

# 第7章 用函数实现模块化程序设计

**7.1**为什么要用函数

**7.2**怎样定义函数

**7.3**调用函数

**7.4**对被调用函数的声明和函数原型

**7.5**函数的多级嵌套调用

**7.6**递归函数设计

**7.7**数组作为函数参数

**7.8**局部变量和全局变量

**7.9**变量的存储类别和生存期

**7.10**函数说明符

**7.11** 内部函数和外部函数

## 7.9变量的存储类别和生存期

### 7.9.1 动态存储方式与静态存储方式

### 7.9.2 局部变量的存储类别

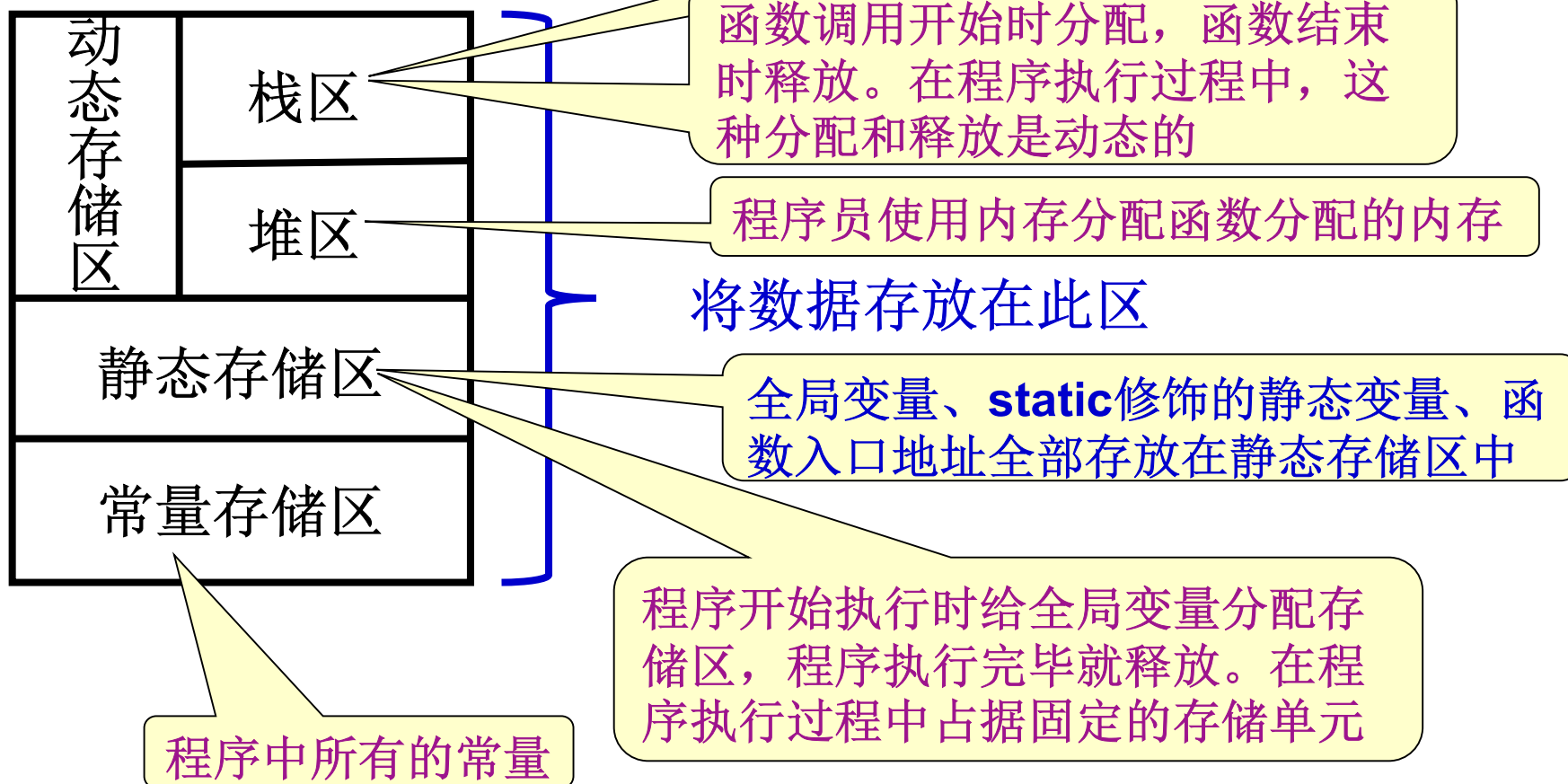
### 7.9.3 全局变量的存储类别

### 7.9.4 存储类别小结

## 7.9.1 动态存储方式与静态存储方式

- 从变量的作用域的角度来观察，变量可以分为全局变量和局部变量
- 从变量值存在的时间(即生存期)观察，变量的存储有两种不同的方式：静态存储方式和动态存储方式
  - ◆ 静态存储方式是指在程序运行期间由系统分配固定的存储空间的方式
  - ◆ 动态存储方式是在程序运行期间根据需要进行动态的分配存储空间的方式

