Files

ESC108A Elements of Computer Science and Engineering

B. Tech. 2017

Course Leaders:

Roopa G.

Ami Rai E.

Chaitra S.

Objectives

- At the end of this lecture, student will be able to
  - Use standard file operations
  - Explain user defined data types

Contents

- File I/O in C
- Bit fields
- User defined Data types

Files

- Many real-life problems involve large volumes of data and in such situations, the console oriented I/O operations pose two major problems
  1. It becomes cumbersome and time consuming to handle large volumes of data through terminals
  2. The entire data is lost when either the program is terminated or the computer is turned-off

- Concept of files to store data
  - A more flexible approach where data can be stored on the disk and read whenever necessary, without destroying the data

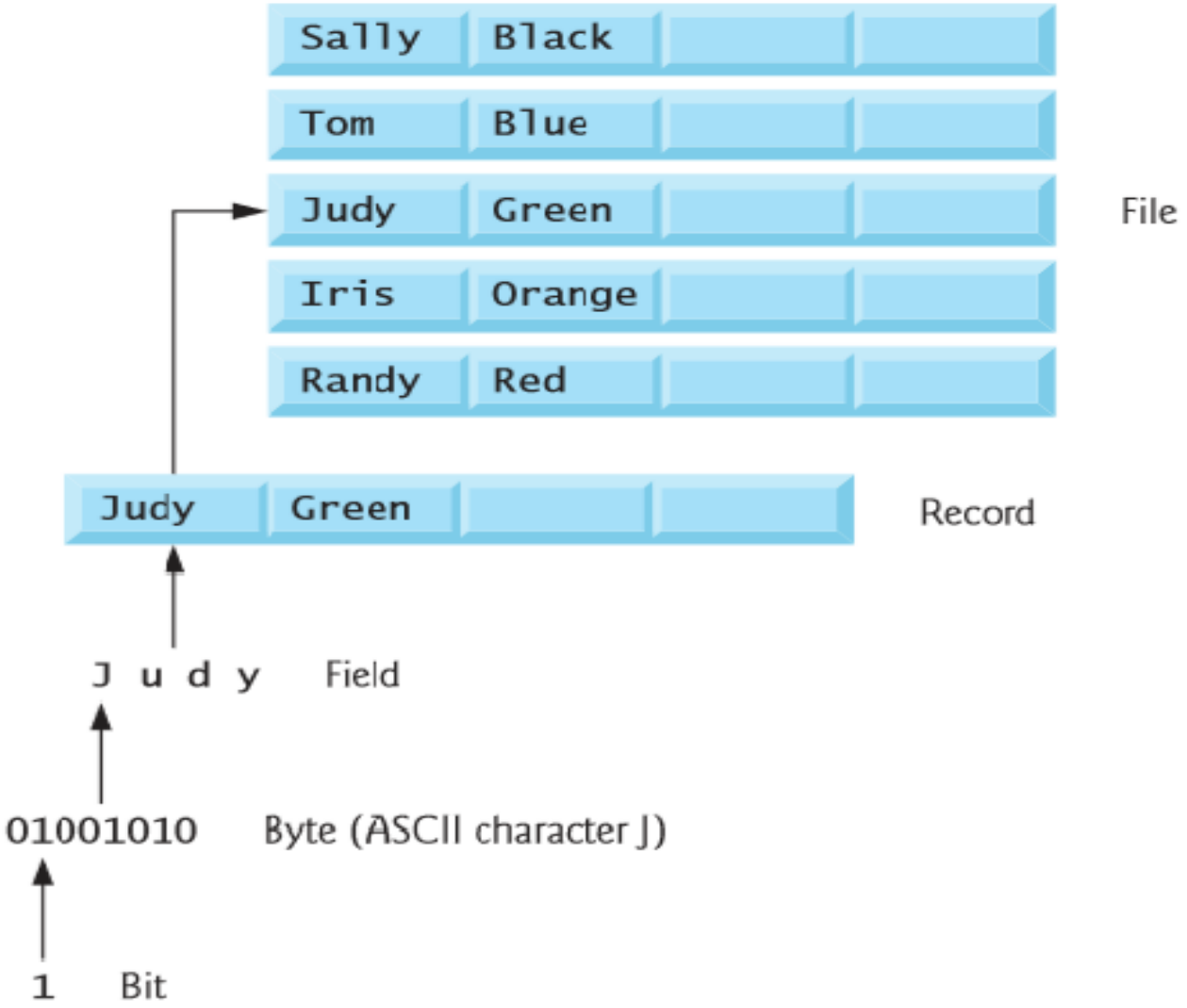
Data Hierarchy

- To build electronic devices that can assume two stable states—0 and 1
- Bit
  - Short for “binary digit”
  - a digit that can assume one of two values
- Byte
  - A pattern of 1s and 0s
- Characters(digits, letters and special symbols) are composed of bits
- Field
  - A group of characters that conveys meaning

Data Hierarchy contd.

