Text and Binary Files Processing

COEN 11

Text Files

Text Files

- Collections of characters saved in a secondary storage (e.g., on a disk)
- Have no fixed size
- End marked with a special character <eof>
- End of lines are marked by a newline ('\n') character

Text Files

- Example
  - Text file:

```
This is a text file!
It has two lines.
```

  - Actual disk file:

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

Text Files

- All textual input and output data are actually a continuous stream of character codes
  - We refer to a data source or destinations as an input stream or an output stream
  - These general terms can be applied
    - to files
    - to the keyboard,
    - to the screen, and
    - to any other sources of input data or destinations of output data

Text Files

- Keyboard and Screen
  - stdin
    - keyboard's input stream
  - stdout
    - "normal" output stream associated with the screen
- Streams are treated like text files
  - Their individual components are characters

Text Files

- To read from stdin

  r = scanf ("%d", &num);

- The characters are read sequentially
  - Consecutive calls to scanf will read consecutive elements from the keyboard

Text Files

- scanf
  - Returns how many elements were read
  - Placeholders define the type
    - Examples
      - %c, %d, %f, %s

Text Files

- To write to stdout

  printf ("%d", num);

- The screen is written sequentially
  - Consecutive calls to printf will write consecutive elements to the screen

Text Files

- To open a text file for reading

```
FILE *infp;

if ((infp = fopen ("data.txt", "r")) == NULL)
   printf ("cannot open the file data.txt\n");
```

- The file is always open for reading from the beginning

Text Files

- To open a text file for reading when the name is stored in a string

```
char  file_name[50];
FILE *infp;

if ((infp = fopen (file_name, "r")) == NULL)
   printf ("cannot open the file %s\n", file_name);
```

Text Files

- To read from the text file

  ret = fscanf (infp, "%d", &num);

- The file is read sequentially
- Returned value
  - fscanf -- number of values converted
- At the end of the file
  - fscanf returns EOF

Text Files

- To open a text file for writing from the beginning

```
FILE *outfp;

if ((outfp = fopen ("data.txt", "w")) == NULL)
   printf ("cannot open the file data.txt\n");
```

Text Files

- To open a text file for writing from the beginning when the name is stored in a string

```
char file_name[50];
FILE *outfp;

if ((outfp = fopen (file_name, "w")) == NULL)
   printf ("cannot open the file %s\n", file_name);
```

Text Files

- To open a text file for writing from the end (append)

```
FILE *outfp;

if ((outfp = fopen ("data.txt", "a")) == NULL)
   printf ("cannot open the file data.txt\n");
```

Text Files

- To open a text file for writing from the end (append) when the name is stored in a string

```
Char  file_name[50];

FILE *outfp;
if ((outfp = fopen (file_name, "a")) == NULL)
   printf ("cannot open the file %s\n", file_name);
```

Text Files

- To write to the text file

  fprintf (outfp, "%d", num);

- The file is written sequentially
  - Consecutive calls to fprintf will write consecutive elements to the file

Text Files

- To close a text file

```
fclose (infp);
fclose (outfp);
```

Practice

Write an int function to return the average of the int numbers read from a text file. The function receives the file name as argument.

Binary Files

Binary Files

- When text files are used
  - Internal data needs to be converted to and from characters
  - These conversions are done by scanf and printf or by the program itself

Binary Files

- When a program produces output files which are used as input files for other programs
  - If there is no need for a human to read the file
    - Converting information into a stream of characters and back into internal format is a waste of computational cycles and time.
- To avoid these unnecessary conversions
  - Use binary files

Binary Files

- Files containing binary numbers that are the computer's internal representation of each file component
- Created by executing a program that stores directly in the computer's internal representation of each file component

Binary Files

- Actually just a stream of zeros and ones and cannot be read with a text editor

Binary Files

- Example
  - If 2 is written to a file as char, the file will have the following data:

00000010

  - If 2 is written to a file as short int, the file will have the following data:

0000000000000010

  - If 2 is written to a file as int, the file will have the following data:

00000000000000000000000000000010

Binary Files

- To open a binary file for reading

```
FILE *infp;
if ((infp = fopen ("data", "rb")) == NULL)
   printf ("cannot open the file data.txt\n");
```

- The file is always open for reading from the beginning
- The file name can also be stored in a string.

Binary Files

- To read from a binary file

  ret = fread (&x, sizeof (int), 1, infp);

  Read one integer from infp and store it to x (i.e., at the location given by &x)

  Return the number of items successfully read

Binary Files

- To read from a binary file

  ret = fread (&x, sizeof (int), 1, infp);

- The file is read sequentially
  - Consecutive calls to fread will read consecutive elements from the file
- The next integer is placed into the variable x
- Function fread returns the number of elements read

Binary Files

- To open a binary file for writing from the beginning

```
FILE *outfp;
if ((outfp = fopen ("data", "wb")) == NULL)
   printf ("cannot open the file data.txt\n");
```

- The file name can also be stored in a string

Binary Files

- To open a binary file for writing from the end (append)

```
FILE *outfp;
if ((outfp = fopen ("data", "ab")) == NULL)
   printf ("cannot open the file data.txt\n");
```

- The file name can also be stored in a string

Binary Files

- To write to a binary file

```
ret = fwrite (&x, sizeof (int), 1, outfp);
```

Binary Files

- To write to a binary file

  ret = fwrite (&x, sizeof (int), 1, outfp);

- The contents of variable x is written to the file
- The file is written sequentially
  - Consecutive calls to fwrite will write consecutive elements to the file
- Function fwrite returns the number of elements written.

Binary Files

- To close a binary file

```
fclose (infp);
fclose (outfp);
```

Files -- Functions

- fopen, fclose
- fprintf, fscanf
- fgets
- fread, fwrite
- fseek

Question 1

Write a function to return the number of even integers in a text file. The function receives the file pointer (FILE *) as an argument.

Question 2

Write a function to return the number of even integers in a binary file. The function receives the file pointer (FILE *) as an argument.

Question 3

Write a function to initialize array x of size SIZE with integers read from a text file. The function receives the name of the file as an argument.

Question 4

Write a function to initialize array x of size SIZE with integers read from a binary file. The function receives the name of the file as an argument.

Question 5

Write a void function to copy a file into another file. The function receives the two FILE pointers (src and dest) as arguments. Use fread, fwrite, and an array to make the process more efficient!

Question 6

Write a function to create a linked list with nodes obtained from a binary file. The function receives the file pointer (FILE *) as an argument.

Question 7

Write an int function to return the average of the int numbers read from a binary file. The function receives the name of the file. Use fread, fwrite, and an array to make the process more efficient!

Question 8

Write a void function to compare two files. Your function should receive the file pointers (src and dest) and printf either "Files are equal.\n" or "Files are not equal.\n". Use fread, fwrite, and an array to make the process more efficient!