第5章 像搭积木一样搭建程序

5.1 习题解析

5.1 请联系生活中的例子,描述一下模块化思想。

参考答案:模块化思想使用非常广泛,在大规模的现代工业化生产随处可见。例如,计算机的制造过程就是模块化的生产过程。计算机的配件是模块化的配件,这些配件一般都有卡槽,通过连接线可以很容易地将它们连接在一起。不同公司生产的计算机配件的功能和性能可能是不同的,但是这些卡槽和连接线的接口一定是相同的。这样计算机的集成商通过选择性价比不同的计算机配件就可以快速地组装出一台性价比不同的计算机产品。

5.2 简要介绍库函数的作用,并举例介绍库函数的使用方法。

参考答案:C 语言的库函数并不是 C 语言本身的一部分,它是由编译程序根据一般用户的需要,编制并提供用户使用的一组程序。也就是说,C 语言库函数是别人实现的自定义函数,只不过这些常用的函数被放在了库中,供其他程序员使用。

C 的库函数极大地方便了用户,同时也补充了 C 语言本身的不足。在编写 C 语言程序时,使用库函数,既可以提高程序的运行效率,又可以提高编程的质量。

程序员使用库函数时,需要把声明库函数的头文件名用#include < >或#include " "加到程序的头部(尖括号或西文双引号内填写文件名)。例如,使用数学计算库函数时,需要包含 math. h,在程序头部需要书写#include <math. h>。

5.3 简要介绍形参与实参的概念,并说明它们之间数据的传递过程。

参考答案:形参全称为形式参数,它在函数定义中出现,可以被看作一个占位符。由于在函数定义的时候形参没有数据,只能等到函数被调用时才接收传递进来的数据,所以称为形式参数。

实参全称为实际参数,它是函数被调用时给出的参数,出现在函数调用过程中。实参包含了实实在在的数据,会被函数内部的代码使用,所以称为实际参数。

形参和实参的功能是传递数据,发生函数调用时,实参的值会传递给形参。即通过函数调用,主调函数可以将它所包含的实参变量的数据传递给被调函数的形参变量,并执行被调函数的函数体中的语句对形参变量的数据进行处理。

5.4 实现一个判别输入正整数是否为素数的函数。比如当函数的输入数据是“17”时,函数的输出为“1”,表示该数为素数;如果输入“20”,则输出为“0”,表示该数为合数。

参考程序:

```
// 函数说明:对代入的一个整数,判断其是否为素数
// 形式参数:n,整型,待判断的整数值
// 返回值:整型,若 n 为素数,为 1,若 n 为合数,为 0,若 n 为不合法的输入,为-1
int isPrime(int n)
{
    int i;
```

```
    int p = 1;

    if (n<=1)
    {
        return -1;
    }
    for(i=2;i*i<=n;i++)
    {
        if(n%i==0)
        {
            p = 0;
            break;
        }
    }

    return p;
}
```

解析：

（1）函数命名为 isPrime，因为只要判断一个正整数，所以参数为一个。

（2）函数内的 p，可以看作一个 flag（标志），默认为 1，默认输入的整数为素数。

（3）return -1 处的判断，是判断输入的数据是否合法，要求输入的数不能小于 1。

（4）for 循环用于寻找 n 的因子，如果能找一个 n 的因子，那么 p 置 0，表示 n 是一个合数，n 的因子找到一个即可，所以利用一个 break 中断循环，这样可以节省循环的时间。

5.5 根据摄氏温度和华氏温度的转换关系

$$C=\frac{5}{9}\times(F-32)$$

其中 C 表示摄氏度，F 表示华氏温度，实现一个从华氏温度到摄氏温度换算的函数。

参考程序：

```
// 函数说明：将华氏温度换算为对应的摄氏温度
// 形式参数：f，float 类型，华氏温度值
// 返回值：float 类型，换算的摄氏温度值
float TempTrans(float f)
{
    float c;
    c = 5.0*(f-32)/9;
    return c;
}
```

