20180405_c++常见面试题30道

原文链接：https://blog.csdn.net/fakine/article/details/51321544

1.new、delete、malloc、free关系

delete会调用对象的析构函数,和new对应free只会释放内存，new调用构造函数。malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都可用于申请动态内存和释放内存。对于非内部数据类型的对象而言，光用maloc/free无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以及一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

2.delete与 delete []区别

delete只会调用一次析构函数，而delete[]会调用每一个成员的析构函数。在More Effective C++中有更为详细的解释："当delete操作符用于数组时，它为每个数组元素调用析构函数，然后调用operator delete来释放内存。"delete与new配套，delete []与new []配套

MemTest *mTest1=new MemTest[10];

MemTest *mTest2=new MemTest;

Int *pInt1=new int [10];

Int *pInt2=new int;

delete[]pInt1; //-1-

delete[]pInt2; //-2-

delete[]mTest1;//-3-

delete[]mTest2;//-4-

在-4-处报错。

这就说明：对于内建简单数据类型，delete和delete[]功能是相同的。对于自定义的复杂数据类型，delete和delete[]不能互用。delete[]删除一个数组，delete删除一个指针。简单来说，用new分配的内存用delete删除；用new[]分配的内存用delete[]删除。delete[]会调用数组元素的析构函数。内部数据类型没有析构函数，所以问题不大。如果你在用delete时没用括号，delete就会认为指向的是单个对象，否则，它就会认为指向的是一个数组。

3.C++有哪些性质（面向对象特点）

封装，继承和多态。

4.子类析构时要调用父类的析构函数吗？

析构函数调用的次序是先派生类的析构后基类的析构，也就是说在基类的的析构调用的时候,派生类的信息已经全部销毁了。定义一个对象时先调用基类的构造函数、然后调用派生类的构造函数；析构的时候恰好相反：先调用派生类的析构函数、然后调用基类的析构函数。

5.多态，虚函数，纯虚函数

多态：是对于不同对象接收相同消息时产生不同的动作。C++的多态性具体体现在运行和编译两个方面：在程序运行时的多态性通过继承和虚函数来体现；

在程序编译时多态性体现在函数和运算符的重载上；

虚函数：在基类中冠以关键字 virtual 的成员函数。 它提供了一种接口界面。允许在派生类中对基类的虚函数重新定义。

纯虚函数的作用：在基类中为其派生类保留一个函数的名字，以便派生类根据需要对它进行定义。作为接口而存在纯虚函数不具备函数的功能，一般不能直接被调用。

从基类继承来的纯虚函数，在派生类中仍是虚函数。如果一个类中至少有一个纯虚函数，那么这个类被称为抽象类（ abstract class ）。

抽象类中不仅包括纯虚函数，也可包括虚函数。抽象类必须用作派生其他类的基类，而不能用于直接创建对象实例。但仍可使用指向抽象类的指针支持运行时多态性。

6.求下面函数的返回值（微软）

```
int func(x)

{

int countx = 0;

while(x)

{

countx ++;

x = x&(x-1);

}

return countx;

}
```

假定x = 9999。 答案：8

思路：将x转化为2进制，看含有的1的个数。

7.什么是“引用”？申明和使用“引用”要注意哪些问题？

答：引用就是某个目标变量的“别名”(alias)，对应用的操作与对变量直接操作效果完全相同。申明一个引用的时候，切记要对其进行初始化。引用声明完毕后，相当于目标变量名有两个名称，即该目标原名称和引用名，不能再把该引用名作为其他变量名的别名。声明一个引用，不是新定义了一个变量，它只表示该引用名是目标变量名的一个别名，它本身不是一种数据类型，因此引用本身不占存储单元，系统也不给引用分配存储单元。不能建立数组的引用。

8.将“引用”作为函数参数有哪些特点？

（1）传递引用给函数与传递指针的效果是一样的。这时，被调函数的形参就成为原来主调函数中的实参变量或对象的一个别名来使用，所以在被调函数中对形参变量的操作就是对其相应的目标对象（在主调函数中）的操作。

（2）使用引用传递函数的参数，在内存中并没有产生实参的副本，它是直接对实参操作；而使用一般变量传递函数的参数，当发生函数调用时，需要给形参分配存储单元，形参变量是实参变量的副本；如果传递的是对象，还将调用拷贝构造函数。因此，当参数传递的数据较大时，用引用比用一般变量传递参数的效率和所占空间都好。

