第七章　作用域和存储属性

7.1 C程序的结构

7.2 作用域和作用域规则

7.3 存储属性和生存期

7.4 变量的初始化

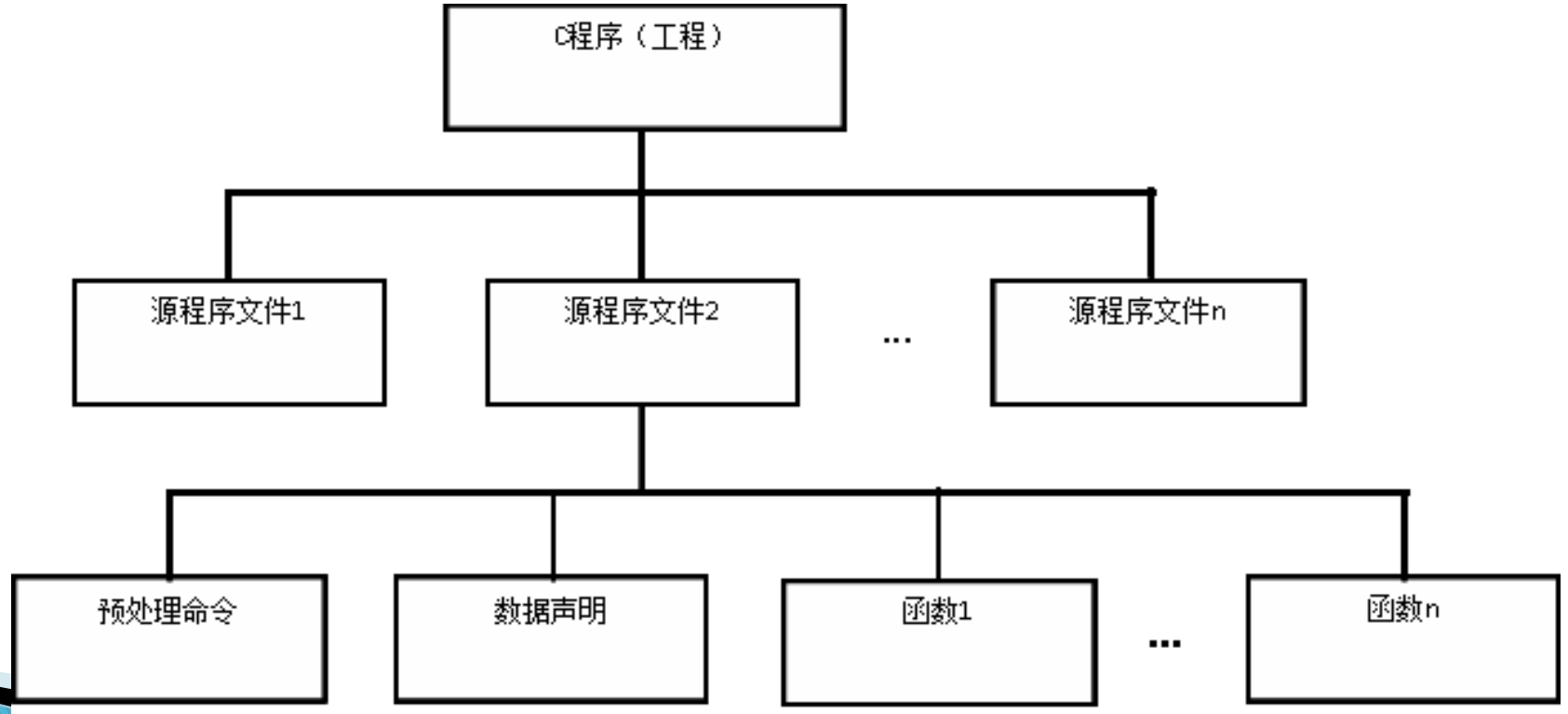
7.1 C程序的结构

7.1.1 模块的概念

- 模块是指为完成特定的任务而建立的相对独立的程序单元。
- 是一段连续的，相邻的程序序列。
- 被边界元素“{ }”限制在一定的范围内
- 有一个**标识符**从整体上代表这些程序语句序列。

模块	模块名	边界元素
函数	函数名	{ }
一段源程序	源文件名	同一个源文件

一个C语言程序结构，它由多个源程序模块（源文件）组成，一个源程序模块由一个或多个函数模块组成，程序从主函数模块开始执行，主函数模块调用其他函数模块，其他函数模块之间也可以相互调用，所有调用结束以后最终返回主函数模块。所以，一个可执行的C程序必须在某个源程序文件中，有且仅有一个主函数



7.1.2 模块与信息隐藏

模块化的目的是为了隐藏信息。对一个模块而言，若它能被程序中的其它模块调用，则我们说：对调用它的模块而言，该模块可见。若模块内部的某些信息（函数，变量等）不被隐藏，则这部分就可被其它程序所引用。

作用域规则，访问控制及连接属性可将一个模块的信息隐藏或开放，从而在C程序的模块之间实现数据共享。

7.2 作用域和作用域规则

作用域指的是程序正文中有效的那部分区域。根据对象定义语句位置的不同，作用域分为块作用域、函数作用域和文件作用域。（区别于生存期！）

7.2.1 块作用域和局部变量：

具有块作用域的对象是定义在一对{ }之内的。**块作用域**的范围**从程序中对象定义处到块结束处的“}”止。**具有块作用域的变量叫**局部变量**。

局部变量又叫内部变量，内部变量分为动态局部变量和静态局部变量。

```
#include <iostream>                          // 包含输入输出接口文件
using namespace std;                         // 引用标准命名空间
int main()
{
    char stuNo[50] = "";                     // 定义字符数组用来存放学号
    unsigned int i = 0 ;                     // 数组下标变量
    unsigned long number = 0;                // 存放转换后得到的整数
    cout << "请输入学生的学号:\t";
    cin >> stuNo;                            // 从键盘输入整型字符串到字符数组中
    while(stuNo[i] != '\0' && stuNo[i] >= '0' && stuNo[i] <= '9')
    {// 定义局部变量k存放字符stuNo[i]的ASCII码和字符'0'的ASCII码之差
        int k = stuNo[i] - 48;               // 字符'0'的ASCII码是48
        number = 10 * number + k;
        i++;
    }
    cout << "转换出的整数为: " << number <<endl;
    return 0;
}
```

