

C语言与程序设计

The C Programming Language



第5章 函数与程序结构

华中科技大学计算机学院

毛伏兵



第5章 函数与程序结构

- 结构化编程和C程序的一般结构
- 函数的机制，包括函数定义、函数声明、函数调用、变量的存储类型等。
- 递归



5.1 C程序的一般结构

5.1.1 结构化程序设计

2条编程标准：

- (1) 程序中的控制流尽可能简单。**
- (2) 把一个问题逐步细化分解为若干子问题，用函数实现子问题。**



5.1.2 蒙特卡罗模拟：猜数游戏

- 计算机随机产生一个数（1~1000）。
- 游戏者猜直到正确为止，计算出猜数的次数。
- 发出提示信息“Too high”或“Too low”。

5.1.2 蒙特卡罗模拟：猜数游戏

- **模拟算法：**编程实现现实世界中的随机事件，例如，抛硬币、掷骰子和玩牌等。
- **蒙特卡罗模拟：**使用随机数来模拟。
- **随机数：**具有不确定性和偶然性特点，应用领域：
软件测试—测试数据，加密系统—密钥，网络—验证码
- **随机数发生器：**生成随机数的函数

```
int rand (void) ;    // 在 stdlib.h中
```
- **伪随机数：**依靠计算机内部算法产生的“随机”数



主程序结构

do {

计算机产生一个1到1000的随机数;

游戏者猜数, 直至猜对;

printf("Play again? (Y/N) ");

scanf("%1s", &cmd);

} while (cmd == 'y' || cmd == 'Y');

int GetNum (void);

void GuessNum(int x);



主程序结构

```
do {  
    magic = GetNum( ); /* 产生随机数*/  
    GuessNum(magic); /* 猜数 */  
    printf("Play again? (Y/N) ");  
    scanf("%1s", &cmd);  
} while (cmd == 'y' || cmd == 'Y');
```

rand是接口**stdlib.h**中的一个函数，
它返回一个非负并且不大于常量**RAND_MAX**的随机整数，
RAND_MAX（32767）的值取决于系统。

int GetNum(void) ;

```
/******
```

函数名称: **GetNum**

函数功能: 产生一个1到**MAX_NUMBER**之间的随机数，供游戏者猜测。

函数参数: 无

函数返回值: 返回产生的随机数

```
*****/
```

```
int GetNum(void) /* 注意：后面无分号 */
```

```
{
```

```
    int x;
```

```
    printf("A magic number between 1 and %d has been chosen.\n",  
           MAX_NUMBER);
```

```
    x=rand();
```

```
    x= x % MAX_NUMBER + 1;
```

```
    return(x);
```

```
}
```

```
/* 调用标准库函数rand产生一个随机数 */
```

```
/* 将这个随机数限制在1~MAX_NUMBER之间 */
```




函数 `void GuessNum(int x)`

`/*游戏者猜数，直至猜对*/`

`for(;;) {`

`输入猜测的数给变量guess`

`if（猜对了） 结束函数`

`else if（小了） 输出太小的提示`

`else 输出太大的提示`

`}`

[源程序\ex5_1_12.c](#)

main函数

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_NUMBER 1000
int GetNum (void);      /* 函数原型 */
void GuessNum(int) ;    /* void GuessNum(int x) ;函数原型 */
int main(void)
{
    char command;
    int magic;
    do
    {
        magic = GetNum( ); /*调用GetNum产生随机数*/
        GuessNum(magic); /* 调用GuessNum猜数 */
        printf("Play again? (Y/N) ");
        scanf("%1s", &command); /* 询问是否继续 */
    } while (command == 'y' || command == 'Y');
    return 0;
}
```

行吗？

伪随机

- 伪随机数是指用数学递推
- 应用最广的递推公式:

$$a_0 = \text{seed}$$

$$a_n = (A * a_{n-1} + B) \% M, n \geq 1$$

其中 A, B, M 是产生器设定的常数, 用户不能更改。

seed: 种子参数, 该发生器从称为种子 (一个无符号整型数) 的初始值开始用确定的算法产生随机数。

- ◆ 产生器给定了初始值
- ◆ **seed** 应该在哪儿定义并初始化?
- ◆ 允许用户设置初始值 (修改)
- ◆ 怎么修改?

