C-Refresher: Session 10
Disk IO

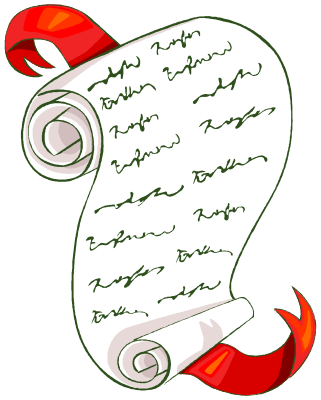
Arif Butt
Summer 2017

I am Thankful to my student Muhammad Zubair bcsf14m029@pucit.edu.pk for preparation of these slides in accordance with my video lectures at

http://www.arifbutt.me/category/c-behind-the-curtain/

Today's Agenda

- Introduction to Program Files
- Opening/Closing a Disk File
- Reading from an Open File
- Random Access
- Binary Files



Introduction

- When a program starts its execution, there are three streams that are open at the start of the program
  i. stdin
  ii. stdout
  iii. stderr
- We know how to read/write from these streams using formatted and unformatted I/O
- To read a file residing on the hard disk, we need to open a new stream between our program and the file residing on the hard disk

Opening/Closing a Disk File

- To open a stream between our program and the file we use `fopen()` library call
- **Syntax**
- `File *fopen(const char* path,const char* mode);`
- It takes two arguments both are of type `string`
- In case of success, it returns a pointer of type `FILE*`, pointing to the file
- In case of failure, it returns a `NULL` pointer
- First argument is the file name with the absolute or relative path
- Second argument specifies the mode for opening the file

Text file Symbol	Binary file Symbol	Mode Description
`r`	`rb`	Opens the file for reading with pointer at start of file Call fails if file doesn't exist
`r+`	`r+b`	Same as `r` but allows writing as well
`w`	`wb`	Opens file for writing with write pointer at start If file doesn't exist, a new file is created
`w+`	`w+b`	Same as `w` but allows reading as well
`a`	`ab`	Opens file for writing, with write pointer at the end If file doesn't exist, creates a new file
`a+`	`a+b`	Same as `a` but allows reading as well

Opening/Closing a Disk File(cont…)

❑**Return value**

- **On success:**
- `fopen()` returns a file pointer that is used in all the subsequent calls, i.e. read, write and finally close the file
- This pointer points to a structure of type `FILE`
- **`FILE`** structure contains information about the file like
  - location of buffer
  - current file offset(`cfo`)
  - Opening mode
  - Flags like EOF(End Of File flag)its value can be checked by `feof(fp)`

Opening/Closing a Disk File(cont…)

- **On Error:** `fopen()` **returns** `NULL`
- A file opened must be closed after you have performed all the necessary operations on it
- Closing a file breaks the connection between the file on disk and the file pointer
- `int fclose(FILE *stream);` /*this function is used to close an opened file*/
- It takes file pointer as the argument and closes that file
- When a program terminates, all the opened files are automatically closed
- But it is a good practice to close all the opened files

Reading from an Open File

- There are two categories of functions
  1. Unformatted
  2. Formatted

❑**Unformatted functions**

1. **Reading character by character and displaying on `stdout` till `EOF`**

- `fputc()` and `fgetc()` are the functions used for this
- **Syntax**
- `int fgetc(FILE* stream);`
- It takes file pointer as argument and reads from that file

Reading from an Open File(cont…)

- `fgetc()` reads a character from the file given as argument and returns it as an unsigned character cast to an `int`
- `int fputc(int c,FILE *stream);`
- `fputc()` writes character `c` to the file given as the 2nd argument
- e.g. `fputc(c,stdout);` //it writes `c` to `stdout`
- Let's see a program using `fgetc()` and `fputc()`

Reading from an Open File(cont…)

/*The program reads from file character by character till EOF and displays it on stdout*/

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char* argv[]){
  if(argc!=2){ //checking if file name given or not
    printf("Invalid number of arguments entered.
Please enter filename to display its contents....\n");
    exit(1);}
  FILE* fp=fopen(argv[1],"r");
```

/*passing argv[1] for the file name, as user will pass file name in it, and opening the file in read only mode*/

Reading from an Open File(cont…)

