第5章

函　　数

CHAPTER 5

案例导读

脉络导图

学习目标

技能目标：

(1) 能编写和阅读模块化结构程序。

(2) 掌握函数的定义及调用方式。

(3) 掌握局部变量和全局变量的区别和典型用法。

(4) 掌握运用函数处理多个任务的能力。

素质目标：

(1) 通过学习函数和模块化程序设计思想，培养学生在工作、

生活中遇到困难时，能够积极面对，将大问题划分成小问题依次去解决。

（2）通过学习预处理程序，使学生明白不打无准备之仗。现在要好好学习专业知识，这样在工作中才能更好地完成任务。

（3）通过程序常见错误分析与改正，使学生明白更加美好的人生需要积累、不断改正不足，争取进步。

（4）通过递归函数的学习，明白把一个大型复杂的任务拆分成一个个小任务进行处理的重要性。

5.1 函数概述

5.1.1 函数引入

前几章内容已涉及函数的概念,例如,标准输入函数 scanf()、标准输出函数 printf()及其他函数。这类函数称为C语言的标准库函数,是由C语言开发环境事先提供给编程人员的。编程人员实际编程时只需调用这些函数即可,至于这些函数是如何实现功能的编程人员不必知晓。有了C语言的标准库函数,编程人员既加强了所编程序的功能,又提高了编程效率。但在实际编程中,若程序的规模比较大,将所有代码都写在 main()函数中,会使 main()函数变得十分庞杂,不易于程序的阅读和维护。这时可以利用函数将程序划分成多个小的模块,从而方便理解和修改程序。

模块化程序设计思想是指将一个较大的程序分为若干程序模块,每个模块用来实现一个特定的功能。在C语言中,用函数来实现模块的功能。一个C程序可由一个 main()函数和若干其他函数构成。由 main()函数调用其他函数,其他函数可以相互调用。同一个函数可以被一个或多个函数调用任意次。例如,在学校组织学生打扫教室卫生这项活动中,一般由老师组织学生来进行。其中,一部分学生擦窗户,一部分学生擦桌子,一部分学生扫地。编写程序就像打扫卫生一样,main()函数如同组织学生的老师,功能是控制每一步程序的执行,定义的其他函数就好比是各部分学生,分别完成特定的功能。

在C语言中,可以从不同的角度对函数分类。

1. 从函数定义角度看

函数可分为库函数和用户定义函数两种。

(1) 库函数。

库函数是由系统提供的,用户不必自己定义,也不必在程序中做类型说明,只需在程序前包含该函数原型的头文件,即可在程序中直接调用。例如,调用 printf()函数和 scanf()函数时需要在程序开头包含 stdio.h 头文件;调用 sqrt()函数和 log()函数时需要包含 math.h 头文件;调用 strcpy()函数和 strlen()函数时需要包含 string.h 头文件。

(2) 用户定义函数。

用户定义函数是由用户按需要编写的函数。对于用户定义函数,不仅要在程序中定义函数本身,而且在主调函数模块中还必须对该被调函数进行类型说明,然后才能使用。

2. 从对函数返回值的需求状况看

C语言函数可分为有返回值函数和无返回值函数两种。

(1) 有返回值函数。

此类函数被调用执行完成后将向调用者返回一个执行结果,称为函数返回值,例如,数学函数。由用户定义的需要返回函数值的函数,必须在函数定义和函数说明中明确返回值的类型。

（2）无返回值函数。

此类函数用于完成某项特定的处理任务，执行完成后不向调用者返回函数值。这类函数并非真的没有返回值，程序设计者也不关心它，此时关心的是它的处理过程。由于函数无须返回值，用户在定义函数时，可指定它的返回为“空类型”，说明符为 void。

3. 从主调函数和被调函数之间数据传送的角度看

C 语言函数可分为无参函数和有参函数。

（1）无参函数。

无参函数指函数定义、函数说明及函数调用中均不带参数，主调函数和被调函数之间不进行参数传送。函数通常用来完成一组指定的功能，可以返回或不返回函数值。

（2）有参函数。

有参函数指在函数定义及函数说明时都有参数，称为形式参数（简称“形参”）。在函数调用时也必须给出参数，称为实际参数（简称“实参”）。进行函数调用时，主调函数将把实参的值传送给形参，供被调函数使用。

4. 从功能角度看

C 语言提供了极为丰富的库函数，这些库函数又可从功能角度分为多种类型。在 C 语言中，所有的函数定义都是平行的，也就是说，在一个函数的函数体内，不能再定义另一个函数，即不能嵌套定义。但函数之间允许相互调用，也允许嵌套调用，习惯上把调用者称为主调函数。函数还可以自己调用自己，称为递归调用。main()函数是主函数，它可以调用其他函数，而不允许被其他函数调用。

名师点睛

C 程序的执行总是从 main()函数开始，完成对其他函数的调用后再返回 main()函数，最后由 main()函数结束整个程序。一个 C 源程序必须有且只能有一个 main()函数。

5.1.2　函数的定义

函数定义的一般格式如下：

```
函数类型 函数名(形参及其类型)
{
    局部变量定义语句;
    可执行语句序列;
}
```

其中：

（1）函数类型是指函数返回值的数据类型，可以是基本数据类型、void 类型、指针类型等。

（2）函数名是一个有效、唯一的标识符，符合标识符的命名规则。函数名不仅用来标识函数、调用函数，同时它本身还存储着该函数的内存首地址。

（3）形参是实现函数功能所要用到的传输数据，它是函数间进行交流通信的唯一途径。

由于形参是由变量充当的,因此必须定义类型。定义形参时,在函数名后的括号中定义。形参可以为空,表示没有参数,也可以由多个参数组成,参数之间用逗号隔开。

(4) 函数体是由{}括起来的一组复合语句,一般包含两部分:声明部分和执行部分。其中,声明部分主要是完成函数功能时所需要使用的变量的定义,执行部分则是实现函数功能的主要程序段。

(5) 对于有返回值的函数,必须用带表达式的 return 语句来结束函数的运行,返回值的类型应与函数类型相同。如果 return 语句中表达式的值与函数定义的类型不一致,则以函数定义类型为准,并自动将 return 语句中的表达式的值转换为函数返回值的类型。

【例 5-1】 编写程序,计算两个整数的差。

```
#include <stdio.h>
int subtract(int i, int j)            /* 自定义函数 subtract() */
{
    int result;
    result = i - j;
    return result;
}
```

程序说明:subtract()函数是用户自定义函数,函数类型为整型,函数名为 subtract,形参为整型变量 i 和变量 j;函数体是实现 subtract()函数功能的语句块,计算两个整数的差。

【例 5-2】 不带参数的函数定义,且函数无返回值。

```
void printmsg()
{
     printf("hello world");
}
```

程序说明:printmsg()函数无参数,函数名 printmsg 前标注了 void,表示函数无返回值,没有 return 语句,因此函数执行完 printf 语句后自动返回。

