C++常见面试题

一、常见字符串

1. 写出在母串中查找子串出现次数的代码。

```
int count(char* str,char* substr)
{
  char* str1;
  char* str2;
int num = 0;
  while(*str!='\0')
  {
     str1 = str;
     str2 = s;
     while(*str2 == *str1&&(*str2!='\0')&&(*str1!='0'))
     {
        str2++;
        str1++;
      }
        if(*str2 == '\0')
        num++;
        str++;
  }
  return num;
}
```

2. 查找第一个匹配子串位置,如果返回的是 str1 长度 len1 表示没有找到。

```
size_t find(char* str1,char* str2)
{
   size_t i=0;
   size_t len1 = strlen(str1)
   size_t len2 = strlen(str2);
```

```
    if(len1-len2<0) return len1;
    for(;i<len1-len2;i++)
    {
        size_t m = i;
size_t j=0
        for(;size_t j=0;j<len2;j++)
        {
            if(str1[m]!=str2[j])
                break;
            m++;
        }
    if(j==len)
        break;
    }
return i<len1-len2?i:len1;
}
```

3. 实现 strcpy 函数。

```
char *strcpy(char *dst, const char *src)
{
assert(dst!=NULL&&src!=NULL);
char* target = dst;
while(*dst++=*src++);
return target ;
}
```

4. 实现字符串翻转。

```
 void reserve(char* str)
{
assert(str != NULL);
    char * p1 = str;
    char * p2 = str-1;
    while(*++p2);          //一般要求不能使用 strlen
    p2 -= 1;
    while(p1<p2)
    {
```

```
            char c = *p1;
            *p1++ = *p2;
            *p2-- = c;
        }
    }
```

5. 实现 strcmp 函数。

```
int strcmp11(char* src,char* dst)
{
    assert(src!=0&&r!=0);
    while(*src == *dst &&*src != '\0')
{
src++,
dst++;
}
    if(*src > *dst)
        return 1;
    else if(*src == *dst)
        return 0;
    return -1;
}
```

6. 用指针的方法，将字符串“ABCD1234efgh”前后对调显示。

```
//不要用 strlen 求字符串长度，这样就没分了
        char str123[] = "ABCD1234efgh";
        char * p1 = str123;
        char * p2 = str123-1;
        while(*++p2);
        p2 -= 1;
        while(p1<p2)
        {
            char c = *p1;
            *p1++ = *p2;
            *p2-- = c;
        }
```

7. 给定字符串 A 和 B,输出 A 和 B 中的最大公共子串。比如 A="aocdfe" B="pmcdfa" 则输出"cdf"。

```
#i nclude<stdio.h>
#i nclude<stdlib.h>
#i nclude<string.h>
char *commanstring(char shortstring[], char longstring[])
{
    int i, j;
    char *substring=malloc(256);
    if(strstr(longstring, shortstring)!=NULL)                    //如果……，那么返回
        return shortstring;   ;
    for(i=strlen(shortstring)-1;i>0; i--)                        //否则，开始循环计算
    {
        for(j=0; j<=strlen(shortstring)-i; j++)
        {
            memcpy(substring, &shortstring[j], i);
            substring[i]='\0';
            if(strstr(longstring, substring)!=NULL)
            return substring;
        }
    }
    return NULL;
}
int main()
{
    char *str1=malloc(256);
    char *str2=malloc(256);
    char *comman=NULL;
    gets(str1);
    gets(str2);

    if(strlen(str1)>strlen(str2))                                //将短的字符串放前面
        comman=commanstring(str2, str1);
    else
        comman=commanstring(str1, str2);
```

```
        printf("the longest comman string is: %s\n", comman);
    }
```

8. 判断一个字符串是不是回文。

```
    int IsReverseStr(char *str)
    {
        int i,j;
        int found=1;
        if(str==NULL)
            return -1;
        char* p = str-1;
        while(*++p!= '\0');
        --p;
        while(*str==*p&&str<p) str++,p--;
        if(str < p)
            found = 0;
        return found;
    }
```

