# 实验7 结构与联合实验

7.1、实验目的

1．通过实验，熟悉和掌握结构的说明和引用、结构的指针、结构数组、以及函数中使用结构的方法。

2．通过实验，掌握动态储存分配函数的用法，掌握自引用结构，单向链表的创建、遍历、结点的增删、查找等操作。

3．了解字段结构和联合的用法。

7.2、实验内容

**7.2.1．表达式求值的程序验证题**

设有说明：

char u[]="UVWXYZ";

char v[]="xyz";

struct T{

int x;

char c;

char \*t;

}a[]={{11,ˊAˊ,u},{100, ˊBˊ,v}},\*p=a;

请先自己计算下面表达式的值，然后通过编程计算来加以验证。(各表达式相互无关)

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **表达式** | **计算值** | **验证值** |
| 1 | (++p)->x | 100 | 100 |
| 2 | p++,p->c | B | B |
| 3 | \*p++->t,\*p->t | x | x |
| 4 | \*(++p)->t | x | x |
| 5 | \*++p->t | V | V |
| 6 | ++\*p->t | V | V |

实现程序代码：

#include<stdio.h>

int main(void)

{

char \*A, \*B, \*C, \*D;

char u[]="UVWXYZ";

char v[]="xyz";

struct T{

int x;

char c;

char \*t;

}a[]={{11,'A',u},{100,'B',v}},\*p=a;

A=u;B=v;C=a[0].t;D=a[1].t;

printf("1.(++p)->x\t%d\n",(++p)->x);

p=a;\*u=\*A;\*v=\*B;a[0].t=C;a[1].t=D;

printf("2.p++,p->c\t%c\n",(p++,p->c));

p=a;\*u=\*A;\*v=\*B;a[0].t=C;a[1].t=D;

printf("3.\*p++->t,\*p->t\t%c\n",(\*p++->t,\*p->t));

p=a;\*u=\*A;\*v=\*B;a[0].t=C;a[1].t=D;

printf("4.\*(++p)->t\t%c\n",(\*(++p)->t));

p=a;\*u=\*A;\*v=\*B;a[0].t=C;a[1].t=D;

printf("5.\*++p->t\t%c\n",(\*++p->t));

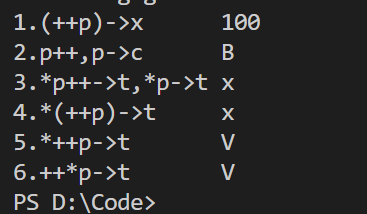
p=a;\*u=\*A;\*v=\*B;a[0].t=C;a[1].t=D;

printf("6.++\*p->t\t%c\n",(++\*p->t));

return 0;

}

运行结果：



**图7-1 程序验证题的测试的运行结果**

**2．源程序修改替换题**

给定一批整数，以0作为结束标志且不作为结点，将其建成一个先进先出的链表，先进先出链表的指头指针始终指向最先创建的结点（链头），先建结点指向后建结点，后建结点始终是尾结点。

1. 源程序中存在什么样的错误（先观察执行结果）？对程序进行修改、调试，使之能够正确完成指定任务。

源程序如下：

#include "stdio.h"

#include "stdlib.h"

struct s\_list{

int data; /\* 数据域 \*/

struct s\_list \*next; /\* 指针域 \*/

} ;

void create\_list (struct s\_list \*headp,int \*p);

void main(void)

{

struct s\_list \*head=NULL,\*p;

int s[]={1,2,3,4,5,6,7,8,0}; /\* 0为结束标记 \*/

create\_list(head,s); /\* 创建新链表 \*/

p=head; /\*遍历指针p指向链头 \*/

while(p){

printf("%d\t",p->data); /\* 输出数据域的值 \*/

p=p->next; /\*遍历指针p指向下一结点 \*/

}

printf("\n");

}

void create\_list(struct s\_list \*headp,int \*p)

