Introduction to Computers & Programming

Comp 1330/ First Semester 2024/2025

Instructor: Saif Harbia

Faculty of Engineering and Technology
Department of Computer Science

Chapter 11

Text and Binary File Processing

Chapter Objectives:

1. Learn about **streams** in **C** and their relationship to files and standard input and output devices.

2. Review how **scanf** , **fscanf** and **printf** , **fprintf** are used to read and write data.

3. Review **escape sequences** and their use in format strings.

4. Review **file pointer** variables and learn how to use functions that process them to make a backup copy of a text file.

5. Learn about **binary files** and understand the differences between binary and text files.

11.1 INPUT/OUTPUT FILES: REVIEW AND FURTHER STUDY

- A **text file** is a named collection of characters saved in secondary storage (e.g., on a disk).
- A **text file** has no fixed size.
- To mark the end of a text file, the computer places a special **end-of-file** character, which we will denote <**eof**>, after the last character in the file.
- As you create a text file using an editor program, pressing the **<return>** or **<enter>** key causes the newline character (represented by **C** as **'\n'**) to be placed in the file.

11.1 INPUT/OUTPUT FILES: REVIEW AND FURTHER STUDY

```
This is a text file!<newline>
It has two lines.<newline><eof>
```

- The disk file consists of a sequence of characters occupying consecutive storage locations on a track of the disk, as shown here:

```
This is a text file!<newline>It has two lines.<newline><eof>
```

- We sometimes refer to a data source or destination as **an input stream** or an **output stream**. (3)

- These general terms can be applied to **files**, to the terminal **keyboard** and **screen**, and to any other sources of **input** data or destinations of **output** data.

11.1 INPUT/OUTPUT FILES: REVIEW AND FURTHER STUDY

The Keyboard and Screen as Text Streams

- The name **stdin** represents the keyboard's **input stream.**
- Two system streams are associated with the screen:
  - The "**normal**" output stream **stdout**
  - The "**error**" output stream **stderr**. (1)

- Normally at the keyboard, we enter one line of data at a time, pressing **<return>** or **<enter>** to indicate the end of a data line. Pressing one of these keys inserts the **newline** character in system stream **stdin** .

- The **eof** character could be used to indicate the end of data (2).

- Writing characters to the streams **stdout** and **stderr** causes a display on the screen in an interactive program (i.e. **Printf**) (3)

11.1 INPUT/OUTPUT FILES: REVIEW AND FURTHER STUDY

- **Newline and EOF**

➢ The **<newline>** can be processed like any other character: It can be input using **scanf** with the **%c** specifier, it can be compared to '**\n**' for equality, and it can be output using **printf** .

➢ Input of the special **eof** character is regarded as a failed operation, (1)

➢ The **C** run-time support system is under no obligation to provide an error message if the program ignores the warning value and continues to attempt to get input from the stream in question.

- **Escape Sequences: p.626 (review)**

- **Formatting Output with printf: p.626 (review)**

11.1 INPUT/OUTPUT FILES: REVIEW AND FURTHER STUDY

- **File Pointer Variables: p. 627 (review):**

- A pointer whose value equals **NULL** is called **a null pointer** (1)
- Using **fopen** with mode "**w**" to open for output a file that already exists usually causes loss of the contents of the existing file. (2)

- **Functions That Take File Pointer Arguments: p.629 (review)**

- **Closing a File: p.630 (review):**
- Function **fclose** disposes of the structure that was created to store file access information and carries out other "cleanup" operations.

- **EXAMPLE 11.1, Figure 11.1**

11.2 BINARY FILES

- When we use text files for storage of data, a program must expend a significant amount of effort to convert the stream of characters from an input file into the binary integers, type double mantissas and exponents, and character strings that are the representation in main memory of the same data.

- The program must again expend time in converting the internal data format back into a stream of characters for storage in an output file of text.

11.2 BINARY FILES

- We can avoid the unnecessary translation by using a **binary file** rather than a text file. (1)
- **A binary file** is created by executing a program that stores directly in the file the computer's internal representation of each file component. (2)

FIGURE 11.3 Creating a Binary File of Integers

```
1. FILE *binaryp;
2. int   i;
3.
4. binaryp = fopen("nums.bin", "wb");
5.
6. for  (i = 2; i <= 500; i += 2)
7.     fwrite(&i, sizeof (int), 1, binaryp);
8.
9. fclose(binaryp);
```

11.2 BINARY FILES

- the second argument to **fopen** is either "**wb**" (**write binary**) for output files or "**rb**" (**read binary**) for input files.

- However, a different **stdio** library function is used for copying values into the file: function **fwrite** , which has four input parameters.

