1. 用预处理指令#define 声明一个常数，用以表明1年中有多少秒（忽略闰年问题）

```
#define SECONDS_PER_YEAR (60 * 60 * 24 * 365)
```

2. 写一个“标准”宏MIN，这个宏输入两个参数并返回较小的一个。

```
#define MIN(A,B) ((A) <= (B) ? (A) : (B))
```

3. 嵌入式系统中经常要用到无限循环，你怎么样用C编写死循环呢？

这个问题用几个解决方案。我首选的方案是：

```
while(1)
{

}
```

一些程序员更喜欢如下方案：

```
for(;1;)
{

}
```

第三个方案是用 goto

```
Loop:
...
goto Loop;
```

4. 用变量a给出下面的定义

a) 一个整型数（An integer）
b) 一个指向整型数的指针（A pointer to an integer）
c) 一个指向指针的的指针，它指向的指针是指向一个整型数（A pointer to a pointer to an integer）
d) 一个有10个整型数的数组（An array of 10 integers）
e) 一个有10个指针的数组，该指针是指向一个整型数的（An array of 10 pointers to integers）
f) 一个指向有10个整型数数组的指针（A pointer to an array of 10 integers）
g) 一个指向函数的指针，该函数有一个整型参数并返回一个整型数（A pointer to a function that takes an integer as an argument and returns an integer）
h) 一个有10个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数（ An array of ten pointers to functions that take an integer argument and return an integer ）

```
a) int a; // An integer
b) int *a; // A pointer to an integer
```

c) int **a; // A pointer to a pointer to an integer
d) int a[10]; // An array of 10 integers
e) int *a[10]; // An array of 10 pointers to integers
f) int (*a)[10]; // A pointer to an array of 10 integers
g) int (*a)(int); // A pointer to a function a that takes an integer argument and returns an integer
h) int (*a[10])(int); // An array of 10 pointers to functions that take an integer argument and return an integer

5. 关键字static的作用是什么？

（1）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；
（2）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；
（3）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；

6. 关键字const是什么含意？ 分别解释下列语句中const的作用？

```
const int a;
int const a;
const int *a;
int * const a;
int const * const a;
```

（1）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；

（2）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；

（3）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；

（4）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；

前两个的作用是一样，a是一个常整型数。第三个意味着a是一个指向常整型数的指针（也就是，整型数是不可修改的，但指针可以）。第四个意思a是一个指向整型数的常指针（也就是说，指针指向的整型数是可以修改的，但指针是不可修改的）。最后一个意味着a是一个指向常整型数的常指针（也就是说，指针指向的整型数是不可修改的，同时指针也是不可修改的）。如果应试者能正确回答这些问题，那么他就给我留下了一个好印象。

7. 关键字volatile有什么含意 并给出三个不同的例子。

一个定义为volatile的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值了。精确地说就是，优化器在用到这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。下面是volatile变量的几个例子：
1). 并行设备的硬件寄存器（如：状态寄存器）
2). 一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)（中断嵌套）
3). 多线程应用中被几个任务共享的变量

8. 一个参数既可以是const还可以是volatile吗？解释为什么。

可以，硬件状态寄存器

9. 一个指针可以是volatile 吗？解释为什么。

是的。尽管这并不很常见。一个例子是当一个中服务子程序修该一个指向一个buffer的指针时。

10. 下面的函数有什么错误：

```
int square(volatile int *ptr)
{
    return *ptr * *ptr;
}
```

这段代码有点变态。这段代码的目的是用来返指针*ptr指向值的平方，但是，由于*ptr指向一个volatile型参数，编译器将产生类似下面的代码：

```
int square(volatile int *ptr)
{
    int a,b;
    a = *ptr;
    b = *ptr;
    return a * b;
}
```

由于*ptr的值可能被意想不到地该变，因此a和b可能是不同的。结果，这段代码可能返不是你所期望的平方值！正确的代码如下：

```
long square(volatile int *ptr)
{
    int a;
    a = *ptr;
    return a * a;
```

```
}
```

11. 嵌入式系统总是要用户对变量或寄存器进行位操作。给定一个整型变量a，写两段代码，第一个设置a的bit 3，第二个清除a 的bit 3。在以上两个操作中，要保持其它位不变。

```
#define BIT3 (0x1<<3)
static int a;

void set_bit3(void)
{
    a |= BIT3;
}

void clear_bit3(void)
{
    a &= ~BIT3;
}
```

