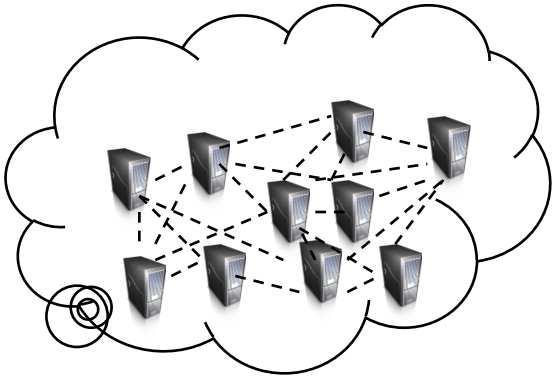
厦门大学非计算机专业本科生公共课

（2012-2013第2学期）



《C语言程序设计》

第6章 函数

林子雨

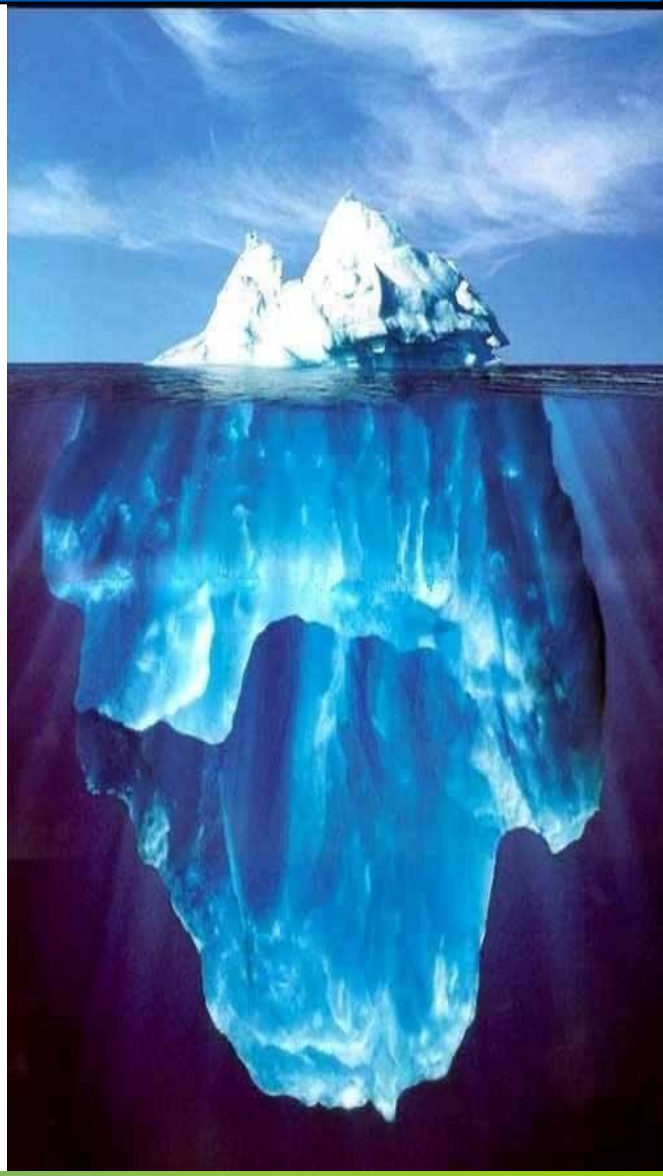
厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

个人主页：http://www.cs.xmu.edu.cn/linziyu

课程提要

- 第一章 绪论
- 第二章 C语言基础
- 第三章 结构化程序设计
- 第四章 选择结构
- 第五章 循环结构程序设计
- 第六章 函数
- 第七章 编译预处理
- 第八章 数组
- 第九章 结构体、共用体和枚举类型
- 第十章 指针



第6章 函数

6.1 函数的概念

- 在C语言中，函数是程序的基本单位
- C程序一般都是由一个main()函数和若干其他函数构成
- 每个函数都具有独立的程序模块
- 由main()函数调用其他函数，其他函数之间也可以相互调用
- 函数可以把程序中不需要了解的具体操作细节隐藏起来，使整个程序结构更加清晰，调试、修改和维护起来更加容易。
- 每个函数都可以被一个或多个函数调用任意次
- 函数的合理运用可以大大提高程序的可重用性，提高程序开发效率

6.1 函数的概念

标准函数库与头文件

- ANSI C的函数库是预先定义好的数百个函数的集合，比如，scanf()、printf()、sqrt()、abs()、puts()和gets()等都是属于函数库中的标准库函数
- ANSI C系统将所有函数的函数原型分成多组，通常每组放在一个头文件(*.h)中

ctype.h	包含测试字符某种属性的函数的函数原型
math.h	包含数学库函数的函数原型
stdio.h	包含标准输入输出库函数的函数原型
stdlib.h	包含数值与文本之间的转换、内存分配、随机函数和其他函数的函数原型
string.h	包含字符串处理函数的函数原型
time.h	包含时间和日期操作的函数的函数原型

