UNIVERSITY of LOUISVILLE
dare to be great

C Chapter 11: File Input and Output

CECS130
Introduction to Programming Languages
Dr. Roman V. Yampolskiy

What is a file?

- Files are collections of information
  - Your source code that you've written for labs is stored in a file
  - This presentation is stored in a file
  - The Bloodshed compiler is stored in files
- A file can contain all sorts of information
- A file has nothing to do with programming, it is a construct of the operating system
- As programmers we need to deal with files from time to time

File Names

- Each operating system has certain rules for the naming of files
- In DOS, the name consists of a base name and an extension
  - Both can be whatever we want, but normally the extension describes the type of contents
    - cpp for C++, c for C
    - doc for Word Documents
    - txt for general text documents
    - exe for executable programs
- In DOS, we also specify where on our hard drive a file is located
  - We specify the drive
  - We specify the path
- Altogether, this is called the fully qualified name:

C:\Documents and Settings\MyDocuments\CECS130\Lectures\File.cpp

File Hierarchy

- Bit – **Bi**nary digi**t** zero or one
- Byte – 8 Bits – 1 character
- Field – grouping of characters [firstName = “Bob”]
- Record – grouping of fields [Bob, 36, Married]
- File – grouping of records

[[Bob, 36, Married]
[Anna, 18, Married]
[John, 26, Single]]

Using Files

- Programmers use files from within our programs to get or put information
- This is much more convenient than using the keyboard and monitor
- We always use 4 basic steps:
    1. Open the file
    2. Confirm we opened it
    3. Use it
    4. Close it

File Streams

- Streams are file or hardware devices such as monitor or printer
- We declare pointers to streams to get access to them
- To point to and manage a stream we use an internal data structure called FILE
- Pointers of type FILE are created like any other pointers:

```
FILE *pRead;

FILE *pWrite;

FILE *pAppend;
```

File Handles

What does it mean to “open” a file?

- Every time we access a file, the computer needs to know a great deal of information about it, e.g.:
  - Size of the file
  - Width of each line
  - Location of the file
  - Format of the contents
- It would be wasteful to determine all that information every time we read or write to the file

Opening Files

- fopen is used to open a file for formatted I/O and to associate a stream with that file.
- A stream is a source or destination of data.
- It may be a buffer in memory, a file or some hardware device such as a port.
- The prototype for fopen is:

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

Opening Files

- fopen returns a file pointer on success or NULL on failure.
- The file pointer is used to identify the stream and is passed as an argument to the routines that read, write or manipulate the file.
- The filename and mode arguments are standard null-terminated strings.

File Opening Modes

- The valid modes are shown below:

Mode	Use
r	open for reading
w	open or create for writing. Discard any previous contents.
a	open or create for writing. Append (write after) any previous contents.

Opening Files: Example

```
#include<stdio.h>

main() {

    FILE *pRead;
    pRead = fopen("myFile.txt", "r");

}
```

Closing Files

- Files are closed with the function fclose
- Its prototype is:

```
int fclose(FILE *stream);
```

- fclose returns zero on success and EOF on failure

Opening and Closing Files: Example

```
#include<stdio.h>

main() {

    FILE *pWrite;
    pWrite = fopen("someFile.dat", "w");
    fclose(pWrite);

}
```

Writing Data to Files: fprintf()

- Data is written to a file using fprintf.
- This function is very similar to printf.
- printf was used to write to standard output, stdout.
- fprintf has one additional argument to specify the stream to send data.
- Its prototype is:

int fprintf(FILE *stream, place holder, variable);

- fprintf returns the number of characters written if successful or a negative number on failure.

Writing Data to Files: fputs()

- Similarly you may write a string s to a file using fputs

  int fputs(const char *s, FILE *stream);

- fputs() return a non-negative number on success, or EOF on error.

- Example: fputs("Roman", pWrite);

Writing Data to Files: Example

```
#include<stdio.h>
main() {
        FILE *pWrite;
        char fName[20]; char lName [20]; char id[15];
        float gpa;
        pWrite = fopen("students.dat", "w");
        if (pWrite == NULL)
            printf("\nFile Can't Be Open\n");
        else {
            printf("\nEnter First Name, Last Name, ID and GPA\n\n");
            printf("Enter data separated by spaces: ");
            scanf("%s%s%s%f", fName, lName, id, &gpa);
            fprintf(pWrite, "%s\t%s\t%s\t%.2f\n", fName, lName, id, gpa);
            fclose(pWrite);
        }
}
```

Reading Data from Files: fscanf()

- Data is read from a file using fscanf.
- This function is very similar to scanf.
- It is used to read data from standard input, stdin.
- fscanf has one additional argument to specify the stream to read from.
- Remember that the argument to store data must be a pointer.
- The prototype for fscanf is:

int fscanf(FILE *stream, place holder, variable);

Reading Data from Files: fgets()

- Similarly you can get (read) a line from file using

  char *fgets(char *s, int n, FILE *stream);

- fgets returns a NULL when an error occurs.
- The EOF is considered to be an ERROR!

Example:

```
char *myString;
fgets(myString, 6, pRead);
```

Reading Data from Files: Example

```
#include<stdio.h>
main() {
        FILE *pRead;
        char name[10];
        char hobby[15];
        pRead = fopen("hobbies.dat", "r");
        if (pRead == NULL)
            printf("\nFile Can't Be Open\n");
        else {
            printf("\nName\tHobby\n\n");
            fscanf(pRead, "%s%s", name, hobby);
        }
        while (!feof(pRead)) {
            printf("%s\t%s\n", name, hobby);
            fscanf(pRead, "%s%s", name, hobby);
        }
        fclose(pRead);
}
```

Name Hobby

Mike Programming
Sheila Shopping
Bob Football
Olivia Dancing

Specifying File Path

- Backslashes in file names
  - Backslash is treated as an escape sequence (\n, \t)
  - To use backslash in a path name, double it

    pRead = **fopen("c:\\wrkspace\\files\\datafile.txt", "r");**

- Don't need a path name if the executable and the data file are in the same directory

goto statement

- goto – considered bad programming practice in object oriented langauges
  - Results in spaghetti like source code
- May be ok if used to assist with error handling in C

- First include a label (a descriptive name) followed by a colon (:) above the location you want to go to
- Next use the keyword goto followed by the label name

goto: Example – Input Validation

```
int x = 0;

getData:
    printf("\n Please enter a positive integer: ");
    scanf("%d", &x);

if (x < 0)
     goto getData;
else printf("Thank you for entering a positive integer");
```

Error Handling: exit()

- exit() function - a way to gracefully terminate your program
- Takes a single argument
  - EXIT_SUCCESS – exit program normally
  - EXIT_FAILURE – exit program with error

Example: **exit(EXIT_SUCCESS);**

- Part of <stdlib.h> library

Error Handling: perror()

- perror() function - a way to send an error message to standard output
- Takes a single argument
  - A text string to be printed before the system generated message about the last error which occured

Example:

perror("The program produced the following error: ");

Output: The program produced the following error: No error

The End!