S&T2024

Computer Programming

(Part 2 – Advanced C Programming Language)

Chapter 5

Lecturer
A/Prof Tay Seng Chuan

E-mail: pvotaysc@nus.edu.sg

Office of the Provost
National University of Singapore

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).

Chapter 5

Advanced File Operations

1. Introduction

- **FILE** is defined in **<stdio.h>** and used for input and output.
- Text file is readable, but not binary file.

  Three files are opened when a C program is executing.

  stdin : default input file

  stdout : default output file

  stderr : a file to which error messages are written.
- On microcomputers **stdin** often represents the keyboard, and **stdout** and **stderr** the screen, but this is by no means universal.

2. Opening and Closing Files

- Reference to files is via variables which are always pointers to a structure of type **FILE**.
- A typical declaration might be

```
FILE *indata;
```

  This file can then be opened by

```
indata = fopen ("monkey.inf","r");
```

- Open the file ***monkey.inf*** for reading.
- Generate a structure to hold whatever control information is needed, and leave indata pointing to it.
- File descriptor may be complicated.

  E.g., "a:\\monkey.inf" where a drive name is appended. The second parameter is also a string, "r" in this case meaning 'read'.

How about c:\\cp2315\\prac5\\monkey.inf ?

The options are:

"r" Open an existing file for reading.

"w" Open a file for writing. If the file does not exist then it will be created and if it does exist then its contents will be deleted before writing.

"a" Open an existing file for writing so that new data will be appended to the end of the current file contents.

"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.

"a+" Open a file for appending but allow reading at the same time.

- 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+"**.

- **fopen** will return a pointer value of **NULL** if the file cannot be opened and so this can be used to test for successful completion of the operation.
- The file opened above can be closed with:
  fclose (indata);
- File closing should not be omitted because it is dangerous to do so, particularly for files which are opened for writing.
- Closing a file releases the space used by the **FILE** structure and ensures that, when a file is being written, the file buffer is emptied.
- When a program requests that a character be written to a disk the request is not carried out immediately. Instead the character is placed in an array, called a buffer, which accumulates characters until there are enough to be worth writing. 'Enough', in this sense, usually means the size of a disc sector.
- If the buffer is not emptied then the file may not include the latest data written.
- ***fclose*** ensures that the buffer will be written, i.e., the contents of the buffer are transferred to a storage device.

The following example shows a program to create a sequential file.

Program

```
/* AC4-1.C */
/* NOTE : A file named "clients.dat" will be generated in the current directory */
#include <stdio.h>
main()
{
  int account;
  char name[30];
  float balance;
  FILE *outdata; /*outdata=clients.dat file pointer */
  if ((outdata=fopen("clients.dat","w")) == NULL )
    printf ("File could not be opened\n");
  else
   {
     printf("Enter the account, name, and balance.\n");
     printf("Enter <EOF> character (CTRL+Z in DOS) to end input.\n");
     printf("? ");
     scanf("%d%s%f",&account,name,&balance);
     while (!feof(stdin))
     {
       fprintf(outdata," %d %s %.2f\n", account,name,balance);
       printf("? ");
       scanf("%d%s%f",&account,name,&balance);
     }
     fclose(outdata);   /* close the data file */
   }
   return 0;
}
```


Screen Output

Enter the account, name, and balance.
Enter <EOF> character (CTRL+Z in DOS) to end input.
? *100 Jones 24.98*
? *200 Doe 345.67*
? *300 White 0.00*
? *400 Stone -42.16*
? *500 Rich 224.62*
? *^Z*

Output File (clients.dat)

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

The following example illustrates the reading and printing of a sequential file.

Program

```
/* AC4-2.C */
/* NOTE : A file named "clients.dat" must be put in current directory */
#include<stdio.h>
main()
{
  int account;
  char name[30];
  float balance;
  FILE *indata;   /* indata = clients.dat file pointer */
   if ((indata=fopen("clients.dat","r")) == NULL )
    printf ("File could not be opened\n");
  else
  {
    printf ("%-10s%-13s%s\n","Account", "Name","Balance");
                                    /* - means left justified */
     printf ("============================\n");
    fscanf(indata,"%d%s%f",&account, name, &balance);
    while (!feof(indata))
    {
      printf("%-10d%-13s%7.2f\n",account, name, balance);
      fscanf(indata,"%d%s%f",&account, name, &balance);
    }
    fclose(indata);      /* close the data file */
  }
  return 0;
}
```


Screen Output

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

Reopening a File

freopen ("monkey2.inf", "r", indata);

- Closes the existing file associated with indata and opens the file monkey2.inf.
- Acts in the same way as **fopen** in that it also delivers a pointer to indata and delivers **NULL** if the file cannot be opened.