6.2 函数的定义、声明和调用

6.2.1函数定义

•语法格式：

•函数类型 函数名([类型名 形式参数1, 类型名 形式参数2,…]) //函数首部

{

声明部分

语句部分 //函数体

}

•说明：

•（1）函数的第一行称为函数首部，指定了函数的类型、名字和参数列表。

•（2）定义函数时的函数名后小括号中出现的形式参数，称为“形参”：

– 定义函数时，形参没有具体值，只有在其他函数具体调用该函数，将具体的值（实参）传给形参时，它才会得到具体的值，因此，**形参相当于在函数中定义的变量**。

– 不管形参如何起名，都不会影响函数的功能，形参只是一个形式上的参数。

– 函数的形参可以有多个，也可以没有。若有多个，形参之间必须用逗号隔开；**若没有形参，也不能省略括号**。

– 在定义函数时，必须分别指定各形参的类型。

6.2.1函数定义

•语法格式：

•函数类型 函数名([类型名 形式参数1, 类型名 形式参数2,…]) //函数首部

– {

– 声明部分

– 语句部分 //函数体

– }

•说明：

•（3）函数类型是指函数返回值的数据类型，缺省类型为int型。

– 函数的返回值通过函数体中的return语句带回，形式为：return 表达式;

– 若定义的函数没有返回值，则选用void作为函数类型，并且函数体中最后的语句为return;，该语句可以省略。

•（4）函数体由被大括号括起来的多条语句构成，这些语句实现了函数的具体功能。

– 注意，在函数体内用到的变量，除形参以外必须在函数体的声明部分给出说明。

– 强调：函数体中不允许再定义函数，即函数的定义不允许嵌套。

6.2.1 函数定义

例6.2.1编写计算n！的函数。（有返回值的函数定义示例）

```
long  fac(int n)                    //定义名为fac的函数，用于计算n！。
{
    long  f=1 ;                     //变量f用于存放阶乘结果
    if(n<0) return 0;
    for(;n>0;n--)
    f*=n;
    return f;                       //以f中的值作为函数的返回值.
}
```

注意：

- fac函数可以单独编译，但是不能单独运行，必须被其他函数调用才能运行。只有“主函数”main()可以自己单独运行。
- 被别的函数调用的函数，称为“被调函数”，而调用别人的函数的函数称为“主调函数”。

6.2.1 函数定义

例6.2.2 编写在屏幕上显示8行“********”的函数。**

（无返回值无参数的函数定义示例）

```
    void printstar ( )    //函数名为printstar，无返回值，无参数
    {
        int i;
        for(i=1;i<=8;i++)
        printf("**********\n");
    }
```

（无返回值有参数的函数定义示例）

```
void printstar (int n)    //参数n是欲显示星号的行数
    {
        int i;
        for(i=1;i<=n;i++)
          printf("**********\n");
    }
```

函数概念区分

- 有参函数
- 无参函数
- 有值函数
- 无值函数

6.2.1 函数定义

- 可以直接运行的程序，演示函数定义和调用（无参数）

```
#include <stdio.h>
void printstar();
void main()
{
        printstar();
}
void printstar()    //函数名为printstar，无返回值，无参数
{
        int i;
        for(i=1;i<=5;i++)
        printf("**********\n");
}
```

6.2.1 函数定义

- 可以直接运行的程序，演示函数定义和调用（有参数）

```
#include <stdio.h>
void printstar(int);
void main()
{
        printstar(8);
}
void printstar(int n)    //函数名为printstar，无返回值，无参数
{
        int i;
        for(i=1;i<=n;i++)
        printf("**********\n");
}
```

6.2.2 函数的声明与函数原型

- 在ANSI C中，所有函数在调用（使用）前，除了必须定义外，还必须被声明；即函数和变量一样，也应该遵循“先声明后使用”。
- 函数声明格式：

  函数类型 函数名(形参类型1, 形参类型2,……);

  或者

  函数类型 函数名(类型名 形式参数1, 类型名 形式参数2,…);
- **注意：在形式上，函数原型等价于函数头后加分号；应当保证函数原型与函数头写法上的一致，即函数类型、函数名、参数个数、参数类型和参数顺序必须相同**
- **对函数进行声明的方法有两种：**
  - **直接将函数原型放在源文件的开始部分，即#include行之后，也可以放在主调函数的函数体开头的声明语句部分；**
  - **将函数原型存放在某个头文件(*.h)中，然后在调用函数的源文件的开始处利用文件包含命令#include将相应的头文件包含进来。**

6.2.3 函数的调用

函数调用的格式：

函数名([实参列表]);

说明：

- (1)当一个函数被调用时，其函数名后小括号中的“实参列表”中每一个表达式称为“实际参数”，简称“实参”。
- (2)如果是调用无参函数，则实参列表可以没有，但圆括号不能省略。如果实参列表包含多个实参，则各参数间用逗号隔开。实参与形参的类型、个数、顺序必须一致。
- 函数一旦被调用，它就从实际参数（如果有的话）中获取数据，完成相应的工作，然后返回主调函数的调用点。
- 由多个函数组成的程序中，函数定义的先后顺序与其被调用的先后次序无关，即函数的定义次序不影响其调用次序。

6.2.3 函数的调用

函数调用形式

（1）表达式调用

例：c=2*sqrt(27.0);

