

CPT109: C Programming & Software Engineering I

Lecture 8: Files & Command Line Arguments

Soon Phei Tin

Office: SD531

Email: soon.tin@xjtlu.edu.cn

Outline of Today's Lecture (8)

- Files – what are they?
- File operations:
 - Opening or creating a file
 - Writing to a file
 - Reading from a file
 - Accessing specific data (moving in the file)
 - Save/Read a structure to/from a file
 - Closing a file
- Command Line arguments

What is a Computer File?

- A file is a **named** unit of storage.
- It is a block of arbitrary information or resource for storing information which is available to a computer program.
- Files are stored on disk, DVD, USB etc.
- Computer files are the modern equivalent of paper documents.

File Structure

- A file occupies a sequential block of memory on the media where it is stored (like an array).
- Files end with an EOF (End-of-File) marker or at a byte number specified by the operating system.
- There are two standard views of a file:
 - Binary
 - Text
- A file can be opened in either mode.

Binary vs. Text Mode/View

Binary Mode

- Each byte stored in the file can be accessed

Text Mode

- Bytes are interpreted as characters

Generally

- Reading stops at the occurrence of EOF even if there is more data
- Escape sequences (e.g. `\n` newline, `\r` carriage return) are seen as `\n`

Declaring a File

- The standard input output library `<stdio.h>` contains several file manipulation functions
- Declaring a file in C:

```
FILE *my_file
```

- Declares that `my_file` is a pointer to a `FILE`.
- Each file must have a unique `FILE` pointer.
- `my_file` is an internal file name used by the C program referring to the external file name which is the actual physical file stored on disk.

Interacting with a File

- The file pointer contains several items of information. The important ones here are:
 - The address of the start of the file.
 - The current location indicated by the byte count
 - Type of file access
- When interacting with the file, the byte count keeps track of the byte number within the file where you are working.

Standard Files

Actually, the operating system uses files to handle ALL inputs and outputs.

Three files are automatically opened when a program starts to run (pointers are generated).

The files are:

File	Pointer name
standard input	stdin
standard output	stdout
standard error	stderr

These are all defined in `stdio.h`

Opening a file; fopen() (1/2)

Before file data can be processed, the file must be either **opened** or **created**.

Syntax for fopen():

```
Internal_filename = fopen(external_filename, openmode);
```

The function fopen() takes 2 arguments:

filename

filemode

Opening a file; fopen() (2/2)

Example:

```
thefile = fopen("my_file.txt", "r");
```

This statement will attempt to open a file called "my_file.txt"

If "my_file.txt" is located in a different directory than the current program directory. The full path needs to be included. For example

```
"c:\\documents\\my_file.txt"
```

The function fopen() returns a pointer to the opened file, otherwise NULL is returned.

Open Modes for fopen()

Mode	Description	starts..
r	open for reading (The file must exist)	beginning
w	open for writing (creates file if it doesn't exist). Deletes content and overwrites the file.	beginning
a	open for appending (creates file if it doesn't exist)	end
r+	open for reading and writing (The file must exist)	beginning
w+	open for reading and writing. If file exists deletes content and overwrites the file, otherwise creates an empty new file	beginning
a+	open for reading and writing (append if file exists)	end

- add a "b" to the end if you want to use binary files instead of text files, like this: "rb", "wb", "ab", "r+b", "w+b", "a+b".

fopen() Failure

If fopen() returns a NULL pointer, the operation failed.

This may be due to:

- Opening a non-existing file.
- Opening a file for reading without access rights.
- Opening a file for writing when no disk space is available.

Hence programs should be written to handle this failure.

