

S&T2024

Computer Programming

(Part 2 – Advanced C Programming Language)

Chapter 4

Lecturer

A/Prof Tay Seng Chuan

E-mail: pvotaysc@nus.edu.sg

Office of the Provost
National University of Singapore

1

Ground Rules

- Switch off your handphone and pager
- No talking while lecture is going on
- No gossiping while the lecture is going on
- Raise your hand if you have question to ask
- Be on time for lecture
- Be on time to come back from the recess break to continue the lecture
- Bring your printed lecture notes to lecture or use a laptop for the e-copy. Do **not** use handphone to read the pdf file.
- Do not open any app, except this pdf file and an editing tool (to take notes).

2

Chapter 4

Introduction to Advanced File Operations

Other options for binary file (not complete):

- **"r+"** : Open a file for reading but allow writing. That is, the file can be both read from and written to. However, errors will be handled as if the file were opened for reading.
- **"w+"** : The same as **"r+"** except that errors will be appropriate to writing.
- If a file is to be opened for binary input or output then a 'b' ('b' means binary) should be added to the above strings e.g. **"wb"**, **"r+b"** or **"rb+"**.

3

Sequential Access versus Random Access

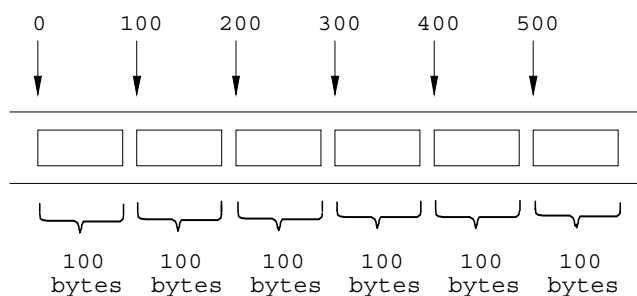
- `fscanf` reads the data in a file in sequence. If the file has values 2 3 4 5, `fscanf` will have to access to 2 3 4 (either reading it or skipping it) before it can access to 5.
- In random access, the data need not be read in sequence.

4

Random Access Files

- Records in a file created with the formatted output function **fprintf** are not necessarily of the same length.
- Individual records of a randomly accessed file are normally fixed in length and may be accessed directly (and thus quickly) without searching through other records in sequence.
- Randomly accessed files are appropriate for airline reservation systems, banking systems, point-of-sale systems, and other kinds of transaction processing systems that require rapid access to **specific** data.
- Because every record in a randomly accessed file normally has the same length, the exact location of a record relative to the beginning of the file can be calculated as a function of the record key. We will soon see how this facilitates immediate access to specific records, even in large files.

5



- After the storages have been created, data can be inserted in a randomly accessed file without destroying other data in the file.
- Data stored previously can also be updated or deleted without rewriting the entire file. The file size remains unchanged after the deletion.

6

Essential Instructions

`size_t` is an unsigned integer.

`size_t fread (void *Data, size_t ObjSize, size_t NumObjs, FILE *indata)`

Reads from `indata` to the array pointed to by `Data` up to `NumObjs` objects each of size `ObjSize`. It returns the number of objects (not bytes) actually read, which may be fewer than `NumObjs` if the end of file is met.

`size_t fwrite (const void *Data, size_t ObjSize, size_t NumObjs, FILE *outdata)`

Writes to `outdata` from the array pointed to by `Data` up to `NumObjs` objects each of size `ObjSize`. It returns the number of objects (not bytes) actually written, which may be fewer than `NumObjs` if a write error occurs.

File processing programs rarely write a single field to a file. Normally, they write one struct at a time.

Example:

We are going to create a credit processing system capable of storing up to 100 fixed-length records. Each record should consist of an account number that will be used as the record key, a last name, a first name, and a balance. The resulting program should be able to update an account, insert a new account record, delete an account, and list all the account records in a formatted text file for printing. Randomly accessed file is used.

