HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

File

Content

- Basic concept
- Open and close files
- File operations
- File processing character by character
- Text file processing line by line
- Reading/writing data in specific formats

File Handling

- C communicates with files using a new datatype called a file pointer.

- File pointer:
  - references a disk file.
  - used by a stream to conduct the operation of the I/O functions.

- FILE *fptr;

4 major operations

- Open the file
- Read from a file → program
- Write to a file: Program → file
- Close the file.

Text File vs Binary File

- Text File:
  - It contains visible characters and can be created using common software like Notepad, Notepad++, Sublime Text, etc.
  - It's convenient for everyday use but lacks security and requires more storage space.

- Binary File:
  - Stores data in binary form (a string of 0s and 1s).
  - It offers better security and storage efficiency.

Opening a file

- fopen() function.
- FILE *fopen(const char *filename, const char *mode);

```
FILE* fptr;
if ((fptr = fopen("test.txt", "r")) == NULL) {
    printf("Cannot open test.txt file.\n");
    exit(1);
}
```

Opening a file

- filename: name of the file.
  - It can be a string literal: **“data.txt”**
  - It may contain the full path of the file: **“/root/hedspi/CProgrammingBasic/Lab1/data.txt”**
  - It may be a character array that contains the file name:

    char file_name[] = “junk.txt”;

- ***NOTE:*** *If the file path is not specified, the file is located in the same folder as the C program.*

Modes for text file

mode	Description
"r"	opens an existing text file for reading.
"w"	creates a text file for writing.
"a"	opens an existing text file for appending.
"r+"	opens an existing text file for reading or writing.
"w+"	creates a text file for reading or writing.
"a+"	opens or create an existing text file for appending.

Modes for binary file

mode	Description
"rb"	opens an existing binary file for reading.
"wb"	creates a binary file for writing.
"ab"	opens an existing binary file for appending.
"r+b"	opens an existing binary file for reading or writing.
"w+b"	creates a binary file for reading or writing.
"a+b"	opens or create an existing binary file for appending.

Closing a file

- The fclose command can be used to disconnect a file pointer from a file.
- int fclose(FILE *stream);
- To close a file when finished with reading or writing operations.

Example

- Example 1. Open and close file

```
#include <stdio.h>
enum { SUCCESS, FAIL };
main() {
   FILE* fptr;
   char filename[] = "haiku.txt";
   int reval = SUCCESS;
   if ((fptr = fopen(filename, "r")) == NULL) {
      printf("Cannot open %s.\n", filename);
      reval = FAIL;
   }
   else {
      printf("The value of fptr: 0x%p\n", fptr);
      printf("Ready to close the file.");
      fclose(fptr);
   }
   return reval;
}
```

HUST

hust.edu.vn fb.com/dhbkhn

Processing a file character by character

Reading and Writing Disk Files

- In C, you can perform I/O operations in the following ways:
  - **Read or write one character at a time.**

  - **Read or write one line of text (that is, one character line) at a time.**

  - Read or write one block of characters at a time.
- **Read or write one character at a time.**
  - Character input and output
    - fgetc() and fputc()
  - int fgetc(FILE *stream);
  - int fputc(int c , FILE *stream);

Example

- Example 1. Create a text file name lab1.txt with the content as you want.
- Write a program to read from a text file one character at a time, then write it to a new file with the name lab1w.txt

```
#include <stdio.h>
enum { SUCCESS, FAIL };

void CharReadWrite(FILE* fin, FILE* fout)
{
    int c;
    while ((c = fgetc(fin)) != EOF) {
        fputc(c, fout);  /* write to a file */
        putchar(c);
        /* display character on the screen */
    }
}
```

Example

```
main(void) {
   FILE* fptr1, * fptr2;
   char filename1[] = "lab1a.txt";
   char filename2[] = "lab1.txt";
   int reval = SUCCESS;

   if ((fptr1 = fopen(filename1, "w")) == NULL) {
      printf("Cannot open %s.\n", filename1);
      reval = FAIL;
   }
   else if ((fptr2 = fopen(filename2, "r")) == NULL) {
      printf("Cannot open %s.\n", filename2);
      reval = FAIL;
   }
   else {
      CharReadWrite(fptr2, fptr1);
      fclose(fptr1);
      fclose(fptr2);
   }
   return reval;
}
```

- Example 2. Write a program to read sentences from a specified file one character at a time.
- Each capital letter is converted into a lower-case letter, and each lower-case letter is converted into a capital letter. The new sentence is then written into another file.
- Note that you must output numbers, the signs as they are.

```
void CharReadWrite(FILE* fin, FILE* fout)
{
    int c;
    while ((c = fgetc(fin)) != EOF) {
        if islower(c) c = toupper(c);
        else if isupper(c) c = tolower(c);
        fputc(c, fout);  /* write to a file */
        putchar(c); /* display character on the screen */
    }
}
```

Exercise

- Exercise 1. "Write a program named 'mycp' that functions similarly to the 'cp' command in UNIX/LINUX operating systems. It can copy a text file to a new file using the syntax:

- mycp <file_1> <file_2>
- The file paths and names are provided as command-line arguments.