```
if(fp==NULL){//checking if the file has opened or not
perror("fopen() failed\n");exit(1);}
int c;
while((c=fgetc(fp))!=EOF)     /*reading from file character by character till EOF*/
fputc(c,stdout);   /*printing character by character on stdout, could also be done using putc(c);*/
fclose(fp);   //closing the file
return 0;}
```

Reading from an Open File(cont…)

- When you execute the program like
  - `./a.out p1.c  //p1.c` is the name of this program file
- It will print the contents of file `p1.c` on screen

Reading from an Open File(cont…)

2. Reading Line by Line and displaying on `stdout` till `EOF`

- `fputs()` and `fgets()` are used for writing and reading respectively
- **Syntax**
- `fgets(char* s,int size,FILE* stream);`
- It reads `size` number of characters from file given as 3rd argument and stores it in string `s`
- If `fgets()` reaches end of line, it places `\n` character in `s` and then `\0` and then ends reading
- `int fputs(const char* s,FILE *stream);`
- It places the string `s` on the file given as the 2nd argument

Reading from an Open File(cont…)

/*The program reads from the file line by line and displays the data read on the screen*/

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char* argv[]){
  if(argc!=2){//checking if file name given or not
    printf("Invalid number of arguments entered.
Please enter filename to display its contents....\n");
    exit(1);
  }
   FILE* fp=fopen(argv[1],"r");
```

/*passing `argv[1]` for the file name, as user will pass file name in it, and opening the file in read only mode*/

Reading from an Open File(cont…)

```
  if(fp==NULL){      //checking if the file has opened or not
  perror("fopen() failed\n");exit(1);}
  char buff[512];
  while(fgets(buff,512,fp)!=NULL) /*reading from file line by line till EOF*/
     fputs(buff,stdout);        /*printing line by line on stdout, could also be done using puts(buff);*/
  fclose(fp);  //closing the file
  return 0;
}
```

Reading from an Open File(cont…)

- When you execute the program like
  - `./a.out p1.c` /*`p1.c` is the name of this program file*/
- It will print the contents of file `p1.c` on screen
- Reading and writing character by character is far slow than reading and writing line by line
- Now that we have done reading using Unformatted functions, let's write a program that writes to a file

Reading from an Open File(cont…)

/*The program takes a string from user and writes it to a file character by character*/

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(int argc,char* argv[]){
  if(argc!=2){
    printf("Invalid number of arguments entered. Please enter filename to display its contents....\n");
    exit(1);}
  FILE* fp=fopen(argv[1],"a");/*opening the file in append mode*/
  if(fp==NULL){
  perror("fopen() failed\n");exit(1);}
```

Reading from an Open File(cont…)

```
/*taking input from the user*/
  char name[20];
  printf("Enter you friends name: ");
  fgets(name,20,stdin);
  int len=strlen(name);
  name[len-1]='\0';/*replacing \n with NULL character*/
  int i=0;
/*writing character by character from index 0 till NULL
character*/
  while(name[i]!='\0')
    fputc(name[i++],fp);
  fputc('\n',fp);    /*placing newline character at the end*/
  fclose(fp);
  return 0;}
```

Reading from an Open File(cont…)

- When you execute the program like
  - `./a.out newFile.txt` /*`newFile.txt` is the name of this program file*/
- It will take input from user and write that to file character by character
- Now let's write a program that takes input from user and writes it to the file line by line

Reading from an Open File(cont…)

/*Program takes a string from user and writes it to a file line by line*/

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(int argc,char* argv[]){
  if(argc!=2){
    printf("Invalid number of arguments entered. Please enter filename to display its contents....\n");
    exit(1);
  }
  FILE* fp=fopen(argv[1],"a");//opening in append mode
```

Reading from an Open File(cont…)

```
if(fp==NULL){
perror("fopen() failed\n");exit(1);}
char name[20];
printf("Enter you friends name: ");
scanf("%[^\n]s",name); /*taking input using scanf(), it will automatically place \0 at the end*/
fputs(name,fp);    //writing the complete name
fputc('\n',fp);
printf("Done..Bye..Bye..\n");
fclose(fp);
return 0;}
```

Reading from an Open File(cont…)

- When you execute the program like
  - `./a.out newFile.txt` `/*newFile.txt` is the name of this program file*/
- It will take input from user and write that to file line by line
- Remember that the file may contain integers or floating point numbers, but we have read them all as characters
- If we want to perform any operations on the values, we may have to manually convert them to the appropriate datatype

Reading from an Open File(cont…)

❑Formatted Functions

- Formatted functions provide us with the ability that we do not have to convert from characters to integers or other datatypes manually, rather they do this for us
- The functions used here are
  - `fscanf();` //which is similar to `scanf()` with an additional initial argument and that is the file pointer from where we want to read
  - `fprintf();` //it is similar to `printf()` with an additional initial argument and that is the file pointer where we want to write

Reading from an Open File(cont…)

