To:
Tenouk

MODULE 9 - THE C FILE INPUT/OUTPUT 1

MODULE 19 - C++ FILE I/O

create this, delete that, write this, read that, close this, open that

My Training Period: hours

C file input/output abilities:

The standard C++ file input/output is discussed in C++ file input/output. For C++ and MFC (Windows GUI programming) it is called Serialization and the topics are in Single Document Interface (SDI) and Multiple Document Interface (MDI). The source code for this Module is: C file input/output program source codes. In this Module, trainee must be able to understand and use:

- The basic of the data hierarchy.
- A sequential access file – Read and Write related functions.
- Characters, lines and blocks disk file reading and writing related functions.
- A Random access files – Read and Write related functions.
- Some C File Management Functions.
- Other C libraries used for file I/O.

9.1 Introduction

- This Module actually shows you how to use the functions readily available in the C standard library. Always remember this, using the standard library (ISO/IEC C, Single Unix specification or glibc); you must know which functions to call and which header files provide these functions. Then, you must be familiar with the proper prototype of the function call.
- The problems normally exist when dealing with the parameters passed to the functions and the return value of the functions. We will explore some of the very nice and one of the heavily used functions that available in the stdio.h header file, for our file processing and management tasks.
- Keep in mind that in C++ we will use member functions in class objects for file processing and some of the advanced file processing examples will be discussed in C++ file I/O Module.
- Storage of data file as you have learned is temporary, all such data is lost when a program terminates. That is why we have to save files on primary or secondary storage such as disks for future usage.
- Besides that we also need to process data from external files that may be, located in secondary storage as well as writing file during software installation and communicating with computer devices such as floppy, hard disk, networking etc.
- And in socket programming (networking) you will also deal a lot with these open, close, read write activities.
- File used for permanent retention of large amounts of data, stored online or offline in secondary storage devices such as hard disk, CD-Rs/DVDs, tape backup or Network Attached Storage (NAS).

9.2 Basic of The Data Hierarchy

- Ultimately, all data items processed by a computer are just combinations of zeroes and ones.
- The smallest data item in computer can assume the value 0 or 1, called a bit (binary digit).
- But, human being prefer to work with data in the form of decimal digits (i.e. 0, 1, 2, 3, 4, 5, 6, 7…9), letters (i.e. A – Z and a – z) and special symbols (i.e. $, @, %, &, *, (,), -, +, ? and many others) or in readable format.
- As you know, digits, letters and special symbols are referred to as characters, the keys on your keyboard based on whether the ASCII, EBCDIC, Unicode or other proprietary characters set.
- Every character in a computer’s character set is represented as a pattern of 1’s and 0’s, called byte (consists 8 bits-ASCII, EBCDIC), and for Unicode it uses multi-byte or wide characters.
- Characters are composed of bits, and then fields (columns) are composed of characters.
- A field is a group of characters that conveys meaning such as a field representing a month of year.
- Data items processed by computer form a data hierarchy in which data items become larger and more complex in structure as we progress from bits, to char (byte) to field and so on.
- A record (row or tuple) is composed of several fields.
- For example, in a payroll system, a record for a particular employee might consist of the following fields:

    1. Name.
    2. Address.
    3. Security Social Number (SSN)/Passport Number
    4. Salary.

5. Year to date earnings.
6. Overtime claims.

- So, a record is a group of related fields.
- For the payroll example, each of the fields belong to the same employee, in reality a company may have many employees, and will have a payroll records for each employee.
- Conceptually, a file is **a group of related records**.
- A company's payroll file normally contains one record for each employee, thus, a payroll file for a company might contain up to 100, 000 records.
- To facilitate the retrieval of specific records from a file, at least one field in each record is chosen as a record key.
- There are many ways of organizing records in a file. Maybe, the most popular type of organization is called a **sequential file** in which records are typically stored in order by the record key field.

