# C 笔试题目及答案

C 语言描述问题比汇编语言迅速，工作量小、可读性好，易于调试、修改和移植，而代码质量与汇编语言相当。C 语言一般只比汇编程序生成的目标代码效率低 10%～20%。下面就由店铺为大家介绍一下 C 笔试题目汇总的文章，欢迎阅读。

C 笔试题目汇总篇 1

1.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。

2. 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。

3. 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。

4. 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;

constint&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 。

5. 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：

(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(这个要注意啦，很多人没意识到，哈哈。。。)。 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。

例 3

```
#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。

6. “引用”与多态的关系?

引用是除指针外另一个可以产生多态效果的手段。这意味着，一个基类的引用可以指向它的派生类实例(见：C++中类的多态与虚函数的使用)。

例 4

Class A;

Class B : Class A

{

// ...

};

B b;

A&ref= b;

7. “引用”与指针的区别是什么?

指针通过某个指针变量指向一个对象后，对它所指向的变量间接操作。程序中使用指针，程序的可读性差;

而引用本身就是目标变量的别名，对引用的操作就是对目标变量的操作。此外，就是上面提到的对函数传 ref 和 pointer 的区别。

8. 什么时候需要“引用”?

流操作符<<和>>、赋值操作符=的返回值、拷贝构造函数的参数、赋值操作符=的参数、其它情况都推荐使用引用。

9. 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。

10. 下面关于“联合”的题目的输出?

a)

```
#include

union

{

int i;

char x[2];

}a;

void main()

{

a.x[0] =10;

a.x[1] =1;
```

```
printf("%d",a.i);

}
```

答案：266 (低位低地址，高位高地址，内存占用情况是 Ox010A)

b)

```
main()

{

union{ /*定义一个联合*/

int i;

struct{ /*在联合中定义一个结构*/

char first;

char second;

}half;

}number;
```

```
number.i=0x4241; /*联合成员赋值*/

printf("%c%c\n", number.half.first, mumber.half.second);

number.half.first='a'; /* 联合中结构成员赋值*/

number.half.second='b';

printf("%x\n",number.i);

getch();

}
```

答案：AB (0x41 对应'A',是低位;Ox42 对应'B',是高位)

6261 (number.i 和 number.half 共用一块地址空间)

C 笔试题目汇总篇 2

1. 已知 strcpy 的函数原型：char *strcpy(char *strDest, const char *strSrc)其中 strDest 是目的字符串，strSrc 是源字符串。不调用 C++/C 的字符串库函数，请编写函数strcpy 。

答案：

/*

编写 strcpy 函数(10 分)

已知 strcpy 函数的原型是

char *strcpy(char *strDest, const char *strSrc);

其中 strDest 是目的字符串，strSrc 是源字符串。

(1)不调用 C++/C 的字符串库函数，请编写函数strcpy

(2)strcpy 能把 strSrc 的内容复制到 strDest，为什么还要 char * 类型的返回值?

答：为了 实现链式表达式。// 2 分

例如 int length = strlen( strcpy( strDest, “hello world”) );

*/

#include

#include

char*strcpy(char*strDest, constchar*strSrc)

```
{
assert((strDest!=NULL) && (strSrc !=NULL)); // 2 分
char* address = strDest;        // 2 分
while( (*strDest++=*strSrc++) !='\0' )       // 2 分
NULL;
return address ;            // 2 分
}
```

另外 strlen 函数如下：

```
#include
#include
int strlen( constchar*str ) // 输入参数 const
{
assert( str != NULL ); // 断言字符串地址非 0
```

```
int len = 0;

while( (*str++) !='\0' )

{

len++;

}

return len;

}
```

2. 已知 String 类定义如下：

```
class String

{

public:

String(const char *str = NULL); // 通用构造函数

String(const String &another); // 拷贝构造函数
```

~String(); // 析构函数

String& operater =(const String &rhs); // 赋值函数

private:

char* m_data; // 用于保存字符串

};

尝试写出类的成员函数实现。

答案：

String::String(constchar*str)

{

if ( str == NULL ) // strlen 在参数为 NULL 时会抛异常才会有这步判断

{

m_data =newchar[1] ;

m_data[0] ='\0' ;

```
}

else

{

m_data =newchar[strlen(str) +1];

strcpy(m_data,str);

}

}

String::String(const String &another)

{

m_data =newchar[strlen(another.m_data) +1];

strcpy(m_data,other.m_data);

}

String& String::operator=(const String &rhs)
```