**程序设计教学大纲**

第13节

2018年3月25日星期日

1. 函数的使用意义

1.如果一组操作需要被反复使用，那么可以将它们定义成一个函数。

2.为了便于简化、结构清晰地编程，需要将一部分特定功能的语句集合到一起。

1. 函数的定义

数据类型 函数名（形式参数表）

{

函数体

｝

//执行语句，最后一定是返回语句，返回的数值与开始定义函数的数据类型必须是一致的或兼容的，如果不需要返回数值，可以省却return语句，也可使用空return;语句。函数体如果为空，则该函数为空函数。

1.关于函数的定义有如下说明：

（1）函数的数据类型是函数的返回值类型（若数据类型为 void ,则无返回值）。

（2）函数名是标识符，一个程序中除了主函数名必须为main外，其余函数的名字按照标识符的取名规则可以任意选取，最好取有助于记忆的名字。

（3）形式参数（简称形参）表可以是空的（即无参函数）；也可以有多个形参，形参间用逗号隔开，不管有无参数，函数名后的圆括号都必须有。形参必须有类型说明，形参可以是变量名、数组名或指针名，它的作用是实现主调函数与被调函数之间的关系。

（4）函数不允许嵌套定义，但可以嵌套引用。

例1：定义一个函数，返回两个数中的较大数。

int max(int x,int y)

{ return x>y?x:y; }

例2：编写一个计算三个数和的函数。

Int Sum3(Int a,int b,int c)

{ Int sum;

Sum=a+b+c;

Return sum;

}

2.函数的定义有两种方法：

1. 在定义变量区，像定义变量一样将函数的形式定义。具体的函数内容可在后面定义。

如：#include <iostream>

Using namespace std;

Int a();

Int b()

Int main()

{ ……

a;

b;

……

}

Int a(int i,int j)

{……}

Int b(int i,int j)

{……}

1. 在定义变量区，直接将函数定义完整。

如：#include <iostream>

Using namespace std;

Int a(int i,int j)

{……}

Int b(int m,int n)

{……}

Int main()

{

……

a;

b;

……

}

3.函数的形式

（1）无参函数——无形式参数，一般也不会带返回值，所以定义时用VOID类型定义函数。

（2）有参函数——带形式参数列表，一般会有返回值。

（3）空函数——用于预留扩展函数用。

1. 函数的引用

注意：被引用的函数必须在引用处前定义。在主函数或某一子函数中直接引用被引用函数的名字+（形式参数列表）即可，因为被引用函数有一个返回值return …调用函数时要注意该返回值的类型，如：

K=a(i,j)/b(m,n) //函数直接在表达式中进行调用

a函数为返回i与j的最大值，b函数为返回m与n的最小值，它们在定义函数时要注意返回值的类型与函数名前面的类型要一致。

另外，函数调用时，会产生参数的传递，调用时的形式参数会将值一一对应给相应顺序和位置的实际参数，如：

Int sum3(int a,int b,int c)

{……}

Sum(3,4,5)

Sum3函数在被调用时，会由调用时的实际参数变量的值直接传递给参数列表中相应位置的形式参数变量，相当于a=3 b=4 c=5

练习1： 要求键盘输入两个数，返回两个数的平均数。

练习2：编写求两个数的最大公约数GCD()函数和最小公倍数LCM()函数。

练习3：要求计算1!+2!+3!+4!+5!+6!+7!+8!+9!+10!

——递归调用——

练习4：（递归调用）求费波那契函数的值f(n)=f(n-1)+f(n-2)。

**注释：过程或函数调用它自身，称为递归。这种用递归来描述的算法称为递归算法。**

**函数直接调用自身称为直接递归，函数a 调用函数b，函数b又调用a，称为间接递归。**

**递归算法的两个必要条件：**

**(1) 递归就是在过程或函数里调用自身;**

**(2) 在使用递增归时，必须有一个明确的递归结束条件，称为递归出口。**

完整程序示例：

#include <iostream>

using namespace std;

long long fibonacci(int n)

{ if (n==1||n==2)

return 1;

else

return fibonacci(n-1)+fibonacci(n-2);

}

int main()

{ int n; //输入数值过大，会耗费很长时间。

cout<<"请输入要求费波那契数列的项数（0-50）：";

cin>>n;

cout<<fibonacci(n)<<endl;

return 0;

}

练习5：汉诺塔问题，有A、B、C三根柱子。A柱子上按从小到大的顺序堆放了N个盘子，现在要把全部盘子从A柱移动到C柱，移动过程中可以借助B柱。移动时有如下要求：

（1）一次只能移动一个盘子；

（2）不允许把大盘放在小盘上边；

（3）盘子只能放在三根柱子上；

算法分析：当盘子比较多的时，问题比较复杂，所以我们先分析简单的情况：

如果只有一个盘子，只需一步，直接把它从A柱移动到C柱；

如果是2个盘子，共需要移动3步:

（1）把A柱上的小盘子移动到B柱，即A→B;

（2）把A柱上的大盘子移动到C柱，即A→C；

（3）把B柱上的大盘子移动到C柱，即B→C；

如果N比较大时，我们先考虑是否能把复杂的移动过程转化为简单的移动过程，如果要把A柱上最大的盘子移动到C柱上去，必须先把上面的N-1个盘子从A柱移动到B柱上暂存。

我们定义一个过程move，为了将n个盘子从a经过b移动到c上，可以调用过程move(n,a,b,c)。

按照上面的这种思路，就可以把N个盘子的移动过程分作3大步：

（1）把A柱上面的N-1个盘子移动到B柱，调用move(n-1,a,c,b)；

（2）把A柱上剩下的一个盘子移动到C柱，即为A→C；

（3）把B柱上面的N-1个盘子移动到C柱，调用move(n-1,b,a,c)；

其中N-1个盘子的移动过程又可按同样的方法分为三大步，这样就把移动过程转化为一个递归的过程，直到最后只剩下一个盘子，按照移动一个盘子的方法移动，递归结束。

定义第一个函数——将最上面的N-1个盘子，先从A柱上经C柱移到B柱上。

定义第二个函数——将最下面的盘子直接从第一个柱子上移动第三个柱子上，输出移动步骤。

Int main()

{初始条件准备；

调用第1个函数；

调用第2个函数；

调用第3个函数；

}

完整程序示例如下：

#include <iostream>

using namespace std;

int num=0; //统计移动总步数

void print(char a,char b)

{

cout<<a<<"柱-->"<<b<<"柱"<<endl;

return;

}

void hanoi(char a,char b,char c,int n)

{

if (n==1)

{

print(a,c);

num++;

}

else

{ hanoi(a,c,b,n-1);

print(a,c);

num++;

hanoi(b,a,c,n-1);

}

return;

}

int main()

{int n;

cout<<"请输入要挪动汉诺塔的层数（3-7）：";

cin>>n;

cout<<"盘子移动如下："<<endl;

hanoi('A','B','C',n);

cout<<"移动的总步数为："<<num<<endl;

return 0;

}