{

struct s\_list \* loc\_head=NULL,\*tail;

if(p[0]==0) /\* 相当于\*p==0 \*/

;

else { /\* loc\_head指向动态分配的第一个结点 \*/

loc\_head=(struct s\_list \*)malloc(sizeof(struct s\_list));

loc\_head->data=\*p++; /\* 对数据域赋值 \*/

tail=loc\_head; /\* tail指向第一个结点 \*/

while(\*p){ /\* tail所指结点的指针域指向动态创建的结点 \*/

tail->next=(struct s\_list \*)malloc(sizeof(struct s\_list));

tail=tail->next; /\* tail指向新创建的结点 \*/

tail->data=\*p++; /\* 向新创建的结点的数据域赋值 \*/

}

tail->next=NULL; /\* 对指针域赋NULL值 \*/

}

headp=loc\_head; /\* 使头指针headp指向新创建的链表 \*/

}

head本应用来保存链表的首地址，而源代码中函数传入的是head的副本，应改为传址引用。

**解答：**

修改后代码如下：

#include <stdio.h>

#include <stdlib.h>

struct s\_list{

int data; /\* 数据域 \*/

struct s\_list \*next; /\* 指针域 \*/

} ;

void create\_list\_v1(struct s\_list \*\*headp,int \*p);

void create\_list\_v2(struct s\_list \*\*headp,int \*p);

int main(void){

struct s\_list \*head=NULL,\*p;

int s[]={1,2,3,4,5,6,7,8,0}; /\* 0为结束标记 \*/

create\_list\_v2(&head,s); /\* 创建新链表 头指针地址传递给内部函数的指针\*/

p=head; /\*遍历指针p指向链头 \*/

while(p){

printf("%d|",p->data); /\* 输出数据域的值 \*/

p=p->next; /\*遍历指针p指向下一结点 \*/

}

printf("\n");

return 0;

}

void create\_list\_v1(struct s\_list \*\*headp,int \*p){//此处纠错，双重指针，指向main函数中指针head

struct s\_list \* loc\_head=NULL,\*tail;

if(p[0]==0) /\* 相当于\*p==0 \*/

;

else{ /\* loc\_head指向动态分配的第一个结点 \*/

loc\_head=(struct s\_list \*)malloc(sizeof(struct s\_list));

loc\_head->data=\*p++; /\* 对数据域赋值 \*/

tail=loc\_head; /\* tail指向第一个结点 \*/

while(\*p){ /\* tail所指结点的指针域指向动态创建的结点 \*/

tail->next=(struct s\_list \*)malloc(sizeof(struct s\_list));

tail=tail->next; /\* tail指向新创建的结点 \*/

tail->data=\*p++; /\* 向新创建的结点的数据域赋值 \*/

}

tail->next=NULL; /\* 对指针域赋NULL值 \*/

}

\*headp=loc\_head; /\* 使头指针headp指向新创建的链表 \*/

}

void create\_list\_v2(struct s\_list \*\*headp,int \*p){//后进先出表，即栈

struct s\_list \* loc\_head=NULL,\*tail;

if(p[0]==0) /\* 相当于\*p==0 \*/

;

else{ /\* tail指向动态分配的第一个结点 \*/

loc\_head=(struct s\_list \*)malloc(sizeof(struct s\_list));

loc\_head->data=\*p++; /\* 对数据域赋值 \*/

loc\_head->next=NULL;

tail=loc\_head; /\* loc\_head指向第一个结点 \*/

while(\*p){

tail=(struct s\_list \*)malloc(sizeof(struct s\_list));

tail->next=loc\_head;

loc\_head=tail;

tail->data=\*p++;

}

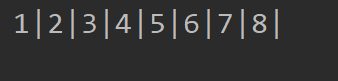
//tail->next=NULL;

}

\*headp=loc\_head;

}

替换后运行结果如下

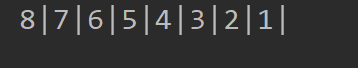


**图7-2 替换题（1）的测试的运行结果**

**（2）**修改替换create\_list函数，将其建成一个后进先出的链表，后进先出链表的头指针始终指向最后创建的结点（链头），后建结点指向先建结点，先建结点始终是尾结点。

**解答：**

程序代码如下：



**图7-3 替换题（2）的测试的运行结果**

**3．编程设计题**

**（1）**设计一个字段结构struct bits，它将一个8位无符号字节从最低位向最高位声明为8个字段，各字段依次为bit0, bit1, …, bit7，且bit0的优先级最高。同时设计8个函数，第i个函数以biti(i=0,1,2,…,7)为参数，并且在函数体内输出biti的值。将8个函数的名字存入一个函数指针数组p\_fun。如果bit0为1，调用p\_fun[0]指向的函数。如果struct bits中有多位为1，则根据优先级从高到低依次调用函数指针数组p\_fun中相应元素指向的函数。8个函数中的第0个函数可以设计为：

void f0(struct bits b)

