程序设计分组训练 实验 4 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 10 月 30 日

目录

1.相关知识准备

1.1 二进制文件与文本文件

在定义和存取方式上二进制文件与文本文件存在区别。

定义上的区别

文本文件：文本文件是一种计算机文件，它是一种典型的顺序文件，其文件的逻辑结构又属于流式文件。简单的说，文本文件是基于字符编码的文件，常见的编码有 ASCII 编码，UNICODE 编码等等。

二进制文件：是基于值编码的文件，你可以根据具体应用，指定某个值是什么意思（这样一个过程，可以看作是自定义编码）。用户一般不能直接读懂它们，只有通过相应的软件才能将其显示出来。二进制文件一般是可执行程序、图形、图像、声音等等。

从上面可以看出文本文件与二进制文件的区别并不是物理上的，而是逻辑上的。这两者只是在编码层次上有差异，文本文件基本上是定长编码的(也有非定长的编码如 UTF-8)。而二进制文件则可看成是变长编码，因为是值编码，多少个比特代表一个值，完全由你决定。

存储方式上的区别

文本工具打开一个文件，首先读取文件物理上所对应的二进制比特流，然后按照所选择的解码方式来解释这个流，然后将解释结果显示出来。

一般来说，你选取的解码方式会是 ASCII 码形式（ASCII 码的一个字符是 8 个比特），接下来，它 8 个比特 8 个比特地来解释这个文件流。

记事本无论打开什么文件都按既定的字符编码工作（如 ASCII 码），所以当他打开二进制文件时，出现乱码也是很必然的一件事情了，解码和译码不对应。

文本文件的存储与其读取基本上是个逆过程。而二进制文件的存取与文本文件的存取差不多，只是编/解码方式不同而已。

二进制文件就是把内存中的数据按其在内存中存储的形式原样输出到磁盘中存放，即存放的是数据的原形式。文本文件是把数据的终端形式的二进制数据输出到磁盘上存放，即存放的是数据的终端形式

1.2 程序计时函数之 clock 函数

C 库函数 clock_t clock(void) 返回程序执行起（一般为程序的开头），处理器时钟所使用的时间。为了获取 CPU 所使用的秒数，您需要除以 CLOCKS_PER_SEC。

本程序中计时代码如下：(可视为伪代码，中间部分代码未展示)

```
01:#include <time.h>
02:clock_t start, end;
03:    double duration;
04:start = clock();
05:……
06:end = clock();
07:    duration = (double )(end - start)/CLK_TCK;
08:    printf("duration of generate random number is %f\n", duration);
```

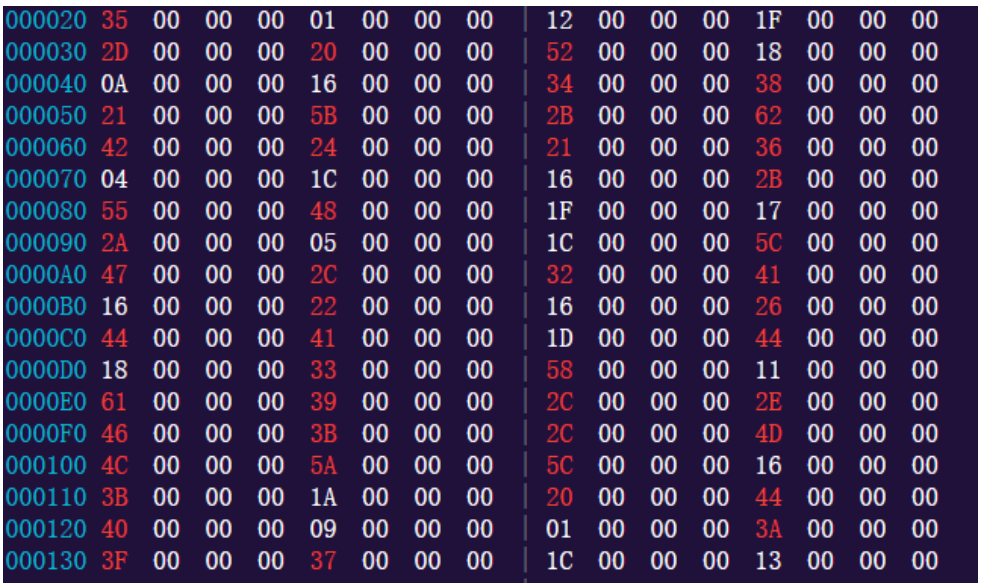
1.3 字节序

字节序，即字节在电脑中存放时的序列与输入（输出）时的序列是先到的在前还是后到的在前。字节序，即字节在电脑中存放时的序列与输入（输出）时的序列是先到的在前还是后到的在前。

