File Input and Output in C

The identifier `FILE` is defined in `stdio.h` as a particular structure, with members that represent the current state of a file. To use files, a programmer does not need to know any details concerning this structure. Three file pointers `stdin`, `stdout`, and `stderr` are also defined in `stdio.h`. Even though they are pointers, we usually refer to them as files.

Written in C	Name	Remark
`stdin`	standard input	connected to the keyboard
`stdout`	standard output	connected to the screen
`stderr`	standard error	connected to the screen

The prototypes for file-handling functions are given in `stdio.h`. Here are the prototypes for `fprintf()` and `fscanf()` given in `stdio.h`:

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

A statement of the form

```
fprintf(file_ptr, control_string, other_arguments);
```

writes to the file pointed to by `file_ptr`. The conventions for `control_string` and `other_arguments` conform to those of `printf()` described earlier. In particular, the statement

`fprintf(stdout, ... );` is equivalent to `printf( ... );`

Similarly, a statement of the form

```
fscanf(file_ptr, control_string , other_arguments);
```

reads from the file pointed to by `file_ptr`. The conventions for `control_string` and `other_arguments` conform to those of `scanf()` described earlier. In particular, the statement

`fscanf(stdin, ... );` is equivalent to `scanf( ... );`

Thus, the two blocks of statements in the following code segment are equivalent.

```
 int p;
//-------------------------
 printf("Enter p :");
 scanf("%d",&p);
 printf("p=%d\n",p);
//-------------------------
 fprintf(stdout,"Enter p :");
 fscanf(stdin,"%d",&p);
 fprintf(stdout,"p=%d\n",p);
//-------------------------
```

As mentioned earlier, `stderr` is used for error message output. It generates the error message that appears on the output devices, not elsewhere. The operating system directly links `stderr` to either a Windows terminal or a Unix terminal. Consider the following program:

```
#include <stdio.h>
int main()
{
printf("A for Apple\n");
fprintf(stderr,"B for Ball\n");
fprintf(stdout,"C for Cat\n");
return(0);
}
```

Listing 69: C Program "errf.c"

If we compile and run the code, then we find the following output:

```
A for Apple
B for Ball
C for Cat
```

However, we can redirect the output to a new file. For example, if we type the following in the terminal

$ ./a.out > outp.dat ↩

then we observe the following output in the terminal:

```
B for Ball
```

On the other hand, we observe that a new file "outp.dat" has been created. To see its content, we execute $ cat outp.dat↩ which gives the following output:

```
A for Apple
C for Cat
```

This confirms that the output of `stderr` cannot be redirected into another file. It appears only on the terminal.

The function fopen() and fclose() A file can be thought of as a stream of characters. After a file is opened, the stream can be accessed using standard library file-handling functions. A file has several important properties: it has a name, it can be opened, and closed. It can be written to, read from, or appended to. Conceptually, until a file is opened, nothing can be done to it. When it is opened, we can access it at its beginning or at the end. To prevent accidental misuse, we must tell the system which of the three activities — reading, writing, or appending — we intend to perform. When we are finished using the file, we close it. Consider the following program that calculates the average of three integers. Instead of reading these integers from the keyboard, we read them from a file `input.dat`. Since we are reading from a file, we do not need to provide an interactive prompt. Using a text editor, we create the file

input.dat and type three integers (with blank spaces between them) in it. We declare two file pointers inp and outp, which will be associated with particular files. We associate file pointer inp with input.dat and use "r" since we are reading from it. Immediately after opening the file, we must verify that the file opened successfully. If it is not successful (e.g., input.dat is not in the working directory, wrong spelling, etc.), then ifp will be NULL and we stop execution. Similarly, we associate file pointer outp with output.dat and use "w" since we are writing in it. The program will create this file during execution. Again, we check whether the file opens successfully. Next, we read from input.dat using fscanf(), which is similar to scanf() except that it has the file pointer inp associated with input.dat as an extra argument. Similarly, we write the values of the integers along with their average in the file output.dat. We use fprintf() for this purpose, which is similar to printf() except that it has file pointer outp associated with output.dat.

```
#include <stdio.h>
int main()
{
 int a,b,c;
 double avg;
 FILE *inp,*outp; // file pointers
 inp=fopen("input.dat","r"); //open input.dat for reading
 if(inp==NULL)
 {
  printf("Error in opening input.dat\n");
  return 0;
 }
 outp=fopen("output.dat","w");  //open output.dat for writing
 if(outp==NULL)
 {
  printf("Error in opening output.dat\n");
  return 0;
 }
 fscanf(inp,"%d%d%d",&a,&b,&c); // read from input.dat
  avg=(a+b+c)/3.0;
 fprintf(outp,"Average of %d, %d and %d is %.2f\n",a,b,c,avg); // write in output.dat
 return 0;
}
```

Listing 70: C Program "file.c"

If we compile and run the executable, then nothing appears on the terminal. But we observe that

`output.dat` appears in the current directory. To see its content, we execute `$ cat output.dat↩` which gives the following output:

```
Average of 4, 6 and 5 is 5.00
```

