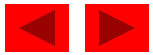
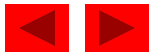
Chapter 11 – File Processing

Outline



Objectives

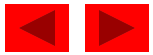
- In this chapter, you will learn:
  - To be able to create, read and write files.
  - To become familiar with sequential access file processing.



11.1 Introduction

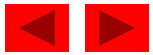
- Data files
  - Can be created, updated, and processed by C programs
  - A file represents a sequence of byte on the disk where a group of related data is stored.

- Why do we need data files?
  - Storage of data in variables and arrays is only temporary—such data is lost when a program terminates.
  - Files are used for permanent storage of large amounts of data.
  - Computers store files on secondary storage devices



11.2 The Data Hierarchy

- Data Hierarchy:
  - Bit – smallest data item
    - Value of `0` or `1`
  - Byte – 8 bits
    - Used to store a character
      - Decimal digits, letters, and special symbols
  - Field – group of characters conveying meaning
    - Example: your name
  - Record – group of related fields
    - Represented by a `struct` or a `class`
    - Example: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.



11.2 The Data Hierarchy

- Data Hierarchy (continued):
  - File – group of related records
    - Example: payroll file
  - Database – group of related files

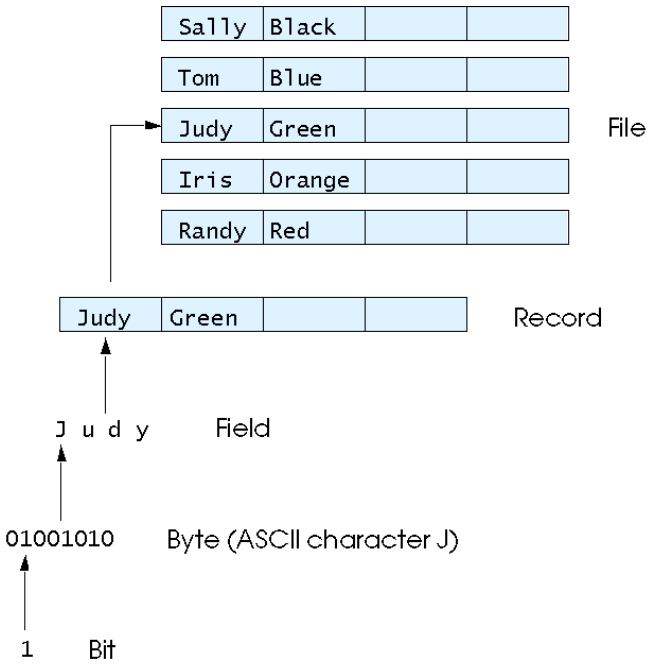
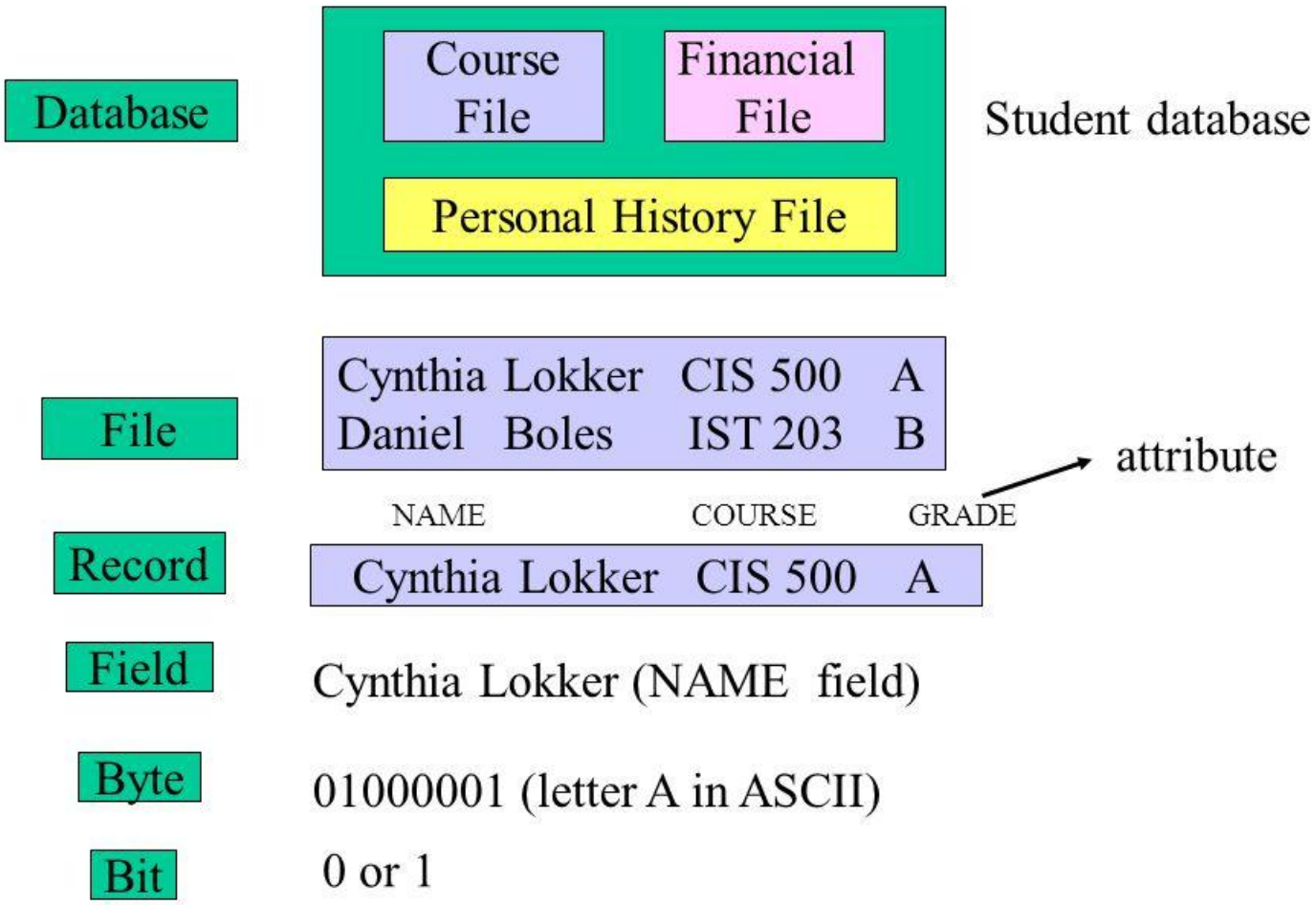


