

第7章 函数 (3)



复习回顾

➤ 上次课的内容：

◆ 习题课

◆ 汉诺塔

◆ 数组作为函数参数

◆ 感觉期中考考得如何？参照上图，你中枪了吗？



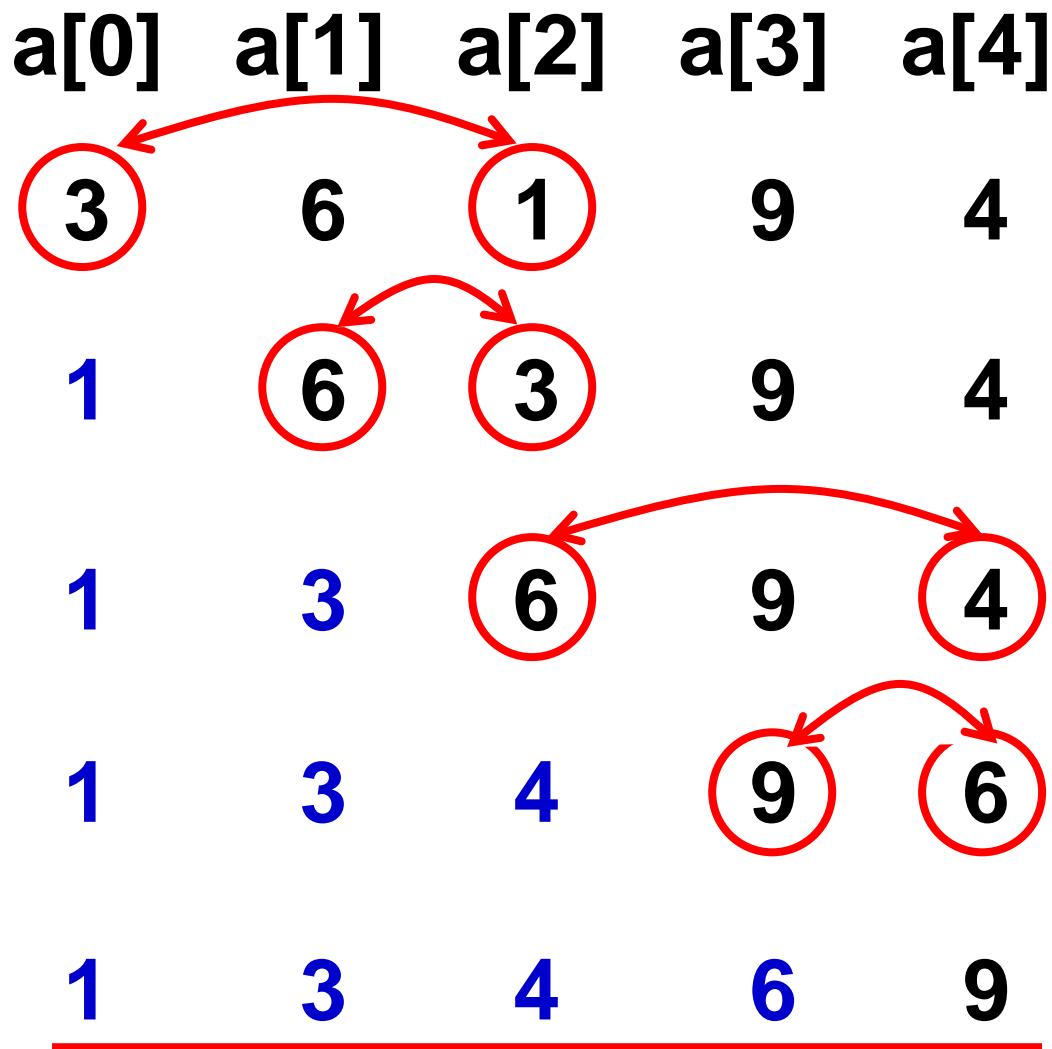
一维数组名做函数参数实例

用选择法对数组中10个整数按由小到大排序

➤ 解题思路：

◆ 所谓**选择法**就是先将10个数中最小的数与a[0]对换；再将a[1]到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

◆ 共比较9轮



小到大排序

一维数组名做函数参数实例

```
1. #include <stdio.h>
2. int main()
3. {
4.     void sort(int array[],int n);
5.     int a[10], i;
6.     printf("enter array:\n");
7.     for (i=0;i<10;i++)
8.         scanf("%d",&a[i]);
9.
10.    sort(a,10);
11.
12.    printf("The sorted array:\n");
13.    for (i=0;i<10;i++)
14.        printf("%d ",a[i]);
15.    printf("\n");
16.    return 0;
17.}
```

选择法排序的实现

```
1. void sort(int array[],int n)//调用方式sort(a,10);
2. {
3.     int i,j,k,t;
4.     for (i=0;i<n-1;i++)
5.     {
6.         k = i;
7.         for (j=i+1;j<n;j++)
8.         {
9.             if (array[j]<array[k])
10.                k = j;
11.         }
12.         t = array[k];
13.         array[k] = array[i];
14.         array[i] = t;
15.     }
16. }
```

在array[i+1]~array[n-1]中，
找到最小数，下标为k

多维数组名做函数参数

- 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一个元素的值，然后将矩阵中各个元素的值与max相比，每次比较后都把“大者”存放在max中，全部元素比较完后，max 的值就是所有元素的最大值

多维数组名做函数参数

```
1. #include <stdio.h>
2. int main()
3. {
4.     int max_value(int array[][4]);
5.
6.     int a[3][4]={ {1,3,5,7}, {2,4,6,8}, {15,17,34,12} };
7.     printf("Max value is %d\n", max_value(a));
8.
9.     return 0;
10. }
```

可以省略

不能省略
要与形参数组第二维大小相同

多维数组名做函数参数

```
1. int max_value(int array[][4])
2. {
3.     int i,j,max;
4.     max = array[0][0];
5.     for (i=0;i<3;i++)
6.     {
7.         for (j=0;j<4;j++)
8.         {
9.             if (array[i][j]>max)
10.                max = array[i][j];
11.         }
12.     }
13.     return (max);
14. }
```

要与实参数组第二维大小相同

变量的作用域

- 可以访问某个变量的代码区域就叫**变量的作用域**
- **C语言所有的变量都有自己的作用域**，作用域表明该变量有效的作用范围
- 按照作用域的不同，C语言中的变量可以分为**局部变量**和**全局变量**。

局部变量

```
float f1(int a)
{
    int b,c;    ....
}
char f2(int x, int y)
{
    int a,b;    ....
}
int main ( )
{
    int m,n,p;
    f1(p);
    f2(m,n).....
    return 0;
}
```

- 局部变量就是在函数内定义的变量，如f1的b，c；
- 形参变量是属于被调函数的局部变量，如f1的a，f2的x，y，而实参变量m，n，p是属于main的局部变量
- 允许在不同函数中使用相同的变量名，但它们代表不同的对象，系统会把它们分开存放，它们彼此之间也不会发生混淆。如f1和f2中都有a，b