5.1.3 函数的调用

函数的使用是通过函数调用语句来完成的。函数调用是指一个函数暂时中断本函数的运行,转去执行另一个函数的过程。C 语言是通过 main()函数来调用其他函数的,其他函数之间可相互调用,但不能调用 main()函数。函数被调用时获得程序控制权,调用完成后,返回到调用函数中断处继续运行。函数调用的一般格式如下:

```
函数名(实参列表)
```

名师点睛

(1) 实参可以是常量、有确定值的变量或表达式及函数调用。

(2) 实参的个数必须与形参的个数一致。实参的个数多于一个时,各实参之间用逗号隔开。

(3) 若调用无参函数,则“实参列表”可以没有,但括号不能省略。

按被调用函数在 main()函数中出现的位置和完成的功能进行划分，函数调用有以下 3 种方式。

(1) 把函数调用作为一个语句。例如，“printf("sum＝%d\n",sum);”以独立函数语句的方式调用函数。

(2) 在表达式中调用函数，这种表达式称为函数表达式。例如，“c＝4 * max(a,b);”是一个赋值表达式，把 4 * max(a,b)的值赋予变量 c。

(3) 将函数调用作为另一个函数的实参。例如，“printf("max＝%d\n",max(a,b));”把 max()函数调用的返回值又作为 printf()函数的实参来使用。

【例 5-3】 求两个实数的平均值。

```
#include <stdio.h>
float average(float x,float y)              /*定义函数用于计算两数的平均值,x 和 y 为形参*/
{
    float av;                               /*定义变量 av 用于存放平均值*/
    av = (x + y)/2.0;
    return av;                              /*返回 av 的值*/
}
int main()
{
    float a = 1.8,b = 2.6,c;
    c = average(a,b);                       /*实参为确定值的变量*/
    printf("%5.2f 和 %5.2f 的平均值是: %5.2f\n",a,b,c);
    c = average(a,a + b);                   /*实参为表达式*/
    printf("%5.2f 和 %5.2f 的平均值是: %5.2f\n",a,a + b,c);
    c = average(2.0,4.0);                   /*实参为常量*/
    printf("2.0 和 4.0 的平均值是: %5.2f\n",c);
    c = average(a,average(a,b));            /*实参为函数调用*/
    printf("平均值是: %5.2f\n",c);
    return 0;
}
```

运行结果：

```
1.80 和 2.69 的平均值是: 2.20
1.80 和 4.40 的平均值是: 3.10
2.0 和 4.0 的平均值是: 3.00
平均值是: 2.00
```

程序说明：求两个实数的平均值函数 average()有两个形参 x 和 y，这两个参数用来接收调用函数时传递来的变量或表达式的值。该程序 main()函数调用了 4 次 average()函数，第 1 次调用时，用形参 x 和 y 接收实参变量 a 和变量 b 的值；第 2 次调用时，用表达式 a＋b 作为实参之一，将 a 和 a＋b 的值传给形参 x 和 y；第 3 次调用时，用常量作为实参，将 2.0 和 4.0 的值传给 x 和 y；第 4 次调用时，用函数调用 average(a,b)作为实参之一，将 c 和 average(a,b)的值传给形参 x 和 y。

5.1.4　函数的声明

编译程序在处理函数调用时，必须从程序中获得完成函数调用所必需的接口信息。函

数的声明是指对函数类型、名称等的说明。为函数调用提供接口信息,对函数原型的声明是一条程序说明语句。

函数原型的声明就是在函数定义的基础上去掉函数体,后面加上分号";"。函数声明定义的一般格式如下:

```
函数类型 函数名(形参及其类型);
```

例如:

```
int max(int a,int b);
```

之所以需要函数的声明,是为了获得调用函数的权限。如果在调用之前定义或声明了函数,则可以调用该函数。

被声明的函数往往定义在其他的文件或库函数中。可以把不同类型的库函数声明放在不同的库文件中,然后在设计的程序中包含该文件。例如,#include "math.h",其中 math.h 文件包含了很多数学函数的原型声明。

这样做的好处是方便调用和保护源代码。库函数的定义代码已经编译成机器码,对用户而言是不透明的,但用户可以通过库函数的原型获得参数说明并使用这些函数完成程序设计。

对于用户自定义函数,也可以这样处理。和使用库函数不同的是,经常把自己设计的函数放在调用函数后。例如,习惯于先设计 main()函数,再设计定义的函数,这时候需要超前调用自定义函数,在调用之前需要进行函数原型声明。

名师点睛

(1) 变量的声明通常是对变量的类型和名称的一种说明,不一定会分配内存,而变量的定义肯定会分配内存空间。

(2) 函数的声明是对函数的类型和名称的一种说明,而函数的定义是一个模块,包括函数体部分。

(3) 声明可以是定义,也可以不是。广义上的声明包括定义性声明和引用性声明,通常所说的声明是指后者。

C 语言规定以下 3 种情况,可以不在主调函数中对被调函数进行声明:

(1) 如果被调函数写在主调函数的前面,可以不必进行声明。

(2) 如果函数的返回值为整型或字符型,可以不必进行声明。

(3) 如果在所有函数定义之前,在源程序文件的开头,即在函数的外部已经对函数进行了声明,则在各个调用函数中不必再对所调用的函数进行声明。

【例 5-4】 求两个整数中较大的值。

```
#include <stdio.h>
int max(int a,int b);                    /* 子函数 max()的声明语句 */
int main()
{
    int x,y,z;
```

```
    printf("请输入两个整数:");
    scanf("%d%d",&x,&y);
    z = max(x,y);              /* 子函数 max()在 main()函数中的调用语句 */
    printf("%d 和 %d 的较大值是 %d!",x,y,z);
    return 0;
}
int max(int a,int b)           /* 子函数 max()的函数头,其中变量 a、变量 b 是形参 */
{
    int m;                     /* 定义函数内部变量 m */
    if(a > b)
        m = a;
    else
        m = b;
    return m;                  /* 子函数返回语句 */
}
```

运行结果：

```
请输入两个整数: 18 33
18 和 33 的较大值是 33!
```

程序说明：程序中定义两个函数——main()函数和 max()函数。max()函数定义变量 m，存放两个参数中较大的数，通过 return 语句把 m 的值返回调用函数。main()函数通过调用语句“z＝max(x,y);”求两个数中较大的数。需要注意子函数的定义、调用和子函数的声明。

