FSCANF() AND FPRINTF()

- similar to scanf() and printf()
- in addition provide file-pointer
- given the following
  - file-pointer f1 (points to file opened in write mode)
  - file-pointer f2 (points to file opened in read mode)
  - integer variable i
  - float variable f
- Example:

```
fprintf(f1, “%d %f\n”, i, f);
fprintf(stdout, “%f \n”, f);      /*note: stdout refers to screen */
fscanf(f2, “%d %f”, &i, &f);
```

- fscanf returns EOF when end-of-file reached

GETW() AND PUTW()

- handle one integer at a time
- syntax: putw(i,fp1);
  - i : an integer variable
  - fp1 : pointer to file ipened with mode **w**
- syntax: i = getw(fp2);
  - i : an integer variable
  - fp2 : pointer to file opened with mode **r**
- file pointer moves by one integer position, data stored in binary format native to local system
- getw() returns end-of-file marker EOF when file end reached

C PROGRAM USING GETW, PUTW, FSCANF, FPRINTF

```
#include <stdio.h>
main()
{ int i,sum1=0;
  FILE *f1;
  /* open files */
  f1 = fopen("int_data.bin","w");
  /* write integers to files in binary
    and text format*/
for(i=10;i<15;i++)              putw(i,f1);
fclose(f1);
f1 = fopen("int_data.bin","r");
   while((i=getw(f1))!=EOF)
    {   sum1+=i;
      printf("binary file: i=%d\n",i);
    } /* end while getw */
printf("binary sum=%d,sum1);
 fclose(f1);
}
```

```
#include <stdio.h>
main()
{  int i, sum2=0;
  FILE *f2;
  /* open files */
  f2 = fopen("int_data.txt","w");
  /* write integers to files in binary
    and text format*/
for(i=10;i<15;i++) printf(f2,"%d\n",i);
fclose(f2);
f2 = fopen("int_data.txt","r");
while(fscanf(f2,"%d",&i)!=EOF)
    { sum2+=i; printf("text file:
    i=%d\n",i);
    } /*end while fscanf*/
  printf("text sum=%d\n",sum2);
  fclose(f2);
}
```

On execution of previous Programs

```
$ ./a.out
binary file: i=10
binary file: i=11
binary file: i=12
binary file: i=13
binary file: i=14
binary sum=60,
$ cat int_data.txt
10
11
12
13
14
```

```
$ ./a.out
text file: i=10
text file: i=11
text file: i=12
text file: i=13
text file: i=14
text sum=60
$ more int_data.bin
^@^@^@^K^@^@^@^L^@^@^
   @^M^@^@^@^N^@^@^@
$
```

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

Error handling

- given file-pointer, check if EOF reached, errors while handling file, problems opening file etc.
- check if EOF reached: feof()
- feof() takes file-pointer as input, returns nonzero if all data read and zero otherwise

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

- ferror() takes file-pointer as input, returns nonzero integer if error detected else returns zero

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

Error while opening file

- if file cannot be opened then fopen returns a NULL pointer

- Good practice to check if pointer is NULL before proceeding

```
fp = fopen("input.dat", "r");

if (fp == NULL)
        printf("File could not be opened \n ");
```

Random access to files

- how to jump to a given position (byte number) in a file without reading all the previous data?
- fseek (file-pointer, offset, position);
- position: 0 (beginning), 1 (current), 2 (end)
- offset: number of locations to move from position

  Example: fseek(fp,-m, 1); /* move back by m bytes from current

  position */

  fseek(fp,m,0); /* move to (m+1)th byte in file */

  fseek(fp, -10, 2); /* what is this? */

- ftell(fp) returns current byte position in file
- rewind(fp) resets position to start of file

Command line arguments

- can give input to C program from command line
  E.g. > prog.c 10 name1 name2 ….
- how to use these arguments?
  main (int argc, char *argv[])
- argc – gives a count of number of arguments (including program name)
- char *argv[] defines an array of pointers to character (or array of strings)
- argv[0] – program name
- argv[1] to argv[argc -1] give the other arguments as strings

Example args.c

```
#include <stdio.h>

main(int argc,char *argv[])
{
  while(argc>0)      /* print out all arguments in reverse order*/
    {
      printf("%s\n",argv[argc-1]);
      argc--;
    }
}
```

```
$ cc args.c -o args.out
$ ./args.out 2  join leave 6
6
leave
join
2
./args.out
$
```