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：基础实验大楼 实验时间：2016.12.21**

1. **实验名称：C语言程序结构和指针用法**
2. **实验学时：3**
3. **实验目的：**
4. 掌握 C 语言的程序结构。
5. 掌握 C 语言指针的用法。
6. 掌握 C 语言指针和数组的关系。
7. 掌握 C 语言字符串的用法。
8. **实验原理：**

局部变量

在函数体内声明的变量称为该函数的局部变量。

默认情况下，其具有**自动存储期限**和**块作用域**两种性质。

自动存储期限是指局部变量的存储单元是在包含该变量的函数被调用时“自动”分配的，函数返回时收回分配，即包含局部变量的函数返回时，局部变量的指无法保留。

块作用域是指可以引用该变量的程序文本的部分。局部变量拥有块作用域，从变量声明的点开始一直到所在函数体的末尾。

静态局部变量

在局部变量声明中放置单词static可以使变量具有静态存储期限而不再是自动存储期限。因为静态存储期限拥有永久的存储单元，在执行整个程序期间都会保留变量的值。

概括来说，静态变量是对其他函数隐藏数据的地方，但是他会为将来同一个函数的在调用保留这些数据。

形式参数

形式参数具有和局部变量一样的性质，即自动存储期限和块作用域。通常情况下，形式参数作为便于理解的变量存在于函数中时阅读者能够顺利阅读程序。

外部变量

传递参数是给函数传送信息的一种方法。函数还可以通过外部变量进行通信，且外部变量是声明在任何函数体外的。

外部变量有时也被称为全局变量，其性质不同于局部变量的性质。外部变量具有**静态存储期限**和**文件作用域**两种性质。

静态存储期限是指存储在外部的变量会永久保存下来，就如声明为static的变量一样。

文件作用域是指从变量被声明开始一直到文件的末尾，所有更随在外部变量之后的函数都可以访问并修改它。

栈

栈是一种数据结构，其像数组一样，可以存储具有相同数据类型的多个数据项。但其操作是受限制的：只可以往栈中压入数据项（即把数据项加入栈顶）或者从栈中弹走数据项（即从同一端移走数据项），并禁止测试或修改不在栈顶的数据项。

C语言中的实现方法是将元素存储在数组中，将其命名为contents。命名为top的一个整型变量来标记栈顶的位置。栈为空时，top的值为0。为了往栈中压入数据项，可以吧数据项简单存储在contents中由top指定的位置上，然后自增top。弹出数据项则要求自减top，然后用它作为contents的索引取回弹出的数据项。

程序块

C语言还允许包含声明的复合语句： **[程序块] {多条声明 多条语句}**

这里用**程序块**（也称块）在描述这类复合语句。

在默认情况下，声明在程序块中的变量的存储期限是自动的：进入时为变量分配存储单元，退出时收回，且变量具有块作用域，即不能再程序块外引用。

函数体是程序块。

作用域

当程序块内的声明命名一个标识符是，如果此标识符已经是可见的，新的声明临时“隐藏”了旧的声明，标识符获得了新的含义。在程序块的末尾，标识符重新获得旧的含义。

构建C程序

迄今为止，已经知道程序可以包含：

诸如#include和#define这样的预处理指令；

类型定义；

外部变量声明；

函数原型；

函数定义。

C语言对上述这些项的顺序要求极少：执行到预处理指令所在的代码时，预处理指令才会起作用，变量只有在声明后才能使用。在调用函数前对其进行声明或定义是一种很好的习惯。

存储方式与方法

大多数计算机都将内存分割为字节，而每个字节可以存储8位的信息。每个字节都有唯一的地址，用来和内存中的其他字节相区别。如果内存中有n个字节，那么可以把地址看作0~n-1的数。程序中的每个变量占有一个或多个字节内存，把第一个字节的 地址称为是变量的地址。虽然一般用数表示地址，但是地址的取值范围可能不同于整数的范围，所以不能用普通整型变量存储地址。我们可以用指针变量存储地址。再用指针变量存储变量的地址时，指针就是地址，指针变量就是存储地址的变量。

指针

指针变量的声明和普通变量基本一样，唯一的不同就是必须在指针变量的名字前放置星号\*。因为指针变量可以指向不属于变量的内存区域，所以我们用术语对象来代替变量。C语言要求每个指针变量只能指向一种特定类型的对象。

