CMSC 21
Fundamentals of Programming

2nd Semester 2011-2012

File Input/Output in C

FILES

Header Files

- stdio.h
  - Contains the FILE data type
  - Contains EOF macro
  - Contains the definitions of file input-output functions

FILE Pointers

- In order to access a file, a file pointer must be declared.
- Use the FILE data type defined in stdio.h

```
FILE *<variable_name>;
```

Opening a File

- `fopen ("<filename>", "<mode>");`
  - Parameters:
    - filename – a string that specifies the file to be read
    - mode – a string that specifies the action to be made on the file specified by the filename
  - Return values:
    - FILE pointer if opening of the file is successful
    - NULL if opening the file fails

Opening a File

- Example:

```
int main () {
  //file pointer
  FILE *fp;
  //open the file for reading
  fp = fopen ("sample.txt", "r");
  …
}
```

Closing a File

- fclose
  - Closes a file, making it unavailable for reading and writing
  - Usage:

```
fclose (<file_pointer>);
```

  - Parameters:
    - file_pointer – the pointer to the opened file

Closing a File

- Example:

```
int main () {
  //file pointer
  FILE *fp;
  //open the file for reading
  fp = fopen (“sample.txt”, “r”);
  …
  fclose (fp);
}
```

File I/O Modes

Mode	Description
"r"	"read" The file is opened for reading only
"w"	"write" The file is opened for writing only Every time the file is opened, the previous data stored in the file is erased
"a"	"append" Same a writing except that the previous data stored in the file remains untouched

File I/O Modes

Mode	Description
"r+"	"read + write" The file is opened for reading and writing Contents of the file are not lost
"w+"	"write + read" The file is opened for reading and writing Every time the file is opened, the previous data stored in the file is erased
"a+"	"read + append" Contents of the file are not lost

File I/O Modes

Mode	r	w	a	r+	w+	a+
Open state	Read	Write	Read	Write	Write	Write
Read allowed	Yes	No	No	Yes	Yes	Yes
Write allowed	No	Yes	Yes	Yes	Yes	Yes
Append allowed	No	No	Yes	No	No	Yes
File must exist	Yes	No	No	Yes	No	No
Contents of existing file lost	No	Yes	No	No	Yes	No

File I/O Modes

- Adding a “t” or a “b” to the mode allows the user to specify the type of file being opened
- Types of files:
  - Text files
  - Binary files
- Examples:
  - `“rt”, “rb”, “wb”, “wt”, “rt+”, “ab+”, “r+b”`

Text Files

- Data are stored using only characters
- Non-character data types are converted into a sequence of characters before being stored in the file
- Data from a text file are read as a sequence of characters and are converted into the correct internal formats before being stored in the memory

Binary Files

- A collection of data stored in the internal format of the computer
- Data do not need to be reformatted as they are read or written
- Data are stored in the file in the same format that they are stored in the memory

File I/O Functions

- Writing
  - `fputc`
  - `fputs`
  - `fprintf`
  - `fwrite`
- Reading
  - `fgetc`
  - `fgets`
  - `fscanf`
  - `fread`

File Reading

- fgetc
  - Usage:

```
fgetc (<file_pointer>);
```

    Parameter:
    - file_pointer – a FILE pointer that refers to the opened file
  - Return Values:
    - If successful, the int value of the character read in returned
    - If the end-of-file is reached or an error occurs, EOF is returned

File Reading

- Example:

```
#include <stdio.h>
int main () {
  FILE *fp; char c;

  fp = fopen("sample.txt", "r");
  c = fgetc(fp);
  fclose (fp);
  return 0;
}
```

File Reading

- fgets
  - Usage:

    ```
    fgets(<string>, <#_of_char>, <file_pointer>);
    ```

  - Parameters:
    - string – string variable in which the read data is to be stored
    - #_of_char – the maximum numbers of characters to be read
    - file_pointer – the pointer to the opened file

