电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2016220201024

姓 名 王籽涵

（实验） 课程名称 C语言程序设计

理论教师 张翔

实验教师 张翔

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：王籽涵 学号：2016220201024 指导教师：张翔**

**实验地点：清水河图书馆 实验时间：16.12.12**

1. **实验名称：C语言的字符与数据处理**
2. **实验学时：3**
3. **实验目的：**
4. 学习并掌握一维／二维数组存储信息的方式并用以解决实际问题。
5. 学习并掌握对字符的处理方法。
6. 学习并掌握函数的声明与定义，明确区分形参与实参，以及如何编写和运用函数。
7. **实验原理：**

2.1 字符处理

字符类型的值可以根据计算机的不同而不同，因为不同的机器可能会有不同 的字符集。当今最常用的字符集是ASCII (美国信息交换标准码), 它用7位代码表示128个字符。ASCII常被扩展为8位代码用于表示256个字符代码，被称为Latin-1,提供一些西欧和许多非洲语言所需的字符。

字符类型的变量能被赋值为任何单个字符：

char ch;

ch = 'a'; /\* lower-case a \*/

ch = 'A'; /\* upper-case A \*/

ch = '0'; /\* zero \*/

ch = ' '; /\* space \*/

字符操作:

C语言会按小整数的方式处理字符。在 ASCII码中，字符的取值范围是0000000~1111111,这个范围可以看成是0~127的整数。字符常量实际上是整数类型，而不是字符类型，当计算中出现字符时，C语言使用它对应的整数值。所以字符可以像数那样进行比较。诸如‘a’<=ch这样的比较使用的是字符所对应的整数值。（在ASC码中字符 ‘a’ 的值为97, ‘A‘的值为65, ’0’ 的值为 48, and ‘ ’的值为32等）

字符常量通常是用单引号括起来的字符。然而，一些特殊符号是无法采用上述这种书写方式的，比如换行符，因为它们是不可见的（无法打印的），或者无法从键盘输入的。转义序列提供了一种呈现这类特殊符号的方法。

转义序列共有两种：字符转义序列和数字转义序列。

转义序列

字符转义序列：

名称 转义序列

Alert (bell) \a

Backspace \b

Form feed \f

New line \n

Carriage return \r

Horizontal tab \t

Vertical tab \v

Backslash \\

Question mark \?

Single quote \'

Double quote \"

字符转义序列使用起来很方便，但是没有包含所有无法打印的ASCII字符。

也无法用于表示基本的128个ASCII码字符以外的字符。

数字转义序列可以表示任何字符，所以它可以解决上述问题。

对特殊字符，数字转义序列使用这些字符的八进制或十六进制值。

例如，ASCII码转义字符(十进制值为27) 对应的八进制值为33，对应的十六进制值为1B

八进制转义序列由字符\和跟随其后的一个最多含有三位数字的八进制数组成，如 \33 或 \033.

十六进制转义序列由\x和跟随其后的一个十六进制数组成，如\x1b 或\x1B.

其中x必须小写，不过十六进制的数字不限大小写。

作为字符常量使用时，转义序列必须用一对单引号括起来。

字符处理函数

调用C语言的toupper库函数是快捷、易于移植的把小写字母转换成大写字母的方法。程序调用toupper函数需要在顶部放置下面这条#include指令：#include <ctype.h>。C函数库提供了其他有用的字符处理函数。

读/写字符

转换说明符%c允许scanf和printf函数对单独一个字符进行读/写操作：

char ch;

scanf("%c", &ch); /\* reads one character \*/

printf("%c", ch); /\* writes one character \*/

scanf函数不会跳过空白字符。为了scanf函数在读入字符前跳过空白字符，需要在格式串转换说明%c前面加上一个空格。

对单个字符的输入和输出，还可以使用gechar函数和putchar函数来代替调用 scanf函数和printf函数。putchar函数打印单独一个字符：putchar(ch)；getchar 函数将读取并返回一个字符：ch = getchar()；注意getchar函数返回一个整数值而不是字符值。和scanf函数一样，gechar函数也不会再读取时跳过空白字符。 使用gechar和puchar函数代替scanf和printf函数可以节约执行时间，因为 scanf和printf函数是设计来读写多种不同格式的类型数据的，有判断分类的过程，耗费了一定时间。

getchar函数具有另一个优点，由于返回的是读入的字符，所以getchar函数可以应用在多种不同的C语言惯用法中。while((ch=getchar())!='\n')是惯用来跳过输入行的剩余部分；变量ch都可以不需要：while (getchar() != '\n')。getchar 函数在用于循环中搜寻字符时和跳过字符一样有效。 利用 getchar 函数跳过不定数量的空格字符：while((ch=getchar())==' ')，当循环终止时，变量ch将包含 getchar函数遇到的第一个非空字符。

