C++面试题-2

11、堆和栈的区别

1. 堆栈空间分配区别：

   栈（操作系统）：由操作系统自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈

   堆（操作系统）：一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收，分配方式类似于链表

2. 堆栈的缓存方式区别

   栈：是内存中存储值类型的，大小为2M（Windows、Linux下默认为8M，可以更改），超出则会报错：内存溢出

   堆：内存中，存储的是引用数据类型，引用数据类型无法确定大小，堆实际上是一个在内存中使用到内存中零散空间的链表结构的存储空间，堆的大小是由引用类型的大小直接决定的，引用类型的大小的变化直接影响到堆的变化

3. 堆栈数据结构上的区别：

   堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

   栈（数据结构）：一种先进后出的数据结构

12、#include<file.h>和#include"file.h"的区别

前者是从标准库路径寻找，后者是从当前工作路径开始寻找

13、什么是内存泄漏？面对内存泄漏和指针越界，你有哪些方法？

动态分配内存所开辟的空间，在使用完毕后未手动释放，导致一直占据该内存，即为内存泄漏。

方法：malloc/free要配套，对指针赋值的时候应该注意被赋值的指针是否需要释放；使用的时候记得指针的长度，防止越界。

14、定义和声明的区别

为变量分配地址和存储空间的称为定义，不分配地址的称为声明。

一个变量可以在多个地方声明，但是只在一个地方定义。

加入extern修饰的是变量的声明，说明此变量将在文件以外火灾文件后面部分定义。

说明:很多时候一个变量，知识声明不分配内存空间，直到具体使用时才初始化，分配内存空间，如外部变量

15、C++文件编译与执行的四个阶段

1. 预处理：根据文件中的预处理指令来修改源文件中的内容
2. 编译：编译成汇编代码
3. 汇编：把汇编代码翻译成目标及其指令
4. 链接：链接目标代码生成可执行程序

16、C++的内存管理

在C++中，内存被分成五个区：栈、堆、自由存储区、静态存储区、常量区

1. 栈：存放函数的参数和局部变量，编译器自动分配和释放
2. 堆：new关键字动态分配的内存，由程序员手动进行释放，否则程序结束后，由操作系统自动进行回收
3. 自由存储区：new所申请的内存则是在自由存储区上，使用delete来释放
4. 堆：堆是操作系统维护的一块内存，而自由存储是C++中通过new与delete动态分配和释放对象的抽象概念，堆与自由存储区并不等价
5. 全局/静态存储区：存放全局变量和静态变量
6. 常量区：存放常量，不允许被修改

17、C++的四种强制转换

类型转化机制可以分为隐式类型转换和显示类型转化（强制类型转换）

`（new-type）expression new-type（expression）`

隐式类型转换比较常见，在混合类型表达式中经常发生；四种强制类型转换操作符：`static_cast` `dynamic_cast` `const_cast` `reinterpret_cast`

1. `static_cast`：编译时期的静态类型检查static_cast静态转换相当于C语言中的强制转换，但不能实现普通指针数据（空指针除外）的强制转换，一般用于父类和子类指针、引用间的相互转换。没有运行时类型检查来保证转换的安全性

   ①用于类层次结构中基类（父类）和派生类（子类）之间指针或引用的转换。不管是否发生多态，父子之间互转时，编译器都不会报错。

   - 进行上行转换（把派生类的指针或引用转换成基类表示）是安全的；
   - 进行下行转换（把基类指针或引用转换成派生类表示）时，由于没有动态类型检查，所以是不安全的，但是编译器不会报错。

   ②用于基本数据类型之间的转换，如把 int 转换成 char，把 int 转换成 enum。这种转换的安全性也要开发人员来保证。 ③把空指针转换成目标类型的空指针。 ④把任何指针类型转换成空指针类型。 ⑤可以对普通数据的 const 和 non_const 进行转换，但不能对普通数据取地址后的指针进行 const 添加和消去。 ⑥无继承关系的自定义类型，不可转换，不支持类间交叉转换。 另外，static_cast 不能转换掉原有类型的 const、volatile、或者 __unaligned 属性。(前两种可以使用 const_cast 来去除)。