- **Syntax**
- `int fscanf(FILE* stream, const char* format,…);`
- `int fprintf(FILE* stream,const char* format,…);`

Reading from an Open File(cont…)

```
/*The program reads from a file using formatted functions*/
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char* argv[]){
  if(argc!=2){
    printf("Invalid number of arguments entered.
Please enter filename to display its contents....\n");
    exit(1);
  }
  FILE* fp=fopen(argv[1],"r");
   if(fp==NULL){
  perror("fopen() failed\n");exit(1);}
```

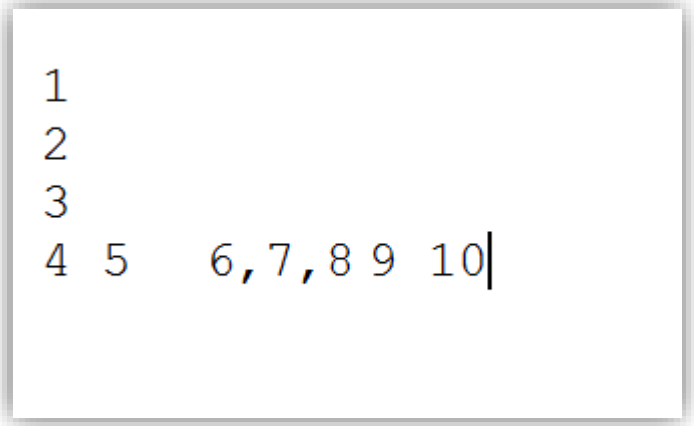
Reading from an Open File(cont…)

```
int n;
fscanf(fp,"%d,",&n);/*Here integer will be read from fp
and stored in n*/
while(feof(fp)==0){
  fprintf(stdout,"%d ",n);
  fscanf(fp,"%d,",&n);      /*comma after %d indicates
that read till a comma too, integers, by default, are read until a
space, tab or newline character occurs, here comma is specified
because the numbers may be comma separated*/
}
printf("\n");
fclose(fp);
return 0;}
```

Reading from an Open File(cont…)

- For the file picture shown, output of the above program is:

```
1 2 3 4 5 6 7 8 9 10
```



- You see that the numbers in the file are space, tab, comma and newline separated
- And here, they have been shown on a single line

Random Access

- Till now, we have read sequential access, i.e. we have read and written the file in a sequence as we cannot jump from one location to the other
- In random access, we can jump from one location to the other location in the file and then read/write there
- Some of the functions related to random access are

1. ftell()

- `long ftell(FILE* stream);`
- It will tell the current location of the file offset in the file whose pointer has been passed as argument

Random Access(cont…)

2. rewind()

- `void rewind(FILE *stream);`
- It will take the current file offset to the beginning of the file

3. fseek()

- `int fseek(FILE *stream, long offset, int whence);`
- First argument is the pointer to the file
- Second argument is the no. of bytes to jump
- Third argument is the location from where offset no. of bytes are to jump

Random Access(cont…)

- In simple words, it says that jump `(offset+whence)` no. of bytes in the file stream
- **Some constants for `whence` are**
- **SEEK_SET** i.e. from the start of the file
- e.g. `fseek(fp,0,SEEK_SET);` //where `fp` is the file pointer
- This statement is just like `rewind(fp);`
- `fseek(fp,50,SEEK_SET);`
- It says that jump `50` bytes from the start of the file
- **SEEK_CUR** i.e. the current position of the file offset

Random Access(cont…)

- `fseek(fp,50,SEEK_CUR);`
- It says that jump `50` bytes ahead from the current position of the file offset
- **SEEK_END** i.e. the end of the file
- `fseek(fp,50,SEEK_END);`
- It says that jump `50` bytes ahead from the end of the file
- In case of reading, if we try to read after jumping `50` bytes, it will create **error**
- However, in case of writing, it will work **OK**

Random Access(cont…)

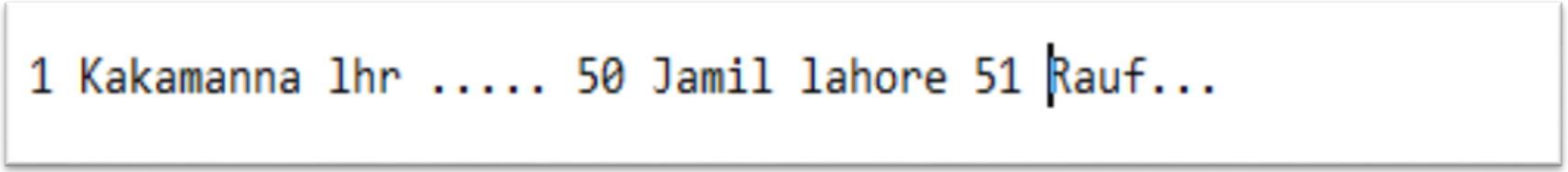
- When we jump `n` bytes ahead from the end, it will create a hole of `n` bytes in the file, which will be containing `NULL`
- However, this does not affect the size of the file
- If we copy this file to another, the new file will not be containing a hole in it and will have more size than the original file