5.1.5　函数的参数传递

函数调用需要向子函数传递数据，一般是通过实参将数值传递给形参。实参向形参的参数传递有两种形式：值传递和地址传递。

值传递是指单向的数据传递(将实参的值赋给形参)，传递完成后，对形参的任何操作都不会影响实参的值。

地址传递是指将实参的地址传递给形参，使形参指向的数据和实参指向的数据相同，因而被调函数的操作会直接影响实参指向的数据。

【例 5-5】 在奖学金评定中，学生的思想品德修养是极为重要的。编程实现比较两位同学的品德修养成绩，输出较高的成绩。

```
#include <stdio.h>
void max(int i,int j,int k)                /* 自定义 max()函数,输出最大值 */
{
    k = i > j?i: j;                        /* 把 i 和 j 里面的最大值赋值给 k */
    printf("品德修养成绩较高的是: %d\n",k);
}
int main()
{
    int a = 0,b = 0,c = 0;
    printf("请输入两位同学的成绩: \n");
    printf("第一位同学的成绩:");
    scanf("%d",&a);
```

```
    printf("第二位同学的成绩:");
    scanf("%d",&b);
    max(a,b,c);
    return 0;
}
```

运行结果:

```
请输入两位同学的成绩:
第一位同学的成绩: 92
第二位同学的成绩: 95
品德修养成绩较高的是: 95
```

程序说明:对 max()函数调用时,直接将实参变量 a、变量 b 和变量 c 的值传递给形参变量 i、变量 j 和变量 k。值传递是从实参到形参的单向传递。

【例 5-6】 函数值传递和地址传递。

```
#include <stdio.h>
void change(int x,int y)                    /*change()函数的功能是交换两个形参的值*/
{
    int t;
    printf("change()子函数中两个参数交换前: x=%d,y=%d\n",x,y);
    t=x;
    x=y;
    y=t;
    printf("change()子函数中两个参数交换后: x=%d,y=%d\n",x,y);
}
void add(int a[])                           /*add()函数功能是批量将每个数组元素值乘 2*/
{
    int i;
    for(i=0; i<10; i++)
        a[i]*=2;                            /*每个数组元素的值乘 2*/
}
int main()
{
    int a,b,i;
    int x[10]={1,3,5,7,9,11,13,15,17,19};
    printf("请输入两个整数:");
    scanf("%d%d",&a,&b);
    change(a,b);                            /*change()函数调用语句*/
    printf("main()函数中两个实参在调用 change()子函数后的值为: a=%d,b=%d\n",a,b);
    printf("原数组 x 中的 10 个元素值为: \n");
    for(i=0; i<10; i++)
        printf("%d ",x[i]);
    add(x);                                 /*add()函数调用语句*/
    printf("\n 调用 add()子函数后,数组 x 中的 10 个元素值为: \n");
    for(i=0; i<10; i++)
        printf("%d ",x[i]);
    return 0;
}
```

运行结果如图 5-1 所示。

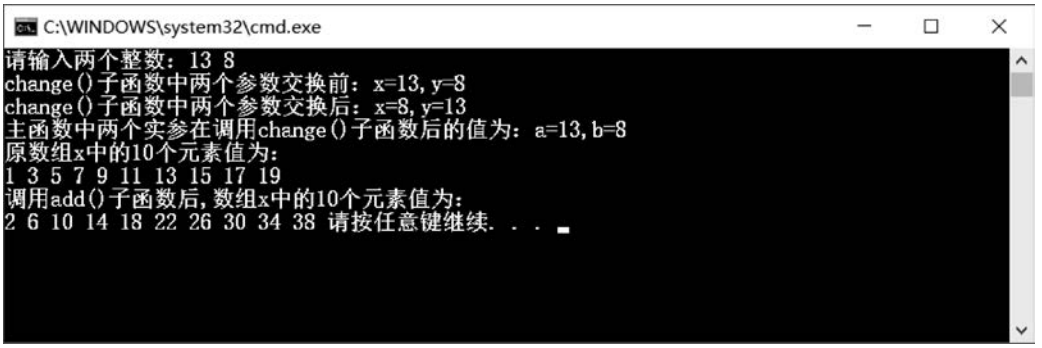

图 5-1　函数值传递和地址传递运行结果

程序说明：因为值传递后，形参值的改变不会影响实参，所以在 change()函数中交换两个形参值后输出这两个值，在 main()函数再重新输出两个实参值，会发现两个实参的值并没有改变。这也证明了值传递方式是单向的数据传递。在 add()函数中将数组作为函数参数，相当于实参和形参共用同一个数组空间，那么对形参中每个数组元素值的改变，也同样对实参数组的每个值改变。在 main()函数中再输出实参数组的每个元素，数组元素值都被乘以 2。需要注意两个子函数书写格式和两个子函数的调用格式。

5.1.6　返回语句和函数返回值

一般情况下，主调函数调用完被调函数后，都希望能够得到一个确定的值。在 C 语言中，函数返回值是通过 return 语句来实现的。函数返回值的一般格式如下：

```
return(表达式);
return 表达式;
return;
```

名师点睛

（1）return 语句可使函数从被调函数中退出，返回到调用它的代码处，并向调用函数返回一个确定的值。

若需要从被调函数返回一个函数值（供主调函数使用），被调函数中必须包含 return 语句且带表达式，此时使用 return 语句的前两种形式均可。若不需要从被调函数返回函数值，应该用不带表达式的 return 语句，也可以不用 return 语句，这时被调函数一直执行到函数体的末尾，然后返回主调函数。

（2）一个函数中可以有多个 return 语句，执行到哪一个 return 语句，哪一个语句就起作用。

（3）在定义函数时应当指定函数的类型，并且函数的类型一般应与 return 语句中表达式的类型一致。当二者不一致时，应以函数的类型为准，即函数的类型决定返回值的类型。对于数值型数据，可以自动进行类型转换。

【例 5-7】　求两个实数的和。

```
#include <stdio.h>
int add(float i,float j)
{
     float k;
     k = i + j;
     return k;
}
int main ()
{
     float a,b,c;
     printf("请输入两个实数:");
     scanf("%f%f",&a,&b);
     c = add(a,b);   /* 函数调用 */
     printf("%5.1f + %5.1f = %5.2f\n",a,b,c);
}
```

运行结果：

```
请输入两个实数: 3.1 2.3
3.1 + 2.3 = 5.00
```

程序说明：输入 3.1 和 2.3，输出结果为“3.1＋2.3＝5.00”，明显结果不正确。因为 add()函数的函数类型为整型，返回值为浮点型，类型不一致，返回值 k 则以函数定义时类型为主，由系统自动将 float 型转换为 int 型。

5.1.7　函数的嵌套调用与递归调用

1. 函数的嵌套调用

嵌套调用是指在调用一个函数并执行该函数的过程中，又调用另一个函数的情况。

图 5-2 给出了函数的嵌套调用示意图，main()函数实现了对 fun1()函数和 fun2()函数的调用。由于 main()函数首先调用 fun1()函数，fun1()函数又对 fun2()函数进行调用，fun1()函数中嵌套了 fun2()函数。函数的嵌套调用如图 5-2 所示。