解析：

本题非常简单，但该函数包括了函数定义的基本要素，并且能够完成一个比较有意义的换算。

5.6 利用主教材 5.4 节中求距离和三角形面积的函数，实现一个求任意凸四边形面积的函数，函数的参数为 4 个顶点的坐标值。

参考程序：

```
// 函数说明：代入三角形的 3 个顶点坐标,计算三角形的面积,使用主教材例 5.22 中的函数
// distance 和 triangleArea
// 形式参数：
// x1,y1,float 类型,第一个顶点的坐标
// x2,y2,float 类型,第二个顶点的坐标
// x3,y3,float 类型,第三个顶点的坐标
// x4,y4,float 类型,第四个顶点的坐标
// 返回值：float 类型,四边形面积
float convexQuadArea(float x1, float y1, float x2, float y2, float x3, float
y3,float x4,float y4)
{
    float a,b,c;
    float s1,s2,s;

    a = distance(x1,y1,x2,y2);
    b = distance(x1,y1,x3,y3);
    c = distance(x2,y2,x3,y3);
    s1 = triangleArea(a,b,c);
    a = distance(x1,y1,x4,y4);
    b = distance(x1,y1,x3,y3);
    c = distance(x4,y4,x3,y3);
    s2 = triangleArea(a,b,c);
    s = s1+s2;

    return s;
}
```

解析：

如图 5.1 所示,任何一个凸四边形都可以划分成两个三角形。所以求凸四边形的面积就可以转换为求两个三角形的面积之和。参考代码体现了像搭积木一样搭建程序的思想。在参考代码中,多次反复调用 distance 函数和 triangleArea 函数,体现了模块的复用性。

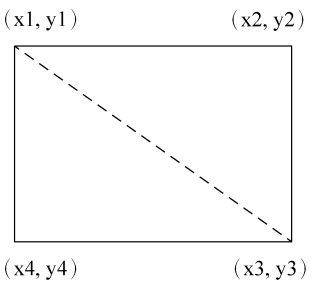

图 5.1 利用三角形计算凸四边形的面积

当然以上参考代码对 4 个顶点的输入要求比较严格,输入的时候按顺时针或逆时针的顺序输入,并且要保证输入的是一个凸四边形。

5.7 利用 5.6 题的方法,设计求任意凸多边形面积的函数,可以将多边形的顶点坐标利用数组进行存储,函数以数组作为参数。数组相关的知识参考主教材第 6 章。

参考程序：

```
// 函数说明：代入凸多边形顶点坐标和边数,计算凸多边形面积,使用主教材例 5.22 中
// 的函数 distance 和 triangleArea
```

```
// 形式参数:
// x[],float 类型数组,各个顶点的横坐标
// y[],float 类型数组,各个顶点的纵坐标
// n,整型,顶点数
// 返回值: float 类型,多边形面积
float convexArea(float x[],float y[], int n)
{
    float a,b,c;
    float s = 0.0;
    int i;
    for(i=1;i<n-1;i++)
    {
        a = distance(x[0],y[0],x[i],y[i]);
        b = distance(x[i],y[i],x[i+1],y[i+1]);
        c = distance(x[0],y[0],x[i+1],y[i+1]);
        s = s + triangleArea(a,b,c);
    }
    return s;
}
```

解析:

此题和习题 5.6 的思路基本一致。不过,凸多边形的边数可以是变化的,也就是说,在参数列表里不能采用习题 5.6 的方式输入。因此,在本题的参考代码里使用了主教材第 6 章数组的知识。x 数组存放所有顶点的横坐标,y 数组存放所有顶点的纵坐标。存放坐标的时候,一定要严格按照顺时针或逆时针的方式顺序存放。参数 n 表示凸多边形的边的条数,也是顶点的个数。

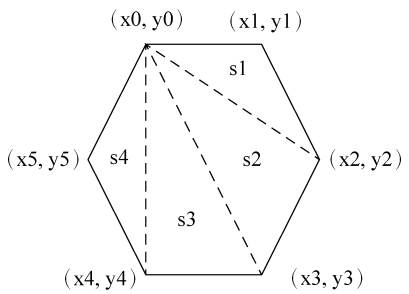

图 5.2 利用三角形计算凸多边形的面积

本题的关键思路是如何把凸多边形切割为多个三角形。如图 5.2 所示,任意 n 个顶点的凸多边形都可以按图 5.2 所示切分为 n-2 个三角形。注意图 5.2 的切分方法,是固定一个顶点,如(x0,y0)顶点(该顶点的坐标值分别存放在 x[0]和 y[0]中),连接不同的顶点构造小三角形的。按照这个规律可以很容易地判断出每个小三角形的顶点组成规律,并表示 3 条边为

