

# 函数

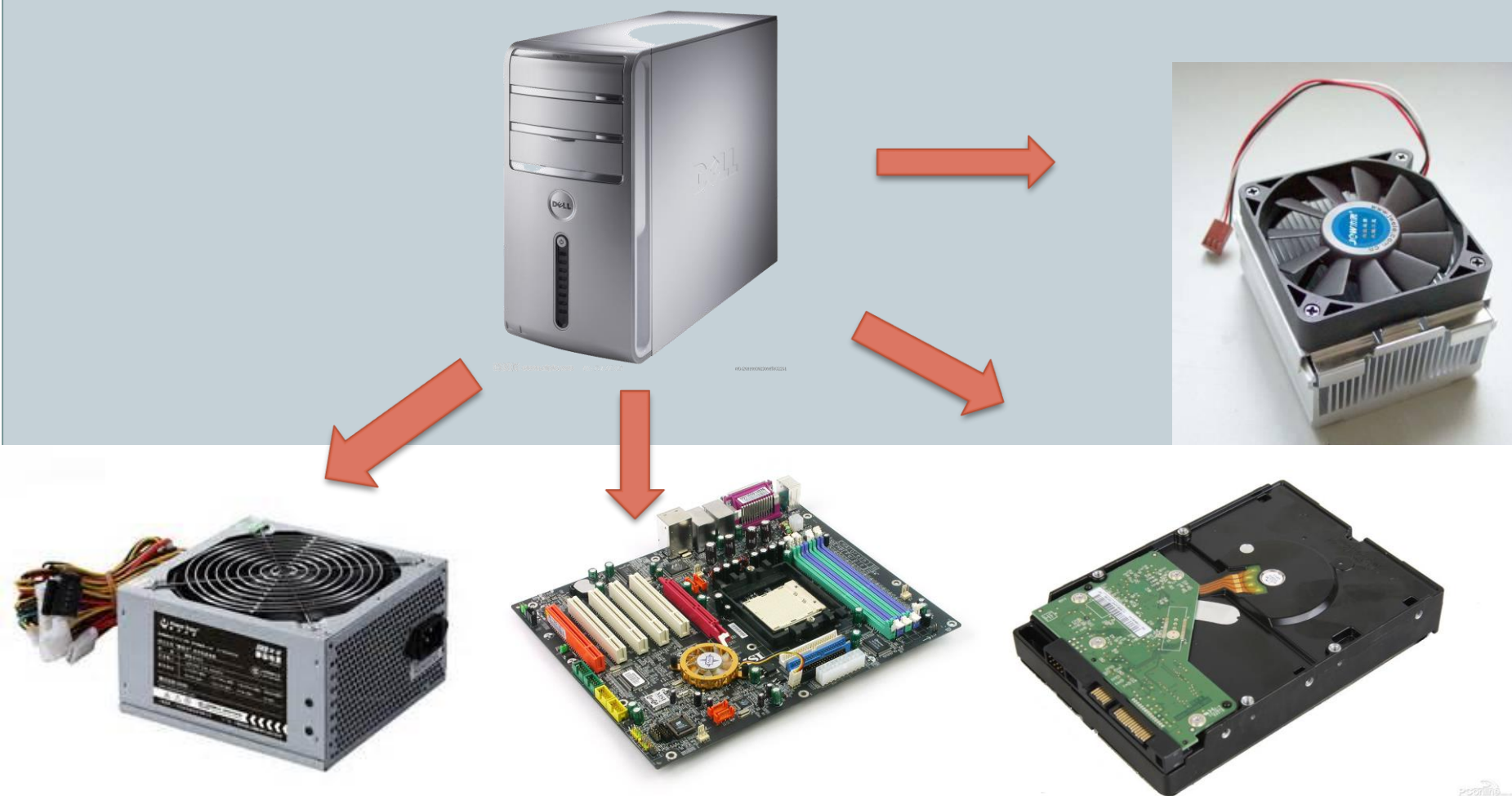


程 森

E-mail: [mew\\_cheng@outlook.com](mailto:mew_cheng@outlook.com)

# 函数的意义

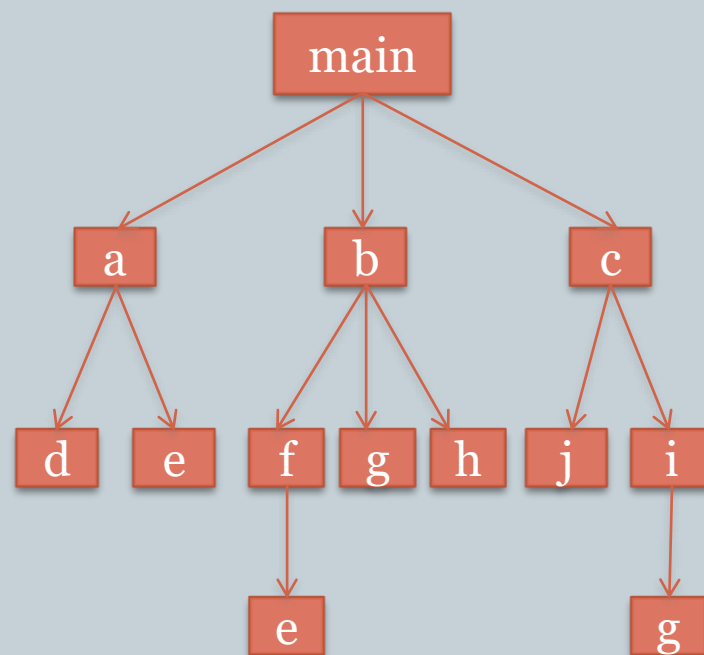
- 采用“组装”的办法来简化程序设计的过程



# 函数的意义



- 在设计一个较大的程序时，往往把它分为若干个程序模块，每一个模块包括一个或多个函数，每个函数实现一个特定的功能



# 函数的意义



- 实现的函数目的是实现功能
- 每一个函数用来实现一个特定的功能



# 函数的意义



- 先看一个例子：想输出以下的结果，用函数调用实现

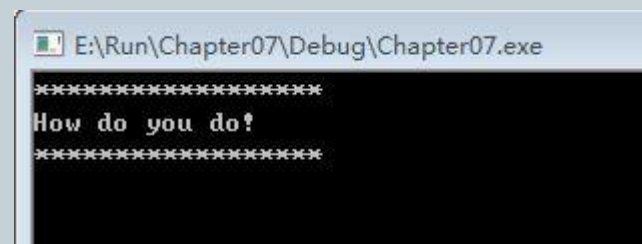
```
#include <stdio.h>

int main()
{
    void print_star();
    void print_message();
    print_star();
    print_message();
    print_star();

    return 0;
}

void print_star()
{
    printf("*****\n");
}

void print_message()
{
    printf("How do you do!\n");
}
```