复合语句中的局部变量

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

2. int main()
3. {
4.     int x=1;
5.     {
6.         int y = x;
7.     }
8.
9.     //注意：此处x换成y则编译错误！
10.    printf("%d",x);
11.
12.    return 0;
13.}
```

- C语言允许在复合语句中定义变量，其作用域只在复合语句范围内有效，如左例5-7行。
- Btw，主函数中定义的变量也只能在主函数中使用，不能在其他函数中使用。同样，主函数也不能使用其他函数中定义的变量。

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)
- 全局变量可以为**本文件中**其他函数所共用
 - ◆ 全局变量类似公有化，而局部变量类似私有化
- **有效范围**：从定义变量的位置开始到本源文件结束

全局变量实例

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

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

```
{
```

```
    int b,c;    .....
```

```
}
```

```
char c1,c2;
```

```
char f2(int x, int y)
```

```
{
```

```
    int i,j;    .....
```

```
}
```

```
int main ( )
```

```
{
```

```
    int m,n;
```

```
    .....
```

```
    return 0;
```

```
}
```

p、q、c1、c2
为全局变量

全局变量的有效范围

```
int p=1,q=5;  
float f1(int a)  
{  
    int b,c;    .....
```

p、q的有效范围

```
char c1,c2;  
char f2(int x, int y)  
{  
    int i,j;    .....
```

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

c1、c2的有效范围

全局变量应用例子

- 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**：调用一个函数可以得到一个函数返回值，现在希望通过**函数调用能得到3个结果**。可以利用全局变量来达到此目的

全局变量应用例子

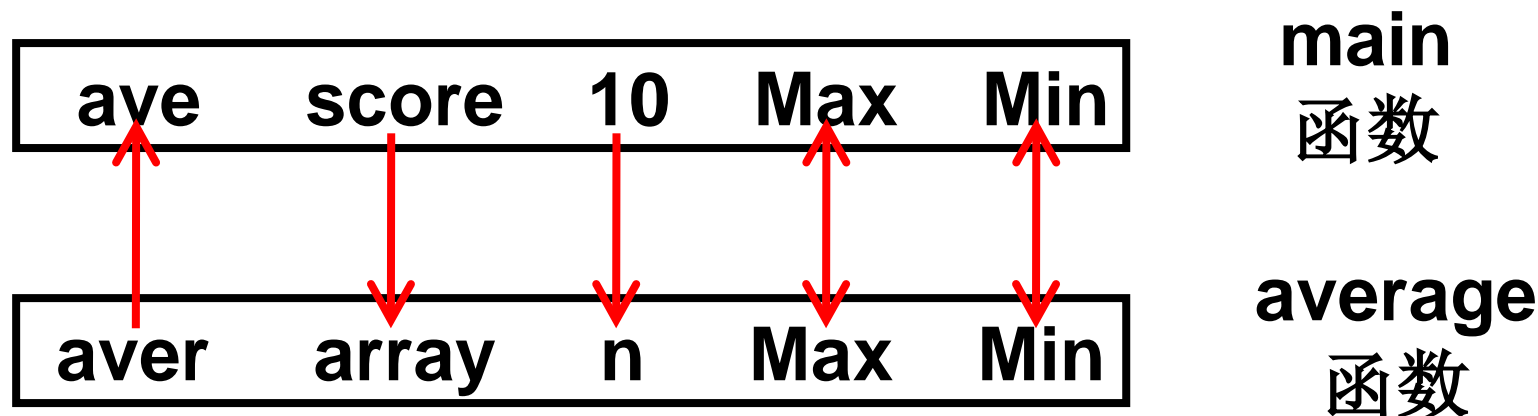
```
1. #include <stdio.h>
2. float Max=0,Min=0;
3. int main()
4. {
5.     float average(float array[],int n);
6.     float ave, score[10];
7.     int i;
8.     printf("Please enter 10 scores:\n");
9.     for (i=0;i<10;i++)
10.    {
11.        scanf("%f",&score[i]);
12.    }
13.    ave = average(score,10);
14.    printf("max=%6.2f\nmin=%6.2f\n\naverage=%6.2f\n",Max,Min,ave);
15.    return 0;
16. }
```

全局变量应用例子

```
1. float average(float array[],int n)
2. {
3.     int i;
4.     float aver,sum=array[0];
5.     Max = Min = array[0];
6.     for (i=1;i<n;i++)
7.     {
8.         if (array[i] > Max)
9.             Max = array[i];
10.        else if (array[i] < Min)
11.            Min = array[i];
12.        sum = sum+array[i];
13.    }
14.    aver = sum/n;
15.    return(aver);
16. }
```

```
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
```

全局变量为函数间交互提供便利



但是，**不在必要时，不要使用全局变量。**

(1) 全局变量破坏了函数的独立性，函数不再以独立的形式来完成各自的功能，**违背了结构化程序设计的思想**；

(2) 全局变量在程序执行过程中始终占用存储单元，**对内存是一种浪费。**

外部变量与局部变量同名

➤ 若外部变量与局部变量同名，分析结果。

```
1. #include <stdio.h>
2. int a=3, b=5;
3. int main()
4. {
5.     int max(int a,int b);
6.     int a=8;
7.     printf("max=%d\n",max(a,b));
8.     return 0;
9. }
10. int max(int a,int b)
11. {
12.     int c;
13.     c = a>b?a:b;
14.     return(c);
15. }
```

这里的**b**是第2行定义的全局变量

max=8

这里的**a**是第6行定义的局部变量，
仅在main函数内有效

外部变量与局部变量同名

➤ 若外部变量与局部变量同名，分析结果。

```
1. #include <stdio.h>
2. int a=3, b=5;
3. int main()
4. {
5.     int max(int a,int b);
6.     int a=8;
7.     printf("max=%d\n",max(a,b));
8.     return 0;
9. }
10. int max(int a,int b)
11. {
12.     int c;
13.     c = a>b?a:b;
14.     return(c);
15. }
```

结论：同名局部变量会在作用域内屏蔽全局变量的影响！

max=8

a、b为局部变量，仅在此函数内有效

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
 - ◆ 注意：extern声明的格式应该与定义变量的格式相容，数据类型可以省略，但不能不同

扩展外部变量作用域的例子

- 调用函数，求3个整数中的大者，要求使用extern关键字。
- **解题思路**：用extern声明外部变量，扩展外部变量在程序文件中的作用域。

扩展外部变量作用域的例子

```
1. #include <stdio.h>
2. int main()
3. {
4.     int max( );
5.     extern int A,B,C;
6.     scanf("%d %d %d",&A,&B,&C);
7.     printf("max is %d\n",max());
8.     return 0;
9. }
10. int A,B,C;
11. int max( )
12. {
13.     int m;
14.     m = (A>B)?A:B;
15.     if (C>m)
16.         m=C;
17.     return(m);
18. }
```

可以省略



```
34 67 12
max is 67
```


外部变量作用域扩展到其他文件

- 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
- 应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”
- 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

作用域扩展到其他文件的例子

- 给定b的值，输入a和m，求 $a * b$ 和 a^m 的值，要求用多文件实现。
- 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。

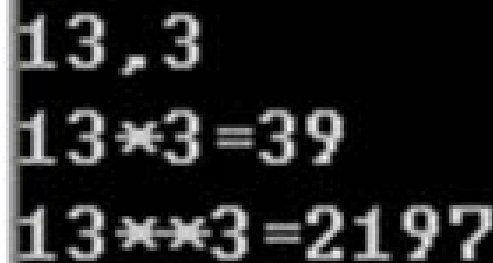
作用域扩展到其他文件的例子

文件file1.c :