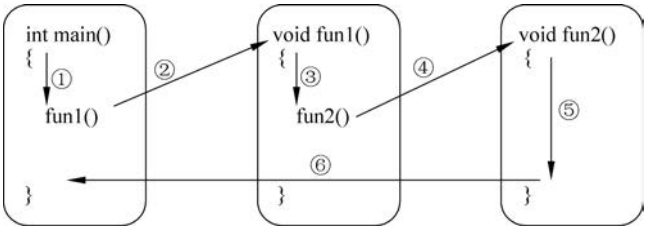

图 5-2　函数的嵌套调用示意图

【例 5-8】　使用函数的嵌套调用计算 1!＋2!＋3!＋…＋10!的值并输出。

```
#include <stdio.h>
#define N 10                          /* 宏定义 */
int main()
{
```

```
    float sum(int n);                    /* 对 sum()函数进行声明 */
    printf("1! + 2! + 3! + 4! + … + 10!= % - 12.5le\n",sum(N));        /* 调用 sum()函数 */
    return 0;
}
float sum(int n)                         /* 定义 sum()函数,求累加 */
{
    float fac(int k)                     /* 对 fac()函数进行声明 */
    int i;
    float s = 0;
    for(i = 1; i < n; ++i)
        s += fac(i);                     /* 调用 fac()函数 */
    return s;
}
float fac(int k)                         /* 定义 fac()函数,计算阶乘 */
{
    int i;
    float t = 1;
    for(i = 2; i <= k; ++i)
        t * = i;
    return t;
}
```

运行结果：

```
1! + 2! + 3! + 4! + … + 10!= 4.09113e + 005
```

程序说明：由于要在 main()函数中调用 sum()函数，因此，在 main()函数的开始要对 sum()函数进行声明。由于要在 sum()函数中调用 fac()函数，因此，在 sum()函数的开始要对 fac()函数进行声明。由于在 main()函数中没有直接调用 fac()函数，因此，在 main()函数中不必对 fac()函数进行声明。当然，所有函数的声明也可以写在 main()函数的前面，这样就不需要在函数内部进行再次声明。

2. 函数的递归调用

函数的递归调用是指函数直接或间接地调用其本身。递归调用有两种方式：直接递归调用和间接递归调用。其中，直接递归调用是指在一个函数中直接调用自身。间接递归调用是指在一个函数中调用其他函数，而在其他函数中又调用了本函数。递归调用的过程分为两个阶段：递推和回归。递推阶段是指从原问题出发，按递归公式递推，最终达到递归终止条件，从而将一个复杂问题分解为一个相对简单且可以直接求解的子问题。回归阶段是指将子问题的结果逐层代入递归公式求值，最终求得原问题的解。

【例 5-9】 用递归调用方法计算 1＋2＋3＋…＋n 的值。

```
#include < stdio.h >
int sum(int);
int main()
{
    int n,s;
    printf("请输入一个整数:");
    scanf(" % d",&n);
```

```
    s = sum(n);
    printf("s = 1 + 2 + 3 + … + %d = %d\n",n,s);
}
int sum(int n)
{
    if(n == 1)
        return 1;
    else
        return n + sum(n - 1);
}
```

运行结果：

```
请输入一个整数: 5
s = 1 + 2 + 3 + … + 5 = 15
```

程序说明：程序的执行流程如图 5-3 所示。

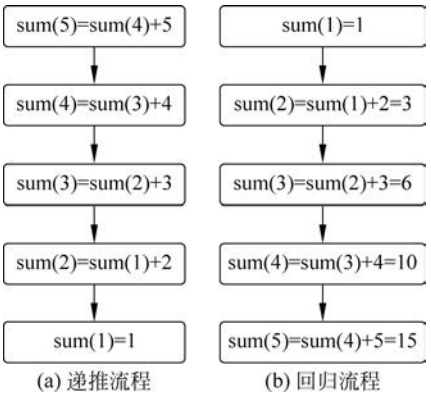

图 5-3　程序的执行流程

5.2　变量的作用域与生命期

5.2.1　变量的作用域

在 C 语言中，用户名命名的标识符都有一个有效的作用域。不同的作用域允许相同的变量和函数出现，同一作用域变量和函数不能重复。

依据变量作用域的不同，C 语言变量可以分为局部变量和全局变量两类。局部变量是指在函数内部或复合语句内部定义的变量。函数的形参也属于局部变量。全局变量是指在函数外部定义的变量。有时将局部变量称为内部变量，全局变量称为外部变量。

名师点睛

(1) 所有函数都是平行关系，main()函数也不例外。main()函数中定义的变量只在 main()函数中有效，不能使用其他函数中定义的内部变量。

(2) 不同的函数内可以定义相同名字的内部变量，它们互不影响。

（3）形参属于被调函数的内部变量，实参属于主调函数的内部变量。

（4）在同一源文件中，若全局变量与局部变量同名，则在局部变量的作用范围内全局变量不起作用。

【例 5-10】 不同函数中的同名变量。

```
#include <stdio.h>
int sub();
int main()
{
    int a = 5,b = 8;
    printf("main: a = %d,b = %d\n",a,b);
    sub();
    printf("main: a = %d,b = %d\n",a,b);
}
int sub()
{
    int a = 1,b = 7;
    printf("sub: a = %d,b = %d\n",a,b);
    return a,b;
}
```

运行结果：

```
main: a = 5,b = 8
sub: a = 1,b = 7
main: a = 5,b = 8
```

5.2.2　变量的生命期

变量的生命期是指变量值在程序运行过程中的存在时间。C 语言变量的生命期分为静态生命期和动态生命期。

一个程序占用的内存空间通常分为两部分：程序区和数据区。数据区可以分为静态存储区和动态存储区。

程序区中存放的是可执行程序的机器指令。静态存储区中存放的是静态数据。动态存储区中存放的是动态数据，如动态变量。动态存储区分为堆内存区和栈内存区。堆和栈是不同的数据结构，栈由系统管理，堆由用户管理。

静态变量是指 main()函数执行前就已经分配内存的变量，其生命期为整个程序执行期；动态变量在程序执行到该变量声明的作用域时才临时分配内存，其生命期仅在其作用域内。

生命期和作用域是不同的概念，分别从时间和空间上对变量的使用进行界定，相互关联又不完全一致。例如，静态变量的生命期贯穿整个程序，但作用域是从声明位置开始到文件结束。

