First Year First Semester Course

M.Tech. (CS) [Batch 2021-23]

Lecture #09

Introduction to Programming

File Handling in C

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

Advantages of File handling

- At times size of the program input is very large.
- During the testing phase providing inputs in the interactive way is tedious.
- The size of the program output may be large enough and will not fit in a single screen.
- You may wish to store the output for future analysis.

File handling in C

- A file ***needs to be opened*** first for any input/output operations on the file.
- It may be opened for ***reading/writing/appending***.
- The file ***must be closed*** once the use/handling of the file is over.
- In between the address of the file will be stored in a pointer data type viz., FILE *.

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 a pointer to the file if successfully opened the file else it returns **NULL**.

"r" opens a file for reading.
"w" creates a file for writing
"a" opens a file for appending

```
FILE *fptr;
char filename[]= "file1.dat";
fptr= fopen (filename,"w");
if (fptr == NULL) {
        printf ("ERROR IN FILE CREATION"); /* DO SOMETHING */
}
```

Closing a file

- A file must be closed as soon as all the operations on it have been completed.
- 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(-1);
}
fprintf (fptr,"Hello World of file handling!\n");
fclose (fptr);
```

Writing to a file using fprintf( )

- ***fprintf()*** works just like ***printf*** except that its first argument is a file pointer.

- General form:

```
fprintf(fp, "control string", list);
```

```
FILE *fptr;
int year=2021;
fptr= fopen("file.dat","w");
fprintf (fptr,"Hello World! %d\n",year);
```

Reading Data Using fscanf( )

- ***fscanf()*** works just like scanf except that its first argument is a file pointer.
- ***General form:***

```
fscanf(fp, "control string", list);
```

```
FILE *fptr;
fptr= fopen ("input.dat","r");
/* Check it's open */
if (fptr==NULL)
{
        printf("Error in opening file \n");
}
fscanf(fptr,"%d%d",&x,&y);
```

Other Functions for Handling File


```
putc(c,fp1);
```

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

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

→ Reads a character from the file associated with the **FILE** pointer ***fp2***

```
while((c=getc(f1))!=EOF)
        printf("%c",c);
```

- Integer-oriented file I/O functions

```
putw(integer,fp);
getw(fp);
```

Error Handling During I/O Operations

- Typical error situations:
  - Trying to read beyond the end-of-file mark
  - Trying to use a file that has not been opened
  - Trying to perform an operation on a file, when the file is opened for another type of operation
  - Opening a file with an invalid filename etc.

feof() → can be used to test for an end of file condition

```
if(feof(fp))
        printf("End of data….\n");
```

ferror() → returns nonzero integer if an error has been detected

```
if(ferror(fp)!=0)
        printf("An error has occurred….\n");
```

Random Access to Files

- **rewind()**: Sets the position to the beginning of the file

```
rewind(fp)
```

- **fseek()**: Sets the position to a desired point in the file

  General form:

  fseek(file_ptr, offset, position)

- **ftell()**: Gives the current position in the file (in terms of bytes from the start)

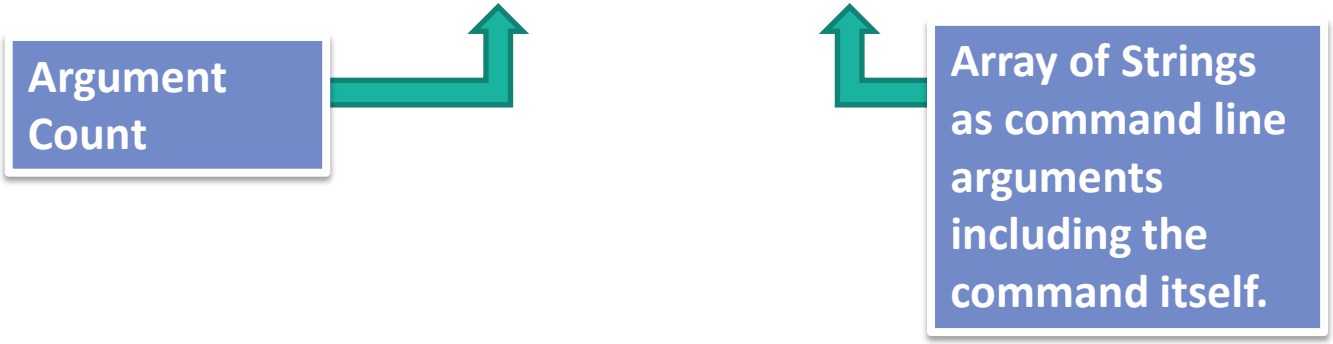
```
n=ftell(fp);
```

Command Line Arguments

Command Line Arguments

- Command line arguments can be passed by specifying them under main( ).

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



Passing parameters to main()

- Two parameters will be passed to function main() through command line – **argc** and **argv**.
- Name of the parameters are fixed.
- **argc** is of integer type and it stores the number of parameters (delimited by space) in the command line.
- **argv** is a 2D array of characters and it stores all the words in the command line.
- By default all the parameters are taken as array of characters (strings)

Passing parameters to main()


