链接
- 无链接 → 函数作用域、块作用域、函数原型作用域
- 外部链接 → 多文件（全局作用域/程序作用域）
- 内部链接 → 翻译单元（1个源代码+n个头文件）
（文件作用域）

```
int giants = 5;
static int dodgers = 3;
int main()
{
    ...
}
```

int giants = 5; → 外部链接

static int dodgers = 3; → 内部链接

static：存储类别说明符

存储期
- 静态 → 在程序的执行期间一直存在（内部链接/外部链接）
- 线程 → 被声明到线程结束一直存在，_Thread_local
- 自动 → 块作用域，程序进入分配内存，离开，释放内存（变长数组：从声明处到块末尾）
（注意，块作用域也可以使用 static 声明静态存储期）
- 动态分配 →

存储类别

存储类别	存储期	作用域	链接	声明方式
自动	自动	块	无	块内
寄存器	自动	块	无	块内，使用关键字register
静态外部链接	静态	文件	外部	所有函数外
静态内部链接	静态	文件	内部	所有函数外，使用关键字static
静态无链接	静态	块	无	块内，使用关键字static

存储类别说明符

```
int main(void)
{
    auto int plox;
```

1、自动变量：自动存储期、块作用域、无链接。可以显式使用auto。

自动变量不会初始化。

```
int main(void)
{
    register int quick;
```

2、寄存器变量：存储在CPU的寄存器中（最快的可用内存中），无法获取变量的地址

在函数头中使用 register。（数据类型有限，例如，可能没有足够大的空间来存储double）

```
void macho(register int n)
```

3、静态无链接变量：在内存中原地不动，具有块作用域，静态存储期，无链接

（局部静态变量）

```
void trystat(void)
{
    int fade = 1;
    static int stay = 1;
```

静态变量和外部变量，在程序被载入内存时已执行完毕。

4. 静态外部链接变量：具有文件作用域，外部链接，静态存储期（外部存储类别→外部变量）。和声明顺序有关系！

外部变量会被自动初始化为 0。
且只能使用常量表达式初始化。

```
int Hocus;
int magic();
int main(void)
{
  int Hocus; // 声明Hocus，默认是自动变量
     ...
}
int Pocus;
int magic()
{
     auto int Hocus; //把局部变量Hocus显式声明为自动变量
     ...
}
```

Pocus在main中不可见，因为声明在main后面

定义和声明的区别

```
int tern = 1; /* tern被定义 */
main()
{
    extern int tern; /* 使用在别处定义的tern */
```

→ 定义式声明

→ 引用式声明，

关键字 extern 表明，该声明不是定义，
它指示编译器去别处查询定义。（不要用 extern 创建外部定义）

外部变量只能初始化一次，且必须在定义该变量时进行，

```
// file_one.c
char permis = 'N';
...
// file_two.c
extern char permis = 'Y'; /* 错误 */
```

5. 静态内部链接变量：具有静态存储期，内部链接，文件作用域。

在所有函数外部，使用 static 定义

```
static int svil = 1;
int main(void)
{
```

→ 静态变量，内部链接，只能用于同一文件的函数

```
int traveler = 1;          // 外部链接
static int stayhome = 1;   // 内部链接
int main()
{
     extern int traveler;  // 使用定义在别处的 traveler
     extern int stayhome;  // 使用定义在别处的 stayhome
     ...
```

→ 其他翻译单元也可见

→ 只在该文件对应的翻译单元内可见

声明，不是定义！