C++常见面试问题

https://blog.csdn.net/qq_43365825

https://blog.csdn.net/chasinguu/article/details/99611071

指针和引用的区别

- 引用必须初始化，与某个对象绑定，并且不可以将其重新指向另一个对象，但是指针可以不初始化或者初始化为NULL
- 引用是变量的别名，而指针指向的是地址
- 指针和引用的自增(++)运算意义不一样(前者运算的是内存地址，后者是改变其引用对象的值)
- (const int *p为指针指向的是常量，不能通过指针来改变其指向的值，int * const p为常量类型的指针，不能修改指针的指向，但可以修改其指向的值)

  (const 引用的目的是禁止通过修改引用值来改变被引用的对象，但对象其本身的值是可以改变的)
- 指针可以有多级，引用只能是一级。
- sizeof引用是对象的大小，sizeof指针是指针的大小。

堆和栈的区别

- 管理方式：栈由编译器自动管理，堆由程序员控制
- 碎片问题：栈不会产生碎片(先进后出)，堆由于频繁的new/delete会产生大量的碎片
- 生长方式：堆向上增长，栈向下增长
- 分配方式：堆通过new malloc等动态分配，栈分为静态分配和动态分配(alloca)但是其动态分配的资源由编译器释放
- 空间大小：堆是不连续的内存区域，其空间比较大，也很灵活，栈是连续区域，其大小一般是由操作系统预定好的
- 分配效率：堆由C/C++函数库提供支持，如果没有足够的大小（可能是因为碎片太多），会向操作系统申请更多的内存；而栈由计算机底层提供支持，%rsp,%rbp放置栈顶和栈底的地址，还有push,pop等命令，效率要高得多

new和delete是如何实现的，new 与 malloc的异同处

new 实现：

- 调用operator new 申请空间(operator new实际上通过malloc来申请空间)

  对于 new T[N] 调用operator new[] ()，operator new[] ()中调用operator new
- 在申请的空间上执行构造函数，完成对象的构造

delete 实现：

- 在空间上执行析构函数
- 调用operator delete(该函数也是通过free来释放空间)

异同：

- 属性：new 是 C++关键词，而malloc 是库函数，需要Include相应的头文件（stdlib）
- 参数：new申请内存无须显式指定内存块大小，编译器会根据类型信息自行计算，而malloc需要显式指出需要分配的内存

- 返回类型：new分配成功后，返回对象类型的指针，其类型安全，而malloc分配成功返回void *,需要通过强制转换
- 分配失败：new 会抛出bad_alloc异常，malloc则会返回NULL
- new会调用构造函数，见上
- 重载：new允许重载，malloc不行
- 内存区域：new 从自由存储区上为对象动态分配内存空间，而malloc则从堆上动态分配。自由存储区是C++基于new操作符的一个抽象概念

C和C++的区别

设计思想上：

C++是面向对象的语言，而C是面向过程的结构化编程语言

语法上：

C++具有封装、继承和多态三种特性

C++相比C，增加多许多类型安全的功能，比如强制类型转换、

C++支持范式编程，比如模板类、函数模板等

C++ Java的联系和区别

http://c.biancheng.net/view/5583.html

两者都是面向对象的语言

区别

- Java有垃圾回收机制，C++要求程序员使用delete来释放申请的内存
- 由于Java运行在JVM上，所以它的可移植性更好
- Java没有指针的概念
- Java没有多继承的概念

C++ struct区别

- 默认继承权限：class 按private，struct按public
- 成员默认访问权限：class 默认private权限，struct默认public权限

其他没啥区别

define 和const的区别（编译阶段、安全性、内存占用等）

- 作用阶段：define在编译预处理阶段，const则是在编译以及运行阶段
- 作用方式以及安全性：define是字符串替换，没有类型检查，而const有编译器类型检查作为保证
- 内存占用：由于在预编译阶段就进行替换，所以define的常量不占用符号表的位置，其不分配内存，而c++中const虽然也不分配内存，但是其定义会保存在符号表中

在C++中const和static的用法（定义，用途）以及使用的注意事项

static:

- 类：如果某个成员（包括数据和函数）为整个类所共有，不属于任何一个具体对象，则采用static来声明其为静态成员，可以通过类名（也可以通过对象名）进行访问

- 变量或者函数
    - 全局：在声明它的文件之外不可见，作用域从定义之处开始到文件结尾
    - 局部：局部静态变量离开局部作用域后并没有销毁，而是驻留在内存中
    - 静态函数：静态函数只在声明它的文件中可见

const:

