File Handling

CS10003 PROGRAMMING AND DATA STRUCTURES

What is a file?

A named collection of data, stored in secondary storage (typically).

Typical operations on files:

- Open
- Read
- Write
- Close

How is a file stored?

- Stored as sequence of bytes, logically contiguous (may not be physically contiguous on disk).

File Types

- The last byte of a file contains the end-of-file character (`EOF`), with ASCII code 1A (hex).
- While reading a text file, the EOF character can be checked to know the end.

Two kinds of files:

- Text :: contains ASCII codes only
- Binary :: can contain non-ASCII characters
  - Image, audio, video, executable, etc.
  - To check the end of file here, the *file size* value (also stored on disk) needs to be checked.

File handling in C

In C we use `FILE *` to represent a pointer to a file.

`fopen` is used to open a file. It returns the special value `NULL` to indicate that it is unable to open the file.

```
FILE *fptr;
char filename[ ]= "file2.dat";

fptr = fopen (filename,"w");

if (fptr == NULL) {
  printf (“ERROR IN FILE CREATION”);
    /* DO SOMETHING */
}
```

Modes for opening files

The second argument of `fopen` is the *mode* in which we open the file. There are three basic modes.

"r" opens a file for reading.

- "r+" allows write

"w" creates a file for writing and writes over all previous contents (deletes the file, so be careful!).

- "w+" allows read

"a" opens a file for appending – writing at the end of the file.

- "a+" allows read

Binary Files

We can add a 'b' character to indicate that the file is a *binary* file.

- “rb”, “wb” or “ab”

```
fptr = fopen (“xyz.jpg”, “rb”);
```

The exit( ) function

Sometimes error checking means we want an *emergency exit* from a program.

In `main()` we can use `return` to stop the program.

In any function we can use `exit()` to do this.

Exit is part of the stdlib.h library.

```
exit(0);
```

exits the program

Usage of exit( )

```
FILE *fptr;
char filename[ ]= "file2.dat";
fptr = fopen (filename,"w");

if (fptr == NULL) {
  printf (“ERROR IN FILE CREATION\n”);
  exit(0);
}
```

Writing to a file using fprintf( )

`fprintf()` works just like `printf()` and `sprintf()`

except that its first argument is a file pointer.

```
int a=10, b=5;
FILE *fptr;
fptr = fopen ( "file.dat", "w“ );

fprintf (fptr, "Hello World!\n");
fprintf (fptr, “%d %d”, a, b);
```

Reading Data Using fscanf( )

```
int x, y;
FILE *fptr;
fptr = fopen (“input.dat”, “r”);

fscanf (fptr, “%d%d”, &x, &y);
```

The file pointer moves forward with each read operation

Reading lines from a file using fgets( )

We can read a string using `fgets()`.

```
FILE *fptr;
char line [1000];
……                                          /* Open file and check it is open */
while (fgets(line, 1000, fptr) != NULL)
{
   printf ("We have read the line: %s\n", line);
}
```

fgets( ) takes 3 arguments – a string, maximum number of characters to read, and a file pointer.
It returns NULL if there is an error (such as EOF).

Closing a file

We can close a file simply using `fclose()` and the file pointer.

```
FILE *fptr;
char filename[ ]= "myfile.dat";

fptr = fopen (filename,"w");

if (fptr == NULL) {
    printf ("Cannot open file to write!\n");
    exit(0);
}

fprintf (fptr,"Hello World of filing!\n");
fclose (fptr);
```

Random Access using fseek()

fseek() can be used to set the position of a file pointer (say, fp).

```
int fseek(FILE *fp, long int offset, int whence)
```

New position specified by 2 more arguments – offset (specified in bytes) and whence.

whence can take one of 3 values:

- SEEK_END end of the file
- SEEK_SET beginning of the file
- SEEK_CUR current position of the file pointer (also returned by ftell(fp))

Example – fseek() and ftell()

```
int main(){
  char c; FILE *fp;
  fp=fopen("file1.dat", "r+");
  printf("\n%ld\n", ftell(fp));
  c=fgetc(fp); c=fgetc(fp);
  printf("%ld\n", ftell(fp));
  fseek(fp, 2, SEEK_CUR);
  printf("%ld\n", ftell(fp));
  fputs("fast purple",fp);
  printf("%ld\n\n", ftell(fp));
  fclose(fp);
  return 0;
}
```

