FILES

Dr Frank Guan

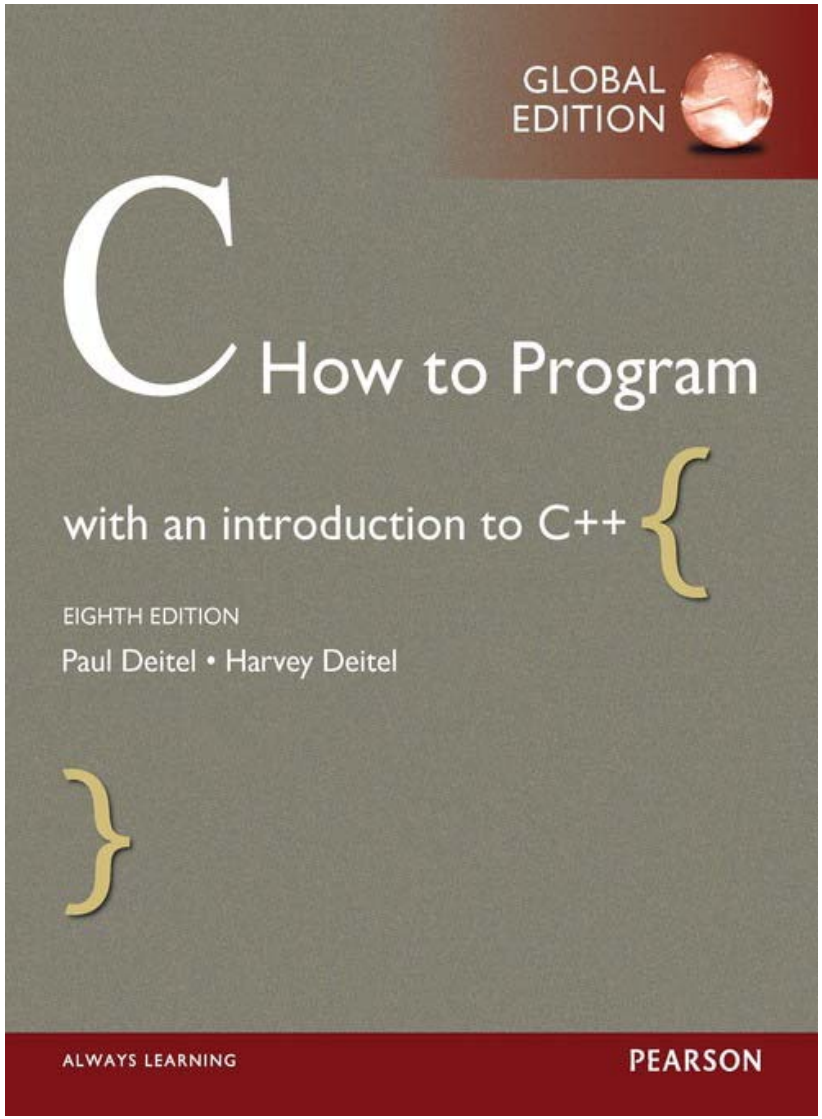
ICT1002 – Programming Fundamentals

Week 12

Agenda

1. Files
2. Putting It All Together

RECOMMENDED READING



Paul Deitel and Harvey Deitel, *C: How to Program*, 8th Edition, Prentice Hall, 2016

- Chapter 11: *C File Processing*

FILES | SiT SINGAPORE INSTITUTE OF TECHNOLOGY

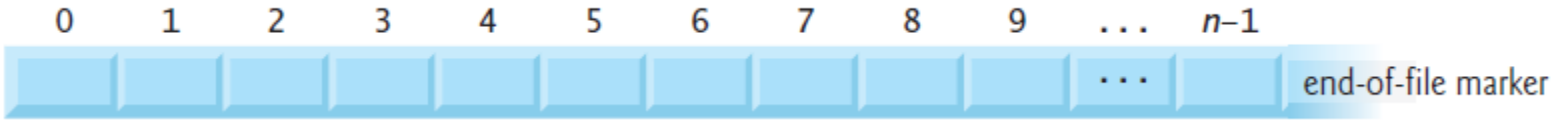
INTRODUCTION

Storage of data in variables and arrays is temporary—such data is lost when a program terminates.

