Foundations of C Programming (Structured Programming)
- File

Outline

- Run arguments
- File reading
- File writing

Data Input

- Data input from keyboard
  - scanf
  - gets
  - command line parameters

Input from Command Line Parameters

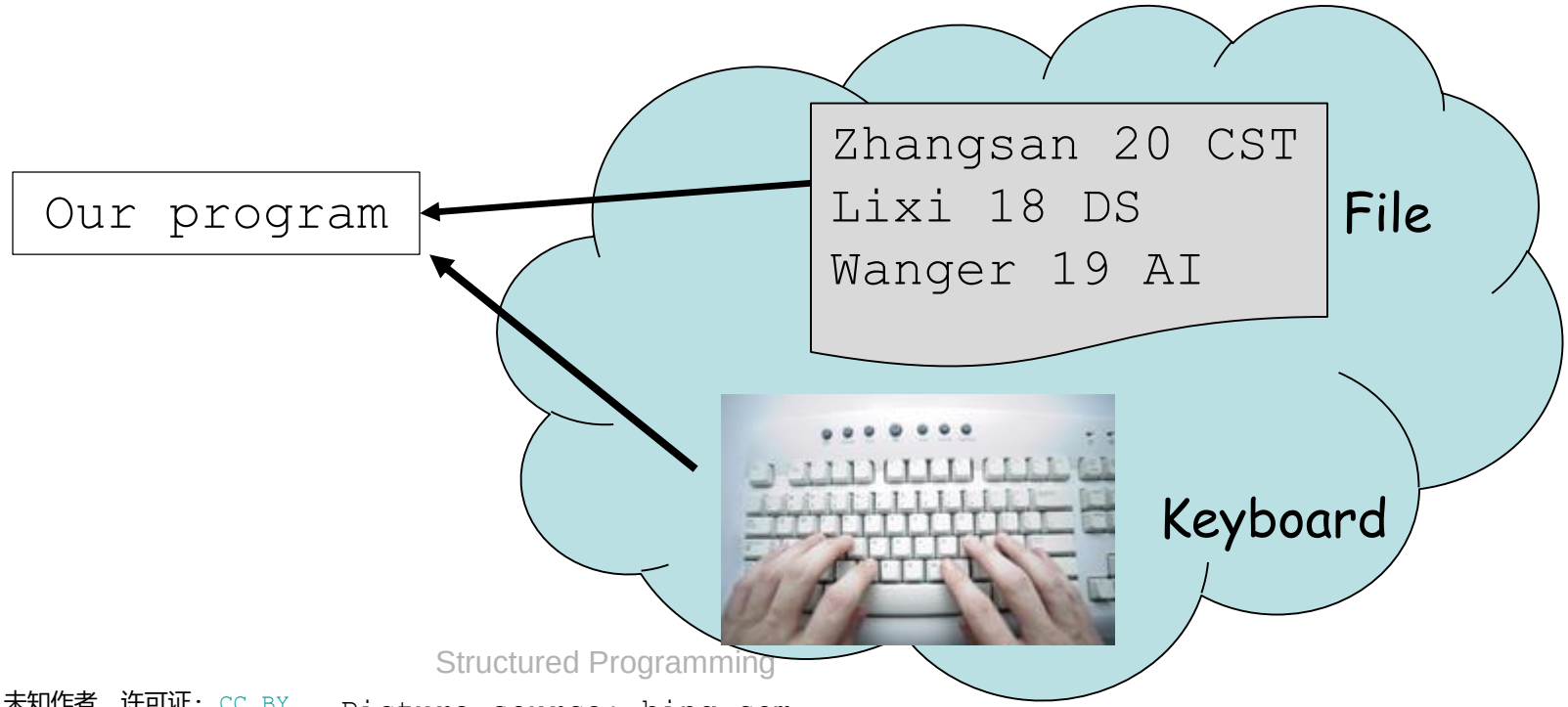
the number of arguments passed

each element points to an argument (参数)

```
#include <stdio.h>
int main(int argc, char *argv[]) {
   int i;
   printf("There are %d command line parameters. They
   are:\n", argc);
   for (i = 0; i < argc; i++)
       printf("%s\n", argv[i]);
   return 0;
}
```

Limitation and Solution

- Use keyboard to input at the execution, we can only input a small amount of data. And we must input for each execution.
- For large volume of data, we need to use File I/O. This is also another source of data.