  The following example illustrates the use of freopen. We want to encode two files. Put the output in the same text file.

Input File-1 (monkey1.inf)

Two sections of commando will attack

Delta-2A at 0200 hour.

Big Bird will give support fire.

Input File-2 (monkey2.inf)

If alpha company is confronted,

call for hawk.

Output File (donkey.ouf)

First encoded message:

Ydl hvxgrlmh lu xlnnzmwl droo zggzxp

Ivogz-2K zg 0200 slfi.

Krt Kriw droo trev hfkklig uriv.

Second endcoded message:

Su zoksz xlnkzmb rh xlmuilmgvw,

xzoo uli szdp.

Program

```
/* AC4-3.C */
#include<stdio.h>
 main()
{
  FILE *indata,*outdata;
  char this1;
  int i;
   indata=fopen("a:\\ac4\\monkey1.inf","r");
  outdata=fopen("a:\\ac4\\donkey.ouf","w");
  fprintf(outdata,"First encoded message :\n");
   for (i=0;i<2;i++)
  {
     while (fscanf(indata,"%c",&this1)==1)
     {
       if (this1>='A' && this1<='Z')
          fprintf(outdata,"%c", ((this1-2*'A'+'K')%26)+'A');
       else
          if (this1>='a' && this1<='z') fprintf(outdata,"%c",'z'-(this1-'a'));
        else
          fprintf(outdata,"%c",this1);
     }
      if (i==0)
     {
       fprintf(outdata,"\nSecond endcoded message:\n");
       if (freopen("a:\\ac4\\monkey2.inf","r",indata) == NULL) i=1;
     }
  }
   fclose(indata);
  fclose(outdata);
   return 0;
}
```

Empty a output buffer even before the buffer is full

- ***fflush*** forces the emptying of output buffers.
- Flushing output buffer is essential if a file is open in one of the 'update' modes ("w+", "r+" or "a+") and a change is made from writing to reading.
- Unless fflush is called before finishing a sequence of write operations, the information remaining in the output buffer will not be written at the correct point in the file.

This program illustrates the use of **fflush** function.

Program

```
/* AC4-4.C */
#include<stdio.h>
main()
{
 int i;
 float result,mark,sum=0;
 FILE *outdata;
  outdata=fopen("test.dat","w+");
 for (i=0;i<5;i++)
 {
   printf("Enter test result>");
   scanf("%f%*c",&result);
   fprintf(outdata,"%f\n",result);
 }
 fflush(outdata);    /* transfer data from buffer to disk */
rewind(outdata);
 for (i=0;i<5;i++)
 {
   fscanf(outdata,"%f",&mark);
   sum += mark;
 }
  fclose(outdata);
 printf("Average = %f\n",sum/5);
 return 0;
}
```

Screen Output

Enter test result>*100.000000*

Enter test result>*45.000000*

Enter test result>*83.000000*

Enter test result>*97.000000*

Enter test result>*60.000000*

Average = 77.000000

3. File Positioning

void rewind (FILE *Stream)

Positions **Stream** so that the next read or write operation will be at the beginning of the file. If this is not possible then there will be no effect.

long int ftell (FILE *Stream)

Returns the current positions within **Stream**, or -1L if an error occurs.

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.

Origin	Measure offset from
SEEK_SET	Beginning of file
SEEK_CUR	Current position
SEEK_END	End of file

Program

```
 /* AC4-5.C */
 #include<stdio.h>
#include<stdlib.h>
 #define MAX 20
 main()
{
  FILE *indata;
  char words[MAX];
   if ((indata=fopen("words.ouf","a+")) == NULL)
  {
    fprintf(stderr,"Can't open \"words\" file.\n");
    exit(1);
  }
   puts("Enter words to add to the file; type the <Enter> key \n");
  puts("at the beginning of a line to terminate.");
  while (gets(words)!=NULL && words[0]!='\0')
     fprintf(indata,"%s\n",words);
   puts("File contents:");
   rewind(indata);    /* go back to the beginning of the file */
   while (fscanf(indata,"%s",words)==1)
     puts(words);
   fclose(indata);
   return 0;
}
```