- 修饰一般常量以及数组表示使用时不能改变常量的值
- 修饰指针和引用：见最上
- 函数：
    - 修饰函数参数：void fun(const A* a)不能对指针或者引用传递进来的对象进行改变
    - 修饰返回值：const A fun2()，一般用const修饰返回值为对象本身（非引用和指针）的情况多用于二目操作符重载函数并产生新对象的时候。如：

      ```
      const Rational operator*(const Rational& lhs, const Rational& rhs)
      {
          return Rational(lhs.numerator() * rhs.numerator(), lhs.denominator()
      * rhs.denominator());
      }
      ```

      可以防止如下的操作

      ```
      Rational a,b;
      Radional c;
      (a*b) = c;
      ```

    - 修饰类成员函数： A fun4( )const；不能修改所在类的任何变量

      const类型函数可以被调用于非const对象，但在对象是const类型时，通过该对象只能调用const类型函数

C++中的const类成员函数（用法和意义），以及和非const成员函数的区别

任何不会修改数据成员的函数都应该声明为const 类型。如果在编写const成员函数时，不慎修改了数据成员，或者调用了其它非const成员函数，编译器将指出错误，这无疑会提高程序的健壮性。

关于Const函数的几点规则：

- const对象只能访问const成员函数,而非const对象可以访问任意的成员函数,包括const成员函数.
- const对象的成员是不可修改的,然而const对象通过指针维护的对象却是可以修改的.
- const成员函数不可以修改对象的数据,不管对象是否具有const性质.它在编译时,以是否修改成员数据为依据,进行检查.
- 然而加上mutable修饰符的数据成员,对于任何情况下通过任何手段都可修改,自然此时的const成员函数是可以修改它的

https://blog.csdn.net/zheng19880607/article/details/23883437

final和override关键字

override关键字告诉编译器，该函数应该覆盖基类中的函数，如果该函数没有覆盖任何函数，则会导致编译器错误

final 关键字修饰函数，防止该函数被子类重写

拷贝初始化和直接初始化，初始化和赋值的区别

https://blog.csdn.net/linda_ds/article/details/82807006

拷贝初始化的6种情况

```
T object = other;      //(1) 当一个非引用类型T的具名变量通过 '=' 被一个表达式初始化时
T object = {other};    //(2) 当一个纯量类型T的具名变量通过 '=' 被一个括号表达式初始化时

f(other)               //(3) 当通过值传递给函数的参数时

return other;          //(4) 当函数返回一个值时

throw object;

catch(T object)        //(5) 当throw/catch 一个表达式的值时

T array[N] = {other}; //(6) 使用聚合初始化，初始化每个元素
```

直接初始化

```
T object(arg);                 // (1) 通过非空括号表达式列表或者初始化列表初始化
T object(arg1, arg2, ...);     // (1)
T object{arg};                 // (2) 初始化non-class类型的对象
T(other)                       // (3) 用函数转型或以带括号表达式列表初始化纯右值临时量
T(arg1, arg2, ...);            // (3)
static_cast<T>(other)          // (4) 用static_cast表达式初始化纯右值临时量
new T(args,...)                // (5) 用带非空初始化器的new 表达式初始化拥有动态存储期的对象
Class::Class()                 // (6) 用构造函数初始化器列表初始化基类或非静态成员
:member(args, ...) {/*...*/}
[arg]() {/*...*/}              // (7) 在lambda 表达式中从以复制捕捉的变量初始化闭包对象成员
```

extern “c”

https://www.jianshu.com/p/5d2eeeb93590

模板函数和模板类的特例化

https://blog.csdn.net/wang664626482/article/details/52372789

C++中的重载和重写的区别

https://blog.csdn.net/qq_34793133/article/details/80938099

C++多态的实现

静态多态：重载，在编译期决议

动态动态：继承重写基类的虚函数实现多太，运行时决议

具体原理：

1. 用virtual关键字申明的函数叫做虚函数，虚函数肯定是类的成员函数。

2. 存在虚函数的类都有一个一维的虚函数表叫做虚表。当类中声明虚函数时，编译器会在类中生成一个虚函数表。
3. 类的对象有一个指向虚表开始的虚指针。虚表是和类对应的，虚表指针是和对象对应的。
4. 虚函数表是一个存储类成员函数指针的数据结构。
5. 虚函数表是由编译器自动生成与维护的。
6. virtual成员函数会被编译器放入虚函数表中。
7. 当存在虚函数时，每个对象中都有一个指向虚函数的指针（C++编译器给父类对象，子类对象提前布局vptr指针），当进行test(parent *base)函数的时候，C++编译器不需要区分子类或者父类对象，只需要再base指针中，找到vptr指针即可）。
8. vptr一般作为类对象的第一个成员。

https://blog.csdn.net/qq_40840459/article/details/80195158

C++虚函数相关（虚函数表，虚函数指针），虚函数的实现原理（包括单一继承，多重继承等）（拓展问题：为什么基类指针指向派生类对象时可以调用派生类成员函数，基类的虚函数存放在内存的什么区，虚函数表指针vptr的初始化时间）