Picture source: bing.com

Basic File Operations

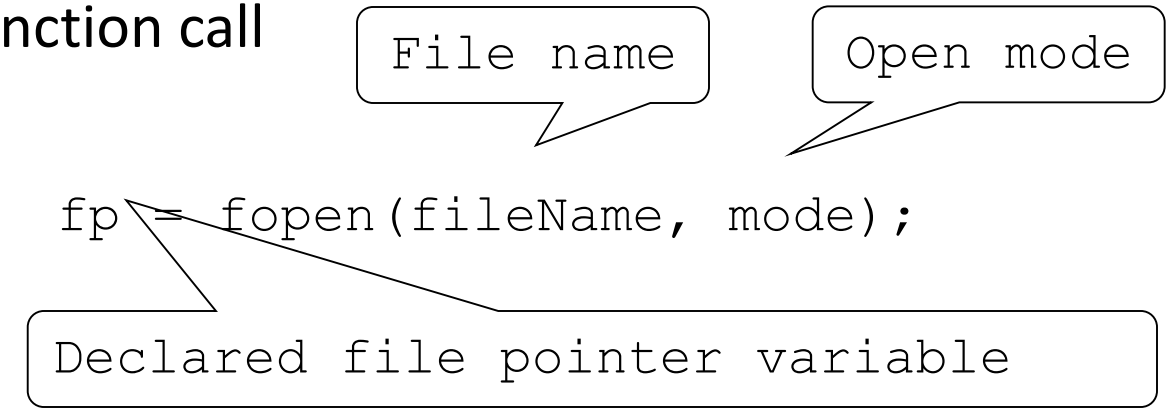
- Open a file （打开文件）
- Sequential access （顺序存取）
  - Read data from a file
  - Write data to a file
- Random access （随机存取）
- Close a file （关闭文件）

Declare a File Pointer

- Format
  - `FILE *fp;`
- declares a pointer variable fp that points to FILE type.
- reading or writing files is through the defined pointer variables (文件指针).

Open a File: fopen

- Prototype: `FILE* fopen(char *fileName, char *mode);`
- Function call



- If the file fails to open, `fopen` returns `NULL` (NULL means the pointer does not refer to a valid object)
- If the file opens successfully, it will return a file pointer to the file object.
- To use this function, must have #include <stdio.h>

Open a File - mode

```
fp = fopen(fileName, mode);
```

"r" ("rb")	Open text (binary) file for reading. The file must exist.
"w" ("wb")	Create an empty text (binary) file for writing. If a file with the same name already exists, its content is erased (擦除) and the file is treated as a new empty file.
"a"(“ab")	Append to a text (binary) file. Append (附加) data at the end of the file. The file is created if it does not exist.
"r+"(“r+b")	Open a text (binary) file for update both reading and writing. The file must exist.
"w+"("w+b")	Create an empty text (binary) file for both reading and writing. If a file with the same name already exists, its content is erased and the file is treated as a new empty file.
"a+"(“a+b")	Open text (binary) file for reading and appending.

An Example

Format: fp = fopen(fileName, mode);

```
FILE *myFile;
myFile = fopen("data.txt", "r");
if (myFile == NULL) {
    // Here print some warning messages
    return;
}
……
```

Close a File: fclose

- Prototype: `int fclose(FILE* filePointer);`
- Function call

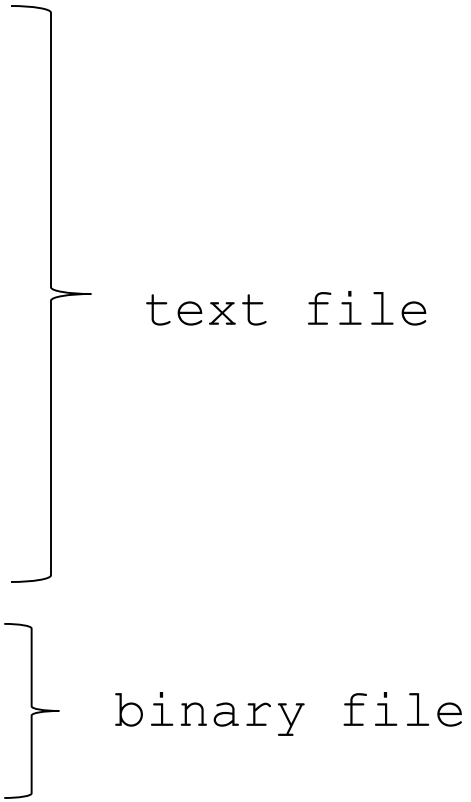
```
fclose(fp);
```

- close the file associated with pointer variable `fp`.
- e.g.,

```
FILE *myFile;
myFile = fopen("data.txt", "r");
...
fclose(myFile);
```

