

1806ICT

Programming Fundamentals

File IO

1

1

Topics

- **Sequential** File Handling in C
- Basic File I/O Functions
- Example: Count words in a file

2

2

File Handling in C

- Files need to be *opened* before use.
 - Associate a "***file handler***" to each file
 - Modes: read, write, or append
- File input/output functions use the file handler (*not* the filename).
- Need to *close* the file after use.
- Basic file handling functions: **fopen()**, **fclose()**, **fscanf()**, **fprintf()**.

3

3

File I/O Example

- Write a program which:
 - reads a list of names from a file called *names.lst*
 - counts the number of names in the list
 - asks a mark for each name in the file
 - outputs the name and corresponding mark to the file *names_marks.dat*
- Note: the tasks above are not necessarily accomplished in that order.

4

4

File I/O (Header)

- *Step 0*: Include **stdio.h**.

```
#include <stdio.h>

int main()
{
    ...

    return 0;
}
```

5

5

File I/O (File Pointer)

- *Step 1*: Declare a **file handler** (a.k.a. **file pointer**) as **FILE *** for each file.

```
int main()
{
    FILE *inputfile = NULL;
    FILE *outputfile = NULL;
    FILE *currentfile = NULL;

    ...

    return 0;
}
```

6

6

File I/O (Open)

- *Step 2: Open* file using **fopen()** .

```
int main()
{
    FILE  *inputfile = NULL;
    FILE  *outputfile = NULL;
    FILE  *currentfile = NULL;

    inputfile = fopen("names.lst", "r");
    outputfile = fopen("marks.dat", "w");
    currentfile = fopen("names_marks.dat", "a");

    ...
    return 0;
}
```

7

7

File I/O (Open)

- *Step 2: Open* file using **fopen()** .

```
int main()
{
    FILE  *inputfile = NULL;
    FILE  *outputfile = NULL;
    FILE  *currentfile = NULL;

    inputfile = fopen("names.lst", "r");
    outputfile = fopen("marks.dat", "w");
    currentfile = fopen("names_marks.dat", "a");

    ...
    return 0;
}
```

File name

8

8

File I/O (Open)

- **Step 2:** Open file using **fopen()**

```
int main()
{
    FILE *inputfile = NULL;
    FILE *outputfile = NULL;
    FILE *currentfile = NULL;
```

```
    inputfile = fopen("names.lst", "r");
    outputfile = fopen("marks.dat", "w");
    currentfile = fopen("names_marks.dat", "a");
```

```
    ...
    return 0;
}
```

Mode
r : read
w : write
a : append

Warning: The "w" mode overwrites the file, if it exists.

9

File I/O (Open)

- **Step 2:** Open file using **fopen()** .

```
int main()
{
    FILE *inputfile = NULL;
    FILE *outputfile = NULL;
    FILE *currentfile = NULL;
```

```
    inputfile = fopen("names.lst", "r");
    outputfile = fopen("marks.dat", "w");
    currentfile = fopen("names_marks.dat", "a");
```

```
    ...
    return 0;
}
```

Associate a **file handler** for **every file** to be used.

10

10

File I/O (Error Check)

- **Step 3:** Check if file is opened successfully.

```
int main()
{
    FILE  *inputfile = NULL;

    inputfile = fopen("names.lst", "r");

    if (inputfile == NULL)
    {
        printf("Unable to open input file.\n");
        return 1;
    }

    ...
    return 0;
}
```

11

11

File I/O (Error Check)

- **Step 3:** Check if file is opened successfully.

```
int main()
{
    FILE  *inputfile = NULL;

    inputfile = fopen("names.lst", "r");

    if (inputfile == NULL)
    {
        printf("Unable to open input file.\n");
        return 1;
    }

    ...
    return 0;
}
```

**File handler
becomes *NULL*
when an *fopen()*
error occurs.**

12

12

File I/O (Error Check)

- **Step 3:** Check if file is opened successfully.

```
int main()
{
    FILE *inputfile = NULL;

    inputfile = fopen("names.lst", "r");

    if (inputfile == NULL)
    {
        printf("Unable to open input file.\n");
        return 1;
    }

    ...
    return 0;
}
```

*Ends program
if inside **main ()**
function.*

13

13

File I/O (Input)

