# 一、c++特性

## 1.程序通过命令行窗口的各类输入

用scanf或是cin遇到空格就会停止输入，无法输入带空格或是空字符串，因此需要使用**getline(cin, str)**来输入字符串。

## 2.c++中struct和class区别

（1）默认成员属性：struct中默认成员属性是public，class中默认成员属性是private；

（2）默认继承方式：struct默认继承方式是public，class默认继承方式是private；

（3）模板使用：class可以用于定义模板参数，struct不用于定义模板参数，如template<typename T>中typename可以是一个类，但不能是一个结构；

（4）初始化：struct由于是c的延伸，在没有构造函数时，可以使用{}列表来赋初值。

## 3.c++多态性（ali）

多态性指的是多种形态，即**使用相同的接口访问不同功能的函数**，包括有**重载多态**、**强制多态**（将一个变元类型加以变化）、**参数多态**（使用函数模板支持不同形参避免重载函数的函数体重复设计）和主要由虚函数实现的**包含多态**。

### 1）强制多态

编译程序通过语义操作，把操作对象的类型强行加以变换，以符合函数或操作符的要求。通常对不同类型变量进行操作符计算时会进行强制多态。例如int+double的操作中会把int转化为double再进行计算。

### 2）重载多态

同一个操作符或函数在不同上下文中有不同类型。例如double+double和int+int。

### 3）包含多态

同一个操作可用于一个类型及其子类型（不是子类）

包含多态实现的是当一个指针指向子类对象时，其调用的函数就是子类变量的。这使得函数能够根据传入参数的不同，具有不同的功能。

举例：void drink(Animal \*a)，如果不使用多态，则在实现该函数时需要针对每种动物子类定义一个重载，例如drink(Dog \*d)。

包含多态主要由虚函数实现。

### 4）参数多态

采用参数化模板（template），通过给出不同的类型参数，使得一个结构有多种类型。(类似模板类吧!)

## 4.重载、重写、重定义

重载即overload，是函数名相同，参数列表不同

重写即override，也叫作覆盖，派生类重新定义基类中有相同名称和参数的虚函数。需要注意的是：1.被重写的函数不能是static的；2.重写函数必须有相同的类型，名称和参数列表；3.重写函数的访问修饰符可以不同，甚至基类中的private在派生类中改写为public或protected也是可以的。

重定义也叫作隐藏，是派生类重新定义基类中有相同函数、相同返回值类型且有相同参数的函数。

## 5.使用未定义的类

当我们定义两个类的时候，前面一个类使用了后面一个类的时，需要在前面的类的前面声明这个类，并且**由于后面的类还没有具体实现，因此在前面一个类中，不能将类的具体对象作为成员，但可以将类的指针作为成员**。

另外，对于前面一个类，不能直接在类的定义中直接使用后面一个类的成员变量或是成员函数，而是要把**相关成员函数的具体实现放在后面相关类的定义的后面，因为类的具体定义还没出现时无法使用类的成员变量或成员函数**。

## 6.友元函数

在定义一个类的时候，可以把一些函数（包括全局函数和其他类的成员函数）声明为“友元”，这样那些函数就成为该类的友元函数，在**友元函数内部就可以访问该类对象的私有成员**了。

将全局函数声明为友元的写法：

friend 返回值类型 函数名(参数表);

将其他类的成员函数声明为友元的写法：

friend 返回值类型 其他类的类名::成员函数名(参数表);

但是不能把其他类的私有成员函数声明为友元。

## 7.类成员变量初始化顺序

类的成员变量初始化分为三步，首先在成员变量声明处初始化，然后在构造函数的初始化列表中初始化，最后在构造函数内初始化。后面的初始化操作会覆盖前面的操作结果。

## 8.const成员变量

const成员变量**只能在成员变量声明处以及构造函数初始化列表中**进行初始化。

## 9.const修饰的成员函数

当const在成员函数名前时，const修饰的是返回值的类型；

当const在成员函数名后时，该成员函数**不能修改非静态成员变量的值**。

## 10.static成员变量

**static修饰的类成员均属于类，不属于某个具体的对象**。

静态成员变量是一种特殊的成员变量。创建多个对象，也只会为其**单独**分配一份内存，并且不占用对象内存，所有对象使用的都是这份内存中的数据。当某个对象修改了该成员变量，也会影响到其他对象。