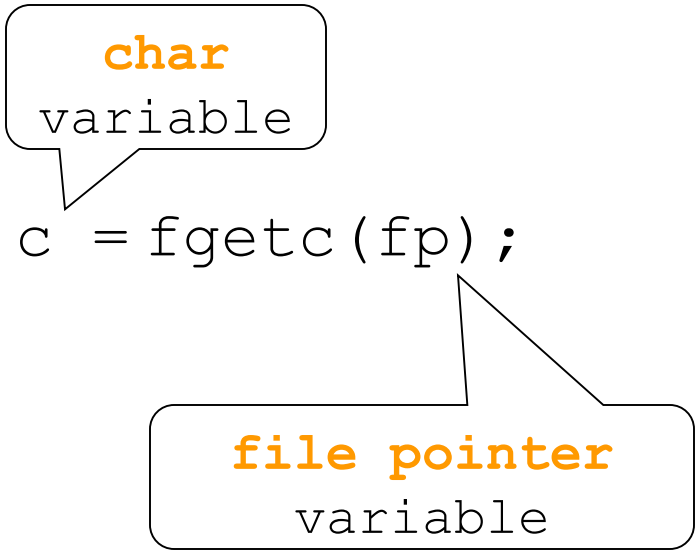
Read From/Write To A File

- Functions (`#include <stdio.h>`)
  - `fgetc()`
  - `fputc()`
  - `fgets()`
  - `fputs()`
  - `fscanf()`
  - `fprintf()`
  - `fread()`
  - `fwrite()`
  - `fseek()`



fgetc

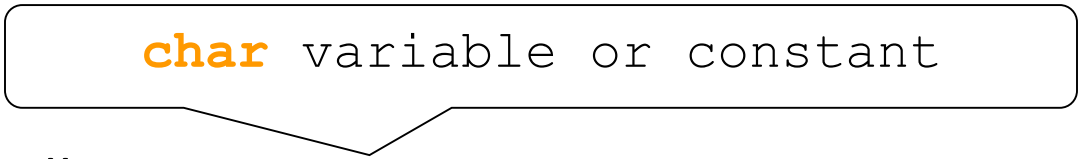
- Prototype: `int fgetc(FILE* filePointer);`



- reads a character from the file associated with `fp`
- if `fp` reaches the end of the file, the character is `EOF` (means the end of file)

fputc

- Prototype: `int fputc(char c, FILE* filePointer);`



- Function call: `fputc(c, fp);`

- write `c`'s value into the file associated with `fp`

An Example

infile.txt

```
bcf
ac
```

```
FILE *fp;
char c;
fp = fopen("infile.txt", "r");
if (fp == NULL) {
  printf("The file does not exist");
  return;
}
while((c = fgetc(fp)) != EOF)
  printf("%c", c);
fclose(fp);
```

Class Exercise

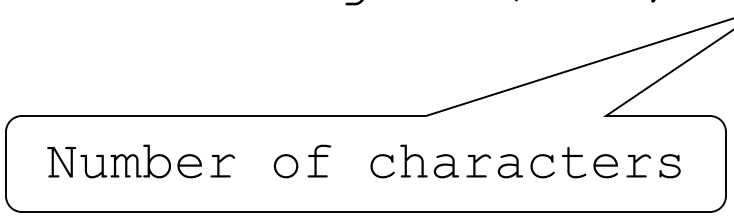
```
int main()
{
    FILE *fp;
    char c;
    fp = fopen("infile.txt", 'r');
    while((c = fgetc(fp)) != NULL)
      printf("%c", c);
    return 0;
}
```

What are the problems with this program? How to revise?

fgets

- Prototype: `char* fgets(char* str, int size, FILE* filePointer);`

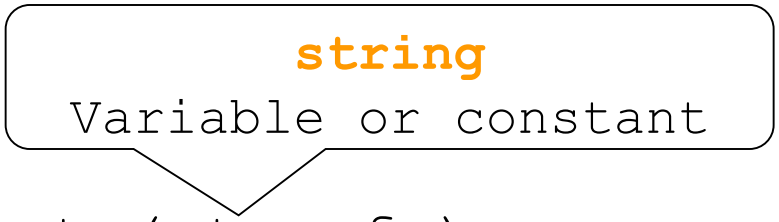
- Function call: `fgets(str, size, fp);`



