第九章 函数的进一步讨论

9.1 变量的作用域和存储类别

变量的作用域是指从空间上对变量的作用范围进行分类，分为全局变量和局部变量。其中全局变量的作用范围宽，局部变量的作用范围窄。

变量的存储类别是指从时间上对变量的存在时间长短进行分类，分为动态变量、静态变量。其中动态变量的存在时间短，静态变量的存在时间长。

9.1 变量的作用域和存储类别

9.1.1 变量的作用域

变量的作用域分为：全局变量、函数体内的局部变量、复合语句内的局部变量三种。

一、全局变量

定义位置：定义在函数体外面

作用域：从变量定义的位置开始到整个 .c 文件结束

注意事项：同一个 .c 文件中的全局变量不能重名

二、函数体内的局部变量

定义位置：定义在函数体内，函数的形参

作用域：仅在本函数内有效

注意事项：同一个函数体内的局部变量不能重名

9.1.1 变量的作用域

续上

三、复合语句内的局部变量

定义位置：定义在复合语句内部

作用域：仅在本复合语句内有效

注意事项：同一个复合语句内的局部变量不能重名

说明：

- 一般来说，“全局变量”的作用域范围较宽，其次是“函数体内的局部变量”，范围最小的是“复合语句内的局部变量”。
- 函数的形参属于函数体内的局部变量。
- 三种变量可以重名，当重名时，作用域小的变量自动屏蔽作用域大的变量

9.1.2 变量作用域练习题

（1）以下程序的运行结果是（ A ）

```
int  a = 3;
main( )
{   int s = 0;
  {
    int a = 5;
    s += a++;
  }
  s += a++;
  printf("%d\n", s);
}
```

定义全局变量 a

定义复合语句内的局部变量 a

引用复合语句内的局部变量 a

引用全局变量 a

A) 8　B) 10　C) 7　D) 11

9.1.2 变量作用域练习题

（2）以下程序的运行结果是（ B ）

```
int a = 2;
int f(int *a)
{  return ++(*a);  }
main( )
{  int  s = 0;
  {  int  a = 5;
     s += f(&a) ;
  }
   s += f(&a) ;
   printf("%d\n", s);
}
```

定义全局变量 a

定义复合语句内的局部变量 a

引用复合语句内的局部变量 a

引用全局变量 a

A) 10　B) 9　C) 7　D) 8

9.1.3 变量的存储类别

变量的存储类别分为： auto（动态型）、static（静态型）、register（寄存器型）、extern（外部型）。

一、auto 型 —— 动态变量

auto型变量只在一个时间段内有效。例如函数的形参、函数体内的局部动态变量。发生函数调用时，系统临时为这类变量分配存储空间，当函数调用结束时，这些变量的存储空间自动释放，变量的值也随之消失。

9.1.3 变量的存储类别

续上

二、static 型 —— 静态变量 重点

static型变量的生存期为程序运行的整个期间。这类变量一旦定义，系统就为之分配存储空间，在整个程序运行过程中，静态变量的存储空间不会释放，因此也称永久存储。

特点：

静态变量定义时如果没有赋初值，则初值是零.这一点与动态变量不同，动态变量定义时如果没有赋初值，则初值是随机数。

9.1.3 变量的存储类别

续上

三、register 型 —— 寄存器变量

register型变量是指存储空间分配在寄存器中的变量。这类变量能够提高程序运行的速度，属于动态变量范畴，变量使用完存储空间自动释放。

特点：寄存器变量仅限整型、字符型、指针型。

四、extern 型 —— 外部型变量

extern型变量是对变量引用的说明，不是定义变量。即如果某个 .c 文件需要使用另外一个 .c 文件中定义的全局变量，那么在引用之前要说明为 extern 型。

9.1.4 变量存储类别练习题

（1）以下叙述正确的是（ B ）

A) 全局变量的作用域一定比局部变量的作用域范围大　全局变量的作用域不一定总是大于局部变量

B）静态（static）类别变量的生存期贯穿于整个程序的运行期间

C）函数的形参都属于全局变量　函数的形参属于局部变量

D）未在定义语句中赋初值的auto变量和static变量的初值都是随机值　static变量未赋初值时初值是零

9.1.4 变量存储类别练习题

（2）以下只有在使用时才为该类型变量分配内存的存储类说明是（ B ）

A) auto和static　B) auto和register

C) register和static　D) extern和register

【分析】

- 只有在使用时才分配存储空间的变量属于动态变量范畴
- 变量的存储类别分为auto、static、register、extern，其中auto和register类别的变量都属于动态变量范畴。
- extern型是对变量的说明，而不是定义变量

9.1.4 变量存储类别练习题

（3）以下程序的输出结果是（ C ）

```
int  x = 3;
incre( )
{  staic   int  x = 1;
    x *= x+1;
    printf("  %d",  x);
}
main()
{  int  i;
   for (i=1; i<x; i++)    incre();
}
```

