File Access

Standard I/O

So far all our I/O operations are read from the standard input (`stdin` - keyboard) and write to the standard output (stdout - VDU) devices. These are defined and connected to the running program (process) by the OS.

stdin & stdout

In Unix/Linux operating systems these input and output streams are treated as two files `stdin` and `stdout` respectively. The `printf()`, `scanf()` functions and the `getchar()`, `putchar()` functions/macros are used to access these files.

File Access Functions

But we like to open other files for read, write or append. Following C library functions are used for this purpose: `fopen()`, `fclose()`, `fseek()`, `fprintf()`, `fscanf()`, `getc()`, `putc()` etc.

Example 1

```
#include <stdio.h>
#include <string.h>
#define MAXNO 100
#define ROLL 9
#define NAME 51

struct studData { // fileOpen1.c
        char rollNo[ROLL] ;
        char name[NAME] ;
        float cgpa ;
};
#define NOTFOUND -1
```

```
int binarySearch(struct studData data[],
                 int lInd,
                 int rInd,
                 char rollNo[]) {

    while(lInd < rInd) {
          int mInd ;

          mInd = (lInd + rInd)/2 ;
          if(strcmp(rollNo, data[mInd].rollNo) <= 0)
               rInd = mInd ;
          else lInd = mInd + 1;
    }
    if(strcmp(rollNo, data[lInd].rollNo) == 0)
```

```
        return lInd ;
    return NOTFOUND ;
}
int main()
{
    int noOfStdnt, i, index ;
    struct studData data[MAXNO] ;
    char rollNo[ROLL] ;
    FILE *fp ;

    fp = fopen("openDat", "r");
    fscanf(fp, "%d", &noOfStdnt);
    for(i=0; i<noOfStdnt; ++i) {
        fscanf(fp, "%s", data[i].rollNo);
```

```
        fscanf(fp, " %[^0-9]", data[i].name);
        fscanf(fp, "%f", &data[i].cgpa);
    }

    for(i=0; i<noOfStdnt; ++i) {
        printf("%s ", data[i].rollNo);
        printf(" %s", data[i].name);
        printf(" %5.2f", data[i].cgpa);
        putchar('\n');
    }
    putchar('\n');

    printf("Enter the Roll No.: ");
    fscanf(stdin, "%s", rollNo);
```

```
    index = binarySearch(data, 0, noOfStdnt-1, rollNo)
    if(index == NOTFOUND) printf("\nStudent not present
    else printf("\n%s %s %f\n", data[index].rollNo,
                                data[index].name,
                                data[index].cgpa);
    fclose(fp);
    return 0;
}
```

Data File: openDat

```
10
02CE2008 A. Maria Watson 8.00
02CH2002 P. K. Singh 8.00
02CH2007 L. P. Yadav 6.50
02CS2005 P. Baluchandran 9.25
02CS2010 D. K. Sarlekar 7.50
02EC2006 V. K. R. V. Rao 9.00
02MA2003 Imtiaz Ali 8.50
02NA2004 S. P. Sengupta 8.25
02PH2001 V. Bansal 7.50
02SI2009 S. V. Reddy 7.00
```

```
FILE *fp
```

The structure `FILE` is defined in `stdio.h`. The pointer `fp` will point to an object of type `FILE`, that will hold information regarding an opened file e.g. file buffer, current byte position, read/write permission, end-of-file status, error etc.

```
fp=fopen("openDat","r");
```

```
FILE *fopen(const char *path, const
char *mode);
```

The C library function `fopen()` is used to open a named file in read `"r"`, write `"w"` (if there is an old file that is truncated), append `"a"`, read-write `"r+"` (old file is not truncated) etc. modes. The function returns the `FILE` pointer for a successful open, else it returns `NULL`.

```
fscanf(fp,"%s", data[i].rollNo);
```

```
int fscanf(FILE *stream, const char
*format, ...);
```

This function is similar to `scanf()` except the first parameter which is a `FILE` pointer of an open file. It will behave like `scanf()` if the file pointer is `stdin`.

```
fscanf(stdin, "%s", rollNo);
```

fclose(fp)

```
int fclose(FILE *stream);
```

It dissociates the named file from the file structure. The function returns 0 on a successful completion, otherwise it returns `EOF`.

Example 2

```
#include <stdio.h>
#include <string.h>
#define MAXNO 100
#define ROLL 9
#define NAME 51

struct studData {
        char rollNo[ROLL] ;
        char name[NAME] ;
        float cgpa ;
};
int main() // fileOpen2.c
```

```
{
    int noOfStdnt, i ;
    struct studData data[MAXNO] ;
    FILE *fpO, *fpI ;

    fpI = fopen("openDat", "r");
    fpO = fopen("outDat", "w");
    fscanf(fpI, "%d", &noOfStdnt);
    for(i=0; i<noOfStdnt; ++i) {
        fscanf(fpI, "%s", data[i].rollNo);
        fscanf(fpI, " %[^0-9]", data[i].name);
        fscanf(fpI, "%f", &data[i].cgpa);
    }
    for(i=0; i<noOfStdnt; ++i) {
```

```
            fprintf(fpO, "%s ", data[i].rollNo);
            fprintf(fpO, " %s", data[i].name);
            fprintf(fpO, " %5.2f", data[i].cgpa);
            putc('\n', fpO);
        }
        fclose(fpI); fclose(fpO);
        return 0;
}
```