12. 嵌入式系统经常具有要求程序员去访问某特定的内存位置的特点。在某工程中，要求设置一绝对地址为0x67a9的整型变量的值为0xaa66。编译器是一个纯粹的ANSI编译器。写代码去完成这一任务。

这一问题测试你是否知道为了访问一绝对地址把一个整型数强制转换（typecast）为一指针是合法的。这一问题的实现方式随着个人风格不同而不同。典型的类似代码如下：

```
int *ptr;
ptr = (int *)0x67a9;
*ptr = 0xaa55;
```

14 . 下面的代码输出是什么，为什么？

```
void foo(void)
{
unsigned int a = 6;
int b = -20;
(a+b > 6) puts("> 6") : puts("<= 6");
}
```

答案是输出是 ”>6”。

原因是当表达式中存在有符号类型和无符号类型时所有的操作数都自动转换为无符号类型。因此-20变成了一个非常大的正整数，所以该表达式计算出的结果大于6。这一点对于应当频繁用到无符号数据类型的嵌入式系统来说是丰常重要的。

15. 评价下面的代码片断：

```
unsigned int zero = 0;
unsigned int compzero = 0xFFFF;
/*1's complement of zero */
```

对于一个int型不是16位的处理器为说，上面的代码是不正确的。应编写如下：

```
unsigned int compzero = ~0;
```

这一问题真正能揭露出应试者是否懂得处理器字长的重要性。好的嵌入式程序员非常准确地明白硬件的细节和它的局限。

16. 尽管不像非嵌入式计算机那么常见，嵌入式系统还是有从堆（heap）中动态分配内存的过程的。那么嵌入式系统中，动态分配内存可能发生的问题是什么？

主要有三种类型:内存泄露、内存碎片和内存崩溃　内存崩溃是内存使用最严重的结果，主要原因有数组访问越界、写已经释放的内存、指针计算错误、访问堆栈地址越界等等。碎片收集的问题，变量的持行时间等等

17.下面的代码片段的输出是什么，为什么？

```
char *ptr;
if ((ptr = (char *)malloc(0)) == NULL)
puts("Got a null pointer");
else
puts("Got a valid pointer");
```

该代码的输出是“Got a valid pointer”。

18. Typedef 在C语言中频繁用以声明一个已经存在的数据类型的同义字。也可以用预处理器做类似的事。例如，思考一下下面的例子,那个更好，为什么？

```
#define dPS struct s *
typedef struct s * tPS;
```

答案是：typedef更好。思考下面的例子：

```
dPS p1,p2;
tPS p3,p4;
```

第一个扩展为

```
struct s * p1, p2;
```

上面的代码定义p1为一个指向结构的指，p2为一个实际的结构，这也许不是你想要的。第二个例子正确地定义了p3 和p4 两个指针。

19. C语言同意一些令人震惊的结构,下面的结构是合法的吗，如果是它做些什么？

```
int a = 5, b = 7, c;
```

```
c = a+++b;
```

上面的代码被处理成：

```
c = a++ + b;
```

因此，这段代码持行后a = 6， b = 7， c = 12。

20.找错题

试题1：

```
void test1()
{
  char string[10];
  char* str1 = "0123456789";
  strcpy( string, str1 );
}
```

试题2：

```
void test2()
{
  char string[10], str1[10];
  int i;
  for(i=0; i<10; i++)
  {
    str1[i] = 'a';
  }
  strcpy( string, str1 );
}
```

试题3：

```
void test3(char* str1)
{
  char string[10];
  if( strlen( str1 ) <= 10 )
  {
    strcpy( string, str1 );
  }
}
```

解答：

试题1字符串str1需要11个字节才能存放下（包括末尾的'\0'），而string

只有10个字节的空间，strcpy会导致数组越界；

对试题2，如果面试者指出字符数组str1不能在数组内结束可以给3分；如果面试者指出strcpy(string, str1)调用使得从str1内存起复制到string内存起所复制的字节数具有不确定性可以给7分，在此基础上指出库函数strcpy工作方式的给10分；

对试题3，if(strlen(str1) <= 10)应改为if(strlen(str1) < 10)，因为strlen的结果未统计’\0’所占用的1个字节。

21．写出字符串strcpy的函数实现过程式

```
void strcpy( char *strdest, char *strsrc )
{
   while( (*strdest++ = * strsrc++) != ‘\0’ );
}
```