C++中类的数据成员和成员函数内存分布情况

this指针

- this指针在成员函数中使用，不能在静态函数使用（因为静态函数不属于具体的某一个对象）
- this在成员函数开始执行前构造，在执行结束后清除

析构函数一般写成虚函数的原因

将可能会被继承的父类的析构函数设置为虚函数，可以保证当我们new一个子类，然后使用基类指针指向该子类对象，释放基类指针时可以释放掉子类的空间，防止内存泄漏。

构造函数、拷贝构造函数和赋值操作符的区别

> 构造函数：对象不存在，没用别的对象初始化
>
> 拷贝构造函数：对象不存在，用别的对象初始化
>
> 赋值运算符：对象存在，用别的对象给它赋值

explicit

https://blog.csdn.net/qq_34269988/article/details/88674502

构造函数为什么一般不定义为虚函数

https://blog.csdn.net/shilikun841122/article/details/79012779

(文中更正:虚函数一般是通过父类指针或者引用调用子类的成员函数)

构造函数的几种关键字(default delete 0)

> = default：将拷贝控制成员定义为=default显式要求编译器生成合成的版本
>
> = delete：将拷贝构造函数和拷贝赋值运算符定义删除的函数，阻止拷贝（析构函数不能是删除的函数 C++Primer P450）

> = 0：将虚函数定义为纯虚函数（纯虚函数无需定义，= 0只能出现在类内部虚函数的声明语句处；当然，也可以为纯虚函数提供定义，不过函数体必须定义在类的外部）

构造函数或者析构函数中调用虚函数会怎样

综合问题看第一个，此问题看第二个

https://blog.csdn.net/qq_43365825/article/details/103291277?spm=1001.2014.3001.5501

https://blog.csdn.net/donotgogentle/article/details/107222597

纯虚函数

纯虚函数的基类为抽象类，不能实例化产生对象

注意点：

1.定义纯虚函数时，不能定义纯虚函数的实现部分。即使是函数体为空也不可以，函数体为空就可以执行，只是什么也不做就返回。而纯虚函数不能调用。

（其实可以写纯虚函数的实现部分，编译器也可以通过，但是永远也无法调用。因为其为抽象类，不能产生自己的对象，而且子类中一定会重写纯虚函数，因此该类的虚表内函数一定会被替换掉，所以可以说永远也调用不到纯虚函数本身）

2."=0"表明程序将不定义该函数，函数声明是为派生类保留一个位置。“=0”的本质是将指向函数体的指针定为NULL。

3.在派生类中必须有重新定义的纯虚函数的函数体，这样的派生类才能用来定义对象。（如果不重写进行覆盖，程序会报错）

静态类型和动态类型，静态绑定以及动态绑定

https://www.cnblogs.com/lizhenghn/p/3657717.html

- 静态类型：对象在声明时采用的类型，在编译期既已确定；
- 动态类型：通常是指一个指针或引用目前所指对象的类型，是在运行期决定的；
- 静态绑定：绑定的是静态类型，所对应的函数或属性依赖于对象的静态类型，发生在编译期；
- 动态绑定：绑定的是动态类型，所对应的函数或属性依赖于对象的动态类型，发生在运行期；

1. 静态绑定发生在编译期，动态绑定发生在运行期；
2. 对象的动态类型可以更改，但是静态类型无法更改；
3. 要想实现动态，必须使用动态绑定；
4. 在继承体系中只有虚函数使用的是动态绑定，其他的全部是静态绑定；

深拷贝和浅拷贝

拷贝有两种：深拷贝，浅拷贝

当出现类的等号赋值时，会调用拷贝函数 在未定义显示拷贝构造函数的情况下，系统会调用默认的拷贝函数——即浅拷贝，它能够完成成员的一一复制。当数据成员中没有指针时，浅拷贝是可行的。 但当数据成员中有指针时，如果采用简单的浅拷贝，则两类中的两个指针将指向同一个地址，当对象快结束时，会调用两次析构函数，而导致指针悬挂现象。 所以，这时，必须采用深拷贝。 深拷贝与浅拷贝的区别就在于深拷贝会在堆内存中另外申请空间来储存数据，从而也就解决了指针悬挂的问题。 简而言之，当数据成员中有指针时，必须要用深拷贝。

