A Introduction to File (I/O) in C

File Input Output in C

- An important part of any computer program is the ability to communicate with the world external to it by reading input from files and writing results to the files.
- Files are in fact a sequence of bytes stored on secondary or external memory storage such as hard disk. They may contain any character code.
- Most of the programming languages allow creating or reading data files in two general formats:
  - Text File: files that have been stored as a sequence of characters and are readable by the text editors. Example: Programming source files.
  - Binary File: files that are normally stored as chunks of bytes that may represent certain objects or data. Examples: computer-program executable files, pdf files, mp3 files, docx files, etc.

C I/O Streams

- In C a file is simply a continuous stream of bytes.
- To be able to work on the files C provides us with a new Type called **FILE** that is defined in the **stdio.h** header file.
- **FILE** is defined in the stdio.h, with a syntax possibly similar to the following with certain tagName.

```
typedef struct tagName{
   // several data member
   …

   …


 } FILE;
```

- **FILE** objects are usually created by calling the C-library function **fopen**, which returns a pointer of type **FILE***.

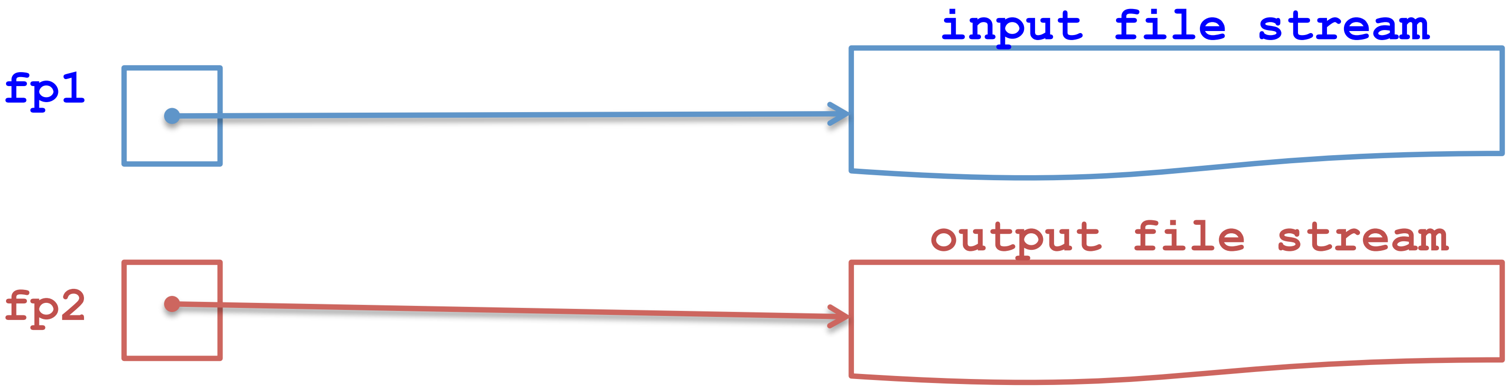
Create an Instance of FILE pointer

- There are four steps associated with accessing a file and reading from and writing into it:
  - Step 1: Declare a pointer of type FILE:

```
FILE* fp1, fp2;
```

  - Step 2: Connect the pointer to the target files – open the files

```
fp1 = fopen(“input.txt”, “r”);
fp2 = fopen(“output.txt”, “w”);
```



  - Step 3: Implement the required operations (read/ write).
  - Step 4: Disconnect the file from i/o streams

Opening a Text File for Writing

- Here is the prototype of library function **fopen**:

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

- Example of access a file for writing in the current working directory:

```
FILE* outp  = fopen (“mydata.txt”, “w”);
```

  - The second argument, **“w”** indicates that you want to open open the file “writing”.

- You can put a complete file path between double quotation marks. Also you can use **“wt**” where **“t”** stands for “text”.

```
outp  =fopen (“/user/mydir/mydata.txt”, “w”);
```

  - **Notice:** that directory separator under the Windows operating system is ‘\\’.

- You should always test whether your file was successfully opened or not. If opening a file fails the FILE pointer will be equal to NULL (zero):

```
if (outp == NULL)  {
   fprintf (stderr, "Error: cannot open the file file “);
   exit(1);
}
```

- When does ***fopen*** function fails when is used to open a file for writing ?
- What happens if file already exits?

How to Write in a Text File

- You can use **fprintf**, similar to **printf**, to write any data into the output stream --In our example into: ***mydata.txt***
- For example you can write the values of an integer and a double into the file **mydata.txt** as follows

```
int a = 80
double b = 4.5
fprintf(outp, “%10d%10\n”, a, b);
```

- Do you know that fprintf also returns an integer value?
- Notice that file pointer outp, has been used as its first argument.