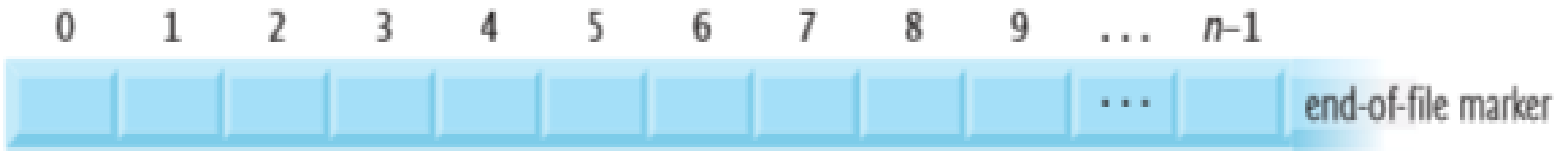
- A record (i.e., a struct in C)
  - Composed of several fields
- A file
  - A a group of related records
- Record key
  - To facilitate the retrieval of specific records from a file
- Database
  - A group of related files

Data Hierarchy contd.



Files and Streams

- C views each file simply as a sequential stream of bytes
- Each file ends either with an end-of-file marker
- When a file is opened, a stream is associated with the file
- Three files and their associated streams are automatically opened when program execution begins
  - standard input, standard output, standard error
- Streams provide communication channels between files and programs



Types of Files

1. Sequential File
   - Records are typically stored in order by the record key field

2. Random access file
   - individual records are normally fixed in length and may be accessed directly (and thus quickly) without searching through other records

File I/O functions

Prototype	Description
FILE	Data type (a structure) to store file information permanently
FILE *fopen(const char *path, const char *mode)	Opens a file and returns FILE pointer. Path is the path to the file. Mode can be combinations of “r”, “w” or “a”. Other possibilities include “+”.
int fclose(file *FP)	Closes a file that was already opened in the program
int fscanf(FILE *stream, const char *format,…)	Similar to scanf for file I/O
int fprintf(FILE *stream, const char *format,…)	Similar to printf for file I/O

Opening and Closing a File

```
int main(int argc, char** argv) {
    FILE *fp;
    fp = fopen("data1.txt", "w");
    if (fp == NULL) {
        printf("File does not exist, please check!\n");
    }
    fclose(fp);
  return (EXIT_SUCCESS);
}
```

File Opening Modes

File Mode	Description
r	Open a text file for reading
w	Create a text file for writing, if it exists, it is overwritten
a	Open a text file and append text to the end of the file
r+	Open a text file both in reading and writing mode
rb	Open a binary file for reading
wb	Create a binary file for writing, if it exists, it is overwritten
ab	Open a binary file and append data to the end of the file

Input/Output Operations on Files

- C provides several different functions for reading/writing

- getc() – read a character from file
- putc() – write a character to file
- fprintf() – write set of data values to file
- fscanf() – read set of data values from file
- getw() – read integer from file
- putw() – write integer to file

getc() and *putc()*

```
int main(int argc, char**
    argv) {
    FILE *fp, char ch;
    fp=fopen("datum.txt","w");

    printf("\nEnter data");
    while((ch=getchar())!=EOF)
        putc(ch,fp);
    fclose(fp);
```

```
fp=fopen("datum.txt","r");
 if(fp==NULL)
        printf("\nFile does not
    exist");
 else{
     while((ch=getc(fp))!=EOF)
            printf("%c",ch);
  }
   fclose(fp);
    return (EXIT_SUCCESS);
}
```

fprintf() and *fscanf()*

```
int main(int argc, char** argv) {
    FILE *fp;
    int day,days;
    char name[10],names[10];
    fp=fopen("reads.txt","w");
    printf("\nEnter data");
    scanf("%d%s",&day,name);
     fprintf(fp,"%d%s",day,name);
    fclose(fp);
```

```
fp=fopen("reads.txt","r");
if(fp==NULL)
    printf("\nFile does not exist");
 else{
    printf("\nDetails are");
    do{
     fscanf(fp,"%d%s",&days,names);
      printf("%d %s",days,names);
    }while(!feof(fp));
}
    fclose(fp); return
    (EXIT_SUCCESS); }
```

Errors that occur during I/O

- Typical errors that occur
  - trying to read beyond end-of-file
  - trying to use a file that has not been opened
  - perform operation on file not permitted by ‘fopen’ mode
  - open file with invalid filename
  - write to write-protected file

Bit Fields

- Member of a structure whose width(in bits) has been specified
- Enable better memory utilization
- Only for ***int*** or ***unsigned***
- Declaring bit fields - Example

```
struct myMacHeader{
    unsigned int dest: 48; /* 48 is width in bits*/
    unsigned int src: 48;;
    unsigned int seq: 16;
    unsigned char * data;
};
```