7

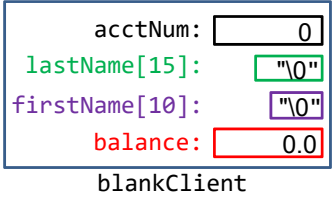
```
// Creating Blank Records, AC 4-6
#include<stdio.h>

struct clientData
{
    int acctNum;
    char lastName[15];
    char firstName[10];
    float balance;
};

main()
{
    int i;
    struct clientData blankClient = { 0, "", "", 0.0 };
    FILE *outdata;

    printf ("\nSize of struct: %d\n", sizeof (blankClient)); // 4+28+4 = 36
    if ((outdata=fopen("d:\credit.dat","wb")) == NULL)
        printf("File could not be opened.\n");
    else
    {
        for (i=1;i<=100;i++) /* write 100 blank records to the file */
            fwrite(&blankClient,sizeof(struct clientData),1,outdata);

        fclose(outdata);
    }
    return 0;
}
```



```
C:\Windows\system32\cmd.exe
Size of struct: 36
Press any key to continue . . .
```

8

36x100

```
Size of struct: 36
Press any key to continue . . .
```

credit.dat

9

```
for (i=1;i<=100;i++)
    fwrite(&blankClient, sizeof(struct clientData), 1, outdata);

fwrite(&blankClient, sizeof(struct clientData), 100, outdata);
```

10

Are these two sets of instructions the same?

Diagram illustrating the iterative steps of the bubble sort algorithm on the array $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$. The array is shown in a sequence of boxes, with the largest element in the unsorted portion being swapped with the last element in each step.

Step 1: $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ → $[0, 1, 2, 3, 4, 5, 6, 8, 7, 9]$

Step 2: $[0, 1, 2, 3, 4, 5, 6, 8, 7, 9]$ → $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

Step 3: $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ → $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

Step 4: $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ → $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

Step 5: $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ → $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

Step 6: $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ → $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

Step 7: $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ → $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

Step 8: $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ → $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

Step 9: $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ → $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

0, "", "", 0.0	& .. @ # df\$	WW@1 #(*&?
...	0, "", "", 0.0	...
		** & trw - SM2

11

The two file sizes are the same.

```
fff
D:\>type creditn.dat
```

The two file contents
are different!!

12

```

#include<stdio.h>
#define MAXSIZE 100

struct tool
{
    int rec_Num;
    char name[16];
    int quantity;
    float cost;
};

main()
{
    struct tool inventory={0, "", 0, 0.0};
    FILE *outdata;
    int size=sizeof(struct tool);

    outdata=fopen("d:\\hardware.dat", "wb");
    if (outdata==NULL)
    {
        printf ("ERROR");
        exit(1);
    }

    fwrite(&inventory, size, 100, outdata);
    fclose(outdata);

    return 0;
}

```

The file size will be the same, but the contents can be different!!

Cannot be used in this answer!!

13

// Creating Blank Records, AC 4-6

```

#include<stdio.h>

struct clientData
{
    int acctNum;
    char lastName[15];
    char firstName[10];
    float balance;
};

main()
{
    int i;
    struct clientData blankClient = { 0, "", "", 0.0 };
    FILE *outdata;

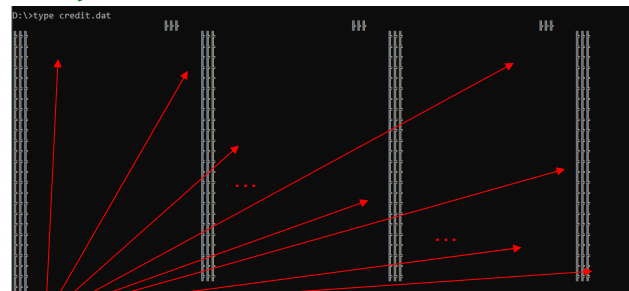
    printf ("\nSize of struct: %d\n", sizeof (blankClient)); // 4+28+4 = 36

    if ((outdata=fopen("d:\\credit.dat", "wb")) == NULL)
        printf("File could not be opened.\n");
    else
    {
        for (i=1; i<=100; i++) /* write 100 blank records to the file */
            fwrite(&blankClient, sizeof(struct clientData), 1, outdata);

        fclose(outdata);
    }

    return 0;
}

```



C:\Windows\system32\cmd.exe

```

D:\>type credit.dat
Size of struct: 36
Press any key to continue . . .

D:\>dir credit.dat
Volume in drive D is New Volume
Volume Serial Number is 8E71-0209

Directory of D:\

13/06/2020  10:10 pm          3,600 credit.dat
               1 File(s)          3,600 bytes
               0 Dir(s)  194,832,465,920 bytes free

```

14

fwrite (&blankClient, sizeof(struct clientData), 1, outdata);

- The program initializes all 100 records of the file "credit.dat" with empty structs using function **fwrite**.
- `struct clientData blankClient = { 0, "", "", 0.0 };`
Each empty struct contains 0 for the account number, **NULL** (represented by empty quotation marks) for the last name, **NULL** for the first name, and 0.0 for the balance.
- The file is initialized in this manner to create storage on the disk in which the file will be stored, and to make it possible to determine if a record contains data.

