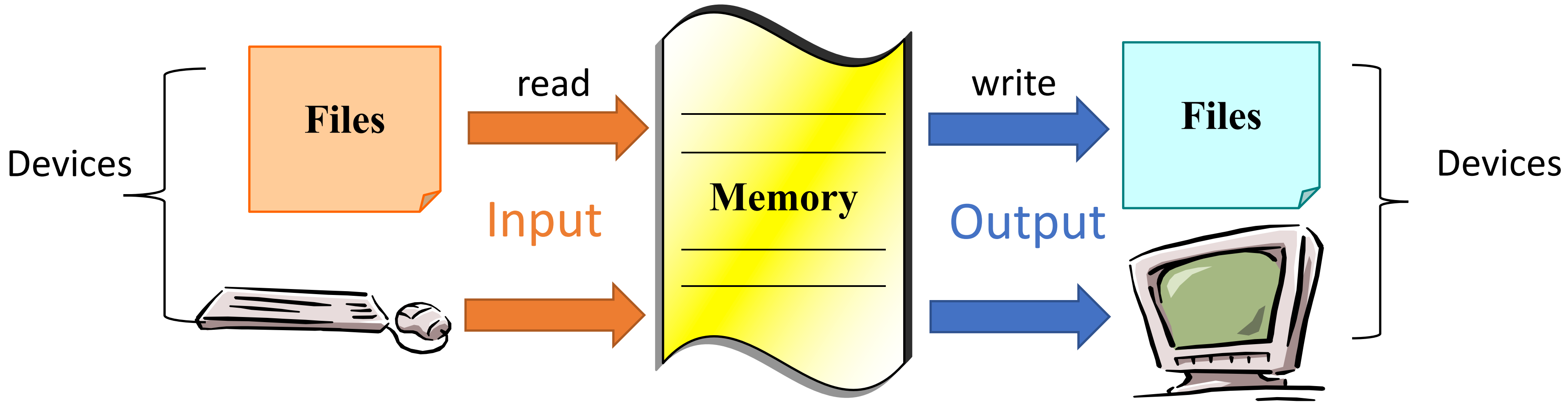
Class 12 File(文件)

Input & Output in C Language

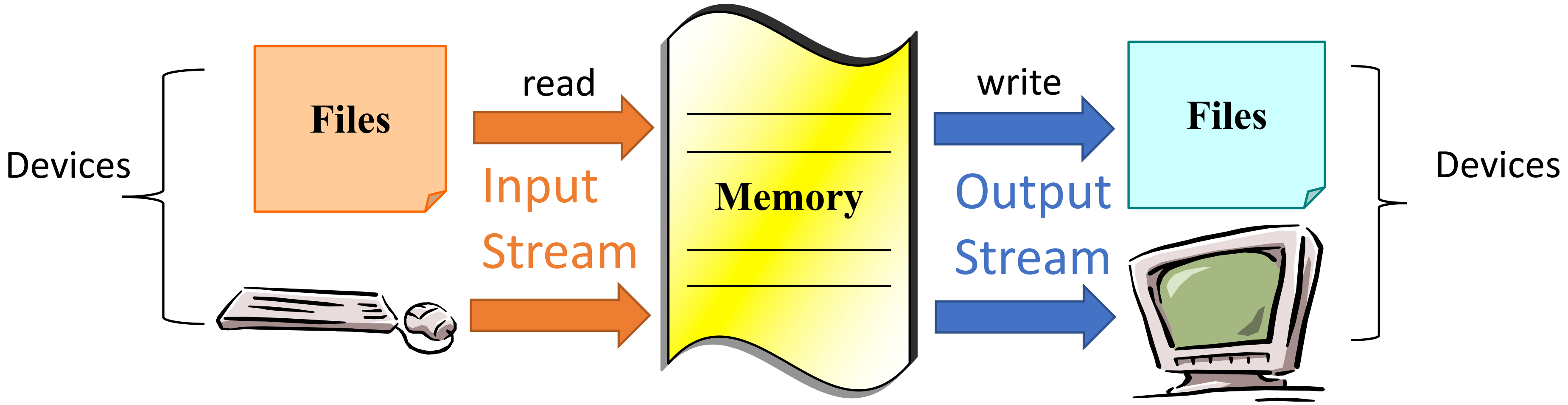
- Input
  - From files to memory
  - From keyboard or mouse to memory
- Output
  - From memory to files
  - From memory to monitors



