Input/Output (1)

Program Design (II)

2022 Spring

Fu-Yin Cherng
Dept. CSIE, National Chung Cheng University

All materials are provided by C Programming: A Modern Approach by K. N. King, 2nd edition, 2008, W. W. Norton & Company.

Outline

- Introduction of Input/Output
- Streams

Introduction

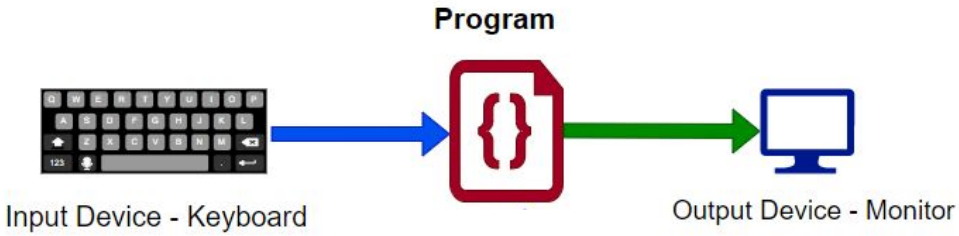
- C's input/output library is the biggest and most important part of the standard library.
- The `<stdio.h>` header is the primary repository of input/output functions, including `printf`, `scanf`, `putchar`, `getchar`, `puts`, and `gets`.
- This chapter provides more information about these six functions.
- It also introduces many new functions, most of which deal with files.

Introduction

- Topics to be covered:
  - Streams, the `FILE` type, input and output redirection, and the difference between text files and binary files
  - Functions designed specifically for use with files, including functions that open and close files
  - Functions that perform "formatted" input/output
  - Functions that read and write unformatted data (characters, lines, and blocks)
  - Random access operations on files
  - Functions that write to a string or read from a string

Streams

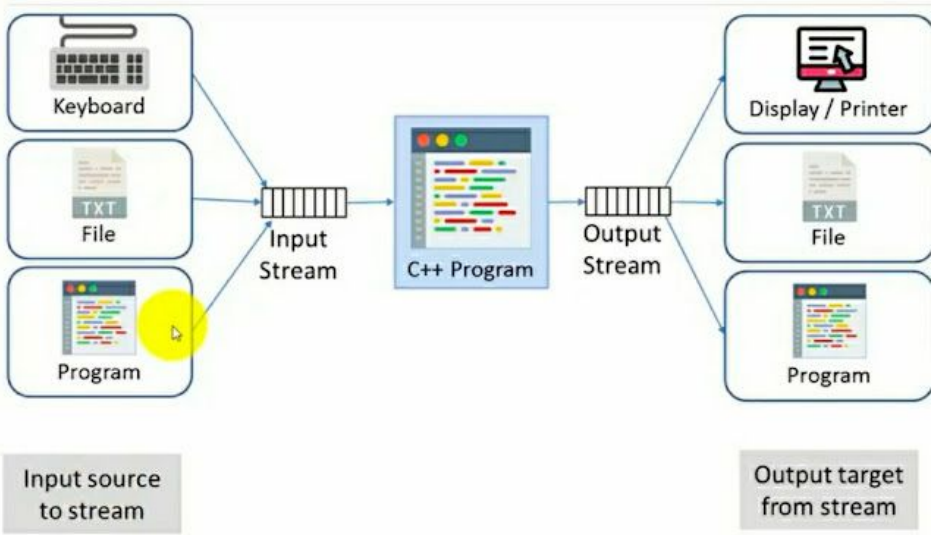
- In C, the term ***stream*** means any source of input or any destination for output.
- Many small programs obtain all their input from one stream (the keyboard) and write all their output to another stream (the screen).



https://stackoverflow.com/questions/38652953/what-does-stream-mean-in-c

Streams

- Larger programs may need additional streams.
- Streams often represent files stored on various media.
- However, they could just as easily be associated with devices such as network ports and printers.



https://knowproinsight.blogspot.com/2019/10/concept-of-stream-and-its-bytestream.html

File Pointers

- Accessing a stream is done through a ***file pointer,*** which has type `FILE *`.
- The `FILE` type is declared in `<stdio.h>`.
- Certain streams are represented by file pointers with standard names.
- Additional file pointers can be declared as needed:

```
FILE *fp1, *fp2;
```

Standard Streams and Redirection

- `<stdio.h>` provides three standard streams:

File Pointer	***Stream***	***Default Meaning***
`stdin`	Standard input	Keyboard
`stdout`	Standard output	Screen
`stderr`	Standard error	Screen

- These streams are ready to use—we don't declare them, and we don't open or close them.