Files are used for permanent retention of data. Computers store files on secondary storage devices such as hard disks and flash memory.

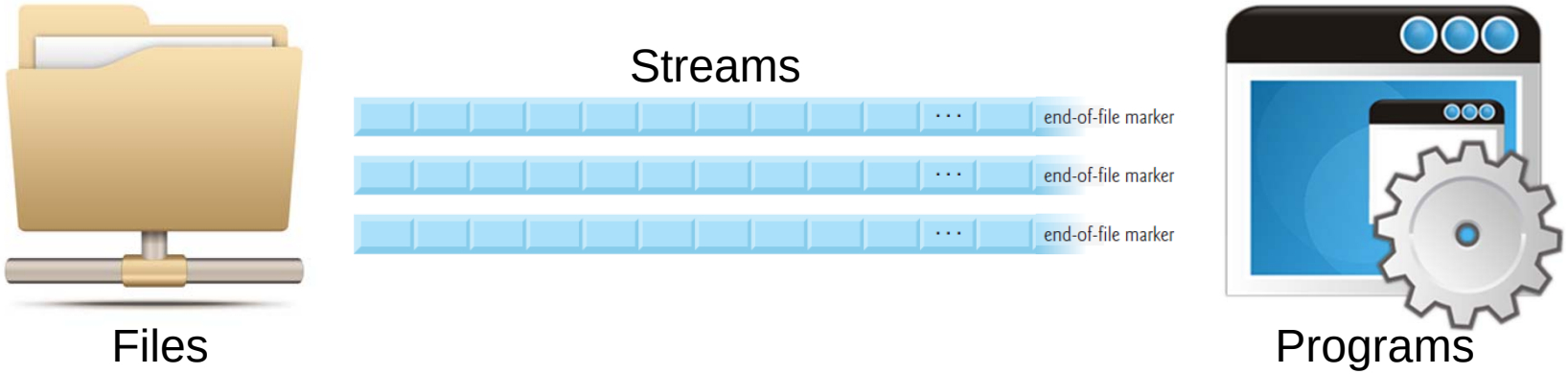
FILES & STREAMS

C views each file as a sequential stream of bytes. Each file ends at the **end-of-file marker**.



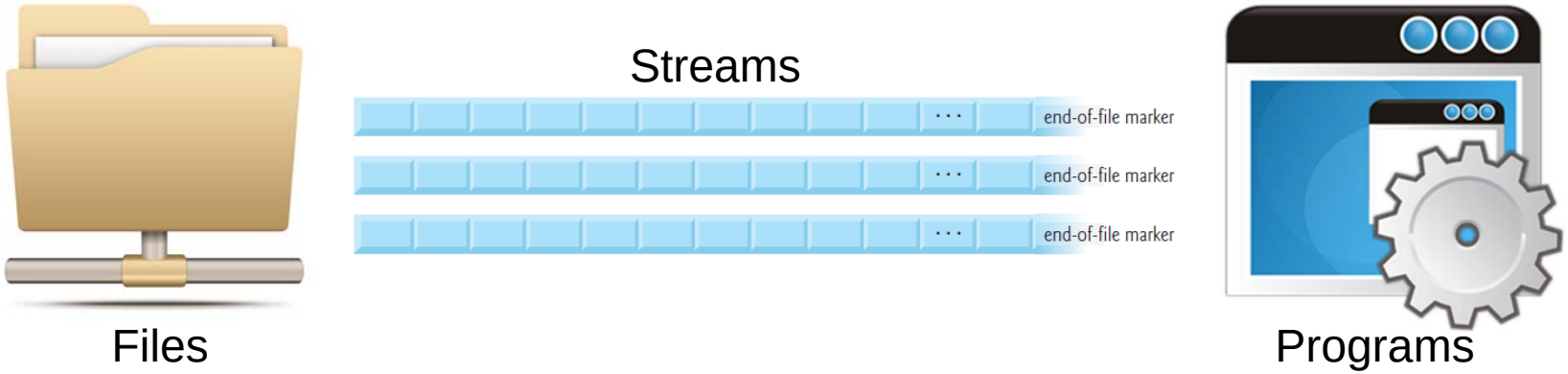
Hello.c

STREAMS



When a file is opened, a `stream` is associated with the file

STREAMS



Three streams are opened when a program starts its execution:

- Standard input: `stdin`
- Standard output: `stdout`
- Standard error: `stderr`

STRUCTURING A FILE

C imposes no structure on a file.