块作用域。定义k为局部变量，仅在while循环内有效，超过这个范围，就不能使用此变量了，否则系统将会报错。

7.2.2 函数作用域和形参变量

- 对象的**函数作用域**是指对象在定义它的整个函数的范围内都有效。
- C语言中，函数的形参变量具有函数作用域。形参变量也是内部变量，因为其起作用的范围限制在一对{ }内。

```
double average(int score[], int n)
{                                                   // average函数开始
    int i = 0;                                      // 数组下标变量
    double ave = 0;                                 // 平均成绩
    int sum = 0;                                    // 成绩总和
    for(i = 0; i < n; i++)
    {
        sum = sum + score[i];                       // 累加求和
    }
    ave = sum / (double)n;                          // 求平均
    return ave;                                     // 将程序流程和平均值ave返回主调函数
}                                                   // average函数结束
```

形参变量score、n，局部变量i、ave和sum的作用域为average函数开始处到average函数结束处。

7.2.3 文件作用域和全局变量：

- 对象的**文件作用域**是指从对象的定义处到整个源文件模块结束处。**全局变量**是指定义在本源文件中所有函数之前的变量。其作用域从定义位置开始，直到程序结束。 C语言中，具有文件作用域的对象有全局变量和函数名（函数名就代表函数对象）

- 在7_3.cpp中定义了全局变量sum，其作用域为整个文件，因此，本文件的所有函数均可使用此变量。

```cpp
//.....
#include <iostream>                          // 包含输入输出接口
using namespace std;                         // 引用标准命名空间
int sum = 0;                                 // 定义全局变量sum
double average(int score[], int n);          // 申明average函数
double average(int score[], int n)
{
    int i = 0;                               // 数组下标变量
    double ave = 0;                          // 平均成绩
    sum = 0;                                 // 求和之前累加变量
    for(i = 0; i < n; i++)
    {
        sum = sum + score[i];                // 累加求和
    }
    ave = sum / (double)n;                   // 求平均
    return ave;                              // 流程和平均值返回主调函数
}
int main()
{
    const int STU_NUM = 5;                   // 定义整型常量代表学生人数
    int sco[STU_NUM] = {0};                  // 存放学生成绩的数组
    int i = 0;                               // 数组下标变量i
    double aver = 0;                         // 存放平均值
    cout << "输入" << STU_NUM << "个学生的成绩。" << endl;
    for(i = 0; i < STU_NUM; i++)
    {
        cin >> sco[i];
    }
    aver = average(sco, STU_NUM);
    cout << endl << "平均成绩为:" << aver << endl;
    cout << endl << "成绩总和为:" << sum << endl;
    return 0;
}
```

•定义了全局变量sum，其作用域为整个文件，因此，本文件的所有函数均可使用此变量。

文件作用域

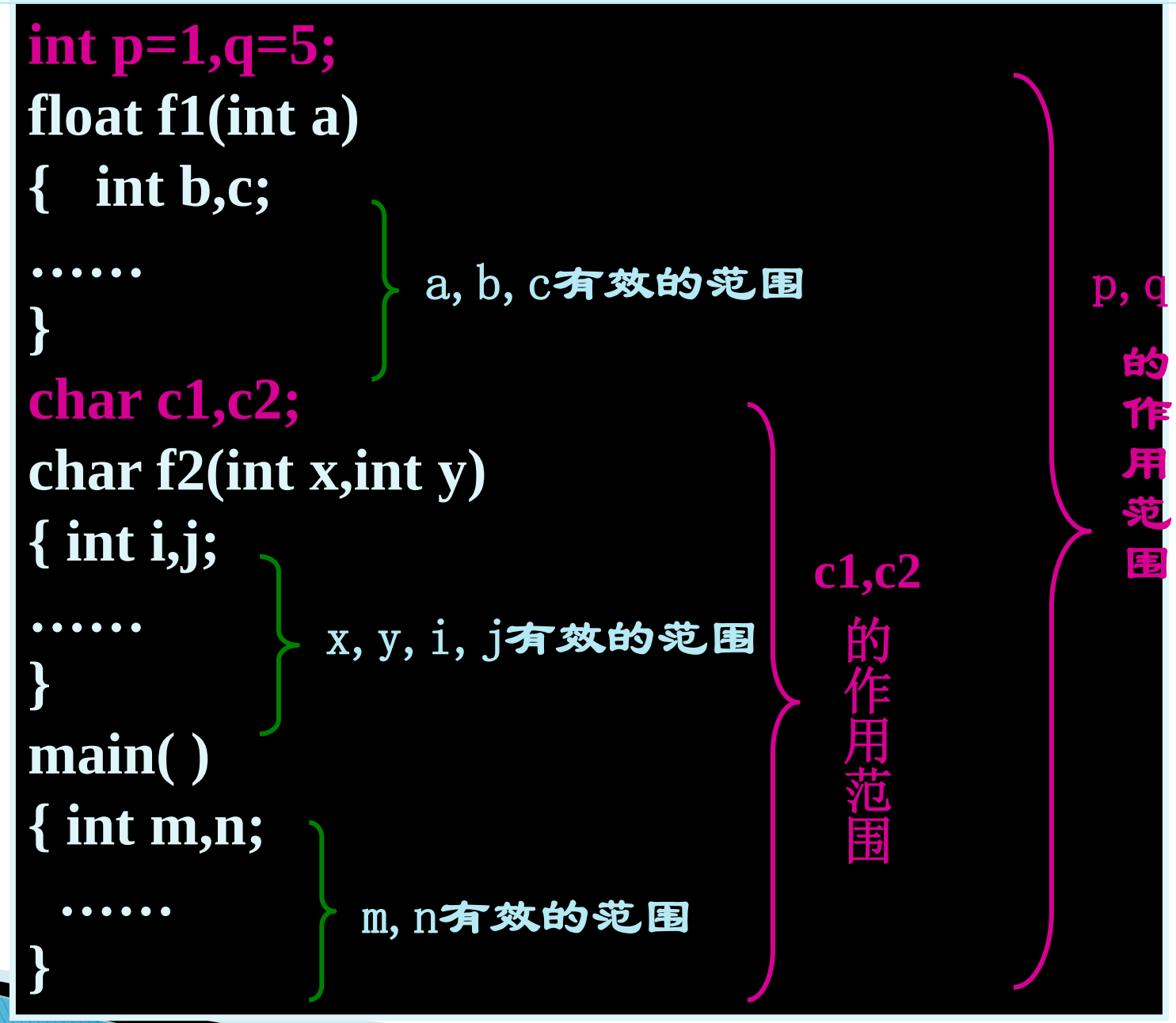
对全局变量的使用，需要做如下说明：

① 由于全局变量可以被同一源文件中的所有函数模块使用，因此全局变量提供了函数间除“实参—形参”相结合传送数据之外的另一种数据传送的渠道。通过全局变量可以共享多个数据。