2.2 数组

数组(array)是含有多个数据值的数据结构，并且每个数据值具有相同的数据类型。这些数据值被称为元素(element)，可以根据元素所处的位置对其进行单独选择。

简单的数组类型就是一维数组。一维数组中的元素一个接一个地编排在单独一行(或者一列)内。为了声明一个数组，需要说明数组元素的类型和数量：int a[10]；数组的元素可以是任何类型；数组的长度可以用任何(整数)常量表达式说明。注意较好的方法是用宏来定义数组的长度。

#define N 10

…

int a[N];

为了存取特定的数组元素，可以在写数组名的同时在后边加上一个用方括号围绕的整数值。这被称为对数组进行下标(subscripting)或索引(indexing)。长度为n的数组，其元素的索引是从0到 n-1。

运算符sizeof可以确定数组的大小(字节数)。如果数组a有10个整数，那么sizeof(a)可以代表40(假设每个整数用4字节存储)。还可以用sizeof来计算数组元素的大小，比如a[0]。

用数组的大小除以数组元素的大小也可以得到数组的长度： sizeof(a) / sizeof(a[0]) 。当需要数组长度时，一些程序员采用上述这个表达式。

for (i = 0; i < sizeof(a) / sizeof(a[0]); i++)

a[i] = 0;

此循环用以清0数组a。

有些编译器会对i< sizeof(a)/sizeof(a[0])表达式给出一条警告信息。变量 i 的类型可能是 int (有符号类型)，而sizeof返回值的类型是 size\_t (一种无符号类型)。

将有符号整数和无符号整数比较是很危险的。为避免这一警告，可以把sizeof(a) / sizeof(a[0]) 强制转化成有符号整数：(int) (sizeof(a) / sizeof(a[0]))； 定义一个宏来表示上述表达式常常是很有帮助的： #define SIZE ((int) (sizeof(a) / sizeof(a[0])))。

数组可以有任意维数（多维数组）。例如二维数组：为了访问i行j列的元素，需要写成m[i][j]形式；C语言在计算机中二维数组的存储是按行主序存储数组的；嵌套的for循环是处理多维数组的理想选择，一个循环遍历每一行，一个循环遍历每一列；多维数组初始化与一维数组类似。

2.3 函数

在编写程序是自定义函数可以大大简化编写复杂度。

函数简单来说就是一连串组合在一起并且命名的语句。每个函数本质上是一个自带声明和语句的小程序。一个函数初可能是某个程序的一部分，但可以将其用于其他程序中。可以利用函数把程序划分成小块，这样便于人们理解和修改程序。可以避免重复编写可多次使用的代码。一个函数最初可能是某个程序的一部分，但可以将其用于其他程序中。

函数的定义与调用

函数名后边有一串形式参数列表。每个形式参数需要说明其类型；形式参数间用逗号进行分隔。如果函数没有形式参数，那么在圆括号内应该出现 void。函数体可以包含声明和语句。函数体内声明的变量专属于此函数，其他函数不能对这些变量进行检查或修改。 在C89中，变量声明必须出现在语句之前。 在C99中，变量声明和语句可以混在一起，只要变量在第一次使用前进行声明即可。