# 函数的组成



- 一个C程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件
- 一个源程序文件由一个或多个函数以及其他有关内容(如指令、数据声明与定义等)组成
- C程序的执行是从main函数开始的，如果在main函数中调用其他函数，在调用后流程返回到main函数，在main函数中结束整个程序的 允许

# 函数的组成



- 所有函数都是平行的，即在定义函数时是分别进行的，是互相独立的
- 从用户使用的角度看，函数有两种
  - 库函数：由系统提供，用户不必自己定义
  - 用户自己定义的函数
- 从函数的形式看，函数分两类
  - 无参函数
  - 有参函数

# 函数的定义



- 定义函数应包括的内容

- 指定函数的名字，以便以后按名调用
- 指定函数的类型，即函数返回值的类型
- 指定函数的参数的名字和类型，以便在调用函数时向它们传递数据
- 指定函数应当完成什么操作，即函数的功能



# 函数的定义



## ● 定义无参函数

```
类型名  函数名()  
{  
    函数题  
}  
或  
类型名  函数名()  
{  
    函数体  
}
```

## 定义有参函数

```
类型名  函数名(形式参数表列)  
{  
    函数体  
}
```

## 定义空函数

```
类型名  函数名()  
{  
}
```



# 函数调用



- 调用函数的一般形式
  - 函数名(实参表列)
- 函数调用方式
  - 函数调用语句, 如 `printf_star()`
  - 函数表达式, 如 `c=2*max(a, b);`
  - 函数参数, 如 `m=max(a, max(b, c));`

# 函数调用



- 函数调用时的数据传递
  - 形式参数和实际参数
  - 实参和形参的数据传递
- 注意
  - 实参可以是常量、变量或表达式
  - 实参与形参的类型应相同或赋值兼容

# 函数调用



- 一个例子：输入两个整数，要求输出其中值较大者。  
要求用函数来找到大数

## ○ 解题思路

- ✦ 定义函数，对给定的参数返回较大者
- ✦ 函数应有两个参数，以便从主函数接收两个整数

# 函数调用



## ● 编写程序

```
int max(int x, int y)
{
    int z;
    z = x>y? x: y;
    return (z);
}

#include <stdio.h>

int main()
{
    int max(int x, int y);
    int a, b, c;
    printf("please enter two integer numbers:");
    scanf("%d %d", &a, &b);
    c = max(a, b);
    printf("max is %d\n", c);

    return 0;
}
```

```
E:\Run\Chapter07\Debug\Chapter07.exe
please enter two integer numbers:12, -34
max is 12
```

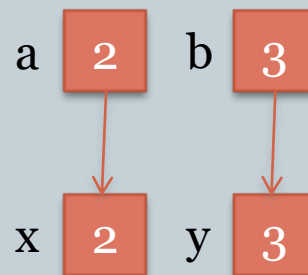
# 函数调用



## 参数传递

`c = max(a, b);` (main函数)

```
int max(int x, int y)    (max函数)
{
    int z;
    z = x > y ? x : y;
    return (z);
}
```



一人一票

# 函数调用



- 定义函数中指定的形参，在未出现函数调用时，它们并不占内存中的存储单元
- 将实参对应的值传递给形参
- 在执行max函数期间，由于形参已经有值，就可以利用形参进行有关的运算

# 函数调用



- 通过return语句将函数值带回到主调函数main
- 调用结束，形参单元被释放
- 实参向形参的数据传递是“值传递”，单向传递，只能由实参传给形参，而不能由形参传给实参



# 函数的返回值



- 通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)
  - 如，`c = max(2, 3);`

到站了(行程完毕)!



# 函数的返回值



- 函数返回值通过return语句获得
- 函数值属于某一个确定的类型
- 在定义函数时指定的函数类型一般应该和return语句中的表达式类型一致
- 对于不带返回值的函数，应定义函数为“void类型”

# 函数的返回值



- 一个例子：将在max函数中定义的变量z改为float型。  
函数返回值的类型与指定的函数类型不同

```
#include <stdio.h>

int main()
{
    int max(float x, float y);
    float a, b;
    int c;
    scanf("%f, %f", &a, &b);
    c = max(a, b);
    printf("max is %d\n", c);

    return 0;
}

int max(float x, float y)
{
    float z;
    z = x > y ? x : y;

    return (z);
}
```

```
E:\Run\Chapter07\Debug\Chapter07.exe
1.5, 2.6
max is 2
```

# 调用函数的声明



- 在一个函数中调用另一个函数(即被调用函数)需要具备如下条件
  - 首先被调用的函数必须是已经定义的函数(是库函数或自定义)
  - 如果使用库函数, 应该在文件开头用#include指令将调用有关库函数时所需用到的信息
  - 如果使用用户自己定义的函数, 而该函数的位置在调用它的函数(即主调函数)的后面(在同一文件中), 应该在主调函数中对被调用的函数作声明(declaration)

# 调用函数的声明



- 对函数的“定义”和“声明”不是同一回事
  - 函数的定义是指对函数功能的确立
  - 函数的声明的作用是把函数的名字、函数类型以及形参的类型、个数和顺序通知编译系统

```
char letter(char, char);  
float f(float, float);  
int i(float, float);  
int main()  
{  
    :  
}  
//下面定义被 main 函数调用的 3 个函数  
char letter(char c1, char c2)  
{  
    :  
}  
float f(float x, float y)  
{  
    :  
}  
int i(float j, float k)  
{  
    :  
}
```

# 函数原型



- 函数类型 函数名(参数类型1,参数名1,参数类型2, 参数2,..., 参数类型n, 参数名n)
- 函数类型 函数名(参数类型1,参数类型2,...,参数类型n)