Output:

```
0
2
4
15
```

Contents of file1.dat

Before: the quick brown fox jumped over the lazy dogs
After: the fast purple fox jumped over the lazy dogs

Three special streams

Three special file streams are defined in the `<stdio.h>` header

- stdin reads input from the keyboard
- stdout send output to the screen
- stderr prints errors to an error device (usually also the screen)

What might this do?

```
fprintf (stdout,"Hello World!\n");
```

An example program

```
#include <stdio.h>
main( )
{
     int i;

    fprintf(stdout,"Give value of i \n");
    fscanf(stdin,"%d",&i);
    fprintf(stdout,"Value of i=%d \n",i);
    fprintf(stderr,"No error: But an example to  show error message.\n");
 }
```

Output:

```
Give value of i
15
Value of i=15
No error: But an example to show error message.
```

Input File & Output File redirection

One may redirect the standard input and standard output to other files (other than `stdin` and `stdout`).

Usage: Suppose the executable file is `a.out`:

```
$ ./a.out <in.dat  >out.dat
```

`scanf()` will read data inputs from the file "`in.dat`", and `printf()` will output results on the file "`out.dat`".

A Variation

```
$ ./a.out <in.dat  >>out.dat
```

`scanf()` will read data inputs from the file "`in.dat`", and
`printf()` will append results at the end of the file "`out.dat`".

Reading and Writing a character

A character reading/writing is equivalent to reading/writing a byte.

```
int getchar( );
int putchar(int  c);
```

} stdin, stdout

```
int fgetc(FILE *fp);
int fputc(int c, FILE *fp);
```

} file

Example:

```
char c;
c = getchar();
putchar(c);
```

A Digression:
Command Line Arguments

What are they?

A program can be executed by directly typing a command at the operating system prompt.

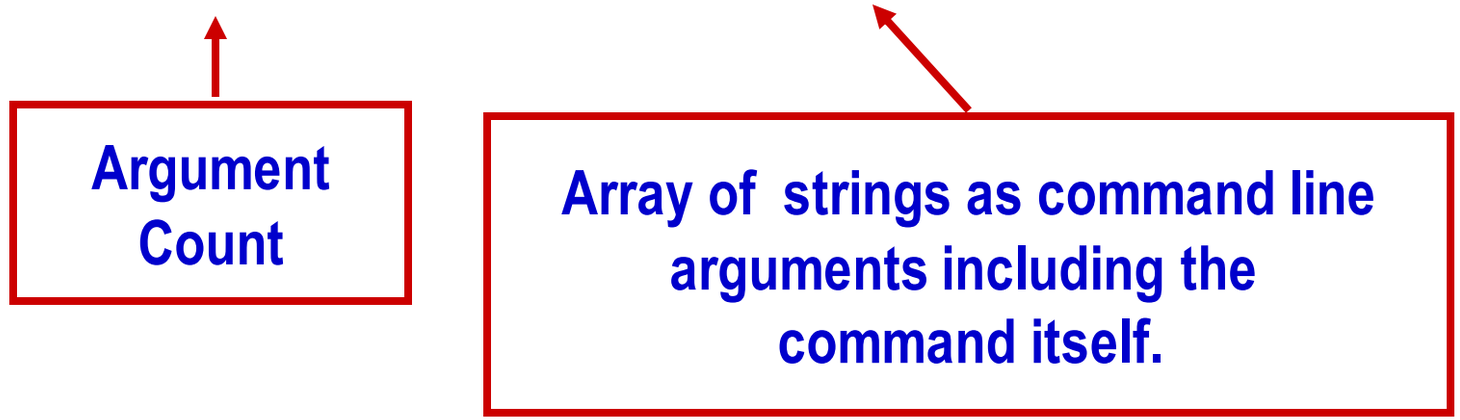
```
$ cc -o test test.c
$ ./a.out in.dat out.dat
$ prog_name param_1 param_2 param_3 ..
```

- The individual items specified are separated from one another by spaces.
  - First item is the program name.
- Variables *argc* and *argv* keep track of the items specified in the command line.