4分

```
void strcpy( char *strdest, const char *strsrc )
//将源字符串加const，表明其为输入参数，加2分
{
   while( (*strdest++ = * strsrc++) != ‘\0’ );
}
```

7分

```
void strcpy(char *strdest, const char *strsrc)
{
  //对源地址和目的地址加非0断言，加3分
  if((strdest == null) || (strsrc == null));
    printf(“……”);
return;
  while( (*strdest++ = * strsrc++) != ‘\0’ );
}
```

10分

```
//为了实现链式操作，将目的地址返回，加3分！

char * strcpy( char *strdest, const char *strsrc )
{
  if((strdest == null) && (strsrc == null));
       return NULL;
  char *address = strdest;
```

```
  while( (*strdest++ = * strsrc++) != '\0' );
    return address;
}
```

22. 经典getmemory问题讨论

试题1：

```
void getmemory( char *p )
{
  p = (char *) malloc( 100 );
}

void test( void )
{
  char *str = null;
  getmemory( str );
  strcpy( str, "hello world" );
  printf( str );
}
```

试题2：

```
char *getmemory( void )
{
  char p[] = "hello world";
  return p;
}

void test( void )
{
  char *str = null;
  str = getmemory();
  printf( str );
}
```

试题3：

```
void getmemory( char **p, int num )
{
  *p = (char *) malloc( num );
  If (*p == NULL) {    }
}

void test( void )
{
  char *str = null;
```

```
  getmemory( &str, 100 );
  strcpy( str, "hello" );
  printf( str );
}
```

试题4：

```
void test( void )
{
  char *str = (char *) malloc( 100 );
  strcpy( str, "hello" );
  free( str );
  ... //省略的其它语句
}
```

解答：

试题1传入中getmemory(char *p)函数的形参为字符串指针，在函数内部修改形参并不能真正的改变传入形参的值，执行完

```
char *str = null;
getmemory( str );
```

后的str仍然为null；

试题2中

```
char p[] = "hello world";
return p;
```

的p[]数组为函数内的局部自动变量，在函数返回后，内存已经被释放。这是许多程序员常犯的错误，其根源在于不理解变量的生存期。

试题3的getmemory避免了试题4的问题，传入getmemory的参数为字符串指针的指针，但是在getmemory中执行申请内存及赋值语句

```
*p = (char *) malloc( num );
```

后未判断内存是否申请成功，应加上：

```
if ( *p == null )
{
  ...//进行申请内存失败处理
}
```

试题4存在与试题3同样的问题，在执行

```
char *str = (char *) malloc(100);
```

后未进行内存是否申请成功的判断；另外，在free(str)后未置str为空，导致可能变成一个“野”指针，应加上：

```
str = null;
```

试题3的test函数中也未对malloc的内存进行释放。

23. 下面的一段程序有什么错误：

```
swap( int* p1,int* p2 )
{
    If ((p1 == NULL) || (p2 == NULL)) {

    }

  int b;
  int *p = &b;
  p = &b;

  *p = *p1;
  *p1 = *p2;
  *p2 = *p;
 }
```

在swap函数中，p是一个“野”指针，有可能指向系统区，导致程序运行的崩溃。在vc++中debug运行时提示错误“access violation”。该程序应该改为：

```
swap( int* p1,int* p2 )
{
  int p;
  p = *p1;
  *p1 = *p2;
  *p2 = p;
}
```

24. 分别给出bool，int，float，指针变量 与“零值”比较的 if 语句（假设变量名为var）

解答：

bool型变量：if(!var)

int型变量： if(0==var)

float型变量：

const float epsinon = 0.00001;

if ((x >= - epsinon) && (x <= epsinon)

指针变量： if(var==null)

剖析：

考查对0值判断的“内功”，bool型变量的0判断完全可以写成if(var==0)，而int型变量也可以写成if(!var)，指针变量的判断也可以写成if(!var)，上述写法虽然程序都能正确运行，但是未能清晰地表达程序的意思。

一般的，如果想让if判断一个变量的“真”、“假”，应直接使用if(var)、if(!var)，表明其为“逻辑”判断；如果用if判断一个数值型变量(short、int、long等)，应该用if(var==0)，表明是与0进行“数值”上的比较；而判断指针则适宜用if(var==null)，这是一种很好的编程习惯。