The programmer must provide a structure to meet the requirements of the program.

Example:

```
<account> <name> <balance>
```

```
100 Jones 24.98
200 Doe 345.67
300 White 0.00
400 Stone -42.16
500 Rich 224.62
```

STREAM MODES

Streams may be opened in one of two modes:

- `text mode` is used for reading and writing text files
  - files are divided into lines by new line characters
  - e.g. plain text, source code, HTML, XML, etc.

- `binary mode` is used for reading and writing all other files
  - files are made up of raw bytes
  - e.g. images, videos, databases, etc.

CREATING A TEXT FILE

Creating a file requires four steps:

Step 1. Declare a pointer to a FILE structure

Step 2a. Open the file using fopen()

2b. Write to the file using fprintf() and friends

Step 3. Close the file using fclose()

All of these types and functions are defined in <stdio.h>.

CREATING A TEXT FILE - EXAMPLE

```
#include <stdio.h>

int main() {

        /* declare a pointer to a FILE structure */
        FILE *f;

        /* open the file with fopen() */
        f = fopen("data.txt", "w");
        if (f == NULL) {
                printf("Could not open data.txt.\n");
                return 1;
        }

        /* write to the file with fprintf() */
        fprintf(f, "Hello! This is a new file.\n");

        /* close the file with fclose() */
        fclose(f);

        return 0;

}
```

fopen() returns NULL if the file cannot be opened.

fprintf() is like printf() but writes to the file given as its first argument.

OPENING A FILE

```
FILE *fopen(const char *filename, const char *mode);
```

File opening modes for text files:

mode	Description
"r"	Open a file for reading. The file must exist.
"w"	Create an empty file for writing. If a file with the same name already exists its content is erased and the file is considered as a new empty file.
"a"	Append to a file. Writing operations append data at the end of the file. The file is created if it does not exist.
"r+"	Open a file for update both reading and writing. The file must exist.
"w+"	Create an empty file for both reading and writing.
"a+"	Open a file for reading and appending.

WRITING TO A SEQUENTIAL FILE
EXAMPLE

```
/* print instructions to the user */
printf("Enter the account, name and balance for each customer.\n");
printf("Enter account 0 to end input.\n");

/* repeatedly prompt for input */
printf("? ");
scanf("%d", &account);
while (account != 0) {
    scanf("%19s%lf", name, &balance);

    /* write this customer to the file */
    fprintf(f, "\n%d %s %.2f", account, name, balance);

    /* prompt for the next customer */
    printf("? ");
    scanf("%d", &account);
}
```

Output (file)

```
1122 Cristal_Ngo 0.0
1123 Leo_Tan 1234.56
1125 Alfred_Teo 44443.56
```

READING FROM A SEQUENTIAL FILE
EXAMPLE

```
/* read until the end of the file */
printf("%10s\t%20s\t%10s\n", "Account", "Name", "Balance");
while (!feof(f)) {

    /* read one record */
    fscanf(f, "%d%19s%lf", &account, name, &balance);

    /* display it to the screen */
    printf("%10d\t%20s\t%10.2lf\n", account, name, balance);

}
```

`feof()` returns a true value if the file pointer is at the end of the file.

`fscanf()` is like `scanf()` but reads from the file given as its first argument.

Output (display)

```
Account                 Name    Balance
   1122          Cristal_Ngo       0.00
   1123              Leo_Tan    1234.56
   1125           Alfred_Teo   44443.56
```

READING FROM A SEQUENTIAL FILE

- Resetting the file position pointer

```
rewind(filePtr);
```

Reset the program's file position pointer to be repositioned to the beginning of the file pointed to by `filePtr`.

The file position pointer is not really a pointer. Rather it's an integer value that specifies the byte location in the file at which the next read or write is to occur. This is sometimes referred to as the file offset.

