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).
- Larger programs may need additional streams.
- However, they could just as easily be **associated with devices such as network ports and printers**.

File Pointers

- Accessing a stream is done through a ***file pointer***, which has type `FILE *`.

- The `FILE` type is declared in `<stdio.h>`.

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.

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.

- 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.

Text Files versus Binary Files

- Text files have characteristics that binary files don't possess.
- ***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'`) followed by line-feed character (`'\n'`)
  - **UNIX and newer versions of Mac OS**: line-feed character '\n'
  - **Old versions of Mac OS**: carriage-return character '\r'

Opening a File

- Opening a file for use as a stream requires a call of the `fopen` function.
- Prototype for `fopen`:

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

- `filename` is the name of the file to be opened.
  - This argument may include information about the file's location, such as a drive specifier or path.
- `mode` is a "mode string" that specifies what operations we intend to perform on the file.

Modes

- Mode strings for text files:

String	*Meaning*
`"r"`	Open for reading
`"w"`	Open for writing (file need not exist)
`"a"`	Open for appending (file need not exist)
`"r+"`	Open for reading and writing, starting at beginning
`"w+"`	Open for reading and writing (truncate if file exists)
`"a+"`	Open for reading and writing (append if file exists)

Opening a File

- In Windows, be careful when the file name in a call of `fopen` includes the `\` character.
- The call

  ```
  fopen("c:\project\test1.dat", "r")
  ```

  will fail, because `\t` is treated as a character escape.
- One way to avoid the problem is to use `\\` instead of `\`:

  ```
  fopen("c:\\project\\test1.dat", "r")
  ```
- An alternative is to use the `/` character instead of `\`:

  ```
  fopen("c:/project/test1.dat", "r")
  ```

Closing a File

- The `fclose` function allows a program to close a file that it's no longer using.
- The argument to `fclose` must be a file pointer obtained from a call of `fopen` or `freopen`.
- `fclose` returns zero if the file was closed successfully.
- Otherwise, it returns the error code `EOF` (a macro defined in `<stdio.h>`).

I/O Functions for Streams

- Character I/O
  - `fputc( ch, fptr)`
  - `ch = fgetc( fptr )`
- Line I/O
  - `fputs("Hello world!\n", fptr)`
  - `fgets( str, sizeof(str), fptr )`
- Formatted I/O
  - `fscanf( fptr, "……", …… )`
  - `fprintf( fptr, "……", …… )`