（3）使用指针作为函数的参数虽然也能达到与使用引用的效果，但是，在被调函数中同样要给形参分配存储单元，且需要重复使用"*指针变量名"的形式进行运算，这很容易产生错误且程序的阅读性较差；另一方面，在主调函数的调用点处，必须用变量的地址作为实参。而引用更容易使用，更清晰。

9.在什么时候需要使用“常引用”？

如果既要利用引用提高程序的效率，又要保护传递给函数的数据不在函数中被改变，就应使用常引用。常引用声明方式：const 类型标识符 &引用名=目标变量名；

例1

```
int a ;

const int &ra=a;

ra=1; //错误

a=1; //正确
```

例2

```
string foo( );

void bar(string & s);
```

那么下面的表达式将是非法的：

```
bar(foo( ));

bar("hello world");
```

原因在于foo( )和"hello world"串都会产生一个临时对象，而在C++中，这些临时对象都是const类型的。因此上面的表达式就是试图将一个const类型的对象转换为非const类型，这是非法的。引用型参数应该在能被定义为const的情况下，尽量定义为const 。

10.将“引用”作为函数返回值类型的格式、好处和需要遵守的规则?

格式：类型标识符 &函数名（形参列表及类型说明）{ //函数体 }

好处：在内存中不产生被返回值的副本；（注意：正是因为这点原因，所以返回一个局部变量的引用是不可取的。因为随着该局部变量生存期的结束，相应的引用也会失效，产生runtime error!

注意事项：

（1）不能返回局部变量的引用。这条可以参照Effective C++[1]的Item 31。主要原因是局部变量会在函数返回后被销毁，因此被返回的引用就成为了"无所指"的引用，程序会进入未知状态。

（2）不能返回函数内部new分配的内存的引用。这条可以参照Effective C++[1]的Item 31。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，但最好是const。这条原则可以参照Effective C++[1]的Item 30。主要原因是当对象的属性是与某种业务规则（business rule）相关联的时候，其赋值常常与某些其它属性或者对象的状态有关，因此有必要将赋值操作封装在一个业务规则当中。如果其它对象可以获得该属性的非常量引用（或指针），那么对该属性的单纯赋值就会破坏业务规则的完整性。

（4）流操作符重载返回值申明为“引用”的作用：

流操作符<<和>>，这两个操作符常常希望被连续使用，例如：cout << "hello" << endl; 因此这两个操作符的返回值应该是一个仍然支持这两个操作符的流引用。可选的其它方案包括：返回一个流对象和返回一个流对象指针。但是对于返回一个流对象，程序必须重新（拷贝）构造一个新的流对象，也就是说，连续的两个<<操作符实际上是针对不同对象的！这无法让人接受。对于返回一个流指针则不能连续使用<<操作符。因此，返回一个流对象引用是惟一选择。这个唯一选择很关键，它说明了引用的重要性以及无可替代性，也许这就是C++语言中引入引用这个概念的原因吧。

赋值操作符=。这个操作符象流操作符一样，是可以连续使用的，例如：x = j = 10;或者(x=10)=100;赋值操作符的返回值必须是一个左值，以便可以被继续赋值。因此引用成了这个操作符的惟一返回值选择。

```
# include

int &put(int n);

int vals[10];

int error=-1;

void main()

{

put(0)=10; //以put(0)函数值作为左值，等价于vals[0]=10;

put(9)=20; //以put(9)函数值作为左值，等价于vals[9]=20;

cout<<vals[0];

cout<<vals[9];

}

int &put(int n)

{

if (n>=0 && n<=9 ) return vals[n];

else { cout<<"subscript error"; return error; }
```

}

（5）在另外的一些操作符中，却千万不能返回引用：+-*/ 四则运算符。它们不能返回引用，Effective C++[1]的Item23详细的讨论了这个问题。主要原因是这四个操作符没有side effect，因此，它们必须构造一个对象作为返回值，可选的方案包括：返回一个对象、返回一个局部变量的引用，返回一个new分配的对象的引用、返回一个静态对象引用。根据前面提到的引用作为返回值的三个规则，第2、3两个方案都被否决了。静态对象的引用又因为((a+b) == (c+d))会永远为true而导致错误。所以可选的只剩下返回一个对象了。