# 调用函数的声明



- 输入两个实数，用一个函数求出它们之和
- 解题思路：编写一个函数，计算两个参数之和

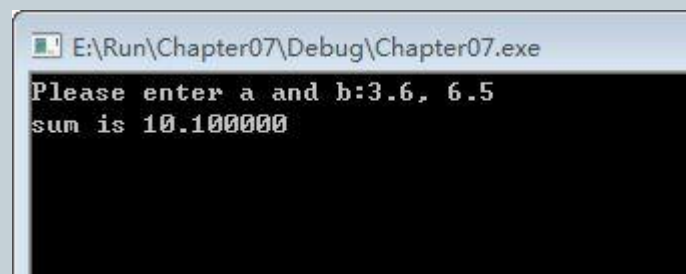
```
#include <stdio.h>

int main()
{
    float add(float x, float y);
    float a, b, c;
    printf("Please enter a and b:");
    scanf("%f, %f", &a, &b);
    c = add(a, b);
    printf("sum is %f\n", c);

    return 0;
}

float add(float x, float y)
{
    float z;
    z = x + y;

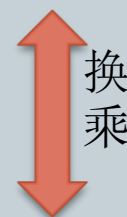
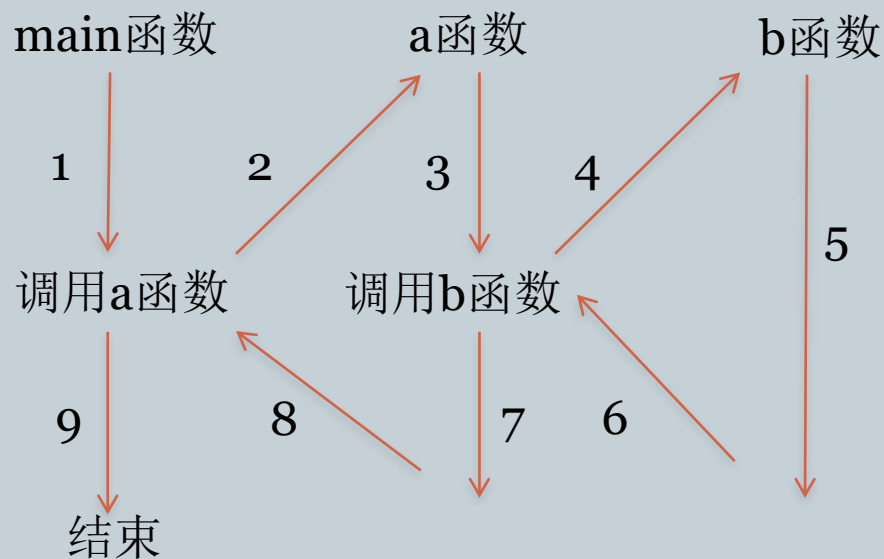
    return (z);
}
```



# 函数的嵌套调用



- C语言的函数定义是互相平行、独立的
- 定义函数时，一个函数内不能再定义另一个函数，也就是不能嵌套定义，但可以嵌套调用函数





# 函数的嵌套调用



- 一个例子：输入4个整数，找出其中最大的数
- 解题思路：在主函数中调用找出4个数中最大者的函数，再在函数中调用另一个函数找出两个数中的大者

# 函数的嵌套调用



## • 编写程序

```
#include <stdio.h>

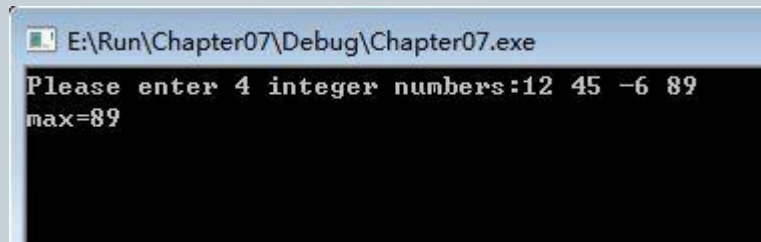
int main()
{
    int max4(int a, int b, int c, int d);
    int a, b, c, d, max;
    printf("Please enter 4 integer numbers:");
    scanf("%d %d %d %d", &a, &b, &c, &d);
    max = max4(a, b, c, d);
    printf("max=%d\n", max);

    return 0;
}

int max4(int a, int b, int c, int d)
{
    int max2(int a, int b);
    int m;
    m = max2(a, b);
    m = max2(m, c);
    m = max2(m, d);

    return (m);
}

int max2(int a, int b)
{
    if (a >= b)
        return a;
    else
        return b;
}
```



```
E:\Run\Chapter07\Debug\Chapter07.exe
Please enter 4 integer numbers:12 45 -6 89
max=89
```

# 函数的递归调用



- 函数的**递归**调用：在调用一个函数的过程中又出现直接或间接地调用该函数本身

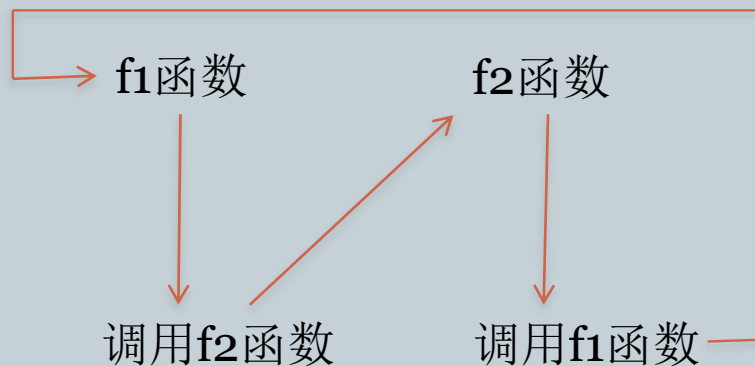
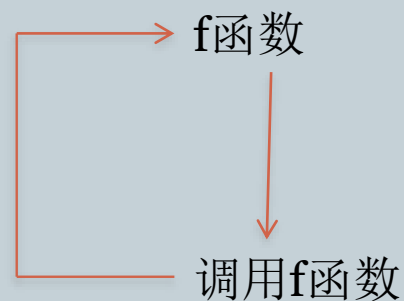


# 函数的递归调用



- 一个递归问题分为“回溯”和“递归”

```
int f(int x)
{
    int y, z;
    z = f(y);
    return (2*z);
}
```