{

Printf(“the function %d is called!\n”,b);

}

**解答：**

1. 算法思路：

此题对于函数的定义的提示有误，在解题时加以改正，输出了当前函数的编号以及传入函数的参数。同时按照题意，应该使用无符号短整形而不仅仅是无符号整形数据。此题主要利用了字段结构的特性，对一个无符号短整形进行按位操作，利用函数指针数组指向八个函数，方便操作。

1. 源程序清单

#include <stdio.h>

typedef struct bits{

unsigned short bit0 : 1;

unsigned short bit1 : 1;

unsigned short bit2 : 1;

unsigned short bit3 : 1;

unsigned short bit4 : 1;

unsigned short bit5 : 1;

unsigned short bit6 : 1;

unsigned short bit7 : 1;

} BITS;

void f0(BITS b);

void f1(BITS b);

void f2(BITS b);

void f3(BITS b);

void f4(BITS b);

void f5(BITS b);

void f6(BITS b);

void f7(BITS b);

int main() {

BITS bit= {1, 1, 1, 1, 0, 1, 0, 1};

void(\*p[8])(struct bits b);

p[0] = f0;

p[1] = f1;

p[2] = f2;

p[3] = f3;

p[4] = f4;

p[5] = f5;

p[6] = f6;

p[7] = f7;

if (bit.bit0) p[0](bit);

if (bit.bit1) p[1](bit);

if (bit.bit2) p[2](bit);

if (bit.bit3) p[3](bit);

if (bit.bit4) p[4](bit);

if (bit.bit5) p[5](bit);

if (bit.bit6) p[6](bit);

if (bit.bit7) p[7](bit);

return 0;

}

void f0(BITS b){

printf("the function 0 is called!The argv is %x\n", b);

}

void f1(BITS b){

printf("the function 1 is called!The argv is %x\n", b);

}

void f2(BITS b){

printf("the function 2 is called!The argv is %x\n", b);

}

void f3(BITS b){

printf("the function 3 is called!The argv is %x\n", b);

}

void f4(BITS b){

printf("the function 4 is called!The argv is %x\n", b);

}

void f5(BITS b){

printf("the function 5 is called!The argv is %x\n", b);

}

void f6(BITS b){

printf("the function 6 is called!The argv is %x\n", b);

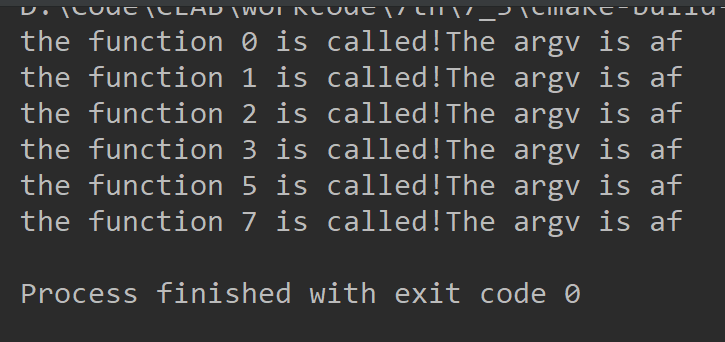
}

void f7(BITS b){

printf("the function 7 is called!The argv is %x\n", b);

}

1. 测试



**图7-4 编程题1的测试运行结果**

**（2）**用单向链表建立一张班级成绩单，包括每个学生的学号、姓名、英语、高等数学、普通物理、C语言程序设计四门课程的成绩。用函数编程实现下列功能：

(1) 输入每个学生的各项信息。

(2) 输出每个学生的各项信息。

(3) 修改指定学生的指定数据项的内容。

(4) 统计每个同学的平均成绩（保留2位小数）。

(5) 输出各位同学的学号、姓名、四门课程的总成绩和平均成绩。

**解答：**

1. 算法思路：

对于必做的程序设计题部分，首先建立一个带头结点的单链表，在创建链表时，就将学生信息输入。利用遍历链表查找学号的方式查找指定学生并修改其指定数据。自此；必做题部分基本完成。