BIG-ENDIAN、LITTLE-ENDIAN 跟 CPU 有关，每一种 CPU 不是 BIG-ENDIAN 就是 LITTLE-ENDIAN。IA 架构(Intel、AMD)的 CPU 中是 Little-Endian，而 PowerPC 、SPARC 和 Motorola 处理器是 Big-Endian。这其实就是所谓的主机字节序。而网络字节序是指数据在网络上传输时是大头还是小头的，在 Internet 的网络字节序是 BIG-ENDIAN。所谓的 JAVA 字节序指的是在 JAVA 虚拟机中多字节类型数据的存放顺序，JAVA 字节序也是 BIG-ENDIAN。

我的电脑使用的字节序为大端字节序。

验证如下：



000020	35	00	00	00	01	00	00	00	12	00	00	00	1F	00	00	00
000030	2D	00	00	00	20	00	00	00	52	00	00	00	18	00	00	00
000040	0A	00	00	00	16	00	00	00	34	00	00	00	38	00	00	00
000050	21	00	00	00	5B	00	00	00	2B	00	00	00	62	00	00	00
000060	42	00	00	00	24	00	00	00	21	00	00	00	36	00	00	00
000070	04	00	00	00	1C	00	00	00	16	00	00	00	2B	00	00	00
000080	55	00	00	00	48	00	00	00	1F	00	00	00	17	00	00	00
000090	2A	00	00	00	05	00	00	00	1C	00	00	00	5C	00	00	00
0000A0	47	00	00	00	2C	00	00	00	32	00	00	00	41	00	00	00
0000B0	16	00	00	00	22	00	00	00	16	00	00	00	26	00	00	00
0000C0	44	00	00	00	41	00	00	00	1D	00	00	00	44	00	00	00
0000D0	18	00	00	00	33	00	00	00	58	00	00	00	11	00	00	00
0000E0	61	00	00	00	39	00	00	00	2C	00	00	00	2E	00	00	00
0000F0	46	00	00	00	3B	00	00	00	2C	00	00	00	4D	00	00	00
000100	4C	00	00	00	5A	00	00	00	5C	00	00	00	16	00	00	00
000110	3B	00	00	00	1A	00	00	00	20	00	00	00	44	00	00	00
000120	40	00	00	00	09	00	00	00	01	00	00	00	3A	00	00	00
000130	3F	00	00	00	37	00	00	00	1C	00	00	00	13	00	00	00

图 1.3

2.实验操作

2.1 思路

按照实验 4 的修改后的程序框图，增加读取文件初始化配置变量函数 read-_configinfo 和二进制文件写入函数 file_write2，最后因为程序要求的鲁棒性下降，run 函数中许多实验 3 的部分可以删除（这也是另一种遗憾）。

2.2 读取文件初始化配置函数

无输入 输出为 CONF 结构体引用

```
01:CONF& read_configinfo(){
02:    CONF tem;
03:    FILE *fp;
04:    fp = fopen("conf.ini","r");
05:    fscanf(fp, "%s", &tem.filesavepath);
```

```
06:    fscanf(fp, "%s", &tem.filename);
07:    fscanf(fp, "%d", &tem.number);
08:    fscanf(fp, "%d", &tem.maxvalue1);
09:    fscanf(fp, "%d", &tem.minvalue1);
10:    fscanf(fp, "%d", &tem.maxvalue2);
11:    fscanf(fp, "%d", &tem.minvalue2);
12:    fscanf(fp, "%d", &tem.recordcount2); //recordcount1 is the lowest
13:    fscanf(fp, "%d", &tem.recordcount1);
14:
15:    fclose(fp);
16:    return tem;
17:
```