# 函数的递归调用



- 一个例子：有5个学生坐在一起，问第5个学生多少岁，他说比第4个学生大2岁。问第4个学生岁数，他说比第3个学生大2岁。问第3个学生，又说比第2个学生大2岁。问第2个学生，说比第1个学生大2岁。最后问第1个学生，他说是10岁。请问第5个学生多大。
- 解题思路：要求第5个学生的年龄，必须依次计算前4个学生的年龄

$$\text{age}(5) = \text{age}(4) + 2$$

$$\text{age}(4) = \text{age}(3) + 2$$

$$\text{age}(3) = \text{age}(2) + 2$$

$$\text{age}(2) = \text{age}(1) + 2$$

$$\text{age}(1) = 10$$

$$\text{age}(n) = 10 \quad (n=1)$$

$$\text{age}(n) = \text{age}(n-1) + 2 \quad (n > 1)$$

# 函数的递归调用



- 编写程序

```
#include <stdio.h>

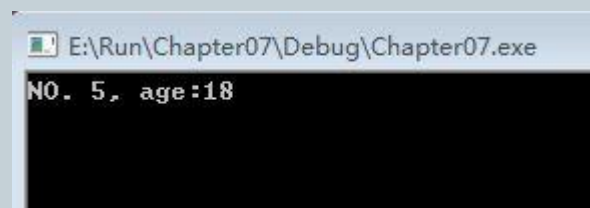
int main()
{
    int age(int n);
    printf("NO. 5, age:%d\n", age(5));

    return 0;
}

int age(int n)
{
    int c;

    if (n == 1)
        c = 10;
    else
        c = age(n-1) + 2;

    return(c);
}
```



# 函数的递归调用



- 例子：用递归方法求n!

- 解题思路：从1开始，乘2，再乘3.....一直乘到n

$$n! = \begin{cases} n! = 1 & (n=0, 1) \\ n \cdot (n-1)! & (n>1) \end{cases}$$

# 函数的递归调用



## ● 编写程序

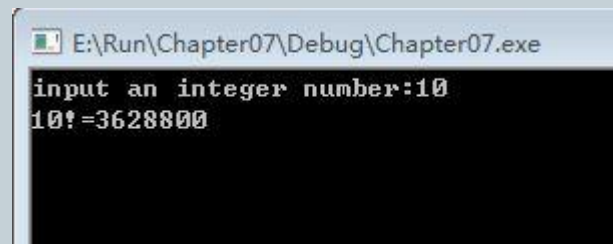
```
#include <stdio.h>

int main()
{
    int fac(int n);
    int n;
    int y;
    printf("input an integer number:");
    scanf("%d", &n);
    y = fac(n);
    printf("%d!=%d\n", n, y);

    return 0;
}

int fac(int n)
{
    int f;
    if (n < 0)
        printf("n < 0, data error!");
    else if (n == 0 || n == 1)
        f = 1;
    else
        f = fac(n-1)*n;

    return(f);
}
```





# 数组作为函数参数



- 数组元素可以用作函数实参，其用法与变量相同，向形参传递数组元素的值
- 数组名也可以作实参和形参，传递的是数组第一个元素的地址

# 数组元素作函数实参



- 数组元素可以用作函数实参，不能用作形参
- 值传递：从实参传到形参，单向传递

# 数组元素作函数实参



- 一个例子：输入10个数，要求输出其中值最大的元素和该数是第几个数
  - 解题思路：定义一个数组，用来存放10个数。设计一个函数，用来求两个数中的大者。定义一个变量m，初值为a[0]，每次调用max函数后的返回值存放在m中

# 数组元素作函数实参



## ● 编写程序

```
#include <stdio.h>

int main()
{
    int max(int x, int y);
    int a[10], m, n, i;
    printf("enter 10 integer numbers:");
    for (i = 0; i < 10; i++)
        scanf("%d", &a[i]);

    printf("\n");

    for (i = 1, m = a[0], n = 0; i < 10; i++)
    {
        if (max(m, a[i]) > m)
        {
            m = max(m, a[i]);
            n = i;
        }
    }

    printf("The largest number is %d\nit is the %dth number.\n", m, n+1);
}

int max(int x, int y)
{
    return(x > y? x: y);
}
```

E:\Run\Chapter07\Debug\Chapter07.exe

```
enter 10 integer numbers:4 7 0 -3 4 34 67 -42 31 -76

The largest number is 67
it is the 7th number.
```

# 数组名作函数实参



- 数组元素作实参时，向形参变量传递的是数组元素的值
- 数组名作函数实参时，向形参(数组名或指针变量)传递的是数组首元素的地址

# 数组名作函数实参



- 一个例子：有一个一维数组score，内放10个学生成绩，求平均成绩
  - 解题思路：定义一个average函数求平均成绩，不用数组元素作为函数实参，而是用数组名作为函数实参，形参也用数组名

# 数组名作函数实参



## ● 编写程序

```
#include <stdio.h>

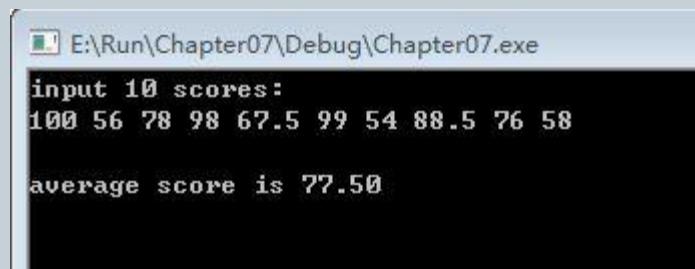
int main()
{
    float average(float array[10]);
    float score[10], aver;
    int i;
    printf("input 10 scores:\n");
    for (i = 0; i < 10; i++)
        scanf("%f", &score[i]);

    printf("\n");
    aver = average(score);
    printf("average score is %5.2f\n", aver);

    return 0;
}

float average(float array[10])
{
    int i;
    float aver, sum = array[0];
    for (i = 1; i < 10; i++)
        sum = sum + array[i];

    aver = sum/10;
    return(aver);
}
```



```
E:\Run\Chapter07\Debug\Chapter07.exe
input 10 scores:
100 56 78 98 67.5 99 54 88.5 76 58

average score is 77.50
```

# 数组名作函数实参



- 一个例子：有两个班级，分别有35名和30名学生，调用一个**average**函数，分别求这两个班的学生们的平均成绩
- 解题思路：定义一个**average**函数不指定数组的长度，在形参表中增加一个整型变量**i**，从主函数把数组的实际长度分别从实参传递给形参**i**