```
#include <stdio.h>
int A;
int main()
{
    int power(int);
    int b=3,c,d,m;
    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;
}
```

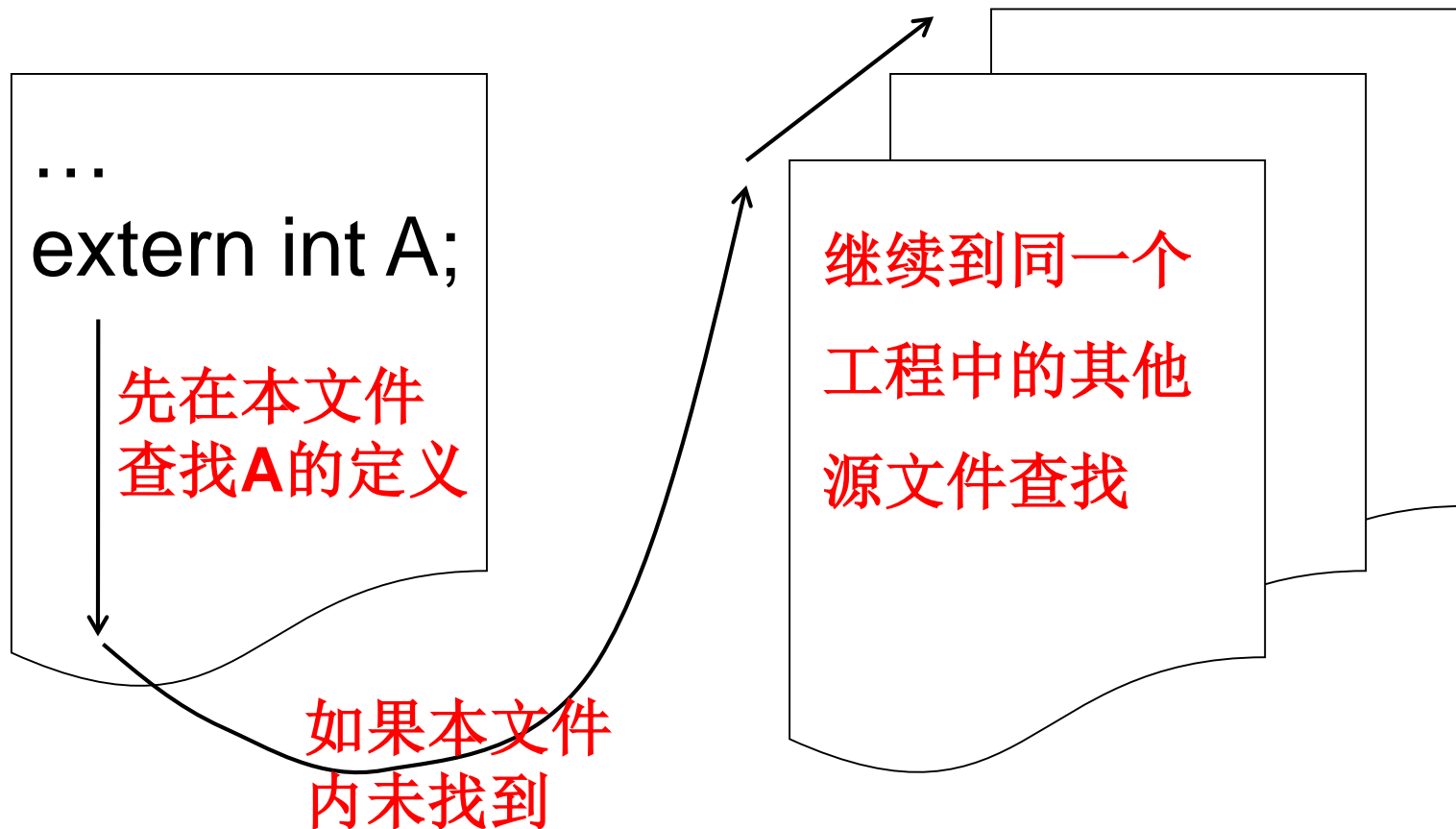
文件file2.c :

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



```
13,3
13*3=39
13**3=2197
```

编译器对extern的处理

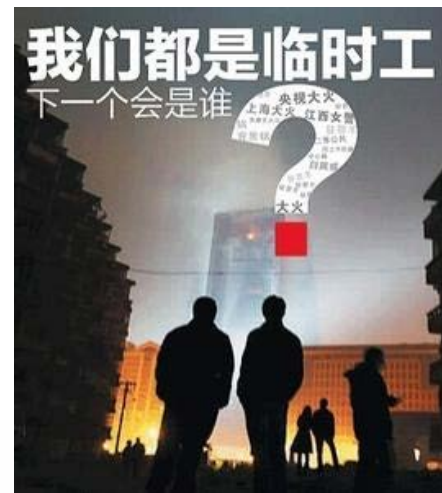


变量的静态/动态存储方式

➤ 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，也是全局变量的存储方式

◆ **动态存储方式**是在程序运行期间**根据需要进行临时性的动态的**分配存储空间的方式，也是局部变量的存储方式



内存：静态存储 vs. 动态存储

全局变量全部存放在静态存储区中

①函数形式参数
②函数中定义的没有用关键字static声明的变量
③函数调用时的现场保护和返回地址等存放在动态存储区

用户区

程序指令区

静态存储区

动态存储区

将数据存放在此区

程序开始执行时给全局变量分配存储区，程序执行完毕才释放。在程序执行过程中占据固定的存储单元

函数调用开始时分配，函数结束时释放。在程序执行过程中，这种分配和释放是动态的

变量的存储类别

- 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**
 - ◆ **数据类型**，说明变量的基本性质和占用内存的大小，如整型、浮点型等
 - ◆ **存储类别**，说明数据在内存中存储的方式，如静态存储和动态存储
 - ◆ C语言用四个关键字表示存储类别，包括：
自动的 (auto)、**静态的 (static)**、
寄存器的 (register)、**外部的 (extern)**

最低调的关键字 auto

➤ 自动变量(auto变量)

◆ 局部变量默认的存储类别，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

static 局部变量

- 如果希望函数中的**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，该变量保持已有值(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。

◆为什么用static？

◆也许参考自“生命在于静止”



静态变量的例子

➤ 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 int 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不是

静态变量的例子

➤ 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;
}
```

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

第一次调用开始

a	b	c
2	0	3

静态变量的例子

➤ 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;    a    b    c
```

```
}
```

2	1	4
---	---	---

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

第一次调用期间

静态变量的例子

➤ 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;
}
```

7

C

4

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

第一次调用期间

静态变量的例子

➤ 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;
}
```

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

第二次调用开始

a	b	c
2	0	4

静态变量的例子

➤ 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;
}
```

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

第二次调用期间

a	b	c
2	1	5

静态变量的例子

➤ 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;
}
```

8

C

5

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

第二次调用期间

静态变量的例子

➤ 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;
}
```

a	b	c
2	0	5

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

第三次调用开始

静态变量的例子

➤ 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;
}
```

a	b	c
2	1	6

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

第三次调用期间

静态变量的例子

➤ 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;
```

```
}
```

9

C

6

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

第三次调用期间

静态变量的例子

➤ 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;
}
```

整个程序结束

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

7
8
9

静态变量的例子

➤ 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;
}
```

