

Lecture 7

Text Files

Today's Topics

- File Concepts
- File Functions
- Translating Flowchart to C Code

Concept of a file

- A file is an **external data**.
- It is usually stored in secondary storages – eg. hard disk
- Why file?
 - the contents of primary memory are lost when the computer is shut down.
 - to store the data in a **permanent** form, we use files.
- Anything in the secondary storage are treated as files – eg. Application programs, Microsoft Word documents, C source codes etc.
- Types of file: text files and binary files
 - **Text files** – the contents are human-readable.
 - **Binary files** – the contents are computer-readable.

Concept of a file

Text file example – prog1.c (a C source code)

```
/* This program uses defined, memory, and literal constants.
   Written by :
   Date:
*/
#include <stdio.h>
#include <conio.h>

#define A 'a'
#define E 'e'

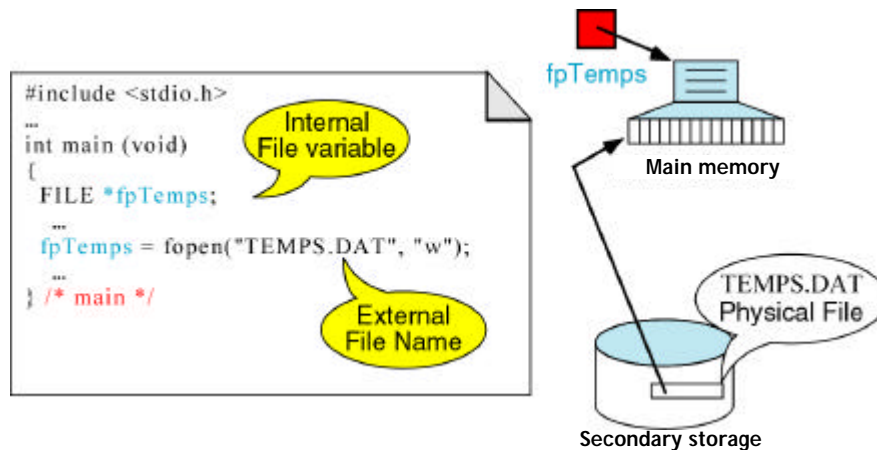
int main (void)
{
    /* Local definitions */
    const int even0 = 0;
    const int even2 = 2;

    /* Statements */
    printf ("%3c%3c\n", A, E);
    printf ("%3d%3d\n",
            even0, even2);
    printf ("%3d%3d%3d\n", 1, 3, 5);

    getch();
    return 0;
} /* end of main */
```


Concept of a file

- Internal vs. external file names



Concept of a file

- We use a file for three purposes:
 - reading data from it
 - writing or printing data into it
 - reading and writing data from/into it
- These are called **file modes**
- A file that is used for reading is called an **input file** (or data file)
- A file that is used for writing is called an **output file**
- We **cannot read** data from an **output file**.
- We **cannot write** data into an **input file**
- Solution: set the file to **read and write mode**. Unfortunately, this topic is not covered.

Concept of a file

- To use a file, four things are usually done:
 - create an internal name of the file
 - open the file
 - use the file: reading or writing
 - close the file
- **Creating an internal name**
 - Creating an internal name means declaring a variable for the file - using pointer data type, FILE
 - e.g.

```
FILE * fpTemp;
```
 - **fpTemp** is called **file variable**

Terms we are going to use:

Internal file name => **file variable**
External file name => **file name**

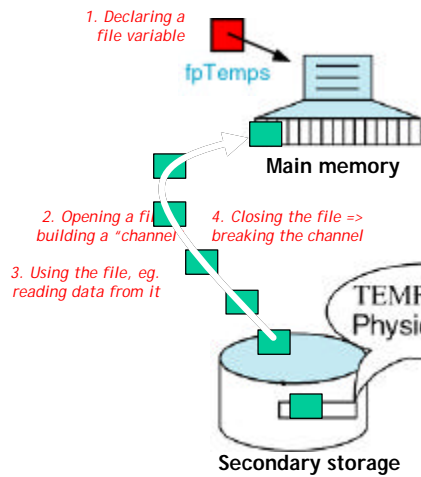
eg. fpTemp => the file variable
"TEMPS.DAT" => the file name

Concept of a file

- **Opening a file**
 - It doesn't mean opening the contents of the file
 - It only creates a "channel" between the internal variable and the external file.
 - This channel provides a medium for transferring data from the secondary storage to the program.
- **Using a file**
 - Using a file : for reading or printing.
 - A file must be opened before it is used.
- **Closing a file**
 - When a file is not used anymore –eg. you don't want to read or print anymore –it must be closed.
 - Closing a file means breaking the "channel"