（2）作函数参数

例：m=min(a,min(b,c));

又如：printf(“%d”,min(a,b));

（3）调用语句

例：printf(“Xiamen University”);

又如：printstar();//见前面的实例

6.2.3 函数的调用

函数调用过程的机制

- 计算实参列表中各表达式的值；
- 执行控制流程从主调函数转移到被调函数，系统为被调函数的所有形参和函数体中局部变量分配变量空间；
- 将前面计算得到的实参值依次赋值给对应的形参变量；如果有必要，还会涉及自动类型转换；
- 从被调函数的函数体起始位置开始执行该函数的语句，直到遇到return语句或遇到函数体的花括号“}”为止；
- 计算return语句中的表达式（若有的话），当返回表达式的值的类型与函数类型不一致时，还要涉及自动类型转换；
- 释放为执行该函数调用在动态存储区中所创建的变量等，即自动销毁局部变量；
- 执行控制返回主调函数的调用点，并用返回值（若有的话）替代函数调用，接着执行主调函数中的后续语句。

6.2.3 函数的调用

例6.2.3 调用函数自定义阶乘函数求1！+2！+3！+……+n!

```
#include  <stdio.h>
void main()
{    int i,n ;
     long  sum=0 ;
     long  fac( int  n );
           //被调函数fac的定义在主调函数main后面，需要“先声明再使用”
     printf ("Enter n:") ;
     scanf("%d",&n);
     for(i=1;i<=n;i++)
           sum=sum+fac(i);                  //调用自编函数fac,计算i!。
     printf("%ld\n",sum);
}
long  fac(int n)                    //函数fac的定义
{    long  f=1 ;
     if(n<0) return 0;
     for(;n>0;n--)
           f*=n;
     return f;
}
```

6.3 函数间参数传递和返回值

调用函数时，通常在主调函数和被调函数之间存在数据传递。函数间的数据传递包括了两个方面：

（1）数据从主调函数传递给被调函数（通过函数的参数实现）。

（2）数据从被调函数返回到主调函数（通过函数的返回值实现）。

- 6.3.1 函数间参数传递
- 6.3.2 函数的返回值

6.3.1 函数间参数传递

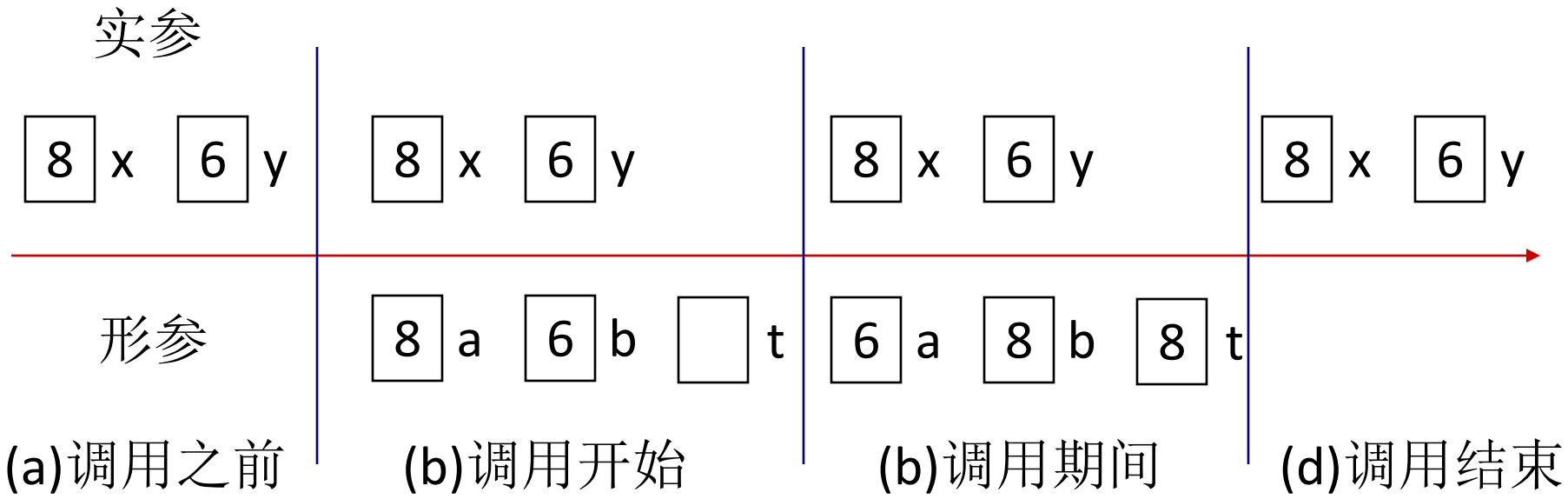
- 实参与形参是调用函数与被调用函数之间传递数据的通道。
- 在函数未被调用时，系统不对形参分配内存单元，但函数一旦被调用，系统会立即为其分配内存单元，并存放相应实参传递过来的数据；函数调用结束后，再释放其形参所占有的内存单元，其中的值也随之消失。
- 当普通变量做形参时，参数传递是单向的，即指允许将实参的值复制给相应的形参，而形参值在被调函数中的改变不会反过来影响主调函数中的实参。