Bit Fields

- Unnamed bit field
  - Field used as padding (empty area) in the structure
  - Nothing may be stored in the bits

```
struct myStruct{
  unsigned a : 13;
  unsigned   : 8; /*reserved, not used*/
  unsigned b : 7;
}
```

User defined Data Type

- C allows user to define an identifier that would represent an existing data type

  typedef

- The general form is

  typedef type identifier;

- Example

  typedef int units;

  typedef float marks;

Enumerated Data Type

- User defined data type
- Short for Enumeration
- Keyword enum
- A set of integer enumeration constants represented by identifiers
- Enumeration constants - like symbolic constants whose values automatically set
  - Values start at **0** and are incremented by **1**
  - Values can be set explicitly with "**=**"
  - Need unique constant names

ENUM

- Declare variables as normal
  - Enumeration variables can *only* assume their enumeration constant values not int values

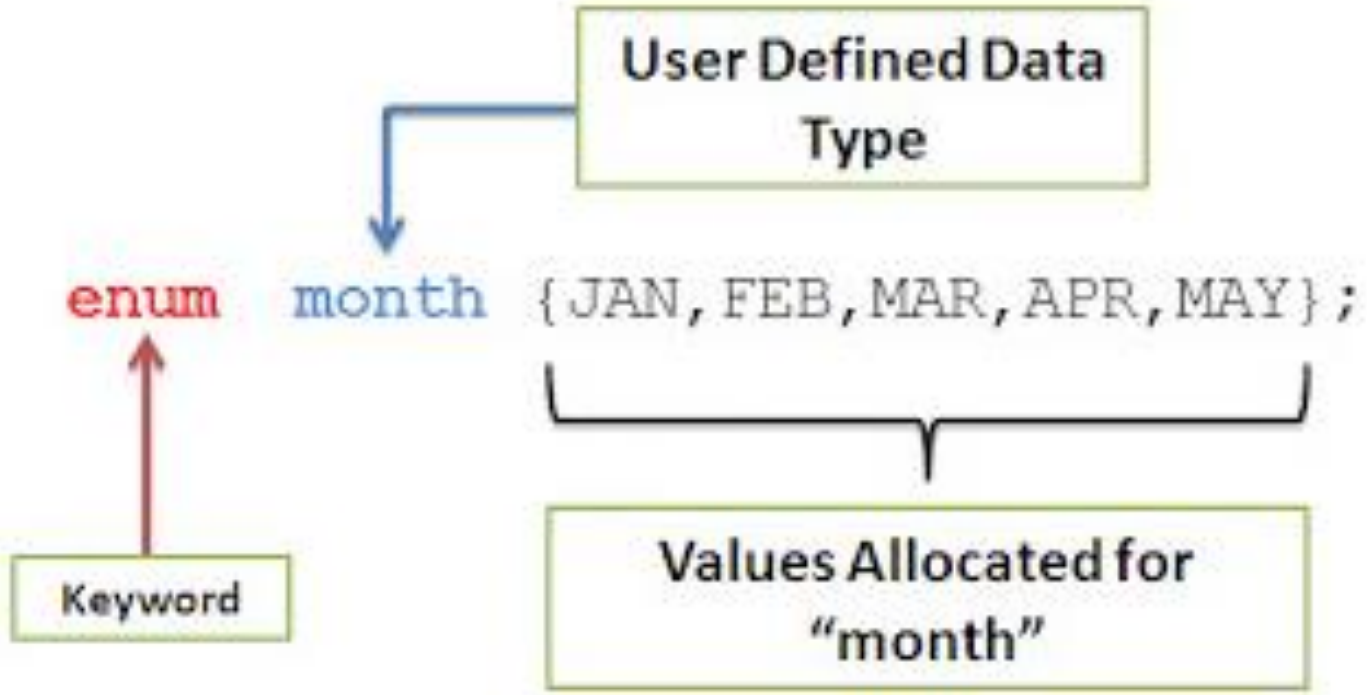
enum DAYS {SUN, MON, TUE, WED, THU, FRI, SAT};

- Starts at **0**, increments by **1**

enum DAYS {SUN= 1, MON, TUE, WED, THU, FRI, SAT};

- Starts at **1**, increments by **1**

ENUM - Example



Summary

- Files can be used to store data
- C standard library provides functions to handle files
- Bitfields help specify member width in bits and can be used only on int and unsigned int members
- User defined Data types are used to define an identifier that would represent an existing data type
- Enumerated data type helps create more readable code by assigning integer constants to enumerated values

Further Reading

Kernighan, B. W. and Richie, D. (1992) *The C Programming Language.* 2nd ed., New Delhi:PHI.