静态成员变量**必须初始化，且必须在类声明的外部初始化**，静态成员**常量(const修饰的量)也必须初始化，但可以在类声明的内部或是外部初始化，这两者初始化时可以赋值也可以不赋值默认初始化为0**。而**非静态成员变量不能在类外初始化**。

静态成员变量既可以通过对象名访问，也**可以通过类名访问**。

## 11.static成员函数

static成员函数不属于对象所以**不能使用this指针**。由于没有this指针，static类成员函数**不能访问非static修饰的类成员**。

## 12.protected、private构造函数与析构函数

构造函数是protected或是private成员函数时，无法从外部调用。因此有两种使用方法：

（1）.使用友元对象调用构造函数来调用，以创建对象；

（2）.在static修饰的public成员函数内可以创建对象。

## 13.继承时报错：基类不存在默认的构造函数

**派生类的构造函数必须在函数一开始时（函数名后面用:来调用）调用基类的一个构造函数**，在有无形参构造函数时，可以在没有指定调用基类的构造函数时自动执行无形参基类构造函数。

## 14.继承时公有继承、保护继承、私有继承都有什么区别

|  |  |  |  |
| --- | --- | --- | --- |
| 继承方式 | 基类的public成员 | 基类的protected成员 | 基类的private成员 |
| public继承 | public成员 | protected成员 | 不可见 |
| protected继承 | protected成员 | protected成员 | 不可见 |
| private继承 | private成员 | private成员 | 不可见 |

## 15.继承私有变量

基类的私有变量子类**可以继承但无法直接访问**，需要使用基类的接口函数访问。

## 16.基类指针作为函数形参

当某函数形参包含某基类指针时，若不加引用&，则可以将其派生类指针作为函数输入调用该函数，此时该指针输入后会被强制类型转换，但继承时的非虚函数的成员函数的重载会失效而调用基类的成员函数；若**添加引用&，则无法将派生类指针作为函数输入**，否则会报错。

## 17.虚函数是什么（ali）

在某基类中声明为virtual并在一个或多个派生类中被重新定义的成员函数为虚函数。

虚函数在定义时，基类结构中增加了一个虚函数指针表virtual table，该表存放各虚函数的调用地址，大小由虚函数个数决定，该表的首地址由基类内部指针vPtr指向。

当创建派生类的时候，每个派生类也会创建一个虚函数表。继承时若重定义了虚函数，则对应位置的虚函数地址会被修改为新的，否则维持基类的虚函数位置。

多重继承会为派生类创建多个虚函数表，对应每一个基类创建一个虚函数表。

另外，**即使派生类没有重定义基类的虚函数，也会为该派生类对应该基类创建一个新的虚函数表**，即派生类不会和基类公用虚函数表，但对于没有重定义的虚函数来说，虚函数表会记录相同的虚函数地址。

## 18.一般成员函数在派生类中的重定义

与虚函数不同，一般成员函数在对象中不会占用内存空间存储。因此在派生类重定义后，会在强制类型转换后无法调用派生类的该成员函数。

## 19.对象的内存分布

空类的对象占用1字节；类的一般成员函数不会占用对象内存空间，函数本身会单独存储在代码区，具体对象对象不会存储一般成员函数地址；对象会存储虚函数表。类和结构体一样，带有4字节对齐功能。总结来说，对象的内存分布如下：

|  |
| --- |
| 基类0的成员虚函数对应的虚函数表地址 |
| 基类0成员变量 |
| 基类1的成员虚函数对应的虚函数表地址 |
| 基类1成员变量 |
| … |
| 基类n成员虚函数对应的虚函数表地址 |
| 基类n成员变量 |
| 成员变量 |

## 20.构造函数可以是虚函数吗（ali）

构造函数不能是虚函数。

构造一个对象的时候，必须知道对象的实际类型，而虚函数行为是在运行期间确定实际类型的。而在构造一个对象时，由于对象还未被构造成功。编译器无法知道对象的实际类型，是该类本身，还是该类的一个派生类，或是更深层次的派生类。

而且，构造函数使用虚函数没有意义，因此没有设计相应的机制。

## 21.为什么基类的析构函数大家都喜欢用虚函数

由于多态的存在，当一个基类函数的指针指向派生类对象时，如果不把析构函数设计为虚函数则在析构该派生类对象时，析构该对象时会调用基类的析构函数，无法正确地调用析构函数。