9. 写函数完成内存的拷贝。

```
    void* memcpy( void *dst, const void *src, unsigned int len )
    {
        register char *d;
        register char *s;
        if (len == 0)
            return dst;
        if ( dst > src )    //考虑覆盖情况
        {
            d = (char *)dst + len - 1;
            s = (char *)src + len - 1;
            while ( len >= 4 )    //循环展开，提高执行效率
            {
                *d-- = *s--;
                *d-- = *s--;
                *d-- = *s--;
```

```
                *d-- = *s--;
                len -= 4;
            }
            while ( len-- )
            {
                *d-- = *s--;
            }
        }
        else if ( dst < src )
        {
            d = (char *)dst;
            s = (char *)src;
            while ( len >= 4 )
            {
                *d++ = *s++;
                *d++ = *s++;
                *d++ = *s++;
                *d++ = *s++;
                len -= 4;
            }
            while ( len-- )
            {
                *d++ = *s++;
            }
        }
        return dst;
    }
```

10. 写一个函数,它的原形是 int continumax(char *outputstr,char *intputstr)。

功能：

在字符串中找出连续最长的数字串，并把这个串的长度返回，并把这个最长数字串付给其中一个函数参数 outputstr 所指内存。例如："abcd12345ed125ss123456789"的首地址传给 intputstr 后，函数将返回 9，outputstr 所指的值为 123456789

```
    int continumax(char *outputstr, char *inputstr)
    {
        char *in = inputstr, *out = outputstr, *temp, *final;
        int count = 0, maxlen = 0;
```

```
        while( *in != '\0' )
        {
            if( *in > 47 && *in < 58 )
            {
                for(temp = in; *in > 47 && *in < 58 ; in++ )
                count++;
            }
        else
        in++;

        if( maxlen < count )
        {
            maxlen = count;
            count = 0;
            final = temp;
        }
        }
        for(int i = 0; i < maxlen; i++)
        {
            *out = *final;
            out++;
            final++;
        }
        *out = '\0';
        return maxlen;
    }
```

二、常识题

1. 请说出 static 和 const 关键字的作用，至少说出两种。

static 关键字至少有下列 5 个作用：

⑴ 函数体内 static 变量的作用范围为该函数体，不同于 auto 变量，该变量的内存只被分配一次，因此其值在下次调用时仍维持上次的值；

⑵ 在模块内的 static 全局变量可以被模块内所用函数访问，但不能被模块外其它函数

访问；

⑶ 在模块内的 static 函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明它的模块内；

⑷ 在类中的 static 成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；

⑸ 在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的 static 成员变量。

const 关键字至少有下列 5 个作用：

⑴ 欲阻止一个变量被改变，可以使用 const 关键字。在定义该 const 变量时，通常需要对它进行初始化，因为以后就没有机会再去改变它了；

⑵ 对指针来说，可以指定指针本身为 const，也可以指定指针所指的数据为 const，或二者同时指定为 const；

⑶ 在一个函数声明中，const 可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值；

⑷ 对于类的成员函数，若指定其为 const 类型，则表明其是一个常函数，不能修改类的成员变量；

⑸ 对于类的成员函数，有时候必须指定其返回值为 const 类型，以使得其返回值不为“左值”。例如：

```
const classA operator*(const classA& a1,const classA& a2);
```

operator*的返回结果必须是一个 const 对象。如果不是，这样的变态代码也不会编译出错：

```
classA a, b, c;
(a * b) = c; // 对 a*b 的结果赋值操作(a * b) = c 显然不符合编程者的初衷，也没有任//何意义
```

2. 如何判断是 C 还是 C++编译器？

```
#ifdefine _CPLUSPLUC
cout<<”C++”;
#else
cout <<”C””;
```