# 程序内存区



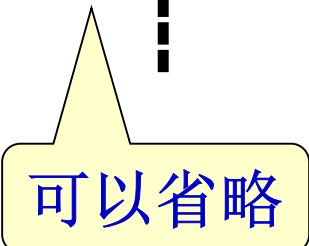
- 在程序运行时，每一个变量和函数都会存储在程序内存区，这样就涉及到两个属性
  - ◆数据类型和数据的存储类别
  - ◆数据类型，如整型、浮点型等
  - ◆存储类别指的是数据在内存中存储的方式（如静态存储和动态存储）
  - ◆存储类别包括：
    - 自动存储、静态存储、寄存器存储、外部存储
  - ◆根据变量的存储类别，可以知道变量的作用域和生存期

## 7.9.2 局部变量的存储类别

### 1. 自动变量(**auto**变量)

- ◆ 局部变量，如果不专门声明存储类别，都是动态地分配存储空间的
- ◆ 调用函数时，系统会给局部变量分配存储空间，调用结束时就自动释放空间。因此这类局部变量称为自动变量
- ◆ 自动变量用关键字**auto**作存储类别的声明

```
int f(int a)
{
    auto int b,c=3;
    !
}
```



可以省略

## 7.9.2 局部变量的存储类别

### 2. 静态局部变量(**static**局部变量)

- 希望函数中的局部变量在函数调用结束后不消失而继续**保留原值**，即其占用的存储单元不释放，在下一次再调用该函数时，该变量已有值(就是上一次函数调用结束时的值)，这时就应该指定该局部变量为“静态局部变量”，用关键字**static**进行声明

例7.14 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

调用三次

```
int f(int a)
{ auto int b=0;
  static c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

每调用一次，开辟新a和b，但c不同

### 3. 寄存器变量(**register**变量)

- 一般情况下，变量（包括静态存储方式和动态存储方式）的值是存放在内存中的
- **寄存器变量**允许将局部变量的值放在**CPU**中的寄存器中
- 现在的计算机能够识别使用频繁的变量，从而自动地将这些变量放在寄存器中，而不需要程序设计者指定
- 寄存器变量存储在**CPU**的寄存器中，所以不能进行取地址运算。
- 寄存器变量是一种存储类别，程序运行时数据存储在**CPU**中的寄存器中，不能定义为全局变量。



## 7.9.3 全局变量的存储类别

- 全局变量都是存放在静态存储区中的。因此它们的生存期是固定的，存在于程序的整个运行过程
  - 一般来说，外部变量是在函数的外部定义的全局变量，它的作用域是**从变量定义处开始**，到本程序**文件的末尾**。在此作用域内，全局变量可以为程序中各个函数所引用。
- 1.** 在一个文件内扩展外部变量的作用域
    - 外部变量有效的作用范围只限于定义处到本文件结束
    - 如果用关键字**extern**对某变量作“外部变量声明”，则可以从“声明”处起，合法地使用该外部变量

```
#include <stdio.h>
```

```
int max( );
```

// 函数声明

```
extern int A,B,C;
```

// 外部声明，扩展作用域

```
int main()
```

```
{
```

```
    printf("%d %d %d\n", A, B, C);
```

```
    scanf("%d %d %d",&A,&B,&C);
```

```
    printf("max is %d\n",max());
```

```
    return 0;
```

```
}
```

```
int A ,B ,C;
```

```
int max( )
```

```
{
```

```
    int m;
```

```
    m=A>B?A:B;
```

```
    if (C>m) m=C;
```

```
    return(m);
```

```
}
```

[Error] 'A' undeclared (first use in this function)

[Note] each undeclared identifier is reported only once for each function it appears in

[Error] 'B' undeclared (first use in this function)

[Error] 'C' undeclared (first use in this function)

## 2. 将外部变量的作用域扩展到其他文件

- ◆一个大程序（系统）往往包含很多程序文件，定义过多外部变量可能出现意料不到的结果。
- ◆例如：一个程序包含两个文件，在两个文件中都要用到同一个外部变量**Num**，不能分别在两个文件中各自定义一个外部变量**Num**。
- ◆应在某一个文件中定义外部变量**Num**，而在另一文件中用**extern**对**Num**作“外部变量声明”
- ◆在编译和连接时，系统会由此知道**Num**有“外部链接”，可以从别处找到已定义的外部变量**Num**，并将在另一文件中定义的外部变量**Num**的作用域**扩展**到本文件

**例7.14** 写一个程序实现给参加答辩的**10**人小组评分、排序、最后按得分从高到低的顺序输出。要求输入、排序、输出都用函数实现，每个函数单独使用一个独立的程序文件，主函数使用一个程序文件，小组得分使用外部数组实现。

➤ **main.c**