EXERCISE

Write a program that:

- Allows a credit manager to obtain lists of customers with zero balances, a credit balance (-) and a debit balance (+).
- Displays a menu allowing the credit manager to enter one of three options to obtain credit information.
- Reads all the information from a file.

```
Enter request
1 - List accounts with zero balances
2 - List accounts with credit balances
3 - List accounts with debit balances
4 - Exit
? 1
Accounts with zero balances:
300 White 0.00
? 2
Accounts with credit balances:
400 Stone -42.16
? 3
Accounts with debit balances:
100 Jones 24.98
200 Doe 345.67
500 Rich 224.62
? 4
Goodbye.
```

EXERCISE - HINTS

- See xSiTe for ICT1002-W12-LEC_credit_skeleton.c that implements the menu.

- We need to implement a function that will perform each action on the menu.
  - Do we have any similar functions already?
  - How could we modify them to perform each action?
  - Do we need three different functions, or just one with some parameters?

WRITING DATA TO A SEQUENTIAL FILE

Data in this type of sequential file cannot be modified without the risk of destroying other data.

Solution? 

RANDOM ACCESS FILES

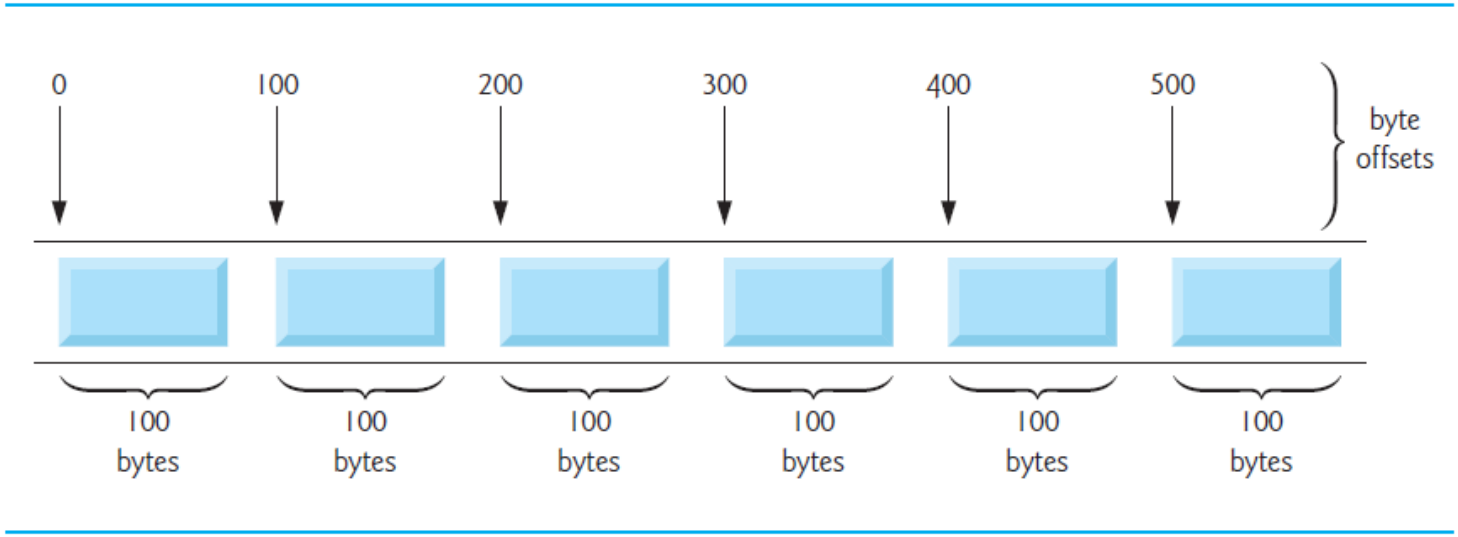
A **random access file** can be read or written in any order.

Writing to one part of the file does not change another part of the file.

But, how do we know where any particular record is located in the file?