File Reading

- Return Values:
  - Success: string read from the file
  - End-of-file: no change made to the string parameter, NULL is returned
  - Error: NULL is returned
- fgets reads all characters until a new line (‘\n’) or the end of file is reached
- ‘\n’ is included in the resulting string if encountered
- ‘\0’ is automatically appended to the string

File Reading

- Example:

```
#include <stdio.h>
int main () {
  FILE *fp; char str[100];
  fp = fopen (“sample.txt”, “r”);
  fgets (str, 100, fp);
  fclose (fp);
  return 0;
}
```

File Reading

- fscanf
  - Usage:

    ```
    fscanf (<file_pointer>, <format>, …);
    ```

  - Parameters
    - file_pointer – the pointer to the opened file
    - format – string containing the expected format of the input
    - additional arguments – a sequence of references to an object of the type specified by the corresponding %-tag in the format string

File Reading

- Return Values:
  - Successful: number of items successfully read
    - If a matching error occurs, the number might be less than the expected number of items
  - Error: EOF is returned
- fscanf is almost exactly the same as scanf, except for the added file pointer parameter
- fscanf is best used if you know the format of the file you are reading

File Reading

- Example:

```
#include <stdio.h>

int main () {
  FILE *fp; char fname[50], lname[50];
  int age;

  fp = fopen ("sample.txt", "r");
  fscanf (fp, "%s %s %d", fname, lname,
   &age);
  fclose (fp);
  return 0;
}
```

File Reading

- fread
  - Usage:

```
fread (<pointer_to_mem_block>,
       <size_of_each_element>,
       <#_of_elements>, <file_pointer>);
```

  - Parameters
    - pointer_to_mem_block – pointer to the memory block which will hold the data to be read from a file
    - size_of_each_element – size in bytes of each element to be read
    - #_of_elements – number of elements to be read
    - file_pointer – the pointer to the opened file

File Reading

- Return Values
  - Success: the total number of elements, should be equal to the # of elements parameter
  - If the return value is not equal to the # of elements parameter passed to the function, either an error occurred or the end-of-file was reached

File Reading

- Example (using strings):

```
#include <stdio.h>
int main  () {
  FILE *fp; char str[50];
  fp = fopen ("file.txt", "r");

  fread (str, sizeof(char), 50, fp);
  fclose (fp);
  return 0;
}
```

File Reading

- Example (using structures):

```
#include <stdio.h>
typedef struct name_s {
  char fname[50];
  char lname[50];
  int age;
} name_t;
```

File Reading

- Example (using structures):

```
main () {
  FILE *fp; name_t p1[3];

  fp = fopen("file.bin", "rb");
  fread(p1, sizeof(name_t), 3, fp);
  fclose (fp);
  return 0;
}
```

File Writing

- fputc
  - Usage:

    ```
    fputc(<character>, <file_pointer>);
    ```

  - Parameters:
    - character – the character to be written to the file
    - file_pointer – the pointer to the opened file
  - Return Values:
    - Success: the character written to the file
    - Error: EOF

File Writing

- Example:

```
#include <stdio.h>

int main () {
  FILE *fp;

  fp = fopen (“sample.txt”, “w”);
  fputc (‘c’, fp);
  fclose (fp);
  return 0;
}
```

File Writing

- fputs
  - Usage:

    ```
    fputs (<string>, <file_pointer>);
    ```

  - Parameters:
    - string – character array terminated by ‘\0’ to be written to the file
    - file_poniter – poniter to the opened file
  - Return values:
    - Success: nonnegative value
    - Error: EOF

File Writing

- Example

```
#include <stdio.h>

int main () {
  FILE *fp; char str[50];
  … //get the value of str;
  fp = fopen ("sample.txt", "w");
  fputs (str, fp);
  fclose (fp);
  return 0;
}
```

File Writing

- fprintf
  - Usage:

    ```
    fprintf (<file_pointer>, <format>, …);
    ```

  - Parameters:
    - file_pointer – pointer to the opened file
    - format – string containing the format of the output
    - Additional arguments – a sequence of additional arguments each with a value to replace the corresponding %-tag in the format string