浮点型变量并不精确，所以不可将float变量用“==”或“！=”与数字比较，应该设法转化成“>=”或“<=”形式。如果写成if (x == 0.0)，则判为错，得0分。

25. 请计算sizeof的值

```
void func ( char *str )
{
  sizeof( str ) = ?
}

void *p = malloc( 100 );
sizeof ( p ) = ?

char str[10] = “hello”;
strlen(str);
cout << sizeof(str) << endl;
```

解答：

……

```
sizeof( str ) = 4
sizeof ( p ) = 4
sizeof(str) = 10
```

26. 写一个“标准”宏min，这个宏输入两个参数并返回较小的一个。另外，当你写下面的代码时会发生什么事？

```
least = min(*p++, b);
```

解答：

```
#define min(a,b) ((a) <= (b) ? (a) : (b))
    Int a = 10 , b = 20;
     Int *P = &a
  //  min(*p++, b)会产生宏的副作用
     ((*p++) <= (b) ? (*p++) : (b));
```

27. 为什么标准头文件都有类似以下的结构？

```
#ifndef __incvxworksh
#define __incvxworksh
#ifdef __cplusplus

extern "c" {
#endif
/*...*/
#ifdef __cplusplus
}

#endif
#endif /* __incvxworksh */
```

解答：

头文件中的编译宏

```
#ifndef  __incvxworksh
#define  __incvxworksh
#endif
```

的作用是防止被重复引用。
为了实现c和c++的混合编程，c++提供了c连接交换指定符号extern "c"来解决名字匹配问题，函数声明前加上extern "c"后，则编译器就会按照c语言的方式将该函数编译为_foo，这样c语言中就可以调用c++的函数了。

28. 编写一个函数，作用是把一个char组成的字符串循环右移n个。比如原来是“abcdefghi”如果n=2，移位后应该是“hiabcdefgh”

函数头是这样的：

```
//pstr是指向以'\0'结尾的字符串的指针
//steps是要求移动的n

void loopmove ( char * pstr, int steps )
{
  //请填充...
}
```

解答：

正确解答1：

```
void loopmove ( char *pstr, int steps )
{
  int n = strlen( pstr ) - steps;
  char tmp[max_len];
  strcpy ( tmp, pstr + n ); //hi
  strcpy ( tmp + steps, pstr); //hiabcdefghi
  *( tmp + strlen ( pstr ) ) = '\0';//hiabcedfg\0i
  strcpy( pstr, tmp ); //hiabcedfg\0
}
```

正确解答2：

```
void loopmove ( char *pstr, int steps )
{
  int n = strlen( pstr ) - steps;
  char tmp[max_len];
  memncpy( tmp, pstr + n, steps ); //hi
  memncpy(pstr + steps, pstr, n ); //hiabcdefg\0
  memncpy(pstr, tmp, steps );
}
```

29. 请写一个c函数，若处理器是big_endian的，则返回0；若是little_endian的，则返回1

解答：

```
int checkcpu()
{
    union w
    {
      int a;
      char b;
       short c;
    } c;
    c.a = 1;
    return (c.b == 1);
}
```

30. 堆和栈的区别?

栈区（stack）- 由编译器自动分配释放 ，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。

堆区（heap） - 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 。

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。

31. struct 和 class 的区别

答案:struct 的成员默认是公有的,而类的成员默认是私有的。struct 和 class 在其他方面是功能相当的。

从感情上讲，大多数的开发者感到类和结构有很大的差别。感觉上结构仅仅象一堆缺乏封装和功能的开放的内存位，而类就象活的并且可靠的社会成员，它有智能服务，有牢固的封装屏障和一个良好定义的接口。既然大多数人都这么认为，那么只有在你的类有很少的方法并且有公有数据(这种事情在良好设计的系统中是存在的!)时，你也许应该使用 struct 关键字，否则，你应该使用 class 关键字。

32.

```
#include "stdafx.h"
#define SQR(X) X*X
```

```
int main(int argc, char* argv[])
{
    int a = 10;
    int k = 2;
    int m = 1;

    a /= SQR(k+m)/SQR(k+m);
    printf("%d\n",a);

    return 0;
}
```

这道题目的结果是什么啊?

