**7 结构与联合实验**

7.1 实验目的

（1）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）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实验内容及要求

**7.2.1**表达式求值的程序验证

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;

请先自己计算表2.1中表达式的值，然后编写程序并运行来加以验证。(各表达式相互无关)

表2.1 表达式值的计算

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | ‘B’ | ‘B’ |
| 4 | \*(++p)->t | ‘x’ | ‘x’ |
| 5 | \*++p->t | ‘V’ | ‘V’ |
| 6 | ++\*p->t | ‘V’ | ‘V’ |

**7.2.2．**源程序修改、替换

下面所给源程序的功能是：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。请完成以下工作：

(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。

(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。

**源程序：**

#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);

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(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");

return 0;

}

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 \*/

}

（1）

**【分析】**源程序无输出，调试发现creat\_list返回时head为NULL，由此发现head传参是传入的是head 的副本，所以应该用二级指针，或者更简单的方法：使creat\_list有返回值，返回头结点的指针。

**【替换】**

#include "stdio.h"

#include "stdlib.h"

struct s\_list{

int data; /\* 数据域 \*/

struct s\_list \*next; /\* 指针域 \*/

} ;

struct s\_list \* create\_list (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 \*/

head = 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");

return 0;

}

struct s\_list \* 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 \*/

}

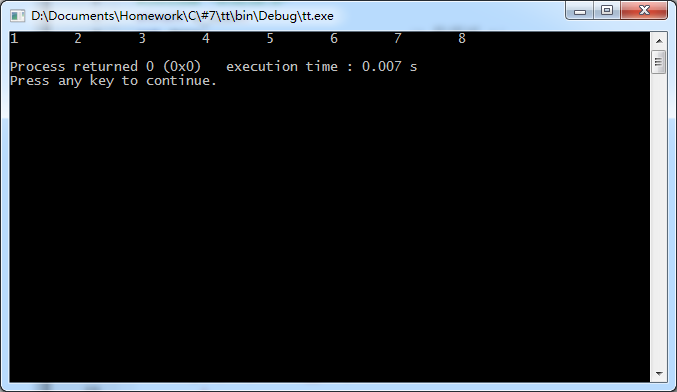
tail->next=NULL; /\* 对指针域赋NULL值 \*/

}

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

}

**【测试】**修改后可以正常输出队列。



（2）

**【分析】**单项链表方向反向，头尾指针互换即可

**【替换】**

#include "stdio.h"

#include "stdlib.h"

struct s\_list{

int data;

struct s\_list \*next;

} ;

struct s\_list \* create\_list (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};

head = create\_list(head,s);

p=head;

while(p){

printf("%d\t",p->data);

p=p->next;

}

printf("\n");

return 0;

}

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

{

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

if(p[0]==0)

;

else {

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

loc\_head->data=\*p++;

loc\_head->next=NULL;

tail=loc\_head;

while(\*p){

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

loc\_head->next=tail;

tail=loc\_head;

loc\_head->data=\*p++;

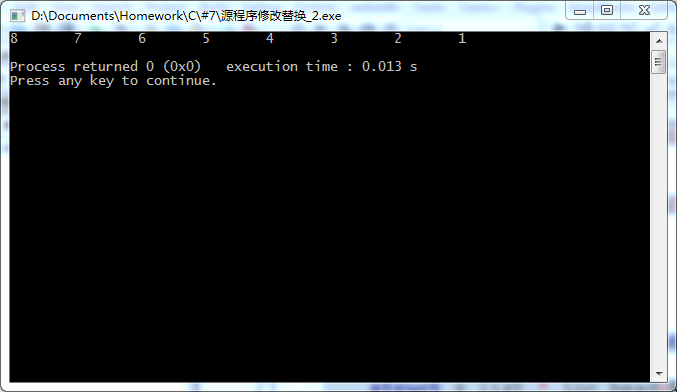
}

}

return loc\_head;

}

**【测试】**



**7.2.3．**程序设计

编写并上机调试运行能实现以下功能的程序或函数：

(1) 编写一个程序,实现以下功能：定义一个字段结构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 0 is called! %d\n”,b.bit0);

}

(2) 假设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,试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。