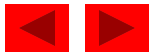
Fig. 11.1 The data hierarchy.

The Data Hierarchy



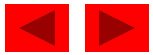
11.3 Files and Streams

- C views each file as a sequence of bytes
  - File ends with the *end-of-file marker*
    - Or, file ends at a specified byte
- Stream created when a file is opened
  - When a file is opened, a stream is associated with it
  - Provide communication channel between files and programs
  - Opening a file returns a pointer to a `FILE` structure
    - Example file pointer:
    - `stdin` - standard input (keyboard)



11.3 Files and Streams

- `FILE` structure
  - **C** Provides smart way to manipulate data using streams. In stdio.h header **file FILE structure** is defined.
  - **FILE structure** provides us the necessary information about a **FILE** or stream which performs input and output operations.

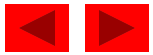


11.4 Steps in Processing a file

- Create the stream via a pointer variable using the **FILE** structure:

```
FILE *p;
```

- Open the file, associating the stream name with the file name.
- Read or write the data.
- Close the file.



11.4 Steps in Processing a file

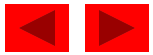
- **Open the file: fopen_s()**

  fopen_s(file pointer address, “file name”, “mode”);

  – Function `fopen_s` returns a `FILE` pointer to file specified

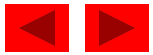
  – Takes 3 arguments – file pointer, file to open and file open mode

  – If open fails, `NULL` returned



11.4 Steps in Processing a file

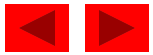
File access mode	Explanation	Action if file already exists	Action if file does not exist
"r"	Open a file for reading	read from start	failure to open
"w"	Create a file for writing	destroy contents	create new
"a"	Append to a file	write to end	create new
"r+"	Open a file for read/write	read from start	error
"w+"	Create a file for read/write	destroy contents	create new
"a+"	Open a file for read/write	write to end	create new



11.4 Steps in Processing a file

- **If you attempt to read from a non-existent file, your program will crash!!**
  - The fopen function was designed to cope with this eventuality. It checks if the file can be opened appropriately. If the file **cannot be opened**, it returns a **NULL** pointer. Thus by checking the file pointer returned by fopen_s, you can determine if the file was opened correctly.