由函数名和跟随其后的实际参数列表组成，其中实际参数列表用圆括号括起来， 如果丢失圆括号，那么将无法进行函数调用。void型的函数调用是语句，所以调用后边始终跟着分号；非void型的函数调用产生的值可存储在变量中，还可以进行测试、显示或 者其他用途，如果不需要，非void型函数的返回值总是可以丢弃。(void) printf("Hi, Mom!\n"); 这里使用(void)可以使别人清楚编写者是故意扔掉返回值的，而不是忘记了。

void型函数可返回与传入空值，非void型的函数调用产生的值可存储在变量中，还可以进行测试、显示或者其他用途。

实际参数：

在C语言中，实际参数是通过值传递的：调用函数时，计算出每个实际参数的值并且把它赋值给相应的形式参数。在函数执行过程中，对形式参数的改变不会影响实际参数的值，这是因为形式参数中包含的是实际参数值的副本。实际参数按值传递既有利也有弊。

实际参数的转换：

C语言允许在实际参数的类型与形式参数的类型不匹配的情况下进行函数调用。管理如何转换实际参数的规则与编译器是否在调用前遇到函数(或者函数的完整定义)的原型有关：

编译器在调用前遇到函数（或者函数的完整定义）的原型。

就像使用赋值一样，每个实际参数的值被隐式地转换成相应形式参数的类型。

例如：如果把int类型的实际参数传递给期望得到double型数据的函数，那么会自动把实际参数转换成double类型。

编译器在调用前没有遇到原型。

编译器执行默认的实际参数提升：

把float型的实际参数转换成double类型。

执行整数的提升，即把char型和short型的实际参数转换成int型(在C99中实现了整数提升)。

当形式参数是一维数组时，可以不说明数组的长度。C语言没有为函数提供任何简便的方法来确定传递给它的数组的长度。但是，如果函数需要，必须把长度作为额外的实际参数提供出来。 函数无法检测通过传递是否获得了正确的数组长度。

函数可以改变数组型形式参数的元素，且改变会在相应的实际参数中体现出来。

1. **实验内容：**

项目1：第7章编程题4

项目2：第7章编程题12

项目3：第8章编程题2

项目4：第8章编程题9

项目5：第8章编程题15

项目6：第9章编程题3

1. **实验器材（设备、元器件）：**

Macbook Pro 2016

1. **实验步骤：**

1.解读实验题目明细；

2.思考如何解决问题；

3.具体代码的书写；

4.运行代码并进行debug；

5.完成后解决更多边界条件。

1. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**

项目1：第7章编程题4

1）实验结果

程序运行成功并达到实验目的，见图1-1。

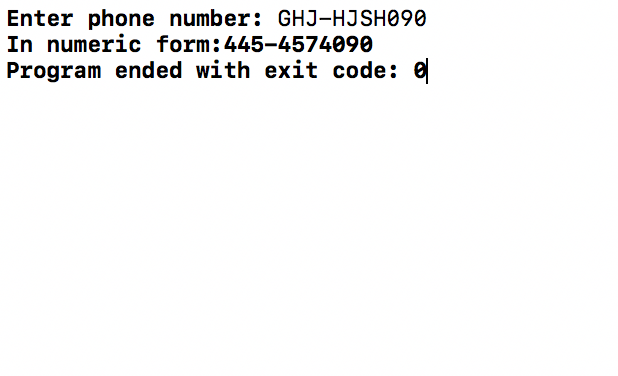


图1-1

2）思路概括与核心分析

…………

…………

case 'P':

case 'Q':

case 'R':

case 'S':

printf("7");

break;

case 'T':

case 'U':

case 'V':

printf("8");

break;

case 'W':

case 'X':

case 'Y':

case 'Z':

printf("9");

break;

}

}

char ch;

printf("Enter phone number: ");

while ((ch=getchar())!='\n'){

switch(ch){

case 'A':

case 'B':

case 'C':

printf("2");

break;

case 'D':

case 'E':

case 'F':

printf("3");

break;

case 'G':

case 'H':

case 'I':

break;

…………

…………

让字母转化成数字，最本质的应用就是用switch函数进行分类判断并进行处理，在这里可讲switch单独写为一个函数，使主函数看起来清晰明了。

项目2：第7章编程题12

1）实验结果

程序运行成功并达到实验目的，见图1-2。

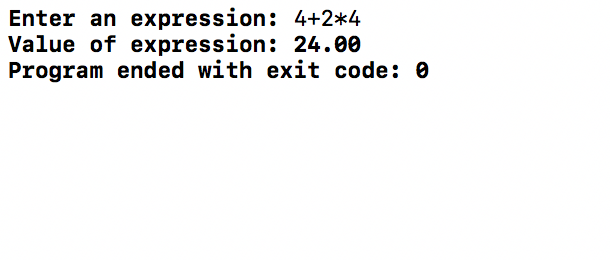


图1-2

2）思路概括与核心代码分析

switch(sign\_1){

case '+':{

mid=a+b;