6.3.1 函数间参数传递

- 例6.3.1 实参和形参之间单向的值传递

```
#include  <stdio.h>
void  sort(int a,int b)                          // 定义sort函数, a，b是形参
{
    int t;                                       //定义在sort函数内使用的变量
    printf("调用sort之初形参a，b的值:a=%d,b=%d\n",a,b) ;
    if(a>b) {t=a;a=b;b=t;}
    printf("返回前形参a，b的值:a=%d,b=%d\n",a,b) ;      //交换形参a，b中的值
}                                //调用完毕，释放形参a，b和变量t所占的存储单元
void main ( )
{
    int x=8, y=6 ;
    printf("调用sort前实参x，y的值:x=%d,y=%d\n",x,y) ;
    sort(x,y);                                   //调用函数sort，x和y是实参
    printf("返回后实参x，y的值:x=%d,y=%d\n",x,y);
 }                               //程序运行结束，释放实参x和y所占的存储单元
```

6.3.1 函数间参数传递



函数调用过程中实参与形参所占存储单元状况

6.3.2 函数的返回值

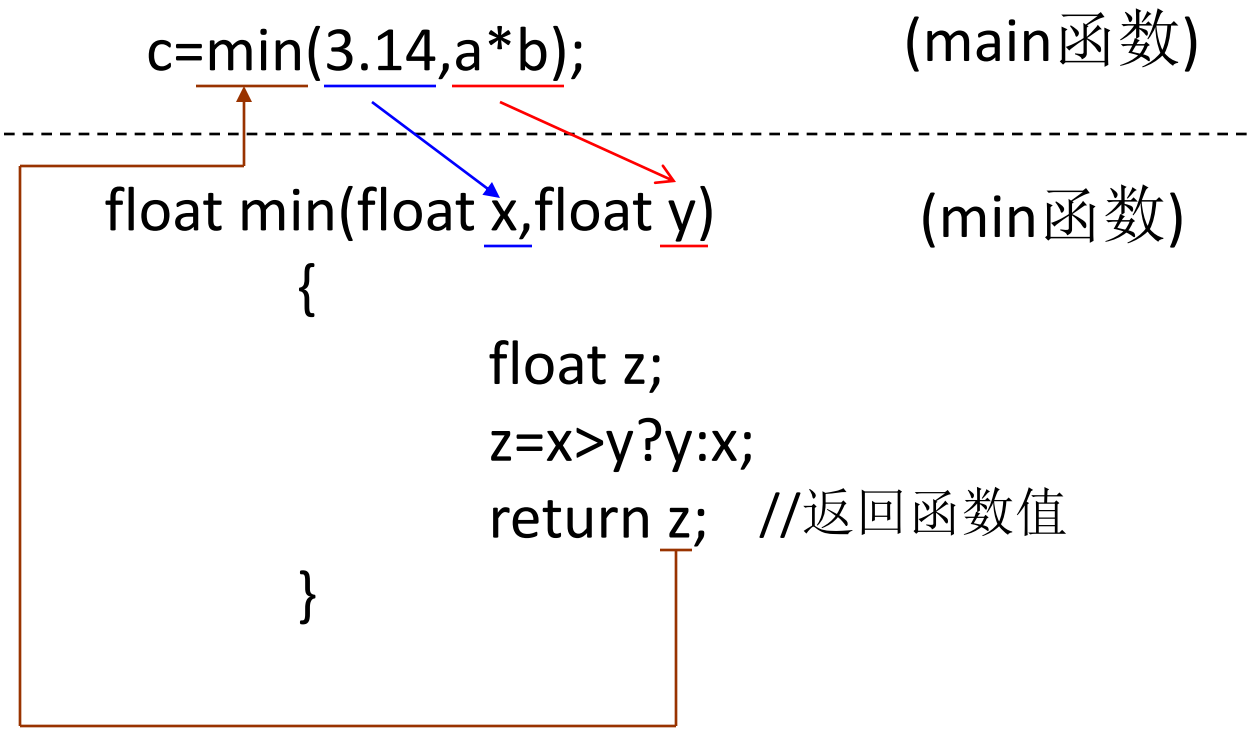
- 函数一旦被调用，它就从实际参数（如果有的话）中获取数据，完成相应的工作，然后返回主调函数的调用点。
- 作为返回操作的一部分，函数也能将计算结果（如果有的话），返回给主调函数。
- 函数可以有返回值，也可以没有返回值。函数的返回值称为函数值。
- 函数的返回值是通过return语句实现的：
  - 格式为：return 表达式; 或者 return(表达式);
  - 表达式类型最好与函数类型一致，如果不一致，则以函数类型为准，先把表达式值转换成函数类型后，再返回；
  - 可以有多个return语句放置在函数体的不同地方，遇到任何一个return语句，就立即结束函数调用，返回主调函数；
  - 对于无返回值的函数(功能是完成一些操作)，可以用“return;”语句，或者省略不写，当函数执行到最后遇到大括号“}”时，也会结束函数调用，自动返回主调函数。