Files and input/output devices can be generally called as devices sometimes

Input & Output in C Language

- During the input or output, data are transferred between the memory and devices.
- The data transmission is implemented by data input/output stream (数据输入输出流) in C Language



File(文件)

- Files are data that stored in the hard disk of a computer

- File name is the unique identifier that can be used to identify the file that is stored in a computer. The full file name includes the following three parts
  - Directory (路径) : indicate where the file is stored
  - File Name (文件名) : it is given by users
  - Extended File Name (扩展文件名) : indicate the category of a file

```
Full File Name – D:\users\data\file.txt
Directory – D:\users\data
File Name – file
Extended File Name – txt
```

Often Used Extended File Name

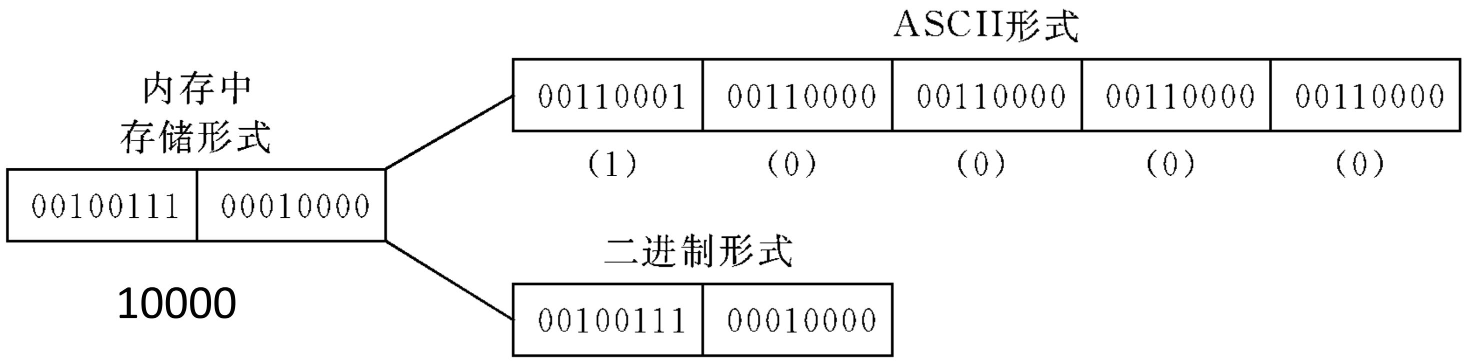
Extended File Name	Description
*.txt	Text files that store text data
*.exe	Executable file in windows OS. It should be noticed that this is not used in Linux or Mac OS
*.c	C Source code file
*.cpp	C plus plus source code file
*.obj	Objective file that is generated from the source code file of C language by compiling the source codes.
*.bmp	One kind of image file. The image is not compressed
*.jpg	One kind of image file. The image is compressed.

Category of Files

- Generally, files can be divided into two categories that are text file (文本文件) and binary file (二进制文件)

- Text File (文本文件)
  - Data are stored as characters that are represented by ASCII codes (We can read it)