(x[0],y[0]) 到 (x[i],y[i])

(x[i],y[i]) 到 (x[i+1],y[i+1])

(x[0],y[0]) 到 (x[i+1],y[i+1])

这样便于利用循环语句分别求解,然后进行累加即可。

5.8 利用递归的思想实现逆序输出整数。例如,设计一个函数 reverse,函数的输入为一个正整数,比如 123456,通过 reverse 函数输出为 654321。

参考程序：

```
// 函数说明：代入整数,在函数中逆序输出各位数字
// 形式参数：i，整型,待转换的整数
// 返回值：void,无返回值
void reverse(int i)
{
    int j = 0;
    if(i<10) printf("%d",i);
    else
    {
        j = i%10;
        printf("%d",j);
        reverse(i/10);
    }
}
```

解析：

本题的核心思想是首先输出一个数的最低位,也就是该数除以 10 的余数。比如 123456,最低位为 6,可以通过 123456%10 得到,参考代码中的 j 就是为了得到该余数而设的。当得到并输出该余数以后,再考虑对 12345 即 123456/10 的结果进行处理。这其实就是递归的思想。

本例中,递归的出口是处理的数小于 10 的时候,这时直接输出个位数即可。

5.9　汉诺塔问题求解。印度神话中有一个关于汉诺塔的故事,汉诺塔内有 3 个柱子 A、B、C,开始的时候 A 柱上有 64 个圆盘,盘子大小不等,大的在下,小的在上。有一个婆罗门想把圆盘从 A 柱上挪到 C 柱上,但是一次只能移动一个,并且要保证大盘在下,小盘在上,移动中可以利用 B 柱子。试编程求解移动的步骤。这是一道必须使用递归方法才能解决的经典问题。即使是用计算机来模拟移动过程,也需要很长很长的时间。在编程的时候,可以只移动 7 个圆盘。

参考程序：

```
#include <stdio.h>
// 函数说明：对 n 个圆盘的汉诺塔问题,输出解决方案
// 形式参数：
// n,整型,圆盘个数
// A,字符型,圆盘所处的初始柱子
// B,字符型,圆盘搬移过程中的辅助柱子
// C,字符型,圆盘搬移后的目的柱子
// 返回值：void,无返回值
void HanoiMove(int n, char A, char B, char C)
{
    if(n==1)                                    //递归退出条件
        printf("%c --> %c \n", A, C);
    else
    {
```

```
        HanoiMove(n-1,A,C,B);           //第一步,从 A 整体搬移 n-1 个圆盘到 B
        printf("%c --> %c \n", A, C);   //第二步,将最大的圆盘从 A 搬移到 C
        HanoiMove(n-1,B,A,C);           //第三步,从 B 整体搬移 n-1 个圆盘到 C
    }
}

// 调用该函数的主函数
int main(void)
{
    HanoiMove(7, 'A', 'B', 'C');
    return 0;
}
```

解析:

这是一个很古老的数学问题,而且只能用递归的方式来编程实现。

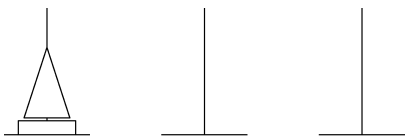

图 5.3　汉诺塔游戏的初始状态示意图

虽然这个游戏规定一次只能挪动一个圆盘,但是我们在考虑问题的时候,可以考虑整体挪动圆盘。

图 5.3 代表初始状态。3 个圆柱分别用 A、B、C 来表示。A 柱上的矩形代表最底下的一块圆盘,三角形代表上面 n-1 块圆盘。

此时在 main 函数内调用 HanoiMove 函数。该函数第一个参数的意思是要挪动多少个圆盘,参数 A 代表圆盘所处的初始的柱子,参数 C 代表最终的柱子,而参数 B 代表中间需要辅助的柱子 B。

在 main 函数调用该函数

```
HanoiMove(7, 'A', 'B', 'C');
```

意思是将 7 个圆盘从 A 柱上利用中间的 B 柱,最终挪到 C 柱上。

那这个工作是如何具体实现的呢?