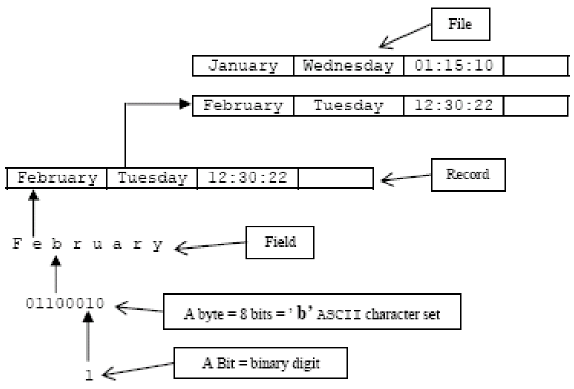

File
January Wednesday 01:15:10
February Tuesday 12:30:22
February Tuesday 12:30:22
Record
F e b r u a r y
Field
01100010
A byte = 8 bits = 'b' ASCII character set
1
A Bit = binary digit

Figure 9.1: An illustration of a simple data hierarchy.

- For example, in a payroll file, records are usually placed in order by Social Security Number (SSN). The first employee record in the file contains the lowest SSN number and subsequent records contain increasingly higher SSN numbers.
- Most business may utilize many different files to store data, for example inventory files, payroll files, employee files and many other types of files.
- For larger application, a group of related files may be called database.
- An application designed to create and manage databases is called a database management system (DBMS). This DBMS term used here just for the data structure discussion, in database it may be different. Popular type of DBMS is Relational DataBase Management System (RDBMS).
- A complete discussion of programming related to the databases can be found in data structure books.
- Here we just want to have some basic knowledge about the construct of the data that the computer processes, from bit to characters to fields and so on until we have a very readable data format organized in a structured manner.

9.3 Files And Streams

- In C, a file can refer to a **disk file**, a **terminal**, a **printer**, a tape drive, sockets or other related devices. Hence, a file represents a concrete device with which you want to exchange information.
- Before you perform any communication to a file, you have to open the file. Then you need to close the opened file after finish exchanging or processing information with it.
- The main file processing tasks may involve the opening, reading, writing and closing.
- The data flow of the transmission from your program to a file, or vice versa, is called a stream, which is a series of bytes. Different with file, a stream is device-independent. All streams have the same behavior including that used in sockets programming such as the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) streams.
- Hence, to perform I/O operations, you can read from or write to any type of files by simply associating a stream to the file.
- There are two formats of streams. The first one is called the text stream, which consists of a sequence of characters (e.g. ASCII data). Depending on the compilers, each character line in a text stream may be terminated by a newline character. Text streams are used for textual data, which has a consistent appearance from one system to another.
- The second format of stream is called the binary stream, which is still a series of bytes. The content of an .exe file would be one example. It is primarily used for non-textual data, which is required to keep the exact contents of the file.
- And for Unicode it is Unicode stream in text or binary modes.
- In C, a memory area, which is **temporarily used to store data** before it is sent to its destination (or received from source), is called a **buffer**. By using buffer, the operating system can improve efficiency by reducing the number of accesses to I/O devices.
- By default all I/O streams are buffered. The buffered I/O is also called the high-level I/O and the low-level I/O refers to the unbuffered I/O.
- Keep in mind that in order to grasp the basic concepts, this Module will deal mainly

with unformatted text files that is disk file.

9.4 Directories, Files and streams

9.4.1 Directories (Folders)

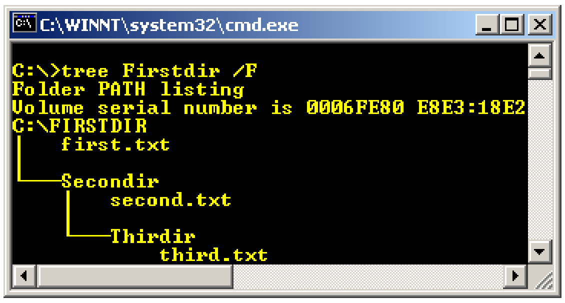
- Every OS have different file system for example ext2, ext3 (Linux), FAT, FAT32, NTFS (Windows). In general this discussion is biased to Linux file system.
- A file system is organized into a hierarchy of directories. For example:

```
C:\Program Files\Microsoft Visual Studio\VC98\Bin
```