3. 如何避免重复包含一个头文件？

```
#ifndef _HEADER_
#define _HEADER_
……
#endif
```

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。

5. 引用与指针有什么区别？

⑴ 引用必须被初始化，指针不必。
⑵ 引用初始化以后不能被改变，指针可以改变所指的对象。
⑶ 不存在指向空值的引用，但是存在指向空值的指针。
⑷ 重载操作符使用引用可以完成串试操作。

6. 全局变量和局部变量在内存中是否有区别？如果有，是什么区别？

全局变量储存在全局静态存储区，局部变量在堆栈。

7. 什么是平衡二叉树？

左右子树都是平衡二叉树 且左右子树的深度差值的绝对值不大于 1。

8. 什么函数不能声明为虚函数？

构造函数。

9. 冒泡排序算法和快速排序算法的时间复杂度分别是什么？

冒泡排序：O(n^2)。
快速排序 o(nlgn)。

10. 进程间通信的方式有哪些?

进程间通信的方式有：共享内存，管道，Socket，消息队列，DDE 等。

11. 纯虚函数如何定义？使用时应注意什么？

```
virtual void f()=0;
```

纯虚函数是接口，子类必须要实现。

12. c 和 c++中的 struct 有什么不同？

c 和 c++中 struct 的主要区别是 c 中的 struct 不可以含有成员函数，而 c++中的 struct 可以。c++中 struct 和 class 的主要区别在于默认的存取权限不同，struct 默认为 public，而 class 默认为 private。

13. const 符号常量：

```
(1) const char *p
(2) char const *p
(3) char * const p
```

说明上面三种描述的区别？

如果 const 位于星号的左侧，则 const 就是用来修饰指针所指向的变量，即指针指为常量；如果 const 位于星号的右侧，const 就是修饰指针本身，即指针本身是常量。

14. 数组和链表的区别？

数组：数据顺序存储，固定大小。

连表：数据可以随机存储，大小可动态改变。

15. 线程与进程的区别和联系？线程是否具有相同的堆栈？dll 是否有独立的堆栈？

进程是死的，只是一些资源的集合，真正的程序执行都是线程来完成的，程序启动的时候操作系统就帮你创建了一个主线程。每个线程有自己的堆栈。

DLL 中有没有独立的堆栈，这个问题不好回答，或者说这个问题本身是否有问题。因为 DLL 中的代码是被某些线程所执 行，只有线程拥有堆栈，如果 DLL 中的代码是 EXE 中的线程所调用，那么这个时候是不是说这个 DLL 没有自己独立的堆栈？

如果 DLL 中的代码是由 DLL 自 己创建的线程所执行，那么是不是说 DLL 有独立的堆栈？

以上讲的是堆栈，如果对于堆来说，每个 DLL 有自己的堆，所以如果是从 DLL 中动态分配的内存，最好是从 DLL 中删除，如果你从 DLL 中分配内存，然后在 EXE 中，或者另外一个 DLL 中删除，很有可能导致程序崩溃。

16. 是不是一个父类写了一个 virtual 函数，如果子类覆盖它的函数不加 virtual ,也能实现多态？

virtual 修饰符会被隐形继承的。private 也被集成，只事派生类没有访问权限而已。virtual 可加可不加。子类的空间里有父类的所有变量(static 除外)。同一个函数只存在一个

实体(inline 除外)。子类覆盖它的函数不加 virtual ,也能实现多态。在子类的空间里，有父类的私有变量。私有变量不能直接访问。

17. 请简单描述 Windows 内存管理的方法。

内存管理有块式管理，页式管理，段式和段页式管理。现在常用段页式管理

块式管理：把主存分为一大块、一大块的，当所需的程序片断不在主存时就分配一块主存空间，把程 序片断 load 入主存，就算所需的程序片度只有几个字节也只能把这一块分配给它。这样会造成很大的浪费，平均浪费了 50%的内存空间，但时易于管理。

