

北京交通大学

程序设计分组训练 实验 5 实验报告

专 业： 计算机科学与技术

班 级：

学生姓名：

学 号：

北京交通大学计算机与信息技术学院

2021 年 12 月 14 日

目录

| | |
|---------------------------------|----|
| 程序设计分组训练 实验 5 实验报告 | 1 |
| 1. 相关知识准备 system 函数使用 | 1 |
| 2. 实验操作 | 2 |
| 2.1 思路 | 2 |
| 2.2 Judgefiletype 函数判断拓展名 | 3 |
| 2.3 读入数据 以要求的数据结构储存 | 4 |
| 2.4 四种数据结构对应的 show () 函数 | 8 |
| 2.5 自动模式与交互模式 | 10 |
| 2.6 结构体声明 与 mode.ini 文件 | 10 |
| 3. 问题解答 | 11 |
| 3.1 程序自检 | 11 |
| 3.2 读入实验 4 生成的数据记录文件 | 12 |
| 3.3 修改配置文件 | 13 |
| 4. 心得体会 | 15 |
| 4.1 自身收获 | 15 |
| 4.2 小组互评 | 18 |

1.相关知识准备 system 函数使用

函数名: `system`

功 能: 发出一个 DOS 命令

用 法: `int system(char *command);`

可以在程序内部实现类似于 `cmd` 命令行的操作。在本次实验中可以由此调用 `lab4.exe`，传入值为路径。

2. 实验操作

2.1 思路

根据图 1-1 之流程，对照实验操作，建立菜单供用户选择功能。效果图如下：

Active code page: 65001

张芝玮的实验5程序：

1. 调用实验 4 程序生成记录文件（文本方式）
2. 调用实验 4 程序生成记录文件（二进制方式）
3. 读取指定数据记录文件（二维数组存储方式）
4. 读取指定数据记录文件（结构体数组存储方式）
5. 读取指定数据记录文件（指针数组存储方式）
6. 读取指定数据记录文件（链表存储方式）
7. 调用实验 4 生成数据记录文件，同时读取数据记录文件（文本方式输出，二维数组方式存储）
8. 调用实验 4 生成数据记录文件，同时读取数据记录文件（文本方式输出，结构体数组方式存储）
9. 调用实验 4 生成数据记录文件，同时读取数据记录文件（文本方式输出，指针数组方式存储）
10. 调用实验 4 生成数据记录文件，同时读取数据记录文件（文本方式输出，链表方式存储）
11. 调用实验 4 生成数据记录文件，同时读取数据记录文件（二进制方式输出，二维数组方式存储）
12. 调用实验 4 生成数据记录文件，同时读取数据记录文件（二进制方式输出，结构体数组方式存储）
13. 调用实验 4 生成数据记录文件，同时读取数据记录文件（二进制方式输出，指针数组方式存储）
14. 调用实验 4 生成数据记录文件，同时读取数据记录文件（二进制方式输出，链表方式存储）
15. 重新设置配置参数值
0. 退出

请输入您要执行的程序序号：

图 2.1-1

由于 `clion` 对中文支持不是太好，之前的使用中文注释总是乱码，这次本想直接使用全英文菜单，但考虑到菜单样式为实验要求的一部分，遂潜心钻研，最后也是使用 `system` 函数解决。

Main 函数中的代码加入：`system("chcp 65001");`

Run 函数中 `switch-case` 的运用恰到好处。

部分截图如下（屏幕太小，代码太长，无法截全）

```
switch(num)
{
    case 1:
        filetype = callLab4txt(path,data); //return 1 signals txt
        break;
    case 2:
        filetype = callLab4dat(path,data); //return 2 signals dat
        break;
    case 3:
        strcpy(path,choose(path));
        filetype = Judgefiletype(path);
        loadTo2DArray(path, filetype);
        break;
    case 4:
        strcpy(path,choose(path));
        filetype = Judgefiletype(path);
        loadToStructArray(path, filetype);
        break;
    case 5:
        strcpy(path,choose(path));
        filetype = Judgefiletype(path);
        loadToPointerArray(path, filetype);
        break;
}
```

图 2.1-2

2.2 Judgefiletype 函数判断拓展名

```
01:///
02:/// \param s pave/name of the file
03:/// \return 1 signals txt; 2 signals dat
04:int Judgefiletype(char *s) {
05:    if(s[strlen(s)-3]=='t')
06:        return 1;
07:    else
```

```
08:     return 2;
09: }
```