```
int rand( )
{
    seed = (A * seed + B) % M;
    return seed;
}
```

初始化随机数

```
void srand(unsigned int x )
{
    seed=x;
}
```

- **初始化随机数发生器**：改变种子**seed**值的函数

void srand (unsigned) ;

- **用什么值做种子？**

用系统时间（由**time**函数得到）作为种子。

函数**time**返回自**1970年1月1日**以来经历的秒数。

srand (time (NULL)) ;

该初始化语句放在程序什么位置？

- 函数**srand**和**rand**的原型在**stdlib.h**中，
函数**time**的原型在**time.h**中

源程序\ex5_1.c



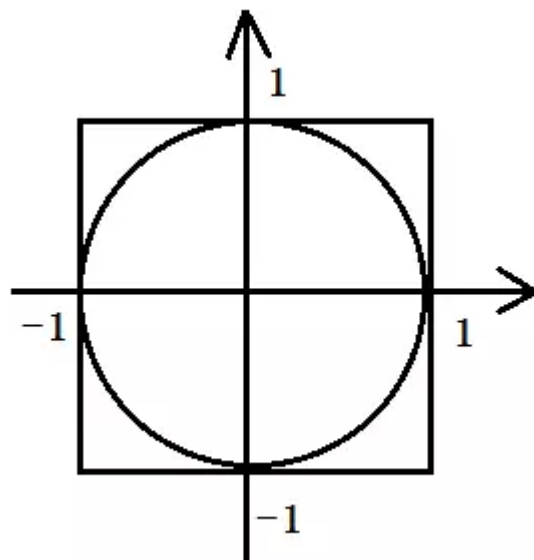
```
unsigned int seed; // 外部变量
```