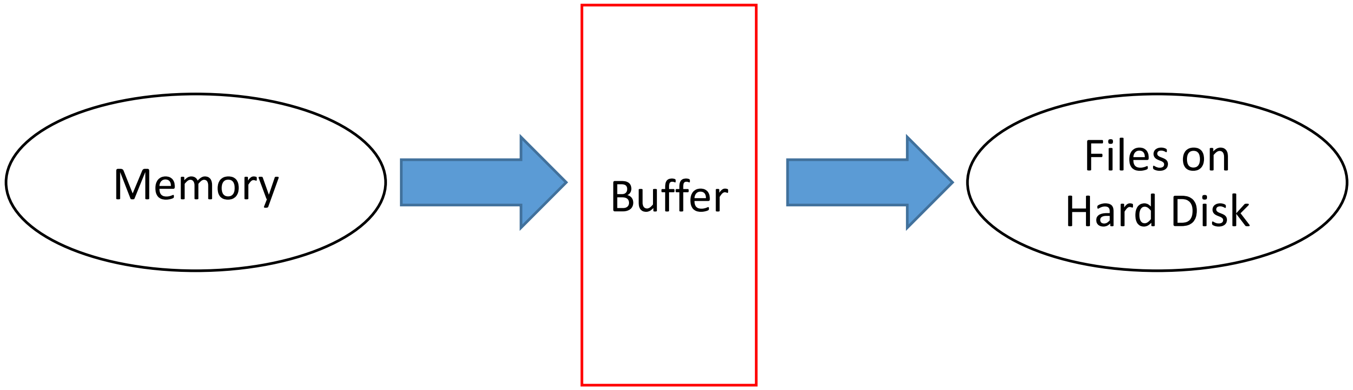
- Binary File (二进制文件)
  - Data are stored as binary numbers (Computers can read it)



Buffer (缓冲区)

- In the data transmission from the memory to a file, Data are not directly transferred from memory to the hard disk. A part of memory called buffer is used in data transmission

- Data are firstly stored in the buffer. When the buffer is filled or users want to finish the data transmission, the data are transferred from the buffer to the hard disk



Pointer of File (FILE*)

- FILE is a structure that is declared in the header file of <stdio.h>. It is used to store information related to files.

- We define a pointer of FILE structure to operate on files, such as opening a file or reading data from a file.

```
#include <stdio.h>

FILE *fp;
```

Steps of File Operation

- Step-1 : Open a File
  - Establish a connection between the file stored on hard disk and the memory
- Step-2 : Read or Write a File
  - Data transmission between a file and the memory
- Step-3 : Close a File
  - Disconnect the file stored on hard disk and the memory

All these steps are accomplished by using standard function that are declared in the head file of <stdio.h>

Open a File

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

- filename – a string that stored the file name
- mode – a string that indicates the mode of opening a file
- Returned Value – return NULL if it is failed, and return a pointer of FILE if successful

The double slash(\\) means a single slash (\). It is required in a string.

Open a text file with a reading mode.
The file name is “d:\data\file.txt”

```
FILE *fp;
fp = fopen(“d:\\data\\file.txt”, “r”);
```

Usually Used Opening Mode

Mode	Description
r	open a text file, and we can only read data from the file
w	open or create a text file, and we can only write data to the file
rb	open a binary file, and we can only read data from the file
wb	open or create a binary file, and we can only write data to the file

Close a File

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

- returned value
  - return 0 if the closing is successful; return EOF(-1) if failed
- If a file is closed, we can not read (write) data from (to) the file

```
FILE *fp;
fp = fopen(“d:\\data\\file.txt”, “r”);
......
fclose(fp);
......
```

Function of Reading and Writing

- Reading or Writing a Character
  - fgetc() and fputc()
- Reading or Writing a String
  - fgets() and fputs()
- Reading or Writing Binary Data
  - fread() and fwrite()
- Formatted Reading or Writing
  - fscanf() and fprintf()

Writing a Character to a File

```
int fputc(int ch, FILE *fp);
```

- ch – a character to be written to the opened file
- fp – pointer of the opened file
- Returned value – return the character that is written to the file if successful, and return EOF(-1) if failed

Reading a Character from a File

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

- fp – pointer of an opened file
- Returned Value – a character that is read from an opened file. If the pointer (fp) reaches the tail of a file, the function returns EOF.