2.3 读入数据 以要求的数据结构储存

二维数组 loadtoarray ()

```
void loadtoarray(char *path, int filetype) {
    FILE * pf;
    int record_num;
    int **item;
    if(filetype==1)
    {
        pf = fopen(path, "r");
        if(!pf)
        {
            printf("file open error1\n");
            exit(1);
        }
        fscanf(pf, "%d", &record_num);

        item = (int **) malloc(sizeof (item)*record_num);

        for(int i=0;i<record_num; ++i)
        {
            item[i] = (int *)malloc(3*sizeof (int ));

            //read in format control (cheak in outputfile)
            fscanf(pf, "%d %d %d\n", &item[i][0], &item[i][1], &item[i][2]);
        }
        fclose(pf);
    }
    else{
        pf = fopen(path, "rb");
        if(!pf)
        {
            printf("file open error1\n");
            exit(1);
        }
    }
}
```

```

        fread(&record_num, 4, 1, pf);
        item =(int **) malloc(sizeof(item)*record_num);
        for (int i = 0; i < record_num; ++i) {
            item[i] = (int *) malloc(sizeof(int)*3);
            fread(&item[i], sizeof (int )*3, 1, pf );
        }
        fclose(pf);
    }
    Show2DArray(item, record_num);
    for (int i = 0; i < record_num; ++i) {
        free(item[i]);
    }
    free(item);
}

```

结构体数组 loadtostructarray ()

```

loadtostructarray ( )
void loadtostructarray(char *path, int filetype) {
    FILE * pf;
    int record_num;
    DATAITEM * item;
    if(filetype==1) {
        pf = fopen(path, "r");
        if (!pf) {
            printf("file open error2\n");
            exit(1);
        }
        fscanf(pf, "%d", &record_num);

        item = (DATAITEM *) malloc(sizeof(DATAITEM) * record_num);
        for (int i = 0; i < record_num; ++i) {
            fscanf(pf, "%d %d %d\n", &item[i].item1, &item[i].item2,
&item[i].item3);
        }
        fclose(pf);
    }
    else{
        pf = fopen(path, "rb");
        if(!pf)

```

```

    {
        printf("file open error2\n");
        exit(1);
    }
    fread(&record_num, 4, 1, pf);
    item = (DATAITEM*) malloc(sizeof(DATAITEM)*record_num);
    for (int i = 0; i < record_num; ++i) {
        fread(&item[i].item1, 4, 1, pf);
        fread(&item[i].item2, 4, 1, pf);
        fread(&item[i].item3, 4, 1, pf);
    }
    fclose(pf);
}
showStructArray(item, record_num);
free(item);
}

```

结构体指针数组 loadtopointerarray

```

void loadtopointerarray(char *path, int filetype) {
    FILE * pf;
    int record_num;
    DATAITEM **item;
    if(filetype==1) {
        pf = fopen(path, "r");
        if (!pf) {
            printf("file open error3\n");
            exit(1);
        }
        fscanf(pf, "%d", &record_num);
        item = (DATAITEM **) malloc(sizeof(DATAITEM *) * record_num);
        for (int i = 0; i < record_num; ++i) {
            item[i] = (DATAITEM *) malloc(sizeof(DATAITEM));
            fscanf(pf, "%d %d %d", &item[i]->item1, &item[i]->item2,
&item[i]->item3);
        }
        fclose(pf);
    }
    else{
        pf = fopen(path, "rb");
        if(!pf)
        {

```

```

        printf("file open error3\n");
        exit(1);
    }
    fread(&record_num, 4, 1, pf);
    item = (DATAITEM**)malloc(sizeof (DATAITEM*)*record_num);
    for (int i = 0; i < record_num; ++i) {
        item[i] = (DATAITEM *) malloc(sizeof(DATAITEM));
        fread(&item[i]->item1, 4, 1, pf);
        fread(&item[i]->item2, 4, 1, pf);
        fread(&item[i]->item3, 4, 1, pf);
    }
    fclose(pf);
}
ShowPointerArray(item, record_num);
for (int i = 0; i < record_num; ++i) {
    free(item[i]);
}
free(item);
}

```

链表 loadtolink ()

```

void loadtolink(char *path, int filetype) {
    FILE *pf;
    int record_num;
    Link *head = (Link *) malloc(sizeof(Link));

    if (filetype == 1) {
        pf = fopen(path, "r");
        if (!pf) {
            printf("file open error4\n");
            exit(1);
        }
        fscanf(pf, "%d", &record_num);
        head->totaldataum = record_num;
        Node *pre = (Node *) malloc(sizeof(Node));
        fscanf(pf, "%d %d %d", &pre->data[0], &pre->data[1], &pre->data[2]);
        head->next = pre;

        for (int i = 0; i < record_num; ++i) {
            Node *p = (Node *) malloc(sizeof(Node));
            fscanf(pf, "%d %d %d", &p->data[0], &p->data[1], &p->data[2]);