【例 5-11】 变量作用域。

```
#include <stdio.h>
int s = 30,x = 12;          /* 定义全局变量 s 和 x,作用域为从定义到程序末尾 */
```

```
int add(int x,int y)
{
     return x + y;          /* 形参 x、y 作用域为子函数 add()内部 */
}
int main()
{
     int x = 5,y = 3,z = 0;  /* 定义局部变量 x、y、z 作用域为 main()函数内部,屏蔽全局变量 x */
     printf("main()函数初始: s = %d,x = %d,y = %d,z = %d\n",s,x,y,z);
     {
          int x = 1;        /* 定义块内的局部变量 x,屏蔽 main()函数的变量 x 和全局变量 x */
          y = 20;           /* 修改 mian()函数中定义的局部变量 y 值 */
          z = add(x,y);
          printf("程序块中: s = %d,x = %d,y = %d,z = %d\n",s,x,y,z);
     }
     z = add(x,y);
     s = 18;                /* 在 mian()函数中直接修改全局变量 s */
     printf("main()函数修改: s = %d,x = %d,y = %d,z = %d\n",s,x,y,z);
}
```

运行结果:

```
main()函数初始: s = 30,x = 5,y = 3,z = 0
程序块中: s = 30,x = 1,y = 20,z = 21
main()函数修改: s = 18,x = 5,y = 20,z = 25
```

程序说明:

(1) 全局变量 s=30 和 x=12,但因为在 main()函数和程序块内都有同名变量,所以变量 x 被屏蔽了。s 在函数外定义,在 main()函数和各个子函数内都可以被改变,所以 main()函数被改为 15。

(2) 在 main()函数内定义的变量 x=5 的作用域在 main()函数的内部,而程序块内又定义了变量 x=1,所以块内的 x 值为 1,直到块结束。而 main()函数内的变量 y=3 可以在块内被直接改变,所以 y 值改为 20。"z=add(x,y);"调用语句中的 x、y 值分别为 1 和 20,则返回值为 21,即 z 值为 21。

(3) 在块程序后面重新调用函数"z=add(x,y);",则语句中的 x、y 值分别为 5 和 20。返回值为 25,即 z=25。

5.2.3 变量的存储类型

变量的存储类型有 4 种,分别由 4 个关键字表示:auto(自动)、register(寄存器)、static(静态)和 extern(外部)。

1. auto 类型

自动变量是指用 auto 定义的变量,可默认为 auto。自动类型的变量值是不确定的,如果初始化,则赋初始值操作是在调用时进行的,且每次调用都要重新赋初值。

在函数中定义的自动变量只在该函数内有效,函数被调用时分配存储空间,调用结束就释放。在复合语句中定义的自动变量只在该复合语句中有效,退出复合语句后,便不能再使用,否则将引起错误。

2. register 类型

寄存器变量是指用 register 定义的变量，是一种特殊的自动变量。这种变量建议编译程序时将变量中的数据存放在寄存器中，而不像一般的自动变量占用内存单元，可以大大提高变量的存取速度。

一般情况下，变量的值都是存储在内存中的。为提高执行效率，C 语言允许将局部变量的值存放到寄存器中，这种变量就称为寄存器变量。

3. static 类型

全局变量和局部变量都可以用 static 来声明，但意义不同。

全局变量总是静态存储，默认值为 0。全局变量前加上 static 表示该变量只能在本程序文件内使用，其他文件无使用权限。对于全局变量，static 关键字主要用于在程序包含多个文件时限制变量的使用范围，对于只有一个文件的程序，有无 static 都是一样的。

局部变量定义在函数体内部，用 static 来声明时，该变量为静态局部变量。静态局部变量属于静态存储，在程序执行过程中，即使所在函数调用结束也不释放。

静态局部变量定义并不初始化，自动赋数字“0”（整型和实型）或'\0'（字符型）。每次调用定义静态局部变量的函数时，不再重新为该变量赋初值，只是保留上次调用结束时的值，所以要注意多次调用函数时静态局部变量每次的值。

4. extern 类型

在默认情况下，在文件域中用 extern 声明（主要不是定义）的变量和函数都是外部的。但对于作用域范围之外的变量和函数，需要使用 extern 进行引用行声明。

对外部变量的声明，只是声明该变量是在外部定义过的一个全局变量在这里引用。而对外部变量的定义，则是要分配存储单元。一个全局变量只能定义一次，可以多次引用。用 extern 声明外部变量的目的是可以在其他文件中调用。

【例 5-12】 静态变量。

```
#include <stdio.h>
int func(int a,int b);
int main()
{
    int k = 4,m = 1,p;
    p = func(k,m);
    printf("第一次调用子函数后结果为 %d.",p);
    p = func(k,m);
    printf("\n 第二次调用子函数后结果为 %d.",p);
}
int func(int a,int b)
{
    static int m = 0,i = 2; /* 定义静态变量 m 和 i,从第二次起每次调用时初值为上次调用结果 */
    i += m + 1;
    m = i + a + b;
    return(m);          /* 在 mian()函数中直接修改全局变量 s */
}
```

运行结果：

```
第一次调用子函数后结果为 8。
第二次调用子函数后结果为 17。
```

程序说明:

(1) 用 static 定义在函数内部的变量是静态局部变量,它们只在函数第一次被调用时赋初值,以后该函数再次被调用时,其静态变量值为上次函数调用后的终值。所以在该程序中注意多次调用函数时静态局部变量 m 和 i 的初值即可。

(2) 第一次调用时,子函数 func()中静态变量 m 初值为 0,i 初值为 2,所以第一次调用后 i 的值为 0+1+2=3,m 的值为 3+4+1=8。第二次调用子函数时,m 初值为 8,i 初值为 3,调用后 i 的值为 3+8+1=12,m 的值为 12+4+1=17。所以程序运行结果第一次为 8,第二次为 17。

【例 5-13】 外部变量和外部函数。

该程序有两个源文件,其中存放 main()函数的文件名为 5-13-1.c,存放子函数的文件名为 5-13-2.c。

源程序 5-13-1.c:

```
#include<stdio.h>
#include"5-13-2.c"
int a;
extern void fun();
int main()
{
    a=35;
    printf("main()函数中 a=%d\n",a);
    fun();
    printf("调用 fun()函数后,main()函数中 a=%d\n",a);
}
```

源程序 5-13-2.c:

```
#include<stdio.h>
extern int a;
void fun()
{
    a=48;
    printf("fun()函数中外部全局变量 a=%d\n",a);
}
```

运行结果:

```
main()函数中 a=35
fun()函数中外部全局变量 a=48
调用 fun()函数后,main()函数中 a=48
```

程序说明:

(1) 本程序主要了解外部函数和外部变量的使用方法。注意,外部函数和外部变量都是将已定义的函数或变量在该位置重新声明一下,而不是重新定义。因为是两个文件,所以

需要在包含 main()函数的文件 5-13-1.c 中将另一个源文件 5-13-2.c 包含到该文件中才能运行。包含命令为：#include"5-13-2.c"。而语句“extern void fun();”是对另一个源文件的 fun()函数进行声明，这样才能在本文件的 main()函数中使用。

(2) 在 main()函数中对全局变量 a 赋值为 35，然后输出该变量值，之后调用 fun()函数。在源文件 5-13-2.c 中对 5-13-1.c 中的全局变量 a 进行了声明(不是重新定义一个新变量 a)，然后为其重新赋值 48，该值也改变了 main()函数中 a 的值(因为是同一个变量)。返回 main()函数中重新输出 a 值，发现 a 值也变成了 48。