Closing Files

- Although all opened files will be automatically closed, when the C programs terminate, but its always a good practice to close them manually, whenever you don't need them anymore.
- The library function **fclose** is use to disconnect the FILE pointer from stream:

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

- In our previous example we close the file as follows

```
fclose(outp);
```

- This function returns zero if the stream is successfully closed. On failure, EOF (-1) is returned.
- Now, lets write a small program that writes several data from an array into a text file

Example – Writing Data into a Text File

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int main() {
   const char* outfile = “/usres/mydir/myoutput.txt”;
   int a[SIZE] = {2543, 465, 100, 300, 600};
   FILE    *outp ;
   outp = fopen(outfile, “w”);

   if (outp == NULL){
      printf ( "Error: cannot open the file %s: “, outfile);
      exit(1);
   }

   for(int i = 0; i < SIZE; i++)
     fprintf (outp, “%10d\n”, a[i]);
   fclose(outp);
   return 0;
}
```

How to read from a Text File

- Open the file in read mode:

```
File *inp;
inp = fopen(“/users/mydir/myoutput.txt, “r”);
```

- Where **“r”** stands for “read” mode.
- Again you can use “rt” instead of “r”
- You can also open a file in append mode by using **`“a”`** instead.
- Here again you should test is file was successfully accessed. If opening a file fails the FILE pointer again returns NULL (zero):

```
if (inp == NULL)  {
    printf ("Error: cannot open the file input file “);
    exit(1);
}
```

- When opening a file for reading may fail?

Reading From a Text File

- One way to read from a text file is to use a library function **fscanf**.
- **fscanf** is used very similar to **scanf:**

```
int a, b;
n = fscanf (inp, ”%d%d”, &a, &b);
```

- Notice that file pointer ***inp*** has been used as first argument of fscanf. **Can we use fscanf to read from keyboard?**
- The returned value for **fscanf** is equal to the number of the items that reads successfully; Or EOF (-1), if **fscanf** reaches the end of the file.
- **Note:** Files do not have an specific character for EOF. The file system keeps track of size of files.
- When may **fscanf** fail to read input, and what does happen next?

Reading Characters and C-Strings

- C library also provides functions to read a single character or a sequence of character up to a '\0'.
  - To read a single character including the white spaces: end of line character, space, and tab. You may use the function fgetc:

    ```
    int fgetc (File* stream);
    ```

  - This function returns the character read, or EOF on end-of-file or error.
- To read a sequence of chars (a C-string) you may use the library function **fgets**:

  ```
  char *fgets(char *str, int n, FILE *stream);
  ```

  - fgets reads a line from the specified stream and stores it into the string pointed to by **str**.
  - It stops when either **(n-1)** characters are read, the newline character is read, or the end-of-file is reached (whichever comes first).
  - When string is less than n-1, also reads the newline character.
  - Returns NULL if fails to read or if reaches the end-of-file.

Example of Using fgetc to read a text file char by char and print them to the screen:

```
#include <stdio.h>
int main () {
      FILE *fp;
      int c;
      int n = 0;
      fp = fopen("file.txt","r");
      if(fp == NULL)  {
         fprinff(stderr, "Error in opening file\n");
         exit(1);
       }
      do {
           c = fgetc(fp);
           if( c == EOF)
              break;
           printf("%c", c);
       }while(1);

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

Question:
How can we change this program to write into another text file, instead writing on the screen?

Binary Files in C

What is a Binary File

- Binary files are usually thought of a being a sequence bytes.
  - In fact the data will not be interpreted as a sequence of single characters like in a text file.
  - The data will be stored in the same format and sequence of bytes when used in you program.
    - A variable stored into double on the computer memory will be stored into a binary file in the same order and sequence of bytes.
- Example:
  - double x = 0.00887776665551, will be stored in a an 8-byes memory space. The same data in a text file will be stored in a 16-byte memory space.

What is a Binary File (continued)