对象复用和零拷贝

对象复用指得是设计模式，对象可以采用不同的设计模式达到复用的目的，最常见的就是继承和组合模式了。

零拷贝：零拷贝主要的任务就是避免CPU将数据从一块存储拷贝到另外一块存储，主要就是利用各种零拷贝技术，避免让CPU做大量的数据拷贝任务，减少不必要的拷贝，或者让别的组件来做这一类简单的数据传输任务，让CPU解脱出来专注于别的任务。这样就可以让系统资源的利用更加有效。

零拷贝技术常见linux中，例如用户空间到内核空间的拷贝，这个是没有必要的，我们可以采用零拷贝技术，这个技术就是通过mmap，直接将内核空间的数据通过映射的方法映射到用户空间上，即物理上共用这段数据。

内存泄露，如何检测

https://blog.csdn.net/wza_1992/article/details/82415755

为什么用成员初始化列表会快一些?

首先把数据成员按类型分类
1。内置数据类型，复合类型（指针，引用）
2。用户定义类型（类类型）

分情况说明：
对于类型1，在成员初始化列表和构造函数体内进行，在性能和结果上都是一样的
对于类型2，结果上相同，但是性能上存在很大的差别

因为类类型的数据成员对象在进入函数体是已经构造完成，也就是说在成员初始化列表处进行构造对象的工作，这是调用一个构造函数，在进入函数体之后，进行的是 对已经构造好的类对象的赋值，又调用个拷贝赋值操作符才能完成（如果并未提供，则使用编译器提供的默认按成员赋值行为）

简单的来说：
对于用户定义类型：
1）如果使用类初始化列表，直接调用对应的构造函数即完成初始化
2）如果在构造函数中初始化，那么首先调用默认的构造函数，然后调用指定的构造函数

所以对于用户定义类型，使用列表初始化可以减少一次默认构造函数调用过程。

https://blog.csdn.net/JackZhang_123/article/details/82590368

C++的四种强制转换

> static_cast
>
> dynamic_cast
>
> const_cast
>
> reinterpret_cast

C++中将临时变量作为返回值的时候的处理过程（栈上的内存分配、拷贝过程）

https://blog.csdn.net/ryhybx/article/details/82847389

voitaile

volatile 关键字是一种类型修饰符，用它声明的类型变量表示可以被某些编译器未知的因素更改，比如：操作系统、硬件或者其它线程等。遇到这个关键字声明的变量，编译器对访问该变量的代码就不再进行优化，从而可以提供对特殊地址的稳定访问。声明时语法：**int volatile vInt;** 当要求使用 volatile 声明的变量的值的时候，系统总是重新从它所在的内存读取数据，即使它前面的指令刚刚从该处读取过数据。而且读取的数据立刻被保存。

inline和宏定义的区别

（1）内联函数在编译时展开，宏在预编译时展开；
（2）内联函数直接嵌入到目标代码中，宏是简单的做文本替换；
（3）内联函数有类型检测、语法判断等功能，而宏没有；
（4）inline函数是函数，宏不是；
（5）宏定义时要注意书写（参数要括起来）否则容易出现歧义，内联函数不会产生歧义；

public，protected和private访问权限和继承

访问范围：

private: 只能由该类中的函数、其友元函数访问,不能被任何其他访问，该类的对象也不能访问.
protected: 可以被该类中的函数、子类的函数、以及其友元函数访问,但不能被该类的对象访问
public: 可以被该类中的函数、子类的函数、其友元函数访问,也可以由该类的对象访问

注：友元函数包括两种：设为友元的全局函数，设为友元类中的成员函数
父类与其直接子类的访问关系如上，无论是哪种继承方式（private继承、protected继承、public继承）。

对于三种继承关系的不同：
public继承：public继承后，从父类继承来的函数属性不变（private、public、protected属性不变，）。
private继承：private继承后，从父类继承来的函数属性都变为private
protected继承：protected继承后，从父类继承过来的函数，public、protected属性变为protected，private还是private。

C++和C的类型安全

c的类型安全只在局部上下文中体现，c只在结构体指针转换上会报错，并且有隐式转换，malloc和new相比也并不是类型安全的。
c++类型安全：
（1）操作符new返回的指针类型严格与对象匹配，而不是void；
*（2）C中很多以void*为参数的函数可以改写为C++模板函数，而模板是支持类型检查的；
（3）引入const关键字代替#define constants，它是有类型、有作用域的，而#define constants只是简单的文本替换；
（4）一些#define宏可被改写为inline函数，结合函数的重载，可在类型安全的前提下支持多种类型，当然改写为模板也能保证类型安全；
（5）C++提供了dynamic_cast关键字，使得转换过程更加安全，因为dynamic_cast比static_cast涉及更多具体的类型检查。