宏替换：a /= 2+1*2+1/2+1*2+1；a = 10/7 = 1;

33.下面是 C 语言中两种 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，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=

34.写出运行结果:

```
{// test2
    union V {
       struct X {
           unsigned char s1:2;
           unsigned char s2:3;
           unsigned char s3:2;
       } x;

       unsigned char c;
    } v;

    v.c = 100;
    printf("%d", v.x.s3);

}
```

Unsigned char 类型 100 的二进制表示为:01100100 v.x.s3 为后三位，及 100，因此答案为 3.

35.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

A1:
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;
```

A2:

```
int a = ~0;
if( a==65535 )
{
    cout<<"16 bit"<<endl;
}
else
{
    cout<<"32 bit"<<endl;
}
```

36.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。

37.在不用第三方参数的情况下，交换两个参数的值

方法一：

```
#include <stdio.h>

void main()
{
        int i=65534;
        int j=65000;
        i=i+j;
        j=i-j;
        i=i-j;
        printf("i=%d\n",i);
        printf("j=%d\n",j);
}
```

方法二：

```
i^=j;  3 5
j^=i;
i^=j;
```

方法三：

```
// 用加减实现，而且不会溢出
a = a+b-(b=a)
```

38. 进程间通信的方式有?

进程间通信的方式有 共享内存， 管道 ， Socket ， 消息队列 ，等

39.

```
struct A
{
    char t:4;
    char k:4;
    unsigned short i:8;
    unsigned long m;
} __attribute__((packed));
```

sizeof(A)=?（不考虑边界对齐）

6

40. 给定结构

```
struct A
{
      char t:4;
```

```
        char k:4;
        unsigned short i:8;
        unsigned long m:12;
};
```

问sizeof(A) = ?

给定结构

```
struct A
{
0       char t:4; 4位
        char k:4; 4位

1       unsigned short i:8; 8位
2       unsigned long m; // 偏移2字节保证4字节对齐
};// 共8字节
```

41. 下面的函数实现在一个固定的数上加上一个数，有什么错误，改正

```
int add_n(int n)
{
  static int i=100;
  i+=n;
  return i;
}
```

答:

因为 static 使得 i 的值会保留上次的值。

去掉 static 就可了

42.

```
union a {
    int a_int1;
    double a_double;
    int a_int2;
};//8

typedef struct
{
    a a1;
    char y;
} b;//12

class c
{
    double c_double;
```

```
    b b1;
    a a2;

};//28
```

输出 cout<<sizeof(c)<<endl;的结果?

28

43. i 最后等于多少?

```
int i = 1;
int j = i++;//i = 2, j = 1
if((i>j++) && (i++ == j)) i+=j;
```

答:
i = 5

44. unsigned short array[]={1,2,3,4,5,6,7};

```
int i = 3;
*(array + i) = ?
```

答:
4

44.简述 Critical Section 和 Mutex 的不同点

答:
对几种同步对象的总结
1.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)

2.Mutex
A.速度慢
B.可用于不同进程
C.不能进行资源统计

3.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)

4.Event
A.速度慢

B.可用于不同进程
C.可进行资源统计

45. 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 ;

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

    fun( a );
}
```

46. #include <filename.h>和 #include “filename.h” 有什么区别?

答:前者用来包含开发环境提供的库头文件,后者用来包含自己编写的头文件。

47.在 C++ 程序中调用被 C 编译器编译后的函数,为什么要加 extern “C”声明?
答:函数和变量被 C++编译后在符号库中的名字与 C 语言的不同,被 extern "C"修饰的变
量和函数是按照 C 语言方式编译和连接的。由于编译后的名字不同,C++程序不能直接调
用 C 函数。C++提供了一个 C 连接交换指定符号 extern“C”来解决这个问题。
48. 回答下面的问题(6 分)
(1).
```
void GetMemory(char **p, int num)
{
    *p = (char *)malloc(num);
}
void Test(void)
{
    char *str = NULL;
    GetMemory(&str, 100);
    strcpy(str, "hello");
    printf(str);
}
```
请问运行 Test 函数会有什么样的结果?

答:输出“hello”

(2).
```
void Test(void)
```

```
{
    char *str = (char *) malloc(100);
    strcpy(str, “hello”);
    free(str);
    if(str != NULL){
    strcpy(str, “world”);
    printf(str);
}
```

请问运行 Test 函数会有什么样的结果?

答：输出“world”，向以 free 的空间中写数据，错误

(3).

```
char *GetMemory(void)
{
    char p[] = "hello world";
    return p;
}

void Test(void)
{
    char *str = NULL;
    str = GetMemory();
    printf(str);
}
```

请问运行 Test 函数会有什么样的结果?