（1）

**【分析】**将一个无符号字符型的各个位依次赋值给结构中的每一个成员变量即可.

**【程序】**

#include<stdio.h>

typedef struct bits{

unsigned char identity;

int bit0;

int bit1;

int bit2;

int bit3;

int bit4;

int bit5;

int bit6;

int bit7;

}Bits;

void f0(struct bits b);

void f1(struct bits b);

void f2(struct bits b);

void f3(struct bits b);

void f4(struct bits b);

void f5(struct bits b);

void f6(struct bits b);

void f7(struct bits b);

int main(void){

Bits init;

init.identity = 0xAA;

init.bit0 = (init.identity&(0x01<<0))>>0;

init.bit1 = (init.identity&(0x01<<1))>>1;

init.bit2 = (init.identity&(0x01<<2))>>2;

init.bit3 = (init.identity&(0x01<<3))>>3;

init.bit4 = (init.identity&(0x01<<4))>>4;

init.bit5 = (init.identity&(0x01<<5))>>5;

init.bit6 = (init.identity&(0x01<<6))>>6;

init.bit7 = (init.identity&(0x01<<7))>>7;

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

p\_fun[0]=f0;

p\_fun[1]=f1;

p\_fun[2]=f2;

p\_fun[3]=f3;

p\_fun[4]=f4;

p\_fun[5]=f5;

p\_fun[6]=f6;

p\_fun[7]=f7;

(init.bit0)?p\_fun[0](init):0;

(init.bit1)?p\_fun[1](init):0;

(init.bit2)?p\_fun[2](init):0;

(init.bit3)?p\_fun[3](init):0;

(init.bit4)?p\_fun[4](init):0;

(init.bit5)?p\_fun[5](init):0;

(init.bit6)?p\_fun[6](init):0;

(init.bit7)?p\_fun[7](init):0;

return 0;

}

void f0(struct bits b)

{

printf("the function 0 is called! %d\n",b.bit0);

}

void f1(struct bits b)

{

printf("the function 1 is called! %d\n",b.bit1);

}void f2(struct bits b)

{

printf("the function 2 is called! %d\n",b.bit2);

}void f3(struct bits b)

{

printf("the function 3 is called! %d\n",b.bit3);

}void f4(struct bits b)

{

printf("the function 4 is called! %d\n",b.bit4);

}void f5(struct bits b)

{

printf("the function 5 is called! %d\n",b.bit5);

}void f6(struct bits b)

{

printf("the function 6 is called! %d\n",b.bit6);

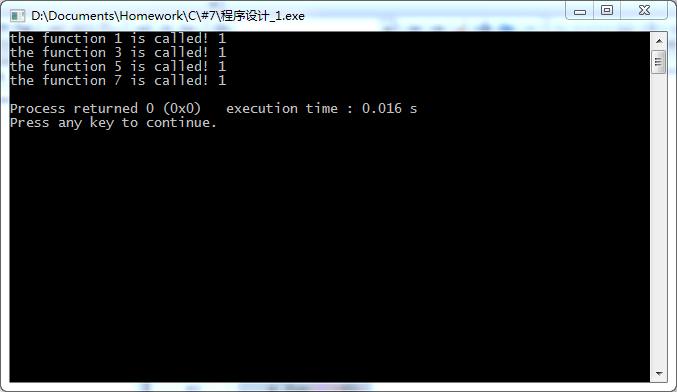
}void f7(struct bits b)

{

printf("the function 7 is called! %d\n",b.bit7);

}

**【测试】**若变量为0xAA,则函数1、3、5、7应该被调用，输出与预想一致。



（2）

**【分析】**使用简单单项链表，每个节点存储一个学生的信息以及指向下一个节点的指针即可。

**【程序】**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct stu{

char num[16];

char name[21];

float E;

float M;

float P;

float C;

struct stu\* next;

}Stu;

int main(void){

int N;

scanf("%d",&N);

Stu \* head = (Stu \*)malloc(sizeof(Stu));

head->num[0]='\0';

Stu \*temp = head;

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

temp->next = (Stu \*)malloc(sizeof(Stu));

temp = temp->next;

scanf("%s %s %f %f %f %f",(char \*)temp->num,(char \*)temp->name,&temp->E,&temp->M,&temp->P,&temp->C);