```

```
        pre->next = p;
        pre = p;
    }
    pre->next = NULL;
    fclose(pf);
}
else{
    pf = fopen(path, "rb");
    if(!pf)
    {
        printf("file open error4\n");
        exit(1);
    }
    fread(&record_num, 4, 1, pf);

    Link * head = (Link*) malloc(sizeof(Link));
    head->totaldataum = record_num;
    Node * pre = (Node*) malloc(sizeof (Node));
    fread(&pre->data[0], 4, 1, pf);
    fread(&pre->data[1], 4, 1, pf);
    fread(&pre->data[2], 4, 1, pf);
    head->next = pre;

    for (int i = 0; i < record_num; ++i) {
        Node * p = (Node*) malloc(sizeof(Node));
        fread(&p->data[0], 4, 1, pf);
        fread(&p->data[1], 4, 1, pf);
        fread(&p->data[2], 4, 1, pf);
        pre->next = p;
        pre = p;
    }
    pre->next = NULL;
    fclose(pf);
    ShowLink(head->next);
}
}
```

2.4 四种数据结构对应的 show () 函数

二维数组 showarray

```
01:void showarray(int **a, int record_num) {
```

```
    for(int i=0; i<record_num; ++i)
    {
        printf("%d, %d, %d\n", a[i][0], a[i][1], a[i][2]);
    }
}
```

结构体数组 showstructarray

```
01: void showstructarray(DATAITEM *a, int record_num) {
02:     for(int i=0 ; i<record_num; ++i)
03:     {
04:         printf("%d, %d, %d\n", a[i].item1, a[i].item2, a[i].item3);
05:     }
06:}}
```

指针数组 showpointerarray

```
07: void showpointerarray(DATAITEM **a, int record_num) {
08:     for(int i =0; i<record_num; ++i)
09:     {
10:         printf("%d, %d, %d", a[i]->item1, a[i]->item2, a[i]->item3);
11:     }
12:}
```

链表 showlink

```
13: void showlink(Node *pHead) {
14:     Node * p = pHead;
15:     while(!p)
16:     {
17:         printf("%d, %d, %d\n", p->data[0], p->data[1], p->data[2]);
18:         p = p->next;
19:     }
20:}
```

2.5 自动模式与交互模式

在自动模式下，`mode.ini` 中为 1，默认使用 `conf.ini` 中默认的信息。

在交互模式下，使用菜单提示信息来接受用户的输入，如文件路径、数据条数信息等。

代码：

```
01:
02:     else if(mode == 1&&num!=0&&num!=15)//自动模式
03:     {
04:         strcpy(path,choose(path));//默认路径
05:         strcpy(data,"r");//数据条数随机
06:         num = 7;
07:     }
08: if(mode == 2&&num!=0&&num!=15)//交互模式的信息收取
09:     {
10:         printf("请输入数据记录文件的路径(不含文件名): \n");
11:         scanf("%s",path);
12:         printf("请输入文件名: \n");
13:         scanf("%s",filename);
14:         printf("请输入文件记录条数: \n");
15:         scanf("%s",data);
16:         strcat(path,"\\");
17:         strcat(path,filename);
18:     }
19:     else if(mode == 1&&num!=0&&num!=15)//自动模式
20:     {
21:         strcpy(path,choose(path));//默认路径
22:         strcpy(data,"r");//数据条数随机
23:         num = 7;
24:     }
```

2.6 结构体声明 与 `mode.ini` 文件

结构体声明如下：

```
01: typedef struct configinfo
02: {
03:     char filesavepath[100];
04:     char filename[100];
```

```
05:    int maxvalue1;
06:    int minvalue1;
07:    int maxvalue2;
08:    int minvalue2;
09:    int recordcount1;
10:    int recordcount2;
11:}CONF;
12:
13:typedef struct{
14:    int item1;
15:    int item2;
16:    int item3;
17:}DATAITEM;
18:
19:typedef struct Node{
20:    int data[3];
21:    Node *next;
22:};
23:
24:typedef struct {
25:    int totaldatanum;
26:    Node * next;
27:}Link;
```