答：无效的指针，输出不确定

48. 编写 strcat 函数

已知 strcat 函数的原型是 char *strcat (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 strcat

答：

```
char * strcat (char * dst, const char * src)
{
    if(dst == NULL) {
        return NULL;
    }

    while( *cp )
    cp++; /* find end of dst */
    while( *cp++ = *src++ ) ; /* Copy src to end of dst */
```

```
    return( dst ); /* return dst */
}
```

（2）strcat 能把 strSrc 的内容连接到 strDest，为什么还要 char＊ 类型的返回值？

答：方便赋值给其他变量

49.程序什么时候应该使用线程，什么时候单线程效率高。

答：1．耗时的操作使用线程，提高应用程序响应
2．并行操作时使用线程，如C/S架构的服务器端并发线程响应用户的请求。
3．多CPU系统中，使用线程提高CPU利用率
4．改善程序结构。一个既长又复杂的进程可以考虑分为多个线程，成为几个独立或半独
立的运行部分，这样的程序会利于理解和修改。
其他情况都使用单线程。

50.TCP/IP 建立连接的过程?(3-way shake)

答：在TCP/IP协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。
第一次握手：建立连接时，客户端发送syn包(syn=j)到服务器，并进入SYN_SEND状
态，等待服务器确认；
第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个
SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN_RECV状态；
第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)
，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

51.ICMP是什么协议,处于哪一层?

答：Internet控制报文协议，处于网络层（IP层）

52. winsock建立连接的主要实现步骤?

答：服务器端：socker()建立套接字，绑定（bind）并监听（listen），用accept（）等待客户端连接。
客户端：socker()建立套接字，连接（connect）服务器，连接上后使用send()和recv（），在套接字上写读数据，直至数据交换完毕，closesocket()关闭套接

字。
服务器端：accept（）发现有客户端连接，建立一个新的套接字，自身重新开始等待连接。该新产生的套接字使用send()和recv（）写读数据，直至数据交换完毕，closesocket()关闭套接字。

53. 动态连接库的两种方式?

答：调用一个DLL中的函数有两种方法：
1. 载入时动态链接（load-time dynamic linking），模块非常明确调用某个导出函数，使得他们就像本地函数一样。这需要链接时链接那些函数所在DLL的导入库，导入库向系统提供了载入DLL时所需的信息及DLL函数定位。
2. 运行时动态链接（run-time dynamic linking），运行时可以通过LoadLibrary或LoadLibraryEx函数载入DLL。DLL载入后，模块可以通过调用GetProcAddress获取DLL函数的出口地址，然后就可以通过返回的函数指针调用DLL函数了。如此即可避免导入库文件了

54. IP组播有那些好处?

答：Internet上产生的许多新的应用，特别是高带宽的多媒体应用，带来了带宽的急剧消耗和网络拥挤问题。组播是一种允许一个或多个发送者（组播源）发送单一的数据包到多个接收者（一次的，同时的）的网络技术。组播可以大大的节省网络带宽，因为无论有多少个目标地址，在整个网络的任何一条链路上只传送单一的数据包。所以说组播技术的核心就是针对如何节约网络资源的前提下保证服务质量。

55. 描述实时系统的基本特性

在特定时间内完成特定的任务，实时性与可靠性。

56. 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？

全局变量储存在静态数据库，局部变量在堆栈。

57. 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。

68. 堆栈溢出一般是由什么原因导致的？

没有回收垃圾资源。

59. 冒泡排序算法的时间复杂度是什么？

时间复杂度是O(n^2)。

60. Internet采用哪种网络协议？该协议的主要层次结构？

Tcp/Ip协议
主要层次结构为： 应用层/传输层/网络层/数据链路层/物理层。

61. Internet物理地址和IP地址转换采用什么协议？

ARP (Address Resolution Protocol)（地址解析協議）

62. IP地址的编码分为哪俩部分？

IP地址由两部分组成，网络号和主机号。不过是要和“子网掩码”按位与上之后才能区分哪些是网络位哪些是主机位。

63. 不能做switch()的参数类型是：

switch的参数不能为实型。
注：必须是整数型常量，包括char，short，int，long等，不能是浮点数。

```
Int main()
{
    Float a=3;
    Switch(a)
    {
    Case 3:
        Printf(“a”);
    }
    Return 0;
}
```