② 只在必要的时候使用全局变量。

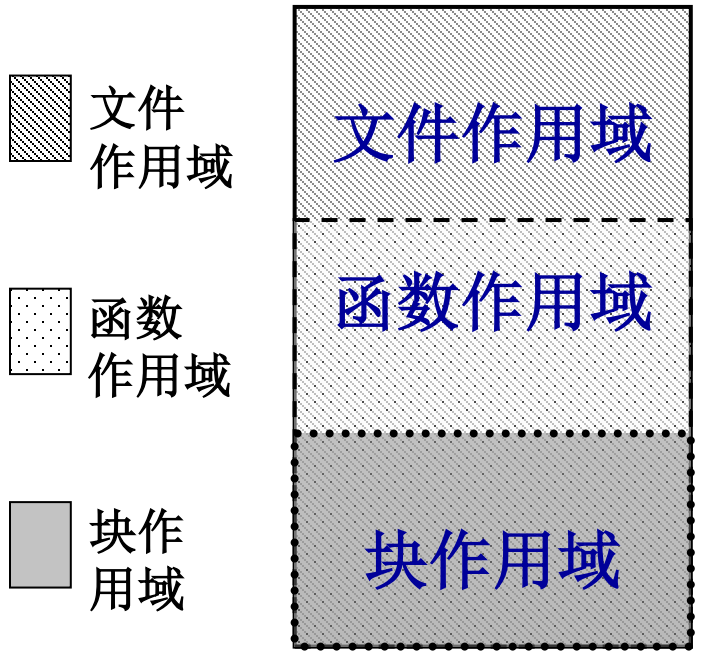
原因：占用内存；降低可移植性；降低可读性；程序出错可能性增加（因为每个函数都可能改变全局变量的值）

进行任务模块划分以及实现函数时，应把函数做成一个具有良好“接口”的封闭体，即只能通过实参 → 形参这个渠道传递信息给函数，函数的实现只和它的形参有关系。这样的程序可移植性好，可读性强。


int p=1,q=5;
float f1(int a)
{ int b,c;
……
}
a, b, c有效的范围
char c1,c2;
char f2(int x,int y)
{ int i,j;
……
}
x, y, i, j有效的范围
main( )
{ int m,n;
……
}
m, n有效的范围
c1,c2 的作用范围
p, q 的作用范围

7.2.4 作用域规则：

- 在一个源文件模块中，块作用域、函数作用域和文件作用域之间的包含关系如下图所示。



作用域包含关系

C语言的作用域需遵循以下规则：

① 内层作用域不能延伸到外层。

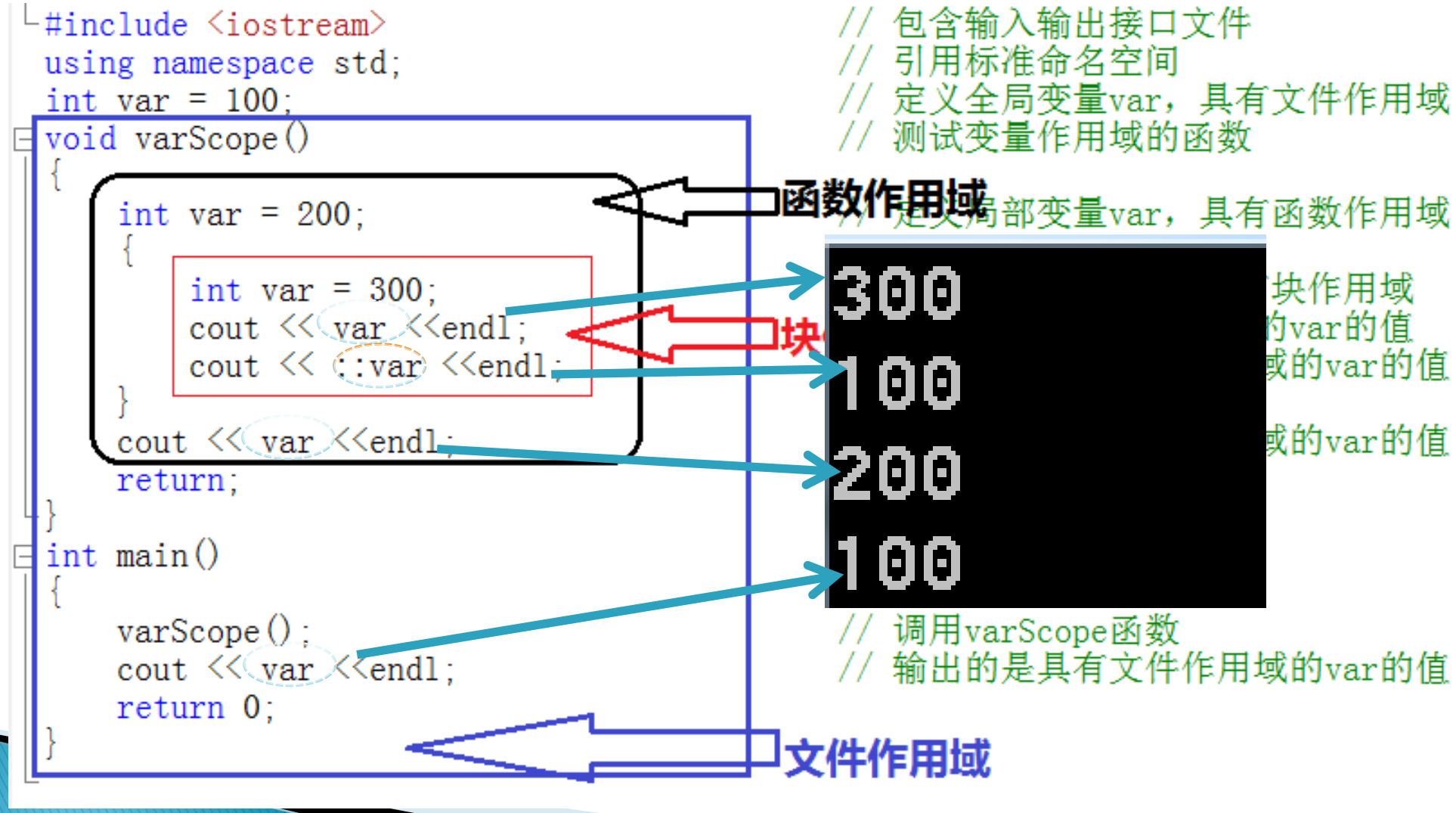
② 在同一源文件中，若内层和外层声明了同名标识符，则内层标识符屏蔽外层的同名标识符。