11、结构与联合有和区别？

(1). 结构和联合都是由多个不同的数据类型成员组成, 但在任何同一时刻, 联合中只存放了一个被选中的成员（所有成员共用一块地址空间）, 而结构的所有成员都存在（不同成员的存放地址不同）。

(2). 对于联合的不同成员赋值, 将会对其它成员重写, 原来成员的值就不存在了, 而对于结构的不同成员赋值是互不影响的。

12、试写出程序结果：

int a=4;

int &f(int x)

{ a=a+x;

return a;

}

int main(void)

{ int t=5;

cout<<f(t)<<endl; a = 9

f(t)=20; a = 20

cout<<f(t)<<endl; t = 5,a = 20 a = 25

t=f(t); a = 30 t = 30

cout<<f(t)<<endl; } t = 60

}

13.重载（overload)和重写(overried，有的书也叫做“覆盖”）的区别？

常考的题目。从定义上来说：

重载：是指允许存在多个同名函数，而这些函数的参数表不同（或许参数个数不同，或许参数类型不同，或许两者都不同）。

重写：是指子类重新定义父类虚函数的方法。

从实现原理上来说：

重载：编译器根据函数不同的参数表，对同名函数的名称做修饰，然后这些同名函数就成了不同的函数（至少对于编译器来说是这样的）。如，有两个同名函数：function func(p:integer):integer;和function func(p:string):integer;。那么编译器做过修饰后的函数名称可能是这样的：int_func、str_func。对于这两个函数的调用，在编译器间就已经确定了，是静态的。也就是说，它们的地址在编译期就绑定了（早绑定），因此，重载和多态无关！

重写：和多态真正相关。当子类重新定义了父类的虚函数后，父类指针根据赋给它的不同的子类指针，动态的调用属于子类的该函数，这样的函数调用在编译期间是无法确定的（调用的子类的虚函数的地址无法给出）。因此，这样的函数地址是在运行期绑定的（晚绑定）。

14.有哪几种情况只能用intialization list 而不能用assignment?

答案：当类中含有const、reference 成员变量；基类的构造函数都需要初始化表。

15. C++是不是类型安全的？

答案：不是。两个不同类型的指针之间可以强制转换（用reinterpret cast)。C#是类型安全的。

16. main 函数执行以前，还会执行什么代码？

答案：全局对象的构造函数会在main 函数之前执行。

17. 描述内存分配方式以及它们的区别?

1）从静态存储区域分配。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量，static 变量。

2）在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集。

3）从堆上分配，亦称动态内存分配。程序在运行的时候用malloc 或new 申请任意多少的内存，程序员自己负责在何时用free 或delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活，但问题也最多。

18.分别写出BOOL,int,float,指针类型的变量a与“零”的比较语句。

答案：

BOOL : if (!a) or if(a)

int : if (a == 0)

float : const EXPRESSION EXP = 0.000001

if (a < EXP && a >-EXP)

pointer : if (a != NULL) or if(a == NULL)

19.请说出const与#define 相比，有何优点？

答案：

const作用：定义常量、修饰函数参数、修饰函数返回值三个作用。被Const修饰的东西都受到强制保护，可以预防意外的变动，能提高程序的健壮性。

1 ）const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误。

2 ）有些集成化的调试工具可以对const 常量进行调试，但是不能对宏常量进行调试。

20.简述数组与指针的区别？

数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。指针可以随时指向任意类型的内存块。

(1)修改内容上的差别

char a[] = “hello”;

a[0] = ‘X’;

char *p = “world”; // 注意p 指向常量字符串

p[0] = ‘X’; // 编译器不能发现该错误，运行时错误

(2) 用运算符sizeof 可以计算出数组的容量（字节数）。sizeof(p),p 为指针得到的是一个指针变量的字节数，而不是p 所指的内存容量。C++/C 语言没有办法知道指针所指的内存容量，除非在申请内存时记住它。注意当数组作为函数的参数进行传递时，该数组自动退化为同类型的指针。

char a[] = "hello world";

char *p = a;

cout<< sizeof(a) << endl; // 12 字节

cout<< sizeof(p) << endl; // 4 字节

计算数组和指针的内存容量

void Func(char a[100])

