Programming Fundamentals

Module H - Files

Le The Anh
`anhlt161@fe.edu.vn`

FPT Education
FPT UNIVERSITY

Objectives

Sequential Text Files

- A file is a named area of secondary storage. Secondary storage is permanent. Unlike the contents of primary memory, which is volatile, the contents of a file is accessible after we have turned the power off and back on at a later time.
- A file is not necessarily stored contiguously on a secondary storage device. The file may be fragmented. The operating system controls the fragmentation, if any.

Bytes

- The fundamental unit of a file is a byte.
- A file is a stream of bytes.
- A file concludes with a special mark called the end of file mark (EOF).

Formats

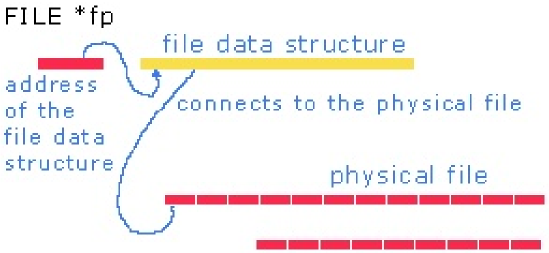
- We store data in a file in either of two formats: binary format, text format.
- In binary format, the data on the file is identical to the data stored in primary memory. Each byte on the file is a direct image of the corresponding byte in memory.
- In text format, the data on the file is in a form that we can display and modify using a text editor. Text format is portability.

Access

- Typically, a file consists of records that we can access in either of two ways: randomly (like CD's) or sequentially (like Cassette Tapes).
- To access records randomly, we fix their size. With the record sizes fixed, we can determine the location of the record that we are seeking, jump to that location and access that record directly.
- Under sequential access, we access the records in the order in which they were created. In this case, the records can vary in size. We do not need to know their sizes beforehand.

Access

- We connect a file to our program using a FILE data structure. This data structure holds the connection information for the file.


FILE *fp
file data structure
address of the file data structure
connects to the physical file
physical file

Connecting

- Declaration:

  ```
  FILE *identifier;
  ```

- For example,

  ```
  #include <stdio.h>
  ...

  FILE *fp = NULL;
  ```

Opening

- The fopen function connects a specific file to a program:
  ```
  FILE *fopen(char file_name[], char mode[]);
  ```
- file_name: a null-byte terminated string containing the name of the file
- mode: a null-byte terminated string containing the connection mode.

Mode	Read	Write	Create New[1]	Truncate
"r"	1	0	0	0
"w"	0	1	1	1
"a"	0	1	1	0
"r+"	1	1	0	0
"w+"	1	1	1	1
"a+"	1	1	1	0

[1] if the file doesn't exist, create a new file

Closing

- The fclose function disconnects a file from a program:
  ```
  int fclose(FILE *);
  ```
- If the program opened the file for writing, fclose writes any data remaining in the file stream's buffer to the file and concludes by appending an end of file mark immediately after the last character.
- If the program opened the file for reading, fclose ignores any data left in the file stream's buffer and closes the connection.

Example

```
#include <stdio.h>

int main( )
{
    FILE *fp = NULL;

    fp = fopen("alpha.txt","w");
    if (fp != NULL) {
        /* we will add statements here later */

        // fclose returns 0 if successful, EOF if unsuccessful
        fclose(fp);
     }
    else
        printf("Failed to open file\n");

    return 0;
}
```

Writing to a File

- The function fprintf sends data from primary memory to a connected file under format control:

  ```
  int fprintf(FILE *, char [], ...);
  ```

- The first parameter receives the address of the file connection data structure.
- The second parameter is the format string containing the text to be written directly to the file and the conversion specifiers, if any, to be applied to the data values received in the parameters following the format string.
- Note the similarity between the fprintf and the printf library functions.