```
int rand( )  
{  
    seed= (A*seed+B) % M;  
    return seed;  
}
```

```
void srand(int x )  
{  
    seed=x;  
}
```

练习----蒙特卡罗法求圆周率近似值

- 有一个如图所示正方形及其内切圆，往正方形内扔飞镖 n 次，当 n 足够大，比值“落在圆内的次数/落在正方形内的次数 n ”将无限接近“圆的面积/正方形的面积”，即 $\pi/4$ 。 n 越大， π 的近似结果越精确。

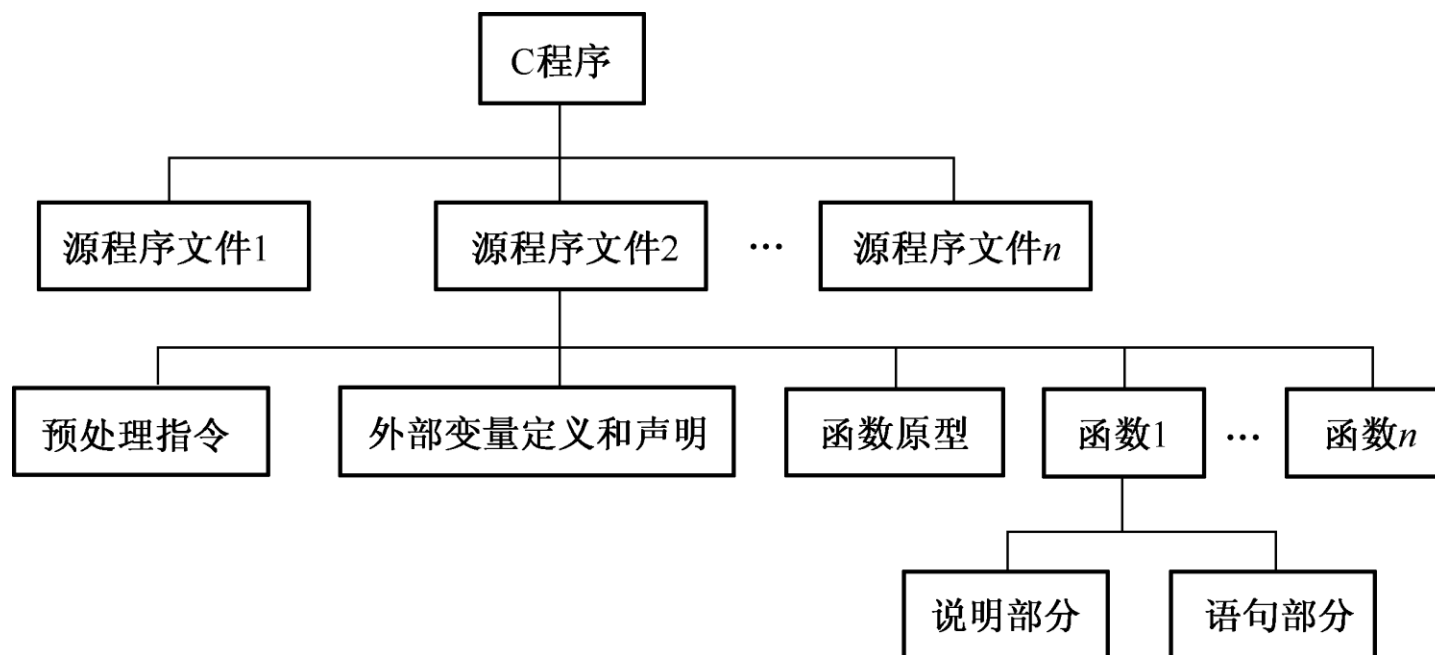




结构化程序设计的益处

- 使程序编制方便，易于管理、修改和调试。
- 增强了程序的可读性、可维护性和可扩充性，方便于多人分工合作完成程序的编制。
- 函数可以公用，避免在程序中使用重复的代码。
- 提高软件的可重用性，软件的可重用性是转向面向对象程序设计的重要因素。

5.1.3 C程序的结构





5.2 函数的定义与函数的声明

- 程序中若要使用自定义函数实现所需的功能，需要做三件事：
 - ① 按语法规则编写完成指定任务的函数，即定义函数；
 - ② 有些情况下在调用函数之前要进行函数声明；
 - ③ 在需要使用函数时调用函数

5.2.1 函数的定义

函数定义的一般形式为：
类型名 函数名（参数列表）

{

声明部分

语句部分

}

形式参数（简称形参）
无参数：有void或无

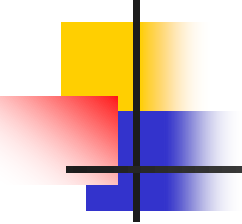
无返回值：类型为void



5.2.2 函数的返回值

- **return**语句可以是如下两种形式之一：
 - (1) **return ;** **/* void函数 */**
 - (2) **return 表达式 ;** **/* 非void函数 */**
- **void**函数也可以不包含**return**语句。如果没有**return**语句，当执行到函数结束的右花括号时，控制返回到调用处，把这种情况称为离开结束。
- 表达式值的类型应该与函数定义的返回值类型一致，如果不相同，就把表达式值的类型自动转换为函数的类型。

函数返回的值，程序可以使用它， 也可以不使用它



```
while(...) {  
    getchar();          /* 返回值不被使用 */  
    c=getchar();        /* 返回值被使用   */  
    ...  
}
```

5.2.3 函数的声明

- 函数定义出现在函数调用后
 - 被调用函数在其它文件中定义
- 必须在函数调用之前给出函数原型

类型名 函数名（参数类型表）；

`int max(int x , int y) ;`

或

`int max(int , int) ;`

无参数：必须写**void**



良好的编程风格

- 在函数调用之前必须给出它的函数定义、函数原型或两者都给出。
- 引入标准头文件的主要原因是它含有函数原型。



5.3 函数调用与参数传递

5.3.1 函数调用

1. 函数调用的形式

函数名（实参列表）



实参是一个表达式
无参函数：为空



函数调用的形式

(1) 作为表达式语句出现。