}

printf("\nID Name English Math Physics C \n");

temp = head;

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

temp = temp->next;

printf("%-15s%-20s%-10.2f%-10.2f%-10.2f%-10.2f\n",temp->num,temp->name,temp->E,temp->M,temp->P,temp->C);

}

int M;

char num[16],sub[20];

float score;

scanf("%d",&M);

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

scanf("%s %s %f",(char \*)num,(char \*)sub,&score);

for(temp = head->next;strcmp(num,temp->num)!=0;temp = temp->next);

switch(sub[0]){

case 'E':

temp->E = score;

break;

case 'M':

temp->M = score;

break;

case 'P':

temp->P = score;

break;

case 'C':

temp->C = score;

break;

}

}

printf("\nAlter:\nID Name English Math Physics C \n");

temp = head;

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

temp = temp->next;

printf("%-15s%-20s%-10.2f%-10.2f%-10.2f%-10.2f\n",temp->num,temp->name,temp->E,temp->M,temp->P,temp->C);

}

printf("\nSumAndAvg:\nID Name SUM AVG \n");

temp = head;

float sum;

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

temp = temp->next;

sum=(temp->E+temp->M+temp->P+temp->C);

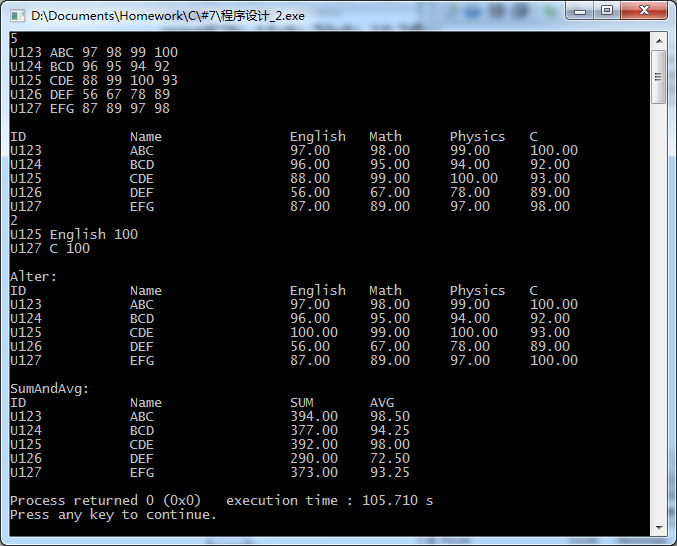
printf("%-15s%-20s%-10.2f%-10.2f\n",temp->num,temp->name,sum,sum/4.0);

}

return 0;

}

**【设计】**采用知实际数据测试，结果与预想一致。



7.2.4．选做题

(1) 对上述程序设计题中第（2）题的程序，增加按照平均成绩进行升序排序的函数，试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中的第（2）题。

**【分析】**采用交换数据域或交换指针域方法类似，为方便起见，这里只写出交换指针域的方法，交换数据域的方法在注释中给出。

**【程序】**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct stu{

char num[16];

char name[21];

float E;

float M;

float P;

float C;

struct stu\* next;

}Stu;

int main(void){

int N;

scanf("%d",&N);

Stu \* head = (Stu \*)malloc(sizeof(Stu));

head->num[0]='\0';

Stu \*temp = head;

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

temp->next = (Stu \*)malloc(sizeof(Stu));

temp = temp->next;

scanf("%s %s %f %f %f %f",(char \*)temp->num,(char \*)temp->name,&temp->E,&temp->M,&temp->P,&temp->C);

}

printf("\nID Name English Math Physics C \n");

temp = head;

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

temp = temp->next;

printf("%-15s%-20s%-10.2f%-10.2f%-10.2f%-10.2f\n",temp->num,temp->name,temp->E,temp->M,temp->P,temp->C);

}

int M;

char num[16],sub[20];

float score;

scanf("%d",&M);

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

scanf("%s %s %f",(char \*)num,(char \*)sub,&score);

for(temp = head->next;strcmp(num,temp->num)!=0;temp = temp->next);

switch(sub[0]){

case 'E':

temp->E = score;

break;

case 'M':

temp->M = score;

break;

case 'P':

temp->P = score;

break;

case 'C':

temp->C = score;

break;