页式管理：把主存分为一页一页的，每一页的空间要比一块一块的空间小很多，显然这种方法的空间利用率要比块式管理高很多。

段式管理：把主存分为一段一段的，每一段的空间又要比一页一页的空间小很多，这种方法在空间利用率上又比页式管理高很多，但是也有另外一个缺点。一个程序片断可能会被分为几十段，这样很多时间就会被浪费在计算每一段的物理地址上（计算机最耗时间的大家都知道是 I/O 吧）。

段页式管理：结合了段式管理和页式管理的优点。把主存分为若干页，每一页又分为若干段。

18. 下面两种 if 语句判断方式。请问哪种写法更好？为什么？

```
int n;
if (n == 10) // 第一种判断方式
if (10 == n) // 第二种判断方式
```

第二种好，因为如果少了个=号,编译时就会报错,减少了出错的可能行,可以检测出是否少了‘=‘。

19. C 和 C++有什么不同?

从机制上：c 是面向过程的（但 c 也可以编写面向对象的程序)；c++是面向对象的，提供了类。但是， c++编写面向对象的程序比 c 容易 从适用的方向：c 适合要求代码体积小的，效率高的场合，如嵌入式；c++适合更上层的，复杂的； llinux 核心大部分是 c 写的，因为它是系统软件，效率要求极高。 从名称上也可以看出，c++比 c 多了+，说明 c++是 c 的超集；那为什么不叫 c+而叫 c++呢，是因为 c++比 c 来说扩充的东西太多了，所以就在 c 后面放上两个+；于是就成了 c++ C 语言是结构化编程语言，C++是面向对象编程语言。C++侧重于对象而不是过程，侧重于类的设计而不是逻辑的设计。

20. 请从下述答案中选择一个正确的答案。

```
class A
 {
  virtual void func1();
  void func2();
}
Class B: class A
{
  void func1(){cout < < "fun1 in class B" < < endl;}
  virtual void func2(){cout < < "fun2 in class B" < < endl;}
}
```

A, A 中的 func1 和 B 中的 func2 都是虚函数.
B, A 中的 func1 和 B 中的 func2 都不是虚函数.
C, A 中的 func2 是虚函数.，B 中的 func1 不是虚函数.
D, A 中的 func2 不是虚函数，B 中的 func1 是虚函数.
答：A

三、编程题

1. 写出下述程序的运行结果。

```
int sum(int a)
{
   auto int c=0;
   static int b=3;
   c+=1;
   b+=2;
   return(a+b+c);
}
int main()
{
   int I;
   int a=2;
    for(I=0;I<5;I++)
    {
```

```
            printf("%d,", sum(a));
        }
    }
    // static 会保存上次结果，记住这一点，剩下的自己写
```

输出：8,10,12,14,16,

2. 求 1000！的未尾有几个 0?

（用素数相乘的方法来做，如 72=2*2*2*3*3）;求出 1->1000 里,能被 5 整除的数的个数 n1,能被 25 整除的数的个数 n2,能被 125 整除的数的个数 n3,能被 625 整除的数的个数 n4.1000!末尾的零的个数=n1+n2+n3+n4;

```
#include<stdio.h>
#define NUM 1000
int find5(int num)
{
    int ret=0;
while(num%5==0)
    {
        num/=5;
        ret++;
    }
    return ret;
}
int main()
{
    int result=0;
    int i;
    for(i=5;i<=NUM;i+=5)
    {
        result+=find5(i);
    }
    printf(" the total zero number is %d\n",result);
    return 0;
}
```