```
GuessNum( );
```

```
putchar(c);
```

(2) 作为表达式中的一个操作数出现。例如：

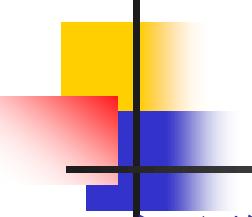
```
c=getchar()
```

```
magic = GetNum( );
```

(3) 作为函数调用的一个实参出现，即嵌套调用。

```
printf("%10.0f",sqrt(x));
```

```
while(putchar (getchar())!='#' );
```

2. 函数调用的执行过程

- 为了说明函数的调用过程，请看下面程序。

【例5.3】编写程序实现如下功能：分列整齐地显示整数1到10的2至5次幂。输出结果如下所示：

Int	Square	Cube	Quartic	Quintic
1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64	256	1024
5	25	125	625	3125
6	36	216	1296	7776
7	49	343	2401	16807
8	64	512	4096	32768
9	81	729	6561	59049
10	100	1000	10000	100000

- 源程序\ex5_3.c

3. 实参的求值顺序

- 由具体实现确定，有的从左至右计算，有的按从右至左计算。

a=1;

power(a,a++)

----从左至右: **power(1,1)**

----从右至左: **power(2,1)** (多数)

为了保证程序清晰、可移植，应避免使用会引起副作用的实参表达式。



5.3.2 参数的值传递

- 参数的传递方式是“值传递”，实参的值单向传递给相应的形参。如果实参、形参都是 x ，被调用函数不能改变实参 x 的值。

【例5.4】 改写例5.3来说明值传递概念。

```
#include<stdio.h>
double power(int,int); /* 函数原型 */
int main(void)
{
    int x, n;
    /* 分列整齐地显示整数1到10的2至5次幂 */
    for(x=1;x<=10;x++)
    {
        for(n=2;n<=5;n++)
            printf("%10.0f", power(x,n));
        printf("\n");
    }
    return 0;
}

double power(int x, int n) // 计算x的n次方
{
    double p;
    for (p=1;n>0;n--)
        p*=x;
    return p;
}
```



传地址（引用）调用

- 传地址（引用）调用是将变量的地址传递给函数，函数既可以使用，也可以改变实参变量的值。标准库函数**scanf**就是一个引用调用的例子，通过地址实参返回一个或多个数据。程序员也可以定义这种带有地址形参的函数，这部分内容将在第9章中介绍。
- `int x; scanf("%d",&x);`



5.4 作用域与可见性

- 作用域是指标识符（变量或函数）的有效范围，也就是指程序正文中可以使用该标识符的那部分程序段。
- 可见性：块多重嵌套时，同名外层变量不可见
- 按照作用域，变量分：
局部变量和全局变量



5.4.1 局部变量和全局变量

- **局部变量**：在函数内部定义的变量，作用域是定义该变量的程序块，程序块是带有说明的复合语句（包括函数体）。不同函数可同名，同一函数内不同程序块可同名。**形式参数是局部变量。**
- **外部变量**：在函数外部定义的变量，其缺省作用域从其定义处开始一直到其所在文件的末尾，由程序中的部分或所有函数共享。