Testing for failure

Check the value of the pointer returned by `fopen()`.

```
FILE *fptr;          /*file pointer*/  
fptr = fopen("hello.txt","r");  
if(fptr == NULL){  
    printf("Failed opening Hello.txt");  
    exit(1);         /*exits program*/  
}
```

Closing the File fclose()

The function fclose() can be used to close the file:

- If the file is successfully closed **return** 0
- On failure, **return** EOF (End of File)

Syntax

```
fclose(fptr);
```

Example program fclose()

```
#include<stdio.h>
main(){
FILE *pfile;
pfile = fopen("hello.txt","w+");
if(!pfile){
    printf("Cannot open file\n");
    exit(1);}
if(fclose(pfile))
    printf("Error closing file\n");
return 0;
}
```

ftell()

ftell () is a function that returns the current file position (number of bytes) from the beginning of the file in the type of a **long int**:

```
FILE *pfile;  
long p1;  
pfile = fopen("hello.txt","w");  
p1 = ftell(pfile);  
printf("Byte count is %ld", p1);
```

What will be printed on the screen?

fgetc()

Once a file is open data can be read and wrote
fgetc() is a function that would read a character from a file.

Syntax

```
ch = fgetc(internal_filename);
```

ch – is a character variable **char**

Internal_filename – is the pointer to the open file

fputc()

fputc() is a function to write a character to a file

Syntax:

```
fputc(ch, internal_filename);
```

```
fputc('a', internal_filename);
```

ch – is a character variable **char**

Internal_filename – is the pointer to the open file

Example

```
FILE *pfile;  
char ch = 'A';  
long p1, p2;  
pfile = fopen("hello.txt","w");  
p1 = ftell(pfile);  
fputc(ch, pfile);  
p2 = ftell(pfile);  
printf("1st position %ld, 2nd position %ld\n", p1, p2);  
rewind(pfile); /*rewind moves the pointer to start of file*/  
p1 = ftell(pfile);  
ch = fgetc(pfile);  
p2 = ftell(pfile);  
printf("1st position %ld, 2nd position %ld", p1, p2);
```

What will be
printed on the
screen?

fscanf()

fscanf() is a function for reading a value or values of specified data types from a file.

Syntax:

```
fscanf(internal_filename, format_specifier(s), variable list);
```

Example:

```
FILE *fptr;  
char name[50];  
int age;  
fptr = fopen("filename.txt", "r");  
fscanf(fptr, "%s %d", name, &age);
```

Why does
name not
have an &?

fprintf()

fprintf() is a function to print values to a file

Syntax:

```
fprintf(internal_filename, format_specifier(s), variable_expression);
```

Examples

```
fprintf(fp_ptr, "This is an example");
```

```
char name[] = "Peter";  
fprintf(fp_ptr, "%s", name);
```

```
char name[] = "Mary";  
fprintf(fp_ptr, "%s %s", "My name is", name);
```

Example program fprintf()

```
FILE *pfile;           /*file pointer pfile*/
char text[20];
pfile = fopen("hello.txt","w+");    /*new file for r&w*/
fprintf(pfile, "HELLO");
printf("%ld", ftell(pfile));
rewind(pfile);          /*rewind moves the pointer to*/
                        /* the start of the file*/
fscanf(pfile, "%s", text);
printf("%s\n", text);
```

What will be printed on the screen?

fseek()

fseek() is a function that allows you to be able to access any particular position inside a file by specifying the amount in bytes to move from the starting_point which can be:

SEEK_SET – start of the file

SEEK_CUR – current position in the file

SEEK_END – end of the file

```
fseek(pfile, offset, starting_point);
```

Example program fseek()

```
FILE *pfile;           /*file pointer pfile*/
char ch;
pfile = fopen("alpha.txt","r+");
fseek(pfile, 0L, SEEK_END);      /*offset is type long*/
printf("Byte count is %ld\n", ftell(pfile));
fseek(pfile, 6L, SEEK_SET);
ch=fgetc(pfile);
printf("The character read was %c",ch);
fseek(pfile, 0L, SEEK_CUR);
fputc('Z',pfile);
fclose(pfile);
}
```

What will be
printed on
the screen?

Read/Save Structure Contents in a File Using fread()/fwrite() (1/2)

- **struct**'s can hold many different variables
- Important tools for constructing databases. Needs to be saved and retrieved from, a file.
- Use fprintf() and fscanf() to read and write **struct** members.
- Use fread() and fwrite() to read and write structure sized units of data.

Read/Save Structure Contents in a File Using fread()/fwrite() (2/2)

Example data write:

pbooks is a file pointer

primer is a variable of type **struct** book

```
fwrite(&primer, sizeof(struct book), 1, pbooks);
```

The fread() function has the same arguments.