可以这样去想:假设我们有办法把 A 柱上面 n-1 个圆盘挪到 B 柱上,然后就可以把最后一个最大的圆盘从 A 挪动到 C 柱子上。只要最后一个圆盘挪动成功了,那在 B 柱子上剩下的 n-1 个圆盘中的最大的圆盘是不是可以利用通用的方法,借助 A 柱挪动到 C 柱上呢?这样问题规模就可以从 n 减到 n-1。按照这个思路,继续执行,要移动的圆盘数会继续减为 n-2,n-3,一直到 1,问题就解决了!

所以汉诺塔问题的解决思路如下:

第一步,将 n-1 个圆盘从 A 柱搬移到 B 柱,此时 A 柱只留一个最大的圆盘,B 柱所有圆盘从小到大叠放在一起(见图 5.4)。

第二步,将最大的圆盘从 A 柱搬移到 C 柱,此时最大的圆盘已经搬移到位了(见图 5.5),只剩下其他 n-1 个圆盘要从 B 柱搬移到 C 柱。

第三步,将剩下的 n-1 个圆盘从 B 柱搬移到 C 柱,就完成了最终的任务。

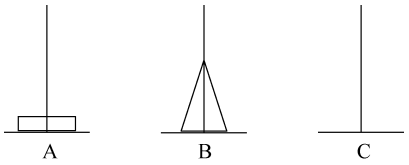

图 5.4 汉诺塔游戏第一步搬移结果示意图,最后一个圆盘保持位置不变,其他圆盘移至 B 柱

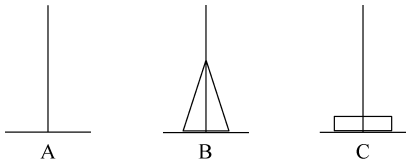

图 5.5 汉诺塔游戏第二步搬移结果示意图,最后一个圆盘移至 C 柱,其他圆盘移至 B 柱

由于大任务中包含着小任务的解决过程,因此可以利用递归方法实现程序。要注意,这里递归的结束条件是:除了最小的圆盘外,其他圆盘都搬移到 C 柱了,待搬移的圆盘只剩最后 1 个。因此在递归调用时,n=1 是结束条件。

5.10 利用函数实现复数的简单运算。复数的形式是 a+bi 的形式,其中 a 和 b 都为实数,a 称为复数的实部,b 称为复数的虚部。复数同样有加减乘除四则运算。现在考虑设计一个简单的复数运算的函数,比如复数的加法 ComplexAdd,假设将该函数定义为

```
double ComplexAdd(double a,double b,double c,double d)
```

其中,a 代表第一个复数的实部,b 代表第一个复数虚部的系数,c 代表第二个复数的实部,d 代表第二个复数虚部的系数。

试问该函数的定义能否返回两个复数相加的结果。如果这种定义方式不便于实现,请参考主教材第 6 章数组和第 7 章结构体的内容,设计一个可以满足需求的函数。

参考答案:不能返回两个复数相加的结果。如果还采用 a、b、c 和 d 分别表示两个复数的实部和虚部,可以采用间接访问的方式,利用指针实现加法计算。这时由于没有需要返回的值,所以函数类型可以定义为 void 类型。

```
// 函数说明:对两个复数计算它们的和
// 形式参数:
// a,double 类型,参与运算的第一个复数实部
// b,double 类型,参与运算的第一个复数虚部
// c,double 类型,参与运算的第二个复数实部
// d,double 类型,参与运算的第二个复数虚部
// r,double 指针类型,记录和的实部
// i,double 指针类型,记录和的虚部
// 返回值:void,无返回值
void ComplexAdd(double a,double b,double c,double d,double *r,double *i)
{
    *r = a + c;
    *i = b + d;
}
```

利用主教材第 7 章结构体的内容可以实现如下的函数。

```
typedef struct {                          //结构体类型的定义,参见第 7 章
    double real;                          //结构体成员类型和名称
```

```
    double imag;
} complex;
// 函数说明：对两个复数计算它们的和
// 形式参数：
// a,complex 结构体类型,参与运算的复数
// b,complex 结构体类型,参与运算的复数
// 返回值：complex 结构体类型,复数加法的结果
complex ComplexAdd(complex a, complex b)
{
    complex c;
    c.real = a.real+b.real;              //结构体变量的成员对应相加
    c.imag = a.imag+b.imag;
    return c;
}
```