为了使用指针，C语言提供了一对特殊设计的运算符。取地址符&，和间接寻址运算符\*。声明变量的时候只是为指针留出空间，并没有把它指向对象。所以在使用指针前的初始化是非常重要的。可以在声明指针变量的同时对它进行初始化。甚至也可以将它们的声明合并，但是需首先声明变量。

一旦指针指向了对象，就可以用\*来访问存储在的对象中的内容。但是不能把\*用于未初始化的指针变量，因为如果这个变量本身具有有效地址，那么对于指针变量的操作可能会对内存单元造成不可预知的影响。

指针的操作与简单运算

C语言允许使用赋值运算符进行指针复制，但是两个指针必须有相同类型。任何数量的指针变量都可以指向同一个对象。当用指针作为实际参数时，在函数体内部就可以对变量进行修改。但是有的时候，传入变量的指针并不意味着我们想对变量进行修改，所以可以用const保护参数，表明函数不会改变指针参数所指向的对象。

指针可以作为参数，当然也可以成为返回值。但是不能返回指向函数内部局部变量的指针，因为函数结束时返回的指针所指向的变量已经不存在了。

在数组中，指针本身也可以做指针算术运算（地址算术运算）。C语言只支持3中格式的运算：

指针加上整数；指针减去整数；指针减去指针；

指针与数组

数组名可以作为指针，指针也可以作为数组名。编译器把p[i]看作\*（p+i）。二维数组可以看作是一维的，因为数组是按行而不是按列存储的。对于任意二维数组a来说，表达式a[i]是指向第i行中第一个元素的指针，所以可以写为p=a[i],而这一行中的元素我们可以通过指针的整数运算来表示。

当指针p加上整数j时，会产生指向元素p后面的j个位置的元素。同理，指针p减去整数j也是使得指针指向j个位置前的元素。两个指针p、j想减，则是指向第p-j个元素。当连个指针指向同一个数组时，可以通过关系运算符进行比较，结果依赖于数组中的元素的相对位置。

C程序员常在处理数组元素的语句中组合使用 \*（间接访问）和 ++ 操作符。下述语句修改当前位置（下标i）的数组元素值，并修改下标前进到下一个元素的位置：

a[i++] = j;

相应地，可以采用指针实现同样功能：

\*p++ = j;

由于后缀的 ++ 运算符的优先级高于\*，编译器将上述语句解释为：

\*(p++) = j;

可能的 \* 和 ++运算符的组合：

\*p++ or \*(p++) 自增前表达式的值为 \*p，然后自增p;

(\*p)++ 自增前表达式的值为 \*p，然后自增\*p;

\*++p or \*(++p) 先自增 p，自增后表达式的值为 \*p;

++\*p or ++(\*p) 先自增 \*p，自增后表达式的值为 \*p;

所以，指针的算术运算允许通过对指针变量进行重复自增来访问数组元素。我们也可以用数组名作为指向数组第一个元素的指针。通常情况下，&a[i]=a+i，两者都表示指向数组a中元素i的指针。但是，不能给数组名赋新的值，不能让数组名指向其他地方。所以，数组名也可以作为传递给函数的指针型实际参数。这样操作会将数组的第一个元素赋值给函数中的形参，而数组本身是不会被复制的。所以作为实参的数组是有可能被改变的。为了使数组型参数不会被改变，可以在声明中包含const。在声明一个变量a时，声明它是指针就相当于它是数组。这只是形参，对于变量而言，声明int a[10]，编译器会预

留10个整数的空间，但是声明int \*a，编译器只会给指针变量分配空间，所以，在后一种情况下，a不是数组，不能当做数组来操作。

字符串和其相关函数

字符串字面量是用一对双引号括起来的字符序列。他可以包含转义序列，换行符＇＼ｎ＇。也可以再第一行用字符＼结尾，Ｃ语言就逊于在下一行延续字符串字面量，除了看不到的末尾的换行符，在同一行不能有其他字符跟在＼后面。另外，当两条或更多条字符串字面量相邻时，仅用空白字符分割时，编译器可以把它们合并成一条字符串，这使得我们可以把字符串分割放在两行或者更多行。