5.2.4　内部函数和外部函数

根据函数能否被其他源程序文件调用，将函数分为内部函数和外部函数。

1. 内部函数

内部函数是指一个函数只能被它所在文件中的其他函数调用。在定义内部函数时，可使用 static 进行修饰。其一般格式如下：

```
static  类型标识符  函数名(形参列表){函数体}
```

例如：

```
static  float  max(float a,float b)
{
    …
}
```

使用内部函数，可以使该函数只限于它所在的文件，即使其他文件中有同名的函数也不会相互干扰，因为内部函数不能被其他文件中的函数所调用。

2. 外部函数

外部函数是指在一个源程序文件中定义的函数除了可以被本文件中的函数调用外，还可以被其他文件中的函数调用。在定义外部函数时，可使用关键字 extern 进行修饰。其一般格式如下：

```
extern  类型标识符 函数名(形参列表)
```

例如：

```
extern char del_str(char r1)
{
    …
}
```

名师点睛

(1) C 语言规定，若在定义函数时省略了 extern，则默认为外部函数。本书前面所用的函数都是外部函数。

(2) 在调用函数的文件中,一般要用 extern 声明所用的函数是外部函数,表示该函数是在其他文件中定义的外部函数。

5.3　预处理程序

5.3.1　宏定义

宏定义是用预处理命令 # define 施行的预处理,它分为两种形式:带参数的宏定义与不带参数的宏定义。

1. 不带参数的宏定义

不带参数的宏定义也称为字符串的宏定义,它用来指定一个标识符代表一个字符串常量。其一般格式如下:

```
#define  标识符  字符串
```

其中,标识符是宏的名字,简称宏;字符串是宏的替换正文,通过宏定义,使得标识符等同于字符串。

例如,definePI 3.14,其中,PI 是宏名,字符串"3.14"是替换正文。预处理程序将程序中以 PI 作为标识符出现的地方都用 3.14 替换,这种替换称为宏替换或宏扩展。这种替换的优点在于,用一个有意义的标识符代替一个字符串,便于记忆,易于修改,提高了程序的可移植性。

【例 5-14】 求 100 以内所有奇数的和。

```
#include<stdio.h>
#define N 100
int main()
{
    int i,sum=0;
    for(i=1; i<N; i=i+2)
    sum=sum+i;
    printf("sum=%d\n",sum);
}
```

经过编译预处理后将得到如下程序:

```
#include<stdio.h>
#define N 100
int main()
{
    int i,sum=0;
    for(i=1; i<100; i=i+2)
    sum=sum+i;
    printf("sum=%d\n",sum);
}
```

名师点睛

（1）对于用得比较多的常量或简单操作，只需要修改宏定义中 N 的替换字符串即可，不需要修改其他地方。

（2）宏定义在源程序中要单独占一行，通常“#”出现在一行的第一个字符的位置，允许#号前有若干空格或制表符，但不允许有其他字符。

（3）每个宏定义以换行符作为结束的标志，这与 C 语言的语句不同，不以“;”作为结束，如果使用了分号，则会将分号作为字符串的一部分一起替换。

（4）宏的名字用大小写字母均可，为了与程序中的变量名或函数名相区别和醒目，习惯用大写字母作为宏名。宏名是一个常量的标识符，它不是变量，不能对它进行赋值。

（5）一个宏的作用域是从定义的地方开始到本文件结束。也可以用#undef 命令终止宏定义的作用域。

（6）宏定义可以嵌套。例如，#define　PI　3.14 #define　TWOPI　(2.0 * PI)，若有语句“s＝TWOPI * r * r;”，则在编译时被替换为“s＝(2.0 * PI) * r * r;”。

2. 带参数的宏定义

C 语言的预处理命令允许使用带参数的宏，带参数的宏在展开时，不是进行简单的字符串替换，而是进行参数替换。带参数的宏定义的一般格式如下：

```
#define　标识符(参数表)　字符串
```

例如，#define SUM(a,b)(a＋b)，其中，SUM 是宏名，a 和 b 是函数的形参，(a＋b)是计算两个参数之和的表达式。

【例 5-15】 带参数的宏定义，求两个数的和。

```
#include <stdio.h>
#define SUM(a,b)(a + b)
int main ()
{
    printf("两数之和为: %d",SUM(3,5));
}
```

运行结果：

```
两数之和为: 8
```

程序说明：带参数的宏并不是将 3 和 5 的值传递给 a 和 b 进行求和，而是将“sum(3,5)”替换为“(3＋5)”，得出两数之和为 8。

名师点睛

（1）在宏定义中宏名和左括号之间没有空格。

（2）带参数的宏展开时，用实参字符串替换形参字符串，可能会发生错误。比较好的方法是将宏的各个参数用小括号括起来。

(3) 带参数的宏调用和函数调用非常相似,但它们毕竟不是一回事。其主要区别在于:带参数的宏替换只是简单的字符串替换,不存在函数类型、返回值及参数类型的问题;函数调用时,先计算实参表达式的值,再将它的值传递给形参,在传递过程中,要检查实参和形参的数据类型是否一致。而带参数的宏替换是用实参表达式原封不动地替换形参,并不进行计算,也不检查参数类型的一致性。

5.3.2　文件包含

文件包含是指把指定文件的全部内容包含到本文件中。文件包含控制行的一般格式如下:

```
#include"文件名"或#include<文件名>
```

例如:

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

在编译预处理时,就把 stdio.h 头文件的内容与当前的文件连在一起进行编译。同样,此命令对用户自己编写的文件也适用。

使用文件包含命令的优点:在程序设计中常常把一些公用性符号常量、宏、变量和函数的说明等集中起来组成若干文件,使用时可以根据需要将相关文件包含进来,这样可以避免在多个文件中输入相同的内容,也为程序的可移植性、可修改性提供了良好的条件。

【例 5-16】 假设有 3 个源文件 5-16-1.c、5-16-2.c、5-16-3.c,它们的内容如下所示,利用编译预处理命令实现多个文件的编译和连接。

源文件 5-16-1.c:

```
#include<stdio.h>
int main()
{
    int a,b,c,s,m;
    printf("\n a,b,c=?");
    scanf("%d,%d,%d",&a,&b,&c);
    s=sum(a,b,c);
    m=mul(a,b,c);
    printf("The sum is %d\n",s);
    printf("The mul is %d\n",m);
}
```