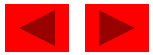
```
if (!fp)
{
    perror("File opening failed"); or printf("File opening failed");
    return EXIT_FAILURE;  //OR  return 1;
}
```

11.4 Steps in Processing a file

- **Read/Write functions in standard library**
  - `fgetc / getc`
    - Reads one character from a file
    - Takes a `FILE` pointer as an argument
    - `fgetc()` equivalent to `getchar()`

```
FILE *fp;
char ch;
…
ch=fgetc(fp);
…
```

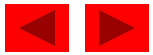



11.4 Steps in Processing a file

- `fputc`

  - Writes one character to a file
  - Takes a `FILE` pointer and a character to write as an argument
  - `fputc( 'a', filePointer )` equivalent to `putchar( 'a' )`

```
FILE *fp;
char ch;
…
fputc(ch,fp);
…
```

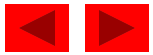
11.4 Steps in Processing a file

- `fgets`
  - Reads a line from a file

```
FILE *fp;
char b[20];
…
fgets(b,sizeof b,fp);
…
```

- `fputs`
  - Writes a line to a file

```
FILE *fp;
char b[20];
…
fputs(b,fp);
…
```

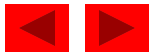
11.4 Steps in Processing a file

- `fprintf`
  - Like printf
  - Takes first argument as file pointer

```
FILE *fp;
float salary;
…
fprintf(fp,”%f”,salary);
…
```

- `fscanf`
  - Like scanf
  - Takes first argument as file pointer

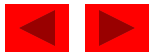
```
FILE *fp;
float salary;
…
fscanf(fp,”%f”,&salary);
```

11.4 Steps in Processing a file

- `feof( FILE pointer )`
  - tests the end-of-file indicator for the given stream.
  - Returns true if end-of-file indicator (no more data to process) is set for the specified file

```
FILE *fp;
…
while(!feof(fp))
{
   …
}
…
```

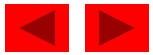



11.4 Steps in Processing a file

- **Close the File: fclose()**

  If function *fclose* is not called explicitly, the operating system normally will close the file when program execution terminates.

```
FILE *fp;
…
…
fclose(fp);
```

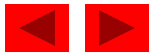



11.4 Steps in Processing a file

- **Reset a file position pointer**
  - The statement

    rewind(fp);

  - causes a program's file position pointer—which indicates the number of the next byte in the file to be read or written—to be repositioned to the beginning of the file (i.e., byte 0) pointed to by fp.
  - The file position pointer is not really a pointer.
  - Rather it's an integer value that specifies the byte in the file at which the next read or write is to occur.
  - This is sometimes referred to as the file offset.
  - The file position pointer is a member of the FILE structure associated with each file.



11.5 Exercises

READING FROM A FILE

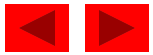
```
#include "stdafx.h"
#include "stdlib.h"
int main()
{
      FILE *fp;
      char b[50], ch;

      fopen_s(&fp, "..\\test.txt", "r");
      if (!fp) {
            printf("File opening failed");
            return EXIT_FAILURE;
      }
      fgets(b, sizeof b, fp);
      printf("%s\n", b); //or puts(b);
      fclose(fp);
      return 0;
}
```

you have to create text file before running the program.
File -> new -> file -> text file
and save it under the project folder.

OUTPUT

```
This is a test file
```

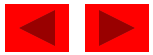
Reading one character at a time

```
#include "stdafx.h"
#include "stdlib.h"
int main()
{
	FILE *fp;
	char b[50], ch;

	fopen_s(&fp, "..\\test1.txt", "r");
	if (!fp) {
		printf("File opening failed");
		return EXIT_FAILURE;
	}
	ch = fgetc(fp);
	while (ch != EOF)//OR (!feof(fp))
	{
		printf("%c", ch);
		ch = fgetc(fp);
	}
	fclose(fp);
	return 0;
}
```

OUTPUT

```
This is a test file
Line1
Line2
Line3
```

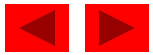
Reading one row/record at a time

```
#include "stdafx.h"
#include "stdlib.h"
int main()
{
      FILE *fp;
      char b[50], ch;

      fopen_s(&fp, "..\\test1.txt", "r");
      if (!fp) {
            printf("File opening failed");
            return EXIT_FAILURE;
      }
      while (!feof(fp))
      {
            fgets(b, sizeof b, fp);
            printf("%s", b);
      }
      fclose(fp);
      return 0;
}
```