- Note: The program should validate the usage syntax (e.g., number of arguments - show error messages and display guidance when necessary)."

Exercise

- Exercise 2: Write a program that takes the names of two files as command-line arguments and appends the content of the second file to the end of the first file. Assuming both files exist.

- Usage syntax:

```
apd <file1> <file2>
```

- Note: The program should validate the usage syntax (e.g., number of arguments - show error messages and display guidance when necessary).

Exercise

- Exercise 3. Write a program called "uconvert" that converts all the letters in the content of a specific file (provided as a command-line argument) to uppercase and writes the modified content back to the same file.

- Syntax: uconvert tata.txt

- For example:
  - Source file tata.txt content: helloworld
  - Content of tata.txt after running the program: HELLOWORD.

Exercise

- Exercise 4. Write a program that can perform both encoding and decoding simultaneously on a text file using the Caesar cipher (additive cipher). The program accepts three arguments: <source file> <shift amount> <destination file>.
- For encoding, run the program with a positive integer shift amount, 'n.' The program will replace each character in the file with a character that is 'n' positions ahead in the ASCII character set. For example, with an offset of 3, A -> D, B -> E.
- For decoding, run the program with the encoded file and a negative shift amount (e.g., offset = -3), which corresponds to the reverse process of the encoding.
- Advanced functionality (optional): For alphabetic characters, perform a circular shift: A -> D, ..., Z -> C.

HUST

Process by line

hust.edu.vn fb.com/dhbkhn

Read or write one line at a time.

- Two functions: fgets() and fputs()
- char *fgets(char *s, int n, FILE *stream);
  - s references an array that is used to store characters
  - n specifies the maximum number of array elements.
- fgets() function can read up to n-1 characters, and can append a null character after the last character fetched, until a newline or an EOF is encountered.
- int fputs(const char *s, FILE *stream);
  - s: array that contains the characters to be written to a file
- return value
  - 0 for success
  - non zero in case of fail.

Example

- Example 1. Re-implement the file content copying programming exercise, but instead of using the pair of functions fgetc and fputc, use the pair of functions fgets and fputs to read from the file and write to the file, processing one line of text content at a time.

```c
#include <stdio.h>
enum { SUCCESS, FAIL };
#define MAX_LEN 81

void LineReadWrite(FILE* fin, FILE* fout)
{
    char buff[MAX_LEN];
    while (fgets(buff, MAX_LEN, fin) != NULL) {
        fputs(buff, fout);
        printf("%s", buff);
    }
}
```

Example

```
main(void) {
    FILE* fptr1, * fptr2;
    char filename1[] = "lab1a.txt";
    char filename2[] = "lab1.txt";
    int reval = SUCCESS;

    if ((fptr1 = fopen(filename1, "w")) == NULL) {
        printf("Cannot open %s.\n", filename1);
        reval = FAIL;
    }
    else if ((fptr2 = fopen(filename2, "r")) == NULL) {
        printf("Cannot open %s.\n", filename2);
        reval = FAIL;
    }
    else {
        LineReadWrite(fptr2, fptr1);
        fclose(fptr1);
        fclose(fptr2);
    }
    return reval;
}
```

Example

- Example 2. Modify the file copying program from the previous slide so that the program only displays the file's content on the screen and then shows the number of lines of text.

- Illustration of the program's interface:
  - Reading file Haiku.txt…. done!
  - Haiku haiku
  - Tokyo
  - Hanoi
  - This file has 3 lines.
- Modify the source code of the LineReadWrite function:
  - Remove the fputs command
  - Increase the line counter each time a line is read.

Example

```
enum { SUCCESS, FAIL };
int LineReadWrite(FILE* fin)
{
   char buff[MAX_LEN]; int count = 0;
   while (fgets(buff, MAX_LEN, fin) != NULL) {
      count++; printf("%s", buff);
   }
   return count;
}
```

```
main() {
   FILE* fptr1; int c = 0;
   char filename1[] = "haiku.txt";
      int reval = SUCCESS;

   if (fptr1 = fopen(filename1, "r")) == NULL){
      printf("Cannot open %s.\n", filename1);
      reval = FAIL;
   }
   else {
      printf("Reading file % s … done!\n", filename1);
      c = LineReadWrite(fptr2, fptr1);
      printf("The file has % d lines.\n", c);
      fclose(fptr1);
   }
   return reval;
}
```

Exercise

- Exercise 1. Write a program called "mycat" that reads and displays the content of a text file on the screen. The program supports two syntaxes as follows:
- cat <filename>: Displays the entire content at once.
- cat <filename> -p: Displays content page by page. The user presses a key to view the next page.

- Exercise 2. Write a program that takes a command-line argument, which is the path to a text file (content is under 80 lines). The program adds a new line at the end of the mentioned file, containing the initial characters of each line in the original file.