③ 在同一源文件中，如果内层想引用外层具有文件作用域的同名标识符（如同名的全局变量），在VS中可使用全局作用域运算符::。

④ 如果要在一个源文件模块中引用另一个源文件模块中定义的全局变量或在同一文件中引用其后定义的全局变量，可使用关键字extern在使用它的源文件中进行引用性声明。

extern　数据类型　被引用的变量名；

同一个源程序文件中，如果有局部变量和全局变量同名，则在该局部变量的作用范围内，全局变量被“屏蔽”，即不起作用。



7.2.4 作用域规则：

```
#include <iostream>                    // 包含输入输出接口文件
using namespace std;                   // 引用标准命名空间
int var1 = 200;                        // var1为所有函数之前定义的外部变量
void varScope()                        // 测试变量作用域的函数
{
    extern float var2;                 // 将外部变量var2作用域向上扩展到本函数
    cout << "var2=" << var2 << endl;
    return;                            // 将程序的执行流程带回到主调函数
}
float var2 = 100;                      // var2为在程序中间定义的外部变量
int main()
{
    varScope();
    cout << "var1=" << var1 << endl;
    return 0;
}
```

```
var2=100
var1=200
```

```cpp
//*************************************************************************
//*  程 序 名: 7_6_1.cpp                                                  *
//*  主要功能:                                                            *
//*     测试外部变量的作用域延伸到源文件7_6_2.cpp                         *
//*************************************************************************
#include <iostream>                                  // 包含输入输出接口文件
using namespace std;                                 // 引用标准命名空间令
int score[10] = { 0 };                               // 定义外部数组
int maxScore = 0, minScore = 0;                      // 定义外部变量
void find_Max_Min();
int main()
{
    int i = 0;                                       // 数组下标变量
    cout << "Enter 10 scores:" << endl;
    for(i = 0; i < 10; i++)
    {
        cin >> score[i];
    }
    maxScore = minScore = score[0];
    find_Max_Min();                                  // 调用函数find_Max_Min()
    cout << "The Max score: " << maxScore << endl;
    cout << "The Min score: " << minScore << endl;
    return 0;
}
```

```cpp
//*************************************************************************
//*  程 序 名: 7_6_2.cpp                                                  *
//*  主要功能:                                                            *
//*     测试外部变量作用域的延伸                                          *
//*************************************************************************
extern int score[10];
//引用性声明,将源文件7_6_1.cpp中定义的数组score作用域扩展到本源文件
extern int maxScore, minScore;
// 引用性声明,将源文件7_6_1.cpp中定义的变量maxScore，minScore
// 作用域扩展到本源文件
void find_Max_Min()                                    //求最大值和最小值函数
{
    int i = 0;                                         //数组下标变量
    for(i = 0; i < 10; i++)
    {
        if(score[i] > maxScore)
        {
            maxScore = score[i];
        }
        if(score[i] < minScore)
        {
            minScore = score[i];
        }
    }
    return;//将程序的执行流程带回到主调函数
}
```

局部变量与全局变量小结：

局部和全局是从变量的作用范围（空间上）划分的。

❖ 局部变量

在一个块内定义的变量叫做局部变量。

局部变量只在定义它的块范围内有效，即程序只有在执行该块时才能访问这些变量，在执行其它函数时不能访问。

```
#include <iostream>
using namespace std;
int main( )
{   int f;   int n = 5;
    f = Fibo(n);
    cout<<"f="<<f<<endl;
    return 0;
}
```

n是main函数中的局部变量

f也是main函数中的局部变量

局部变量的有关说明

1.主函数之中定义的变量同样属于局部变量。其它函数不能访问主函数中的局部变量，主函数也不能访问其它函数中的局部变量。

2.不同函数中可以定义相同名字的变量。由于它们是各自函数内部的局部变量，所以互不影响。

3.形参变量也是局部变量。

4.在函数内部，还可以在复合语句(花括号括起的一组语句)内定义变量，这些变量只在该复合语句内起作用。

全局变量

一个源程序文件中，在所有函数外定义的变量叫做全局变量。

全局变量的作用范围从定义的位置开始到该源程序文件结束。也就是说，在这个范围内，所有的函数都可以访问它（当然就可以改变它的值）。

❖ 全局变量的作用：

由于在全局变量的作用范围内，所有的函数都可以访问它，所以利用全局变量就可以在函数之间增加传递信息的通道。包括调用函数时传递信息进去，以及函数返回后得到多个返回值。