- **Step 4a:** Use **fscanf ()** for input.

```
#include <stdio.h>
#define MAXLEN 100

int main()
{
    FILE *inputfile = NULL;
    char name[MAXLEN];
    int count;
```

listnames.c

14

14

File I/O (Input)

- **Step 4a:** Use **fscanf()** for input.

```
#include <stdio.h>
#define MAXLEN 100

int main()
{
    FILE *inputfile = NULL;
    char name[MAXLEN];
    int count;
```

Recall: **Macro definition**

15

15

File I/O (Input)

- **Step 4a:** Use **fscanf()** for input.

listnames.c

```
/* Assuming "names.lst" contains a
   list of names, open this file for
   reading. */

inputfile = fopen("names.lst", "r");

if (inputfile == NULL)
{
    printf("Error opening names file.\n");
    return 1;
}
```

16

16

File I/O (Input)

- **Step 4a:** Use **fscanf()** for input.

listnames.c

```
/* Read in each name, and keep count how
   many names there are in the file. */

count = 0;
while ( fscanf(inputfile, "%s", name) == 1 )
{
    count++;
    printf("%d. %s\n", count, name);
}

printf("\nNumber of names read: %d\n", count);
return 0;
}
```

17

17

File I/O (Input)

- **Step 4a:** Use **fscanf()** for input.

listnames.c

```
/* Read in each name, and keep count how
   many names there are in the file. */

count = 0;
while ( fscanf(inputfile, "%s", name) == 1 )
{
    count++;
    printf("%d. %s\n", count, name);
}

printf("\nNumber of names
return 0;
}
```

Requires the **file handler** ("stream"), **not** the file name.

18

18

File I/O (Input)

- **Step 4a:** Use **fscanf()** for input.

listnames.c

```
/* Read in each name, and keep count how
   many names there are in the file. */

count = 0;
while ( fscanf(inputfile, "%s", name) == 1 )
{
    count++;
    printf("%d. %s\n", count, name);
}

printf("\nNumber of names\n");
return 0;
}
```

Other parameters:
like ordinary **scanf()**.

19

19

File I/O (Input)

- **Step 4a:** Use **fscanf()** for input.

listnames.c

```
/* Read in each name, and keep count how
   many names there are in the file. */

count = 0;
while ( fscanf(inputfile, "%s", name) == 1 )
{
    count++;
    printf("%d. %s\n", count, name);
}

printf("\nNumber of names\n");
return 0;
}
```

fscanf() returns the **number of**
input items converted and
assigned successfully .

20

20

File I/O (Input)

- **Step 4a:** Use **fscanf()** for input.

listnames.c

```
/* Read in each name, and keep count how
   many names there are in the file. */

count = 0;
while ( fscanf(inputfile, "%s", name) == 1 )
{
    count++;
    printf(" ");
}

printf("\nNumber of names: %d", count);
return 0;
}
```

Used to check if a **read or assignment error** occurred, or **end of input file** has been reached.

21

File I/O (Output)

- **Step 4b:** Use **fprintf()** for output.

listnames2.c

```
#include <stdio.h>
#define MAXLEN 100

int main()
{
    FILE    *inputfile = NULL;
    FILE    *outfile = NULL;
    char    name[MAXLEN];
    int     count;
    float   mark;

    /* Assuming "names.lst" contains a list of names,
       open this file for reading. */

    inputfile = fopen("names.lst", "r");
    if (inputfile == NULL)
    {
        printf("Error opening names file.\n");
        return 1;
    }
}
```

22

22

File I/O (Output)

- **Step 4b:** Use **fprintf()** for output.

listnames2.c

```
/* The output file "names_marks.dat" will  
contain the list of names and  
corresponding marks. */
```

```
outfile = fopen("names_marks.dat", "w");
```

```
if (outfile == NULL)  
{  
    printf("Error opening output file.\n");  
    return 1;  
}
```

23

23

File I/O (Output)