在函数调用时赋初值

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

在编译时赋初值

静态变量的例子

➤ 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;
}
```

若不赋初值，不确定

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

若不赋初值，是0

静态变量的例子

➤ 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;
}
```

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

仅在本函数内有效

静态局部变量像买来的房子，
一般局部变量像租来的房子。

静态变量的例子

- 输出1到5的阶乘值，要求使用函数内部的静态变量存放阶乘值。
- **解题思路**：可以编一个函数用来进行连乘，如第1次调用时进行1乘1，第2次调用时再乘以2，第3次调用时再乘以3，依此规律进行下去。

静态变量的例子

```
#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);
}
```

若非必要，不要过多用静态局部变量

```
1!=1
2!=2
3!=6
4!=24
5!=120
```

静态变量小测验

➤ 下列代码的运行结果是什么？

```
#include <stdio.h>
static int i,j; int k = 0;
int fun1()
{
    static int i=0; i++; return i;
}
void fun2()
{
    j=0; j++;
}
int main()
{
    for (k=0; k<10; k++) { i=fun1(); fun2(); }
    printf("%d,%d", i, j);
    return 0;
}
```

10,1

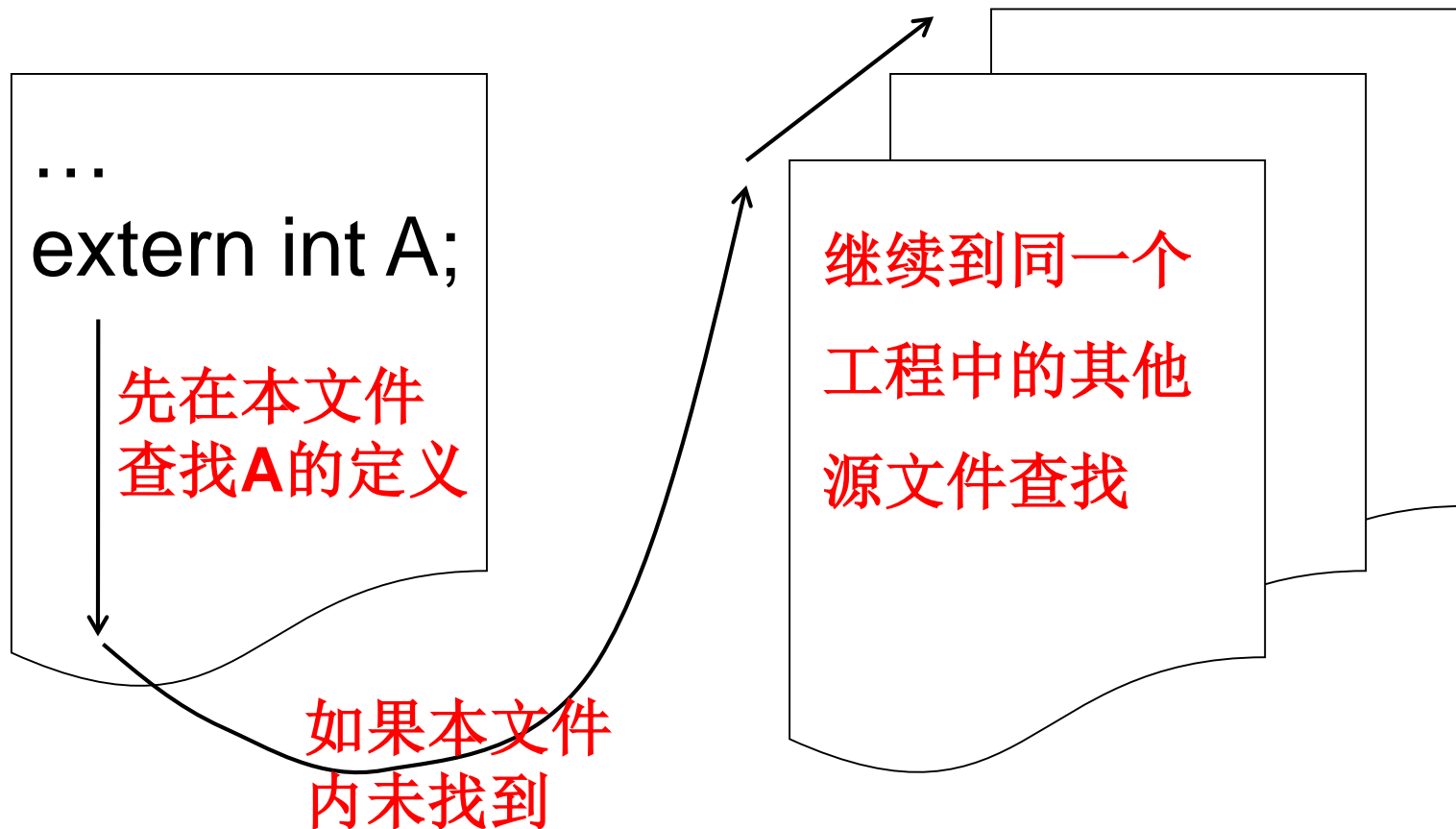
寄存器变量(register变量)

- 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（用关键字register修饰）允许将局部变量的值放在CPU中的寄存器中，以提高变量的访问速度。
- 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