Binary Files

- Binary files are used for reading files of custom formats i.e. the files other than the text format
- For example, `a.out` is a binary file
- You cannot read `a.out` using `cat` program because cat program has been written to read only text files
- **`readelf`** program can be used to read binary files
- Other examples of binary files include image and video files

Binary Files(cont…)

- Let's discuss a scenario is which we need to create a binary file
- For example, here is an image of a file



- The file contains records of different friends with their Record no., Name and City
- Now, for example, we want to change the city of Jamil from Lahore to Rawalpindi

Binary Files(cont…)

- Now as Lahore takes 6 characters and Rawalpindi takes 10 characters, so if we try to make changes in this file it will overwrite Rauf's data
- So, **one solution** is that we make new file, write in it all the record till Jamil as it is, then write Rawalpindi instead of Lahore and then write the remaining record
- After that delete the old file and renaming the new file with the previous file name
- Doing all this stuff is obviously not a good idea, as it is going to take a lot of time

Binary Files(cont…)

- A **better solution** to all this is, we use Binary files instead of text files
- In binary files, we will allocate fix space to each record
- Consider, for example, the following configuration

Record Number	Name	City	Total Bytes:
4 Bytes(int)	20 Bytes	30 Bytes	54

- Now, in this configuration we can easily move to a record number
- For example, to move to record number 11, we will use the following statement

Binary Files(cont…)

- `fseek(fp,11*54,SEEK_SET);`
- As each record requires `54` bytes and we are to move to the `11th` record so we are stepping `11*54` bytes ahead from the start
- The functions used for reading and writing to binary files are `fread()` and `fwrite()`
- **Syntax**
- `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`
- Here, first argument is the structure pointer, where `fread()` will store the record read from file

Binary Files(cont…)

- Second argument is the size of structure
- Third is the number of records to read
- Fourth is the `FILE*` pointer from where we will read the record
- `size_t fwrite(const void* ptr, size_t size, size_t nmemb, FILE *stream);`
- Here, arguments are just like the previous ones with the difference that here data read from the structure variable will be written to the file specified
- Let's write some program to understand use of `fread()` and `fwrite()`

Binary Files(cont…)

/*The program reads from a binary file using `fread()` and displays the data on screen*/

```
#include<stdio.h>
#include<stdlib.h>
struct Student{
  int id;
  char name[20];
  char address[30];};
int main(int argc,char* argv[]){
  if(argc!=2){
    printf("Invalid number of arguments entered.
Please enter filename to display its contents....\n");
    exit(1);    }
  struct Student s1;
```

Binary Files(cont…)

```
  FILE* fp=fopen(argv[1],"rb");
  if(fp==NULL){
  perror("fopen() failed\n");exit(1);}
/*fread() used to store 1 record from fp to s1*/
  fread((struct Student*)&s1,sizeof(s1),1,fp);
  while(!feof(fp)){//read till EOF
    printf("Student ID: %d\n",s1.id);
    printf("Name: %s\n",s1.name);
    printf("Address: %s\n",s1.address);
    fread((struct Student*)&s1,sizeof(s1),1,fp);}
  fclose(fp);
  return 0;}
```

Binary Files(cont…)

/*The program takes a student's data from user, stores it in a Student structure and saves the result in a binary file*/

```
#include<stdio.h>
#include<stdlib.h>
struct Student{
  int id;
  char name[20];
  char address[30];
};
int main(int argc,char* argv[]){
  if(argc!=2){
    printf("Invalid number of arguments entered.
Please enter filename to display its contents....\n");
```

Binary Files(cont…)

```
exit(1);
}
struct Student s1;
FILE* fp=fopen(argv[1],"wb");
if(fp==NULL){
perror("fopen() failed\n");exit(1);}
printf("Roll Number: ");
scanf("%d",&s1.id);
getchar();
printf("Name: ");
scanf("%[^\n]s",s1.name);
getchar();
```

Binary Files(cont…)

```
printf("Address: ");
scanf("%[^\n]s",s1.address);
fwrite((struct Student*)&s1,sizeof(s1),1,fp);
printf("Done..Bye Bye...\n");
fclose(fp);
return 0;}
```

SUMMARY