## 22.c++stl库各类的特性和内存分配

（1）vector（allocator）

支持快速随机访问，因此内存是连续排布的。插入元素时，先会检查内存分配**（capacity）**是否足够，足够则使用分配的内存空间，否则**重新分配1.5倍或2倍于当前vector容量大小的内存**；如果无法在当前位置分配空间，则寻找新的空间供使用，并释放原先的空间。

若使用reserve或是resize函数则是按函数参数修改内存分配。

clear和erase会清除数据，但**不会回收内存**。

（2）string（basic\_string(allocator)）

与vector类似。

在visual studio上，内存处理有区别：初始化后vector按照初始化参数分配内存，而string直接分配15个字节的内存；内存不足时，vector每次会分配1.5倍内存，而string第一次扩容分配2倍内存，之后每次分配1.5倍内存。

（3）list（allocator）

双向链表。不使用连续的空间，因此无法随机存取，但可以以较低的复杂度实现任意位置的插入和删除的操作。

（4）deque（allocator）

双端队列。内存分配是**分段连续**的，并且维持着“整体连续”的使用方法，分配内存时**按“页”或“块（chunk）”分配存储器**。可以实现在开头和结尾插入元素达到O(1)时间复杂度（优于vector），也可以实现随机访问达到O(1)时间复杂度（优于list）。

（5）stack（deque）

栈。无法随机存取，只能后入先出（LIFO）。

（6）queue（deque）

队列。无法随机存取，只能先入先出（FIFO），但在stl库里提供了读取最后进入的元素信息的功能。

（7）set（allocator）

哈希集合。内存不连续。

（8）map（allocator）

哈希表。内存不连续。各元素按大小顺序排布。

（9）unordered\_map（allocator）

无序哈希表。各元素是无序的。

（10）priority\_queue（container(vector/deque)）

本质是一个最大堆/最小堆。**可以以O(log n)的效率查找一个队列中的最大值或者最小值**。由于使用的容器是vector/deque，因此分配的内存应该是连续的。

## 23.::开头的类名

指再前面没有namespace，类似于地址里面的/。

## 24.拷贝构造函数

1.拷贝构造函数声明和定义

复制构造函数也是构造函数，但它只有一个参数，这个参数是本类的对象(不能是其他类的对象)，而且采用对象的引用的形式(一般约定加const声明，使参数值不能改变，以免在调用此函数时因不慎而使对象值被修改)。如果用户自己未定义复制构造函数，则编译系统会自动提供一个默认的复制构造函数，其作用只是简单地复制类中每个数据成员。

声明：类名（类名 &对象名）;

定义：类名::类（类名 &对象名） {…}//拷贝构造函数的实现

2.浅拷贝和深拷贝

所谓浅拷贝，指的是在对象复制时，只对对象中的数据成员进行简单的赋值，默认拷贝构造函数执行的也是浅拷贝。大多情况下“浅拷贝”已经能很好地工作了，但是一旦对象存在了动态成员（指针），那么浅拷贝就会出问题了。浅拷贝只会简单拷贝指针值，而不会申请新的空间复制指针指向的内容。

如果需要避免浅拷贝，则需要自行定义拷贝构造函数实现深拷贝，为每一个指针成员变量申请新空间复制指针指向的内容。

3.调用的时机

1）当函数的参数为类的对象时，函数的形参是用拷贝构造函数生成的

2）函数的返回值是类的对象时，返回的对象会通过拷贝构造函数输出给调用时指定的变量，然后析构函数里该返回值对象

3）对象需要通过另外一个对象进行初始化，如SampleClass B = A;

## 25.const和constexpr

constexpr是c++11引入的。

对于修饰Object来说，

const并未区分出编译期常量和运行期常量。

constexpr限定在了编译期常量。

constexpr修饰的函数，返回值不一定是编译期常量。

## 26.静态全局变量和静态局部变量

（1）Static局部变量指定义在函数内的静态变量。Static局部变量不会随着函数调用的结束而消失，其一直被存储于内存中，在下次同一函数再次被调用时可继续使用。当该变量被初始化后，例如static int i=1;i++;，那么下次再调用该函数，并不会再次为该变量赋1，而是初值为上一次函数结束时的最终值2。在不赋初值的情况下，普通变量可能为任意值，但静态局部变量会初始化为0或空字符。