解析：

如果按照题干的思路,所给出的函数首部是不能实现返回一个复数的功能的。因为一个函数只能返回一个值,而复数包括实部和虚部两部分。如果需要同时返回实部和虚部,必然要返回两个值。因此仅通过返回值的方式无法实现。

从函数的定义来看,本参考程序不是很复杂。这里用到了结构体的相关知识。通过结构体定义了一个新的数据类型 complex。该类型包括两个成员,分别表示复数的实部和虚部。定义新的结构体类型以后,函数返回值可以直接返回一个 complex 类型的数据。

5.11　设计一个函数计算两个日子相隔多少天。函数的参数可以为 6 个,分别代表起始的年、月、日和截止的年、月、日。注意,在计算过程中,可能需要多次计算某一个年份是否为闰年,所以可以首先设计一个判断某年为闰年的函数。

参考程序：

```
#include <stdio.h>
// 函数说明：代入一个整数,判断该年份是否为闰年
// 形式参数：year,整型,年份
// 返回值：整型,1 表示闰年,0 表示平年
int isLeap(int year)
{
    int i = 0;
    if(((year%4 == 0)&&(year%100 != 0))||(year%400) == 0)
    {
        i = 1;
    }
    return i;
}

// 函数说明：给定一个日期(年、月、日),计算当年已经过去的天数
// 形式参数：year,month,day,整型,年,月,日
// 返回值：整型,一年中已经经过的天数
int daysPastofYear(int year,int month,int day)
{
```

```
    int sum = 0;
    switch(month)
    {
        case 1:sum = day;break;
        case 2:sum = 31 + day;break;
        case 3:sum = 59 + day;break;
        case 4:sum = 90 + day;break;
        case 5:sum = 120 + day;break;
        case 6:sum = 151 + day;break;
        case 7:sum = 181 + day;break;
        case 8:sum = 212 + day;break;
        case 9:sum = 243 + day;break;
        case 10:sum = 273 + day;break;
        case 11:sum = 304 + day;break;
        case 12:sum = 334 + day;break;
        default :
            printf("输入的月份有错误\n");
            break;
    }
    if(month>2)
    {
        if(isLeap(year)){
            sum = sum+1;
        }
    }
    return sum;
}

// 函数说明：代入两个年份,计算它们相差的天数
// 形式参数：year1,year2,整型,两个参与比较的年份
// 返回值：整型,两个年份相差的天数
int daysBetweenYears(int year1,int year2)
{
    int sum_year_day = 0;
    int i = 0;
    sum_year_day = (year2-year1)*365;
    for(i=year1;i<year2;i++)
    {
        if(isLeap(i)){
            sum_year_day = sum_year_day+1;
        }
    }
    return sum_year_day;
}

// 函数说明：代入两个日期(年、月、日),计算它们之间相差的天数
// 形式参数：
// year1,month1,day1,整型,第一个日期(年、月、日)
// year2,month2,day2,整型,第二个日期(年、月、日)
```

```
// 返回值：整型,两个日期相差的天数
int daysBetweenDays(int year1,int month1,int day1, int year2,int month2,
int day2)
{
    return daysBetweenYears(year1,year2) -
    daysPastofYear(year1,month1,day1) + daysPastofYear(year2,month2,day2);
}

int main(void)
{
    int year1 = 1921,month1 = 7,day1 = 1;
    int year2 = 2021,month2 = 7,day2 = 1;
    int sum = 0;

    sum = daysBetweenDays(year1,month1,day1,year2,month2,day2);
    printf("它们之间相差的天数为：%d\n",sum);
    return 0;
}
```

解析：

本题看似复杂,但是思路并不难。我们判断两个日期相隔的时间,其实就是要把中间过了多少年算清楚。某一年不是 365 天(平年),就是 366 天(闰年),所以先定义一个判断闰年的函数。