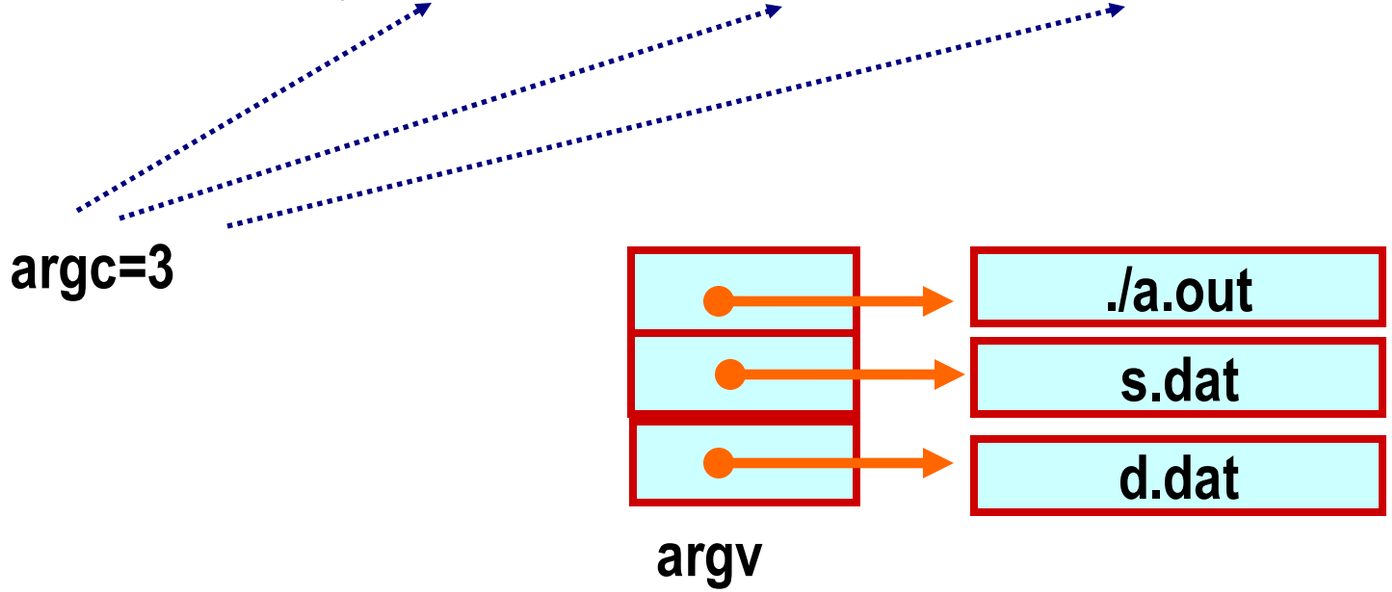
How to access them?

Command line arguments may be passed by specifying them under `main()`.

```
int main (int argc, char *argv[]);
```



```
$  ./a.out  s.dat  d.dat
```



argv[0] = “./a.out” argv[1] = “s.dat” argv[2] = “d.dat”

Back to File Handling

Example: Program for Copying a File

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

int main( int argc, char *argv[ ] )
{
     FILE *ifp, *ofp;
     int i, c;
     char src_file[100], dst_file[100];

     if (argc!=3) {
                    printf ("Usage: ./a.out <src_file> <dst_file> \n");  exit(0);
     }
    else {
                    strcpy (src_file, argv[1]);   strcpy (dst_file, argv[2]);
}
```

Example: contd.

```
if ((ifp = fopen(src_file,"r")) == NULL)  {
    printf (“Input File does not exist.\n");   exit(0);
 }

 if ((ofp = fopen(dst_file,"w")) == NULL)  {
    printf (“Output File not created.\n");   exit(0);
 }

 while ((c = fgetc(ifp)) != EOF)  fputc (c,ofp);    // This is where the copying is done

 fclose(ifp);  fclose(ofp);
}
```

Practice Problems

1. Write a program that uppercase characters, lowercase characters, digits, spaces (including tabs) and newlines in a file.
2. Write a program that reads a 2-d array of integers from a file and replaces the contents of the file with the transpose of the matrix represented by the 2-d array.
3. Write a program that reads student records containing name (string), roll_number (int), CGPA (float) from the user and writes them in a file, one record per line.
4. Write a program that reads the records written by the above program into an array of structures. The structure should contain name, roll_number and CGPA as members.
5. Write a program that takes as a command line argument a C program filename and outputs the number of occurrences of the keywords int, float, double, long, short in the file.