- read a string to `str` with the length `size` or a line from the file associated with the `fp`
- when reach the end of the file, return `NULL`

fputs

- Prototype: `int fputs(char* str, FILE *filePointer);`



- Function call: `fputs(str, fp);`

- write the string into the file associated with `fp`

An Example

```
FILE *src, *dst;
char str[256];
src = fopen("infile.txt", "r");
dst = fopen("outfile.txt", "w");
if (src == NULL || dst == NULL)
  return;
while(fgets(str, 256, src ) != NULL )
  fputs(str, dst );
fclose( src );
fclose( dst );
```

infile.txt

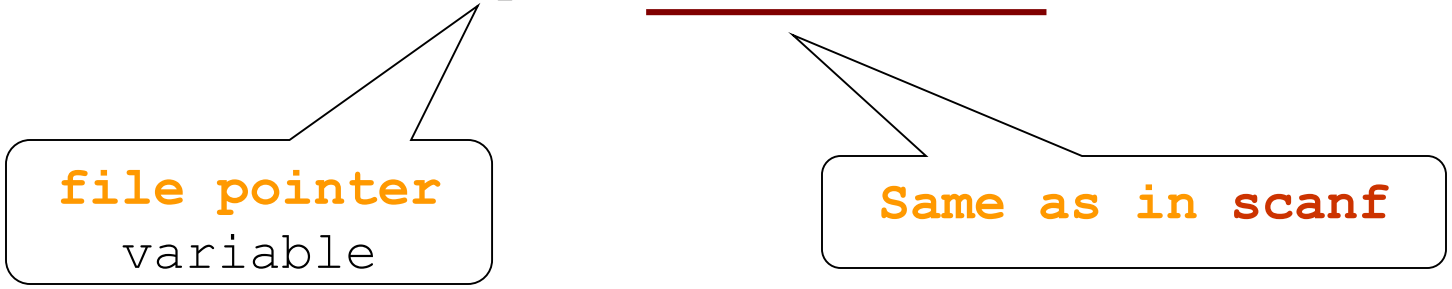
```
Hello, FOC
UICer
```

outfile.txt

fscanf

- Prototype: `int fscanf(FILE *filePointer, const char *format, ...);`

- Function call: `fscanf(fp, format, ...);`



- read data in the designated format from a file associated with `fp` like `scanf` from keyboard
- when reach the end of the file, return `EOF`

An Example

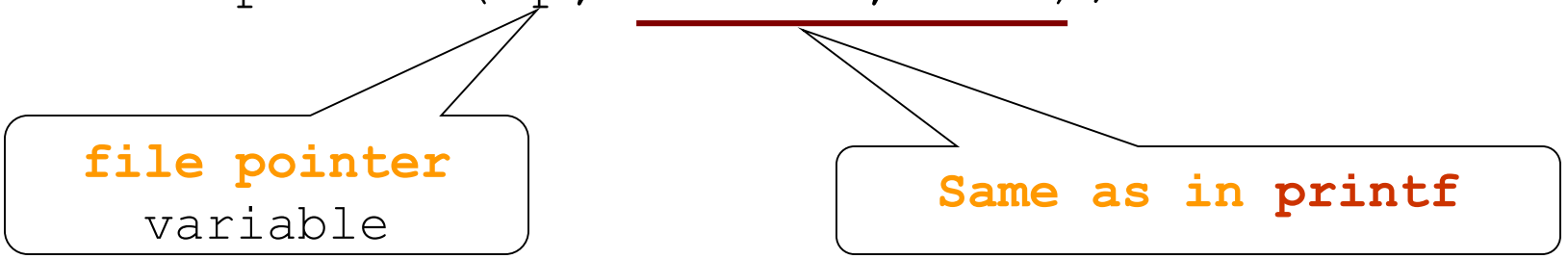
infile.txt

```
Jerry 20
Jim 10
Tony 12
```

```
FILE *fp;
char stuName[20];
int stuID;
fp = fopen("infile.txt", "r");
if (fp == NULL) return;
while(fscanf(fp, "%s %d",stuName,&stuID)!= EOF)
  printf("%s %d\n", stuName, stuID);
fclose(fp);
return 0;
```

fprintf

- Prototype: `int fprintf(FILE *filePointer, const char *format, ...);`
- Format: `fprintf(fp, format, ...);`



- Write data in the designated format to a file associated with `fp` like `printf` to monitor.

