FILE HANDLING IN C

LECTURE 40

Introduction

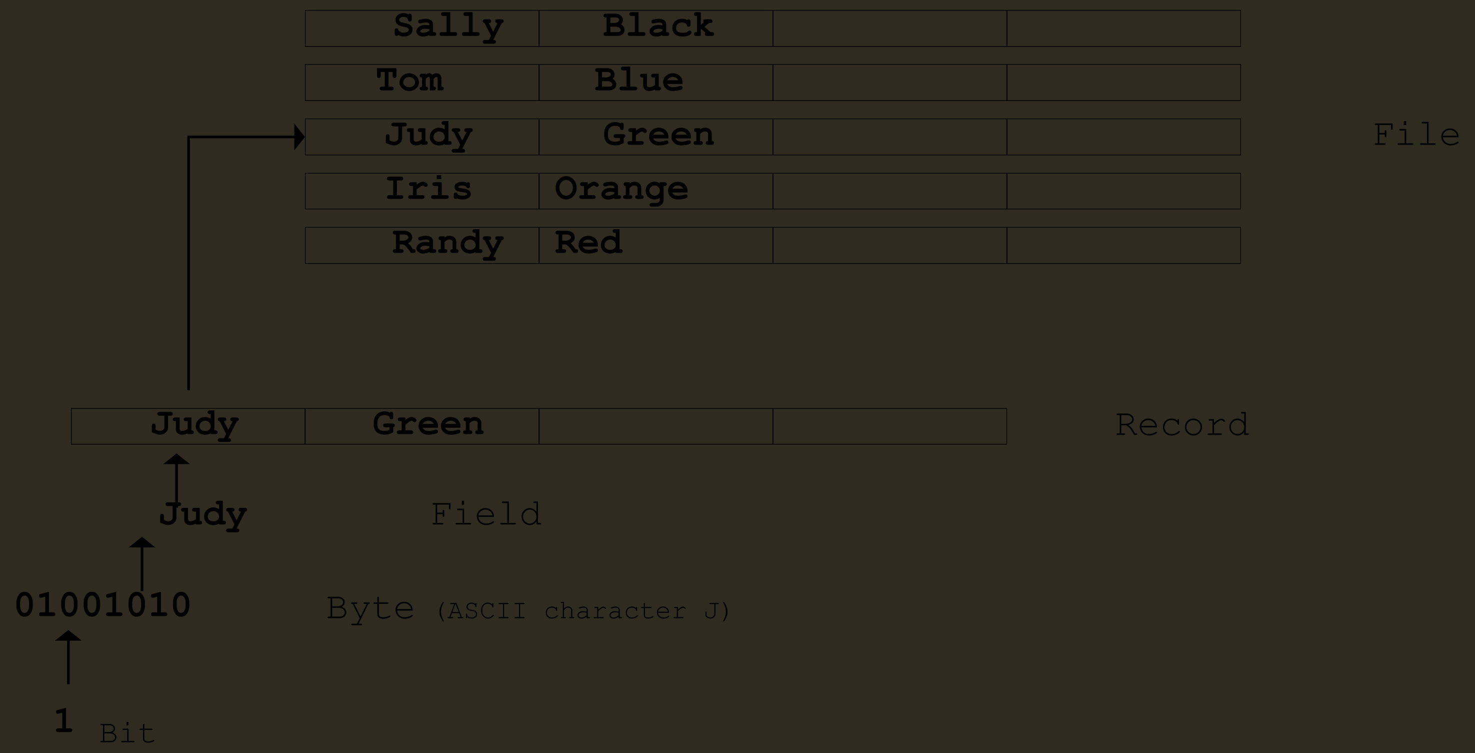
- Data files
  - Can be created, updated, and processed by C programs
  - Are used for permanent storage of large amounts of data
    - Storage of data in variables and arrays is only temporary

The Data Hierarchy

- Data Hierarchy:
  - Bit – smallest data item
    - Value of **`0`** or **`1`**
  - Byte – 8 bits
    - Used to store a character
      - Decimal digits, letters, and special symbols
  - Field – group of characters conveying meaning
    - Example: your name
  - Record – group of related fields
    - Represented by a **`struct`** or a **`class`**
    - Example: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.

The Data Hierarchy

- Data Hierarchy (continued):
  - File – group of related records
    - Example: payroll file
  - Database – group of related files



The Data Hierarchy

- Data files
  - Record key
    - Identifies a record to facilitate the retrieval of specific records from a file
  - Sequential file
    - Records typically sorted by key

Files and Streams

- C views each file as a sequence of bytes
  - File ends with the *end-of-file marker*
    - Or, file ends at a specified byte
- Stream created when a file is opened
  - Provide communication channel between files and programs
  - Opening a file returns a pointer to a **`FILE`** structure
    - Example file pointers:
    - **`stdin`** - standard input (keyboard)
    - **`stdout`** - standard output (screen)
    - **`stderr`** - standard error (screen)
- **`FILE`** structure
  - File descriptor
    - Index into operating system array called the open file table
  - File Control Block (FCB)
    - Found in every array element, system uses it to administer the file

DEFINING AND OPENING A FILE :

Data structure of a file is defined as FILE in the library of standard I/O function

definitions. Therefore all the file should be declared as type FILE before they are used.

FILE is defined data type.