```
int a[10];  
void input();  
void sort();  
void output();  
int main() {  
    input();  
    sort_desc();  
    output();  
    return 0;  
}
```

➤ **in.c**

```
#include <stdio.h>
extern int a[10];
void input(){
    int i;
    for(i=0; i<10; i++)
        scanf("%d", &a[i]);
}
```

➤ **out.c**

```
#include <stdio.h>
extern int a[10];
void output()
{
    int i;
    for(i=0; i<10; i++)
        printf("%d ", a[i]);
    printf("\n");
}
```

➤ **sort.c**

```
extern int a[10];  
void sort_desc()  
{  
    int i, j;  
    for(i=0; i<9; i++)  
    {  
        for(j=0; j<9-i; j++)  
        {  
            if(a[j]<a[j+1])  
            {  
                int t = a[j];  
                a[j] = a[j+1];  
                a[j+1] = t;  
            }  
        }  
    }  
}
```

### 3. 将外部变量的作用域限制在本文件中

- 有时在程序设计中希望某些外部变量只限于被本文件引用。这时可以在定义外部变量时加一个**static**声明。

只能用于本文件

```
file1.c  
static int A;  
int main ( )  
{  
    .....  
}
```

本文件仍然不能用

```
file2.c  
extern A;  
void fun (int n)  
{  
    .....  
    A=A*n;  
    .....  
}
```

## 7.9.4 存储类别小结

➤ 说明:

- ◆ 变量是动态存储还是静态存储，是由程序运行时为其分配的内存区域决定的，而不是在定义变量时是否使用存储类别修饰符**static**来决定。所以：**不要认为**对外部变量加**static**声明后就采取静态存储方式，而不加**static**的就是采取动态存储，外部变量都是静态存储的。
- ◆ 声明局部变量的存储类型和声明全局变量的存储类型的含义是不同的
- ◆ 对于**局部变量**来说，声明存储类型的作用是指定变量存储的区域以及由此产生的生存期的问题，而对于**全局变量**来说，声明存储类型的作用是变量作用域的扩展问题



- 用**static** 声明一个变量的作用是：
  - (1) 对局部变量用**static**声明，把它分配在静态存储区，该变量在整个程序执行期间不释放，程序运行时为其所分配的内存空间始终存在。
  - (2) 对全局变量用**static**声明，则该变量的作用域只限于本程序文件（即被声明的文件中）。
- 注意存储类型说明符**auto/register/static/extern**的使用

## 7.10 函数说明符

- 函数说明符一般应出现在函数声明中。**main**函数不能使用函数说明符。函数说明符可以在函数声明中不止一次出现，但行为与仅出现一次相同。
- **inline**
  - ◆ 在函数声明时，使用函数说明符**inline**，则该函数称为内联函数。内联函数在调用时，建议编译器将指定的函数体插入并取代每一处调用该函数的地方，使得程序尽可能快。
- **\_Noreturn**
  - ◆ **\_Noreturn**指定函数不会执行到**return**语句或到达函数体结尾而返回(可通过执行**abort/exit**等返回)，若声明**\_Noreturn**的函数返回，则行为未定义。若编译器能检测此错误，则推荐编译器诊断。

# 7.11 内部函数和外部函数

## 7.11.1 内部函数

## 7.11.2 外部函数

## 7.11.1 内部函数

- 如果一个函数只能被本文件中其他函数所调用，它称为**内部函数**。
- 在定义内部函数时，在函数名和函数类型的前面加**static**，即：  
**static** 类型名 函数名(形参表)
- 内部函数又称静态函数，因为它是用**static**声明的
- 通常把只能由本文件使用的函数和外部变量放在文件的开头，前面都冠以**static**使之局部化，其他文件不能引用
- 提高了程序的可靠性

## 7.11.2 外部函数

- 如果在定义函数时，在函数首部的最左端加关键字 **extern**，则此函数是外部函数，可供其他文件调
- 如函数首部可以为  
**extern int fun (int a, int b)**
- 如果在定义函数时省略 **extern**，则默认为外部函数
- 仍以 **7.14** 为例