An Example

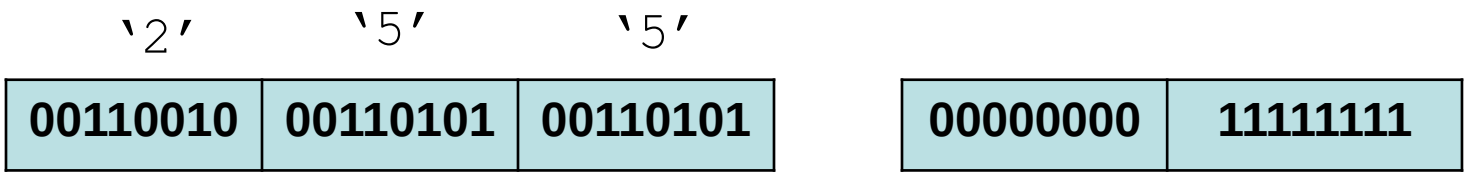
```
fp = fopen("test.txt", "w");
if (fp == NULL) {
    printf("Error: can't create file.\n");
    return 1;
}
else{
  int i = 2022;
  char str[100] = "UICer";
  fprintf(fp, "Hello %s, %d", str, i);
}
fclose(fp);
```

```
Hello UICer, 2022
```

test.txt

Text Content and Binary Content

- All file content is in binary form (0s and 1s).
  - If a file primarily uses the binary codes for characters (for instance, ASCII or Unicode) to represent text, then it is a text file; it has text content.
  - If the binary values in the file represent machine-language code or numeric data or image or music encoding, the content is binary
  - For example: `short int num = 255`



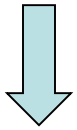
text file
(文本文件)

binary file
(二进制文件)

Text Content and Binary Content

```
short int num = 255;
```

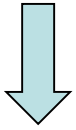
```
fprintf(fp, "%d", num);
```



00110010	00110101	00110101

text file

```
fwrite(&num, sizeof(short int), 1, fp);
```



00000000	11111111

binary file

`fread` Function

- Used for binary file reading
- Format
  - size_t fread(void * object_ptr, size_t type_size, size_t num_objects, FILE *fp);
  - Read `type_size*num_objects` bytes starting from the place pointed by `fp`
  - Examples

```
double buffer[256];
fread(buffer, sizeof(double), 256, fp);
```

Read `sizeof(double)* 256` bytes starting from the place pointed by `fp` and assign the data to the memory pointed by `buffer`.

size_t: unsigned integer

`fwrite` Function

- Used for binary file writing
- Format
  - size_t fwrite(void * object_ptr, size_t type_size, size_t num_objects, FILE *fp);
    - Write `type_size*num_objects` bytes starting from the place pointed by fp
  - Examples

```
double buffer[256];
fwrite(buffer, sizeof(double), 256, fp);
```

Write `sizeof(double)* 256` bytes starting from the memory (內存) pointed by buffer to the place pointed by fp.

size_t: unsigned integer

`fseek` Function

- Usually file is read or written sequentially.
  - After one read/write is finished, `fp` automatically moves to the next place to read/write
- Format
  - `int fseek(FILE *stream, long int offset, int whence);`
  - In a binary stream, `fp` points to the new position, measured in bytes (以字节为单位) from the beginning of the file, is obtained by adding `offset` to the position specified by `whence`.

`fseek` Function

- Examples
  - `fseek(fp, 0L, SEEK_SET); // go to the beginning of the file`
  - `fseek(fp, 10L, SEEK_SET); // go 10 bytes into the file`
  - `fseek(fp, 2L, SEEK_CUR); // advance 2 bytes from the current position`
  - `fseek(fp, 0L, SEEK_END); // go to the end of the file`
  - `fseek(fp, -10L, SEEK_END); // back up 10 bytes from the end of the file`

More Functions

- After class, learn more file functions by yourself
- For example
  - `ftell`
    - returns the current file position of the given stream.
  - `rewind`
    - sets the file position indicator to the beginning of the file
  - `clearerr`
    - clears the end-of-file and error indicators

Summary

- Introduced how to read and write a file
- File handling is very important in information handling
- File stream pointer is used in reading and writing