# 数组名作函数实参



## ● 编写程序

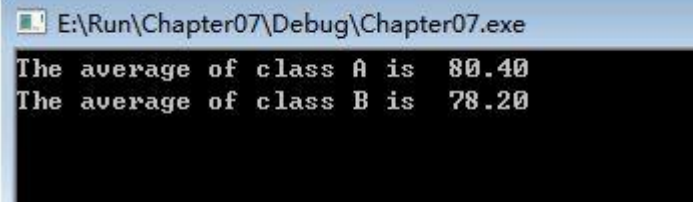
```
#include <stdio.h>

int main()
{
    float average(float array[], int n);
    float score1[5] = {98.5, 97, 91.5, 60, 55};
    float score2[10] = {67.5, 89.5, 99, 69.5, 77, 89.5, 76.5, 54, 60, 99.5};
    printf("The average of class A is %.2f\n", average(score1, 5));
    printf("The average of class B is %.2f\n", average(score2, 10));
    return 0;
}

float average(float array[], int n)
{
    int i;
    float aver, sum = array[0];
    for (i = 1; i < n; i++)
        sum = sum + array[i];

    aver = sum / n;

    return (aver);
}
```



```
E:\Run\Chapter07\Debug\Chapter07.exe
The average of class A is 80.40
The average of class B is 78.20
```

# 数组名作函数实参



- 7.12:用选择法对数组中10个整数按由小到大排序
  - 解题思路：使用选择法，先将10个数中最小的数与a[0]对换，在将a[1]~a[9]中最小的数与a[1]对换.....每比较一轮，找出一个未经排序的数中最小的一个

# 数组名作函数实参

## ● 编写程序

```
#include <stdio.h>

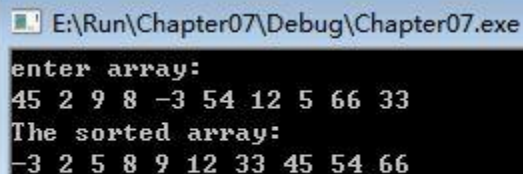
int main()
{
    void sort(int array[], int n);
    int a[10], i;
    printf("enter array:\n");
    for (i = 0; i < 10; i++)
        scanf("%d", &a[i]);
    sort(a, 10);
    printf("The sorted array:\n");
    for (i = 0; i < 10; i++)
        printf("%d ", a[i]);

    printf("\n");

    return 0;
}

void sort(int array[], int n)
{
    int i, j, k, t;
    for (i = 0; i < n-1; i++)
    {
        k = i;
        for (j = i+1; j < n; j++)
            if (array[j] < array[k])
                k = j;

        t = array[k];
        array[k] = array[i];
        array[i] = t;
    }
}
```



```
E:\Run\Chapter07\Debug\Chapter07.exe
enter array:
45 2 9 8 -3 54 12 5 66 33
The sorted array:
-3 2 5 8 9 12 33 45 54 66
```

# 多维数组作函数实参



- 多维数组名作为函数的实参和形参，在被调用函数中对形参数组定义时可以指定每一维的大小，也可以省略第一维的大小说明。例如，
  - `int array[3][10];` 或
  - `int array[][10];`
- 但是不能把第2维以及其他高维的大小说明省略，如
  - `int array[][];`
- 不能只指定第1维而省略第2维，如
  - `int array[3][];`

# 多维数组作函数实参



- 一个例子：有一个 $3*4$ 的矩阵，求所有元素中的最大值
  - 解题思路：先使变量`max`的初值等于矩阵中第1个元素的值，然后将矩阵中各个元素的值与`max`相比

# 多维数组作函数实参



## • 编写程序

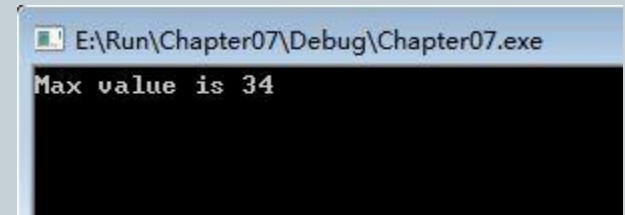
```
#include <stdio.h>

int main()
{
    int max_value(int array[][4]);
    int a[3][4] = {{1, 3, 5, 7}, {2, 4, 6, 8}, {15, 17, 34, 12}};

    printf("Max value is %d\n", max_value(a));
    return 0;
}

int max_value(int array[][4])
{
    int i, j, max;
    max = array[0][0];
    for (i = 0; i < 3; i++)
        for (j = 0; j < 4; j++)
            if (array[i][j] > max) max = array[i][j];

    return (max);
}
```



# 局部变量和全局变量



# 局部变量



- 每一个变量都有一个作用域问题，即它们在什么范围内有效
- 定义变量的3种情况：
  - 在函数的开头定义
  - 在函数内的复合语句内定义
  - 在函数的外部定义



# 局部变量



- 在一个函数内部定义的变量只在本函数范围内有效
- 在复合语句内定义的变量只在本复合语句范围内有效

```
float f1(int a)
{
    int b,c;
    :
}
```

} a,b,c 有效

```
char f2(int x,int y)
{
    int i,j;
    :
}
```

} x,y,i,j 有效

```
int main()
{
    int m,n;
    :
    return 0;
}
```

} m,n 有效

```
int main ()
{
    int a,b;
    :
    {
        int c;
        c=a+b;
        :
    }
    :
}
```

} c 在此复合语句内有效

} a,b 在此范围内有效

# 全局变量



- 在函数内定义的变量是**局部变量**，而在函数之外定义的变量称为**外部变量**，外部变量是**全局变量**(也称全程变量)

```
int p=1,q=5;           //定义外部变量
float f1(int a)         //定义函数 f1
{
    int b,c;           //定义局部变量
    :
}
char c1,c2;             //定义外部变量
char f2 (int x, int y)  //定义函数 f2
{
    int i,j;
    :
}
int main()              //主函数
{
    int m,n;
    :
    return 0;
}
```

全局变量 p,q 的作用范围

全局变量 c1,c2 的作用范围

# 全局变量



- 建议不在必要时不要使用全局变量
  - 全局变量在程序的全部执行过程中都占用存储单元，而不是仅在需要时才开辟单元
  - 它使函数的通用性降低了
  - 使用全局变量过多，会降低程序的清晰性

# 全局变量



- 一个例子：有一个一维数组，内放10个学生成绩，写一个函数，当主函数调用此函数后，能求出平均分、最高分和最低分
  - 解题思路：利用全局变量。调用一个函数可以得到一个函数返回值，希望通过函数调用能得到3个结果。