定义全局变量 / 引用静态 / 引用全局变量 x

- 第一次循环调用incre函数：静态变量x=1 □ x=2 □ 打印2
- 第二次循环调用incre函数：静态变量x=2 □ x=6 □ 打印6

A) 3 3　B) 2 2　C) 2 6　D) 2 5

9.1.4 变量存储类别练习题

（4）以下程序的输出结果是（ A ）

```
fun(int  a)
{   int   b = 0;
   static int  c = 3;
   b++;   c++;
   return(a+b+c);
}
main()
{   int  a = 2,  i;
 for(i=0; i<3; i++)    printf("%d  ",  fun(a));
}
```

本题的关键

- 第一次循环调用fun函数：a=2,b=0,c=0 □ b=1,c=4 □ 返回7 于是主函数打印7
- 第二次循环调用fun函数：a=2,b=0,c=4 □ b=1,c=5 □ 返回8 于是主函数打印8
- 第三次循环调用fun函数：a=2,b=0,c=5 □ b=1,c=6 □ 返回9 于是主函数打印9

A) 7 8 9　B) 7 9 11

C) 7 10 13　D) 7 7 7

9.2 编译预处理命令

在C语言中，凡是以 “#” 开头的行都称为编译预处理命令行，命令行的末尾不能加“;”号。

本节介绍 “#define” 和 “#include” 两个编译预处理命令。

9.2.1 宏替换 #define

一、不带参数的宏替换

一般形式为： #define 宏名 替换文本

说明：

- “替换文本”可以是常量、表达式等。例如：#define PAI 3.14159
- 替换文本中可以包含已经定义过的宏名。
- 如果替换文本中没有小括号，在宏展开时不能随意添加小括号。
- 宏名习惯用大写字母, 宏定义一般写在文件开头

9.2.1 宏替换 #define

（1）以下程序的输出结果是（ C ）

```
#define  N  5
#define  M1  N+2
#define  M2  M1*3
main()
{   int  i;
 i = M1 + M2;
 printf("%d\n",  i);
}
```

M1 + M2 展开为

N + 2 + N + 2 * 3

5 + 2 + 5 + 2 * 3

18

A) 10　B) 28　C) 18　D) 30

9.2.1 宏替换 #define

二、带参数的宏替换

一般形式为：#define 宏名(参数表) 替换文本

说明：

- 书写时，宏名和小括号必须紧挨着；小括号不能省略。
- 参数表中只有参数名称，没有类型说明 。 重点
- 如果替换文本中有括号，则进行宏替换时必须有括号；反之，如果替换文本中本身没有括号，则宏替换时不能随便加括号 。

9.2.1 宏替换 #define

（2）以下程序的输出结果是（ D ）

```
#define    M(x, y, z)    x*y+z
main()
{   int  a=1, b=2, c=3;
   printf("%d\n", M(a+b, b+c,  c+a));
}
```

A) 19　B) 17　C) 15　D) 12

M(a+b, b+c, c+a) 展开为：

a + b * b + c + c + a

代入数值为：1+2*2+3+3+1 = 12

9.2.1 宏替换 #define

（3）以下程序的输出结果是（ A ）

```
#define   F(X,Y)   (X)*(Y)
main()
{   int  a=3, b=4;
  printf("%d\n", F(a++, b++));
}
```

A) 12　B) 15　C) 16　D) 20

F(a++, b++) 展开为：

(a++) * (b++)

代入数值为: (3++) * (4++) = 12

9.2.2 文件包含 #include

文件包含的一般形式为：#include <文件名> 或者 #include “文件名”

功能：把指定的文件插入该命令行位置取代该命令行，从而把指定的文件和当前的源程序文件连成一个源文件。

说明：

（1）一个include命令只能指定一个被包含文件，若有多个文件要包含，则需用多个include命令。

（2）文件包含允许嵌套，即在一个被包含的文件中又可以包含另一个文件。

9.2.2 文件包含 #include

续上

（3）include命令中的文件名即可以用双引号括起来，也可以用尖括号括起来，但是这两种形式是有区别的。

- 用尖括号括起来，系统将直接按照系统指定的标准方式到有关目录中去寻找。库文件名一般就使用尖括号括起来。
- 用双引号括起来，是指系统先在源程序所在的目录查找指定的包含文件，如果找不到，再按照系统指定的标准方式到有关目录中去找。用户自定义的文件名一般就使用双引号括起来。

9.2.2 文件包含 #include

【练习题】程序中头文件type1.h的内容是

```
#define  N    5
#define  M1  N*3
```

程序如下：

```
#include  "type1.h"
#define   M2   N*2
main()
{   int  i;
  i = M1 + M2;       printf("%d\n",  i);
}
```

包含用户自定义文件名 typ1.h

M1+M2 展开为: N*3+N*2 □ 5*3+5*2 = 25

程序编译后运行的输出结果是（ C ）

A) 10　B) 20　C) 25　D) 30

9.3 main函数的参数

我们之前编写程序时，main函数后面的一对小括号中是空的，没有参数。实际上，在运行C程序时，系统可以通过命令行把参数传递给main函数，main函数通常可用两个参数：

```
void main(int argc,  char **argv)
{     ……     }
```