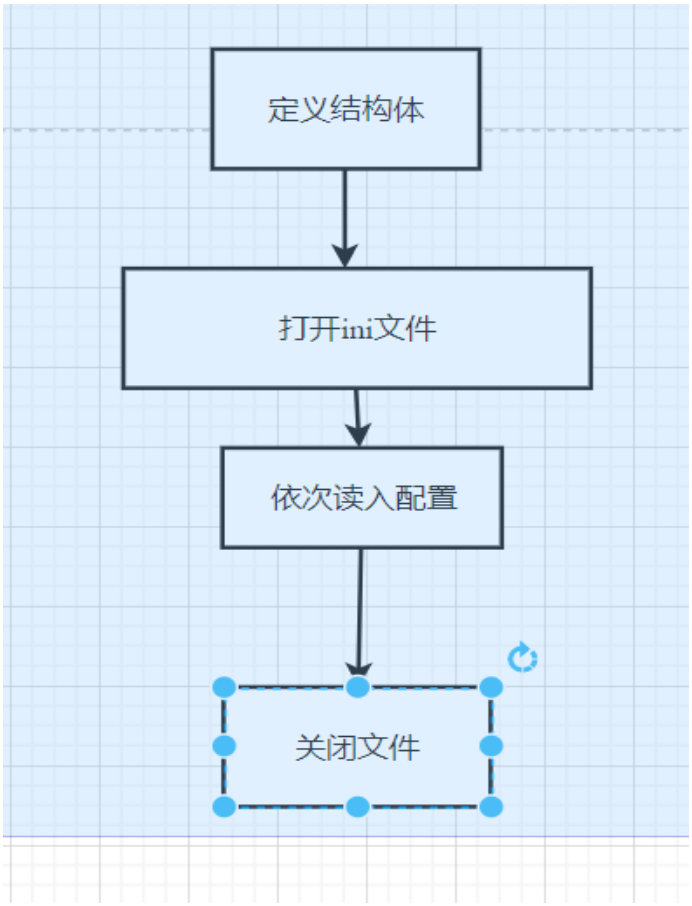


图 2.2

2.3 二进制方式写入文件函数

思路：

参照实验 3 中 file_write 函数，仅需要把文件后缀名改成.dat 即可。再有，还需要加入程序计时部分，这里参照“知识准备”部分，不表。

输入为 CONF 结构体 无输出

```
01:void file_write2(CONF conf) {
02:    FILE *pf;
03:    clock_t start, end;
04:    double duration;
05:    DataItem *item = (DataItem*) malloc(sizeof (DataItem)*conf.number);
06:    // operate the name of pave
07:
08:    if (access(conf.filesavepath,0)!=0) //等于0，文件存在 不等于0，文件不存在
09:    {
10:        mkdir(conf.filesavepath);
11:    }//判断文件存在与否
12:
13:    strcat(conf.filesavepath, conf.filename);
14:    int len = strlen(conf.filesavepath);
15:    conf.filesavepath[len-1]='t';
16:    conf.filesavepath[len-2]='a';
17:    conf.filesavepath[len-3]='d';
18:
19:    start = clock();
20:    for(int i=0; i<conf.number; ++i){
21:        item[i].item1 = random_int(conf.minvalue1, conf.maxvalue1);
22:        item[i].item2 = random_int(conf.minvalue1, conf.maxvalue1);
23:        while(item[i].item1 == item[i].item2)
24:            item[i].item2 = random_int(conf.minvalue1, conf.maxvalue1);
25:        item[i].item3 = random_int(conf.minvalue2, conf.maxvalue2);
26:    }
27:    end = clock();
28:    duration = (double )(end - start)/CLK_TCK;
29:    printf("duration of generate random number is %f\n", duration);
30:
31:    pf = fopen(conf.filesavepath, "wb");
32:    start = clock();
33:    fwrite(&conf.number, sizeof(int), 1, pf);
34:    fwrite(item, sizeof (DataItem), conf.number, pf);
35:    fclose(pf);
36:    free(item);
37:
```

```
38:    end = clock();
39:    duration = (double )(end- start)/CLK_TCK;
40:    printf("%f", duration);
41:}
```