OUTPUT

```
This is a test file
Line1
Line2
Line3
```

Counting number of characters and lines

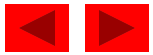
```
#include "stdafx.h"
#include "stdlib.h"
int main()
{
      FILE *fp;
      char b[50], ch;
      int nlines = 0, nc = 0;
      fopen_s(&fp, "..\\test1.txt", "r");
      if (!fp) {
            printf("File opening failed");
            return EXIT_FAILURE;
      }
      ch = fgetc(fp);
      while (!feof(fp))
      {
            if (ch == '\n')
                  nlines++;
            nc++;
            ch = fgetc(fp);
      }
      printf("There are %d characters\n", nc);
      printf("There are %d lines \n", nlines);
      fclose(fp);
      return 0;
}
```

Test1.txt

```
This is a test file
Line1
Line2
Line3
```

OUTPUT

```
There are 38 characters
There are 4 Lines
```

WRITING TO A FILE

```
#include "stdafx.h"
#include "stdlib.h"
int main()
{
      FILE *fp;
      fopen_s(&fp, "..\\test2.txt", "w");
      if (!fp) {
            printf("File opening failed");
            return EXIT_FAILURE;
      }
      fprintf(fp, "This is testing for fprintf...\n");
      fputs("This is testing for fputs...\n", fp);

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

Test2.txt

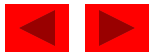
```
This is testing for fprintf...
This is testing for fputs...
```


ADDING TO A FILE

```
#include "stdafx.h"
#include "stdlib.h"
int main()
{
      FILE *fp;
      fopen_s(&fp, "..\\test2.txt", "a");
      if (!fp) {
            printf("File opening failed");
            return EXIT_FAILURE;
      }
      fprintf(fp, "This is testing for ADDING...\n");
      fputs("This is testing for ADDING a line...\n", fp);
      fclose(fp);
      return 0;
}
```

Test2.txt

```
This is testing for fprintf...
This is testing for fputs...
This is testing for ADDING...
This is testing for ADDING a line...
```

ENTERING DATA FROM KEYBOARD AND WRITING TO A FILE

End-of-file key combination is CTRL + Z

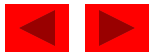
```
#include "stdafx.h"
#include "stdlib.h"
int main()
{
      FILE *fptr;
      int n;
      fopen_s(&fptr, "..\\program.txt", "w");
      if (!fptr) {
            printf("File opening failed");
            return EXIT_FAILURE;
      }
      printf("Enter n: ");
      scanf_s("%d", &n);
      while (!feof(stdin))
      {
            fprintf(fptr, "%4d", n);
            printf("Enter n: ");
            scanf_s("%d", &n);
      }
      fclose(fptr);
      return 0;
}
```

SCREEN

```
Enter n: 2
Enter n: 3
Enter n: 4
Enter n: ^Z
^Z
^Z
```

program.txt

```
  2   3   4
```

WRITE AND READ (with rewind() function)

```c
#include "stdafx.h"
#include "stdlib.h"
int main()
{
	FILE *fptr;
	char ch;
	fopen_s(&fptr, "..\\program1.txt", "w+");
	if (!fptr) {
		printf("File opening failed");
		return EXIT_FAILURE;
	}
	fprintf(fptr, "This is testing for w+");
	rewind(fptr);
	ch = fgetc(fptr); // OR ch = getc(fptr);
	while (ch != EOF)  //OR while(!feof(fptr))
	{
		printf("%c", ch); // OR putchar(ch);
		ch = fgetc(fptr);
	}
	fclose(fptr);
	return 0;
}
```

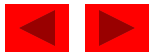
```c
#include "stdafx.h"
#include "stdlib.h"
int main()
{
	FILE *fptr;
	char st[30]= "This is testing for w + ";
	fopen_s(&fptr, "..\\program1.txt", "w+");
	if (!fptr) {
		printf("File opening failed");
		return EXIT_FAILURE;
	}
	fprintf(fptr, st);
	rewind(fptr);
	fgets(st,sizeof st, fptr);
	puts(st);
	fclose(fptr);
	return 0;
}
```

program1.txt

```
This is testing for w+
```

Screen

```
This is testing for w+
```

WRITE AND READ (without rewind() function)

```
#include "stdafx.h"
#include "stdlib.h"
int main()
{
     FILE *fptr;
     char st[30] = "This is testing for w and r ";
     fopen_s(&fptr, "..\\program3.txt", "w");
     if (!fptr) {
          printf("File opening failed");
          return EXIT_FAILURE;
     }
     fprintf(fptr, st);
     fclose(fptr);
     fopen_s(&fptr, "..\\program3.txt", "r");
     fgets(st, sizeof st, fptr);
     puts(st);
     fclose(fptr);
     return 0;
}
```