3. 编程实现：把十进制数(long 型)分别以二进制和十六进制形式输出，不能使用 printf 系列库函数。

```
char* test3(long num)
{
    char* buffer = (char*)malloc(11);
    buffer[0] = '0';
buffer[1] = 'x';
    buffer[10] = '\0';
    char* temp = buffer + 2;
    for (int i=0; i < 8; i++)
    {
        temp[i] = (char)(num<<4*i>>28);
        temp[i] = temp[i] >= 0 ? temp[i] : temp[i] + 16;
        temp[i] = temp[i] < 10 ? temp[i] + 48 : temp[i] + 55;
    }
return buffer;
}
```

4. 将一个数字字符串转换为数字."1234" -->1234。

```
int atoii(char* s)
{
    assert(s!=NULL);
    int num = 0;
    int temp;
    while(*s>'0' && *s<'9')
    {
        num *= 10;
        num += *s-'0';
      s++;
    }
     return num;
 }
```

5. 实现任意长度的整数相加或者相乘功能。

```
void bigadd(char* num,char* str,int len)
```

```
{
for(int i=len;i>0;i--)
    {
        num[i] += str[i];
          int j = i;
        while(num[j]>=10)
        {
            num[j--] -= 10;
            num[j] += 1;
        }
    }
}
```

6. 不用递归实现斐波拉契数。

int Funct(int n) // n 为非负整数

```
{
    int a=1;
int b=1;
    int c;
    if(n==0 || n == 1)
        return 1;
    for(int i=1;i<n;i++)
    {
     c=a+b;
     a=b;
  b=c;
    }
    return b;
}
```

7. 用递归算法判断数组 a[N]是否为一个递增数组。

递归的方法，记录当前最大的，并且判断当前的是否比这个还大，大则继续，否则返回 false 结束：

```
bool fun( int a[], int n )
{
```

```
if( n= =1 )
        return true;
    if( n= =2 )
        return a[n-1] >= a[n-2];
    return fun( a,n-1) && ( a[n-1] >= a[n-2] );
}
```

8. 写一个函数找出一个整数数组中，第二大的数 。

```
const int MINNUMBER = -32767 ;
int find_sec_max( int data[] , int count)
{
   int maxnumber = data[0] ;
   int sec_max = MINNUMBER ;
  for ( int i = 1 ; i < count ; i++)
   {
      if ( data[i] > maxnumber )
     {
        sec_max = maxnumber ;
        maxnumber = data[i] ;
      }
      else
      {
        if ( data[i] > sec_max )
           sec_max = data[i] ;
      }

 return sec_max ;
```

9. 已知两个链表 head1 和 head2 各自有序，请把它们合并成一个链表依然有序，这次要求用递归方法进行。

```
Node * MergeRecursive(Node *head1 , Node *head2)
{
   i ( head1 == NULL )
   return head2 ;
   if head2 == NULL)
```

```
    return head1 ;
    Node *head = NULL ;
   if head1->data < head2->data )
   head = head1 ;
    head->next = MergeRecursive(head1->next,head2);
  else
     head = head2 ;
     head->next = MergeRecursive(head1,head2->next);
}
return head ;
}
```

0. 文件中有一组整数，要求排序后输出到另一个文件中。

```
#include<iostream>
#include<fstream>
using namespace std;
void Order(vector<int>& data) //bubble sort
{
   int count = data.size() ;
   int tag = false ; //  设置是否需要继续冒泡的标志位
   for ( int i = 0 ; i < count ; i++)
   {
     for ( int j = 0 ; j < count - i - 1 ; j++)
       {
          if ( data[j] > data[j+1])
            tag = true ;
int temp = data[j] ;
data[j] = data[j+1] ;
data[j+1] = temp ;
}
}
if ( !tag )
break ;
}
}

int main( void )
```

```
{
vector<int>data;
ifstream in("c:\\data.txt");
if ( !in)
{
cout<<"file error!";
exit(1);
}
int temp;
while (!in.eof())
{
in>>temp;
data.push_back(temp);
}
in.close(); //关闭输入文件流
Order(data);
ofstream out("c:\\result.txt");
if ( !out)
{
cout<<"file error!";
exit(1);
}
for ( i = 0 ; i < data.size() ; i++)
out<<data[i]<<" ";
out.close(); //关闭输出文件流
}
```