# 全局变量



## ● 编写程序

```
#include <stdio.h>

float Max = 0, Min = 0;

int main()
{
    float average(float array[], int n);
    float ave, score[10];
    int i;
    printf("Please enter 10 scores:");
    for (i = 0; i < 10; i++)
        scanf("%f", &score[i]);

    ave = average(score, 10);
    printf("max = %6.2f\nmin=%6.2f\naverage=%6.2f\n", Max, Min, ave);

    return 0;
}

float average(float array[], int n)
{
    int i;
    float aver, sum = array[0];
    Max = Min = array[0];

    for (i = 1; i < n; i++)
    {
        if (array[i] > Max) Max = array[i];
        else if (array[i] < Min) Min = array[i];
        sum = sum + array[i];
    }

    aver = sum/n;
    return(aver);
}
```

E:\Run\Chapter07\Debug\Chapter07.exe

```
Please enter 10 scores:89 95 87.5 100 67.5 97 59 84 73 90
max = 100.00
min= 59.00
average= 84.20
```

# 全局变量



- 如果在同一个源文件中，全局变量与局部变量同名
  - 出错
  - 局部变量无效，全局变量有效
  - 在局部变量的作用范围内，局部变量有效，全局变量被“屏蔽” ✓

# 全局变量



- 一个例子：若外部变量与局部变量同名

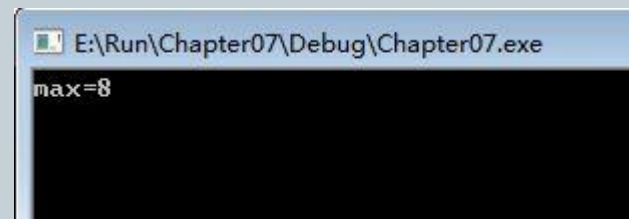
```
#include <stdio.h>

int a = 3, b = 5;

int main()
{
    int max(int a, int b);
    int a = 8;
    printf("max=%d\n", max(a, b));

    return 0;
}

int max(int a, int b)
{
    int c;
    c = a > b ? a : b;
    return (c);
}
```



# 变量的存储方式和生存期





# 变量的存储方式和生存期

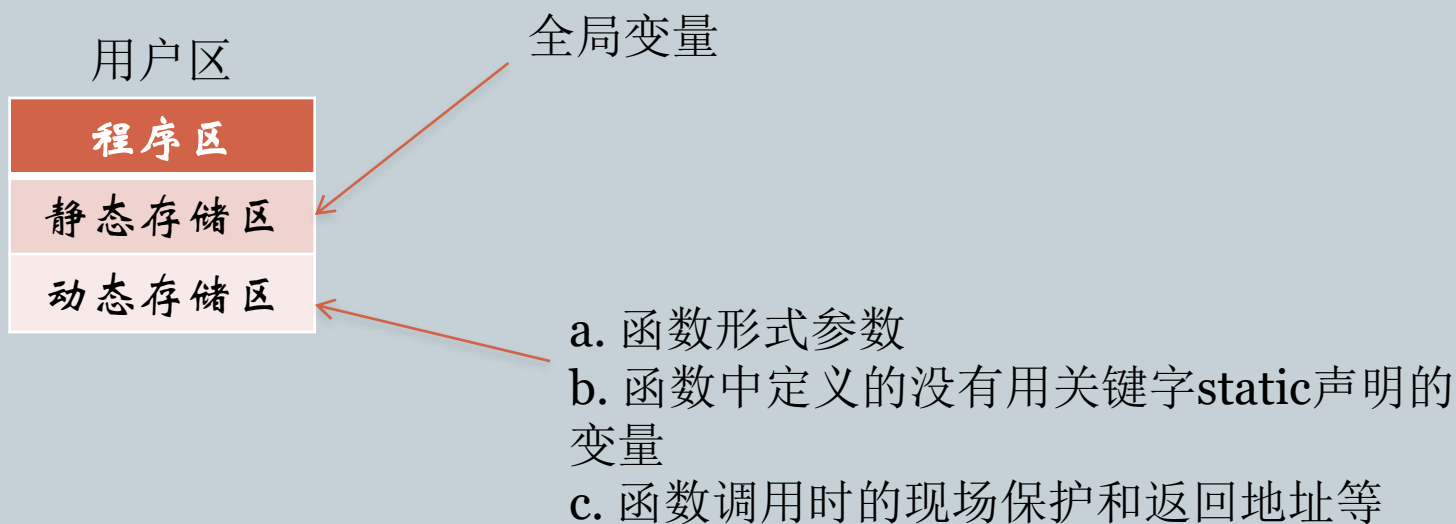


- 有的变量在程序运行的整个过程都是存在的，而有的变量则是在调用其所在的函数时才临时分配存储单元，而在函数调用结束后该存储单元就马上释放了，变量不存在了
  - 静态存储方式
  - 动态存储方式

# 变量的存储方式



- 在C语言中，每一个变量和函数都有两个属性
  - 数据类型和数据的存储类别



# 变量的存储方式



- C的存储类别包括
  - 自动的(auto)
  - 静态的(static)
  - 寄存器的(register)
  - 外部的(extern)

# 局部变量的存储类别



- 自动变量(auto变量)

- 函数中的局部变量

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

- 实际上，关键字“auto”可以省略，不写auto则隐含指定为“自动存储类别”

# 局部变量的存储类别



- 静态局部变量(**static**局部变量)
  - 函数中的局部变量的值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
  - 用关键字**static**进行声明

# 局部变量的存储类别



- 一个例子：考察静态局部变量的值

# 局部变量的存储类别



- 一个例子：输出1到5的阶乘值

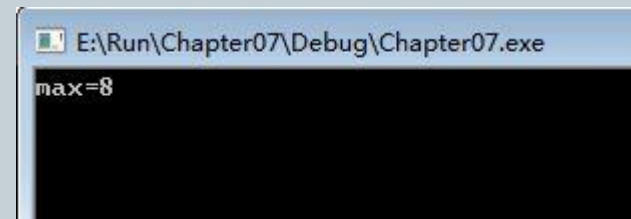
```
#include <stdio.h>

int main()
{
    int fac(int n);
    int i;
    for (i = 1; i <= 5; i++)
        printf("%d != %d\n", i, fac(i));

    return 0;
}

int fac(int n)
{
    static int f = 1;
    f = f*n;

    return(f);
}
```