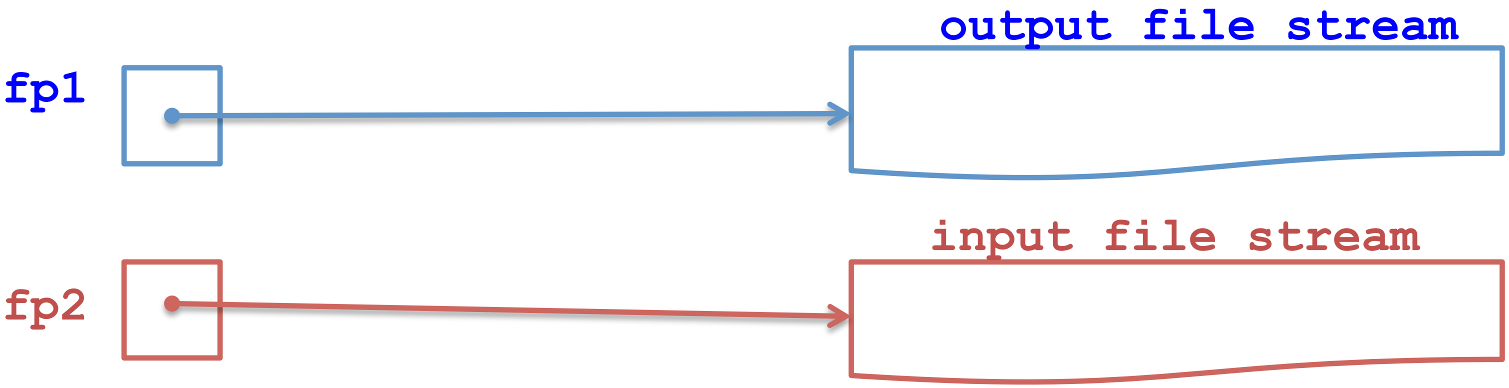
- A binary file is normally more compressed that a text file.
  - Most digital data are stored in binary files
- Reading ad writing data from and into file are faster, using binary data.
- Binary file can be viewed or read properly like a text file using a text editor. Here is an example of a binary file that I opened by an editor on a Mac computer:

```
oe˙.¸†8![.[__text__TEXT#.‹a(.__debug_frame__DWARF$|
‰cdebug_info__DWARF†.K`dc__debug_abbrev__DWARF.P.EW__debug_aranges__DWARFT
Z__debug_macinfo__DWARFT‹Z__debug_loc__DWARFT‹Z__debug_pubnames__DWARFT.‹Z__debug_pubtypes__DWARF≥T$s[__debug
_str__DWARF◊T.[__debug_ranges__DWARF◊T.[__data__DATA◊T.
[__StaIcInit__TEXT‡T{†[‹d.__bss__DATA[__cstring__TEXT[UCÄ__mod_init_func__DATA†U`Ä
```

Opening files in binary mode

- Us fopen in the following format:

```
FILE* fp1, fp2;
fp1 = fopen(“output.bin”, “wb”);
fp2 = fopen(“input.bin”, “rb”);
```



How to write into a binary file

- You can use **fwrite**, to write any data into the output stream --In our example into: ***output.bin.***
- Here is the prototype of the fwrite library function:

```
size_t fwrite(const void* ptr, size_t size,
                         size_t count, FILE *stream);
```

- You can write the values of a double value into the file **output.bin** as follows

```
int n;
double b = 4.5
n = fwrit(&b, sizeof(double),1, fp1);
```

- fwrite returns the number items successfully written into the stream. In this case n will be 1, if it is successfully written into the stream.
- You need to close the file when writing is done.

How to read from a binary file

- You can use **fread**, to read any data from input stream --In our example from: **input.bin.**
- Here is the prototype of the fwrite library function:

```
size_t fread(const void* ptr, size_t size,
                        size_t count, FILE *stream);
```

- For example you can read a double value from the file **output.bin** as follows

```
int n;
double b;
n = fread(&b, sizeof(double), 1, fp2);
```

- fread returns the number items successfully read from stream. In this case n will be 1.
- You need to close the file when reading is done.

Example – Writing Data into a Binary File

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int main() {
  const char* outfile = “/usres/mydir/myoutput.bin”;
  int a[SIZE] = {2543, 465, 100, 300, 600};
  FILE    *outp ;
  outp = fopen(outfile, “wb”);

  if (outp == NULL){
   fprintf (stderr, "Error: cannot open the file %s: “, outfile);
    exit(1);
  }

  fwrite(a, sizeof(a), 1, outp);
  fclose(outp);
  return 0;
}
```

The same program can be written in a different way:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int main() {
   const char* outfile = “/usres/mydir/myoutput.bin”;
   int a[SIZE] = {2543, 465, 100, 300, 600};
   FILE    *outp ;
   outp = fopen(outfile, “wb”);

   if (outp == NULL){
       fprintf (stderr, "Error: cannot open the file %s: “, outfile);
      exit(1);
   }

   for(int j = 0; j < SIZE; j++)
      fwrite(&a[i], sizeof(int), 1, outp);

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

Random Access to the File

- Library function fseek allows us to set the file position indicator for the stream to an offset position.

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

  - Return value: 0 upon success, nonzero value otherwise.
- Sets the file position indicator for the file stream stream to an offset position from origin.
- Origin can be set to:
  - SEEK_SET
  - SEEK_CUR
  - SEEK_END
- Library function ftell, allows us to indcate the current value of the position of indicator of the file stream in number of bytes:

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

- Returns the current value of the position indicator of the stream.

Other file I/O functions

- There are many more function in the C library. Here are couple of them:

  int feof(FILE *stream);

  - Checks if the end of the given file stream has been reached.
  - Returns nonzero value if the end of the stream has been reached, otherwise 0
  - Example:

  int ferror(FILE *stream);

  - Checks the given stream for errors.
  - Returns nonzero value if the file stream has errors occurred, 0 otherwise.
  - Example:

```
c = fgetc(fp);
  if( ferror(fp) )
  {
    printf("Error in reading from file : file.txt\n");
  }
```

- The following code segments shows how feof and ferror are used:

```
while(1)  {
    c = fgetc(fp);
    if( ferror(fp) ) {
      printf("Error in reading from file.\n");
      exit(1);
    }

    if( feof(fp) )
       break ;
    printf("%c", c);
}
```