在C语言中，字符串是作为数组处理的，所以它也无法用运算符进行复制和比较。C语言提供了一系列函数。在<string.h>的头中，有strcpy函数：char \*strcpy(char \*s1, const char \*s2)。strcpy拷贝s2指向的字符串到s1指向的字符数组strcpy ，返回s1 (指向目的字符串的指针)。strcpy不检查str2字符串是否能够容纳str1指向的数组。如果没法装入，则发生未定义的行为。调用strncpy 则是一个较慢但是更安全数组拷贝方式。strncpy 需要第三个参数来限制拷贝的字符的个数。用 strncpy 来拷贝 str2 到str1:strncpy(str1, str2, sizeof(str1))。如果str2 的长度大于或者等于str1数组的长度，strncpy将保持拷贝的结果而不能给str1增加一个字符串结束符。

strlen 函数：size\_t strlen(const char \*s)。它可以返回字符串s的长度，也就是s中第一个空字符之前的字符个数（不包括空字符）。

在拼接字符串时，有strcat函数：cahr \*strcat(char \*s1，const char \*s2)。Strcat函数把字符串s2的内容追加到字符串s1的末尾，并返回字符串s1的指针。但是如果s1指向的数组没有足够大的空间，那么结果无法估计。同样，有更安全的strncat函数，更安全，也更慢。与strncpy一样，它也有第三个参数来进行限制：strncat(str1，str2，sizeof(str1)-strlen(str2)-1)。

在进行字符串比较时，有strcmp函数：int strcmp(const char \*s1,const char \*s2)。然后根据s1是小于、等于还是大于s2，返回一个小于、等于或大于0的值。Strcmp函数利用字典顺序进行比较。如果s1和s2前i个字符一致，只要s1的后一个字符小于s2的后一个字符，那么s1小于s2。如果s1和s2的所有字符一致，但是s1更短，那么s1小于s2。

1. **实验内容：**
2. 第十章编程题1

修改10.2节的栈示例使它而不是整数。接下来，增加main函数，用来要求用户输入一串圆括号或者花括号，然后指出它们之间的嵌套是否正确：

Enter parentheses and/or braces: ((){}{()})

Parenteses/braces are nested properly

提示：读入左圆括号或左花括号的时候，把他们像字符一样压入栈中。当读入右圆括号或右花括号的时候，把栈顶的项弹出，并且检查弹出项是否是匹配的圆括号或者花括号。（如果不是，那么圆括号或者花括号嵌套不正确）。当程序读入换行符时，检查栈是否为空，如果为空，那么那么圆括号和花括号匹配。如果栈不为空，那么圆括号或花括号嵌套不正确（或者如果曾经调用过stack\_underflow函数），那么圆括号和花括号不匹配。如果调用stack\_overflow函数，程序显示信息Stack overflow,并且立刻终。

1. 第十一章编程题4：

修改10.5节的poker.c程序，把所有的外部变量移到main函数中，并修改各个函数使它们通过参数进行通信。Anlayze\_hand函数需要修改变量straight,flush,four,three和pairs,所以它需要指向这些变量的指针作为参数。

1. 第十二章编程题1a:

编写程序读一条消息，然后逆序打印出这条消息：

Enter a message: Don’t get mad, get even.

Reversal is: .neve teg ,dam teg t’not’noD

提示：一次读取消息中的一个字符（用getchar函数），并且把这些字符存储在数组中，当数组满了或者读到’\n’时停止读操作。

1. 十二章编程题1b:

修改上述程序，用指针代替整数来跟踪数组中的当前位置。

1. 第十二章编程题3：

利用数组名可以用做指针的事实简化程序题1的（b）程序

1. 第十三章编程题1

Enter word: dog

Enter word: zebra

Enter word: rabbit

Enter word: catfish

Enter word: walrus

Enter word: cat

Enter word: fish

Smallest word: cat

Largest word: zebra

提示：使用两个名为smallest\_word和largest\_word的字符串用来分别记录所有输入中的“最小”单词和“最大”单词。用户每输入一个新单词，都用strcmp函数把它与smallest\_word进行比较。如果新的单词比smallest\_word“小”，就用strcpy函数把新的保存到smallest\_word中。用类似的方式与largest\_word进行比较。用strlen函数用来判断用户是否输入了4个字母的单词。