6.3.2 函数的返回值

例6.3.2 函数间数据的传递。

```
#include  <stdio.h>
void main()
{    float min(float x,float y);              //函数原型声明
     float a,b,c;
     printf("Enter a,b:");
     scanf("%f,%f",&a,&b);
     c=min(3.14,a*b);                  //调用函数min，3.14和a*b是实参
     printf("min=%f\n",c);
}
float min(float x,float y)                  //定义有参函数min，形参为浮点型变量x，y
{    float z;                        //定义在min函数内使用的变量
     z=x>y?y:x;
     return z;                        //返回函数值z
}                              //调用完毕，形参x，y和变量z所占的内存单元被释放
```

6.3.2 函数的返回值


c=min(3.14,a*b);
(main函数)
float min(float x,float y)
(min函数)
{
float z;
z=x>y?y:x;
return z;
//返回函数值
}

6.3.2 函数的返回值

例6.3.3 寻找并输出11~9999之间的正整数m, 它满足m、m^2、m^3均为回文数。

```
#include<stdio.h>
int symm(long n)
{
    long i,g,new_num=0;
    i=n;
    while(i!=0)
    {       g=i%10;
            new_num=new_num*10+g;
            i=i/10;
    }
    return (new_num= =n);    //n是回文数，函数返回值为1，否则返回0。
}
main()
{   long m;
    for(m=11;m<=9999;m++)
    if(symm(m)&&symm(m*m)&&symm(m*m*m))
            printf("m=%ld\tm*m=%ld\tm*m*m=%ld\n",m,m*m,m*m*m);
}
```

6.3.2 函数的返回值

例：将一个大于6的偶数表示成两个素数之和。

```
#include<stdio.h>
int sushu(int x)
{
	int i;
	for(i=2;i<x;i++)
	{
		if(x%i==0)
			return 0;
	}
	return 1;
}
void main()
{
	int i,n;
	printf("输入一个偶数:");
	scanf("%d",&n);
	printf("输入的偶数为：%d\n",n);
	for(i=2;i<=n/2;i++)
	{
		if(sushu(i)&&sushu(n-i))
			printf("该偶数可以分解为两个素数之和：%d+%d=%d\n",i,n-i,n);
	}
}
```

6.6 变量的作用域与存储类别

6.6.1 变量的生存期和可见性

变量的生存期

- 变量的生存期是一个时间概念，是指变量占用存储空间的时间期限，就是变量从诞生到结束的这段时间。
- 在C程序中，有些变量的生存期很长，它们在程序的整个执行期间都独自占有固定的内存单元并保留其值；
- 有些变量的生存期是短暂的，有些变量会被反复地创建和释放，即在函数被调用时系统才给这些变量分配一块新的、临时存储空间（变量生命诞生）；当函数调用结束时，这些变量所占用的存储空间则立即被释放（变量生命结束，其值丢失），其生存期也结束。

6.6.1 变量的生存期和可见性

变量的可见性

- 变量的可见性，是在程序中的哪个部分可以引用该变量；
- 一个变量是“全程可见的”，是指该变量在整个程序（无论分成多少个源程序文件）范围内可访问；
- 一个变量是“在某个函数或程序块内是可见的”，是指该变量只在该函数或程序块内部可以被访问；
- 注意：有时变量即使存在也不一定可以被引用（不可见）；但是变量不存在则必不可被引用。

6.6.2 变量的作用域

- 变量的作用域是变量在程序正文内能被分辨或访问的那部分程序段；
- 每个变量都有一定的有效范围，超出这个范围，变量就失去作用；
- 根据作用域范围大小，将作用域分为：
  - 程序级
  - 文件级
  - 函数级
  - 块级
- 变量的作用域是一个空间的概念，是由变量定义的位置来确定的；
- 根据变量定义的位置不同，可将变量分为：局部变量和全局变量。

6.6.2 变量的作用域

局部变量

- 如果变量定义在一个函数（或复合语句）内部，则称该变量为局部变量，也称为内部变量；
- 局部变量只在本函数（或复合语句）范围内有效，即只在本函数（或符合语句）内才能使用它，在此函数（或语句块）以外是不能使用它的；
- 局部变量的作用域是函数级或者块级。

6.6.2 变量的作用域

```
main()
{
    int x,y;
    ……
    {
        int z;
        ……
        z=x>y?x:y;
        ……
    }
}

int fun(int z)
{
    int x,y;
    ……
}
```

z在此范围内有效

x,y在此范围内有效

x,y,z在此范围内有效

6.6.2 变量的作用域

例3.2.1复合语句的嵌套

```
#include<stdio.h>
void main()
{
    int x=1, y=10;
    {
        int x;
        x=5;
        {
            int a;
            a=y;
            y=20;
            printf("a=%d,x=%d\n",a,x);
        }
    }
    printf("x=%d,y=%d\n",x,y);
}
```

运行结果：

a=10,x=5

x=1,y=20

6.6.2 变量的作用域

全局变量

- 如果变量定义在所有函数外部，则称该变量为全局变量，也称为外部变量；
- 全局变量即可以定义在源文件的开头，也可以定义在两个函数的中间或源文件的尾部；
- 全局变量的作用域是从它定义的位置开始到本源文件结束。
- 全局变量可以被其作用域范围内的多个函数（若有的话）所使用和修改；
- 可以通过对全局变量使用引用声明（利用关键字extern），使其作用域扩展至整个源文件，甚至可以扩大到本程序的其他文件（若有的话）；
- 全局变量的作用域是文件级或者程序级；
- 全局变量在程序的整个执行期间都独自占有固定的内存单元，并保留其值，在整个程序的运行期（不管在函数内外），总是存在。