【回顾】编译器对extern的处理



将变量的作用域限制在本文件

- 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;
    .....
}
```

将变量的作用域限制在本文件

➤ 补充说明:

- ◆ **不要误认为**对外部变量加static声明后才采取静态存储方式，而不加static的是采取动态存储
- ◆ 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

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

存储类别小结

➤ 对一个数据的定义，需要指定两种属性：

◆ **数据类型**和**存储类别**，分别使用两个关键字

例如：

static int a;

静态局部整型变量或静态
外部整型变量

auto char c;

自动变量，在函数内定义

register int d;

寄存器变量，在函数内定义

◆ 可以用**extern**声明已定义的外部变量

例如：**extern b;**

将已定义的外部变量**b**的作
用域扩展至此

存储类别小结

(1) 从**作用域角度**分，有**局部变量**和**全局变量**。

它们采用的存储类别如下：

形式参数可定义为
自动变量或寄存器
变量

按作用域角度分

局部变量

自动变量

静态局部变量

寄存器变量

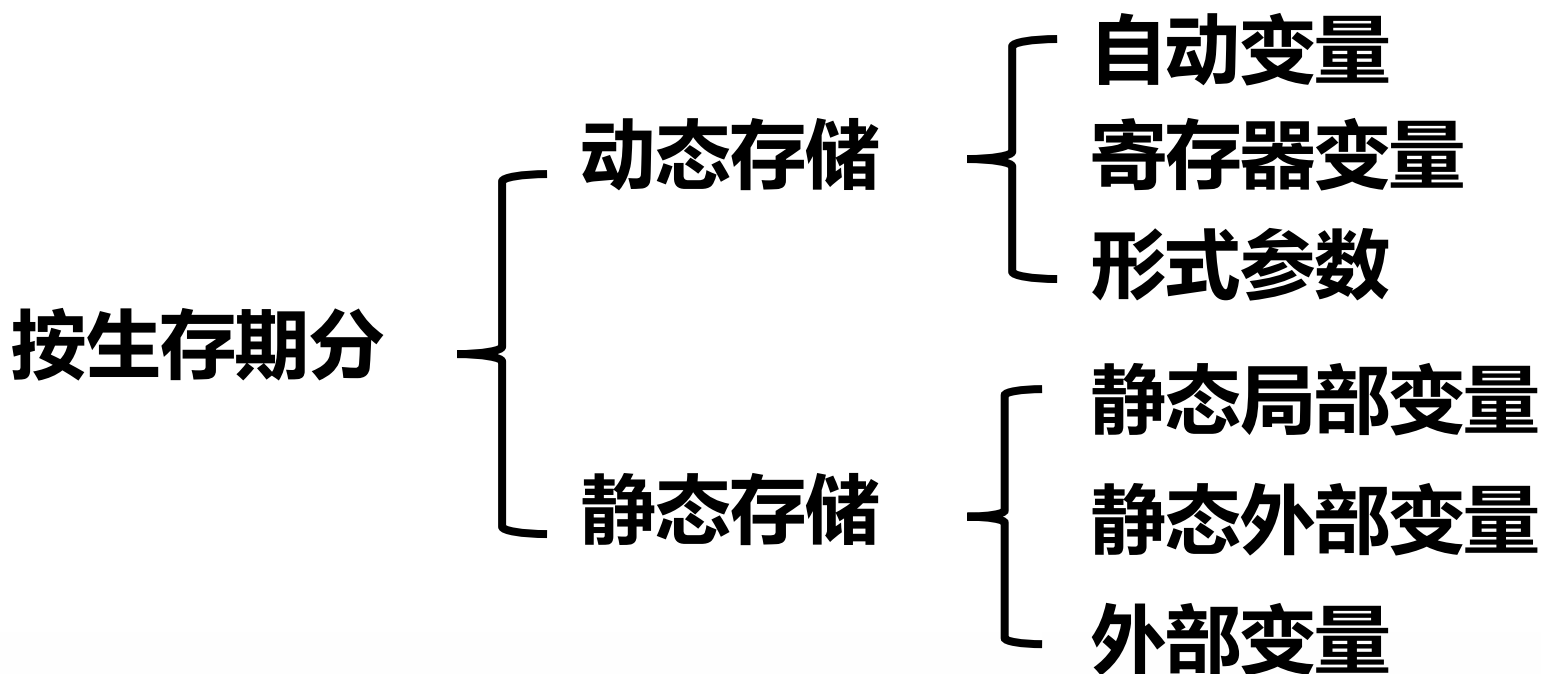
全局变量

静态外部变量

外部变量

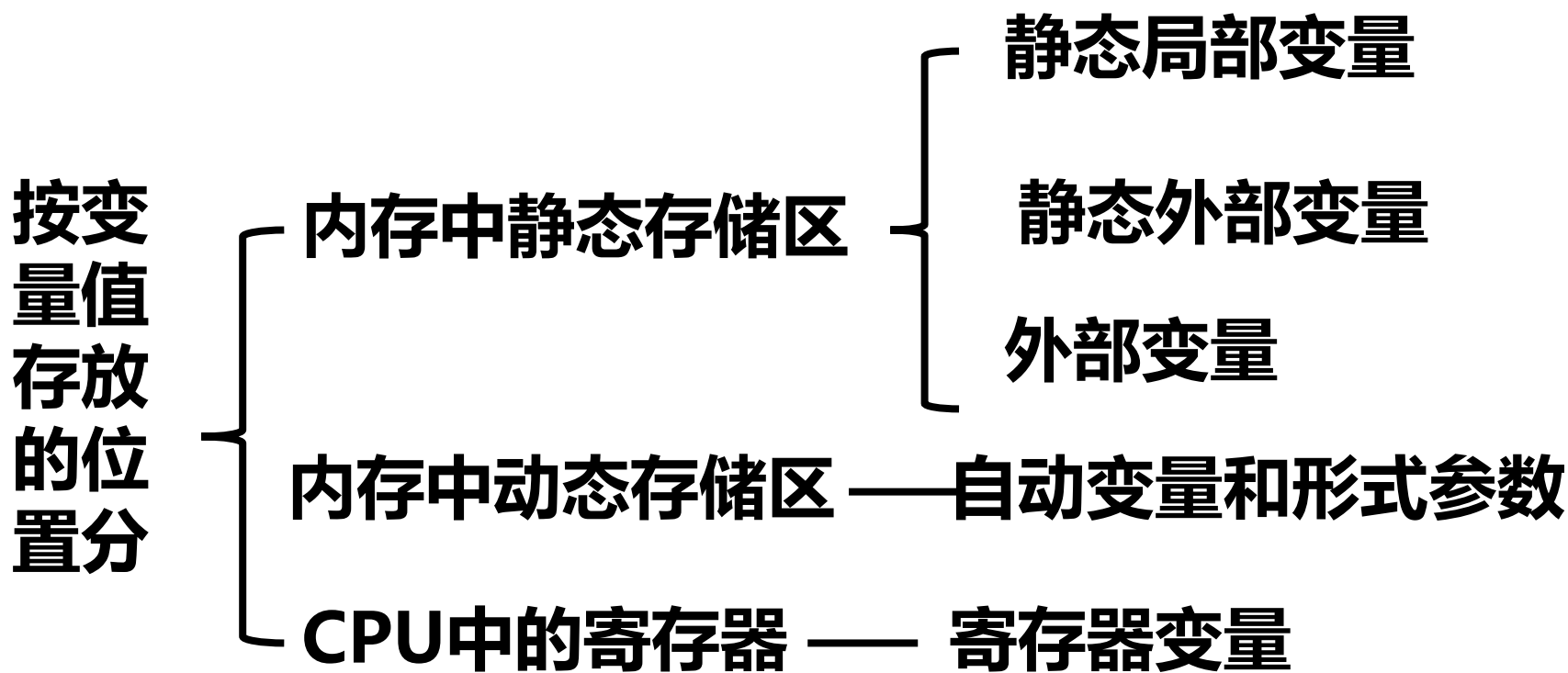
存储类别小结

(2) 从**变量存在的时间**区分,有**动态存储**和**静态存储**两种类型。静态存储是程序整个运行时间都存在,而动态存储则是在调用函数时临时分配单元



存储类别小结

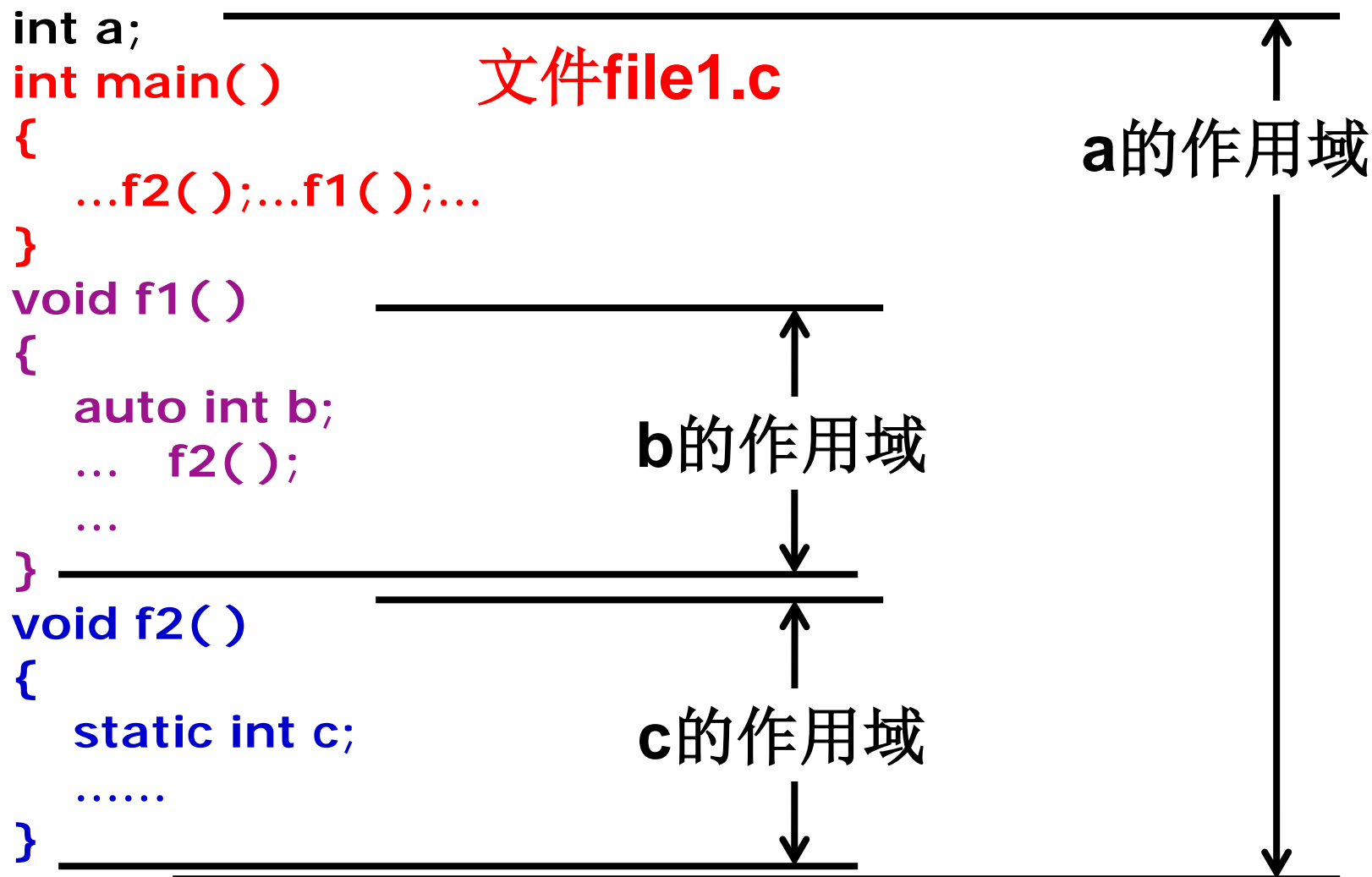
(3) 从**变量值存放的位置**来区分,可分为:



关于作用域和生存期的概念

- 对一个变量的属性可以从两个方面分析：
 - ◆ **作用域**：如果一个变量在某个文件或函数范围内是有效的，就称该范围为该变量的**作用域**
 - ◆ **生存期**：如果一个变量值在某一时刻是存在的，则认为这一时刻属于该变量的**生存期**
- 作用域来自**空间**的角度，生存期来自**时间**的角度
- **二者有联系但不是同一回事**

作用域示意图



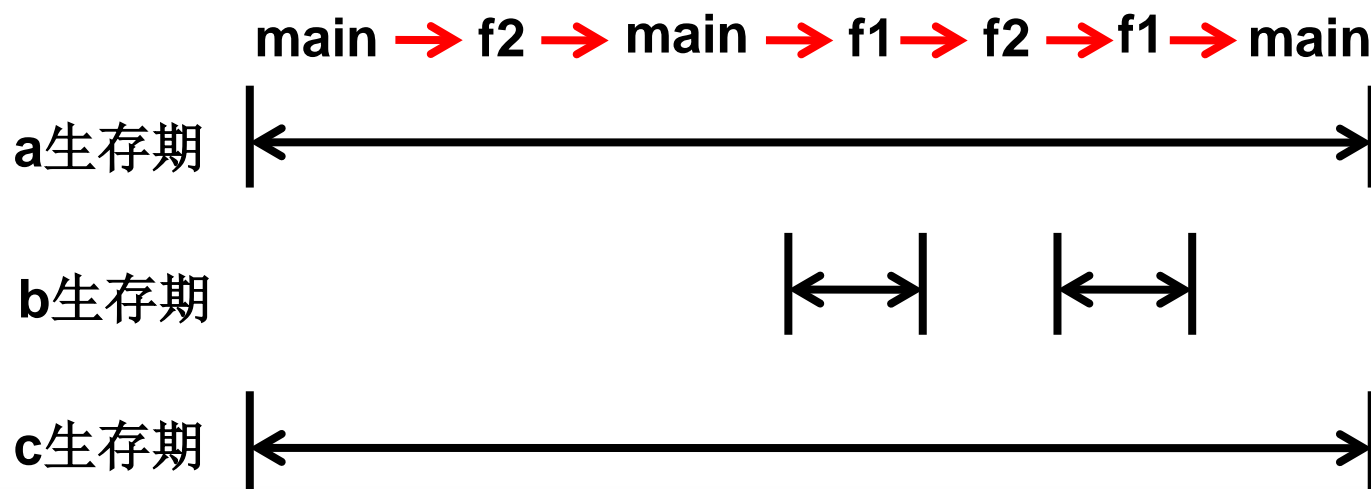
生存期示意图