源文件 5-16-2.c:

```
int sum(int x,int y,int z)
{
    return (x+y+z);
}
```

源文件 5-16-3.c:

```
int mul(int x,int y,int z)
{
     return (x * y * z);
}
```

处理的方法是在含有 main()函数的源文件中使用预处理命令 #include 将其他源文件包含进来即可。这里需要把源文件 5-16-2.c 和 5-16-3.c 包含在源文件 5-16-1.c 中，则修改后 5-16-1.c 的内容如下：

```
#include <stdio.h>
#include "5 - 16 - 2.c"
#include "5 - 16 - 3.c"
int main()
{
     int a,b,c,s,m;
     printf("a,b,c = ?\n");
     scanf(" % d, % d, % d",&a,&b,&c);
     s = sum(a,b,c);
     m = mul(a,b,c);
     printf("The sum is % d\n",s);
     printf("The mul is % d\n",m);
     return 0;
}
```

运行结果：

```
a,b,c = ?
2,3,4
The sum is 9
The mul is 24
```

程序说明：文件 5-16-2.c 中的 sum()函数和文件 5-16-3.c 中的 mul()函数都被包含到文件 5-16-1.c 中，如同文件 5-16-1.c 中定义了这两个函数一样，所以说文件包含处理也都是模块化程序设计的一种手段。

名师点睛

（1）一个 include 命令只能指定一个被包含文件，若要包含 n 个文件，则需要用 n 个 include 命令。

（2）文件包含控制行可出现在源文件的任何地方，但为了醒目，大多放在文件的开头部分。

（3）#include 命令的文件名，使用双引号和尖括号是有区别的：使用尖括号仅在系统指定的“标准”目录中查找文件，而不在源文件的目录中查找；使用双引号表明先在正在处理的源文件目录中搜索指定的文件，若没有，再到系统指定的“标准”目录中查找。所以使用系统提供的文件时，一般使用尖括号，以节省查找时间；若包含用户自己编写的文件（这些文件一般在当前目录中），使用双引号比较好。

（4）文件包含命令可以是嵌套的，在一个被包含的文件中还可以包含其他的文件。

5.3.3 条件编译

一般情况下,源程序中所有的行都参加编译。但是有时希望对其中一部分内容只在满足一定条件时才进行编译,也就是对一部分内容指定编译条件,这就是“条件编译”。有时希望当满足某条件时对一组语句进行编译,而当条件不满足时则编译另一组语句。

条件编译命令有以下 3 种形式。

(1) 使用 # ifdef 的形式。

```
#ifdef  标识符
    程序段 1
#else
    程序段 2
#endif
```

此语句的作用是当标识符已经被 # define 命令所定义时,条件为真,编译程序段 1;否则条件为假,编译程序段 2。它与选择结构的 if 语句类似,else 语句也可以没有。

【例 5-17】 程序调试信息的显示。

```
#define  DEBUG
#ifdef   DEBUG
printf("x = %d,y = %d,z = %d\n",x,y,z);
#endif
```

程序说明:printf()函数被编译,程序运行时可以显示 x、y 和 z。在程序调试完成后,不再需要显示 x、y 和 z 的值,则只需要去掉 DEBUG 标识符的定义。

名师点睛

虽然直接使用 printf 语句也可以显示调试信息,在程序调试完成后去掉 printf 语句同样也达到了目的。但若程序中有很多处需要调试观察,增删语句既麻烦又容易出错,而使用条件编译则相当清晰、方便。

(2) 使用 # ifndef 的形式。

```
#ifndef  标识符
    程序段 1
#else
    程序段 2
#endif
```

此语句的作用是当标识符未被 # define 命令所定义时,条件为真,编译程序段 1;否则条件为假,编译程序段 2。与上面的条件编译类似,else 语句也可以没有。

(3) 使用 # if 的形式。

```
#if  表达式
    程序段 1
#else
```

```
    程序段 2
#endif
```

它的作用与 if-else 语句类似，当表达式的值为非 0 时，条件为真，编译表达式后的程序段 1；否则条件为假，编译程序段 2。

【例 5-18】 输入一行字母字符，根据需要设置条件编译，使之能将字母全改为大写输出或全改为小写输出。

```
#include <stdio.h>
#define LETTER 1
int main()
{
    int i = 0;
    char c;
    char str[25] = "I Love my country China";
    printf("String is: %s\n",str);
    printf("Change String is:");
    while((c = str[i])!= '\0')
    {
        i++;
        #if  LETTER
            if(c >= 'a'&&c <= 'z')
                c = c - 32;
        #else
            if(c >= 'A'&&c <= 'Z')
                c = c + 32;
        #endif
            printf("%c",c);

    }
    printf("\n");
    return 0;
}
```

运行结果：

```
String is: I Love my country China
Change String is: I LOVE MY COUNTRY CHINA
```

程序说明：在程序中，LETTER 通过宏定义值为 1(非 0)，则在编译时对第一个 if 语句进行编译，即选择将小写字母转换为大写字母。

名师点睛

事实上条件编译可以用 if 语句代替，但使用 if 语句目标代码比较长，因为所有的语句均要参与编译；而使用条件编译，只有一部分参与编译，且编译后的目标代码比较短，所以很多地方使用条件编译。

5.3.4 特殊符号处理

编译预处理程序可以识别一些特殊的符号,并对在源程序中出现的这些符号用合适的值进行替换,从而可以实现某种程度上的编译控制。常见的定义好的供编译预处理程序识别和处理的特殊符号如下所示(不同的编译器还可以定义自己的特殊函数的符号)。

FILE:包含当前程序文件名的字符串。

LINE:表示当前行号的整数。

DATE:包含当前日期的字符串。

STDC:若编译器遵循 ANSI C 标准,则它是个非 0 值。

TIME:包含当前时间的字符串。

名师点睛

符号中都是双下画线,而不是单下画线,并且日期和时间都是一个从特定的时间起点开始的长整数,并不是通常熟悉的年月日时分秒格式。

【例 5-19】 编译预处理中特殊符号的显示。

```
#include <stdio.h>
int main()
{
    printf("%d\n",__LINE__);          /* 显示所在行号 */
    printf("%s\n",__func__);          /* 显示所在函数 */
    printf("%s\n",__TIME__);          /* 显示当前时间 */
    printf("%s\n",__DATE__);          /* 显示当前日期 */
    printf("%s\n",__FILE__);          /* 显示当前程序文件名 */
    printf("%d\n",__STDC__);          /* 编译器遵循 ANSI C 标准时该标识被赋值为 1 */
    return 0;
}
```

运行结果:

```
4
main
13: 05: 02
Jul 4 2022
C:\user\5 - 19.c
1
```

【例 5-20】 演示 #line 的用法。

```
#line 7                /* 初始化行计数器 */
#include <stdio.h>
int main()
{
    printf("本行为第%d行!\n",__LINE__);
}
```

运行结果:

```
本行为第 10 行!
```

名师点睛

标识符_LINE_和_FILE_通常用来调试程序；标识符_DATE_和_TIME_通常用来在编译后的程序中加入一个时间标志，以区分程序的不同版本；当要求程序严格遵循 ANSI C 标准时，标识符_STDC_就会被赋值为 1。