Copying a data block of size **struct** book into primer

Note: fread()/fwrite() are for binary files. Hence, fopen should use binary form.

Text and Binary

```
fscanf(fp_trp, "%s%lf%d%lf%lf",  
        planet.name,  
        &planet.diameter,  
        &planet.moons,  
        &planet.orbit,  
        &planet.rotation);
```

```
fprintf (fp_tr, "%s%lf%d%lf%lf",  
        planet.name,  
        planet.diameter,  
        planet.moons,  
        planet.orbit,  
        planet.rotation);
```

```
fread(&planet,  
        sizeof(planet), 1,  
        fp_tr);
```

```
fwrite(&planet,  
        sizeof(planet), 1,  
        fp_trp);
```

EOF (End of File)

The EOF symbol is part of `stdio.h`, it marks the end of the file.

The function `feof()` can be used to check when the EOF is reached.

`feof()` returns non-zero if it tries to read a character but finds the EOF marker

```
if(feof(pfile))  
    printf("You've reached the end of the file");
```

Example program EOF test

```
FILE *fptr;  
char ch = 'A';  
  
fptr = fopen("hello.txt","r");  
  
while(!feof(fptr))  \*!=EOF*\n    fscanf(fptr, "%c", &ch);
```

Can we use the EOF check with a binary file????

Quiz

1.State whether the following statements are **true** or **false**

- A file must be opened before it can be used.
- All file must be explicitly closed.
- Files are always referred to by name in C programs.
- Function fseek may be used to seek from the beginning of the file only.
- Using fseek to position a file beyond the end of the file is an error.

Interacting with Files (1/7)

How your program interacts with data stored in a file depends on the types of functions you want to perform:

e.g. Sorting, Searching, Deleting, Adding

Decisions should be made on this before starting to program.

Interaction style will affect speed and memory use.

Interacting with Files (2/7)

Interaction Options:

- Copy all data from file to memory
- Copy only required data from file to memory

Considerations:

- Accessing data in a file from HDD takes longer than accessing RAM/Processor memory
- For large data files, large memory requirements

Interacting with Files (3/7)

Example – Contact List

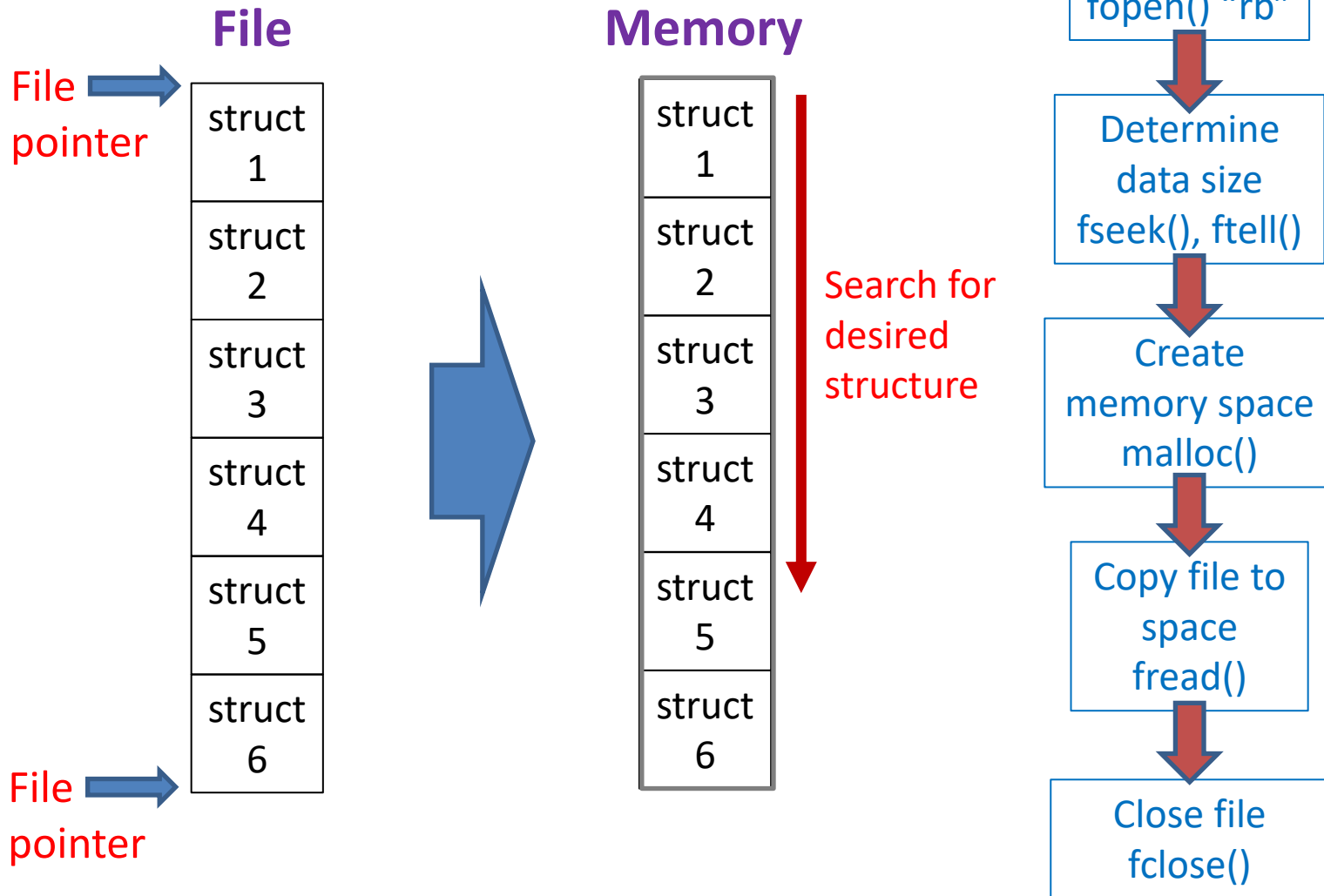
Considerations:

- How often do you need to access it
- How often do you need to make changes to it

Say you want to perform operations (view, edit etc) with a particular contact...

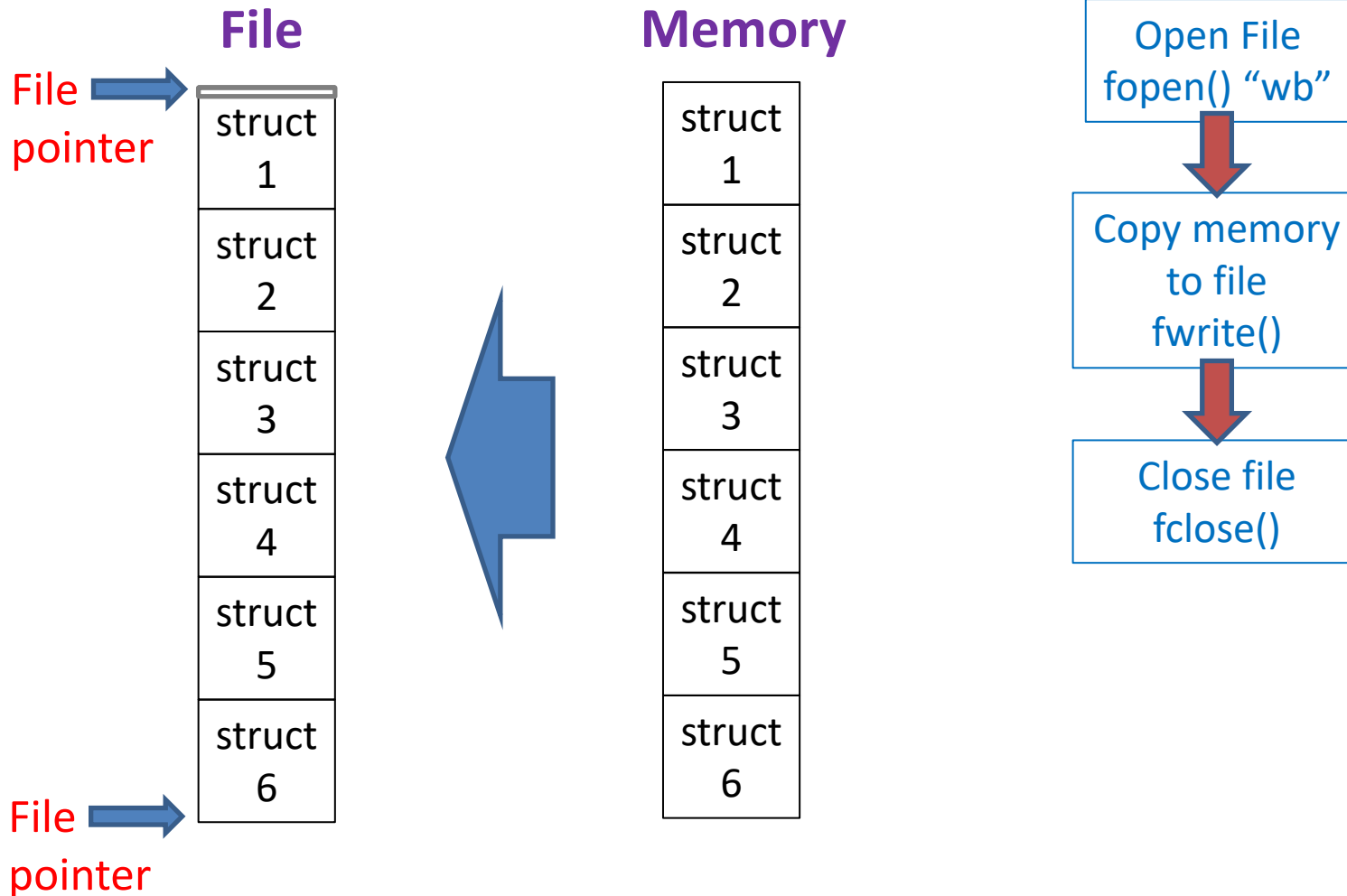
Interacting with Files (4/7)