HUST

hust.edu.vn fb.com/dhbkhn

Read and write formated text

Read and write formated text

- These are useful functions for handling structured data (across various data types) from text.

- int fscanf(FILE *stream, const char *format, ...);
  - The fscanf function works similarly to scanf, but the difference is that it reads content from a file (represented by the file pointer) to write into variables.

- int fprintf(FILE *stream, const char *format, ...);
  - Similar to printf, but instead of displaying content on the screen, it writes the content from the arguments to a file (represented by the file pointer).

Example

- Example 1. Write a program to read each line of text from a file, then calculate the length of the character string in each line and write to a new file in the following format: <line length> <line content>
- For example, for a line in the input file:
  - The quick brown fox jumps over the lazy dog.
  - The corresponding line in the output file will be:
  - 44 The quick brown fox jumps over the lazy dog.

```
void LineReadWrite(FILE* fin, FILE* fout)
{
   char buff[MAX_LEN];
   int len;
   while (fgets(buff, MAX_LEN, fin) != NULL) {
      len = strlen(buff) - 1;
      fprintf(fout, "%d %s", len, buff);
      printf("%s", buff);
   }
}
```

Example

- Example 2. Write a program to read a sequence of numbers from the keyboard and write them to the file 'out.txt' in reverse order. Additionally, write the sum of the numbers at the end of the file.
- The input syntax from the keyboard is as follows: The first number is the count of the numbers to be input, followed by the sequence of integers. For example, if the user enters: 4 12 -45 56 3
- "4" is the count of numbers to be input, including "12 -45 56 3". The content of the 'out.txt' file will be, with 26 as the sum of the 4 numbers:
- 3 56 -45 12 26
- Because the number of input numbers varies with each run, the program needs to dynamically allocate memory for these numbers using the malloc() function

Example

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

enum { SUCCESS, FAIL };

int main(void)
{
   FILE* fp;
   int* p;
   int i, n, value, sum;
   int reval = SUCCESS;

   printf("Enter a list of numbers with the first is the size of list: \n");
   scanf("%d", &n);
   p = (int*)malloc(n * sizeof(int)); i = 0; sum = 0;
   while (i < n) {
      scanf("%d", &value);
      p[i++] = value;
      sum += value;
   }
```

Example

```
    if ((fp = fopen("out.txt", "w")) == NULL) {
        printf("Can not open %s.\n", "out.txt");
        reval = FAIL;
    }
    for (i = n - 1; i >= 0; i--) {
        fprintf(fp, "%d ", p[i]);
    }
    fprintf(fp, "%d ", sum);
    fclose(fp);
    free(p);
    return reval;
}
```

Example

- Example 3. Create a text file named 'product.txt', where each line contains information about a product: ID (integer type), Product Name (a string without whitespace characters), Price (double type). The data fields are separated by a space or tab character. For example
    - 1 Samsung_Television_4K 20000000
    - 2 Apple_MacBook_2020   18560000
- Write a program to read the file into an array of structure elements and then display the content of the array on the screen in the form:
    - No   Product Name   Price
    - 1 Samsung_Television_4K   20000000
    - ...
- Suggestion
    - When reading double-precision floating-point numbers, use fscanf with the format string "%lf".
    - In cases where the data fields are separated by delimiters such as ';' or ',' (delimiters), you can combine fscanf and fgetc to read each field.
    - Example: "1000, John_Allan, 28, NewYork"

Example

```
#include <stdio.h>
enum { SUCCESS, FAIL };
#define MAX_ELEMENT 10
typedef struct {
    int no;
    char name[20];
    double price;
}product;

int main(void) {
    FILE* fp;
    product arr[MAX_ELEMENT];
    int i = 0, n;
    int reval = SUCCESS;
    printf("Loading file...\n");
    if ((fp = fopen("product.txt", "r")) == NULL) {
        printf("Can not open %s.\n", "product.txt");
        reval = FAIL;
    }
```

Example

```
    else {
        while (fscanf(fp, "%d%s%lf", &arr[i].no, arr[i].name, &arr[i].price) != EOF) {
            //printf("%-6d%-24s%-6.2f\n",arr[i].no,arr[i].name,arr[i].price);
            i++;
        }
        n = i;
        for (i = 0; i < n; i++)  printf("%-6d%-24s%-6.2f\n",
                        arr[i].no, arr[i].name, arr[i].price);
    }
    fclose(fp);
    return reval;
}
```

Exercise

- Exercise 1. Create a text file with the content of a class list containing at least 6 students. Each line should consist of the following 4 fields:
- Order ID Student ID Full name (no spaces) Phone number. Example
  - 1 20181110 Bui_Van 0903112234
  - 2 20182111 Joshua_Kim 0912123232
- Write a program to read the file into a suitable array of structures. The program should prompt to add an additional field of scores for each student and then write the results to a file named "bangdiem.txt" (transcript.txt), including all the aforementioned fields (and the field for scores).