- Or in Linux:

```
/testo1/testo2/testo3
```

- Or by issuing a tree command at Windows command prompt:



```
C:\>tree Firstdir /F
Folder PATH listing
Volume serial number is 0006FE80 E8E3:18E2
C:\FIRSTDIR
│   first.txt
│
└───Secondir
    │   second.txt
    │
    └───Thirdir
            third.txt
```

- A directory is a file that contains information to associate other files with names; these associations are called links (shortcuts) or directory entries. Actually, a directory only contains pointers to files, not the files themselves but as users we usually just say "files in a directory".
- The name of a file contained in a directory entry is called a file name component.
- In general, a file name consists of a sequence of one or more such components, separated by the forward slash character (/). So, a file name which is just one component, names a file with respect to its directory. A file name with multiple components, names a directory, and then a file in that directory, and so on.
- Some other documents, such as the POSIX standard, use the term pathname for what was call a file name here and either filename or pathname component should refer to the same meaning in this discussion.

9.4.2 File Name Resolution

- A file name consists of file name components separated by slash (/) characters. On the systems that the GNU C library supports, multiple successive / characters are equivalent to a single / character.
- The process of determining what file a file name refers to is called file name resolution. This is performed by examining the components that make up a file name in left-to-right order, and locating each successive component in the directory, named by the previous component.
- Each of the files that are referenced as directories must actually exist, be directories instead of regular files, and have the appropriate permissions to be accessible by the process; otherwise the file name resolution fails.
- Unlike some other operating systems such as Windows, the Linux system doesn't have any built-in support for file types (or extensions) or file versions as part of its file name prototype.
- Many programs and utilities use conventions for file names. For example, files containing C source code usually have names suffixed with .c and executable files have .exe extension, but there is nothing in the Linux file system itself that enforces this kind of convention.
- May be you can better differentiate those file types by using the -F option for ls directory listing command (`ls -F`).
- If a file name begins with a `/`, the first component in the file name is located in the root directory of the process (usually all processes on the system have the same root directory). In Windows it is normally a C: drive. Such a file name is called an absolute file name.
- Otherwise, the first component in the file name is located in the current working directory and this kind of file name is called a relative file name. For example, the Secondir and Thirdir should be relative file name and Firstdir is an absolute filename.

```
/Firstdir/Secondir/Thirdir
```

- The file name components '`.`' ("dot") and '`..`' ("dot-dot") have special meanings.

```
C:\Firstdir>dir /a
 Volume in drive C has no label.
 Volume Serial Number is E8E3-18E2

 Directory of C:\Firstdir
```

```
04/18/2005  03:09p      <DIR>          .
04/18/2005  03:09p      <DIR>          ..
04/18/2005  03:08p                   0 first.txt
04/18/2005  03:09p      <DIR>          Secondir
               1 File(s)              0 bytes
               3 Dir(s)   1,326,395,392 bytes free
```

- Every directory has entries for these file name components. The file name component '`.`' refers to the directory itself, while the file name component '`..`' refers to its parent directory (the directory that contains the link for the directory in question). That is why if we want to change to the parent directory of the current working directory we just issue the '`cd ..`' command for Linux and Windows.
- Then in Linux, to run a program named testrun in the current working directory we issue the following command:

```
./testrun
```

- As a special case, '`..`' in the root directory refers to the root directory itself, since it has no parent; thus '`/..`' is the same as `/`.
- Here are some examples of file names:

File name	Description
/a	The file named a, in the root directory.
/a/b	The file named b, in the directory named a in the root directory.
a	The file named a, in the current working directory.
/a/./b	This is the same as /a/b.
./a	The file named a, in the current working directory.
../a	The file named a, in the parent directory of the current working directory.