EOF ： End of File

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

int main(int argc, char *argv[])
{
    char str[256];
     FILE *fp1, *fp2;
    int ch, i;
    if ( argc != 2 )
    {
        printf("parameter error!\n");
        printf("Usage : %s [FileName]\n",
                    argv[0]);
        return 1;
    }
    gets(str);

     fp1 = fopen(argv[1], "w");   i = 0;
    while( str[i] != '\0' )
    {
        ch = (int)(str[i]);
        fputc(ch, fp1); i++;
    }
   ......
```

```c
     ........
     fclose(fp1);
     fp2 = fopen(argv[1], "r");
    while( 1 )
    {
        ch = fgetc(fp2);
        if (ch == EOF) break;
        putchar((char)ch);
    }
    putchar('\n');
    fclose(fp2);
    return 0;
}
```

Writing a String to a File

```
int fputs(char *string, FILE *fp);
```

- string – a pointer to a string that would be written to a file
- fp – pointer of a opened file
- Returned value – return a non-zero value if successful, and return EOF(-1) if failed

Reading a String from a File

```
char* fgets(char *string, int n, FILE *fp);
```

- string – a pointer that points to where the read string are stored.
- n – the maximal number of characters that are read from the file
- fp – a character to be written to a file
- Returned Value – a pointer to the string that is read from the file

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

int main(int argc, char *argv[])
{
    char str1[256], str2[256];
    FILE *fp1, *fp2;

    if ( argc != 2 )
    {
        printf("parameter error!\n");
        printf("Usage : %s [FileName]\n",
                argv[0]);
        return 1;
    }
    gets(str1);

    fp1 = fopen(argv[1], "w");
    fputs(str1, fp1);
    fclose(fp1);
    ......
```

```c
    ........

    fp2 = fopen(argv[1], "r");
    fgets(str2, 256, fp2);
    printf("%s\n", str2);
    fclose(fp2);
    return 0;
}
```

Writing Binary Data to a File

```
int fwrite(void *p, int size, int n, FILE *fp);
```

- p – the initial address of the data that would be written to an opened file
- size – the size of elements of the data
- n – the number of elements of the data
- fp – the pointer of an open file
- return value – return the number of elements that is written to the file successfully (the value would be less than *n,* if there are errors in the writing)

Reading Binary Data to a File

```
int fread(void *p, int size, int n, FILE *fp);
```

- p – the initial address of the data that would be read from an opened file
- size – the size of elements of the data
- n – the number of elements of the data
- fp – the pointer of an open file
- return value – return the number of elements that is read from the file successfully (the value would be less than *n,* if there are errors in the reading)

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

#define NUM 3
struct student
{
    char name[15];
    float score;
}s[NUM] = { {"Jack", 84.5f},
            {"Tom", 92.1f},
            {"Alen", 82.0f} };

int main(int argc, char *argv[])
{
    struct student *pData;
    FILE *fp1, *fp2;
    int i;
    if ( argc != 2 )
    {
        printf("parameter error!\n");
        printf("Usage : %s [FileName]\n", argv[0]);
        return 1;
    }
    ......
    ........

    fp1 = fopen(argv[1], "wb");
    fwrite(s, sizeof(struct student), NUM, fp1);
    fclose(fp1);

    pData = (struct student*)malloc(sizeof(struct student)*NUM);
    fp2 = fopen(argv[1], "rb");
    fread(pData, sizeof(struct student), NUM, fp2);
    fclose(fp2);

    for ( i = 0 ; i < NUM ; i++ )
    {
        printf("%s %f\n", pData[i].name, pData[i].score);
    }

    free(pData);

    return 0;

}
```

Formatted Reading and Writing

```
fscanf(FILE *fp, formatted string, variable lists);
fprintf(FILE *fp, formatted string, variable lists);
```

- The usage of fscanf() and fprintf() is similar to scanf() and printf()
- fp – the pointer of an open file
- formatted string – a string that defines the rule of formatted input and output
- variable lists – variables of the data that are read (write) to (from) the file.

```
int a = 5; float b = 3.6f;
fprintf(fp1, “%d %f”, a, b); //write data to a file
......
fscanf(fp2, “%d %f”, &a, &b); //read data from a file
```