1. **实验器材（设备、元器件）：**

Macbook Pro 16

1. **实验步骤：**
2. 明确实验项目要求
3. 编写并编译代码
4. 测试程序的强壮性
5. 在完成基本功能之后考虑其边界条件
6. **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10章编程题1

1）实验结果

程序运行成功并达到实验目的，见图1-1。

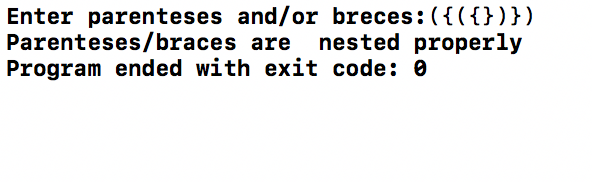


图1-1

2）思路概括与核心分

int main(){

char a,b;

printf("Enter parenteses and/or breces:");

for(;;)

{

scanf("%c",&a);

if(a != '}' && a != ')' && a != '\n')

push(a);

else if(a != '\n')

{

b=pop();

if((a == '}' && b == '{') || (a == ')' && b == '('))

continue;

else{

printf("Parenteses/braces are not nested properly\n");

break;

}

}

else if(a == '\n'){

if(is\_empty()){

printf("Parenteses/braces are nested properly\n");

break;

}

else{

printf("Parenteses/braces are not nested properly\n");

break;

}

}

}

return 0;

}

bool is\_empty(void){

return top==0;

}

bool is\_full(void){

return top==STACK\_SIZE;

}

void push(int i){

contents[top++]=i;

}

int pop(void){

return contents[--top];

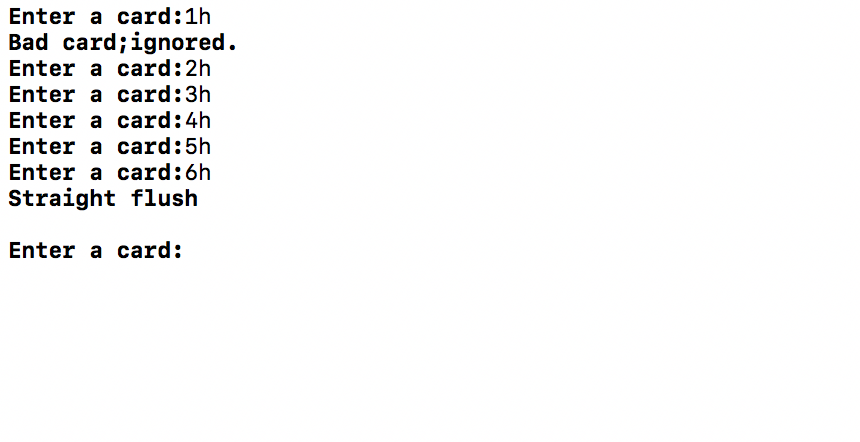
}

本题以栈为主，通过数组的相关操作来实现。首先编写出一个栈，top为0，然后编写对于栈的相关操作函数，判定为满、判定为空、压入元素、取出元素。再通过主函数的条件判断和循环来操作并实现。

项目2：第11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1-2。



2）思路概括与核心分析

void read\_cards(int r[],int s[]);

void analyze\_hand(int \*a,int \*b,int \*c,int \*d,int \*e,int r[],int s[]);

void print\_result(int \*a,int \*b,int \*c,int \*d,int \*e);

本题与书本上的原例最大的区别就是，本题将全局变量以指针形式在各个函数中发挥作用。

项目3：第12章编程题1a

1）实验结果

程序运行成功并达到实验目的，见图1-3。

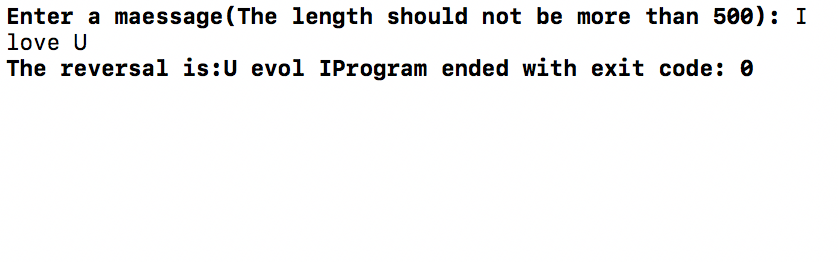


图1-4

1. 思路概括与核心分析

char ch,a[N];

int i;

printf("Enter a maessage(The length should not be more than %d): ",N);

for(i=0;(ch=getchar())!='\n' && i<N;i++)

a[i]=ch;

printf("The reversal is:");

for(;i>=0;i--)

