- **全局变量**
  定义在函数外面的变量是全局变量，全局变量具有全局的生存期和作用域，它们与任何函数都无关，任何函数内部都可以使用它们

```
printf("%d",__func__,gall);
```

__func__的作用是输出函数名

- **全局变量初始化**
  没有做初始化的全局变量会得到0值，指针会得到NULL值，只能用编译时已知的值来初始化全局变量，它们的初始化发生在main函数之前

```
int gall=f(); //error,编译时要执行函数
int gall=12;
int gtwo=gall;//error,编译时要变量赋值

const int gall=12;
int gtwo=gall;//right;因其是常量
```

- **被隐藏的全局变量**
  如果函数内部存在与全局变量同名的变量，则全局变量被隐藏
- **静态本地变量**
  在本地变量定义时加上static修饰符，就成为静态本地变量
  当函数离开时，静态本地变量会继续存在，并保存其值，
  静态本地变量的初始化只会在第一次加入这个函数时做，以后进入函数时会保持上次离开时的值
  静态本地变量实际上是特殊的全局变量，它们位于相同的内存区域
  静态本地变量具有全局的生周期，函数内的局部作用域，
  static在这儿的意思是局部作用域，本地可访问
- **返回指针的函数**
  返回本地变量的指针是危险的
  返回全局变量或静态本地变量的地址是安全的
  返回在函数内malloc的内存是安全的，但容易造成问题
  最好的做法是返回传入的指针
- **tips**
  不要使用全局变量在函数间传递参数或结果
  尽量避免使用全局变量

使用全局变量和静态本地变量的函数是线程不安全的

- **编译预处理指令**
  #开头的是编译预处理指令
  它们不是c语言的成份，但c语言程序离不开它们
  #define用来定义一个宏

```
#define pi 3.1415926
#define format "%f\n"
```

名字必须是一个单词，值可以是各种东西
在c语言的编译器开始编译之前，编译预处理程序（cpp）会把程序中的名字替换成值，是完全的文本替换

```
gcc --save-temps //用来查编译过程
```

- **宏**
  如果一个宏的值中有另外一个宏的名字，也是会被替换的
  如果一个宏的值超过一行，最后一行之前的行末需要加\
  宏的值后面出现的注释不会被当作宏的值的一部分

```
#define pi 3.1415926
#define pi2 2*pi //pi*2
#define prt printf("%f",pi);\
            printf("%f\n",pi2);
```

- **没有值的宏**

```
#define _DEBUG
```

这类宏是用于条件编译的，后面有其他的编译预处理器指令来检查这个宏是否已经被定义过，如果存在我们编译这部分代码，如果不存在我们编译另外一部分代码

- **预先定义的宏**

```
__LINE__ 当前所在行的行号
__FILE__ 代码的文件名
```

```
__DATE__ 编译时的日期
__TIME__ 时间
__STDC__ 如果编译器符合c标准，那么返回1
```

- **定义像函数的宏**

```
#define cube(x) ((x)*(x)*(x))
#define radtodeg1(x) (x*5.204848)
#define radtodeg2(x) ((x)*5.204848)
#define min(a,b) ((a)>(b)?(b):(a))
```

宏可以带参数，但参数没有类型，注意区分加括号和没加
带参数的宏的原则：一切都要括号，整个值要括号，参数出现的每个地方都要括号
带参数的宏可以带多个参数，也可以组合嵌套使用其他宏
带参数的宏在大型程序的代码中使用十分普遍
可以非常复杂，如产生函数，在#和##这两个运算符的帮助下
存在东西方文化差异，中国程序员使用宏比较少
部分函数会被inline函数替代
宏的一个重大缺点是没有类型检查

```
#define TOUPPER(c) ('a'<=(c)&&(c)<='z'?(c)-'a'+'A':(c))
设s是一个足够大的字符数组，i是int型变量，则以下代码段的输出是：
strcpy(s, "abcd");
i = 0;
putchar(TOUPPER(s[++i]));
答案是D
```

- **多个.c文件**
  main里面的代码太长了，适合分成几个函数，一个源代码文件太长了，适合分成几个文件，两个独立的源代码文件不能编译形成可执行的程序
  新建一个项目，将源代码文件都加进去
  编译单元：一个.c文件是一个编译单元，而编译器每次编译只处理一个编译单元，编译后形成.o文件，然后由链接器将它们链接在一起，有的ide分开两个按钮，编译和构建
- **头文件**
  如果有两个.c文件，其中一个文件中没有函数的原型声明，编译器会默认是int型的，但在第二个文件中是double型的，当链接器将两个文件链接起来时，便会发生错误。

解决方法，头文件，把函数原型声明放在一个头文件里，在需要调用这个函数的源文件代码（.c文件）中#include这个头文件，就能让编译器在编译时知道函数的原型。

- **#include**
  #inlucde是一个编译预处理指令，和宏一样，在编译之前就处理了
  它把那个文件的全部文本内容原封不动地插入到它所在的地方
  所以也不是一定要在.c文件的最前面#include
- **""还是<>**
  #include有两种形式来指出要插入的文件
  ""要求编译器首先在当前目录（.c文件所在目录）寻找这个文件，如果没有，到编译器指定的目录去找。
  <>让编译器只在指定的目录去找
  编译器自己知道自己的标准库的头文件在哪里
  环境变量和编译器命令行参数也可以指定寻找头文件的目录
- **#include的误区**
  #include不是用来引入库的
  stdio.h里面只有printf的原型，printf的代码在另外的地方，某个.lib(windows)或者.a(unix)中
  现在的c语言编译器默认会引入所有的标准库
  #include<stdio.h>只是为了让编译器知道printf函数的原型，保证你调用时给出的参数值是正确的类型
- **头文件**
  在使用和定义这个函数的地方都应该#include这个头文件
  一般的做法就是任何.c都有对应的同名.h，把所有对外公开的函数的原型和全局变量的声明都放进去