说明：

- 其中argc和argv是两个的参数名，也可由用户自己命名，但它们的类型固定。
- agvc必须是整型，它保存从命令行输入命令的个数，每个命令是一个字符串，输入时用空格隔开。

9.3 main函数的参数

- argv是一个字符型指针数组，数组中的每个元素是一个指针变量，用来指向命令行中的每一个命令（每个命令都是一个字符串），因此，第二个参数argv还可以定义为 char *argv[ ]。

- 当在操作系统的命令行中执行源程序编译连接后生成的.exe文件时，即可以向main函数传递参数。

- 要求输入的各个命令中，第一个命令字符串必须是文件的名称。例如生成的可执行文件全名为“myc.exe”，则命令行中的第一个命令必须是 myc

9.3 main函数的参数

假定以下程序经编译连接后生成可执行文件prog.exe，如果在此可执行文件所在目录的DOS提示符下键入：prog ABCD HIJK<回车>，则输出结果为（ D）

```
main( int argc, char *argv[ ]))
{     while(--argc > 0)
            printf("%s", argv[argc]);
      printf("\n");
}
```

A) ABCD　　B) HIJK

C) ABCDHIJKL　　D) HIJKABCD

9.3 main函数的参数

【分析】根据main函数参数的原理可知，该题从命令行输入的命令字符串共有3个，第一个字符串是“prog”、第二个字符串是“ABCD”、第三个字符串是“HIJK”。那么main函数的两个参数获得值为：

argc = 3；指针argv[0]指向字符串“prog”；指针argv[1]指向字符串“ABCD”； argv[2]指向字符串“HIJK”。因此，while循环过程如下：

（1）“--argc” 表达式的值为2，于是打印指针 argv[2]指向的字符串——HIJK；

（2）“--argc” 表达式的值为1，于是打印指针 argv[1]指向的字符串——ABCD；

难点 **9.4 函数的递归调用** 重点

一个函数如果自己调用自己，称为直接递归调用。如果A函数调用B函数，B函数又调用A函数则称为间接递归调用。

在递归调用中， 主调函数又是被调函数。递归调用程序设计应注意设置合理的结束条件。

例如：

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

这个函数是一个递归函数。但是运行该函数将无休止地调用其自身，这当然是不正确的。为了防止递归调用无终止地进行， 必须在函数内有终止递归调用的手段。常用的办法是加条件判断， 满足某种条件后就不再作递归调用，然后逐层返回

9.4 函数的递归调用

[举例]用递归的方法求 x!

```
int  fun( int n)
{
  if(n == 1)
    return (1);
  else
    return (n * fun(n-1));
}
main()
{
   int x = 5, y;
   y = fun(x);
   printf("%d! = %d\n", x, y);
}
```

递归过程：

首先一次一次地调用函数自己，直至递归的结束条件为止；然后一次一次地返回，直至返回到第一次函数调用的位置。

有几次调用过程，就相应的有几

9.4 函数的递归调用

练习题

以下程序的输出结果是 (A)

```
int  fun( int n)
{    long s;
 if(n==1 || n==2)   s = 2;
 else    s = n - fun(n-1);
 return s;
}
main()
{    printf("%d\n", fun(3));   }
```

A) 1　　B) 2　　C) 3　　D) 4

主函数调用fun(3)
打印 1

执行fun(3)的过程：
形参n=3，执行else分支，s = 3-fun(2)
= 3 - 2 = 1，返回1

执行fun(2)的过程：
形参n=2，执行 if 分支，s=2，返回2

9.5 函数指针

在C语言中，函数名代表该函数的入口地址，因此可以定义一种指向函数的指针变量来存放这种地址，这样的指针变量称为指向函数的指针，简称函数指针。利用该指针可以调用它所指向的函数。

函数指针的定义及赋值

定义一个子函数

```
double fun(int a, char b) {  … }
```

定义函数指针

```
void main(void)
{  double (*pf)(int, char);   double  y;
   pf = fun;
   y = pf(10, 'a');
}
```

为函数指针赋值

等价于 y=fun(10,'a');

利用函数指针调用函数

9.5 函数指针

说明：

（1）以上例子中，定义了一个函数指针pfun，使用的定义语句是： double (*pfun)(int, char);

对函数指针定义的说明：

① “*pfun” 两侧的小括号不能缺省；

② “double” 表示函数指针pfun所指向的子函数返回值是double型。

③ “(int, char)”表示函数指针pfun所指向的子函数的两个形参的类型。

（2）以上例子中，使用语句 pfun = fun; 将函数指针pfun指向一个子函数fun。由于函数名fun代表了该函数的入口地址，赋值语句表示将

续上

9.5 函数指针

《

需要指出的是：为函数指针赋一个函数名，