2.4 Run 函数的改写

1） 以实验 3 的 run 函数为母版进行改写，switch-case 中需要加入 case 4

2） 鉴于实验 4 对于鲁棒性要求失速下降，本人大刀阔斧把很多保障输入合法性的代码砍了。

3） 再参照实验要求完成即可。一下为 copy 实验要求部分

a) 命令行参数的顺序必须严格按照：“记录条数、文件名、输出文件格式”的顺序输入，用户可以选择 全部或部分输入命令行参数，输入的顺序必须按照上述顺序；

b) 当用户不输入命令行参数时，记录条数及文件名按实验 3 要求默 认处理，实验 4 程序默认同时输出文 本文件和二进制文件，文本文 件内容和二进制文件中存储的记录条数和数据记录值是一样的；

c) 当用户只输入 1 个命令行参数时，程序认为该参数为数据记录条 数，该参数必须为数值或字幕‘r’， 否则视为参数错误。其他两个 参数按默认情况处理（文件名参数的默认处理逻辑遵循实验 3 要求， 输出文件格式参数的默认处理与 b）相同）

程序框图如下：

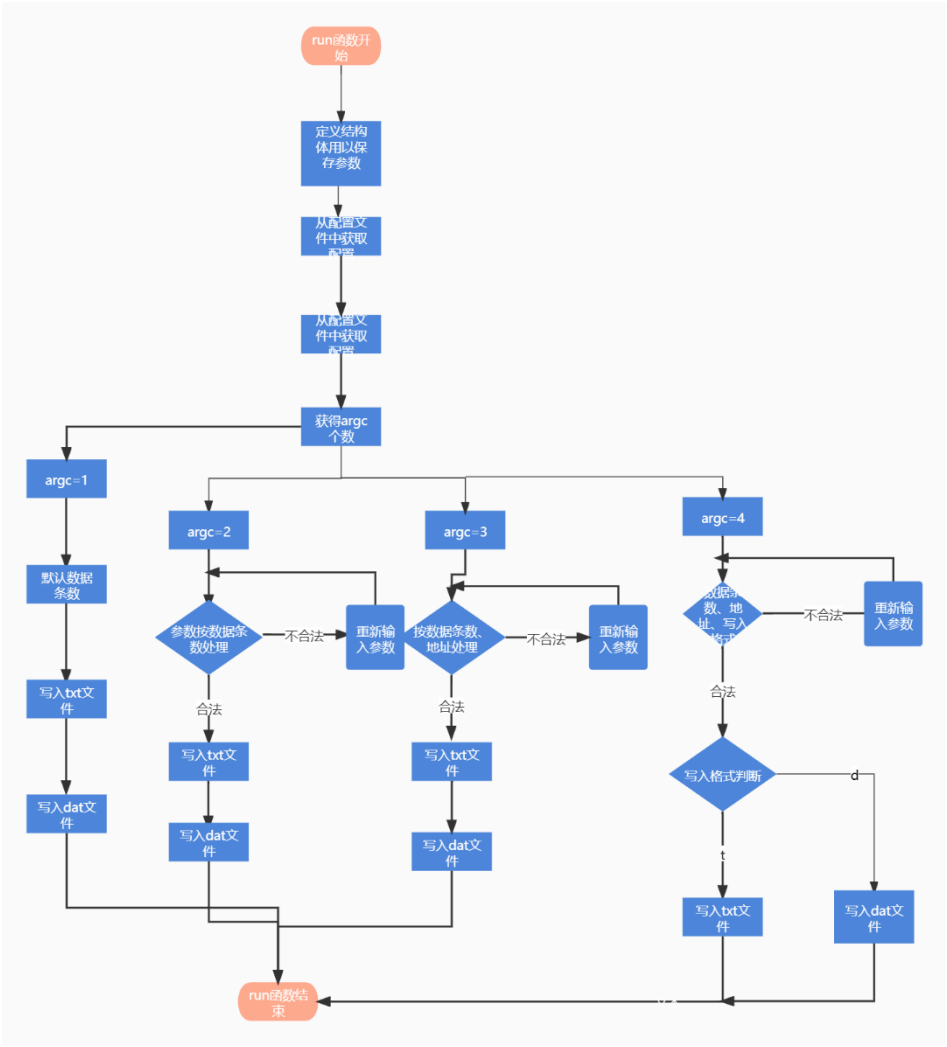


图 2.4

2.5 二进制与文本储存方式的耗时比较

这里是利用 clock 函数得到的二进制与文本储存方式程序占用 CPU 的时间和系统直接给出的文件大小对比表格。鉴于本人电脑性能很好，生成的数据条数跨度很大。

如图：

数据条数	大小/kb		写入耗时/ms	
	文本	二进制	文本	二进制