为保证移植性，路径全部为相对路径
\\ cmake-build-debug\\mode.ini
其中数据默认为 1 即自动模式。

3. 问题解答

3.1 程序自检

本次实验中的外部条件自然就是 `conf.ini` 和 `mode.ini`，需要检测两个文件中的信息是否正常。

代码如下：

```
01: void selfcheck(){
02:     FILE *pf1 = fopen("conf.ini", "r");
03:     if(!pf1)
04:     {
05:         printf("file open error\n");
06:         exit(1);
07:     }
08:     FILE *pf2 = fopen("mode.ini", "r");
09:     if(!pf2)
10:     {
11:         printf("file open error\n");
12:         exit(1);
13:     }
14:
15:     int choice;
16:     fscanf(pf2, "%d", &choice);
17:     if(choice != 1 && choice != 2)
18:     {
19:         printf("the mode.ini file error\n");
20:         exit(0);
21:     }
22:     fclose(pf2);
23:     fclose(pf1);
24: }
```

3.2 读入实验 4 生成的数据记录文件

自动模式下, **test4.exe** 按默认情况保存, 则建立一 **choose** 函数按默认情况以相对路径打开即可。

交互模式下, 保存用户输入的路径, 以此打开即可。

```
01: char *choose(char *str) {
02:     static char file1[100];
03:     char filesavepath[100];
04:     char filename[100];
05:     if(strcmp(str, "\\0") == 0)
06:     {
07:         FILE *fp = fopen("conf.ini", "r");
08:         if (fp == NULL)
```

```

09:     {
10:         printf("file open error\n");
11:         exit(1);
12:     }
13:     fscanf(fp,"%s\n",filesavepath);
14:     fscanf(fp,"%s",filename);
15:     strcpy(file1,filesavepath);
16:     strcat(file1,"\\");
17:     strcat(file1,filename);
18: }
19: else
20:     strcpy(file1, str);
21: return file1;
22:}

```

3.3 修改配置文件

修改实验 4 的 `conf.ini` 文件。

由菜单提示信息给出,由用户选择修改哪些参数,一次修改一个。在程序中,则是读入所有 `conf.ini`,按行修改,再全部写入。

代码:

```

01://switch—case中的
02:case 15:
03:     printf("请选择需要改变的参数: \n"
04:         "1.数据记录文件路径\n"
05:         "2.数据记录文件名\n"
06:         "3.数据记录三元组中第1、2个元素取值的上限\n"
07:         "4.数据记录三元组中第1、2个元素取值的下限\n"
08:         "5.数据记录三元组中第3个元素取值的上限\n"
09:         "6.数据记录三元组中第3个元素取值的下限\n"
10:         "7.记录条数时条数值的上限\n"
11:         "8.记录条数时条数值的下限\n"
12:         "9.工作模式(自动模式/交互模式) \n"
13:         "请输入您要执行的程序序号:\n");
14:void Changeconf(int line)
15:{
16:    FILE *pf = fopen("conf.ini", "r");
17:    if(!pf)
18:    {

```

```
19:     printf("file open error\n");
20:     exit(1);
21: }
22:
23: char command[8][30];
24: for(int i=0; i<8; ++i)
25: {
26:     fscanf(pf, "%s", command[i]);
27: }
28: fclose(pf);
29:
30: if(line>=1 && line <=8)
31: {
32:     if(line == 1)
33:         printf("please enter new file save path\n");
34:     else if(line == 2)
35:         printf("please enter new file name\n");
36:     else if(line == 3)
37:         printf("enter fist, second elements' max\n");
38:     else if(line == 4)
39:         printf("enter fist, second elements' min\n");
40:     else if(line == 5)
41:         printf("enter third element's max\n");
42:     else if(line == 6)
43:         printf("enter third element's min\n");
44:     else if(line == 7)
45:         printf("enter the max of record count number\n");
46:     else if(line == 8)
47:         printf("enter the min of record count number\n");
48:     scanf("%s",command[line-1]);
49: }
50: pf = fopen("conf.ini", "w");
51: if(!pf)
52: {
53:     printf("file open error\n");
54:     exit(1);
55: }
56: for(int i=0; i<8; ++i)
57: {
58:     fprintf(pf, "%s\n", command[i]);
59: }
60: fclose(pf);
61: printf("change success\n");
```



```
62:}
```

修改 mode.ini 文件

Switch-case 提示信息上边一起给出，功能主要以 changemode () 函数完成

代码：

```
01:void ChangeMode(){
02:    printf("please choose one operating mode\n");
03:    printf("1 auto mode\n");
04:    printf("2 interactive mode\n");
05:    printf("please enter the choice\n");
06:    int choice;
07:    scanf("%d", &choice);
08:    FILE *pf = fopen("mode.ini", "w");
09:    if(pf==NULL){
10:        printf("file open error\n");
11:        exit(1);
12:    }
13:    fprintf(pf, "%d", choice);
14:    fclose(pf);
15:    printf("change success\n");
16:}
```