64. 局部变量能否和全局变量重名？

答：能，局部会屏蔽全局。要用全局变量，需要使用"::"
局部变量可以与全局变量同名，在函数内引用这个变量时，会用到同名的局部变量，而不会用到全局变量。对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，比如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内。

65. 如何引用一个已经定义过的全局变量？

答：extern
可以用引用头文件的方式，也可以用extern关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变理，假定你将那个变写错了，那么在编译期间会报错，如果你用extern方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错。

66. 全局变量可不可以定义在可被多个.C文件包含的头文件中？为什么？

答：可以，在不同的C文件中以static形式来声明同名全局变量。
可以在不同的C文件中声明同名的全局变量，前提是其中只能有一个C文件中对此变量赋初值，此时连接不会出错。

67. 语句for(; 1 ;)有什么问题？它是什么意思？

答：无限循环，和while(1)相同。

68. do……while和while……do有什么区别？

答：前一个循环一遍再判断，后一个判断以后再循环。

69. 程序的局部变量存在于（栈）中，全局变量存在于（静态区 ）中，动态申请数据存在于（堆）中。

70. 设有以下说明和定义：

```
typedef union {long i; int k[5]; char c;} DATE;//20
struct data { int cat; DATE cow; double dog;} too;//32
DATE max;
```

则语句 printf("%d",sizeof(struct date)+sizeof(max));的执行结果是：
___52____

71. 队列和栈有什么区别？

队列先进先出，栈后进先出

72. 请找出下面代码中的所以错误
说明：以下代码是把一个字符串倒序，如“abcd”倒序后变为“dcba”

```
#include"string.h"
main()
{
    Char *src="hello,world";
    Char *dest=NULL;
     int len=strlen(src);
     dest=(char*)malloc(len);
     char *d=dest;
     char *s=src[len];
     while(len--!=0)
    d++=s--;
     printf("%s",dest);
     return 0;
}
```

答：

```
int main()
```

```
{
    char* src = "hello,world";
    int len = strlen(src);
    char* dest = (char*)malloc(len+1);//要为\0分配一个空间
    char* d = dest;
    char* s = &src[len-1];//指向最后一个字符
    while( len-- != 0 )
    *d++=*s--;
    *d = 0;//尾部要加\0
    printf("%s\n",dest);
    free(dest);// 使用完，应当释放空间，以免造成内存汇泄露
    return 0;
}
```

73. 用两个栈实现一个队列的功能？要求给出算法和思路！

设2个栈为A,B, 一开始均为空.

入队:

将新元素push入栈A;

出队:

(1)判断栈B是否为空;
(2)如果不为空，则将栈A中所有元素依次pop出并push到栈B;
(3)将栈B的栈顶元素pop出;

74. 对于一个频繁使用的短小函数,在C语言中应用什么实现,在C++中应用什么实现?

c用宏定义，c++用inline

75. 软件测试都有那些种类?

黑盒：针对系统功能的测试　　白合：测试函数功能，各函数接口

76. 进程和线程的差别。

线程是指进程内的一个执行单元,也是进程内的可调度实体.
与进程的区别:
(1)调度：线程作为调度和分配的基本单位，进程作为拥有资源的基本单位
(2)并发性：不仅进程之间可以并发执行，同一个进程的多个线程之间也可并发执行
(3)拥有资源：进程是拥有资源的一个独立单位，线程不拥有系统资源，但可以访问隶属于进程的资源.

(4)系统开销：在创建或撤消进程时，由于系统都要为之分配和回收资源，导致系统的开销明显大于创建或撤消线程时的开销。

77. 网络编程中设计并发服务器，使用多进程 与 多线程 ，请问有什么区别？

1，进程：子进程是父进程的复制品。子进程获得父进程数据空间、堆和栈的复制品。
2，线程：相对与进程而言，线程是一个更加接近与执行体的概念，它可以与同进程的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。
两者都可以提高程序的并发度，提高程序运行效率和响应时间。
线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源管理和保护；而进程正相反。同时，线程适合于在SMP机器上运行，而进程则可以跨机器迁移。

78.测试方法

人工测试：个人复查、抽查和会审
机器测试：黑盒测试和白盒测试

79. Heap与stack的差别。

Heap是堆，stack是栈。
Stack的空间由操作系统自动分配/释放，Heap上的空间手动分配/释放。
Stack空间有限，Heap是很大的自由存储区
C中的malloc函数分配的内存空间即在堆上,C++中对应的是new操作符。
程序在编译期对变量和函数分配内存都在栈上进行,且程序运行过程中函数调用时参数的传递也在栈上进行