When we open a file, we must specify what we want to do with the file. For example we

may write data to the file or read the already existing data.

General format for declaring and opening a file:

```
FILE *fp;
Fp = fopen("filename", "mode");
```

The first statement declares the variable fp as a "pointer to the data type FILE". The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp.

Mode can be one of the following :

r	**Open the file for reading only.**
w	**Open the file for writing only.**
a	**Open the file for reading only.**

Many recent compilers includes additional modes of operation. They include:

r+	The existing file is opened to the beginning for both reading and writing
w+	Same as w except for both reading and writing.
a+	Same as a except for both reading and writing.

When trying to open a file, one of the following things may happen:

When the mode is ‘writing’, a file with the specified name is created if the file does not exist. The content are deleted if the file already exists.

When the purpose is ‘appending’, the file is opened with the current contents safe. A file with the specified name is created if the file does not exist.

If the purpose is ‘reading’ , and if it exists, then the file is opened with the current contents safe; otherwise an error occurs.

CLOSING A FILE:

A file must be closed as soon as all operations on it have been completed. This ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken.

fclose(file_pointer);

This would close the file associated with the FILE pointer file_pointer. The following segment of a program:

```
…………
…………
FILE  *p1, *p2;
          p1 = fopen("INPUT", "w");
p2 = fopen("OUTPUT", "r");
          …………
………….
fclose(p1);
          fclose(p2);
          ………..
```

This program opens two files and closes them after all operations on

LECTURE 41, 42, 43

INPUT/OUTPUT OPERATIONS ON FILES

INPUT/OUTPUT OPERATIONS ON FILES

Once a file is opened , reading out or writing to it is accomplish ed by using the standard I/O routines.

The getc() and putc() Functions

The simplest file I/O function are getc() and putc(). These are analogous to getchar and putchar function and handle one character at a time. Assume that a file is opened with mode wand file pointer fp1.

Then statement

putc(c,fp1);

Writes the character contained in character variable c to file associated with FILE pointer fp1.

Similarly getc() is used to read a character from a file that has been opened in read mode.

The statement

```
c = getc(fp2);
```

Would read a character from the file whose file pointer is fp2.

The getw() and putw() Functions

The getw and putw are integer oriented functions. They are similar to the getc and putc

functions and are used to read and write integer values. These function would be useful

when we deal with only integer data. The general forms of getw and putw are:

```
putw(integer, fp);


getw(fp);
```

The fprintf() and fscanf() Functions

Most complier support two other functions, namely fprintf and fscanf that can handle a group of mixed data simultaneously.

The function fprintf and fscanf performsI/O operation that are similar to printf and scanf

function, except they work on files.
The general form of fprintf is:

```
fprintf(fp, “control string”, list);
```

Where fp is the file pointer associated with the file that has been opened for writing.

The control string contains output specification for the items in the list. The list may include variables, constants and strings.

For example:

```
fprintf(f1, “%s%d%f” ,name,age,7.5);
```

The general form of fscanf is:

```
fscanf(fp, “control string”, list);
```

For example:

fscanf(f2, “%s%d” ,item, &quqntity);

Like scanf , fscanf also returns the number of items that are successfully read. When the end of file is reached, it returns the value EOF.

LECTURE 44, 45

ERROR HANDLING DURING I/O OPERATIONS

ERROR HANDLING DURING I/O OPERATIONS

1. **Trying to read beyond the end-of-file.**
2. **Device overflow.**
3. **Trying to use a file that has not been opened.**
4. **Trying to perform an operation on a file, when the file is opened for another type of information.**
5. **Opening a file with an invalid file name.**
6. **Attempting to write to a write protected file.**

If we fail to check such read and write errors, a program may behave abnormally when an error occurs.

An unchecked error may result in a premature termination of program or incorrect output.

C has two library functions **feof and ferror** that can help us to detect I/O errors.

If fp is a pointer to a file that has just been opened for reading :

```
if(feof(fp))
 printf("End of data");
```

The ferror function reports the status of the file indicated.

It returns a nonzero integer if an error has been detected upto that point, during processing.

It returns zero otherwise.

```
   tf(ferror(fp)!=0)
   printf("An error has occurred");
```

When we open a file using fopen function, a file pointer is returned .

If the file cannot be opened for some reason, then the function returns null pointer.

This facility can be used to test whether a file has been opened or not.

```
if(fp= =NULL)
printf(“file cannot be opened”);
```

LECTURE 46-50

RANDOM ACCESS TO FILES

RANDOM ACCESS TO FILES

This can be achieved with the help of the following functions:

- **fseek,**
- **ftell**
- **rewind**

ftell

Takes a file pointer and returns a number of type long, that corresponds to the current position. This function is useful in saving the current position of the file, which can be used later in program.

It takes the following form:

```
n = ftell(fp);
```

n would give the relative offset(in bytes) of the current position.

rewind : **This function is used to take the file pointer and resets the position to the start of the .The statement:**

```
 rewind(fp)
n = ftell(fp)
```

fseek: **This function is used to move the file position to a desired location within the file.**

It takes the following form :

fseek(fp, offset, position);

fp is a file pointer to the file concerned offset is a number or variable of type long.

offset may be positive, meaning move upward or negative meaning backward.

position is an integer no.

The position can take one of the following three values.

Value	Meaning
0	Beginning of the file
1	Current position
2	End of the file.