6.6.2 变量的作用域

例6.6.1 全局变量与局部变量同名。

```
#include  <stdio.h>
int x=10,y=6;    //x;y均为全局变量
min(int x, int y) //默认是int类型
{
      int z;
      z=x<y?x:y;
      return (z);
 }
void main( )
 {
      int x=3;
      printf (“%d\t”,min (x,y));
      {
              int x=1;   //x为复合语句内定义的局部变量
              y=2;
              printf ("%d\t",min (x,y));
      }
      printf ("%d\n",min (x,y));
}
```

形参x和y的作用域

局部变量x的作用域

局部变量x的作用域

运行结果：

3 1 2

6.6.2 变量的作用域

- **例6.6.1的说明：**
- （1）程序中重复使用x和y作变量名，要注意区分不同的x、y的含义和作用范围；
- （2）若全局变量和局部变量同名，则在它们的公共作用范围内，起作用的是局部变量，而全局变量被屏蔽（存在但不可见），不起作用。注意：变量的作用域讨论的是变量的有效范围，可见性是讨论变量是否可以被应用。
- （3）若同一函数不同程序块中有同名的变量，则在它们的公共作用范围内，起作用的是内层局部变量，外层的同名变量暂时挂起。

6.6.2 变量的作用域

例6.6.2分析下列程序的运行结果

```
#include<stdio.h>
int  x,y;                    //全局变量x、y的定义，x、y的作用域从此开始
void swap()
{
    int t;
    t=x; x=y; y=t;
}
void main()
{
    printf("please enter x and y:");
    scanf("%d%d",&x,&y);
    swap();
    printf("x=%d,y=%d\n",x,y);
}
```

若从键盘输入x=3,y=8
则输出结果为：
x=8,y=3

6.6.2 变量的作用域

例6.6.2的一个变种

```
#include<stdio.h>
int  x,y;                    //全局变量x、y的定义，x、y的作用域从此开始
void swap()
{
    int t;
    t=x; x=y; y=t;
}
void main()
{
    int  x,y;
    printf("please enter x and y:");
    scanf("%d%d",&x,&y);
    swap();
    printf("x=%d,y=%d\n",x,y);
}
```

若从键盘输入x=3,y=8
则输出结果为：
x=3,y=8

6.6.2 变量的作用域

- **一般不提倡使用全局变量：**
  - 耗费存储空间
  - 影响函数独立性
  - 降低函数“内聚性”
  - 程序可读性变差
- **选择变量的原则：**
  - 当变量只在某个函数或复合语句内使用时，不要定义成全局变量；
  - 当多个函数引用同一个变量时，在这些函数上面定义全局变量，而且定义部分尽量靠近这些函数。

6.6.3 变量的存储类别

- 变量的存储类别是指数据在内存中存储的方式，它能确定用户所定义变量在内存中的存储位置，从而决定变量的生存期。
- 变量的存储类别包括：自动存储类和静态存储类，数据分别存放在动态存储区和静态存储区。
  - **动态存储区**：是在函数调用过程中进行动态分配的存储单元，用来存放auto局部变量、形参及函数调用时的现场保护和返回地址等数据；
  - **静态存储区**：是在程序开始执行时就分配的固定存储单元，用以存放全局变量及static局部变量等数据。

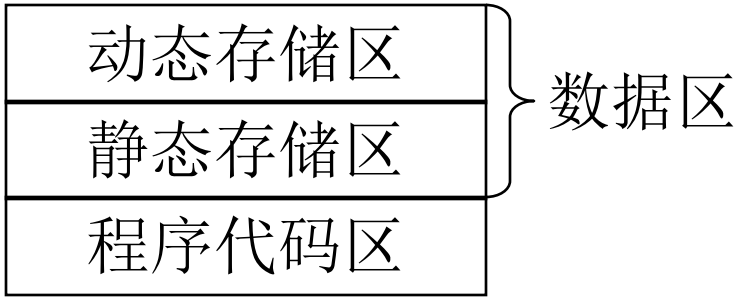


图6.6.1 C程序的内存分配图

6.6.3 变量的存储类别

- **自动存储**：对于自动存储的变量，当进入定义它的函数或复合语句时，系统立即自动为它分配存储单元，当函数或复合语句执行结束时，其所占有的存储单元自动被释放。
- 自动存储有两种存储方式：
  - auto变量（自动变量）被存放在内存的动态存储区；
  - register变量（寄存器变量）被存放在CPU的寄存器中；
- **静态存储**：对于静态存储的变量，在源程序编译的时候即被分配空间，而且在程序运行期间一直占用固定的存储单元，直到程序运行结束。
- 静态存储只有一种方式，就是在程序整个运行期间都将变量存放在静态存储区的固定存储单元中。