总结来说，静态局部变量改变了变量的存储方式（栈->静态存储区）。

（2）未定义Static的全局变量在整个程序块可见（例如由.h声明，所有包含该.h的.cpp文件），而定义了Static的全局变量仅在当前cpp内可见。

总结来说，静态全局变量限制了变量的作用域。

## 27.指针与引用

指针是一个变量，进行指针操作时是改变了目标的地址。

引用是原变量的一种重命名。其与原变量占用相同的一个存储空间。

指针VS引用：

1）指针可以为空，引用必须初始化。一旦引用已经定义，它就不能再指向其他的对象，这就是为什么它要被初始化的原因。

2）可以有const指针。不允许非const引用指向需要临时对象的对象或值，即，编译器产生临时变量的时候引用必须为const。const引用表示，**试图通过此引用去(间接)改变其引用的对象的值时，编译器会报错**！这并不意味着此引用所引用的对象也因此变成const类型了，我们**仍然可以改变其指向对象的值，只是不能通过引用**

3）可以有多级指针，但没有多级引用。

4）指针初始化后可指向其他地址，引用只能改变当前内存地址中的值。

5）Sizeof指针获得指针大小，sizeof引用获得变量大小。

6）指针的自增和引用的自增意义不一样。

# 二、c++底层

## 1.32位系统和64位系统变量长度



## 2. C++的五大内存分区

在C++中，内存分成5个区，他们分别是堆、栈、自由存储区、全局/静态存储区和常量存储区。

1）栈，就是那些由编译器在需要的时候分配，在不需要的时候自动清楚的变量的存储区。里面的变量通常是局部变量、函数参数等。

2）堆，就是那些由new分配的内存块，他们的释放编译器不去管，由我们的应用程序去控制，一般一个new就要对应一个delete。如果程序员没有释放掉，那么在程序结束后，操作系统会自动回收。

3）自由存储区，就是那些由malloc等分配的内存块，他和堆是十分相似的，不过它是用free来结束自己的生命的。

4）全局/静态存储区，全局变量和静态变量被分配到同一块内存中，在以前的C语言中，全局变量又分为初始化的和未初始化的，在C++里面没有这个区分了，他们共同占用同一块内存区。

5）常量存储区，这是一块比较特殊的存储区，他们里面存放的是常量，不允许修改

## 3.new与malloc的区别

malloc是C语言中申请内存空间的函数。需要使用free释放，在申请时无法初始化。

new是C++中引入的操作符，其能够根据数据类型，自动计算所需空间，返回对应类型指针，并在所开辟空间执行构造函数。而与之配套的delete会在释放空间前调用析构函数进行处理。因此使用new更加安全。

## 4.C++编译运行过程

C++的编译运行过程主要包含四步：**预编译**->**汇编**->**编译**->**链接**。

1）预编译主要是去除程序中的宏定义并展开头文件。

2）汇编过程将程序转化为汇编代码。

3）编译过程进一步将汇编代码转化为目标代码，即二进制文件。

4）链接过程将前面过程中单个文件的处理结果进行解析，关联不同的单元文件和应用到的库文件，例如用到的库函数、其他单文件中定义的全局变量等。

## 5. 静态库与动态库

Windows中静态库为.lib，动态库为.dll；Linux中静态库为.a，动态库为.so。

静态库与动态库的区别主要在链接阶段。静态库会在链接时与目标文件一同被合成为一个可执行文件，而动态库则是在程序运行时被载入。

### 1）静态库

**优势**：

1）程序在运行时与函数库无关，方便移植。

**劣势**：

1） 空间浪费。不同可执行文件包含相同库时，每个可执行文件都占用额外的存储空间。

2）内存资源浪费。如果两个程序包含相同的静态库，其会被传入内存两份用于各自程序的运行。

3）库程序更新时，需要重新编译所有文件并发布给客户。

### 2）动态库

**优势**：

1）可执行文件占用空间小。

2）内存中每个库最多读取一份。

3）库程序更新时直接覆盖之前的动态库即可。

劣势：

1）程序移植时需要考虑动态库的调用情况。

## 6. 面向对象三大特征：封装、继承、多态

封装：封装是一个概念，它的含义是把方法、属性、事件集中到一个统一的类中，并对使用者屏蔽其中的细节问题。数据被保护在抽象数据类型的内部，尽可能地隐藏内部的细节，只保留一些对外接口使之与外部发生联系。

继承：如果两个类存在继承关系，则子类会自动继承父类的方法和变量，在子类中可以调用父类的方法和变量，如果想要在子类里面做一系列事情，应该放在父类无参构造器里面。