Let us change the integer values in `input.dat` and run the executable again. Then the previous content of `output.dat` will be erased, and the new output will be written. If we want to keep the previous output, then we open `output.dat` in the appending mode. The program below implements this.

```
#include <stdio.h>
int main()
{
 int a,b,c;
 double avg;
 FILE *inp,*outp; // file pointers
 inp=fopen("input.dat","r"); //open input.dat for reading
 if(inp==NULL)
 {
  printf("Error in opening input.dat\n");
  return 0;
 }
 outp=fopen("output.dat","a"); // open output.dat for appending
 if(outp==NULL)
 {
  printf("Error in opening output.dat\n");
  return 0;
 }
 fscanf(inp,"%d%d%d",&a,&b,&c); // read from input.dat
 fclose(inp);
  avg=(a+b+c)/3.0;
 fprintf(outp,"Average of %d, %d and %d is %.2f\n",a,b,c,avg); // write in output.dat
 fclose(outp);
 return 0;
}
```

Listing 71: C Program "afile.c"

We again compile and run the executable. Now `$ cat output.dat↩` gives the following output:

```
Average of 4, 6 and 5 is 5.00
Average of 7, 2 and 3 is 4.00
```

Once we have read integers from `input.dat`, we are no longer going to use it. Hence, we close the file using the associated file pointer `inp`. Similarly, we close the file `output.dat`.

Note that a call of the form `fopen(filename, mode)` returns a file pointer to `filename`. There are a number of possibilities for the mode.

Mode	Meaning
"r"	Open text file for reading
"w"	Open text file for writing
"a"	Open text file for appending
"rb"	Open binary file for reading
"wb"	Open binary file for writing
"ab"	Open binary file for appending

Each of these modes can end with a + character. This means the file is to be opened for both reading and writing.

Mode	Meaning
"r+"	Open text file for reading and writing
"w+"	Open text file for writing and reading
⋮	⋮

You need some extra function(s) to deal with a file that is opened for both reading and writing using a + in the mode. We will not pursue this further.

Finally, we write a program that reads a matrix from a file and writes its transpose to another file. For this purpose, we use functions to read and write a matrix in a file. The following program is self-explanatory.

```
#include <stdio.h>
#include <stdlib.h>
#define N 100            // max row & col dim of a matrix
void read_mat(double [][N],int *,int *,char *);
void printms(double [][N],int,int,char);
void trans_mat(double [][N],double [][N],int,int);
void print_mat(double [][N],int, int, char *);
int main()
{
 int m,n;  // actual row & col dim of A
 double A[N][N],B[N][N]; // B=A^T
 read_mat(A,&m,&n,"matA.dat"); //read matrix A from matA.dat
 printms(A,m,n,'A');// print matrix A in the terminal
 trans_mat(A,B,m,n);   // calculate B=A^T
 printms(B,n,m,'B'); // print matrix B in the terminal
```

```
 print_mat(B,n,m,"matB.dat"); // print matrix B in matB.dat
 return 0;
 }
void read_mat(double A[][N],int *m,int *n,char *file_name)
{
 int i,j,p,q;
 FILE *inf;
 inf=fopen(file_name,"r");
 if(inf==NULL)
 {
  printf("Error in opening %s\n",file_name);
  exit(1);
 }
 fscanf(inf,"%d%d",&p,&q);    //read actual row & col dim
 for(i=0;i<p;i++)      //read matrix row-wise
 {
  for(j=0;j<q;j++)
  {
   fscanf(inf,"%lf",&A[i][j]);
  }
 }
 *m=p;
 *n=q;
}
void printms(double A[][N],int m,int n,char c)
{
 int i,j;
 printf("Matrix %c is:\n",c);
 for(i=0;i<m;i++)      //write matrix row-wise
 {
  for(j=0;j<n;j++)
  {
   printf("%-6.2lf    ",A[i][j]);
  }
  printf("\n");
 }
}

void trans_mat(double A[][N],double B[][N],int m,int n)
```

```
{
 int i,j;
 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   B[i][j]=A[j][i];
  }
 }
}
void print_mat(double A[][N],int m,int n,char *file_name)
{
 int i,j;
 FILE *outf;
 outf=fopen(file_name,"w");
 if(outf==NULL)
 {
  printf("Error in opening %s\n",file_name);
  exit(1);
 }
 fprintf(outf,"%d  %d\n",m,n);    //write actual row & col dim
 for(i=0;i<m;i++)      //write matrix row-wise
 {
  for(j=0;j<n;j++)
  {
   fprintf(outf,"%-6.2lf    ",A[i][j]);
  }
  fprintf(outf,"\n");
 }
}
```

Listing 72: C Program "ftransp.c"

We create a file `matA.dat` for the matrix A in which the first line contains the row and column dimensions. The rest of the file contains the matrix in row-wise format. The command `$ cat matA.dat↩` gives the following output:

```
3  4
-1.2  4.0  -7.75  9.5
2.0   5.4   6.25  12.5
20.4  -12   0     0.25
```

It is always a good practice to print a matrix in the terminal to see that the matrix is read properly. Similarly, we also print the transpose of A in the terminal. When we run the executable, the following appears on the screen.

```
Matrix A is:
-1.20     4.00      -7.75     9.50
2.00      5.40      6.25      12.50
20.40     -12.00    0.00      0.25
Matrix B is:
-1.20     2.00      20.40
4.00      5.40      -12.00
-7.75     6.25      0.00
9.50      12.50     0.25
```

The program also creates the data file `matB.dat` in which the transpose is also written in the same format as that of matrix A. Further, $ cat matB.dat↩ gives the following output:

```
4  3
-1.20     2.00      20.40
4.00      5.40      -12.00
-7.75     6.25      0.00
9.50      12.50     0.25
```

Return to Lecture Topics