第一题

static全局变量与普通的全局变量有什么区别：

答：static**全局变量只初始化一次，防止在其他文件单元中被引用**;

static局部变量和普通局部变量有什么区别：

答：static**局部变量只被初始化一次，下一次依据上一次结果值**;

static函数与普通函数有什么区别：

答：static**函数在内存中只有一份，普通函数在每个被调用中维持一份拷贝。**

第二题

对于一个频繁使用的短小函数,在C语言中用宏实现,在C++中应用什么实现?

答：**C语言中使用宏，C++中使用内联函数**;

第三题

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;

答：int a=5,b=7,c; 合法，相当于int a=5;int b=7;int c;

```
c=a+++b;依然是合法的，编译器会把它变成    c=a++ +b;
```

编译器是从左向右结合的，在变量a之后，先扫描到++ 然后扫描到一个二目运算符+ 因此是c=a++ +b;

再来看下面的一个例子：

```
#include <stdio.h>
int main()
{
  int a = 5, b = 7, c;
  c = a+++++b;
  printf("a = %d,b = %d,c = %d",a,b,c);
  return 0;
}
```

那么这一次能成功吗?

很不幸，这一次直接报错了。

我们来分析一下：

> 对于a+++++b这一段代码，编译系统从左至右扫描整条语句，先遇到a++，判断出来是一个a的后缀自加运算；
>
> 然后接着扫描，遇到一个+，+是一个二目运算符，它的左边已经有一个运算数a++了，系统就向右搜索第二个运算数；
>
> 又遇到一个+，++比+的运算级别要高，这时，编译系统就将两个+看成一个整体来处理；
>
> 既然是++，编译系统就认定，肯定它的左边或右边有一个变量，编译系统先搜索左边，发现++，不是变量；
>
> 再搜索右边，发现+b，+b是什么东西？编译系统是无法搞明白的；
>
> 因此它就认为++是一个缺少左值的自增运算符，于是提示错误给用户：lvalue required as increment operand.

原理解析一下

> C语言在这里遵循词法解析的贪婪匹配原则。优先匹配尽可能多字符的符号，无论是否有语法错误（因为词法分析时还没有语法检查）。
>
> 于是a+++++b会被当作a ++ ++ + b，这是非法的表达式，因此产生编译错误。

第四题

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? 答案一: 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。 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(Symmetric Multi-Processing，对称多处理结构的简称，是指在一个计算机上汇集了一组处理器(多CPU),各CPU之间共享内存子系统以及总线结构。)机器上运行，而进程则可以跨机器迁移。

答案二:

根本区别就一点：用多进程每个进程有自己的地址空间(address space)，线程则共享地址空间。所有其它区别都是由此而来的： 1。速度：线程产生的速度快，线程间的通讯快、切换快等，因为他们在同一个地址空间内。 2。资源利用率：线程的资源利用率比较好也是因为他们在同一个地址空间内。 3。同步问题：线程使用公共变量/内存时需要使用同步机制还是因为他们在同一个地址空间内。

多线程的优势：

1. 创建与环境切换开销少

2. 同步的开销少（进程间同步一般牵涉OS内核）
3. 数据复制可通过“进程局部内存”

多线程的局限性：

1. 性能损失 （1）单处理器、“计算任务繁重”的应用程序不会从多线程中获益 （2）高精度的锁定策略会带来高同步开销
2. 健壮性降低 线程之间接收到的“MMU保护”很少或没有
3. 缺乏高精度的访问控制

第五题

全局变量和 局部变量有什么区别？是怎么实现的？操作系统和编译器怎么知道的？

> 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 。

第六题

有以下定义语句 double a,b; int w; long c;

若各变量已正确赋值，则下列选项中正确的表达式是_____。 A、a=a+b=b++ B、w%((int)a+b) C、(c+w)%(int)a D、w=a=b;

答：**C。**

> A选项，b++前面不可以是表达式，错误。
>
> B选项，w是整型，a转换成了int ,而括号内((int)a+b)仍旧是doublle 类型，而double类型的数据不能使用%运算符取余。错误。
>
> C选项正确。
>
> D选项，w是整型，a和b是double类型，赋值后，w的精度下降，错误。

第七题

程序的局部变量存在于（ ）中，全局变量存在于（ ）中，动态申请数据存在于（ ）中。

答：**栈区** ， **全局区** ， **堆区。**

第八题

语句for(；1；)有什么问题？它是什么意思？

答：**没有设置循环的终止条件，是无限循环，相当于while(1)**。

关于const指针

const指针是指针变量的值一经初始化，就不可以改变指向，初始化是必要的。其定义形式如下：type *const 指针名称;　　声明指针时，可以在类型前或后使用关键字const，也可在两个位置都使用。例如，下面都是合法的声明，但是含义大不同：

```
const int * pOne;    //指向整形常量 的指针，它指向的值不能修改
int * const pTwo;    //指向整形的常量指针 ，它不能在指向别的变量，但指向（变量）的值可以修改。
const int *const pThree;  //指向整形常量 的常量指针 。它既不能再指向别的常量，指向的值也不能修改。
```

理解这些声明的技巧在于，**查看关键字const右边来确定什么被声明为常量，如果该关键字的右边是类型，则值是常量；如果关键字的右边是指针变量，则指针本身是常量。**

下面的代码有助于说明这一点：

```
const int *p1;  //常量指针：即指向常量的指针
int * const p2; // 指针常量：指针本身是个常量
```

关于指针常量和常量指针的区别：

> 指针常量：指针本身是个常量（其实就是指针变量存储的地址不可改变），因此指针的指向不可修改，但指针所指向的值可以修改（对这个地址解引用的值可以改变）。

> 常量指针：指向常量的指针，就是字面上的意思，指针的指向可以修改，但是指针指向的值本身是个常量，因此不可以修改。

> const既修饰类型又修饰变量：指针的指向和指针所指的值都不可以修改。