```
int p=1, q=5;
```

全局变量

```
float f1( int a)
```

```
{ int b,c;
```

a,b,c有效

```
.....
```

局部变量

```
}  
char c1,c2;
```

p,q有效

```
main( )
```

```
{ int m, n;
```

m,n有效

```
.....
```

```
}
```

c1,c2有效

extern 声明

```
srand(int x)
```

```
{
```

```
    seed=x;
```

```
}
```

```
int seed=C;
```

```
int rand( )
```

```
{
```

```
    seed = (A*seed+B) % M ;
```

```
    return seed;
```

```
}
```

引用出错：无作用

外部变量的**定义性**声明语句
从定义点开始有效直至文件尾



extern声明

```
srand(int x)
```

```
{
```

```
    extern int seed;
```

```
    seed=x;
```

```
}
```

```
int seed=C;
```

```
int rand( )
```

```
{
```

```
    seed = (A*seed+B) % M ;
```

```
    return seed;
```

```
}
```

外部变量的**引用性**声明语句

外部变量的**定义性**声明语句
从定义点开始有效直至文件尾



定义性声明和引用性声明的区别

- ◆ **外部变量的定义性声明**---- 分配存储单元
 - ✓ 位于所有的函数之外
 - ✓ 定义一次
 - ✓ 可初始化
- ◆ **外部变量的引用性声明**---- 通报变量的类型
 - ✓ 位于在函数内或函数外
 - ✓ 出现多次
 - ✓ 不能初始化

外部变量 PK 形式参数

- 函数之间可以通过外部变量进行通信。
- 一般地，函数间通过形式参数进行通信更好。这样有助于提高函数的通用性，降低副作用。

```
int main(void)
{ int m;
  m = rand( )%10+1;
  guess(m);
  guess(12);
}
void guess(int x)//猜x
{ ..... }
```

Int magic;

Int main(void)

```
{ Int m;
  m = rand( )%10+1;
  guess(m);
  } guess();
```

void guess() //猜magic

void guess() //猜magic

```
{ ..... }
```

```
int min;
```

外部变量

```
int maxmin(int x,int,y);
```

```
int maxmin (int x, int y)
```

```
{ int z;
```

```
min=(x<y)?x : y;
```

```
z=(x>y)? x : y ;
```

```
return z;
```

函数值为4

```
}
```

```
int main (void)
```

```
{ int a=4,b=1,max;
```

```
max=maxmin(a , b) ;
```

```
printf("The max is %d\nThe min is %d",max,min);
```

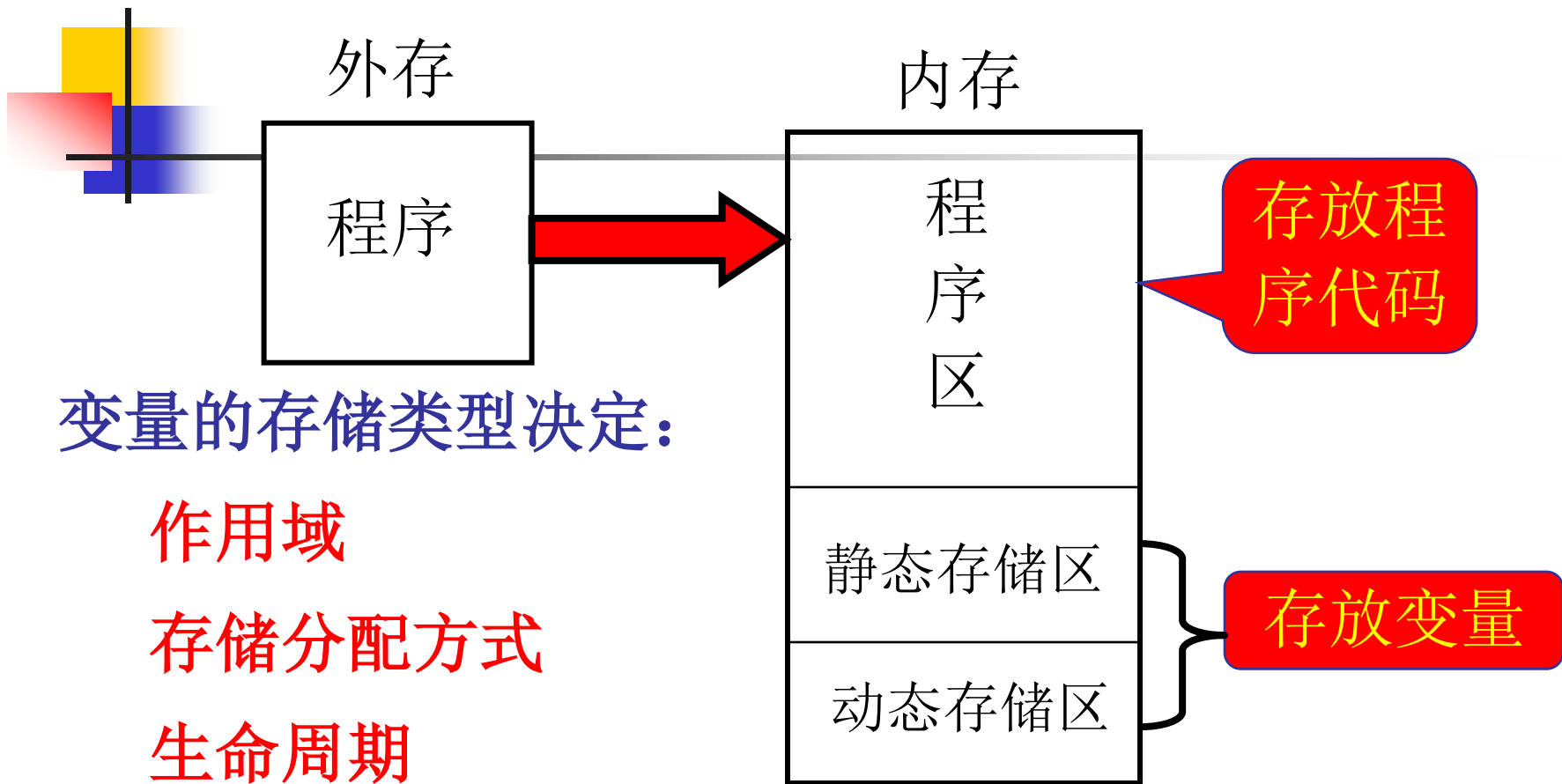
```
return 0;
```