Standard Streams and Redirection

- The I/O functions discussed in previous chapters obtain input from `stdin` and send output to `stdout`.
- Many operating systems allow these default meanings to be changed via a mechanism known as ***redirection.***

Standard Streams and Redirection

- A typical technique for forcing a program to obtain its input from a file instead of from the keyboard:

  ```
  demo <in.dat
  ```

  This technique is known as ***input redirection.***

Standard Streams and Redirection

- ***Output redirection*** is similar:

  ```
  demo >out.dat
  ```

  All data written to `stdout` will now go into the `out.dat` file instead of appearing on the screen.

Standard Streams and Redirection

- Input redirection and output redirection can be combined:

```
demo <in.dat >out.dat
```

- The < and > characters don't have to be adjacent to file names, and the order in which the redirected files are listed doesn't matter:

```
demo < in.dat > out.dat
demo >out.dat <in.dat
```

Reveiw of redirection - Text Formatting Program

- W5 - Writing Large Programs (2)
- The < symbol informs the **operating** system that `justify` will **read from the file** `input.txt` instead of **accepting input from the keyboard**.
- This feature, supported by UNIX, Windows, and other operating systems, is called ***input redirection.***

```
./justify <input.txt
```

```
 C     is quirky,  flawed,    and  an
enormous   success.      Although accidents of   history
 surely  helped,   it evidently    satisfied   a   need

    for  a   system  implementation    language    efficient
 enough   to  displace         assembly   language,
   yet sufficiently   abstract   and fluent    to describe
  algorithms   and     interactions    in a   wide   variety
of   environments.
                                --      Dennis     M.
```

Example: Text Formatting

- The output of `justify` will **normally** appear on the **screen**, but we can save it in a file by using ***output redirection:***

```
C is quirky,  flawed,  and  an  enormous  success.  Although
accidents of history surely helped, it evidently satisfied a
need for a system implementation language  efficient  enough
to displace assembly language, yet sufficiently abstract and
fluent to describe algorithms and  interactions  in  a  wide
variety of environments. -- Dennis M.
```

```
./justify <input.txt >output.txt
```

Standard Streams and Redirection

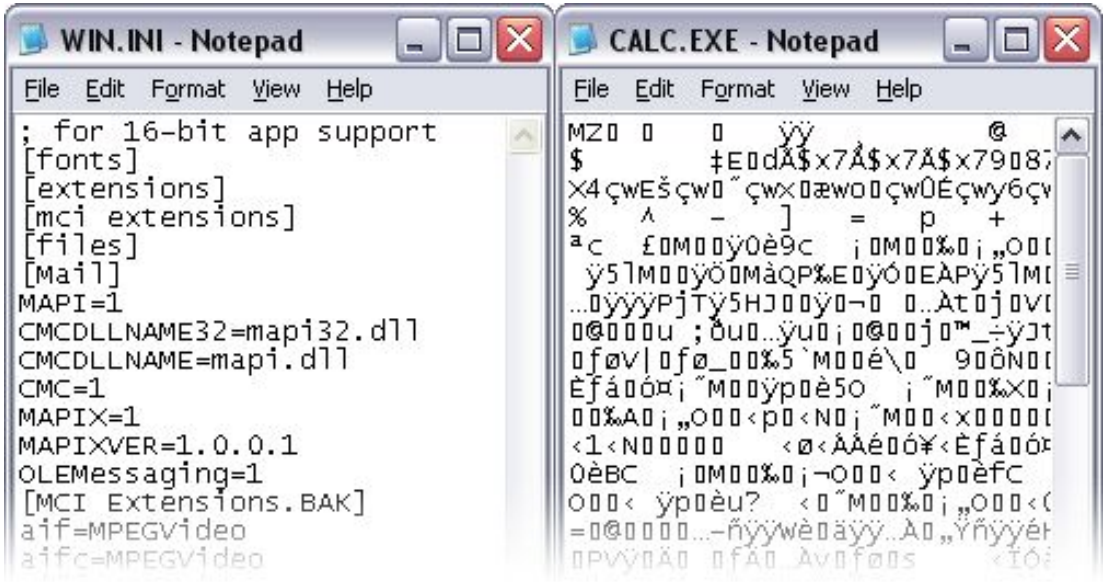
- One problem with output redirection is that *everything* written to `stdout` is put into a file.
- Writing error messages to `stderr` instead of `stdout` guarantees that they will appear on the screen even when `stdout` has been redirected.

```
#include <stdio.h>

int main(){
    // if detected error
    fprintf(stderr, "Error Message");
}
```