Concept of a file

The concepts



The C codes

```
#include<stdio.h>

void main(void)
{
    FILE *fpTemps; // step 1

    int data;

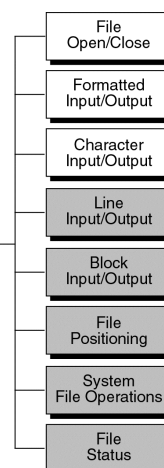
    fpTemps = fopen("TEMPS.DAT", "r");
    fscanf(fpTemps,"%d",&data); //step 3
    fclose(fpTemps); //step 4
}
```

File functions

- <stdio.h> library contains several input/output functions. The categories are as below.
- This lecture discusses only the first two.
- File functions that you are going to learn are:

fopen()
fclose()
fprintf()
fscanf()
feof()

Categories of I/O Functions



fopen function

- It is used for preparing a file to be used.
- It doesn't mean open the contents of a file. It is only open a "channel" from the program to the file.
- Syntax:

```
file_variable = fopen("filename", "file_mode");
```

The **file_variable** must be declared as **FILE *** data type.

Example:

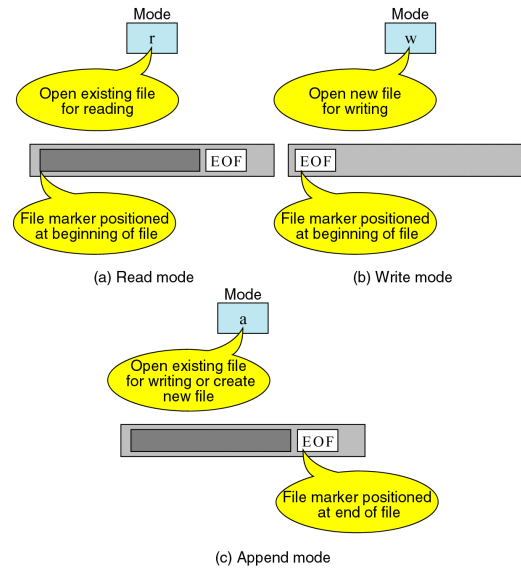
```
FILE *fpTemps;  
fpTemps = fopen("TEMPS.DAT", "r");
```

fopen function

- **File mode** tells the program how you intend to use the file –for reading or writing or both.

File mode	Meaning
r	Open file for reading => an input file <ul style="list-style-type: none">• If file exists, the file marker is positioned at beginning.• If file doesn't exist errors returned.
w	Open file for writing => an output file <ul style="list-style-type: none">• If file exists, it is emptied.• If file doesn't, it is created.
a	Open file for appending = > an output file <ul style="list-style-type: none">• If file exists, the file marker is positioned at end of file.• If file doesn't exist, it is created.

file modes



fclose function

- It is used to close a file when it is no longer needed.
- Closing a file means breaking the "channel" - no more reading or writing operation can be performed to the file.
- Syntax:

```
fopen(file_variable);
```

Example:

```
fclose(fpTemps);
```

fprintf function

- **fprintf** is same as **printf**, except it prints into a file
- It can only be used after the file has been opened.
- Syntax:

```
printf("format string",value_list);  
fprintf(file_variable,"format string",value_list);
```

Example:

```
fprintf( fpTemps, "%d", number);
```

fscanf function

- **fscanf** is same as **scanf**, except it reads from a file
- It can only be used after the file has been opened.
- Syntax:

```
scanf("format string",address_list);  
fscanf(file_variable,"format string",address_list);
```

Example:

```
fscanf( fpTemps,"%d", &number);
```

feof function

- It is used to test whether the file marker has reached the end of file (EOF)
- This function returns a **non-zero** value if the file marker has reached the EOF, otherwise it returns 0 – means the EOF has not been reached.
- Syntax:

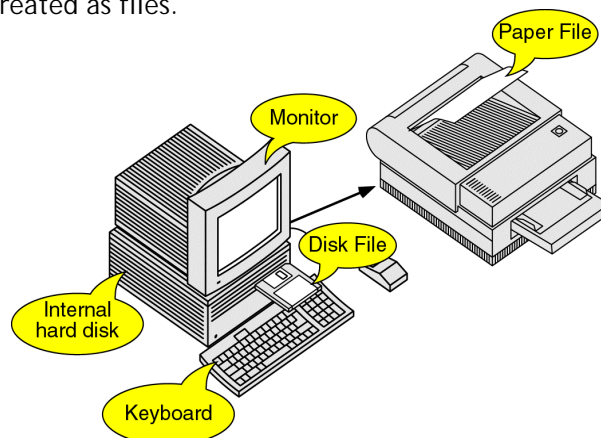
```
feof(file_variable);
```

Example: The following statement tells the program to keep reading more number if the EOF has not been reached.

```
while ( feof(fpTemps) != 0 )  
{  
    fscanf(fpTemps,"%d", &number);  
}
```

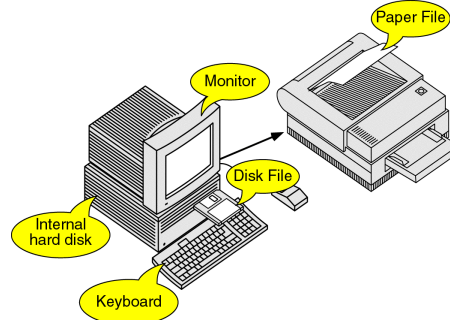
Standard files

- In C, the concept of files is not limited to secondary storage files.
- Any input and output devices – as shown below - may be treated as files.