outDat

```
02CE2008  A. Maria Watson   8.00
02CH2002  P. K. Singh   8.00
02CH2007  L. P. Yadav   6.50
02CS2005  P. Baluchandran   9.25
02CS2010  D. K. Sarlekar   7.50
02EC2006  V. K. R. V. Rao   9.00
02MA2003  Imtiaz Ali   8.50
02NA2004  S. P. Sengupta   8.25
02PH2001  V. Bansal   7.50
02SI2009  S. V. Reddy   7.00
```

Note

`fpO = fopen("outDat", "w");` - opens the `outDat` file in output mode and returns the file pointer `fpO`.

`fprintf(fpO, "%s ", data[i].rollNo);` - similar to `printf()`, the first parameter is the file pointer.

Note

`putc('\n', fp0);` - similar to `putchar()`, the second parameter is the file pointer.

`int fgetc(FILE *stream);` is similar to `getchar()`.

Example 3

```
#include <stdio.h>
#include <string.h>
#define MAXNO 100
#define ROLL 9
#define NAME 51

struct studData {
        char rollNo[ROLL] ;
        char name[NAME] ;
        float cgpa ;
};
int main() // fileOpen3.c
```

```
{
    int noOfStdnt, i ;
    struct studData data[MAXNO] ;
    FILE *fp ;
    char roll[ROLL], name[NAME];
    float cgpa;

    fp = fopen("openDat", "a");
    printf("Enter roll no., name and cgpa:\n");
    scanf("%s %[^0-9] %f", roll, name, &cgpa);
    fprintf(fp, "%s %s %5.2f\n", roll, name, cgpa);
    fclose(fp);

    fp = fopen("openDat", "r");
```

```
fscanf(fp, "%d", &noOfStdnt);
++noOfStdnt;
for(i=0; i<noOfStdnt; ++i) {    // one more student
    fscanf(fp, "%s", data[i].rollNo);
    fscanf(fp, " %[^0-9]", data[i].name);
    fscanf(fp, "%f", &data[i].cgpa);
}
for(i=0; i<noOfStdnt; ++i) {
    printf("%s ", data[i].rollNo);
    printf(" %s", data[i].name);
    printf(" %5.2f", data[i].cgpa);
    putchar('\n');
}
putchar('\n');
```

```
    fclose(fp);
    return 0;
}
```

Note

The file is opened in append mode `fp = fopen("openDat", "a");` and new data is written at the end. But the initial count cannot be updated. The file is closed and reopened in read mode:

```
fp = fopen("openDat", "r");
```

Example 4

```
#include <stdio.h>
#include <string.h>
#define MAXNO 100
#define ROLL 9
#define NAME 51

struct studData {
        char rollNo[ROLL] ;
        char name[NAME] ;
        float cgpa ;
};
int main() // fileOpen4.c
```

```
{
    int noOfStdnt, i ;
    struct studData data[MAXNO] ;
    FILE *fp ;
    char roll[ROLL], name[NAME];
    float cgpa;

    printf("Enter roll no., name and cgpa:\n");
    scanf("%s %[^0-9] %f", roll, name, &cgpa);
    fp = fopen("openDat", "r+");
    fseek(fp, 0L, SEEK_END);
    fprintf(fp, "%s %s %5.2f\n", roll, name, cgpa);
    fseek(fp, 0L, SEEK_SET);
    fscanf(fp, "%d", &noOfStdnt);
```

```
++noOfStdnt;
fseek(fp, 0L, SEEK_SET);
fprintf(fp, "%d", noOfStdnt);

for(i=0; i<noOfStdnt; ++i) {    // one more student
    fscanf(fp, "%s", data[i].rollNo);
    fscanf(fp, " %[^0-9]", data[i].name);
    fscanf(fp, "%f", &data[i].cgpa);
}
for(i=0; i<noOfStdnt; ++i) {
    printf("%s ", data[i].rollNo);
    printf(" %s", data[i].name);
    printf(" %5.2f", data[i].cgpa);
    putchar('\n');
```

```
    }
    putchar('\n');
    fclose(fp);
    return 0;
}
```

fseek()

```
int fseek(FILE *stream, long offset,
int whence);
```

The function sets the file position. The second parameter specifies the offset. The third parameter specifies the place from where the offset is measured. SEEK_CUR - from the current position, SEEK_SET - from the beginning and SEE_END - from the end.

Note

The file is opened in read/write mode, `fp = fopen("openDat", "r+");` the position is set to the end, `fseek(fp, 0L, SEEK_END);` and new data is written. The file is repositioned at the beginning, the record count is read and updated.