Copy file:



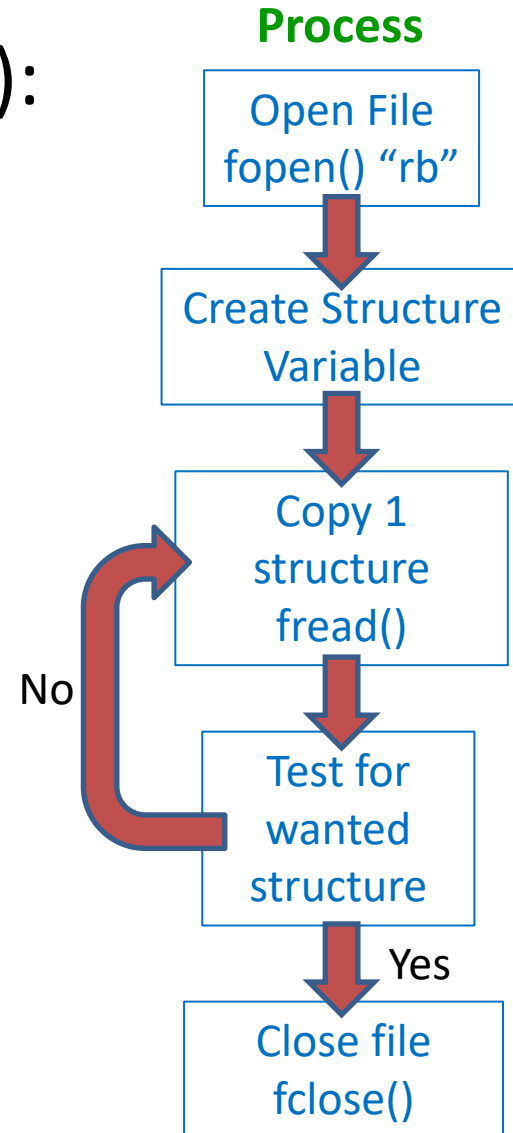
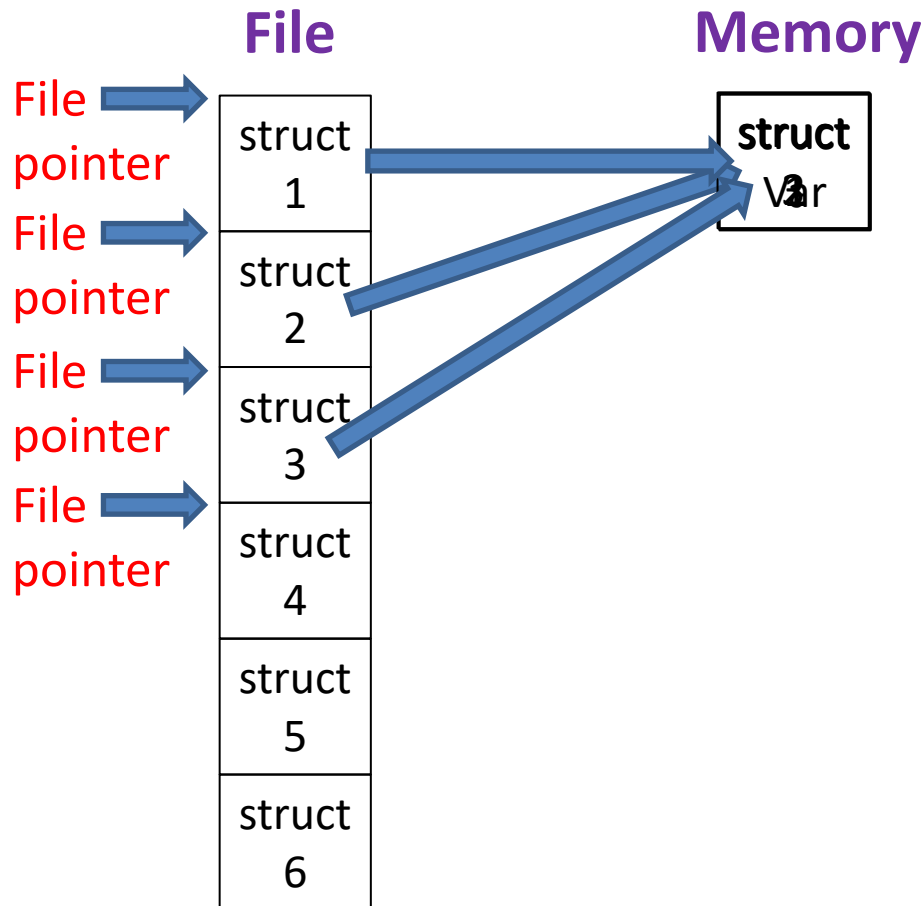