# 局部变量的存储类别



- 寄存器变量(register变量)
  - 为提高执行效率，允许将局部变量的值放在CPU中的寄存器中，需要用时直接从寄存器取出参加运算，不必再到内存中去存取
  - `register int f;`



# 全局变量的存储类别



- **外部变量**是在函数的外部定义的**全局变量**
  - 作用域从变量的定义处开始，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

# 外部变量



- 在一个文件内扩展外部变量的作用域
  - 在定义点之前的函数需要引用该外部变量，则应该在引用之前用关键字**extern**对该变量作“外部变量声明”
  - 提倡将外部变量的定义放在引用它的所有函数之前，这样可以避免在函数中多加一个**extern**声明

# 外部变量



- 一个例子：调用函数，求3个整数中的大者

```
#include <stdio.h>

int main()
{
    int max();

    extern int A, B, C;
    printf("Please enter three integer numbers:");
    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);
}
```

```
E:\Run\Chapter07\Debug\Chapter07.exe
Please enter three integer numbers:34 67 12
max is 67
```

# 外部变量



- 如果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**，否则在进行程序的连接时会出现“重复定义”的错误。
- 在任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”，即“**extern Num**”

# 外部变量



- 一个例子：给定**b**的值，输入**a**和**m**，求 $a*b$ 和 $a^m$ 的值
  - 解题思路：
    - ✦ 分别编写两个文件模块，其中文件**file1**包含主函数，另一个文件**file2**包含求 $a^m$ 的函数
    - ✦ 在**file1**文件中定义外部变量**A**，在**file2**中用**extern**声明外部变量**A**，把**A**的作用域扩展到**file2**文件中

# 外部变量



## • 编写程序

file1.c

```
#include <stdio.h>

int A;

int main()
{
    int power(int);
    int b = 3, c, d, m;
    printf("enter the number a and its power m:\n");
    scanf("%d, %d", &A, &m);
    c = A*b;
    printf("%d * %d=%d\n", A, b, c);
    d = power(m);
    printf("%d ** %d=%d\n", A, m, d);

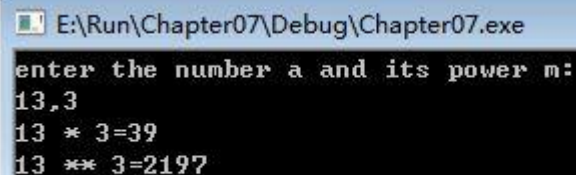
    return 0;
}
```

file2.c

```
extern A;

int power(int n)
{
    int i, y = 1;
    for (i = 1; i <= n; i++)
        y *= A;

    return(y);
}
```



```
E:\Run\Chapter07\Debug\Chapter07.exe
enter the number a and its power m:
13,3
13 * 3=39
13 ** 3=2197
```

# 外部变量



- 将外部变量的作用域限制在本文件中
  - 在程序设计中希望某些外部变量只限于被本文件引用，而不能被其他文件引用
  - 可以在定义外部变量时加一个**static**声明，称为静态外部变量

file1.c

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

file2.c

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



# 外部变量



- 对于局部变量，声明存储类型的作用是指定变量存储的区域(静态存储区或动态存储区)以及由此产生的生存期的问题
- 对于全局变量，都是在编译时分配内存的，都存放在静态存储区，声明存储类型的作用是变量作用域的扩展问题



# 外部变量



- 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，其所分配的空间始终存在
- 对全局变量用**static**声明，则该变量的作用域只限于本文件模块(即被声明的文件中)

# 存储类别小结

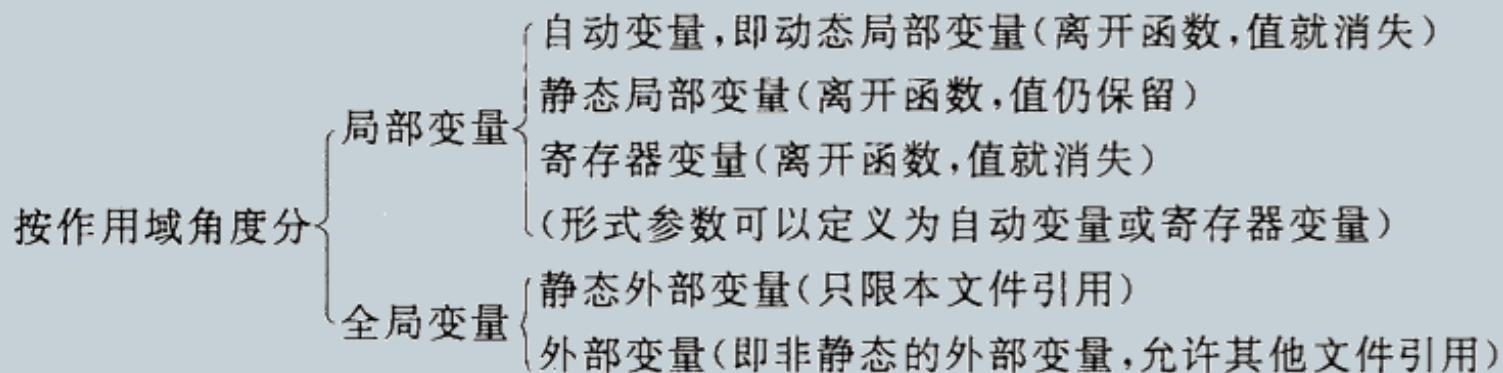


- 对一个数据定义，需要指定两种属性，分别使用两个关键字
  - 数据类型
  - 存储类别
- 例如
  - `static int a;`
  - `auto char c;`
  - `register int d;`