Table 9.1: File names examples

- A file name that names a directory may optionally end in a `/`. You can specify a file name of `/` to refer to the root directory, but the empty string is not a meaningful file name.
- If you want to refer to the current working directory, use a file name of '`.`' or '`./`'. For example to run a program named testprog that located in the current working directory we just prefixes the `./` to the program name.

```
./testprog
```

9.4.3 Streams and FILE structure

- The type of the C data structure that represents a stream is called FILE rather than "stream". Since most of the library functions deal with objects of type FILE *, sometimes the term file pointer is also used to mean "stream". This leads to confusion over terminology in many reference materials and books on C.
- The FILE type is declared in the stdio.h header file.

FILE data type
This is the data type used to represent stream objects. A FILE object holds all of the internal state information about the connection to the associated file, including such things as the file position indicator and buffering information. Each stream also has error and end-of-file status indicators that can be tested with the ferror() and feof() functions.

Table 9.2: FILE data type

- FILE objects are allocated and managed internally by the I/O library functions.

9.4.4 Standard Streams

- When the main function of your program is invoked, it already has three predefined streams open and available for use. These represent the standard input and output channels that have been established for the process. A process here means a running program.
- These streams are declared in the stdio.h header file and summarized in the following Table.

Standard stream	Description
FILE * stdin	The *standard input* stream variable, which is the normal source of input for the program.
FILE * stdout	The *standard output* stream variable, which is used for normal output from the program.
FILE * stderr	The *standard error* stream variable, which is used for error messages and diagnostics issued by the program.

Table 9.3: Standard streams

- In the Linux system, you can specify what files or processes correspond to these streams using the pipe and redirection facilities provided by the shell.
- Most other operating systems provide similar mechanisms, but the details of how to use

them can vary.

- In the GNU C library, stdin, stdout, and stderr are normal variables which you can set just like any others. For example, to redirect the standard output to a file, you could do:

```
fclose(stdout);
stdout = fopen ("standard-output-file", "w");
```

- However, in other systems stdin, stdout, and stderr are macros instead of variables that you cannot assign to in the normal way. But you can use for example freopen() function to get the effect of closing one and reopening it.

9.5 Links Story

9.5.1 Hard Links

- In POSIX systems, one file can have many names at the same time. All of the names are equally real, and no one of them is preferred to the others. In Windows it is called shortcuts.
- To add a name to a file, use the link() function (The new name is also called a **hard link** to the file). Creating a new link to a file does not copy the contents of the file; it simply **makes a new name** by which the file can be known, in addition to the file's existing name or names.
- One file can have names in several directories, so the organization of the file system is not a strict hierarchy or tree.
- In most implementations, it is not possible to have hard links to the same file in multiple file systems. link() reports an error if you try to make a hard link to the file from another file system when this cannot be done.
- The prototype for the link() function is declared in the header file unistd.h and is summarized below.

int link(*const char *oldname, const char *newname*)	
The link() function makes a new link to the existing file named by *oldname*, under the new name *newname*. This function returns a value of 0 if it is successful and -1 on failure. In addition to the usual file name errors, for both *oldname* and *newname*, the following errno error conditions are defined for this function:	
EACCES	You are not allowed to write to the directory in which the new link is to be written.
EEXIST	There is already a file named *newname*. If you want to replace this link with a new link, you must remove the old link explicitly first.
EMLINK	There are already too many links to the file named by *oldname*. (The maximum number of links to a file is LINK_MAX).
ENOENT	The file named by *oldname* doesn't exist. You can't make a link to a file that doesn't exist.
ENOSPC	The directory or file system that would contain the new link is full and cannot be extended.
EPERM	In the GNU system and some others, you cannot make links to directories. Many systems allow only privileged users to do so. This error is used to report the problem.
EROFS	The directory containing the new link can't be modified because it's on a read-only file system.
EXDEV	The directory specified in *newname* is on a different file system than the existing file.
EIO	A hardware error occurred while trying to read or write the to filesystem.