}

}

printf("\nAlter:\nID Name English Math Physics C \n");

temp = head;

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

temp = temp->next;

printf("%-15s%-20s%-10.2f%-10.2f%-10.2f%-10.2f\n",temp->num,temp->name,temp->E,temp->M,temp->P,temp->C);

}

printf("\nSumAndAvg:\nID Name SUM AVG \n");

temp = head;

float sum;

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

temp = temp->next;

sum=(temp->E+temp->M+temp->P+temp->C);

printf("%-15s%-20s%-10.2f%-10.2f\n",temp->num,temp->name,sum,sum/4.0);

}

temp = head;

Stu\* p[N];

for(int i=0;i<N;i++)

temp = head->next,p[i] = temp;

int tt1,tt2;

for(tt1=0;tt1<N-1;tt1++)

for(tt2=tt1+1;tt2<N;tt2++)

if( ((p[tt1])->E+(p[tt1])->M+(p[tt1])->P+(p[tt1])->C) > ((p[tt2])->E+(p[tt2])->M+(p[tt2])->P+(p[tt2])->C) )

temp = p[tt1];p[tt1]=p[tt2],p[tt2]=temp;

// temp = malloc(sizeof(Stu);

// for(tt1=0;tt1<N-1;tt1++)

// for(tt2=tt1+1;tt2<N;tt2++)

// if( ((p[tt1])->E+(p[tt1])->M+(p[tt1])->P+(p[tt1])->C) > ((p[tt2])->E+(p[tt2])->M+(p[tt2])->P+(p[tt2])->C) ){

// memcpy(temp,p[tt1],sizeof(Stu));

// memcpy(p[tt1],p[tt2],sizeof(Stu));

// memcpy(p[tt2],temp,sizeof(Stu));

// }

printf("\nSort:\nID Name AVG \n");

temp = head;

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

temp = temp->next;

sum=(temp->E+temp->M+temp->P+temp->C);

printf("%-15s%-20s%-10.2f\n",temp->num,temp->name,sum/4.0);

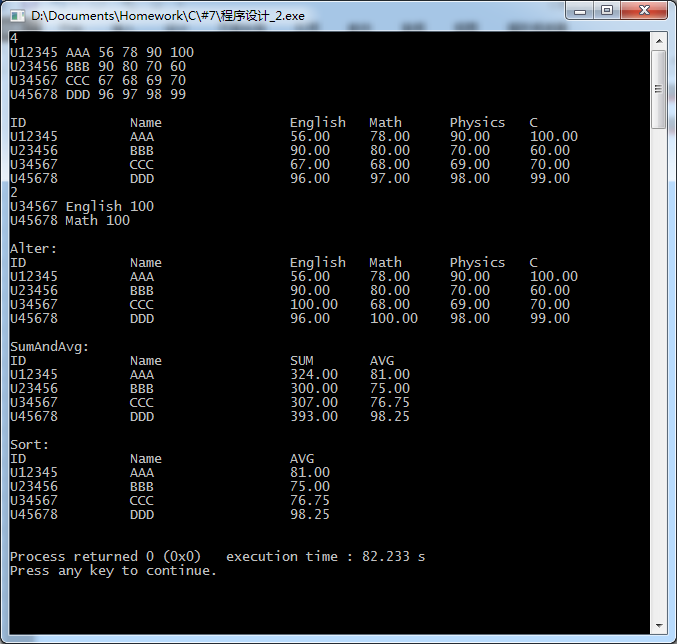
}

putchar('\n');

return 0;

}

**【测试】**采用4组数据测试，功能均能实现



**（3）**

**【分析】**使用双向链表可以在录入成绩的时候直接根据平均成绩进行插入排序，省去了根据平均成绩对链表进行排序的步骤，较为便捷，但由于程序内容与前两题极为相似，故不在列出。

**7.3实验小结**

通过本次试验，我掌握了结构的说明和引用、结构的指针、结构数组、以及函数中使用结构的方法，并掌握了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、查找等操作。这对项目的看至及以后数据结构的学习有很大的帮助，结构的使用可以使编程的某些部分（如逻辑，操作等）更为简洁、清晰。