Screen Output

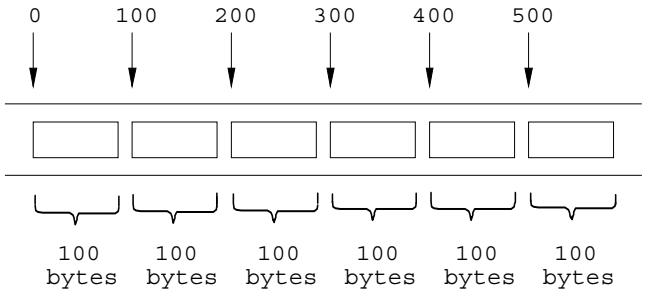
```
 C>ac4-5
Enter words to add to the file; type the <Enter> key
at the beginning of a line to terminate.
See the canoes
 File contents:
See
the
canoes
 C>ac4-5
Enter words to add to the file; type the <Enter> key
at the beginning of a line to terminate.
on the
Sea
File contents:
See
the
canoes
on
the
Sea
```

Output File (words.ouf)

```
See the canoes
on the
sea
```

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.
- 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.



- 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.

Essential Instructions

size_t is an unsigned integer in Turbo C.

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.

Creating a Random Access File.

Program

```
/* AC4-6.C */
/* NOTE : A data file named "credit.dat" will be created for program
        "AC4-7.C" and "AC4-8.C"
*/
#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;
  if ((outdata=fopen("credit.dat","w")) == 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;
}
```


Output File (credit.dat, binary format and not readable)

- The program initializes all 100 records of the file **"credit.dat"** with empty structs using function **fwrite**.

- 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 space on the disk in which the file will be stored, and to make it possible to determine if a record contains data.

fwrite

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

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.

Program

```
 /* AC4-7.C */
 /* Writing to a random access file */
#include<stdio.h>
 struct clientData
{
  int acctNum;
  char lastName[15];
  char firstName[10];
  float balance;
};
 main()
{
 struct clientData client;
 FILE *outdata;
if ((outdata=fopen("credit.dat","r+")) == NULL)
   printf("File could not be opened.\n");
 else
  {
    printf("Enter account number (1 to 100, 0 to end input)\n? ");
    scanf("%d",&client.acctNum);
     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("Enter account number\n? ");
      scanf("%d",&client.acctNum);
    }
  }
 fclose(outdata);
 return 0;
}
```

fseek

fseek(outdata, (client.accNum - 1) * sizeof(struct clientData), SEEK_SET);

Screen Output

```
Enter account number (1 to 100, 0 to end input)
? 30
Enter lastname, firstname, balance
Kassim Abula 75.3
Enter account number
? 27
Enter lastname, firstname, balance
Lily Tay 42.5
Enter account number
? 56
Enter lastname, firstname, balance
Angeelo Ali 70.89
Enter account number
? 34
Enter lastname, firstname, balance
Keng Heng  0.00
Enter account number
? 10
Enter lastname, firstname, balance
Milly Ken 203.41
Enter account number
? 0
```

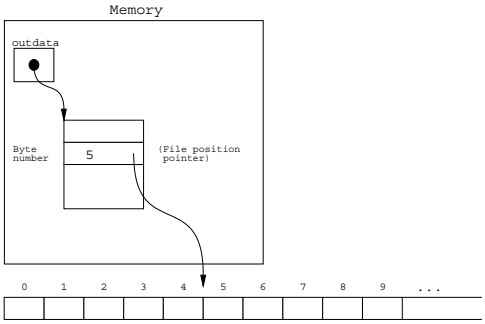

Other Use of fseek

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.

Reading Data Randomly from a Random Access File

- Use of fseek and fread, or fseek and fwrite to read/write records from/to a file randomly.
- Position the file pointer to the desired record we want to read or write, then we perform the operation on the record.

Program

```
/* 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("credit.dat","r+")) == NULL)
  printf("File could not be opened.\n");
 else
 {
  printf("1. Read a record\n");
  printf("2. Change a record\n");
  printf("Enter your choice >");
  scanf("%d%*c",&option);
   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);
   }
 }
  fclose(indata);
 return 0;
}
```

Screen Output

1. Read a record
2. Change a record
Enter your choice >*1*
Enter account number >*27*
27 Lily Tay 42.50

1. Read a record
2. Change a record
Enter your choice >*2*
Enter account number >*34*
Enter lastname, first name, balance
Keng Heng 100.00

1. Read a record
2. Change a record
Enter your choice >*1*
Enter account number >*80*
Account 80 has no information.