2. `dynamic_cast`：运行时的检查 动态转换的类型和操作数必须是完整类类型或空指针、空引用，说人话就是说：只能用于类间转换，支持类间交叉转换，不能操作普通数据。

主要用于类层次结构中基类（父类）和派生类（子类）之间指针或引用的转换， ①进行上行转换（把派生类的指针或引用转换成基类表示）是安全的，允许转换； ②进行下行转换（把基类指针或引用转换成派生类表示）时，由于没有动态类型检查，所以是不安全的，不允许转化，编译器会报错； ③发生多态时，允许互相转换。④无继承关系的类之间也可以相互转换，类之间的交叉转换。 ⑤如果 dynamic_cast 语句的转换目标是指针类型并且失败了，则结果为 0。如果转换目标是引用类型并且失败了，则 dynamic_cast 运算符将抛出一个 std::bad_cast 异常

3. `const_cast`：const_cast用于强制去掉不能被修改的常数特性，其去除常量性的对象一般为指针或引用
4. `reinterpret_cast`：在C++语言中，`reinterpret_cast`主要有几种强制转换用途：将指针或引用转换为一个足够长度的整型、将整型转换为指针或引用类型。
5. 为什么不使用C的强制转换？ C的强制转换表面上看起来功能强大什么都能转，但是转化不够明确，不能进行错误检查，容易出错

18、extern"C"作用

extern"C"的主要作用就是为了能够正确实现C++代码调用其他C语言代码。加上extern"C"之后，会指示编译器这部分代码按C语言的进行编译，而不是C++的。

19、typedef和define区别

#define 是预处理命令，在预处理是执行简单的替换，不做正确性的检查 typedef 是在编译时处理的，它是在自己的作用域内给已经存在的类型一个别名 typedef (int*) pINT; #define pINT2 int* 效果相同？实则不同！实践中见差别：pINT a,b; 的效果同 int *a; int *b; 表示定义了两个整型指针变量。而 pINT2 a,b; 的效果同 int *a, b; 表示定义了一个整型指针变量 a 和整型变量 b。

20、引用作为函数参数以及返回值的好处

对比值传递，引用传参的好处：

1. 在函数内部可以对此参数进行修改
2. 提高函数调用和运行的效率（所以没有了传值和生成副本的时间和空间消耗） **值传递：** 形参是实参的拷贝，改变形参的值并不会影响外部实参的值。从被调用函数的角度来说，值传递是单向的（实参 -> 形参），参数的值只能传入，不能传出。当函数内部需要修改参数，并且不希望这个改变影响调用者时，采用值传递。 **指针传递：** 形参为指向实参地址的指针，当对形参的指向操作时，就相当于对实参本身进行的操作 **引用传递：** 形参相当于是实参的 “别名”，对形参的操作其实就是对实参的操作，在引用传递过程中，被调函数的形式参数虽然也作为局部变量在栈中开辟了内存空间，但是这时存放的是由主调函数放进来的实参变量的地址。被调函数对形参的任何操作都被处理成间接寻址，即通过栈中存放的地址访问主调函数中的实参变量。正因为如此，被调函数对形参做的任何操作都影响了主调函数中的实参变量。 用引用作为返回值最大的好处就是在内存中不产生被返回值的副本。

但是有以下的限制：

1. 不能返回局部变量的引用。因为函数返回以后局部变量就会被销毁
2. 不能返回函数内部 new 分配的内存的引用。虽然不存在局部变量的被动销毁问题，可对于这种情况（返回函数内部 new 分配内存的引用），又面临其它尴尬局面。例如，被函数返回的引用只是作为一 个临时变量出现，而没有被赋予一个实际的变量，那么这个引用所指向的空间（由 new 分配）就无法释放，造成 memory leak
3. 可以返回类成员的引用，但是最好是 const。因为如果其他对象可以获得该属性的非常量的引用，那么对该属性的单纯赋值就会破坏业务规则的完整性。