Writing Data Randomly to a Random Access File

The next program writes data to the file "credit.dat".

It uses the combination of **fseek** and **fwrite** to store data at specific locations in the file. Function **fseek** sets the file position pointer to a specific position in the file, then **fwrite** writes the data.

15

File Positioning

int fseek (FILE *Stream, long int Offset, int Origin)

Sets the file position within Stream so that subsequent reading or writing will occur from there. the position is in the form of an **Offset** relative to an **Origin**. The possible values of **Origin** are **SEEK_SET**, **SEEK_CUR** and **SEEK_END**. If an error occurs then a non-zero value is returned; a return value of zero indicates success.

<u>Origin</u>	<u>Measure offset from</u>
SEEK_SET	Beginning of file
SEEK_CUR	Current position
SEEK_END	End of file

Offset has positive value (to move the file cursor forward), or negative value (to move the file cursor backward).

16


```
int fseek(FILE *stream, long int offset, int whence);
```

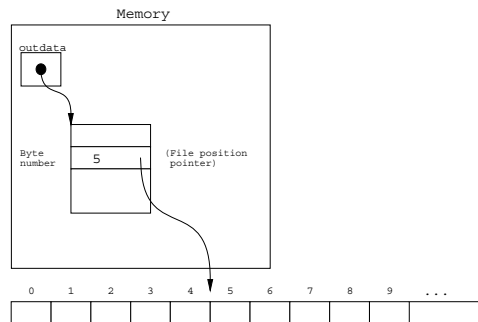
offset : is the number of bytes from location whence

whence : SEEK_SET

(origin) SEEK_CUR

SEEK_END

These three symbolic constants are defined in the stdio.h header file.



The file position pointer indicating an offset of 5 bytes from the beginning of the file.

17

```
/* Data Entry to a random access file, AC4-7.C */
#include<stdio.h>
```

```
struct clientData
{
    int acctNum;
    char lastName[15];
    char firstName[10];
    float balance;
};
```

```
main()
{
    struct clientData client;
    FILE *outdata;
```



client:

acctNum:	<input type="text"/>
lastName[15]:	<input type="text"/>
firstName[10]:	<input type="text"/>
balance:	<input type="text"/>

```
if ((outdata=fopen("d:\\credit.dat","r+b")) == NULL) // seek and write
    printf("File could not be opened.\n"); // wb does not work
else
{
    printf("Enter account number (1 to 100, 0 to end)\n? ");
    scanf("%d",&client.acctNum);
```

18

```

while (client.acctNum != 0)
{
    printf("Enter lastname, firstname, balance \n? ");
    scanf("%s%s%f", client.lastName, client.firstName, &client.balance);
    fseek(outdata, (client.acctNum-1)*sizeof(struct clientData), SEEK_SET);
    fwrite(&client, sizeof(struct clientData), 1, outdata);
    printf("\nEnter account number (1 to 100, 0 to end) \n? ");
    scanf("%d", &client.acctNum);
}

fclose(outdata);
return 0;
}

```

C:\Windows\system32\cmd.exe
Enter account number (1 to 100, 0 to end)
? 10

acctNum: 10
lastName[15]: Wang
firstName[10]: Qiqi
balance: 100

client

19

```

D:\>type credit.dat

```

credit.dat

Before
Data Entry

```

fseek(outdata, (client.acctNum-1)*sizeof(struct clientData), SEEK_SET);
fwrite(&client, sizeof(struct clientData), 1, outdata);

```

```

D:\>type credit.dat

```

After
Data Entry

10 Wang Qiqi 100
20 He Qiqi 200
30 Li Qiqi 300
80 Zhang Qiqi 800
90 Yang Qiqi 900

20

After Data Entry

First Update

```
C:\Windows\system32\cmd.exe
1. Read a record
2. Change a record
Enter your choice (0 to end) > 1
```