7.3 存储属性和生存期

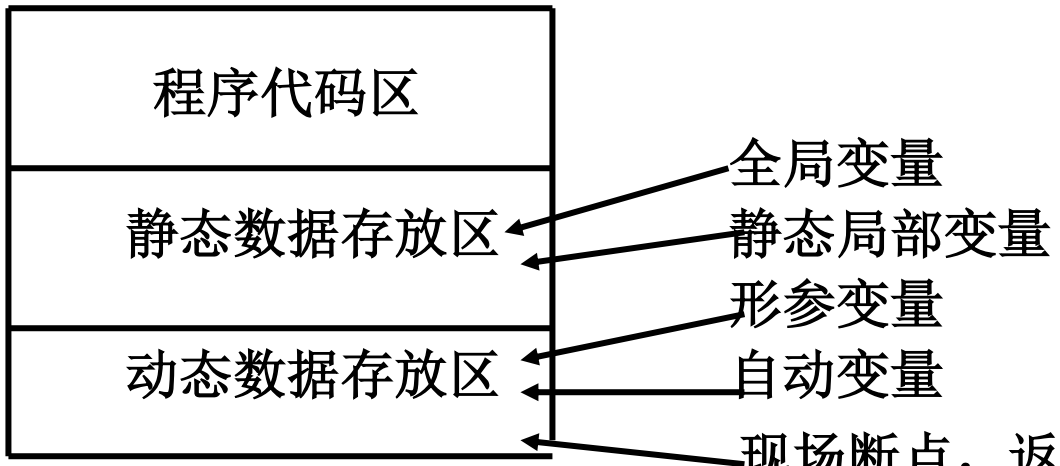
7.3.1 变量的存储属性

在C语言中使用的每个变量都具有两个属性：数据类型和存储类型。数据类型决定了变量在内存中分配的存储单元的多少。

存储类型决定了变量所分配的存储区的类型，而存储区的类型又决定了变量的生存期。所以，变量的存储类型决定了变量的生存期。

变量的存储类型是变量在时间方面的属性。

在程序运行期间，该程序所占据的内存空间叫用户区。用户区又划分为：（内存）用户区



程序区用于存放编译连接后的代码，数据区用于存放程序运行过程中涉及的所有数据。它是变量在内存中的映像。

数据区分为静态数据存储区和动态数据存储区：

- 动态变量存放在动态数据存放区，动态数据存放区由系统在程序运行期间根据需要动态分配，需要时分配，不需要时释放。
- 静态变量存放在静态数据存放区，静态数据存放区一经分配，便要到程序运行结束后才会被释放。

C语言中定义变量存储类型的关键字有以下四个：

① 自动型(auto)；② 外部型(extern)；

③ 静态型(static)；④ 寄存器型(register)。

格式：　存储类型 数据类型 变量名表;

7.3.2 自动变量

自动变量的定义使用关键字auto。关键字auto可省略。

```
例　(auto)int a;
```

- 关键字auto可以缺省，本章以前函数内所定义的变量都是自动变量。
- 自动变量存放在内存的动态数据区，其作用域范围内的代码一旦执行完，这些变量的存储空间就释放了。
- 由于自动变量是内部变量，其作用域仅局限于它定义所在的模块内。

7.3.3 外部变量

外部变量的定义使用关键字extern。

```
例   extern int a;
```

- 外部变量定义在所有函数之外，其作用域从它定义处到源文件结束处，具有文件作用域。
- 关键字extern一般可以缺省。
- 外部变量存放在内存的静态数据区。
- 在一个源文件中，如外部变量要在其定义之前使用，用关键字extern在使用它的函数中进行引用性声明。
- 可以用extern将外部变量的作用域扩展到其它文件

```
#include <iostream>                          // 包含输入输出接口文件
using namespace std;                        // 引用标准命名空间
int var1 = 200;                             // var1为所有函数之前定义的外部变量
void varScope()                             // 测试变量作用域的函数
{
    extern float var2;                      // 将外部变量var2作用域向上扩展到本函数
    cout << "var2=" << var2 << endl;
    return;                                 // 将程序的执行流程带回到主调函数
}
float var2 = 100;                           // var2为在程序中间定义的外部变量
int main()
{
    varScope();
    cout << "var1=" << var1 << endl;
    return 0;
}
```

7.3.4 静态变量

静态变量的定义使用关键字static：

```
例   static int a;
```

根据作用域不同，静态变量可分为静态局部变量和静态全局变量。

- 函数内定义的静态变量为静态局部变量。
- 静态局部变量的作用域同自动变量，两者区别在于生存期的不同。在整个程序的运行过程中,静态局部变量一直存在。
- 静态局部变量能保留上次调用时的值。
- 静态局部变量在编译时赋初值,即只赋初值1次。如果程序中没有给静态局部变量显示赋初值,则编译时,系统自动赋初值0(对数值型变量)或空字符(字符型变量)。

```cpp
#include <iostream>                    // 包含输入输出接口文件
using namespace std;                   // 引用标准命名空间
int fact(int n)                        // 计算n的阶乘
{
    static int result = 1;             // 定义静态变量result并初始化
    result = result * n;               // 计算n的阶乘
    return result;                     // 将程序的执行流程和result的值带回到主调函数
}
int main()
{
    for(int i = 1; i <= 6; i++)
    {
        cout << i << "!= " << fact(i) <<endl;
    }
    return 0;
}
```

```
1!= 1
2!= 2
3!= 6
4!= 24
5!= 120
6!= 720
```

```
int c = 0;                                  // 定义全局变量并初始化
void test();                                // 函数声明
int main()
{
    int a = 1, b = 2;                       // 定义内部变量
    c = 3;                                  // 主函数中改变全局变量c的值
    cout << " ----------------test()-------------" << endl;
    cout << "\n静态局部变量  自动变量  全局变量" << endl;
    for(int i = 1; i <= 3; i++)
    {
        test();                             // 调用test()函数
    }
    cout << "\n -----------main()--
    cout << "\n局部变量\t全局变量"
    cout << "a=" << a << ", b=" <<
    return 0;
}
void test()
{
    static int a = 0;
    auto int b = 0;
    a++;
    b++;
    c++;
    cout << "  a= " << a << "\t\tb=
    return;                                 // 将程序的执行流程带回到主调函数
}                                           // 函数定义结束
```

仅初始化1次！内存数据调用结束后保留！

```
 ----------------test()-----------

静态局部变量   自动变量   全局变量
  a= 1           b= 1      c= 4
  a= 2           b= 1      c= 5
  a= 3           b= 1      c= 6

 -----------main()-------------

局部变量          全局变量
a=1, b=2           c= 6
请按任意键继续. . .
```