RANDOM ACCESS FILES

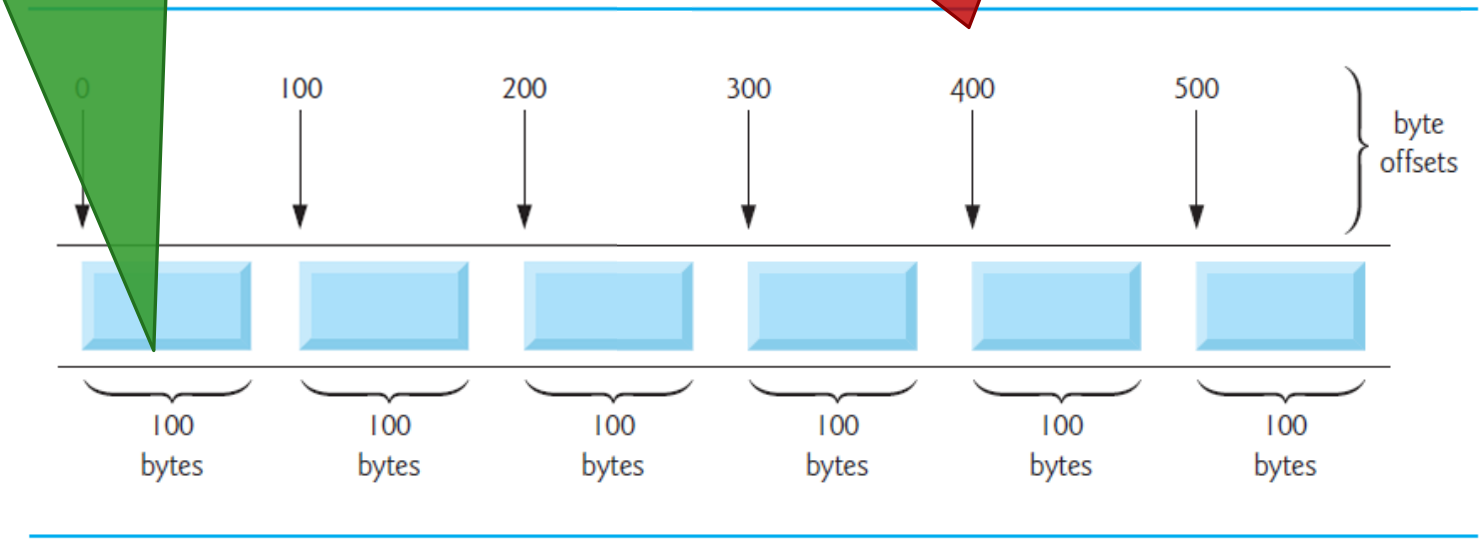
The simplest kind of random access file consists of a series of records of fixed length:



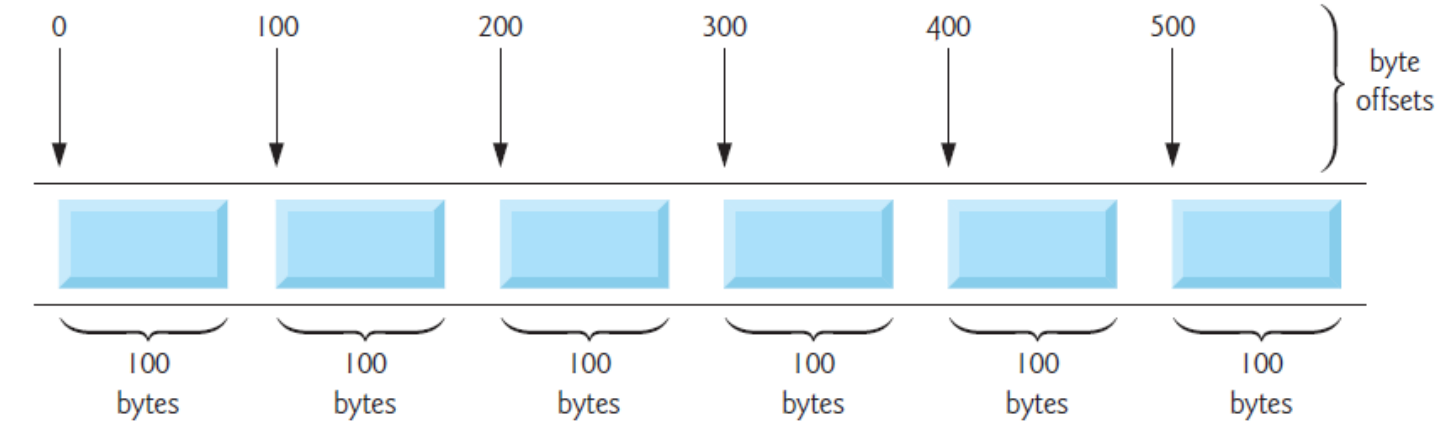
RANDOM ACCESS FILES

Every record in a random access file has the same length

The exact location of a record can be calculated from the record key and length.