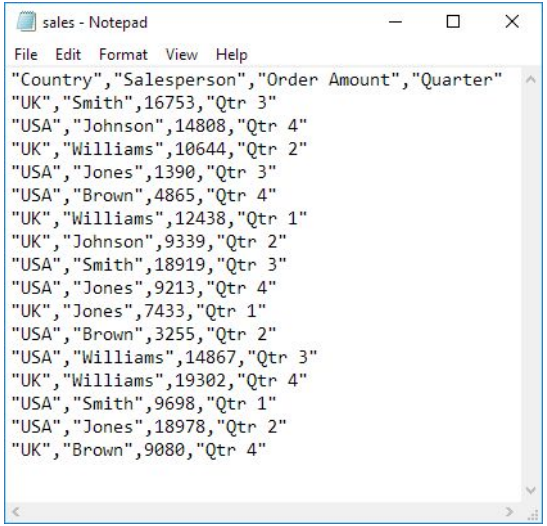
Text Files versus Binary Files

- `<stdio.h>` supports two kinds of files: **text** and **binary**.



Text Files versus Binary Files

- The bytes in a ***text file*** represent characters, allowing humans to examine or edit the file.
  - The source code for a C program is stored in a text file.




```c
#include<stdio.h>

int main()
{
    unsigned int m = 32;
    printf("%x\n", ~m);
    return 0;
}
```

Text Files versus Binary Files

- In a ***binary file,*** bytes don't necessarily represent characters.
  - Groups of bytes might represent other types of data, such as integers and floating-point numbers.
  - An executable C program is stored in a binary file.

```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

A 318 byte
Wikipedia favicon,
or Wikipedia's icon.

Text Files versus Binary Files

- Text files have two characteristics that binary files don't possess.
- ***1. Text files are divided into lines.*** Each line in a text file normally ends with one or two special characters.
  - Windows: carriage-return character (`'\r'` or ASCII value `'\x0d'`) followed by line-feed character (`'\n'` or ASCII value `'\x0a'`)
  - UNIX and newer versions of Mac OS: line-feed character
  - Older versions of Mac OS: carriage-return character
- Extra Reading: [What are carriage return, linefeed, and form feed?]

Text Files versus Binary Files

- ***2. Text files may contain a special "end-of-file" marker.***
- In a binary file, there are no end-of-line or end-of-file markers; all bytes are treated equally.

Text Files versus Binary Files

- When data is written to a file, it can be stored in text form or in binary form.
- One way to store the number 32767 in a file would be to write it in text form as the characters 3, 2, 7, 6, and 7:

00110011	00110010	00110111	00110110	00110111
'3'	'2'	'7'	'6'	'7'

Text Files versus Binary Files

- The other option is to store the number in binary, which would take as few as two bytes
- Storing numbers in binary can often save space.

01111111	11111111

Text Files versus Binary Files

- Programs that read from a file or write to a file must take into account whether it’s text or binary.
- A program that displays the contents of a file on the screen will probably assume it’s a text file.

Text Files versus Binary Files

- A file-copying program, on the other hand, can't assume that the file to be copied is a text file.
  - If it does, binary files containing an end-of-file character won't be copied completely.
- When we can't say for sure whether a file is text or binary, it's safer to assume that it's binary.

Summary

- Introduction of Input/Output
- Streams
  - File Pointer
  - Standard Streams and Reireaction
  - Text Files versus Binary Files
- How to open binary files using VS code?