```
#include <stdio.h>
int main(int argc, char *argv[])
{
        int i;
        for(i=0;i<argc;i++) {
                printf("%d: %s\n",i,argv[i]);
        }
        return 0;
}
```

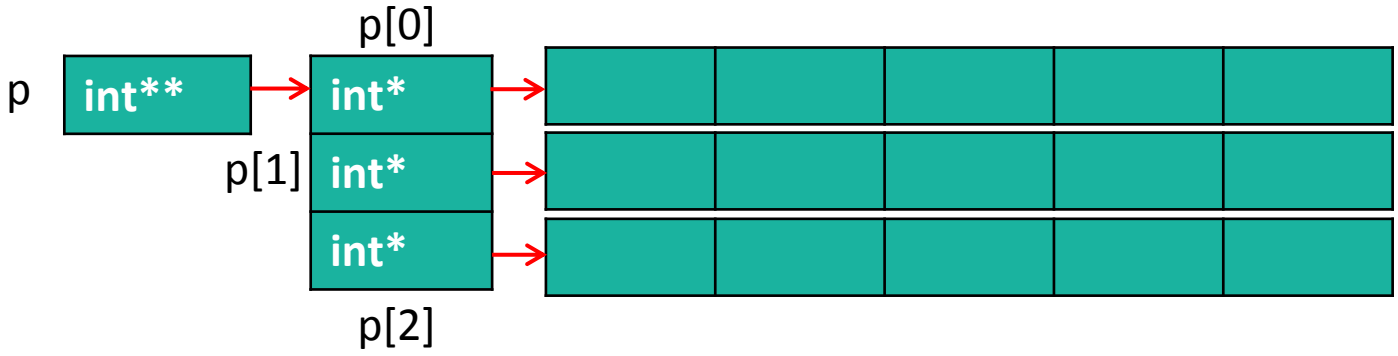
```
$ cc -Wall week14_cmdline.c
$ ./a.out
0: ./a.out
$ ./a.out Hello
0: ./a.out
1: Hello
```

```
$ ./a.out Indian Statistical Institute
0: ./a.out
1: Indian
2: Statistical
3: Institute
```

More on Pointers and Structures

Pointer to Pointer


```
int **p;
p=(int **) malloc(3 * sizeof(int *));
p[0]=(int *) malloc(5 * sizeof(int));
p[1]=(int *) malloc(5 * sizeof(int));
p[2]=(int *) malloc(5 * sizeof(int));
```



Structures within Structures

- Nesting of structures is permitted in C

```
struct salary
{
        char name;

        char department;

        int dearness_ allowance;

        int house_rent_ allowance;

        int city_ allowance;

}employee;
```

```
struct salary
{
        char name;
        char department;
        struct
        {
                int dearness;
                int house_rent;
                int city;

        }allowance;
}employee;
```

Questions?