Example

```c
#include <stdio.h>

int main()
{
    FILE *fp = NULL;
    char phrase[] = "I was created by a C program";

    fp = fopen("alpha.txt","w");
    if (fp != NULL)
    {
        fprintf(fp, "%s\n", phrase);
        fclose(fp);
    }
    else
        printf("Failed to open file\n");

    return 0;
}
```

Write to File

- The fputc function writes a single character to a file:
  ```
  int fputc(int ch, FILE *fp);
  ```
- The fputs function writes a null-byte terminated string to a file:
  ```
  int fputs(char str[], FILE *fp);
  ```

Reading from a File

- The fscanf function reads a sequence of bytes from a connected file into primary memory under format control.
  ```
  int fscanf(FILE *, char [], ...);
  ```
- For example,
  ```
  #include <stdio.h>

  int main( )
  {
      FILE *fp = NULL;
      char phrase[61];

      fp = fopen("alpha.txt","r");
      if (fp != NULL)
      {
          fscanf(fp, "%60[^\n]\n", phrase);
          printf("You read : %s\n", phrase);
          fclose(fp);
      }
      else
          printf("Failed to open file\n");

      return 0;
  }
  ```

Reading from File

- The fgetc function extracts a single character from a file:
  ```
  int fgetc(FILE *fp);
  ```
  fp is the address of the connection data structure for the file. fgetc returns the next character from fp, or EOF in the event of an error.
- The fgets function extracts a contiguous series of bytes from a file
  ```
  char* fgets(char str[], int max, FILE *fp);
  ```
  The string str will hold the set of bytes ending in a newline character up to max-1 bytes from fp. fgets appends the null byte to the string stored in primary memory. fgets returns the address of str, or NULL.

Rewinding

- The rewinding function is used to jump to the begining of a file (without the need of disconnecting and reconnecting the file).

  ```
  void rewind(FILE *fp);
  ```

Rewinding

- The example below read the file twice:

```c
#include <stdio.h>

int main( )
{
    FILE *fp = NULL;
    char phrase[61];

    fp = fopen("spring.txt","r");
    if (fp != NULL) {
        while (fscanf(fp, "%60[^\n]%*c", phrase) != EOF)
            printf("%s\n", phrase);
        rewind(fp);
        while (fscanf(fp, "%60[^\n]%*c", phrase) != EOF)
            printf("%s\n", phrase);

        fclose(fp);
     }
     else
        printf("Failed to open file\n");

    return 0;
}
```

End of File

- The function feof indicates whether or not a program has attempted to read beyond the last character in a file.

  ```
  int feof(FILE *fp);
  ```

Comparison

- The library functions for file I/O share some common properties with the standard I/O library functions, but also some differences.

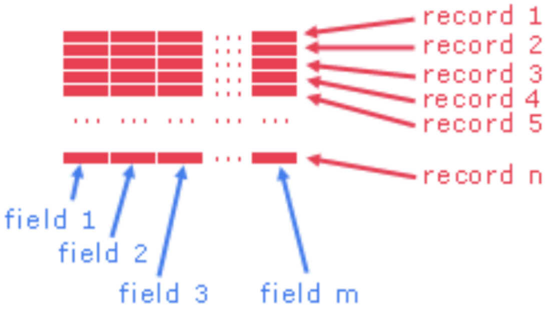
Type	Standard I/O	File I/O	Comments
int	**getchar()**	**fgetc(fp)**	in both cases, check to see if the return value is **EOF**, before converting it to a **char** data type
int	**putchar(ch)**	**fputc(ch, fp)**	in both cases, check to see if the return value is **EOF**
char *	**gets(str)**	**fgets(str, max, fp)**	both functions add the **'\0'** byte; **gets** discards the **'\n'** delimiter; **fgets** returns **NULL** on encountering an end of file mark
int	**puts(str)**	**fputs(str, fp)**	**puts** adds **'\n'** to the file; **fputs** does not add **'\n'** to the file;

In-Class Practice

- Given a text file named "inp.txt" that contains one integer n, write a program to read n from the inp.txt file, then print to the "out.txt" file the first three prime numbers after n.