Interacting with Files (5/7)

Save to file:



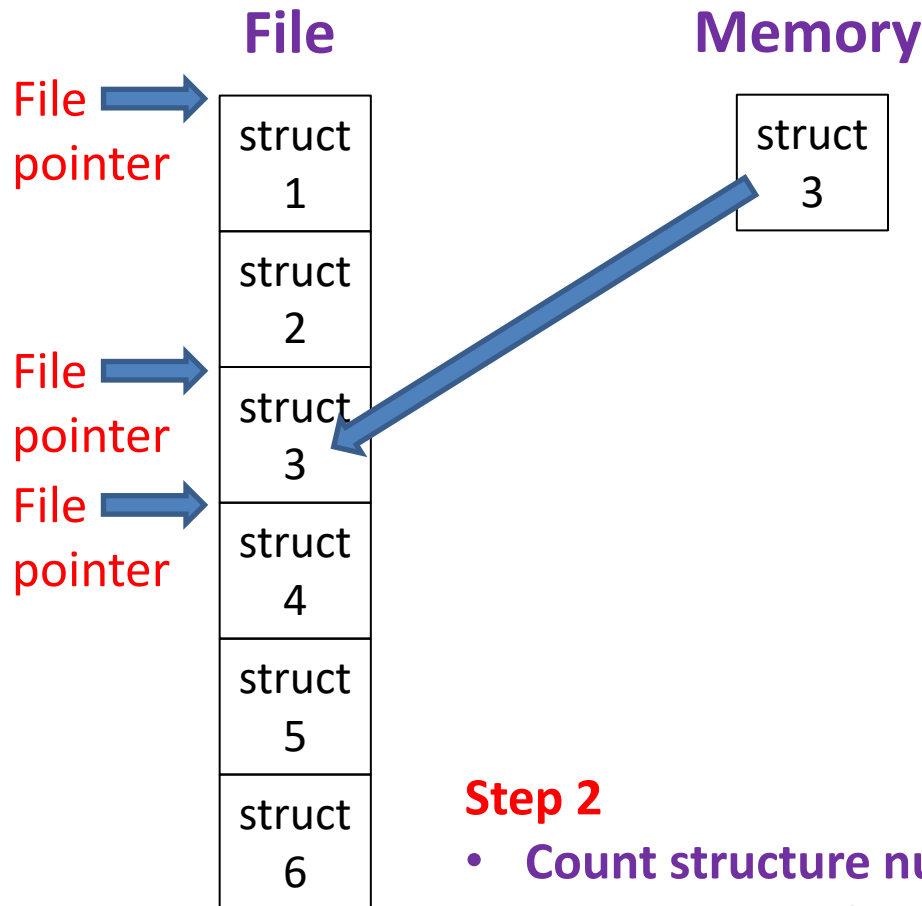
Interacting with Files (6/7)