- **Step 4b:** Use **fprintf()** for output.

```
/* Read in each name, ask for the mark, and write name and mark to  
output file. Also keep count how many names there are in the  
file. */
```

```
count = 0;  
while ( fscanf(inputfile, "%s", name ) == 1 )  
{  
    count++;
```

```
    printf("Enter mark for %s: ", name);  
    scanf("%f", &mark);
```

```
    if ( fprintf(outfile, "%s %f\n", name, mark) <= 0 )  
    {  
        printf("Error writing to output file.\n");  
        return 1;  
    }  
}  
/** etc **/
```

listnames2.c

24

File I/O (Output)

- **Step 4b:** Use **fprintf()** for output.

```
/* Read in each name, ask for the mark, and write name and mark to an
output file. Also keep count how many names there are in the
file. */

count = 0;
while ( fscanf(inputfile, "%s", name) == 1 )
{
    count++;

    printf("Enter mark for %s: ", name);
    scanf("%f", &mark);

    if ( fprintf(outfile, "%s %f\n", name, mark) <= 0 )
    {
        printf("Error writing to output file.\n");
        return 1;
    }
}
/**** etc ****/
```

File handler, not the file name.

25

25

File I/O (Output)

- **Step 4b:** Use **fprintf()** for output.

```
/* Read in each name, ask for the mark, and write name and mark to an
output file. Also keep count how many names there are in the
file. */

count = 0;
while ( fscanf(inputfile, "%s", name) == 1 )
{
    count++;

    printf("Enter mark for %s: ", name);
    scanf("%f", &mark);

    if ( fprintf(outfile, "%s %f\n", name, mark) <= 0 )
    {
        printf("Error writing to output file.\n");
        return 1;
    }
}
/**** etc ****/
```

Other parameters: like ordinary printf().

26

26

File I/O (Output)

- **Step 4b:** Use **fprintf()** for output.

```
/* Read in each name, ask for the mark, and write name and mark to
   output file. All done in one file. */

count = 0;
while ( fscanf(inp, "%s %f", &name, &mark) != EOF )
{
    count++;

    printf("Enter mark for %s: ", name);
    scanf("%f", &mark);

    if ( fprintf(outfile, "%s %f\n", name, mark) <= 0 )
    {
        printf("Error writing to output file.\n");
        return 1;
    }
}

/** etc **/
```

fprintf() returns the number of characters written out successfully, or negative if an error occurs.

listnames2.c

27

File I/O (Close)

- **Step 5:** Close file using **fclose()**

```
int main()
{
    /** etc **/

    printf("\n");
    printf("Number of names read: %d\n", count);

    fclose(inputfile);
    fclose(outfile);

    return 0;
}
```

28

28

File I/O (Close)

- **Step 5:** Close file using **fclose()**

```
int main()
{
    /*** etc ***/

    printf("\n");
    printf("Number of names read: %d\n", count);

    fclose(inputfile);
    fclose(outfile);

    return 0;
}
```

**File handler, not the
file name.**

29

29

File I/O (Close)

- **Step 5:** Close file using **fclose()**

```
int main()
{
    /*** etc ***/

    printf("\n");
    printf("Number of names

    fclose(inputfile);
    fclose(outfile);

    return 0;
}
```

- **Clears input buffer.**
- **Flushes output buffer.**
- **fclose()** fails when the file was not opened successfully.

30

Notes on Filenames

- Unless a directory path is specified, the program will look for the file in the current directory.

- Directory paths in filenames: Windows

```
sysFile = fopen("C:\\win\\system.ini", "r");
```

- Directory paths in filenames: Unix

```
passFile = fopen("/usr/etc/passwd", "r");
```

31

31

Notes on Filenames

- Variable filenames:

```
FILE *outFile = NULL;
char someName[MAX_NAME_LEN];

printf("Please enter output filename: ");
scanf("%s", someName);

outFile = fopen(someName, "w");
```

32

32

File I/O and Streams

- A *stream* serves as a channel to convey characters between I/O devices and programs.
- Standard streams: **stdin**, **stdout**, **stderr**
- A file handler/pointer serves as a *stream* to/from a file.
- Once an item is read from an input stream, the *file position* moves automatically, and the next item (if any) becomes available for the next read ("*sequential access*").