```
int a;  
int main()  
{  
    ...f2();...f1();...  
}
```

```
void f1()  
{  
    auto int b;  
    ... f2();  
    ...  
}
```

```
void f2()  
{  
    static int c;  
    .....  
}
```

程序执行过程



各类存储类型的作用域和存在性

变量存储类别	函 数 内		函 数 外	
	作用域	存在性	作用域	存在性
自动变量和寄存器变量	√	√	×	×
静态局部变量	√	√	×	√
静态外部变量	√	√	√(只限本文件)	√
外部变量	√	√	√	√

static的不同作用

- static使**局部变量**由动态存储方式改变为静态存储方式
- static使**全局变量**使局部化(局部于本文件)，但仍为静态存储方式
- 从**作用域角度**看，凡有static声明的，其作用域都是局限的，或者是局限于本函数内(静态局部变量)，或者局限于本文件内(静态外部变量)

哪个是声明，哪个是定义？

A. `int i;`

B. `extern int i;`

- 所谓定义是指创建一个对象，为这个对象分配一块内存并取一个名字
- 所谓声明有两重含义
 - ◆ 告诉编译器，这个名字已经匹配到一块内存上了
 - ◆ 告诉编译器，这个名字已经被预定了

区分定义和声明

- 一般为了叙述方便，把**建立存储空间的变量声明称定义**，而把**不需要建立存储空间的声明称为声明**
- 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
- 在函数中对其他函数的声明不是函数的定义

关于声明和定义的例子

```
1. #include <stdio.h>
2. int main()
3. {
4.     int max();           //这是声明
5.     extern int x,y;      //这也是声明
6.     .....
7.     x = 0;               //链接错误
8.     y = 0;               //正确
9.     return 0;
10.}
11. int y;                  //这是定义
12. int max()               //这也是定义
13. {
14.     .....
15. }
```

关于声明和定义的例子