10 Wang Qiqi 100	10 Wang Qiqi 100
20 He Qiqi 200	20 He Qiqi 200
30 Li Qiqi 300	30 Li Qiqi 300
80 Zhang Qiqi 800	35 Mao Wenwen 350
90 Yang Qiqi 900	80 Zhang Gonggong 867.23
	90 Yang Qiqi 900

credit.dat

21

```
/* Update the Credit file, AC4-8.C */
#include<stdio.h>
struct clientData
{
    int acctNum;
    char lastName[15];
    char firstName[10];
    float balance;
};

main()
{
    struct clientData client;
    FILE *indata;
    int option,accountNum;

    if ((indata=fopen("d:\\credit.dat","r+b")) == NULL)
        printf("File could not be opened.\n");
    else
    {
        printf("\n1. Read a record\n");
        printf("2. Change a record\n");
        printf("Enter your choice (0 to end) > ");
        scanf("%d%c",&option);
    }
}
```

22

```

while (option==1 || option==2)
{
    printf("Enter account number > ");
    scanf("%d%c",&accountNum);

    fseek(indata,(accountNum-1)*sizeof(struct clientData),SEEK_SET);
    if (option==1)
    {
        fread(&client,sizeof(struct clientData),1,indata);
        if (client.acctNum==0)
            printf("Account %d has no information.\n",accountNum);
        else
            printf("%-6d %-10s %-11s %10.2f\n", client.acctNum,
                client.lastName, client.firstName,client.balance);
    }
    else
    {
        printf("Enter lastname, first name, balance\n");
        scanf("%s%s%f",client.lastName,client.firstName,&client.balance);
        client.acctNum=accountNum;
        fwrite(&client,sizeof(struct clientData),1,indata);
    }

    printf("\n1. Read a record\n");
    printf("2. Change a record\n");
    printf("Enter your choice (0 to end) > ");
    scanf("%d%c",&option);
}
fclose(indata);
return 0;
}

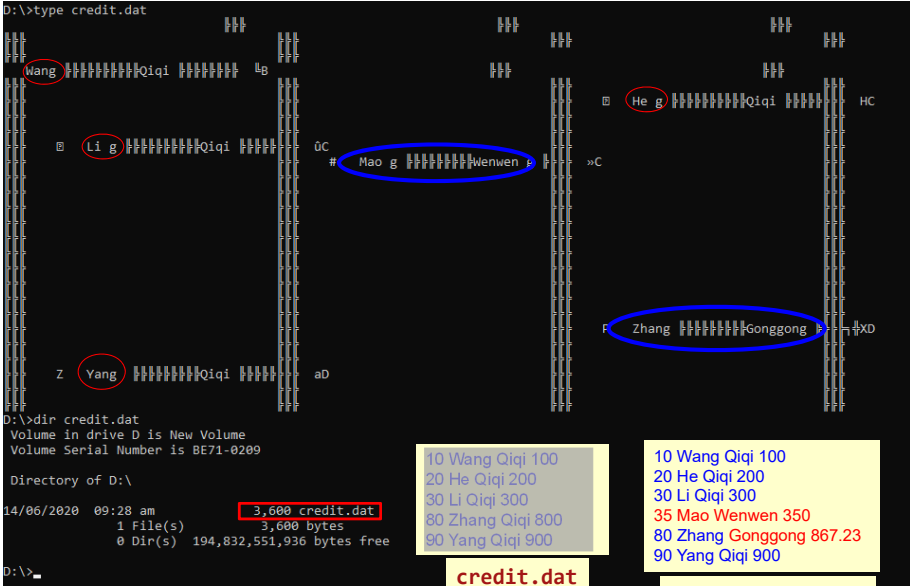
```



client

23

After First Update



Directory of D:\

File Name	Size (bytes)
10 Wang Qiqi 100	100
20 He Qiqi 200	200
30 Li Qiqi 300	300
80 Zhang Qiqi 800	800
90 Yang Qiqi 900	900

File Name	Size (bytes)
10 Wang Qiqi 100	100
20 He Qiqi 200	200
30 Li Qiqi 300	300
35 Mao Wenwen 350	350
80 Zhang Gonggong 867.23	867.23
90 Yang Qiqi 900	900