```
}
```

The max is 4

The min is 1

5.5 变量的存储类别



变量的存储类型决定：

作用域

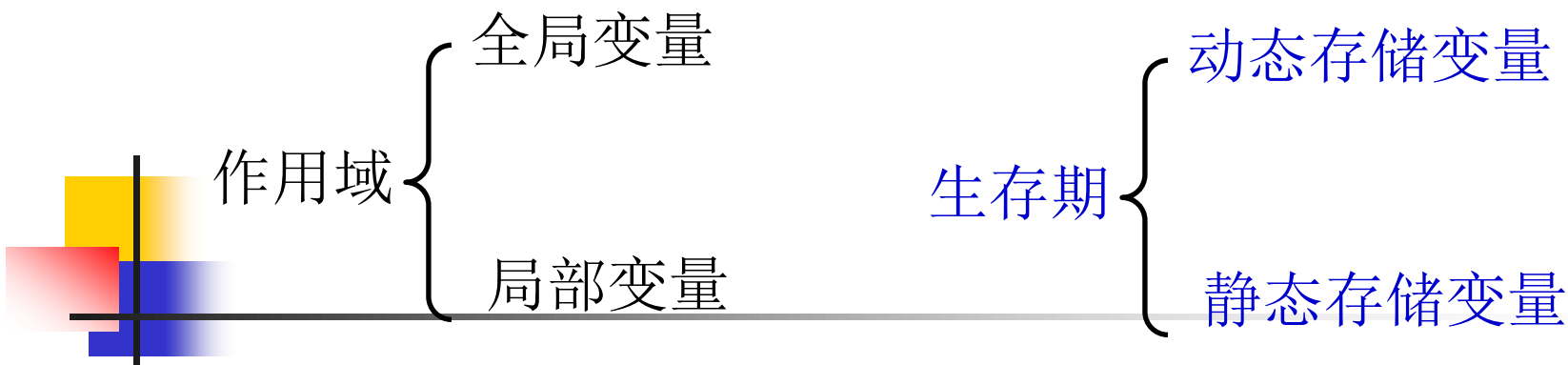
存储分配方式

生命周期

初始化方式

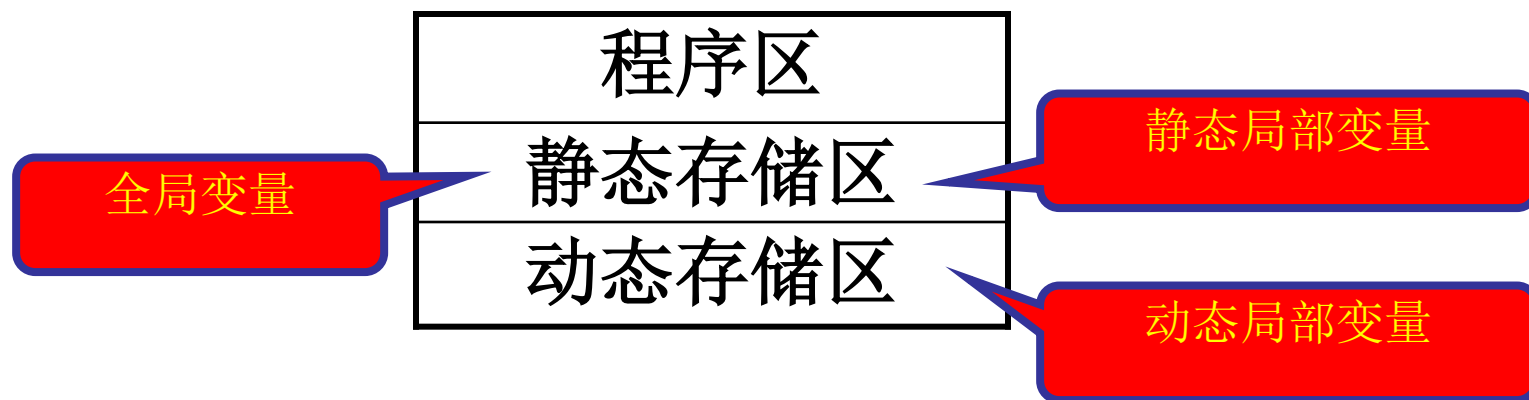
需要区分变量的存储类型

关键字有：**auto**、**extern**、**static**和**register**



静态存储：在文件运行期间有固定的存储空间，直到文件运行结束。

动态存储：在程序运行期间根据需要分配存储空间，**函数结束后立即释放空间**。若一个函数在程序中被调用两次，则每次分配的单元有可能不同。





5.5.1 存储类型auto

- 局部变量的缺省存储类型是**auto**，称自动变量
- `auto int a;` /*等价于`int a;` 也等价于`auto a;` */
- **作用域**：局部于定义它的块，从块内定义之后直到该块结束有效。
- **存储分配方式**：动态分配方式,即在程序运行过程中分配和回收存储单元。
- **生命周期**：短暂的，只存在于该块的执行期间。
- **初始化方式**：定义时没有显示初始化，其初值是不确定的；有显示初始化，则每次进入块时都要执行一次赋初值操作。



5.5.2 存储类型extern

- 外部变量的存储类型是**extern**，但在定义时不使用关键字**extern**。
- 外部变量的**作用域**:从定义之后直到该源文件结束的所有函数，通过用**extern**做声明，外部变量的作用域可以扩大到整个程序的所有文件。
- **存储分配方式**:静态分配方式,即程序运行之前，系统就为外部变量在静态区分配存储单元，整个程序运行结束后所占用的存储单元才被收回。
- **生命周期**:永久的，存在于整个程序的执行期间。
- **初始化方式**:定义时没有显示初始化，初值**0**；有显示初始化，只执行一次赋初值操作。

定义点前或其它文件引用： 需用**extern**作引用声明