}

break;

case '-':{

mid=a-b;

}

break;

case '\*':{

mid=a\*b;

}

break;

case '/':{

if(b==0){

printf("Illeagle expression!!!\n");

return 0;

}

else{

mid=a/b;

};

};

break;

}

此题同样是对于switch的应用，与上一题一样，判断字符并处理，可单独写成函数。

项目三：第8章编程题2

1）实验结果

程序运行成功并达到实验目的，见图1-3。

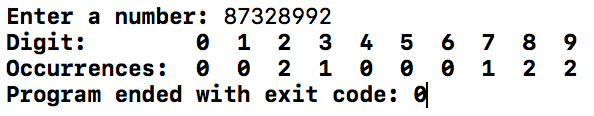


图1-3

2）思路概括与核心代码分析

int digit\_seen[10]={0};

int digit;

long n;

printf("Enter a number: ");

scanf("%ld",&n);

for (;n>0;n/=10){

digit = n%10;

digit\_seen[digit]++;

};

printf("Digit: ");

for(digit=0;digit<=9;digit++){

printf("%3d",digit);

};

printf("\nOccurrences:");

for(digit=0;digit<=9;digit++){

printf("%3d",digit\_seen[digit]);

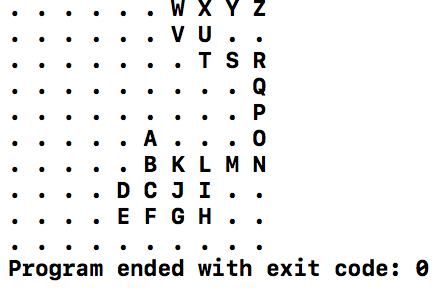
};

将用户所输入的数依次%10，取最后一位，并让其在数组中的位置+1，结束后又将原数／10来去除最后一位。循环操作直至没有数。

项目4：第8章编程题9

1）实验结果

程序运行成功并达到实验目的，见图1-4。



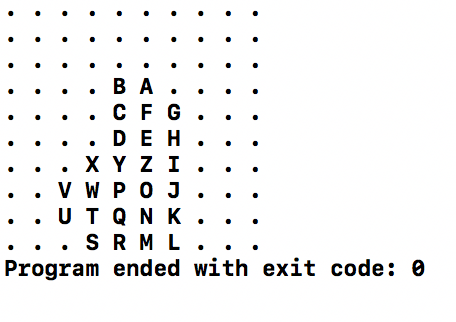


图1-4

2）思路概括与核心代码分析

int r,m,n;

char ch='A',A[10][10];

for(m=0;m<10;m++)

for(n=0;n<10;n++)

A[m][n]='.';

srand((unsigned)time(NULL));

m=rand()%9;

n=rand()%9;

for(A[m][n]=ch;ch<'Z';){

r=rand()%4;

if((m==0&&n==0&&A[1][0]!='.'&&A[0][1]!='.')||

(m==0&&n==9&&A[0][8]!='.'&&A[1][9]!='.')||

(m==9&&n==0&&A[8][0]!='.'&&A[9][1]!='.')||

(m==9&&n==9&&A[8][9]!='.'&&A[9][8]!='.'))

break;

if((m==0&&A[1][n]!='.'&&A[0][n-1]!='.'&&A[0][n+1]!='.')||

(m==9&&A[8][n]!='.'&&A[9][n-1]!='.'&&A[9][n+1]!='.')||

(n==0&&A[m][1]!='.'&&A[m-1][0]!='.'&&A[m+1][0]!='.')||

(n==9&&A[m][8]!='.'&&A[m-1][9]!='.'&&A[m+1][9]!='.'))

break;

if(A[m+1][n]!='.'&&A[m-1][n]!='.'&&A[m][n+1]!='.'&&A[m][n-1]!='.')