1. The first parameter is the address of the first memory cell whose contents are to be copied to the file. (1)
2. The second parameter of function **fwrite** is the number of bytes to copy to the file for one component. (2)

11.2 BINARY FILES

- A **C** operator **sizeof** can be applied to any data type name to find the number of bytes that the current implementation uses for storage of the data type. (1)

1.
```
printf("An integer requires %d bytes ", sizeof (int));
printf("in this implementation.\n");
```
the binary file. (2)

2. The final argument to **fwrite** is a file pointer to the file being created, a file previously opened in mode "**wb**" using function **fopen** .

11.2 BINARY FILES

- For example, if array score is an array of ten integers, the following statement writes the entire array to the output file.

```
fwrite(score, sizeof (int), 10, binaryp);
```

- Writing the value of an integer variable **i** to a binary file using **fwrite** is faster than writing **i** to a text file.

- For example, if the value of **i** is 244 , the statement from the for loop copies the internal binary representation of **i** from memory to the file accessed by **binaryp** . (1)

```
fwrite(&i, sizeof (int), 1, binaryp);
```

11.2 BINARY FILES

➢ Assuming **textp** is a pointer to a text output file, the following statement writes the value of **i** to the file using four characters (four bytes). (1)

```
fprintf(textp, "%d ", i);
```

- Obviously, it takes more time to do the conversion and copy each character than it does to copy the internal binary representation to disk.

- Also, twice as much disk space is required to store four characters as to store the internal binary representation of the type **int** value (four bytes versus two).

11.2 BINARY FILES

- **The negative side to binary side usage:**

1. A binary file created on one computer is rarely readable on another type of computer.

2. A person cannot proofread the file by printing it out or by examining it in a word processor. (1)

3. A binary file cannot be created or modified in a word processor(2)

11.2 BINARY FILES

- The **stdio** library includes an input function **fread** that is comparable to **fwrite** .
- Function **fread** also requires four arguments:

1. **Address** of first memory cell to fill.
2. **Size** of one value.
3. **Maximum number of elements** to copy from the file into memory.
4. **File pointer** to a binary file opened in mode "**rb**" using function **fopen**

- Function **fread** returns as its value an integer indicating how many elements it successfully copied from the file (1)

11.2 BINARY FILES

➢ It is very important not to mix file types.

- A binary file created (written) using **fwrite** must be read using **fread** .
- A text file created using **fprintf** must be read using a text file input function such as **fscanf** .

- **Table 11.5 p.637** compares the use of text and binary files for input and output of data of various types. (1)

11.2 BINARY FILES

- In **Example 1 of Table 11.5** , we use **fopen** to open our input files, and we store the file pointers returned by **fopen** in variables of type **FILE*** .

- Notice that the type of the file pointer does not vary.

- Also notice that the form of the call to **fopen** for opening a binary file differs from the call for opening a text file only in the value of the mode (second argument). In fact, even this difference is optional .

11.2 BINARY FILES

- We see a similar situation in the opening of output files in **Example 2**.
- One consequence of this similarity is that the ability of the **C** compiler and run-time support system to detect misuse of a file pointer is severely limited.
- It is the programmer's responsibility to keep track of which type of file each file pointer accesses and to use the right I/O function at the right time.

11.2 BINARY FILES

- In **Examples 3 and 4 of Table 11.5** , we compare input/output of a user-defined structure type as it is done with text and binary files.

- In **Examples 5 and 6**, we see input/output of an array of type double values (1)

- **Example 7** demonstrates partially filling array **nums** and setting **n** to the number of elements filled.

- **Example 8** shows that all files—binary or text, input or output—are closed in the same way.

11.4 COMMON PROGRAMMING ERRORS

- Remember to declare a file pointer variable (type **FILE***) for each file you want to process.

- In a program that manipulates both file types, choose names for your file pointers that remind you of the type of file accessed. (2)

- It is easy to use the wrong library function with a file pointer. (1)

- It is also critical that you remember that library functions **fscanf** , **fprintf** , **getc** , and **putc** must be used for text I/O only;

- Functions **fread** and **fwrite** are applied exclusively to binary files. (3)

11.4 COMMON PROGRAMMING ERRORS

- The fact that **fprintf** , **fscanf** , and **getc** take the file pointer as their first argument while **putc** , **fread** , and **fwrite** take it as the last argument is definitely confusing at first.

- Keep in mind that opening a file for output by calling **fopen** with a second argument of "**w**" or "**wb**" typically results in a loss of any existing file whose name matches the first argument.

- It is easy to forget that binary files cannot be created, viewed, or modified using an editor or word processor program (1)

Thanks!

Refernces

Problem Solving and Program Design in C, 7th Ed., by Jeri R. Hanly and Elliot B. Koffman