11. 输入一个字符串，将其逆序后输出。（使用 C++，不建议用伪码）

```
#include <iostream>
using namespace std;
void main()
 {
  char a[50];memset(a,0,sizeof(a));
  int i=0,j;
  char t;
  cin.getline(a,50,'\n');
```

```
  for(i=0,j=strlen(a)-1;i <strlen(a)/2;i++,j--)
   {
  t=a[i];
       a[i]=a[j];
  a[j]=t;
  }
  cout < <a < <endl;
 }
```

12. 下面的代码有什么问题？

```
 void DoSomeThing(...)
{
char* p;
...
 p = malloc(1024);  // 分配 1K 的空间
if (NULL == p)
  return;
...
p = realloc(p, 2048); // 空间不够，重新分配到 2K
if (NULL == p)
  return;
 ...
}
```

答案 p = malloc(1024); 应该写成： p = (char *) malloc(1024); 没有释放 p 的空间，造成内存泄漏。

13. 下面的代码有什么问题？并请给出正确的写法。

```
 void DoSomeThing(char* p)
{
char str[16];
int n;
assert(NULL != p);
sscanf(p, "%s%d", str, n);
if (0 == strcmp(str, "something"))
{
```

```
    ...
  }
  }
```

答案：sscanf(p, "%s%d", str, n); 这句改为： sscanf(p, "%s%d", str, &n);

14. 写出运行结果:

```
 {// test2
    union V {
struct X {
  unsigned char s1:2;
  unsigned char s2:3;
  unsigned char s3:3;
} x;
unsigned char c;
    } v;
    v.c = 100;
    printf("%d", v.x.s3);
}
```

运行结果：3

15. 用 C++写个程序，如何判断一个操作系统是 16 位还是 32 位的？不能用 sizeof()函数 。

16 位的系统下，

```
 int i = 65536;
cout < < i; // 输出 0；
int i = 65535;
cout < < i; // 输出-1；
```

32 位的系统下，

```
 int i = 65536;
cout < < i; // 输出 65536；
int i = 65535;
cout < < i; // 输出 65535；
```

16. 试编写函数判断计算机的字节存储顺序是开序(little endian)还是降序(bigendian) 。

```
 bool IsBigendian()
```

```
 {
unsigned short usData = 0x1122;
unsigned char   *pucData = (unsigned char*)&usData;
return (*pucData == 0x22);
}
```

17. 简述 Critical Section 和 Mutex 的不同点。

答:

⑴ Critical Section

A.速度快

B.不能用于不同进程

C.不能进行资源统计(每次只可以有一个线程对共享资源进行存取)

⑵ Mutex

A.速度慢

B.可用于不同进程

C.不能进行资源统计

⑶ Semaphore

A.速度慢

B.可用于不同进程

C.可进行资源统计(可以让一个或超过一个线程对共享资源进行存取)

⑷ Event

A.速度慢

B.可用于不同进程

C.可进行资源统计

18. 用 C 写一个输入的整数,倒着输出整数的函数,要求用递归方法 。

```
 void fun( int a )
{
printf( "%d", a%10 );
a /= 10;
if( a <=0 )return;
fun( a );
}
```

19. 写出程序结果:

```
void Func(char str[100])
{
  printf("%d\n", sizeof(str));
}
```

答: 4

20. 请问运行 Test 函数会有什么样的结果?

```
void GetMemory(char *p)
{
  p = (char *)malloc(100);
}
void Test(void)
{
  char *str = NULL;
  GetMemory(str);
  strcpy(str, "hello world");
  printf(str);
}
```

答：程序崩溃。

因为 GetMemory 并不能传递动态内存，Test 函数中的 str 一直都是 NULL。strcpy(str, "hello world");将使程序崩溃