break;

if((r==0&&(m==0||A[m-1][n]!='.'))||

(r==1&&(m==9||A[m+1][n]!='.'))||

(r==2&&(n==0||A[m][n-1]!='.'))||

(r==3&&(n==9||A[m][n+1]!='.')))

continue;

if(r==0){

m--;

A[m][n]=++ch;

}

if(r==1){

m++;

A[m][n]=++ch;

}

if(r==2){

n--;

A[m][n]=++ch;

}

if(r==3){

n++;

A[m][n]=++ch;

}

}

for(m=0;m<10;m++){

for(n=0;n<10;n++)

printf("%c ",A[m][n]);

printf("\n");}

本题的难点在于具体算法实现时对边界条件的书写。

在上述代码中很好的实现了对各种边界条件的把握与判断，分别是走完了所有字母、走出了范围、被以前所走的路径堵住。在此代码中用了三个if分别描述了三种情况，清晰明了。

项目5：第8章编程题15

1）实验结果

程序运行成功并达到实验目的，见图1-5。

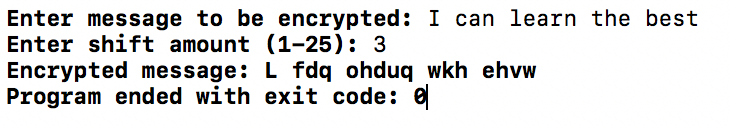


图1-5

2）思路概括与核心代码分析

char ch[80];

int i,n,m;

printf("Enter message to be encrypted: ");

for(i=0;(ch[i]=getchar()) != '\n';i++);

m=i;

printf("Enter shift amount (1-25): ");

scanf("%d",&n);

for(i=0;i<=m;i++){

if('A'<=ch[i]&&ch[i]<='Z')

ch[i]=((ch[i]-'A')+n)%26+'A';

if('a'<=ch[i]&&ch[i]<='z')

ch[i]=((ch[i]-'a')+n)%26+'a';

}

printf("Encrypted message: ");

for (i=0;i<=m;i++){

printf("%c",ch[i]);

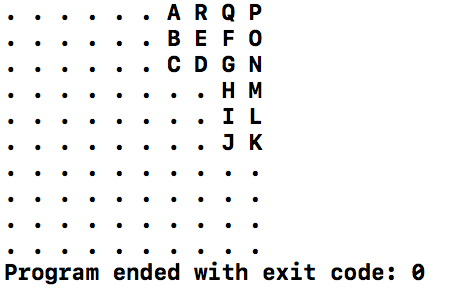
}

在此代码中先是将字符串读入数组中，然后循环判断其是否为大小写并作出相应的加密操作，%26表示其不会超出字母的范围。

项目6：第9章编程题3

1）实验结果

程序运行成功并达到实验目的，见图1-6。



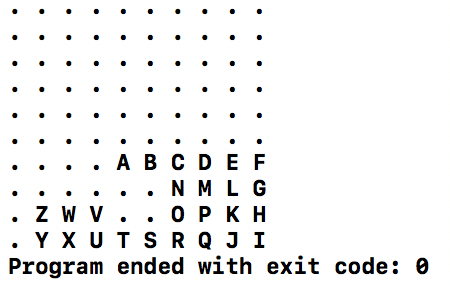


图1-6

2）思路概括与核心代码分析

void generate\_random\_walk(char A[10][10]);

void print\_array(char A[10][10]);

int main(void)

{

char A[10][10];

generate\_random\_walk(A);

print\_array(A);

return 0;

}

此代码在上一题代码的基础上将主函数内部复杂的功能分成两个功能代码，在主函数中调用即可，大大的增加了程序的可读性。

1. **总结及心得体会：**

在这次实验中，虽然题目难度较大，但是，通过了相应的学习和考验之后，对C中的字符处理和函数调用有了更好的理解。也更加深切的体会到了老师上课所说的分而治之的重要性，但还需要继续深入的学习才行。

1. **对本实验过程及方法、手段的改进建议：**

在实现了基础功能之后，一定要好好想想边界条件。

在处理大型的、较棘手的问题时，记得分而治之。

**报告评分：**

**指导教师签字：**