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

return 0;

本题原理较为简单，定义一个字符数组，循环移位，逆序打印即可。

项目4：第12章编程题1b

1）实验结果

程序运行成功并达到实验目的，见图1-4。

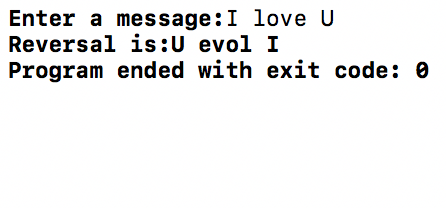


图1-4

2）思路概括与核心分析

void read\_line(char \*s)

{

char ch;

a=&s[0];

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

if ((ch=getchar())=='\n') return;

\*(a+i)=ch;

}

return;

}

本题要点在于对读行函数的书写，传入指针从而进行修改所指向的值，要熟悉这一特性才能完成。

项目5：第12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1-5。

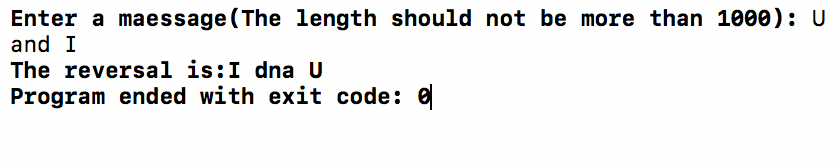


图1-5

2）思路概括与核心分析

int main(){

char ch,a[N],\*p;

p=a;

printf("Enter a maessage(The length should not be more than 1000): ");

for(;(ch=getchar())!='\n';p++)

\*p=ch;

printf("The reversal is:");

for(;p>=a;p--)

printf("%c",\*p);

printf("\n");

return 0;

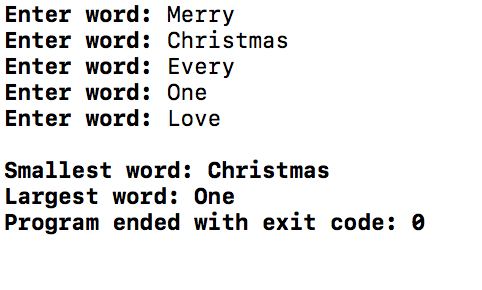
}

本题的不同之处在于直接用指针p来对数组进行相应的操作，起本质并未改变，总体与前两题相同。

项目6：第13章编程题1

1）实验结果

程序运行成功并达到实验目的，见图1-6。



2）思路概括与核心分析

char smallest\_word[WORD\_LENTH], largest\_word[WORD\_LENTH], new\_word[WORD\_LENTH];

printf("Enter word: ");

scanf("%s", new\_word);

strcpy(smallest\_word, new\_word);

strcpy(largest\_word, new\_word);

for(;;)

{

printf("Enter word: ");

scanf("%s", new\_word);

if(strcmp(smallest\_word, new\_word)>0){

strcpy(smallest\_word, new\_word);

}

if(strcmp(largest\_word, new\_word)<0){

strcpy(largest\_word, new\_word);

}

if(strlen(new\_word)==4)

break;

}

printf("\nSmallest word: %s\n",smallest\_word);

printf("Largest word: %s\n",largest\_word);

return 0;

将用户输入的单词存储在new\_word中，然后将循环的最大单词与新输入的单词比较并进行相应的替换操作。

1. **总结及心得体会：**

本章的题目较难，有的需要反复拿捏把握，需要以后在此方面多下功夫才行。并且指针是C语言的灵魂，更是要掌握牢固，到信手拈来的地步。

1. **对本实验过程及方法、手段的改进建议：**

许多问题实在以往的问题上小改进得来的，所以要多理解一些问题的本质，在此之上小改进并多考虑边界条件。

**报告评分：**

**指导教师签字：**