5.4　常见错误分析

5.4.1　使用库函数时忘记包含头文件

在使用库函数时需要用 # include 命令将该原型函数的头文件包含进来，不少初学者容易忘记。

【例 5-21】 使用库函数，未包含头文件。

```
#include <stdio.h>
int main()
{
    int a = 4;
    printf("%f",sqrt(a));
    return 0;
}
```

编译报错信息如图 5-4 所示。

图 5-4　未包含头文件编译报错信息

错误分析：在使用 sqrt( ) 函数时，忘记包含头文件，应在程序的开头加上 # include < math.h >。

5.4.2　忘记对所调用的函数进行函数原型声明

若函数的返回值不是整型或字符型，并且函数的定义在主调函数之后，那么在调用函数前必须对函数进行原型声明。

【例 5-22】 未对调用函数进行原型声明。

```
#include <stdio.h>
float add(float x,float y);
```

```
int main()
{
     float a,b;
     printf("Please enter a and b:");
     scanf("%f%f", &a,&b);
     printf("the sum is: %f\n", add(a,b));
     return 0;
}
float add(float x,float y)
{
     float z = 0;
     z = x + y;
     return(z);
}
```

编译报错信息如图 5-5 所示。

图 5-5 未对调用函数进行原型声明编译报错信息

错误分析：add()函数是非整型函数，且调用在先，定义在后，因此，应在调用之前进行函数声明。可在 main()函数之前或 main()函数中加上函数原型的声明语句“float add(float x,float y);”。

5.4.3 函数的实参和形参类型不一致

函数一旦被定义，就可多次调用，但必须保证形参和实参数据类型一致。若实参和形参数据类型不一致，则按不同类型数值的赋值规则进行转换。

【例 5-23】 函数实参和形参类型不一致。

```
#include <stdio.h>
int sum(int i,int j)
{
     int k;
     k = i + j;
     return k;
}
int main()
{
     float a,b,c;
     printf("请输入两个实数:");
     scanf("%f%f",&a,&b);
     c = sum(a,b);
     printf("a + b = %f",c);
     return 0;
}
```

编译报错信息如图 5-6 所示。

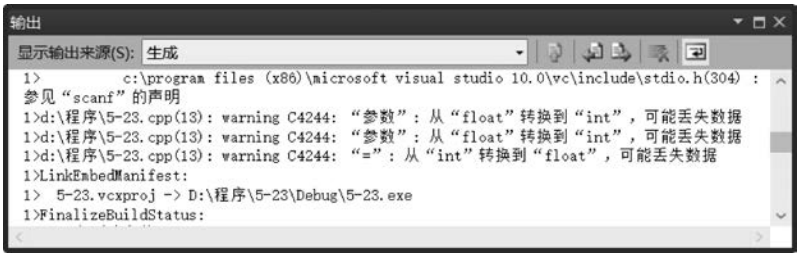

图 5-6　参数类型不一致编译报错信息

错误分析：实参 a 和 b 为 float 型，形参 i 和 j 为 int 型。在编译时，系统给出了“警告”。a 和 b 的值传递给 i 和 j 时，会按赋值规则处理，把小数部分删去，从而导致程序结果错误。

5.4.4　使用未赋值的自动变量

未进行初始化时，自动变量的值是不确定的，在使用时要特别注意。

【例 5-24】　未初始化变量导致错误。

```
#include<stdio.h>
int main()
{
    int i;
    printf("%d\n",i);
}
```

编译报错信息如图 5-7 所示。

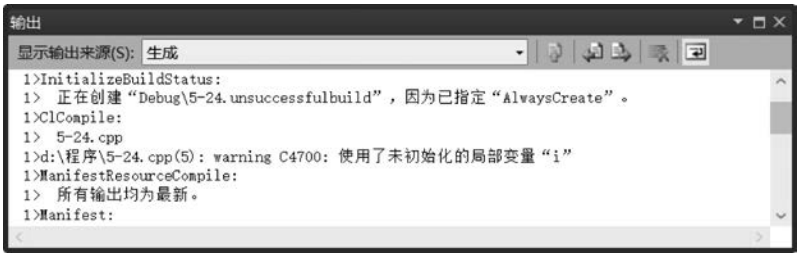

图 5-7　变量未初始化编译报错信息

错误分析：编译提示使用了未初始化的局部变量 i，程序运行结果－858 993 460 是一个不可预知的数。因此，在引用自动变量时，必须对其初始化或对其赋值。


视频讲解

5.5　分组实现函数功能应用实战

5.5.1　实战背景

随着软件系统的规模越来越庞大，软件开发过程中的分工越来越细，靠单兵作战来实现复杂系统越来越难。各种新知识、新技术不断推陈出新，需要团队合作。众人拾柴火焰高。

要求组织成员之间相互依赖、相互关联、共同合作,提高工作效率,依靠团队合作的力量创造奇迹。

5.5.2 实战目的

(1) 掌握函数定义及调用方式。

(2) 具备将较复杂的问题进行抽象分解成若干功能块的能力,并能编写相应的功能函数。

5.5.3 实战内容

将班级的学生分成3组,对输入不超过50个的整数,分别负责编写数据输入函数、数据排序函数和数据输出函数。

5.5.4 实战过程

```
#include <stdio.h>
#include <stdlib.h>
void inputdata(int a[],int n)
{
    int i;
    for(i=0; i<n; i++)
    {
        printf("请输入第%d个数据:",i+1);
        scanf("%d",&a[i]);
    }
}
void outputdata(int a[],int n)
{
    int i;
    for(i=0; i<n; i++)
        printf("%d ",a[i]);
    printf("\n");
}
void sort(int a[],int n)
{
    int i,j,k,t;
    for(i=0; i<n-1; i++)
    {
        k=i;
        for(j=i+1; j<n; j++)
            if(a[k]>a[j])
                k=j;
        if(k!=i)
        {
            t=a[i];
            a[i]=a[k];
            a[k]=t;
        }
    }
}
```

```
int main()
{
     int data[50],num;
     printf("请输入数据个数(1 - 50):");
     scanf(" % d",&num) ;
     inputdata(data,num);
     printf("排序前的数据为: \n");
     outputdata(data,num);
     sort(data,num);
     printf("排序后的数据为: \n");
     outputdata(data,num);
     return 0;
}
```

运行结果如图 5-8 所示。

图 5-8　技能实战运行结果

5.5.5　实战意义

通过实战，在掌握函数功能的同时，增强了学生之间团结合作意识，同伴之间互相帮助，各取所长，使得学习效率更高，进步更快。

在信息时代，学生们更需要拥有与他人合作的能力，这样才能在未来的工作中取得成功。任何人的成功、任何企业的成功，都集中体现了集体的智慧，都是团队合作的结果。因此，一个人的成功并不是真正的成功，一个团队的成功才是真正的成功。