此题难度主要集中于选做部分要求对指针域进行操作进行排序的部分。 由于在创建链表时已经添加了空的头结点，指针操作时略微简化，在更改了创建链表的函数之后，创建成双链表，可以进一步简化操作。

1. 源程序清单
   * + 1. main.c

#include "LinkedList.h"

int main(void){

StuInfoList Class8Info = malloc(sizeof(StuInfoList));

Position head = malloc(sizeof(Position));

StuInfoList sortedList = malloc(sizeof(struct StuNode));

int StuSum;

int StuNum;

printf("How many people in Class:");

scanf("%d", &StuSum);

head = newStu(StuSum);

printf("Who's data need to be change? 0 to skip");

scanf("%d", &StuNum);

if(StuNum)

Change(head, StuNum);

StuAverage(head, StuSum);

TraversalOutput(Class8Info, head);

sortedList = PtrSort(head);

TraversalOutput(Class8Info, sortedList);

return 0;

}

* + - 1. LinkedList.h

//

// Created by Think on 2017/12/27.

//

#ifndef INC\_7\_4\_LINKEDLIST\_H

#define INC\_7\_4\_LINKEDLIST\_H

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct StuNode{ //定义成绩单中链表结点

struct StuNode \*prev;

int StuNum;

char StuName[20];

int Calculus;

int English;

int Physics;

int CPL;

double Average;

struct StuNode \*next;

};

typedef struct StuNode \*PtrToNode;//类型定义指向结点的指针

typedef PtrToNode StuInfoList;//用于列表

typedef PtrToNode Position;//用于位置指示

Position FindStu(int StuNum, StuInfoList L);//寻找特定学生函数原型

StuInfoList newStu(int StuSum);//添加新学生

Position FindPre(Position head, Position target);//寻找前一个结点

int IsEmpty(StuInfoList L);

int IsLast(Position P, StuInfoList L);

void TraversalOutput(StuInfoList L, Position Head);

void PrintNode(Position P);

void Change(Position P, int StuNum);

void StuAverage(Position P, int StuSum);

void Delete(StuInfoList L, Position target);

StuInfoList SelSort(StuInfoList L);

StuInfoList PtrSort(StuInfoList L);

#endif //INC\_7\_4\_LINKEDLIST\_H

* + - 1. Function.c

//

// Created by jyxk on 2017/12/27.

//

#include "LinkedList.h"

Position FindStu(int StuNum, StuInfoList L){

Position P;

P = L->next;

while(P != NULL && P->StuNum != StuNum)

P = P->next;

return P;

}

StuInfoList newStu(int StuSum){

Position head, tail;

head = tail =(Position)malloc(sizeof(struct StuNode));

//head->prev = NULL;

tail->prev = NULL;

for(int i = 0; i < StuSum; i++){

tail->next = (Position)malloc(sizeof(struct StuNode));

tail->next->prev = tail;

tail = tail->next;

printf("Input the student's Num:");

scanf("%d", &tail->StuNum);

printf("Input his/her name:");

scanf("%s", tail->StuName);

printf("Input the grade of 4 subject:\nEnglish Calculus Physics CPL\n");

scanf("%d%d%d%d", &tail->English, &tail->Calculus, &tail->Physics, &tail->CPL);

}

tail->next = NULL;

return head;

}

int IsEmpty(StuInfoList L){

return L->next == NULL;

}

int IsLast(Position P, StuInfoList L){

return P->next == NULL;

}

void TraversalOutput(StuInfoList L, Position Head){

Position prev = Head;

while(!IsLast(prev, L)){

prev = prev->next;

PrintNode(prev);

}

prev->next = NULL;

}

void PrintNode(Position P){

Position prev = P;

printf("\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Name:%s\t", prev->StuName);

printf("StuNum:%d\n", prev->StuNum);

printf("English:%d\nCalculus:%d\nPhysics:%d\nCPL:%d\n", prev->English, prev->Calculus, prev->Physics, prev->CPL);

printf("The Average:%.2lf\n", prev->Average);

printf("\*\*\*\*\*\*\*\*\*\*\*\n");

