件 1.如果文件不存在则创建一个新文件 2.如果文件存在则清空文件 ofstream的默认打开方式 文件指针在文件的开始处
ios::app -> 以追加的方式打开文件 1.如果文件不存在则出错 2.否则打开成功 文件指针在文件的结尾处
ios::binary -> 以二进制方式打开文件 如果不指定则默认为文本方式

ios::out | ios::in 以两种方式打开
```

- 文本文件读写的具体方法

```
#include <iostream>
#include <fstream> // step1 引入加载头文件

int main()
{
//-------------写文件------------------//
    ofstream ofs; //step2 定义输出对象
    ofs.open("name of file", ios::out); //step3 打开文件
    if(ofs.fail()) //step4 检验文件是否打开成功
    {
        cout << "open file false";
    }

    ofs << "......."; //step5 向文件流里放东西
    ofs.close(); //step6 关闭文件

//-------------读文件-----------------//
    ifstream ifs;
    ifstream ifs("name of file", ios::in);
    if(ifs.fail())
    {
        ...
    }

    //--------多种方式读出来-----------//
    //way1 直接用 << 读 一次只能读一个字符
    while(!ifs.eof())
    {
        f >> x; // x必须和文件里的数据类型相符
        cout << x << " ";
    }
   //
    char buf[1024] = {0};
    while(ifs>>buf)
    {
        cout << buf << " ";
    }
    //way2 用函数来读
    //-1 get //一个字符一个字符来读 效率低下
    char c;
    while(c = ifs.get() != eof)
    {
        cout << c;
    }
```

```
    //-2 getline() 把一行当作一个字符串来处理 效率最高
    string buf;
    while(getline(ifs, buf))
    {
        cout << buf;
    }
    //-3 getline()
    char buf[1024];
    while(ifs.getline(buf, sizeof(buf)))
    {
        cout << buf;
    }
}
```

- 二进制文件的具体读写方法<重点>

  用read() 和 write() 读写文件

```
f.open("r.dat", ios::out|ios::binary);
       读入写入的数据指针指示起始位置   读入写入的数据大小
f.write((char*)&a,                  sizeof(...));
f.read ((char*)&b,                  sizeof(...));
```

  常用函数

  ----输入

  - tellg() 返回输入文件指针当前的位置
  - seekg(位置) 移动输入文件指针到某位置
  - seekg(位移量， 参照位置) 确定位置

  ----输出

  - tellp() 返回输出指针当前的位置
  - seekp(位置) 移动输出文件指针到某位置
  - seekp(位移量， 参照位置) 确定位置

  ----参照物

  - ios::beg 开头
  - ios::cur 当前位置
  - ios::end 结尾

new&delete重载 参考(2条消息) C++动态内存：（二）重载new和delete_zxx910509的博客-CSDN博客_重载new

刷有所得：

Q1

```
struct MyClass{ int num; };
```

则MyClass结构体的成员num是(公有数据成员)。

别被虚晃了

Q2

假定CTest为一个类，则执行“CTest a[10]；”语句时，系统自动调用该类的构造函数的次数为10。

Q3

销毁对象时 对象里的静态数据成员也会被销毁