{

cout<< sizeof(a) << endl; // 4 字节而不是100 字节

}

第21题： int (*s[10])(int) 表示的是什么？

int (*s[10])(int) 函数指针数组，每个指针指向一个int func(int param)的函数。

第22题：栈内存与文字常量区

char str1[] = "abc";　　char str2[] = "abc";

　　const char str3[] = "abc";　　const char str4[] = "abc";

　　const char *str5 = "abc";　　const char *str6 = "abc";

　　char *str7 = "abc";　　char *str8 = "abc";

　　cout << (str1 == str2) << endl;//0 分别指向各自的栈内存　　cout << (str3 == str4) << endl;//0 分别指向各自的栈内存　　cout << (str5 == str6) << endl;//1指向文字常量区地址相同

　　cout << (str7 == str8) << endl;//1指向文字常量区地址相同

　　结果是：0 0 1 1

　　解答：str1,str2,str3,str4是数组变量，它们有各自的内存空间；而str5,str6,str7,str8是指针，它们指向相同的常量区域。

第23题：将程序跳转到指定内存地址

要对绝对地址0x100000赋值，我们可以用(unsigned int*)0x100000 = 1234;那么要是想让程序跳转到绝对地址是0x100000去执行，应该怎么做？

　　((void ())*( ))0x100000) ( );*　　首先要将0x100000强制转换成函数指针,即:　　(void (*)())0x100000　　然后再调用它:　　*((void ()*)())0x100000)();*　　用typedef可以看得更直观些:　　typedef void(*)() voidFuncPtr;　　*((voidFuncPtr)0x100000)();

第24题：int id[sizeof(unsigned long)];这个对吗？为什么？

答案:正确 这个 sizeof是编译时运算符，编译时就确定了,可以看成和机器有关的常量。

第25题：引用与指针有什么区别？

【参考答案】
1) 引用必须被初始化，指针不必。

2) 引用初始化以后不能被改变，指针可以改变所指的对象。

3) 不存在指向空值的引用，但是存在指向空值的指针。

第26题：const 与 #define 的比较 ，const有什么优点?

【参考答案】

（1）const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误（边际效应）。

（2）有些集成化的调试工具可以对 const 常量进行调试，但是不能对宏常量进行调试。

第27题：复杂声明

void * (* (*fp1)(int))[10];

float ((fp2)(int,int,int))(int);

int (* (* fp3)())10;

分别表示什么意思？【标准答案】

1.void * (* (*fp1)(int))[10]; *fp1是一个指针，指向一个函数，这个函数的参数为int型，函数的返回值是一个指针，这个指针指向一个数组，这个数组有10个元素，每个元素是一个void型指针。*

2.float ((fp2)(int,int,int))(int); fp2是一个指针，指向一个函数，这个函数的参数为3个int型，函数的返回值是一个指针，这个指针指向一个函数，这个函数的参数为int型，函数的返回值是float型。 3.int (* (* fp3)())10; fp3是一个指针，指向一个函数，这个函数的参数为空，函数的返回值是一个指针，这个指针指向一个数组，这个数组有10个元素，每个元素是一个指针，指向一个函数，这个函数的参数为空，函数的返回值是int型。

第28题：内存的分配方式有几种?

【参考答案】

一、从静态存储区域分配。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量。

二、在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。

三、从堆上分配，亦称动态内存分配。程序在运行的时候用malloc或new申请任意多少的内存，程序员自己负责在何时用free或delete释放内存。动态内存的生存期由我们决定，使用非常灵活，但问题也最多。

第29题：基类的析构函数不是虚函数，会带来什么问题？

【参考答案】派生类的析构函数用不上，会造成资源的泄漏。

第30题：全局变量和局部变量有什么区别？是怎么实现的？操作系统和编译器是怎么知道的？

【参考答案】

生命周期不同：

全局变量随主程序创建和创建，随主程序销毁而销毁；局部变量在局部函数内部，甚至局部循环体等内部存在，退出就不存在；

使用方式不同：通过声明后全局变量程序的各个部分都可以用到；局部变量只能在局部使用；分配在栈区。

操作系统和编译器通过内存分配的位置来知道的，全局变量分配在全局数据段并且在程序开始运行的时候被加载。局部变量则分配在堆栈里面 。