# 存储类别小结



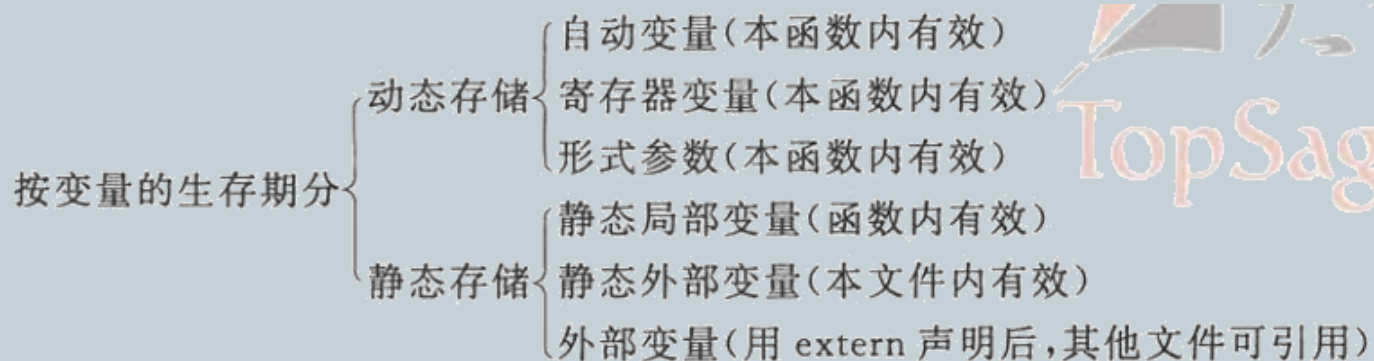
- 从作用域角度分，有局部变量和全局变量



# 存储类别小结



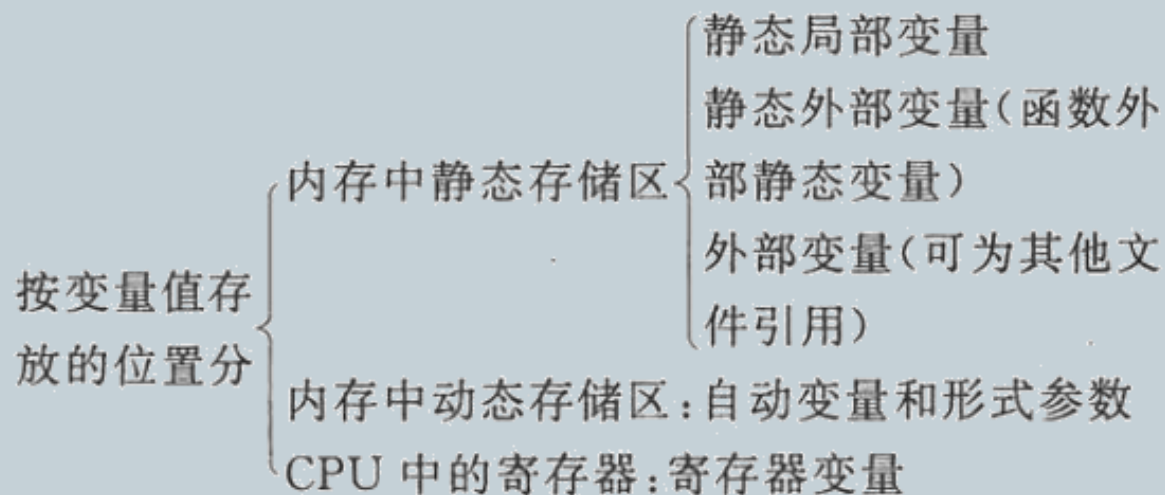
- 从变量存在的时间(生存期)来区分, 有动态存储和静态存储两种类型



# 存储类别小结



- 从变量值存放的位置来区分，可分为：



# 存储类别小结

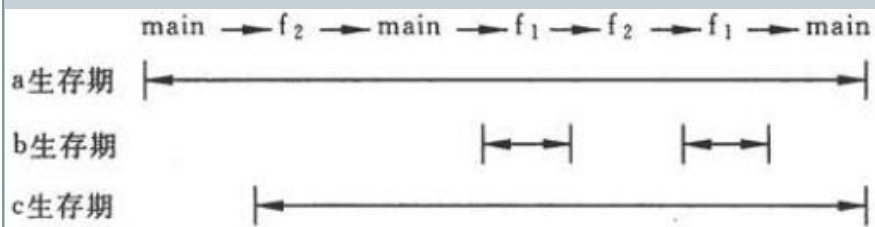


- 作用域和生存期

- 变量的**作用域**
- 变量值存在时间的长短，即**生存期**

文件file1.c

```
int a;  
int main( )  
{  
    :  
    f2( );  
    :  
    f1( );  
}  
void f1( )  
{  
    auto int b;  
    :  
    f2( );  
    :  
}  
void f2( )  
{  
    static int c;  
    :  
}
```



# 变量的声明和定义



- 在声明部分出现的变量
  - 定义性声明：需要建立存储空间
  - 引用性声明：不需要建立存储空间
- 比如
  - `int a;` 既是声明，又是定义
  - `extern a;` 是声明，而不是定义
- 建立存储空间的声明称定义
- 不需要建立存储空间的声明称为声明

# 变量的声明和定义



- 外部变量定义和外部变量声明的含义是不同的
  - 外部变量的定义只能有一次，它的位置在所有函数之外
  - 在同一文件中，可以有多次对同一外部变量的声明
  - 对外部变量的初始化只能在“定义”时进行，而不能在“声明”中进行
- 在函数中出现的对变量的声明(除了用extern声明的以外)都是定义



# 内部函数和外部函数



- 根据函数能否被其他源文件调用，将函数区分为内部函数和外部函数
- 如果一个函数只能被本文件中其他函数所调用，称为**内部函数**，又称**静态函数**
  - `static 类型名 函数名(形参表);`
  - `static int fun(int a, int b);`
- 如果函数可以为其他文件调用，称为**外部函数**
  - `extern 类型名 函数名(形参表)`
  - `extern int fun(int a, int b)`

# 内部函数和外部函数



- 使用内部函数，可以使函数的作用域只局限于所在文件
  - 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
- 外部函数可以为其他文件调用
  - C语言规定，如果在定义函数时省略，则默认为外部函数
  - 在需要调用外部函数的文件中，需要对此函数作声明

# 内部函数和外部函数



- 一个例子：有一个字符串，内有若干个字符，现输入一个字符，要求程序将字符串中该字符删去
- 解题思路：定义3个函数用来输入字符串、删除字符、输出字符串

# 内部函数和外部函数



## • 编写程序

```
#include <stdio.h>

int main()
{
    extern void enter_string(char str[]);
    extern void delete_string(char str[], char ch);
    extern void print_string(char str[]);

    char c, str[80];
    enter_string(str);
    scanf("%c", &c);
    delete_string(str, c);
    print_string(str);
    return 0;
}
```

```
void enter_string(char str[80])
{
    gets(str);
}

void delete_string(char str[], char ch)
{
    int i, j;
    for (i=j=0; str[i] != '\0'; i++)
        if (str[i] != ch)
            str[j++] = str[i];

    str[j] = '\0';
}

void print_string(char str[])
{
    printf("%s\n", str);
}
```