从第一个日期所处的年份开始,一直数到第二个日期所处年份的前一年,这期间度过的是整年份的时长。这个结果减去第一个日期所处的年份已经过去的时间,然后再加上第二个日期所处的年份已经过去的时间,就是要求的结果。

要判断某一天在本年度是第几天,把本年度前面的日子加起来就行了。因此这里给出了多个函数的设计实现。每个函数对应一个功能。这个实现的思路用流程图表示如图 5.6 所示,其中函数用阴影方框显示。

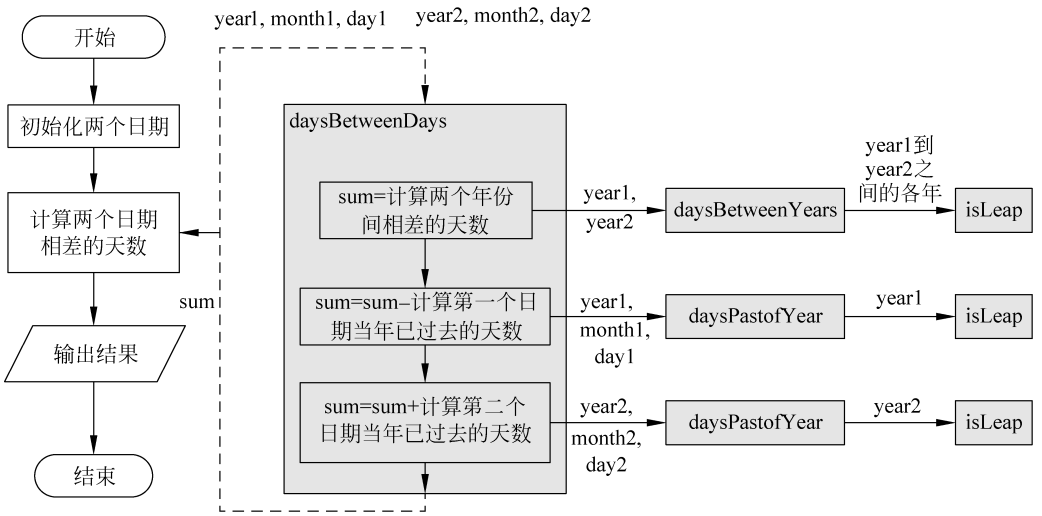

图 5.6　习题 5.11 参考程序的流程图

以上代码给出的例子运行结果见图5.7。从1921年7月1日到2021年7月1日,总共经过了36525天。

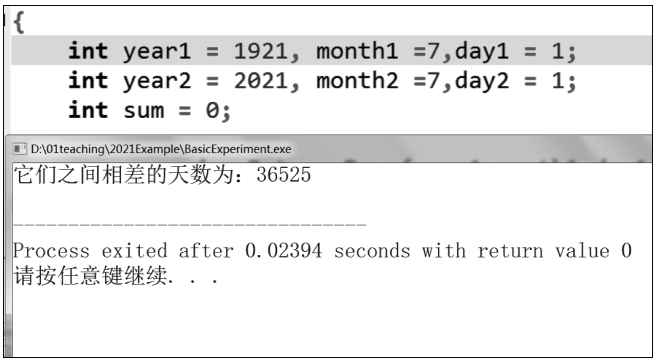

图5.7 习题5.11参考程序的执行结果

5.2 补充习题

1. 下面关于指针基类型的叙述正确的是(　　)。

(A) 基类型不同的指针,其地址值不能相同

(B) 指针的基类型决定通过该指针访问的每个内存单元包含多少个字节

(C) 基类型相同的指针,可以进行加、减、乘、除运算

(D) 基类型为void的指针,可以存取任何类型的数据

2. 若函数调用时的实参为变量,则以下关于函数形参和实参的叙述中正确的是(　　)。

(A) 函数的形参和实参分别占用不同的存储单元

(B) 形参只是形式上的存在,不占用具体存储单元

(C) 同名的实参和形参占同一存储单元

(D) 函数的实参和其对应的形参共占同一存储单元

3. 以下正确的函数声明是(　　)。

(A) double fun(int x,int y)

(B) double fun(int x; int y)

(C) double fun(int x, int y);

(D) double fun(int x,y);