Find correct structure (struct 3):

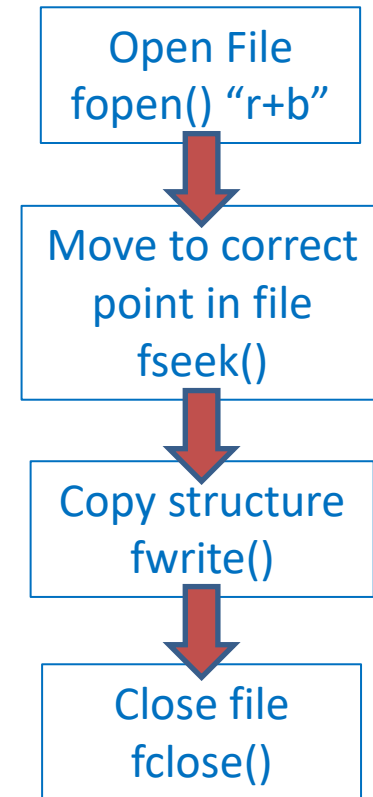


Interacting with Files (7/7)

Save correct structure (struct 3):



Process



Step 2

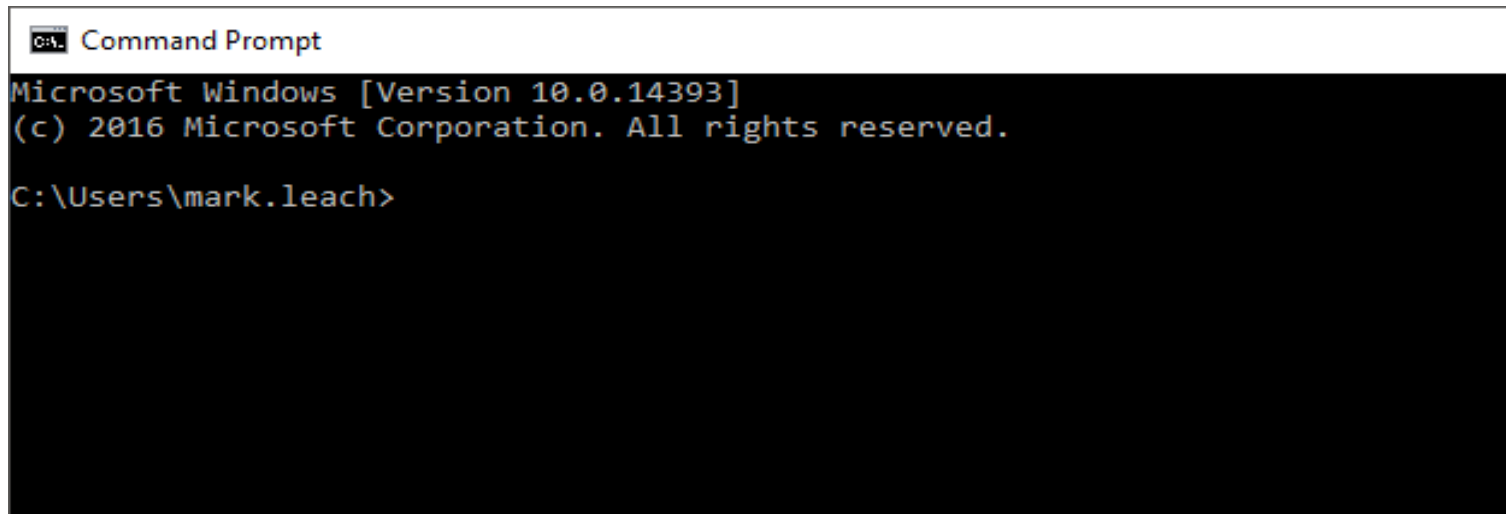
- Count structure number during initial read
- Re-search the file to find correct structure for replace