6.6.3 变量的存储类别

- **存储类型说明符**：告诉编译程序如何保存有关变量
  存储类型说明符 数据类型名 变量名表;
- 四个存储类型说明符：
  - auto（自动）
  - extern（外部）
  - register（寄存器）
  - static（静态）
- **说明**：
- （1）定义变量时，用户不但可以指定变量的名字和数据类型，而且还可以用存储类别说明符来指定变量的存储类别。

```
auto char c1,c2;   //说明c1,c2为自动变量
register int i,j;   //说明i,j为寄存器变量
static float x,y,z;   //说明x,y,z为静态变量
```

6.6.3 变量的存储类别

说明

- （2）变量的数据类型是其操作属性，它决定变量可参与的运算以及占内存的大小。而存储类别是变量的存储属性，它决定变量在内存中的存储位置，从而决定了变量值存在的时间，即变量的生存期。
- （3）关键字auto、register将定义的局部变量说明为自动存储类；static关键字将定义的变量说明为静态存储类。注意：局部变量既可以声明称自动存储类，也可以说明成静态存储类；而全局变量只能是静态存储类。

总之，变量的作用域是从变量在什么范围（即空间）内有效的角度，来将程序中的变量分为局部变量和全局变量。而变量的存储类别则从变量在内存中存储单元的分配模式及变量值的存在时间（即生存期）的角度，将变量分类自动变量、寄存器变量、外部变量和静态变量。

6.6.4 局部变量的存储类型

- 6.6.4.1 auto变量(自动变量)
- 6.6.4.2 static 局部变量(静态局部变量)
- 6.6.4.3 register变量(寄存器变量)

6.6.4.1 auto变量(自动变量)

- 在函数（或复合语句）内部定义，格式如下：
  [auto]　　数据类型名　变量名表;
  例：auto int a;
- 说明：
  - 由于auto可省略，所以，函数内所有未加存储类别说明符定义的局部变量均为自动变量。
  - 自动变量属于自动存储类。
  - 自动变量仅在定义该变量的函数体或程序块内有效（见例6.6.1）。
  - 对自动变量赋初值，不是在编译时进行的，而是在函数调用时进行，每调用函数一次，就重新给一次初值（见例6.6.3程序1）。如果自动变量在定义时未被初始化，则它的值是不确定的。
  - 自动变量是C程序中使用最多的变量。好处是：用之则建，用完即撤。节省存储空间。
  - 自动变量是局部变量；函数的形参也是一种自动变量。

6.6.4.2 static局部变量（静态局部变量）

- 在函数（或复合语句）内部定义，格式如下：

  static　　数据类型名　　变量名表;

  例：static int a=3;

- 与自动变量相同的是，静态局部变量的作用域也是从其定义的位置起，到函数体（或复合语句）结束为止(见例6.6.3程序2)。与自动变量不同的是：
  - （1）在整个程序运行期间，静态局部变量在内存的静态存储区中占据固定的存储单元，即在定义该变量的函数被调用结束后其所占据的存储单元并不释放，下次该函数再被调用时，静态局部变量仍使用原来的存储单元。由于并不释放这些存储单元，因此，上次调用结束时保存在这些存储单元中的值得以保留，并且静态局部变量的生存期是程序的整个运行期间。而自动局部变量则时而存在时而撤销。
  - （2）静态局部变量的初值是在编译时赋给的（仅赋值一次），即在程序运行时它已有初值。以后定义该变量的函数每次调用时，不再赋初值，而是使用上次函数调用结束时留下来的值。如果基本类型的静态局部变量在定义时未赋初值，编译时会自动被初始化为0。而对自动变量，系统不会自动为其初始化，因此，在其定义时，若无显式地赋予初值的话，其值是不确定的。

6.6.4.2 static局部变量（静态局部变量）

- 例6.6.3编一程序，观察比较自动局部变量与静态局部变量在调用过程中的情况。

程序1:

```
#include  <stdio.h>
void fun()
{           int x=1;
//每次调用时，都要重新初始化；
            x++;
            printf("x=%d\n",x);
}
main()
{           int i;
            for(i=0;i<3;i++)
            fun();
}
```

运行结果：
x=2
x=2
x=2

程序2：

```
#include  <stdio.h>
void fun()
{
            static int x=1;  //仅初始化一次
            x++;
            printf("x=%d\n",x);
}
main()
{           int i;
            for(i=0;i<3;i++)
            fun();
}
```

运行结果：
x=2
x=3
x=4

6.6.4.3 register变量(寄存器变量)

- 在函数（或复合语句）内部定义，格式如下：

  register　　数据类型名　　变量名表;
- 关键字register要求C编译器把定义的变量的值存储在CPU的寄存器中，而不像普通变量那样存储在内存中。这样，对于register变量的操作速度可以远远快于存储在内存中的普通变量。
- register只能用于局部变量和函数形参。对于循环次数较多的循环控制变量、循环体内反复使用的变量及形参等均可定义为寄存器变量。
- 现代编译器都有优化功能，自动将频繁使用的变量放在寄存器中，不需要编程者指定，因此，不需要使用register关键字。