100000	1172	1338	952	4
500000	5860	6690	5874	24
1000000	11719	13380	8837	42

表 2.5

结论：当生成数据条数相同时，txt 文件耗时远多于 dat 文件，但文件大小反之。文本文件费时省空间，二进制文件省时废空间。

查找资料可知：C 的文本方读写与二进制读写的差别仅仅体现在回车换行符的处理 上．文本方式写时，每遇到一个换行符，它将其换成回车换行，然后再写入文件；当文本读取时，它每遇到一 个''\r\n''将其反变化为''\n''，然后送到读缓冲区．正因为文本方式有 ''\n''到''\r\n''之间的转换，其存在转换耗时．二进制读写时，其不存在任何转换，直接将写缓冲区中数据写入文件。

3.心得体会

3.1 自身收获

这次实验充分感受到了工欲善其事必先利其器的道理。一开始使用 word 来绘制程序框图，又慢又不好看，在网上随便找的在线软件也是功能太差。画的快吐血的时候，被小组同学推荐的软件拯救了。短短几分钟就绘制好了，精致又便捷。

本次实验中完成鲁棒性的代码部分的删除，一想到这是在实验三中反复拉锯的成果就十分遗憾。但考虑发到每次实验有不同的训练目的，也就释怀了。

附上，运行结果图：

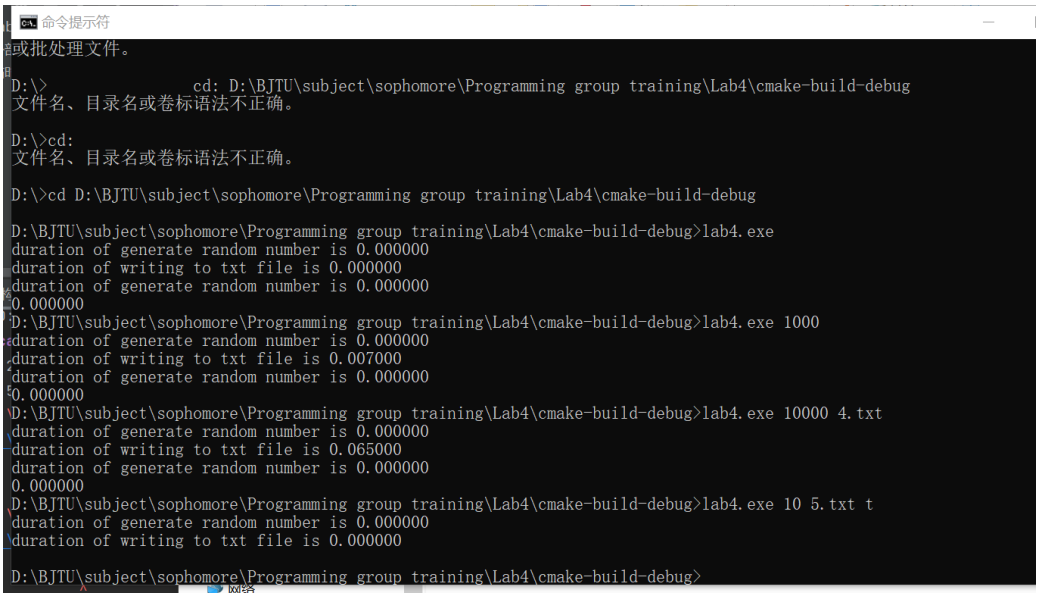


图 3.1

3.2 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。