C语言

Chapter 5 函数

函数的定义和调用

函数的定义

函数定义的一般形式

```
函数类型 函数名（形式参数表）{//函数首部
    函数实现过程              //函数体
}
```

函数首部

函数类型：函数返回结果的类型（如 `return 0;` 返回的 0 就是 `int` 类型的，则此函数的类型就是 `int` ）
形式参数表：`类型1 形参1,类型2 形参2`
函数首部例如 `double cylinder(double r,double h)`

函数体

函数体中除去形参，都是需要定义的普通变量

函数的调用

函数调用的过程

从主函数 main () 开始，运行到含有函数的语句时转而执行函数，执行完后返回到原位置，继续执行主函数

函数调用的形式

一般形式：
`函数名（实际参数表）`
实际参数，简称实参，可以是**常量、变量和表达式**
调用函数例
`volume=cylinder(radius,height);`

参数传递

上述例子中的形参：
函数定义中指明形参 `double r,double h`
主函数中的是实参 `radius,height`

函数调用中，实参 radius 和 height 的值被依次传递给形参 r, h
实参可以是变量、常量、表达式；形参只能是变量（形参的定义过程确保形参只能是变量啊）
参数传递，将实参的值复制给形参，**参数的传递过程是单向的，只允许把实参的值复指给形参，形参的值即使在函数中改变了，也不会反过来影响实参**
形参的值不会影响实参。

函数结果返回

1. 存在返回值的函数 `return 表达式;` 表达式的类型与函数类型一致；表达式只能返回一个值
2. 不存在返回值的函数，没有 return 语句，函数的类型为 `void`

变量

局部变量和全局变量

局部变量

定义：定义在函数内部的变量。
作用范围：仅集中在此函数内部。形参是局部变量
局部变量还可以作用在复合语句内部，用作小范围内的临时变量。
优点：避免各个函数之间的变量相互干扰，可应用在结构化程序设计中。

全局变量

定义：定义在函数外而不属于任何函数的变量称为全局变量。
作用范围：从定义开始到程序所在文件的结束，它对作用范围内的所有函数都起作用。
优点：解决多个函数间的变量共用。

全局变量与局部变量同名：在该局部变量所在的函数中，局部变量起作用。

变量生存周期和静态局部变量

变量生存周期

定义：变量从定义开始分配存储单元，到运行结束存储单元被回收，这个过程称为变量生存周期。
自动变量：函数的局部变量称为自动变量。函数被调用时，系统自动为其局部变量分配存储单元；一旦该函数调用结束，所有分配给局部变量（包括形参和函数内定义的普通变量）的单元由系统自动回收。
全局变量：

作用范围：从定义开始到该程序所在文件结束
生存周期：整个程序执行周期；

变量存储的内存分布

- 系统存储区
- 用户存储区
  - 程序区（C 程序代码）：如主函数 main ( )、自定义函数 complex_add ( ) 等
  - 数据区
    - 静态存储区
      - 全局变量
      - 静态局部变量
    - 动态存储区（如自动变量）
      - main ( ) 变量区：
        - Real 1, ima 1, i, j 等
      - Complex_add ( ) 存储区
        - Real 1, imag 1 等等

静态变量

静态局部变量：
生存周期：从定义持续到程序结束
不像普通局部变量一样，在相应函数被调用结束后存储空间就被收回，而是一直保留存储单元，一旦含有静态局部变量的该函数再次被调用，此变量就会重新激活，**上一次函数调用的值仍然保留**
定义格式：
`static 类型名 变量表`

结构化程序设计思想

1. 自顶向下分析问题，拆解成树状图
2. 模块化设计
3. 结构化编码主要原则

Chapter 7 Arrays

一维数组的定义和引用

`Int a[10]={1,2,3,4,5,6,7,8,9,10};

定义

引用

一维数组的初始化

对 普通数组 初始化

`int a[10]={1,2,3,4,5,6,7,8,9,10};`
普通数组只定义长度，系统不会自动初始化为全零，而是随机值；
`int a[10];`
但对数组赋部分值后，其他的值会自动赋 0
`int a[5]={1,2};`
等价于
`int a[5]={1,2,0,0,0};`

对 静态数组 初始化

如果没有手动初始化赋值，系统会自动给所有元素赋 0
`static int b[5];`
等价于
`static int b[5]={0,0,0,0,0};`

使用一维数组编程

斐波那契数列

"C:\Users\14837\Desktop\C\src\04_data_structures\Fibonacci_Sequence. C"
Git version 1

```
/*计算斐波那契数列前46个数，并按照每行5个的格式输出*/

#include <stdio.h>



int main(){

    int fib[46]={1,1};

    int i,j;

    i=2;

    for(i;i>=2&&i<=45;i++){
```

```
        fib[i]=fib[i-1]+fib[i-2];

    }

    for(j=0;j<=45;j++){

        printf("%d\n",fib[j]);



    }

    system("pause");  // 会显示 "Press any key to continue..."

    return 0;



}
```

新增修改：每隔5个换行
Git version 2

顺序查找法

C:\Users\14837\Desktop\C\src\04_data_structures\ordering_search. C

查找法（Search）

二分查找法（Binary Search）

效率较高，但要求数组元素**有序**
查找的退出的条件：
`low>high`

练习

一个需要多注意的问题：用于判断的标志位，每次循环都要手动置位。

二维数组的定义和引用

定义

`类型名 数组名[行长度][列长度]`

引用

引用行下标和列下标
二维数组在内存中的存储方式：
从第 0 行开始，先行后列。
注意：二维数组的表示中，行和列均从 1 开始，为了保持和一维数组的一致性，二维数组的行列也从 0 开始。

二维数组的初始化

分行赋初值

`int a[3][3]={{1,2,3},{4,5,6},{7,8,9}}`
只对部分元素赋值：
`int a[3][3]={{1,2,3},{},{7,8,9}}`
表示矩阵

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

顺序赋值法

`int a[3][3]={1,2,3,4,5,6,7,8,9}`

使用二维数组编程

字符串

一维字符数组

字符数组的定义、初始化、引用
`char str[80];`
前几个元素赋值，后面的元素自动赋 0
`static char str[6]={'H','a','p','p','y'};`
等价于
`static char str[6]={'H','a','p','p','y','\0'};`

字符串

定义：字符序列，包含字符串的有效字符，和最后的结束标志'\0'；
字符串的有效长度：有效字符的个数
C 语言将字符串作为一个特殊的一维字符数组来处理（也就是任意一个字符串，都要存储到数组里）
字符串的存储——数组初始化
`static char s[6]={'H','a','p','p','y','\0'};`
等价于
使用字符串常量初始化
`static char s[6]={"Happy"};`
或
`static char s[6]="Happy";`
数组长度长于字符串：
只对数组前几个元素赋值，后面的值不确定
字符串的操作
遍历数组：比较数组元素值是否等于‘\0’来控制循环
字符串的存储——赋值和输入

```
static char s[80];
s[0]='a';
s[1]='\0';
```

将字符串赋给数组
输入情况：
输入字符串，需要设定一个输入结束符，表示字符串输入结束，并将输入结束符转换为字符串结束符'\0'。

使用字符串编程

字符串与数字数组不同的区别：字符串数量的不确定，以及字符串末尾的标识符'\0'

Chapter 8 指针

指针及指针变量

二级标题

二级标题