Standard files

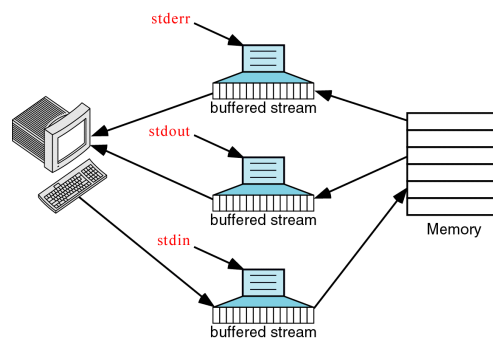
- In C, the concept of files is not limited to secondary storage files.
- Any input and output devices – as shown below - may also be treated as files.



- Monitor, keyboard and printer are called **standard files**
- Files that are created using FILE type are called **user files**

Standard files

- There are three standard file variables in C - **stdin**, **stdout** and **stderr**



- **stdin** is linked to the primary input device – usually the keyboard.
- **stdout** and **stderr** are linked to the primary output device – usually the monitor

Standard files

- We may use the standard file variables without declaring or opening them.
- They are automatically declared and opened in <stdio.h>
- In the following example, the input is read from the standard input – keyboard- and the output goes to the standard output - screen.

```
#include<stdio.h>

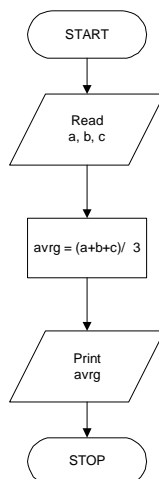
void main(void)
{
    int number;

    fscanf(stdin,"%d",&number);

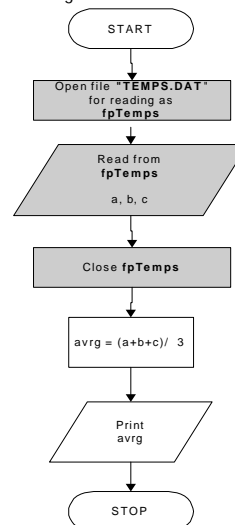
    fprintf(stdout,"%d", number);
}
```

Translating Flowchart to C Code

Without file:
Reading three numbers **from the keyboard** and then prints their average onto the screen

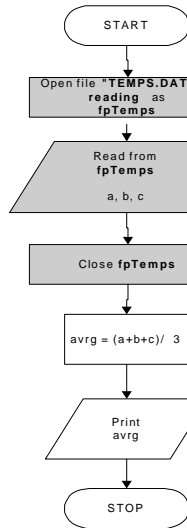


With file:
Reading three numbers **from a file** and then prints their average



Translating Flowchart to C Code

Reading three numbers from a file and then prints their average onto the screen



```

#include<stdio.h>

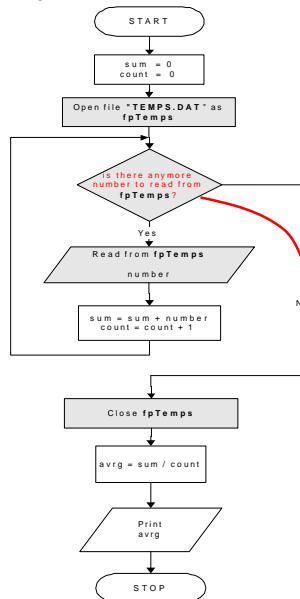
void main(void)
{
    FILE *fpTemps;
    int a, b, c;
    float avrg;

    fpTemps = fopen("TEMPS.DAT", "r");
    fscanf(fpTemps,"%d",&a, &b, &c);
    fclose(fpTemps);

    avrg = (a+b+c)/3.0;
    printf("%.2f", avrg);
}
    
```

Translating Flowchart to C Code

Reading all numbers from a file and then prints their average onto the screen



```

#include<stdio.h>

void main(void)
{
    FILE *fpTemps;
    int number;
    int count;
    int sum;
    float avrg;

    sum=0;
    count=0;

    fpTemps = fopen("TEMPS.DAT", "r");

    while ( feof(fpTemps) != 0 )
    {
        fscanf(fpTemps,"%d",&number);
        sum += number;
        count++;
    }

    fclose(fpTemps);

    avrg = (float)sum/(float)count;
    printf("%.2f", avrg);
}
    
```