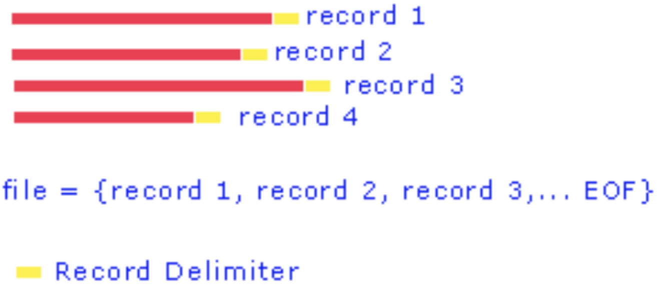
Records, Fields and Tables

- A file consists of records. A record consists of fields. A collection of records with equal numbers of fields in each record constitutes a table.



Records

- We call each line in a text file a record. That is, a record is a sequence of characters that ends with a newline delimiter. Typically, one record refers to one entity of information.



Example

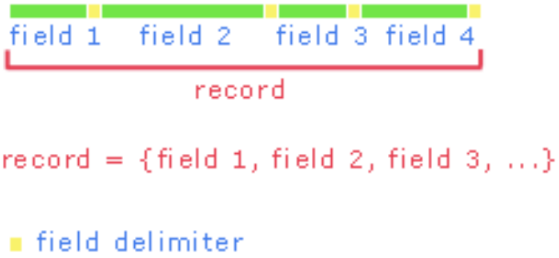
```c
// determine the number of records in winter.txt file
// by counting number of newline characters

#include <stdio.h>
int main()
{
    FILE *fp = NULL;
    int c, nrecs;

    fp = fopen("winter.txt","r");
    if (fp != NULL)
    {
        nrecs = 0;
        do
        {
            c = fgetc(fp);
            if (c != EOF)
            {
                if ((char)c == '\n')
                    nrecs++;
            }
        } while (!feof(fp));
        printf("%d records on file\n", nrecs);
        fclose(fp);
    }
    return 0;
}
```

Fields

- A field is one element of a record.
- A field holds an integral piece of data.
- A field may be of any data type.
- We separate any two adjacent fields in a record using a field delimiter.



Example

- Consider the file “winter.txt” containing the following information:
  5 Pairs of Boots
  2 Coats
  3 Hats
  3 Pairs of Gloves
- Write a program to read the records on the file and display them in tabular format.

Example

```c
#include <stdio.h>

int main()
{
    FILE *fp = NULL;
    char label [21];
    int n;

    fp = fopen("winter.txt", "r");
    if (fp != NULL)
    {
        printf("        Winter Items\n"
               "        ============\n\n"
               "No Description       \n"
               "---------------------\n");
        while (fscanf(fp,"%d %20[^\n]%*c", &n, label) == 2)
            printf("%6d %-20s\n", n, label);
        fclose(fp);
    }
    return 0;
}
```

Tables

- Consider the file string.txt below:
  2;Light Jacket;95.89
  3;Long-Sleeved Shirts;67.89
  2;Large Skateboards;45.98
- Write a program to read the records on the file and display them in tabular format.

Tables

```c
#include <stdio.h>

int main( void )
{
    FILE *fp = NULL;
    char label [21];
    int n;
    double price;

    fp = fopen("spring.txt","r");
    if (fp != NULL)
    {
        printf("        Spring Items\n"
                "       ============\n\n"
                "No Description        Price\n"
                "---------------------------\n");
        while (fscanf(fp,"%d;%20[^;];%lf%*c", &n, label, &price) == 3)
            printf("%2d %-20s%5.2lf\n", n, label, price);
        fclose(fp);
    }
    return 0;
}
```

Oversize Strings

```c
#include <stdio.h>

int main()
{
    FILE *fp = NULL;
    char label [21], c;
    int n;
    double price;

    fp = fopen("spring.txt","r");
    if (fp != NULL)
    {
        printf("        Spring Items\n"
               "        ============\n\n"
               "No Description       Price\n"
               "--------------------------\n");
        while (fscanf(fp,"%d;%20[^;]%c", &n, label, &c) == 3)
        {
            if (c != ';')
                fscanf("%*[^;];%lf%*c", &price);
            else
                fscanf("%lf\n", &price);
            printf("%2d %-20s%5.2lf\n",
                     n, label, price);
        }
        fclose(fp);
    }
    return 0;
}
```

Summary

- Records, Fields and Tables
  - Records
  - Fields
  - Tables

Q&A

In-Class Practice