Command Line Arguments

Command-line Arguments (1/4)

We sometimes want to pass arguments into a program (i.e. into function main) when it begins executing i.e. from the command-line.



```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Users\mark.leach>
```

Command-line Arguments (2/4)

Example

We want run a program called `read_file` to:

- Read a number of lines `1000`
- From a specific file `student_record.txt`

Type on the command line:

```
read_file student_file 1000
```

How does this work?

Command-line Arguments (3/4)

In C, when main is called it has 2 arguments. (These can be ignored if you are not using them.)

```
main(int argc, char *argv[])
```

argc is the **int** number of command line arguments.

argv is an array of pointers to strings where the arguments are stored

Notes:

argv[0] is the name of the program and so **argc** is always at least 1.

Also argv[argc] = NULL

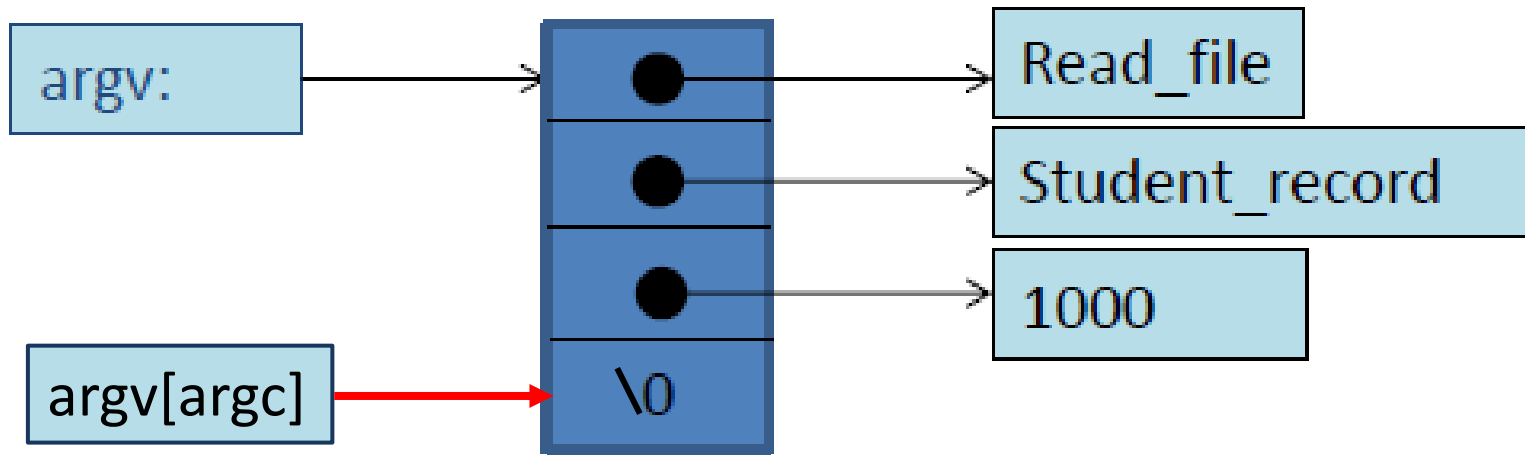
Command-line Arguments (3/3)

In the example:

```
read_file student_file 1000
```

What does argc equal?

argc=3





Questions?

Week 9 is over...almost finished