RANDOM ACCESS FILES



Fixed-length records **enable data to be inserted/updated/deleted** in a random access file **without destroying** other data

OPENING A RANDOM ACCESS FILE

File opening modes:

rb	Opens an existing binary file for reading.
wb	Opens a binary file for writing. If it does not exist, a new file is created. If it exists, it will be overwritten.
ab	Opens a binary file for appending. If it does not exist, a new file is created.
rb+	Opens a binary file for reading and writing both.
wb+	Opens a binary file for reading and writing both.
ab+	Opens a binary file for reading and writing both.

ACCESSING A RANDOM ACCESS FILE

(i) use `fseek` to move to a specific position in the file

Calculate the position of the desired record.

```
int pos = sizeof(Record) * record_no;
fseek(filePtr, pos, SEEK_SET);
```

Position the file position pointer `pos` bytes relative to …

`SEEK_SET` – the start of the file

`SEEK_CUR` – the current position

`SEEK_END` – the end of the file

ACCESSING A RANDOM ACCESS FILE

(ii) use `fwrite` and `fread` to write and read blocks

Read precisely one group of `sizeof(Record)` bytes from a file represented by `filePtr` and store the data in the variable `result`.

```
Record result;
fread(&result, sizeof(Record), 1, filePtr);
```

The bytes are read from the current position of the file pointer.

`fwrite()` is similar, but writes data to the file.

RANDOM ACCESS FILE
EXAMPLE

Write a program to read and write clients' credit data using a random access file.

```
typedef struct client_struct
{
    int acc_num;
    char last_name[15];
    char first_name[10];
    double balance;
} Client;
```

The account number is used to manage the location of the record in the file.

Use the structure above for representing a client's data.

RANDOM ACCESS FILE
EXAMPLE – CREDIT PROGRAM

```
FILE *f = fopen(filename, "wb+");
if (f == NULL) {
        printf("Could not open %s.\n", filename);
        return 1;
}
```

Open a binary file for reading and writing and make sure that the file can be opened.

RANDOM ACCESS FILE
EXAMPLE – CREDIT PROGRAM

```
/* read account data from the user */
printf("Enter account number (1-100, 0 to end)\n? ");
scanf("%d", &client.acc_num);
while (client.acc_num != 0) {

    /* read the data for this record */
    printf("Enter last name, first name and balance\n? ");
    scanf("%14s%9s%lf", client.last_name, client.first_name, &client.balance);

    /* go to this record's position in the file */
    fseek(f, (client.acc_num - 1) * sizeof(Client), SEEK_SET);

    /* write the client data structure */
    fwrite(&client, sizeof(Client), 1, f);

    /* ask for another record */
    printf("Enter account number (1-100, 0 to end)\n? ");
    scanf("%d", &client.acc_num);
}
```

Ask the user for client data and write to the file.

RANDOM ACCESS FILE
EXAMPLE – CREDIT PROGRAM

```
/* read account data from the user */
printf("Enter account number (1-100, 0 to end)\n? ");
scanf("%d", &client.acc_num);
while (client.acc_num != 0) {

    /* read the data for this record */
    printf("Enter last name, first name and balance\n? ");
    scanf("%14s%9s%lf", client.last_name, client.first_name, &client.balance);

    /* go to this record's position in the file */
    fseek(f, (client.acc_num - 1) * sizeof(Client), SEEK_SET);

    /* write the client data structure */
    fwrite(&client, sizeof(Client), 1, f);

    /* ask for another record */
    printf("Enter account number (1-100, 0 to end)\n? ");
    scanf("%d", &client.acc_num);
}
```

Ask the user for an account number **(1-100)**.