Table 9.4: link() function

9.5.2 Symbolic Links

- The Linux system for example, supports soft links or symbolic links. This is a kind of "file" that is essentially a pointer to another file name.
- Unlike hard links, symbolic links can be made to directories or across file systems with no restrictions. You can also make a symbolic link to a name which is not the name of any file. (Opening this link will fail until a file by that name is created).
- Likewise, if the symbolic link points to an existing file which is later deleted, the symbolic link continues to point to the same file name even though the name no longer names any file.
- The reason symbolic links work the way they do is that special things happen when you try to open the link. The open() function realizes you have specified the name of a link, reads the file name contained in the link, and opens that file name instead.
- The stat() function (used for file attributes information) likewise operates on the file that the symbolic link points to, instead of on the link itself.
- By contrast, other operations such as deleting or renaming the file operate on the link itself. The functions readlink() and lstat() also refrain from following symbolic links, because their purpose is to obtain information about the link.
- link(), the function that makes a hard link, does too. It makes a hard link to the symbolic link, which one rarely wants.
- Some systems have for some functions operating on files have a limit on how many symbolic links are allowed when resolving a path name. The limit if exists is published in the sys/param.h header file.
- The following lists functions and macros used for links.

int MAXSYMLINKS

The macro MAXSYMLINKS specifies how many symlinks some function will follow before returning ELOOP. Not all functions behave the same and this value is not the same as a returned for _SC_SYMLOOP by sysconf. In fact, the sysconf result can indicate that there is no fixed limit although MAXSYMLINKS exists and has a finite value.

Table 9.5: MAXSYMLINKS macro

- Prototypes for most of the functions listed in the following section are in unistd.h.

int symlink(*const char *oldname, const char *newname*)	
The symlink() function makes a symbolic link to *oldname* named *newname*. The normal return value from symlink() is 0. A return value of -1 indicates an error. In addition to the usual file name prototype errors, the following errno error conditions are defined for this function:	
EEXIST	There is already an existing file named *newname*.
EROFS	The file *newname* would exist on a read-only file system.
ENOSPC	The directory or file system cannot be extended to make the new link.
EIO	A hardware error occurred while reading or writing data on the disk.

Table 9.6: symlink() function

int readlink(*const char *filename, char *buffer, size_t size*)	
The readlink() function gets the value of the symbolic link *filename*. The file name that the link points to is copied into *buffer*. This file name string is *not* null-terminated; readlink() normally returns the number of characters copied. The *size* argument specifies the maximum number of characters to copy, usually the allocation size of *buffer*. If the return value equals *size*, you cannot tell whether or not there was room to return the entire name. A value of -1 is returned in case of error. In addition to the usual file name errors, following errno error conditions are defined for this function:	
EINVAL	The named file is not a symbolic link.
EIO	A hardware error occurred while reading or writing data on the disk.

Table 9.7: readlink() function

- In some situations it is desirable to resolve all the symbolic links to get the real name of a file where no prefix, names a symbolic link which is followed and no filename in the path is '`.`' or '`..`'.
- This is for example desirable if files have to be compared in which case different names can refer to the same inode. For Linux system we can use canonicalize_file_name() function for this purpose.
- The UNIX standard includes a similar function which differs from canonicalize_file_name() in that the user has to provide the buffer where the result is placed in. It uses realpath() function.
- The advantage of using this function is that it is more widely available. The drawback is that it reports failures for long path on systems which have no limits on the file name length.

9.6 The Basic Of Disk File I/O

9.6.1 Opening And Closing A Disk File

- Before we dive into the details, take note that the program examples presented here just for basic file I/O that applies to DOS and Linux.
- For Windows, you have to study the Win32 programming that provides specifics file I/O and other related functions. Here we do not discuss in details regarding the permission, right and authorization such as using Discretionary Access Control List (DACL) and Security Access Control List (SACL) implemented in Windows OS.
- Furthermore for DOS type OS also, Microsoft uses Microsoft C (C-Runtime – CRT). Nevertheless the concepts still apply to any implementation.
- As explained before, in C, a FILE structure is a file control structure defined in the header file stdio.h. A pointer of type FILE is called a file pointer, which references a disk file.
- A file pointer is used by stream to conduct the operation of the I/O functions. For instance, the following declaration defines a file pointer called fpter:

```
FILE  *fpter;
```

- In the FILE structure there is a member, called the file position indicator, which points to the position in a file where data will be read from or written to.