File Writing

- Return values
  - Success: the number of characters written
  - Error: negative number
- fprintf is almost the same as printf, except for the additional FILE pointer parameter
- fprintf is best used for formatted output to a file

File Writing

- Example

```
#include <stdio.h>

int main () {
  FILE *fp; char fname[50], lname[5];
  int age;
  … //get values for fname, lname, age
  fp = fopen (“sample.txt”, “w”);
  fprintf (fp, “%s %s %d”, fname, lname, age);
  fclose (fp);
  return 0;
}
```

File Writing

- fwrite
  - Usage:

```
fwrite(<pointer_to_mem_block>,
       <size_of_each_element>,
       <#_of_elements>,
       <file_pointer>);
```

  - Parameters:
    - pointer_to_mem_block – pointer to the memory block which will hold the data to be written
    - size_of_each_element – size in bytes of each element to be written
    - #_of_elements – number of elements to be written
    - file_pointer – the pointer to the opened file

File Writing

- Return values
  - Success: The number of elements successfully written
  - If the return value is not equal to the # of elements parameter, then there might have been an error.

File Writing

- Example (using string)

```
#include <stdio.h>
int main() {
   FILE *fp; char str[50];
   fp=fopen("file.txt", "w");

   ... //get data for str
   fwrite(str, sizeof(char), 50, fp);
   fclose(fp);
   return 0;
}
```

File Writing

- Example (using structures):

```
#include <stdio.h>
typedef struct name_s {
  char fname[50];
  char lname[50];
  int age;
} name_t;
```

File Writing

- Example (using structures):

```
main () {
  FILE *fp; name_t p1[3];
  …//get values for p1
  fp = fopen("file.bin", "rb");
  fwrite(p1, sizeof(name_t), 3, fp);
  fclose (fp);
  return 0;
}
```

Other Functions

- ungetc
  - Usage:

    ```
    ungetc (<character>, <file_pointer>);
    ```

  - Parameters:
    - character – the character to be “pushed back” to the file
    - file_pointer – pointer referring to the opened file
  - Return values:
    - Success – the int value of the character
    - End of file or error - EOF

Other Functions

- Example:

```
#include <stdio.h>
int main(void) {
   FILE *fp; char c;

   fp=fopen("file.txt", "r");
   c=fgetc(fp);
   ungetc(c, fp);
   fclose(fp);
   return 0;
}
```

Other Functions

- feof
  - Usage:

    ```
    feof (<file_pointer>);
    ```

  - Parameters:
    - file_pointer – the pointer to the opened file
  - Return Values:
    - If the end-of-file indicator is not yet is set, a nonzero number is returned
    - Otherwise, zero is returned

File I/O Checking

- In cases where the execution of a program is dependent on successfully opening a file for reading or writing, it is ideal to check if fopen is successful before continuing.

File I/O Checking

- Example:

```
#include <stdio.h>
int main(void) {
   FILE *fp;
   fp=fopen("file.txt", "r");

   //if file cannot be opened, exit
   if(fp==NULL){
          printf("Error opening file.\n");
          exit(0);
   }
   ... //continue execution of program
   return 0;
}
```

File I/O Checking

- It is also a good idea to check if the end-of-file has been reached when reading from a file.

File I/O Checking

- Example:

```
#include <stdio.h>
int main(void) {
   FILE *fp;
   fp=fopen("file.txt", "r");
   ... //check if file is open
   while(!feof(fp)) {...}
   ... //continue execution of program
   fclose(fp);
   return 0;
}
```

Notes

- Before reading or writing to a file, it should first be opened using fopen.
- Before terminating the program, a file should first be closed using fclose. Otherwise, your changes might not be saved.
- If the file being opened for writing/appending does not exist, it will be created.
- A file must exist for it to be read.

Notes

- Formatted input/output, character input/output, and string input/output functions can be used only with text files