静态全局变量

- 全局变量和静态全局变量的存储方式均为静态存储方式，两者的区别在于作用域的不同。
- 静态全局变量作用域为其所在的源程序文件，即只能被该源程序中的函数使用；
- 全局变量可以通过关键字extern将作用域扩展到其他源程序文件。

```cpp
//*************************************************************************
//*  程 序 名: 7_6_1.cpp                                                  *
//*  主要功能:                                                            *
//*     测试外部变量的作用域延伸到源文件7_6_2.cpp                         *
//*************************************************************************
#include <iostream>                                  // 包含输入输出接口文件
using namespace std;                                 // 引用标准命名空间令
static int score[10] = { 0 };                        // 定义外部数组
static int maxScore = 0, minScore = 0;               // 定义外部变量
void find_Max_Min();
int main()
{
    int i = 0;                                       // 数组下标变量
    cout << "Enter 10 scores:" << endl;
    for(i = 0; i < 10; i++)
    {
        cin >> score[i];
    }
    maxScore = minScore = score[0];
    find_Max_Min();                                  // 调用函数find_Max_Min()
    cout << "The Max score: " << maxScore << endl;
    cout << "The Min score: " << minScore << endl;
    return 0;
}
```

```
//*************************************************************************
//*  程 序 名: 7_6_2.cpp                                                  *
//*  主要功能:                                                            *
//*     测试外部变量作用域的延伸                                          *
//*************************************************************************
extern int score[10];
//引用性声明,将源文件7_6_1.cpp中定义的数组score作用域扩展到本源文件
extern int maxScore, minScore;
// 引用性声明,将源文件7_6_1.cpp中定义的变量maxScore，minScore
// 作用域扩展到本源文件
void find_Max_Min()                                  //求最大值和最小值函数
{
    int i = 0;                                       //数组下标变量
    for(i = 0; i < 10; i++)
    {
        if(score[i] > maxScore)
        {
            maxScore = score[i];
        }
        if(score[i] < minScore)
        {
            minScore = score[i];
        }
    }
    return;//将程序的执行流程带回到主调函数
}
```

```
//*  主要功能:                                                    *
//*      使用静态全局变量求10个成绩的最大值和最小值                *
//*************************************************************
#include <iostream>                          // 包含输入输出接口文件
using namespace std;                         // 引用标准命名空间
static int score[10] = { 0 };                // 定义静态全局数组
static int maxScore = 0, minScore = 0;       // 定义静态全局变量
void find_Max_Min();                         // 函数声明
int main()
{
    int i = 0;                               // 数组下标变量
    cout << "Enter 10 scores:" << endl;
    for(i = 0; i < 10; i++)
    {
        cin >> score[i];
    }
    maxScore = minScore = score[0];
    find_Max_Min();                          // 调用函数find_Max_Min()
    cout << "The Max score: " << maxScore << endl;
    cout << "The Min score: " << minScore << endl;
    return 0;
}
void find_Max_Min()                          // 求最大值和最小值函数
{
    int i = 0;                               // 数组下标变量
    for(i = 0; i < 10; i++)
    {
        if(score[i] > maxScore)
        {
            maxScore = score[i];
        }
        if(score[i] < minScore)
        {
            minScore = score[i];
        }
    }
    return;                                  // 将程序的执行流程带回到主i
}
```

```
Enter 10 scores:
98 89 97 79 67 76 56 65 79 94
The Max score: 98
The Min score: 56
```

7.3.5 寄存器变量：

寄存器变量的定义使用关键字register：

```
例  register int i,j,k;
```

寄存器变量存放在寄存器中。寄存器器存取速度远高于内存，主要存放一些使用频繁的局部变量。

- 只有内部变量和形参变量可以定义为寄存器变量。
- 函数调用时为定义为寄存器变量的形参变量分配寄存器，调用结束后就释放所分配的寄存器。
- 寄存器变量是内部变量，其作用域为块作用域或函数作用域，其生命期为函数的每次调用。

```cpp
//*************************************************************
//*  程 序 名: 7_10.cpp                                       *
//*  主要功能:                                                *
//*     使用寄存器变量计算1+2+3+……+n的值                      *
//*************************************************************
#include <iostream>                          // 包含输入输出接口文件
using namespace std;                         // 引用标准命名空间
long sum(int n)                              // 求1到n的自然数之和
{
    register int i = 0,s = 0;                // 定义寄存器变量
    for(i = 1; i <= n; i++)
    {
        s = s + i;
    }
    return s;                                // 将程序的执行流程和累加和带回到主调函数
}
int main()
{
    int n = 0;
    cout << "n=";
    cin >> n;
    cout << "1到" << n <<"的自然数之和为: " << sum(n) << endl;
    return 0;
}
```

```
n=10000
1到10000的自然数之和为: 50005000
```

作用域和生存期的小结

- 文件作用域中定义的对象（全局变量）具有静态生存期。即在程序执行过程中它们占据固定的存储单元一直到程序结束。
- 函数作用域和块作用域中定义的对象（局部变量）具有动态生存期。即在程序执行过程中它们所占据的存储单元是动态地进行分配和释放。
- 函数作用域和块作用域中定义的对象要具有静态生存期必需在其定义时加关键字　　（声明为静态局部变量）