If the account number is not 0, enter **last_name**, **first_name** and **balance**.

RANDOM ACCESS FILE
EXAMPLE – CREDIT PROGRAM

```
/* read account data from the user */
printf("Enter account number (1-100, 0 to end)\n? ");
scanf("%d", &client.acc_num);
while (client.acc_num != 0) {

    /* read the data for this record */
    printf("Enter last name, first name and balance\n? ");
    scanf("%14s%9s%lf", client.last_name, client.first_name, &client.balance);

    /* go to this record's position in the file */
    fseek(f, (client.acc_num - 1) * sizeof(Client), SEEK_SET);
```

(client.acc_num - 1)*sizeof(Client) is the offset or displacement within the file. The symbolic constant **SEEK_SET** indicates that the file position pointer is to be positioned relative to the beginning of the file.

```
void read_client_data(const char *filename) {

    FILE *f;
    Client client;

    /* open the data file */
    f = fopen(filename, "rb");
    if (f == NULL) {
        printf("Could not open %s.\n", filename);
        return;
    }

    /* print title */
    printf("%-6s%-16s%-11s%10s\n",
        "Acct", "Last Name", "First Name", "Balance");

    /* read one record at a time until we reach EOF */
    fread(&client, sizeof(Client), 1, f);
    while (!feof(f)) {
        if (client.acc_num != 0)
            printf("%-6d%-16s%-11s%10.2lf\n",
                client.acc_num, client.last_name,
                client.first_name, client.balance
            );
        fread(&client, sizeof(Client), 1, f);
    }

    fclose(f);
}
```

RANDOM ACCESS FILE

EXAMPLE – CREDIT PROGRAM, READ DATA

RANDOM ACCESS FILE
EXAMPLE – CREDIT PROGRAM

Improvement – Exercises:

- Allow the user to print a single record only by entering the account number.
- Allow the user to update a certain record.

```
Enter account to update ( 1 - 100 ): 37
37 Barker Doug 0.00
Enter charge ( + ) or payment ( - ): +87.99
37 Barker Doug 87.99
```

PUTTING IT ALL TOGETHER

SiT SINGAPORE INSTITUTE OF TECHNOLOGY

PUTTING IT ALL TOGETHER

In the past two weeks, we have seen:

- how to read and write records from a file
- how to store and retrieve data in linked lists

Let's put it into a program that:

- reads records from a file and stores them in a linked list
- allows the user to search, insert and delete records
- writes the modified records back to disk

PUTTING IT ALL TOGETHER

The program menu will look like:

```
Acct  Last Name       First Name    Balance
44    Sheppard        Nicholas        44.44
45    Ngo             Cristal         11.11
33    Wang            Zhengkui       314.33
29    Avnit           Karin          -24.54
33    Wong            Steven          33.33
=============================================
Enter your choice:
 1. Insert a client into the list.
 2. Delete a client from the list.
 3. Search for a client.
 4. Print the list of clients.
 5. Read data from disk.
 6. Write data to disk.
 0 Exit.
?
```

See ICT1002-W12-LEC_accounts_skeleton.c on xSiTe.

PUTTING IT ALL TOGETHER

First, modify the node structure so that it contains a client record instead of an integer:

```
typedef struct node_struct {
        Client data;
        struct node_struct *next;
} Node;
```

PUTTING IT ALL TOGETHER

Second, modify the list operations to use `data.acc_num` to identify nodes:

```
temp = head;
while (temp != NULL && temp->data.acc_num != value)
    temp = temp->next;
```

PUTTING IT ALL TOGETHER

Third, modify the file reading and writing functions to use a linked list instead of the console:

```
fread(&client, sizeof(Client), 1, f);
while (!feof(f)) {
    if (client.acc_num != 0) {
        /* allocate a new node */
        /* insert it into the list */
    }
    fread(&client, sizeof(Client), 1, f);
}
```

END-OF-WEEK CHECKLIST

- [ ] Files
- [ ] Streams
- [ ] File control blocks (FCBs)
- [ ] Opening a file
- [ ] Writing to a file
- [ ] Reading from a file
- [ ] Sequential access files
- [ ] Random access files
- [ ] File pointers