33

33

Notes on Strings and fscanf()

- Reading in a string:
fscanf(*stream*, "%s", *string*)
 - Reads only a "word" at a time.
 - Words are separated by a *white-space*: (space, tab, newline, or any combination of these)
 - Moves to the next word in the stream automatically after each read.
- **scanf**("%s", *string*)
 - behaves similarly, except input stream is **stdin**.
 - **eg:== fscanf(stdin, "%s", *string*)**

34

34

Checking for EOF

- Both **scanf()** and **fscanf()** return:
 - the **number of input items** converted and assigned successfully
 - or the constant value **EOF** when an error or end-of-file occurs, *but...*

Not recommended!

```
count = 0;
while ( fscanf(inputfile, "%s", name) != EOF )
{
    count++;
    printf("%d. %s\n", count, name);
}
```



35

35

Checking for EOF

- Warning! Use **EOF** with caution!

Can cause bad problems if the conversion specifiers *do not match* the file's contents.

```
while ( fscanf(inpfile, "%s %f", name, &mark) != EOF )
{
    printf("%s\t %f\n", name, mark);
}
```

listmarks.c

Example: `infile: Jake absent`

36

36

Checking for EOF

- To check for **end-of-file** (or any other **input error**), check that the **number of items** converted and assigned successfully is **equal** to the **expected** number of items.



```
while ( fscanf(inp, "%s %f", name, &mark) == 2 )  
{  
    printf("%s\t %f\n", name, mark);  
}
```

listmarks.c

37

37

Checking for EOF

- To check for **end-of-file** (or any other **input error**), check that the **number of items** converted and assigned successfully is **equal** to the **expected** number of items.

Ditto for scanf().



```
if ( scanf("%d %d %d", &page, &row, &col) != 3 )  
{  
    printf( "I cannot go on without 3 integers :-( \n" );  
    exit(1);  
}
```

testscanf1.c

38

38

Checking for EOF

- To check for end-of-file (or any other input error), check that the number of items converted is equal to the number expected.

The **exit()** function causes the program to **terminate immediately**;
Requires **#include <stdlib.h>**

```
if ( scanf("%d %d", &page, &row, &col) != 3 )
{
    printf("I cannot go on without 3 integers :-( \n" );
    exit(1);
}
```

39

39

Example: Count Words

- Write a program which counts the number of "words" in a file.
 - Note that as far as **scanf()** and **fscanf()** are concerned, any sequence of non-whitespace characters is a "word."

40

40

Count Words: Algorithm

ask the user for the name of the file
open the file
check if file is opened successfully

count the number of words in the file

print out the count
close the file

41

41

Count Words: Algorithm

To count the number of words in a file:

```
set count to 0
loop
{
  read a word from the file
  if attempt to read a word failed
  then { exit loop }
  add 1 to count
}
```

*Gotta write a
function for
this!*

42

42

Function: countWords ()

- Function prototype:
`int countWords (FILE *inpf);`
- Description:
 - This function returns the number of "words" in the input stream **inpf**.

43

43

Function: countWords ()

- PRE-Conditions:
 - It assumes that **inpf** is a pointer to a file which has been opened successfully. There is no check for that within the function.
 - It also assumes that the file position is at the start of the input file/stream.
 - Note that a "word" here means any string of characters, separated from other words by any whitespace (ie. space, tab, newline, or combination).
 - It assumes that no "word" in the file has more than (**MAXLEN** - 1) characters.

44

44

Function: countWords ()

- PRE-Conditions:

- It assumes that `inpf` is a pointer to a file which has been opened within the function. It checks for that.
- It also assumes that the start of the input file/
- Note that a "word" is defined as a sequence of characters, separated by any whitespace (i.e., space, tab, newline, or combination).
- It assumes that no "word" in the file has more than (**MAXLEN** - 1) characters.

**We will use a
#define to set the
value of MAXLEN.
(See countwords.h)**

45

45

Function: countWords ()

- POST-Conditions:

- At the end of the function, the file position will be at the end of file.
- The function returns an integer value which is the number of "words" in the file.

46

46

Function: countWords ()

```
int
countWords ( FILE *inpf )
{
    char  word[MAXLEN];
    int   count = 0;

    while ( fscanf(inpf, "%s", word) == 1 )
    {
        count++;
    }

    return count;
}
```

countwords.c

47

47

Count Words: Algorithm

ask the user for the name of the file
open the file
check if file is opened successfully

count the number of words in the file

print out the count

close



*I can also write a
reusable function
for these!*

48

48

Function: openInput ()

- Function prototype:
`FILE* openInput (void);`
- Description:
 - This function keeps asking the user for a filename, until it is able to open the file successfully for input.