}

void Change(Position P, int StuNum){

Position prev = FindStu(StuNum, P);

int option;

printf("What do you want to change:\n");

printf("1.English\t2.Calculus\n3.Physics\t4.CPL\n");

scanf("%d", &option);

switch(option){

case 1:

printf("please input a new mark:");

scanf("%d", &prev->English);

break;

case 2:

printf("please input a new mark:");

scanf("%d", &prev->Calculus);

break;

case 3:

printf("please input a new mark:");

scanf("%d", &prev->Physics);

break;

case 4:

printf("please input a new mark:");

scanf("%d", &prev->CPL);

break;

}

}

void StuAverage(Position P, int StuSum){

Position prev = P;

for (int i = 0; i <= StuSum; ++i) {

prev->Average = prev->CPL+prev->Physics+prev->Calculus+prev->English;

prev->Average = prev->Average / 4;

prev = prev->next;

}

}

StuInfoList SelSort(StuInfoList L){

StuInfoList p, q, small;

double temp;

int tempi;

char \*temps = malloc(sizeof(char)\*20);

for(p = L->next; p->next != NULL; p = p->next){

small = p;

for(q = p->next; q; q = q->next){

if(q->Average > small->Average)

small = q;

}

if(small != p){

temp = p->Average;

p->Average = small->Average;

small->Average = temp;

temp = p->StuNum;

p->StuNum = small->StuNum;

small->StuNum = temp;

tempi = p->English;

p->English = small->English;

small->English = tempi;

tempi = p->Calculus;

p->Calculus = small->Calculus;

small->Calculus = tempi;

tempi = p->Physics;

p->Physics = small->Physics;

small->Physics = tempi;

temp = p->CPL;

p->CPL = small->CPL;

small->CPL = temp;

strcpy(temps, p->StuName);

strcpy(p->StuName, small->StuName);

strcpy(small->StuName, temps);

}

}

return L;

}

StuInfoList PtrSort(StuInfoList L){

Position p, q;

Position temp;

for(p = L->next; p->next != NULL; p = p->next){

for(q = p->next; q->next; q = q->next){

if(q->Average > p->Average){

temp = p->prev;

p->prev = q->prev;

q->prev = temp;

temp = p->next;

p->next = q->next;

q->next = temp;

}

}

}

return L;

}

Position FindPre(Position head, Position target){

Position p = head;

while(p->next != NULL && p->next != target)

p = p->next;

return p;

}

void Delete(StuInfoList L, Position target){

Position p, temp;

p = FindPre(L, target);

if(!IsLast(p, L)){

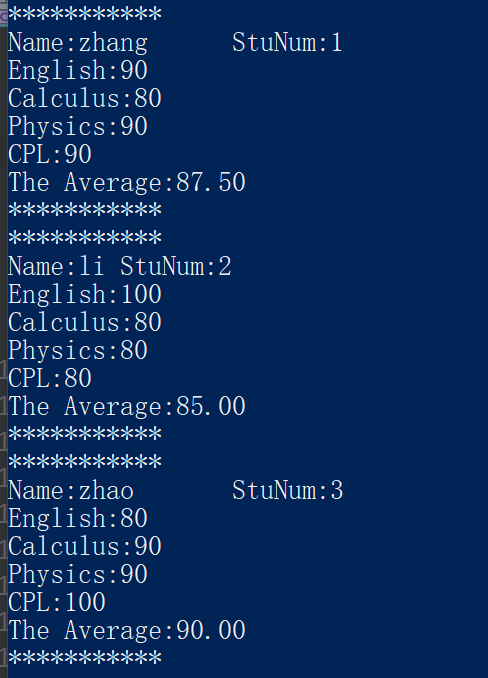
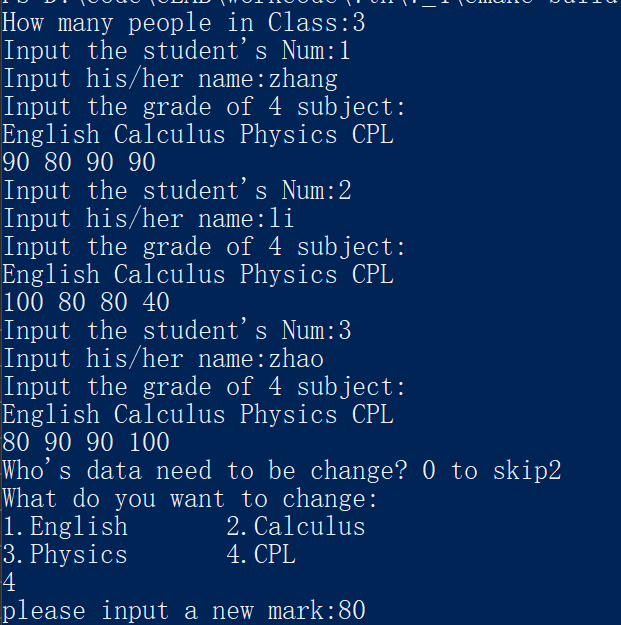
temp = p->next;