6.6.5 全局变量的存储类型

- **全局变量**是在所有函数的外部定义的变量，其缺省类别是extern，因为又称为“外部变量”。
- 全局变量的作用域为从变量的定义的位置开始，到本源文件结束为止（见例6.6.5）。
- 全局变量属于静态存储类，编译时将全局变量分配在静态存储区，其生存期是程序的整个运行期间，即在程序的整个运行期间，它都在内存的静态存储区中占据固定的存储单元。
- 全局变量的初值是在编译时赋给的（仅赋值一次），如果全局变量在定义时未赋初值，编译时会自动被初始化为0（字符型为’\0’）。
- 全局变量可以通过存储类型说明符extern来扩展其作用域，使它可以被程序中各个函数所引用。如果使用static关键字，则它只可以被定义它的文件中的各函数所引用。

6.6.5 全局变量的存储类型

- 如果全局变量不在文件开头定义，其有效范围只限于定义处到文件尾。如果在定义点之前的函数想引用该全局变量，则应该在引用之前用关键字extern对该变量引用性声明，以告诉编译器该变量在本文件的某处已经被定义。

例6.6.5 将全局变量的作用域扩展到单个源程序文件（作用域向上扩展示例）。

```
#include<stdio.h>
extern int  x,y;        //外部变量x、y的声明，将x、y作用域向上扩展到该位置
//也可以写成：extern x,y;
main()
{           int min(int a,int b);
            printf("%d"，min(x,y));
}
int  x=111,y=10;        //外部变量x、y的定义，x、y的作用域从此开始
int min(int a,int b)     //定义min函数
{           int c;
            c=a<b?a:b;
            return(c);
}
```

6.6.6 内部函数和外部函数

- 在C语言中，根据函数能否被其他文件中的函数所调用，将函数分为内部函数和外部函数。

  - 内部函数：又称为“静态函数”，只能被本文件中其他函数调用，不能被其他文件中的函数所调用。定义格式如下：

    static 类型标识符 函数名(形参列表) {函数体}

    例：static int fun(int a) {…;}

  - 外部函数：不仅能被本文件中其他函数所调用，而且能被其他文件中的函数所调用。定义格式如下：

    [extern] 类型标识符 函数名(形参列表) {函数体}

    例：int fun(int a) {…;}

6.6.6 内部函数和外部函数

例6.6.7 分析下面程序的运行结果，体会不同存储类别的差别（用外部函数实现）。

file1.c（文件1）

```
#include<stdio.h>
int x=100;                                          //全局变量
void main()
{
      extern void fun1(void);                                   //函数原型
      extern void fun2(void);                                   //函数原型
      extern void fun3(void);                                   //函数原型
       int  x=6;                                          //main函数的局部变量
       printf("主函数main外层程序块局部变量x的值为%d\n",x);
     {          int x=8;                                    //复合语句的局部变量
                printf("主函数main内层程序块局部变量x的值为%d\n",x);
      }
      printf("主函数main外程序块局部变量x的值为%d\n",x);
      fun1();                                           //函数fun1拥有自动变量x
      fun2();                                           //函数fun2拥有静态局部变量x
      fun3();                                           //函数fun3使用全局变量x
      fun1();                                         //函数fun1对自动变量x重新初始化
      fun2();                                  //静态局部变量x保持了上次调用结束时的值
      fun3();                                   //全局变量x也保持其值
      printf("主函数main外程序块局部变量x的值为%d\n",x);
}
```

6.6.6 内部函数和外部函数

file2.c（文件2）

```
#include <stdio.h>
void fun1(void)
{    int x=10;          //每次调用函数fun1时都会对变量x初始化
      printf("进入函数fun1后其局部变量x的值为%d\n",x);
     x--;
      printf("退出函数fun1前其局部变量x的值为%d\n",x);
}
void fun2(void)
{    static int x=20;   //只在首次调用函数fun2时对静态局部变量x初始化
     printf("进入函数fun2时其静态局部变量x的值为%d\n",x);
     x--;
     printf("退出函数fun2时其静态局部变量x的值为%d\n",x);
}
```

file3.c(文件3)

```
#include <stdio.h>
extern int x;
void fun3(void)
{    printf("进入函数fun3时全局变量x的值为%d\n",x);
     x-=3;
      printf("退出函数fun3时全局变量x的值为%d\n",x);
}
```

附件：课程教材 （2012-2013第2学期）

- 《C语言程序设计（第2版）》
- 清华大学出版社，黄保和，江弋 编著
- 版次：2011年10月第2版
- ISBN:978-7-302-26972-4
- 定价：35元

附件：课程和班级网站 （2012-2013第2学期）

- 课程介绍网站：
http://dblab.xmu.edu.cn/node/124
- 班级网站：
http://dblab.xmu.edu.cn/node/347

附件：课程教师和助教（2012-2013第2学期）

主讲教师：林子雨

单位：厦门大学信息科学与技术学院计算机科学系
办公地点：福建省厦门市思明区厦门大学海韵园
E-mail:ziyulin@xmu.edu.cn
个人主页：http://www.cs.xmu.edu.cn/linziyu

助教：刘颖杰

单位：厦门大学计算机科学系2012级硕士研究生
E-mail: 376339705@qq.com
手机：18020761782

Department of Computer Science, Xiamen University, April 9, 2013