Updated Contents

24

```
C:\Windows\system32\cmd.exe
1. Read a record
2. Change a record
Enter your choice (0 to end) > 1
```

Second Update

```
10 Wang Qiqi 100
20 He Qiqi 200
30 Li Qiqi 300
35 Mao Wenwen 350
80 Zhang Gonggong 867.23
90 Yang Qiqi 900
```

credit.dat

```
10 Wang Qiqi 100
20 He Qiqi 200
30 Li Qiqi 300
35 Mao Wenwen 350
50 Ren Baobao 92.46
80 Zhang Gonggong 867.23
89 Xi Baobao 89.77
90 Yang Qiqi 900
```

Updated Contents

25

After Second Update

```
D:\>type credit.dat
Wang Qiqi 100
He Qiqi 200
Li Qiqi 300
Mao Wenwen 350
Ren Baobao 92.46
Zhang Gonggong 867.23
Xi Baobao 89.77
Yang Qiqi 900

D:\>dir credit.dat
Volume in drive D is New Volume
Volume Serial Number is 8E71-0209

Directory of D:\

14/06/2020  09:49 am          3,600 credit.dat
               1 File(s)          3,600 bytes
               0 Dir(s) 194,832,547,840 bytes free

D:\>
```

```
10 Wang Qiqi 100
20 He Qiqi 200
30 Li Qiqi 300
35 Mao Wenwen 350
50 Ren Baobao 92.46
80 Zhang Gonggong 867.23
89 Xi Baobao 89.77
90 Yang Qiqi 900
```

Updated Contents

26

Additional Useful Functions

rewind()

The `rewind()` function sets the file position indicator to the beginning of the given file stream.

```
// test_rewind.c
#include <stdio.h>

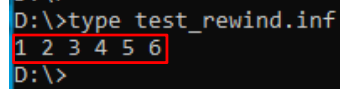
main()
{
    FILE *indata;
    int a1, a2, a3, a4;

    indata=fopen("d:\\test_rewind.inf", "rt");
    if (indata==NULL)
    {
        printf ("ERROR");
        exit(0);
    }

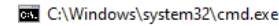
    fscanf (indata, "%d%d", &a1,&a2);
    printf ("\nRead From Top: %d %d", a1,a2);
    rewind(indata);

    fscanf (indata, "%d%d%d%d", &a1,&a2,&a3,&a4);
    printf ("\n After rewind: %d %d %d %d \n", a1,a2,a3,a4);

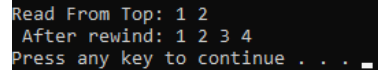
    return 0;
}
```



```
D:\>type test_rewind.inf
1 2 3 4 5 6
D:\>
```



```
C:\Windows\system32\cmd.exe
```



```
Read From Top: 1 2
After rewind: 1 2 3 4
Press any key to continue . . .
```

27

fflush()

The C library function `int fflush(FILE *stream)` flushes the output buffer of a stream.

```
int fflush(FILE *stream)
```

Parameters

- **stream** – This is the pointer to a FILE object that specifies a buffered stream.

Return Value

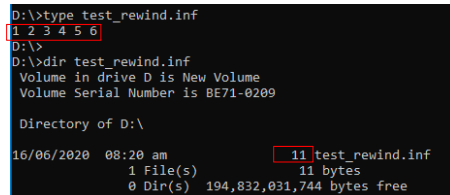
This function returns a zero value on success. If an error occurs, EOF (-1) is returned.

```
// test_fflush.c
#include <stdio.h>

int main ()
{
    FILE *outdata;
    int a1=77, a2=888, a3=9999;

    if ((outdata=fopen("d:\\test_rewind.inf", "a"))
        == NULL)
    {
        printf("File could not be opened.\n");
    }
    else
    {
        fprintf(outdata, "%d%d%d", a1,a2,a3);
        fflush(outdata);
    }

    fclose (outdata);
    return 0;
}
```



```
D:\>type test_rewind.inf
1 2 3 4 5 6
D:\>
D:\>dir test_rewind.inf
Volume in drive D is New Volume
Volume Serial Number is BE71-0209

Directory of D:\

16/06/2020  08:20 am          11 test_rewind.inf
               1 File(s)              11 bytes
               0 Dir(s) 194,832,031,744 bytes free
```

28