- The I/O function fopen() gives you the ability to open a file and associate a stream to the opened file. You need to specify the way to open a file and the filename with the fopen() function. The prototype is:

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

- Here, filename is a char pointer that references a string of a filename. The filename is given to the file that is about to be opened by the fopen() function. mode points to another string that specifies the way to open the file.
- The fopen() function returns a pointer of type FILE. If an error occurs during the procedure to open a file, the fopen() function returns a null pointer.

- Table 9.8 shows the possible ways to open a file by various strings of modes.
- Note that, you might see people uses the mode rb+ instead of r+b. These two strings are equivalent. Similarly, wb+ is the same as w+b, ab+ is equivalent to a+b.
- The following program segment example try to open a file named test.txt, located in the same folder as the main() program for reading.

```
FILE    *fptr;
if((fptr = fopen("test.txt","r")) == NULL)
{
   printf("Cannot open test.txt file.\n");
   exit(1);
}
```

- Here, "r" is used to indicate that the text file is about to be opened for reading only. If an error occurs such as the file is non-exist, when the fopen() function tries to open the file, the function returns a null pointer.
- Then an error message is printed out by the printf() function and the program is aborted by calling the exit() function with a nonzero value to handle the exception.

Mode	Description
r	Open a file for reading.
w	Create a file for writing. If the file already exists, discard the current contents.
a	Append, open or create a file for writing at the end of the file.
r+	Open a file for update (reading and writing).
w+	Create a file for update. If the file already exists, discard the current contents.
a+	Append, open or create a file for update, writing is done at the end of the file.
rb	Opens an existing binary file for reading.
wb	Creates a binary file for writing.
ab	Opens an existing binary file for appending.
r+b	Opens an existing binary file for reading or writing.
w+b	Creates a binary file for reading or writing.
a+b	Opens or creates a binary file for appending.

Table 9.8: Possible ways opening a file by various strings of modes in C

- After a disk file is read, written, or appended with some new data, you have to disassociate the file from a specified stream by calling the fclose() function.
- The prototype for the fclose() function is:

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

- Here, stream is a file pointer that is associated with a stream to the opened file. If fclose () closes a file successfully, it returns 0. Otherwise, the function returns EOF.
- By assuming the previous program segment successfully opened the test.txt for reading, then to close the file pointer we should issue the following code:

```
fclose(fptr);
```

- Normally, the fclose() function fails only when the disk is removed before the function is called or there is no more space left on the disk.
- The end-of-file (EOF) combination key for different platform is shown in table 9.9. You have to check your system documentation.

Computer system	Key combination
UNIX® systems	`<return> <ctrl> d`
IBM® PC and compatibles	`<ctrl> z`
Macintosh® - PowerPC	`<ctrl> d`
VAX® (VMS)	`<ctrl> z`

Table 9.9: End-of-file (EOF) key combinations for various computer systems.

- Since all high-level I/O operations are buffered, the fclose() function flushes data left in the buffer to ensure that no data will be lost before it disassociates a specified stream with the opened file.
- A file that is opened and associated with a stream has to be closed after the I/O operation. Otherwise, the data saved in the file may be lost or some unpredictable errors might occur during the next time file opening.
- Let try the following program example, which shows you how to open and close a text file and how to check the returned file pointer value as well.
- First of all you have to create file named **tkk1103.txt**. This file must be in the same folder as your running program, or you have to provide the full path string if you put it in other folder.
- By default program will try finding file in the same folder where the program is run.
- For example, if you run your C program in folder:

C:\BC5\Myproject\testing\

- Then, make sure you put the **tkk1103.txt** file in the same folder:

C:\BC5\Myproject\testing\tkk1103.txt

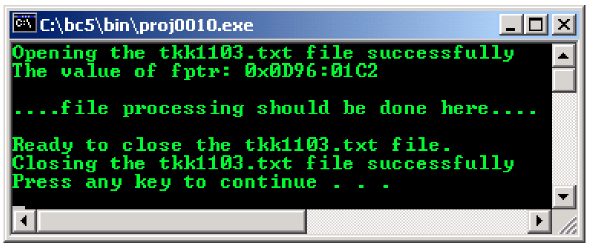
```c
// opening and closing a file example
#include <stdio.h>
// #include <stdlib.h>

// SUCCESS = 0, FAIL = 1 using enumeration
enum {SUCCESS, FAIL};

int main (void)
{
   FILE   *fptr;
   // the filename is tkk1103.txt and located
   // in the same folder as this program
   char filename[ ] = "tkk1103.txt";

   // set the value reval to 0
   int reval = SUCCESS;
   // test opening file for reading, if fail...
   if((fptr = fopen(filename, "r")) == NULL)
   {
        printf("Cannot open %s.\n", filename);
        reval = FAIL;      // reset reval to 1
   }
   // if successful do...
   else
   {
        printf("Opening the %s file successfully\n", filename);
        // the program will display the address where
        // the file pointer points to..
        printf("The value of fptr: 0x%p\n", fptr);
        printf("\n....file processing should be done here....\n");
        printf("\nReady to close the %s file.\n", filename);
        // close the file stream...
        if(fclose(fptr)==0)
        printf("Closing the %s file successfully\n", filename);
   }
   // for Borland…can remove the following pause and the library,
   // stdlib.h for other compilers
   // system("pause");
   return reval;
}
```

40 lines: Output:

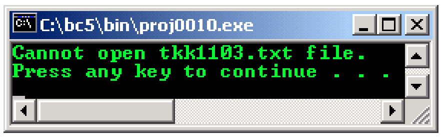


```
Opening the tkk1103.txt file successfully
The value of fptr: 0x0D96:01C2

....file processing should be done here....

Ready to close the tkk1103.txt file.
Closing the tkk1103.txt file successfully
Press any key to continue . . .
```

- If opening the file is fails, the following will be output:



```
Cannot open tkk1103.txt file.
Press any key to continue . . .
```

- Remember, for the "r" mode, you have to create and save **tkk1103.txt** file in the same folder where the .exe file for this program resides or provide the full path strings in the program.
- This program shows you how to open a text file. fopen() function tries to open a text file with the name contained by the string array filename for reading. The filename (stored in array) is defined and initialized with to **tkk1103.txt**.
- If an error occurs when you try to open the text file, the fopen() function returns a null pointer. Next line then prints a warning message, and assigns the value represented by the enum name FAIL to the int variable reval. From the declaration of the enum data type, we know that the value of FAIL is 1.
- However, if the fopen() function opens the text file successfully, the following statement:

```c
printf("The value of fptr: 0x%p\n", fptr);
```

- Will print the value contained by the file pointer fptr.
- At the end line of code tells the user that the program is about to close the file, and then fclose(fptr); closes the file by calling the fclose() file.

- return reval; the return statement returns the value of reval that contains 0 if the text file has been opened successfully or 1 otherwise.
- From the output, the value held by the file pointer is 0x0D96:01C2 (memory address) after the text file is open successfully. Different pc will have different address.
- If your **tkk1103.txt** file is not in same folder as your main() program, you have to explicitly provide the full path of the file location.
- For example, if your tkk1103.txt is located in C:\Temp folder, you have to change:

```
char    filename[ ] = "tkk1103.txt";
```

- To

```
char    filename[ ] = "c:\\Temp\\tkk1103.txt";
```

C & C++ programming tutorials

Further C file i/o related reading and digging:

1. For C++ and MFC (Windows GUI programming) it is called Serialization and the topics are in Single Document Interface (SDI) and Multiple Document Interface (MDI).
2. The source code for this Module is: C file input/output program source codes.
3. Check the best selling C / C++ books at Amazon.com.
4. Wide character/Unicode is discussed Character Sets, Unicode & Locale and the implementation using Microsoft C is discussed Windows Users & Groups C programming.
5. Implementation specific information for Microsoft can be found Microsoft C Run-Time Tutorials and More Win32 Windows C Run-Time programming Tutorials.