```
1. #include <stdio.h>
2. extern int x = 0;           // 正确，这是定义！
3. int main()
4. {
5.     int max();
6.     .....
7.     x = 1;                  // 正确
8.     y = 0;                  // 错误
9.     return 0;
10.}
11. int y;
12. int max()
13. {
14.     .....
15. }
```

对全局变量使用extern，可以初始化，而声明则变成了定义

关于声明和定义的例子

```
1. #include <stdio.h>
2. int main()
3. {
4.     int max();
5.     extern int x = 0; //错误，不能初始化
6.     extern int y;
7.     .....
8.     y = 0;
9.     return 0;
10.}
11. int y;
12. int max()
13. {
14.     .....
15. }
```

对局部变量使用extern，不可以初始化，此时声明还是声明

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)

内部函数的作用

- 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
- **作用**：提高了程序的可靠性

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**，可供其他文件调用。
- 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**

外部函数应用的例子

➤ 有一个字符串，内有若干个字符，今输入一个字符，要求程序将字符串中该字符删去。用外部函数实现。

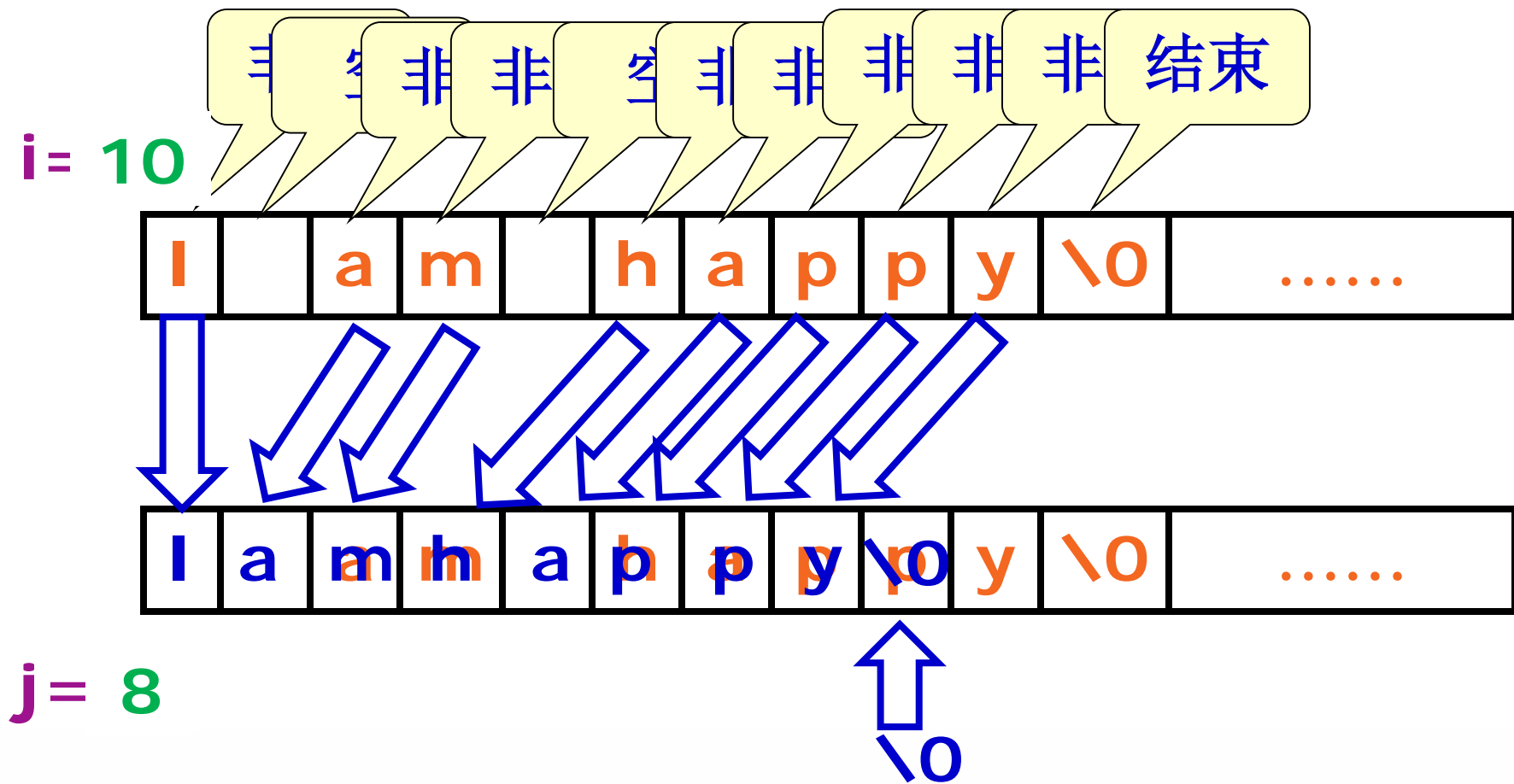
➤ **解题思路：**

◆ 分别定义3个函数用来输入字符串、删除字符、输出字符串

◆ 按题目要求把以上3个函数分别放在3个文件中。main函数在另一文件中，main函数调用以上3个函数，实现题目的要求

外部函数应用的例子

删除空格的思路



外部函数应用的例子

```
#include <stdio.h>
void enter_string(char str[80])
{
    gets(str);
}
```

file2.c (文件2)

```
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';
}
```

file3.c (文件3)

```
#include <stdio.h>
void print_string(char str[])
{
    printf("%s\n",str);
}
```

file4.c (文件4)

外部函数应用的例子

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