```
srand(int x)
{
    extern int seed;
    seed=x;
}
int seed=C;
int rand( )
{
    seed = (A*seed+B) % M ;
    return seed;
}
```

引用在前，定义在后

文件file1.c中的内容为:

```
int a;
```

```
int main(void)
```

```
{ extern int fun (int);
```

```
int b=3, c, d, m;
```

```
...
```

```
c=a*b;
```

```
d= fun(m);
```

```
...
```

```
}
```

外部变量定义

外部变量说明

引用本文件外
定义的全局变量

文件file2.c中的内容为:

```
extern int a;
```

```
int fun (int n )
```

```
{ int i, y=1;
```

```
...
```

```
y*=a;
```

```
return y;
```

```
}
```



5.5.3 存储类型static

关键字**static**有两个重要而截然不同的用法:

- (1) 用于定义局部变量, 称为静态局部变量。
- (2) 用于定义外部变量, 称为静态外部变量。


存储分配方式:静态分配方式

生命周期:永久的,

缺省初值: 0, 只执行一次

静态局部变量和自动变量有根本性的区别。

静态外部变量和外部变量区别: 作用域不同。



```
 srand(int x)
{
    seed=x; // seed在此无作用
}
```

```
int rand( )
{
    int seed=D; // 自动变量，存于栈，每次调用初始化为D
    seed = (A*seed+B) % M ;// 不能产生随机系列
    return seed;
}
```



```
srand(int x)
```

```
{  
    seed=x; // seed在此无作用，用户无法改变种子参数  
}
```

```
int rand( )
```

```
{  
    static int seed=D; // 静态局部变量，存于静态区  
                        // 初始化只执行1次  
    seed = (A*seed+B) % M; // 可以产生随机系列  
    return seed;  
}
```



```
int seed=D; // 外部变量，存于静态区， 初始化只执行1次  
srand(int x)
```

```
{  
    seed=x; // seed在此有作用， 用户可改变种子参数  
}
```

```
int rand( )  
{  
    seed = (A*seed+B) % M ; // 可以产生随机系列  
    return seed;  
}
```



1. 静态局部变量

- 作用域：自动变量一样

【例5.7】 编程计算 $1!+2!+3!+4!+\dots n!$

- 将求阶乘定义成函数，要求使用**static**使计算量最小。


```
scanf("%d",&n);
for(i=1;i<=n;i++)
    sum+=fac(i);
printf("1!+2!+...+%d!=%ld\n",n,sum);
return 0;
```

```
}
```

```
/******
```

函数名称: fac

函数功能: 利用静态局部变量的特性求一个整数的阶乘。

函数参数: 形参n是int型

函数返回值: 返回n的阶乘, 类型long

```
*****/
```

```
long fac(int n)
```

```
{
```

```
    static long f=1;    /* 静态局部变量 */
```

```
    f *=n;
```

```
    return f;
```

```
}
```

2. 静态外部变量

```
static int seed=D;
srand(int x)
{
    seed=x;
}
int rand( )
{
    seed = (A*seed+B) % M ;
    return seed;
}
```

静态外部变量

只能作用于定义它的文件，
其它文件中的函数不能使用



2. 静态外部变量

- 静态外部变量和外部变量的区别是作用域的限制。
- 静态外部变量只能作用于定义它的文件，其它文件中的函数不能使用，
- 外部变量用**extern**声明后可以作用于其它文件。
- 使用静态外部变量的好处在于：当多人分别编写一个程序的不同文件时，可以按照需要命名变量而不必考虑是否会与其它文件中的变量同名，**保证文件的独立性**。
- 和局部变量能够屏蔽同名的外部变量一样，一个文件中的静态外部变量能够屏蔽其他文件中同名的外部变量。在静态外部变量所在的文件中，同名的外部变量不可见。

5.5.4 存储类型register

- 用来定义局部变量, **register** **建议**编译器把该变量存储在计算机的高速硬件寄存器中, 除此之外, 其余特性和自动变量完全相同。
- 使用**register**的目的是为了提高程序的执行速度。程序中最频繁访问的变量, 可声明为**register**。

```
register int i;           /* 等价于register i; */  
for (i=0;i<=N;i++) {    ...}
```

- 不可多, 必要时使用。
- 无地址, 不能使用**&**运算。