p->next = temp->next;

}

}

1. 测试



**图7-5 编程题2的测试运行结果**

**4．选做题**

（1）对编程设计题第（2）题的程序，增加按照平均成绩进行升序排序的函数，写出用交换结点数据域的方法升序排序的函数，排序可用选择法或冒泡法。

（2）对选做题第（1）题，进一步写出用交换结点指针域的方法升序排序的函数。

（3）采用双向链表重做编程设计题第（2）题。

解答：已与程序设计第二题结合完成，代码与思路见上题解答。

## 7.3 实验小结

第七次C语言上机实验，主要学习并熟练了结构的说明和引用、结构的指针、结构数组、函数中使用结构的方法、动态储存分配函数的用法、自引用结构、单向链表的创建、遍历、结点的增删、查找以及字段结构和联合的用法。

1. **验证题**

此题首先考察了对于运算符优先级以及结合性的熟悉程度，其次考察了对结构指针的用法的掌握。虽然在实际编程中不会用到类似的表达方式，但是对于我们理解与掌握基本的语法是很有好处的。

1. **程序修改替换实验**

此题向我介绍了最基本的数据结构之一：链表。在熟悉与掌握了如何构建一个简单的单链表，同时利用尾插法和头插法达到不同的效果之后，我理解了设计结构这一聚合数据类型的意义。作为抽象数据结构的一部分，一个结构体既可以储存数据，亦可以包含与前后两个结点之间的关系，方便抽象数据结构的形成。同时，对于结构指针以及动态内存分配的用法与好处也有了更深层的理解与认识。

1. **程序设计实验**

在第一题中熟悉了对于字段结构的操作之后，来到第二题。此题对于初学者来说，难度略大。首先要正确地理解结构体与结构指针，结构体与链表本身的区别。其次，要认识到在单链表中，所有操作都应从一个不变的头结点开始进行。并且要熟悉结构指针的使用。掌握了以上两点之后，我们便可开始设计各个函数，以完成不同的功能。在创建链表的函数中，一个低级的动态分配的错误耽误了很多时间，由于并没有为结构指针分配正确大小的地址，导致程序在运行时出现了很奇怪的错误，在请教老师后，发现了这一错误。同时老师在审阅我的代码的同时指出，我在无意识间创建了一个带头结点的单链表，可以为很多对于单链表的操作做很多的简化。在查阅资料，了解了头结点等概念后，我完善了对这个带头结点的链表的设计。这在选做题中，也发挥了很大的作用

1. **选做题**

有了成功解决必做题的经验，我有了充足的信心。利用选择排序解决了通过交换数据域来对链表进行排序的题目之后，我立刻从繁琐的代码实现中了解到了寻找新的方法的重要性。因此，进行交换指针域排序的函数的编写。由于此前已经创建了带头结点的链表，对于头结点的处理方便许多，但是前后一共涉及四个结点的指针操作仍然显得繁琐而复杂。在最后学习了双向链表的基础知识之后，我终于找到了实现链表排序的略微简洁的方式。同时意识到，对于简洁的行事方式的追求，也许是人们进行各种研究的动力。