```
2. int main()
```

```
3. {
```

```
4.     extern void enter_string(char str[]);
```

```
5.     extern void delete_string(char str[], char ch);
```

```
6.     extern void print_string(char str[]);
```

```
7.
```

```
8.     char c, str[80];
```

```
9.
```

```
10.    enter_string(str);
```

```
11.    scanf("%c",&c);
```

```
12.    delete_string(str,c);
```

```
13.    print_string(str);
```

```
14.
```

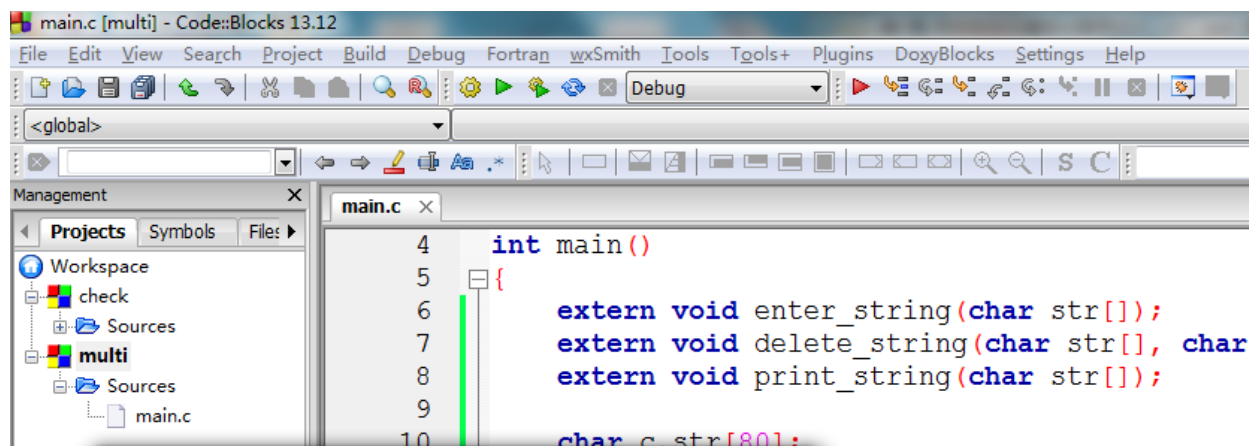
```
15.    return 0;
```

```
16. }
```

main.c (文件1)

声明在本函数中将要调用的已在其他文件中定义的3个函数

多文件程序实例 (1)



Please enter the file's location and name and whether to add it to the active project.

Filename with full path:

C:\code\multi\file3.c

☒ Add file to active project
In build target(s):

☒ Debug
☒ Release

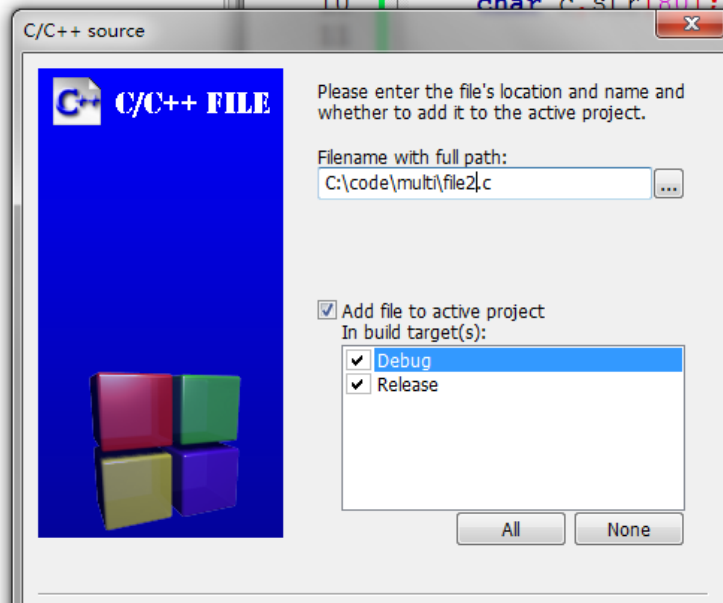
Please enter the file's location and name and whether to add it to the active project.

Filename with full path:

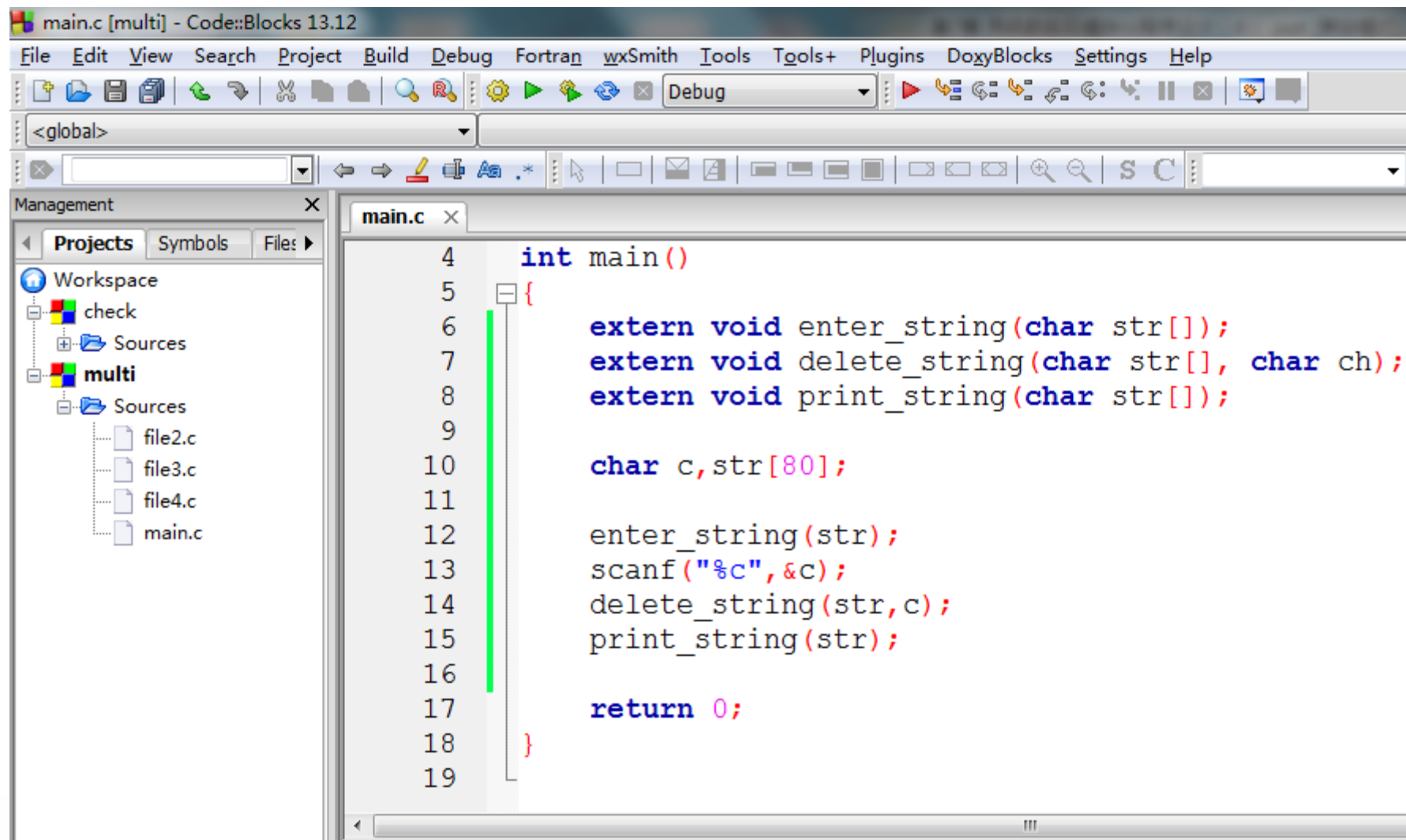
C:\code\multi\file4.c

☒ Add file to active project
In build target(s):

☒ Debug
☒ Release



多文件程序实例（2）



```
main.c [multi] - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
<global>
Management
Projects Symbols Files
Workspace
check
Sources
multi
Sources
file2.c
file3.c
file4.c
main.c

main.c
4 int main()
5 {
6     extern void enter_string(char str[]);
7     extern void delete_string(char str[], char ch);
8     extern void print_string(char str[]);
9
10    char c, str[80];
11
12    enter_string(str);
13    scanf("%c", &c);
14    delete_string(str, c);
15    print_string(str);
16
17    return 0;
18 }
19
```

作业 2017/11/29

➤ 按下列要求编写程序，提交手写源代码

1. 全民大博饼

- ① 参考上上节课提供的随机数生成代码，实现一个扔骰子的函数，要求使用数组作为函数参数，返回一次博饼的结果（如 {1,2,3,4,5,6}）
- ② 实现一个函数，判断生成的博饼结果的等级（包括落空，一秀，二举，三红，四进，对堂，状元，具体规则网上查）。
- ③ 在主函数调用上面两个函数，并输出本次博饼结果（包括骰子的面值和等级）。

2. 【选做】用递归实现123456789的全排列输出（提示：用数组名做递归函数参数）