4. 以下关于return语句的叙述中正确的是(　　)。

(A) 一个自定义函数中必须有一条return语句

(B) 一个自定义函数中可以根据不同情况设置多条return语句

(C) 定义成void类型的函数中可以有带返回值的return语句

(D) 没有return语句的自定义函数在执行结束时不能返回到调用处

5. 以下选项中叙述错误的是(　　)。

(A) C 程序函数中定义的自动变量,系统不自动赋确定的初值

(B) 在 C 程序的同一函数中,各复合语句内可以定义变量,其作用域仅限本复合语句内

(C) C 程序函数中定义的赋有初值的静态变量,每调用一次函数,赋一次初值

(D) C 程序函数的形参不可以说明为 static 型变量

6. 下列程序的输出结果为(　　)。

```
#include <stdio.h>
int max(int x,int y) { return x>y?x:y; }
int main(void)
{
    int a = 3;
    int b = 4;
    int c;
    c = max(a,b);
    printf("%d,%d,%d",a,b,c);
    return 0;
}
```

(A) 3,4,3　　(B) 3,4,4　　(C) 3,4,0　　(D) 3,4,1

7. 下列程序的输出结果为(　　)。

```
#include <stdio.h>
int change(int x,int y) { x = 1; y = 2; return x+y; }
int main(void)
{
    int x = 3;
    int y = 4;
    int z;
    z = change(x,y);
    printf("%d,%d,%d",x,y,z);
    return 0;
}
```

(A) 1,2,3　　(B) 3,4,3　　(C) 3,4,7　　(D) 1,2,7

8. 有以下代码,则代码填充处可以正确执行的语句为(　　)。

```
#include <stdio.h>
int main(void)
{
/************代码填充处********/

/************end**************/
}
int Max(int x,int y)
{
```

```
    int max;
    if(x>y) max = x;
    else max = y;
    return max;
}
```

(A) int z = Max(0,1);

(B) int Max(int x,int y); int z = max(0,1);

(C) int max(int x,int y); int z = max(x,y);

(D) int Max(int x,int y); int z = Max(0,2)+7;

9. 有以下程序:

```
#include <stdio.h>
int func(int a,int b)
{
    int c;
    c = a+b;
    return c;
}
int main(void)
{
    int x = 6,y = 7,z = 8,r;
    r = func((x--,y++,x+y),y);
    printf("%d\n",r);
    return 0;
}
```

则程序的输出结果是(　　)。

(A) 11　　(B) 10　　(C) 21　　(D) 31

10. 有以下程序:

```
#include <stdio.h>
int fun(int a, int b)
{
    if(b == 0) return a;
    else return(fun(--a, --b));
}
main(void)
{
    printf("%d\n", fun(4, 2));
}
```

程序运行后的输出结果是(　　)。

(A) 1　　(B) 2　　(C) 3　　(D) 4

11. 已定义以下函数:

```
int fun( int *p)
{  return  * p;}
```

fun 函数的返回值是(　　)。

(A) 一个整数　　(B) 不确定的值

(C) 形参 p 中存放的值　　(D) 形参 p 的地址值

12. 程序源代码如下所示,它的输出结果为(　　)。

```
#include <stdio.h>
void hello_world(void)
{
    printf("Hello, world!\n");
}
void three_hellos(void)
{
    int counter;
    for (counter = 1; counter <= 3; counter++)
    hello_world( );                    // 调用 hello_world 函数
}
void main(void)
{
    three_hellos( );                   // 调用 three_hellos 函数
}
```

13. 如下源代码的执行结果是(　　)。

```
#include <stdio.h>
int main(void)
{
    int i,num;
    num = 2;
    for(i=0;i<3;i++) {
        printf(" The num equal %d\n",num);
        num++;
        {
            static int num = 1;
            printf(" The internal block num equal %d\n",num);
            num++;
        }
    }
    return 0;
}
```

14. 编写一个函数,输入 n 为偶数时,计算 1/2+1/4+…+1/n; 当输入 n 为奇数时,计算 1/1+1/3+…+1/n。

15. 斐波那契数列问题: 有一对兔子,从出生后第三个月起每个月都生一对兔子,小兔子长到第三个月后每个月又生一对兔子,假如兔子都不死,问每个月的兔子总数为多少? 利用递归方法编程计算结果。