49

49

Function: openInput ()

- PRE-Condition:
 - It assumes that the filename fits in a string of size MAXLEN (including the '`\0`').

50

50

Function: openInput ()

- POST-Conditions:
 - It can cause the program to terminate if the user chooses to abort the operation.
 - It returns the file handler/pointer for the specified file.
 - It assumes that the calling function has the corresponding variable to catch the return value.
 - It also assumes that the calling function takes care of closing the file.

51

51

```
FILE*  openInput ( void )
{
    FILE  *handle = NULL;
    char  theFile[MAXLEN];
    int   option;

    while (1)
    {
        printf("Enter file name: ");
        scanf("%s", theFile);

        if ((handle = fopen(theFile, "r")) == NULL )
        {
            /* Insert code to handle open error. */
        }
        else
        {
            break;
        }
    }
    return handle;
}
```

countwords.c

52

52

Code to handle open error:

```
printf("Error opening file.\n");

option = 0; /* Set default to abort. */

printf("\nEnter 1 to try again, ");
printf("or any number to abort: ");

scanf("%d", &option);
printf("\n");

if ( option != 1 )
{
    printf("Alright then. ");
    printf("Program terminated.\n");
    exit(1);
}
```

countwords.c 53

53

Main Algorithm

set *file* to be the result of **openInput()**

set *count* to the result of **countWords(file)**

print out the *count*

close the *file*

54

54

Test Program #1

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

#include "countwords.h"

int main()
{
    FILE *inputFile = NULL;
    int count;

    inputFile = openInput();

    count = countWords(inputFile);
    printf("\nThere are %d words in the file.\n", count);

    fclose(inputFile);

    return 0;
}
```

testcount1.c

55

55

Test Program #2

```
#include <stdio.h>
#include <stdlib.h>
#include "countwords.h"

int main()
{
    FILE *inputFile = NULL;
    int count;

    inputFile = openInput();

    count = countWords(inputFile);
    printf("\nThere are %d words in the file.\n", count);

    count = countWords(inputFile);
    printf("\nThere are %d words in the file.\n", count);

    fclose(inputFile);
    return 0;
}
```

What is the result if we call the countWords() function a second time over the same file?

testcount2.c

56

Reading a Line of Text

- So far we have seen how to read in a word
- To read in a line of words (text), we can use
`char *fgets(char *str, int n, FILE *stream)`
- It reads a line from the specified stream, and stores it into the string pointed to by `str`
- It stops when either $(n-1)$ characters are read, the newline character is read, or the end-of-file is reached, whichever comes first

57

57

Example:

```
#include <stdio.h>

#define MAXLEN 100

int main()
{
    char str[MAXLEN];

    while (fgets(str, sizeof(str), stdin) != NULL)
    {
        printf("%s\n", str);
    }

    return 0;
}
```

58

58

Summary

- Sequential File handling
- EOF/Error Checking
- Additions to your "C vocabulary":
 - **FILE ***
 - **fopen() , fscanf() , fprintf() , fclose()**
 - **EOF**
 - **exit()**

59

59

fgetc()

- **fgetc()** can be used to read in a single char:
`char c = fgetc();`

60

60

fputs()

- `fputs()` write a string to a file:

```
fputs(outfile, "String to write");
```

```
// Equivalent to:
```

```
fprintf(outfile, "%s", "String to  
write");
```

61

61

Writing to a string

- C provides functions similar to `fprintf()`, `fscanf()`, etc. for writing and reading to/from strings:
 - `sprintf()`
 - `sscanf()`
- Instead of passing a `FILE` pointer, a string is passed in

62

62

Writing it a string

- The following example writes a number of variables to a string:

```
char s[100];  
char message[] = "Hello";  
int x = 2;  
float y = 0.5;  
sprintf(s, "%s %d %f", message, x, y);
```

63

63

Reading from a string

- In the following example a line is read from the file and then read using `sscanf ()`:

```
char s[100];  
fgets(inputFile, s); // read a line  
char word1[10];  
char word2[10];  
sscanf(s, "%s %s", word1, word2);
```

64

64