CS19001: Programming and Data Structures Laboratory

Aritra Hazra
CSE, IIT Kharagpur

http://cse.iitkgp.ac.in/~aritrah/course/lab/PDS/Autumn2019/

18-Oct-2019

Files

- Help in permanent storage of data in HDD
- Data stored as sequence of bytes, "logically" contiguous
- The last byte of a file contains the end-of-file character (EOF)
- Two types –
  - text (ASCII only, EOF terminated),
  - binary (Can have non-ASCII chars)
- Basic Operations – Open/Close, Read/Write

Files

```
FILE *fptr;
char filename[ ]= "file2.dat";
fptr = fopen (filename,"w");
if (fptr == NULL) {
      printf ("ERROR IN FILE CREATION");
      /* DO SOMETHING */
}
```

File opening modes

The second argument of fopen is the mode in which we open the file.

- "r" : opens a file for reading (can only read). Error if the file does not already exists
- "r+" : allows write also
- "w" : creates a file for writing (can only write). Will create the file if it does not exist.
  Caution: writes over all previous contents if the file already exists
- "w+" : allows read also
- "a" : opens a file for appending (write at the end of the file)
- "a+" : allows read also

Writing to a file

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

Reading from a file

```
FILE *fptr;
fptr = fopen ("input.dat", "r");
/* Check whether it is open */
if (fptr == NULL)
  {
     printf("Error in opening file \n");
     exit(-1);
  }
fscanf (fptr,"%d %d",&x, &y);
```

Reading from a file

```
char ch;
while (fscanf(fptr,"%c", &ch) != EOF)
{
    /* not end of file; read */
}
/* EOF checking in loop */
```

Reading lines from a file

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

- Takes three parameters : a character array "str", maximum number of characters to read "size", and a file pointer "fp"
- Reads until any one of these happens - (i) number of characters read = size - 1, (ii) '\n' is read (char '\n' is added to "str"), (iii) *EOF* is reached or an error occurs
- '\0' added at end of "str" if no error. Returns *NULL* on error or *EOF*, otherwise returns pointer to "str"

Writing lines to a file

- use `fputs()`
- Takes two parameters : A string "str" (null terminated) and a file pointer "fp"
- Writes the string pointed to by "str" into the file
- Returns non-negative integer on success, *EOF* on error

Reading/Writing a character

```
//as declared in header
char fgetc(FILE *fp);

//as declared in header
int fputc(char c, FILE *fp);

//usage
char c;
c = fgetc(fp1);
fputc(c, fp2);
```

Moving File Pointers


```
//as declared in header
int fseek(FILE *stream, long int offset,
int whence)
```

The function takes the following three arguments and moves the file pointer an *offset* number of bytes from *whence*.

- stream: Pointer to a FILE object
- offset: Number of bytes offset from whence.
- whence: Constants that point to a position of the file
  1. SEEK_SET: Beginning of file
  2. SEEK_CUR: Current position of file
  3. SEEK_END: End of file

Moving File Pointers Example

Consider a file containing "abcdefghijklmnop"

```
fseek(fp,4,SEEK_SET);
fscanf(fp,"%c",&c);
printf("%c\n",c);
fseek(fp,3,SEEK_CUR);
fscanf(fp,"%c",&c);
printf("%c\n",c);
fseek(fp,-2,SEEK_CUR);
fscanf(fp,"%c",&c);
printf("%c\n",c);

//Output
e
h
f
```

Locate file pointer position


```
long ftell(FILE *stream);
```

The function obtains the current value of the position of the file pointer *stream*(in terms of number of bytes) with respect to the starting of the file.
Consider the same file containing "abcdefghijklmnop"

```
fgets(string,10,fp);
printf("%ld", ftell(fp));
//Output
9
```

Closing a file

```
FILE *fptr;
char filename[]= "myfile.dat";
fptr = fopen (filename,"w");
fprintf (fptr,"Hello World of filing!\n");
... ... ... ... ...
fclose (fptr);
```

- Should close a file when no more read/write to a file is needed in the rest of the program

Command Line Arguments

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

- User input:
  `./a.out s.dat d.dat`
- Effect:
  argc=3,
  argv[0] = ./a.out
  argv[1] = s.dat
  argv[2] = d.dat

Convert Argument string to usable data

```
int main(int argc, char *argv[ ]) {
  int i, n1, n2;
  printf("No. of arg is %d\n", argc);
  for (i=0; i<argc; ++i)
    printf("%s\n", argv[i]);
  sscanf(argv[1], "%d", &n1);
  sscanf(argv[2], "%d", &n2);
  printf("Sum is %d\n", n1+n2);
  return 0;
}
```

Execution:

```
$ ./a.out  32  54
No. of arg is 3
./a.out
32
54
Sum is 86
```

Bitwise Operators

Bitwise operators perform logical operations (OR, AND, XOR) at the bit-level. Operators include:

- '&' → bitwise-AND; '|' → bitwise-OR; '^'→ bitwise-XOR;
- '∼' → bitwise-NOT; '>>' → right-shift; '<<' → left-shift.

```
int a=7, b=9; //a:00000111, b:00001001
printf("%d", a|b)//Output is 15(00001111)
int a=0, b=1; //a:0000000, b:00000001
printf("%d\n", a&b) //Output is 0(00000000)
```

Aritra Hazra
CSE, IIT
Kharagpur

Thank You