static和extern关键字的小结

☞*static* 关键字用在局部变量定义时，是将该动态局部变量变为静态局部变量；

☞*static* 关键字用在全局变量定义时，是将该全局变量的有效范围限定在定义该全局变量的源文件中；

☞*static* 关键字用在函数定义时，是将该函数的有效范围限定在定义该函数的源文件中；

☞*extern* 关键字可对先使用后定义的全局变量在第一次使用该全局变量处进行‘引用声明’。（在同一文件中扩展外部变量的作用域）

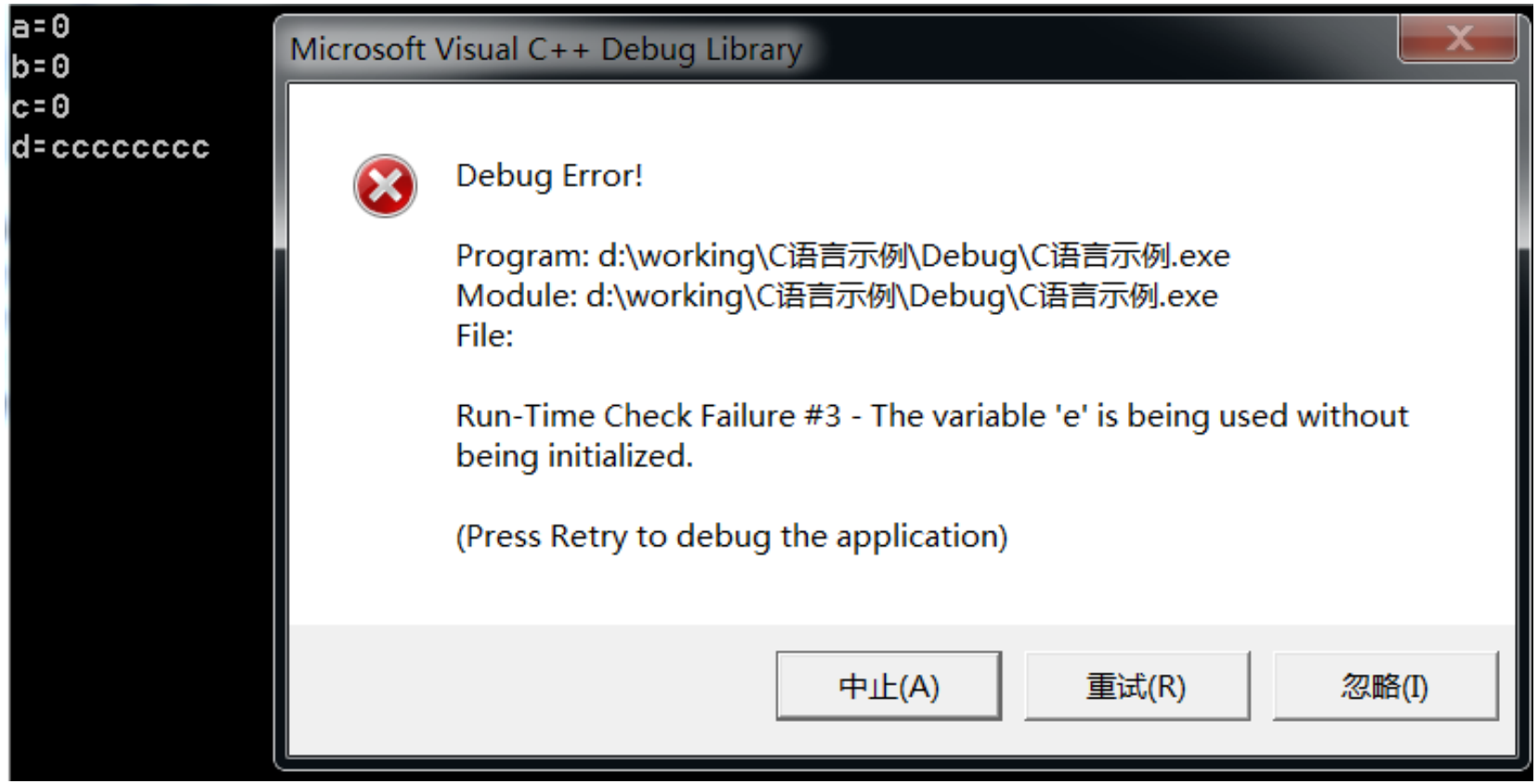
☞*extern* 关键字可对在别的源文件中定义的全局变量作‘引用声明’ （将外部变量的作用域扩展到不同源文件中）

7.4 变量的初始化

7.4.1 内部变量的初始化

- 这里的内部变量指自动变量。自动变量的初始化是在函数调用时进行的，每调用一次函数都将重新进行一次初始化。
- 不同编译器对未初始化的自动变量所赋初值不同，如VS会为未初始化整型自动变量赋初值0xCCCCCCCC；而devCpp则随机赋值（每个变量可能会有不同的值）。
- 如果定义自动变量时未进行初始化，或自动变量未被赋值，则其值将不确定，这将导致得到错误的结果，在程序设计中应注意避免。

```cpp
//.....................................
#include <iostream>
using namespace std;
int a;                          // 全局变量
static int b;                   // 静态全局变量
int main()
{
    static int c;               // 静态局部变量
    register int d;             // 寄存器变量
    int e;                      // 自动变量
    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    cout << "c=" << c << endl;
    cout << "d=" <<hex<< d << endl; // 以输出16进制数
    cout << "e=" <<hex<< e << endl; // 以输出16进制数
    return 0;
}
```




```
a=0
b=0
c=0
d=cccccccc
e=cccccccc
```

```
a=0
b=0
c=0
d=28ff48
e=760f9e34
```

7.4.2　外部变量的初始化

- 外部变量可以被本源程序文件中其他函数使用，其作用域从其定义的位置开始，一直到本源程序结束。
- 外部变量在程序执行过程中，占有固定的存储单元，其生存期为整个程序。
- 在定义外部变量时，如未进行初始化，则数值型外部变量的值缺省为0，字符型外部变量的值缺省为字符'\0'。

```cpp
#include <iostream>
using namespace std;
int a;                          // 全局变量
static int b;                   // 静态全局变量
int main()
{
    static int c;               // 静态局部变量
    register int d;             // 寄存器变量
    int e;                      // 自动变量
    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    cout << "c=" << c << endl;
    cout << "d=" <<hex<< d << endl; // 以输出16进制数
    cout << "e=" <<hex<< e << endl; // 以输出16进制数
    return 0;
}
```

```
a=0
b=0
c=0
d=cccccccc
e=cccccccc
```

7.4.3 静态变量的初始化

- 静态变量，不管是静态全局变量，还是静态局部变量，都是在编译时，系统分配固定的存储单元，并始终存在，直到源程序运行结束。
- 如在定义静态变量时进行初始化，则静态变量只在第一次使用时被赋初值。如未进行初始化，则数值型静态变量的缺省值为0，字符型静态变量的缺省值为字符'\0'。