4. 心得体会

4.1 自身收获

随着学习的深入，实验的要求也越发繁杂，但就算法要求本身而言其实并不是很难，主要就是各个功能之间的配合，也就是接口比较复杂，并且最后做菜单的时候篇幅也比较长。这里我对工程任务的分解的重要性和接口的一般性的理解更加深入。首先是通读实验要求，明确任务，做出程序框图进行指引，最重要的

是分解任务，以不同的函数去完成各个小任务，这样一次只关注一个小部分，完成了整体到局部的对立统一。还有必要性注释的书写，每隔几行或重要部位，写上几句注释，不管什么时候来看，都知道当时为什么这样写，也便于后期的维护和优化。

这样的学习对之后的工程性代码的完成也有很大的助力。

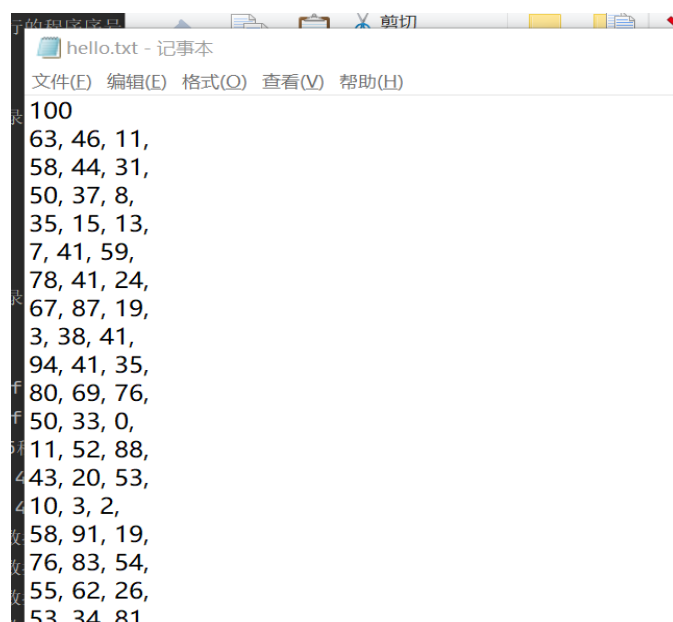
附上，运行结果图：（功能过多不能一一展示，可自行运行）

```

10.调用实验 4 生成数据记录文件，同时读取数据记录文件（文本方式输出，链表方式存储）
11.调用实验 4 生成数据记录文件，同时读取数据记录文件（二进制方式输出，二维数组方式存储）
12.调用实验 4 生成数据记录文件，同时读取数据记录文件（二进制方式输出，结构体数组方式存储）
13.调用实验 4 生成数据记录文件，同时读取数据记录文件（二进制方式输出，指针数组方式存储）
14.调用实验 4 生成数据记录文件，同时读取数据记录文件（二进制方式输出，链表方式存储）
15.重新设置配置参数值
0. 退出|
请输入您要执行的程序序号：
1
请输入数据记录文件的路径(不含文件名):
moren
moren
请输入文件名:
hello.txt
hello.txt
请输入文件记录条数:
100
100
duration of generate random number is 0.000000
duration of writing to txt file is 0.001000
张芝玮的实验5程序：
1. 调用实验 4 程序生成记录文件（文本方式）

```

图 4.1-1



```

100
63, 46, 11,
58, 44, 31,
50, 37, 8,
35, 15, 13,
7, 41, 59,
78, 41, 24,
67, 87, 19,
3, 38, 41,
94, 41, 35,
80, 69, 76,
50, 33, 0,
11, 52, 88,
443, 20, 53,
410, 3, 2,
58, 91, 19,
76, 83, 54,
55, 62, 26,
53 34 81

```

图 4.1-2

4.2 小组互评

这次临近期中，小组成员其他杂事都比较多，导致实验进度缓慢。由此，我们应借助小组合作机制建立相互督促机制。

张开元同学的报告最后还是完成的比较好的，他的代码清晰，构成逻辑严密。因为是相互交流比较多，所以不借助注释也可以很快看明白。这恰恰说明小组合作的重要性。

王麒越同学的报告完成的很好，只是在变量命名上还不能做到见名知义，还需要一起多努力。