80. 一语句实现x是否为2的若干次幂的判断

```
int i = 512;
 ((i & (i - 1)) ? false : true);
```

81.下列程序的输出结果：

```
 main()
{
   int a[5]={1,2,3,4,5};
   int *ptr=(int *)(&a+1);// (&a+1)指向下一个一维数组
   printf("%d,%d",*(a+1),*(ptr-1));
}
```

输出：2,5

82.

```
char* s="AAA";
printf("%s",s);
s[0]='B';
printf("%s",s);
```

有什么错？

"AAA"是字符串常量。s是指针，指向这个字符串常量，所以声明s的时候就有问题。

```
cosnt char* s="AAA";
```

然后又因为是常量，所以对是s[0]的赋值操作是不合法的。

83. 列举几种进程的同步机制，并比较其优缺点。

原子操作
信号量机制
自旋锁
管程，会合，分布式系统

84.进程死锁的原因

资源竞争及进程推进顺序非法

85.死锁的4个必要条件

互斥、请求保持、不可剥夺、环路

86.死锁的处理

鸵鸟策略、预防策略、避免策略、检测与解除死锁

87. 操作系统中进程调度策略有哪几种？

FCFS(先来先服务)，优先级，时间片轮转，多级反馈

88.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

89.ISO的七层模型是什么？tcp/udp是属于哪一层？tcp/udp有何优缺点？

应用层
表示层
会话层
运输层

网络层
物理链路层
物理层
tcp /udp属于运输层
TCP 服务提供了数据流传输、可靠性、有效流控制、全双工操作和多路复用技术等。
与 TCP 不同， UDP 并不提供对 IP 协议的可靠机制、流控制以及错误恢复功能等。由于 UDP 比较简单， UDP 头包含很少的字节，比 TCP 负载消耗少。
tcp: 提供稳定的传输服务，有流量控制，缺点是包头大，冗余性不好
udp: 不提供稳定的服务，包头小，开销小

90. (void *)ptr 和 (*(void**))ptr的结果是否相同?

其中ptr为同一个指针(void *)ptr 和 (*(void**))ptr值是相同的

91. 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 ) ( );
Int add(int x, int y) //0x100

Int (*ptr)(int x, int y)

Ptr = add;
Ptr = &add;
Int z = add(10, 20);
```

92. 给出一下程序的输出结果：

```
typedef struct AA
{
        int b1:5;
        int b2:2;
}AA;

void main()
{
        AA aa;
        char cc[100];
        strcpy(cc,"0123456789abcdefghijklmnopqrstuvwxyz");
        memcpy(&aa,cc,sizeof(AA));
        cout << aa.b1 <<endl;
        cout << aa.b2 <<endl;
}
```

答案是 -16和 1
0的ASCII值00110000，因此，b1为10000，b2为001，扩展成32位再按照十进制输出。

93.分析：

```
struct bit
{  int a:3;
   int b:2;
   int c:3;
};

int main()
{
    bit s;
    char *c=(char*)&s;
    cout<<sizeof(bit)<<endl;
    *c=0x99;//10011001
    cout << s.a <<endl <<s.b<<endl<<s.c<<endl;
    int a=-1;
    printf("%x",a);
    return 0;
}
```

输出为什么是

```
4
1
-1
-4
ffffffff
```

94. 改错：

```
#include <stdio.h>
int main(void) {
    int (*p)[100];
   int arr[100] = {0, 1, 2, 3, 4, 5 … …};//arr = &arr[0] arr int *
    p = &arr; int **
    return 0;
}
```

解答：
搞错了,是指针类型不同,
int **p; //二级指针

&arr; //得到的是指向第一维为100的数组的指针

```
#i nclude <stdio.h>
int main(void) {
int **p, *q;
int arr[100];
q = arr;
p = &q;
return 0;
}
```

95.下面这个程序执行后会有什么错误或者效果:

```
 #define MAX 256
 int main()
{
    unsigned char A[MAX],i;//i被定义为unsigned char
    for (i=0;i<MAX;i++)
    A[i]=i;
}
```

解答：死循环加数组越界访问（C/C++不进行数组越界检查）