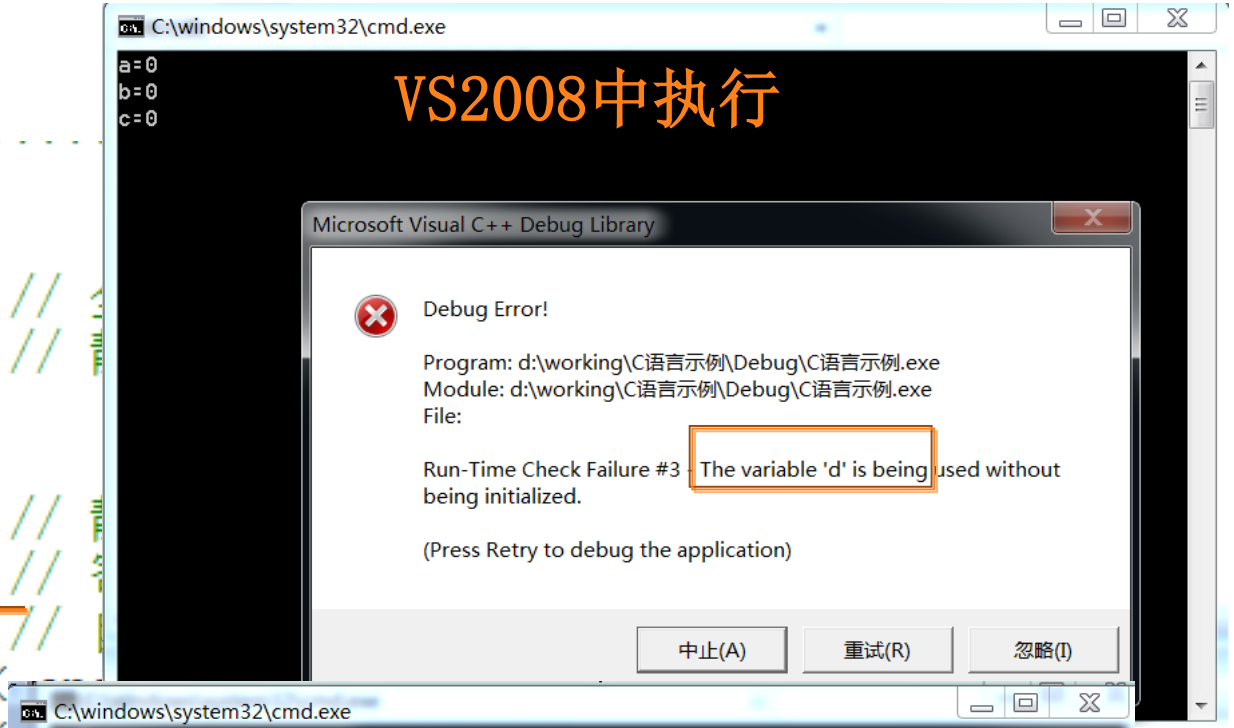
```
//.....................................
#include <iostream>
using namespace std;
int a;                  // 全局变量
static int b;           // 静态全局变量
int main()
{
    static int c;       // 静态局部变量
    register int d;     // 寄存器变量
    int e;              // 自动变量
    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    cout << "c=" << c << endl;
    cout << "d=" <<hex<< d << endl; // 以输出16进制数
    cout << "e=" <<hex<< e << endl; // 以输出16进制数
    return 0;
}
```

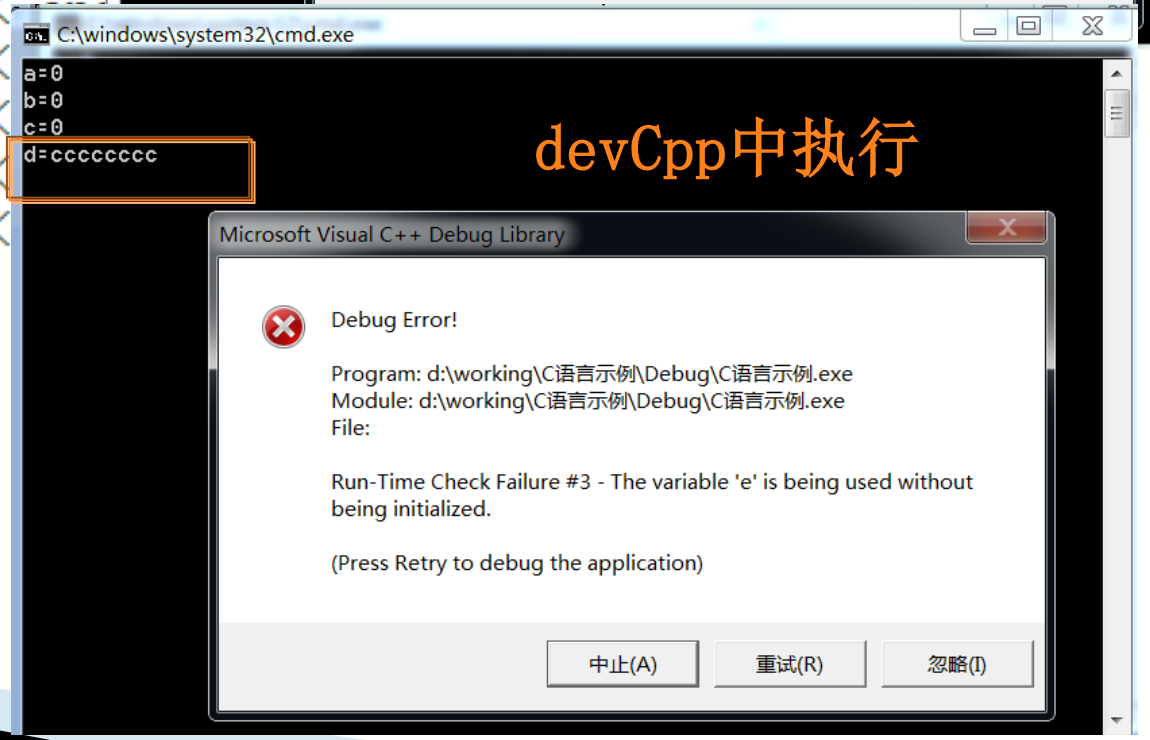
```
a=0
b=0
c=0
d=cccccccc
e=cccccccc
```

7.4.4　寄存器变量的初始化

- 寄存器变量作用域同局部变量，函数调用结束后，寄存器变量的存储空间被释放。
- 寄存器变量的初始化是在函数调用时进行的，如未进行初始化，其值将不确定，这一点与自动变量相同。

```
#include <iostream>
using namespace std;
int a;
static int b;
int main()
{
    static int c;
    register int d;
    int e;
    cout << "a=" << a <<
    cout << "b=" << b <<
    cout << "c=" << c <<
    cout << "d=" <<hex<<
    cout << "e=" <<hex<<
    return 0;
}
```





1.凡是函数中未指定存储类型的局部变量，其默认的存储类型是（　　）。

A. auto　　B. register

C. extern　　D. static

A

2.在C语言规范中，形参的缺省存储类型是(　　)。

A. auto　　B. register

C. extern　　D. static

A

3.在一个C源程序文件中，若要定义一个只允许本源文件中所有函数使用的全局变量，则该变量需要使用的存储类型是(　　)。

A. auto　　B. register

C. extern　　D. static

D

14. 以下程序的正确运行结果是(　　)。

```
#include <iostream>
using namespace std;
int m=1;
int n=5;
int min(int a, int b);
void main( )
{
        int m=100, n=10, t;
        t=min(m, n);
        cout << t << endl;
}
int min(int a, int b)
{
        int c;
        c=(a<=b)? a:b;
        return(c);
}
```

A. 0　　B. 1　　C. 5　　D. 10

D