FILES

- File access is one of the most basic services provided by any operating system.
- This involves
  - opening and closing files
  - reading from and writing to files
  - gathering information about files
- You can access files from a C++ program in one of two ways
  - at the API level, using Win32 systems calls
  - from a higher-level C++ library

FILES

- In many cases, the C++ library level will be sufficient when you just want to open a file, read to or write from it, and close it again.
- However, in many cases it is benificial to drop down to the API level to access files.
- This is because many features are not exposed at the higher level, such as NT's security options.

What is a file?

- Physically, a file is a set of sectors on a disk identified under a unique file name.
- In the Win32 API, a file is treated as a collection of bytes.
- You can seek to any byte offset, and read a block of bytes of any size.
- If you want to work at a higher level, (e.g. you want to treat a file as a set of text lines), you use the high-level C++ libraries.

Opening and Reading from a file in C++

- The easiest way to open and read from a file is to use the I/O library that comes with C++.

```
#define MAXPATH 30
#include <iostream.h>
#include <fstream.h>
void main()
{
  char filename[MAXPATH];
  char ch;
  cout << "Enter filename: ";
  cin >> filename;            //get the file name
  ifstream infile(filename);  //open it
  if(infile)                  //read until EOF (End
  Of File)
  while(infile.get(ch)) //one character at a time
          cout << ch;  //echo to screen
  infile.close();             //close the file
}
```

Files in C++

- In C++, you open a file by linking it to a stream.
- The definitions are found in <fstream.h>
  - To create an input stream, create an **ifstream** object,
  - for output, create an **ofstream** object
  - for both input and output create an **fstream** object.

Files in C++

- You can either open the file in the constructor, or with the member function **open()**
- Here you must specify the filename and the mode, which may be a combination of such values as:

  ios::app
  ios::in
  ios::out

  Etc…

Files in C++

- You can move to random positions within the file using seekp() and seekg().
- For example, to start reading at the beginning of the file use:
  in.seekg(0,ios::beg);
- to start writing at the beginning of the file use:
  out.seekp(0,ios::beg);

Files in MFC

- **CFile** is the base class for Microsoft Foundation file classes.
- Use **CFile** and its derived classes for general-purpose disk I/O. Use iostream classes for formatted text sent to a disk file.
- Normally, a disk file is opened automatically on **CFile** construction and closed on destruction. Static member functions permit you to interrogate a file's status without opening the file.

CFile::CFile

- There are 3 forms of the CFile constructor:

CFile( );
CFile(int *hFile* **);**
CFile(LPCTSTR *lpszFileName*, **UINT** *nOpenFlags* **);**
throw(CFileException);

(This is an example of the MFC error handling mechanisms – you can examine the CFileException object to get more information about any error that occurs)

CFile::CFile

- *hFile:* The handle of a file that is already open.
- *lpszFileName:* A string that is the path to the desired file. The path can be relative or absolute.
- *nOpenFlags:* Sharing and access mode. Specifies the action to take when opening the file. You can combine options listed below by using the bitwise-OR (**|**) operator. One access permission and one share option are required;

Typical values of nOpenFlags are :

CFile::modeCreate Directs the constructor to create a new file. If the file exists already, it is truncated to 0 length.

CFile::modeNoTruncate Combine this value with **modeCreate**. If the file being created already exists, it is not truncated to 0 length.

CFile::modeRead Opens the file for reading only.
CFile::modeReadWrite Opens the file for reading and writing.
CFile::modeWrite Opens the file for writing only.

CFile::modeNoInherit Prevents the file from being inherited by child processes.

CFile::shareDenyNone Opens the file without denying other processes read or write access to the file.
CFile::shareDenyRead Opens the file and denies other processes read access to the file.

Example 1 - opening a file

```
char* pFileName = "test.dat";
CFile f( pFileName, CFile::modeCreate|
                    CFile::modeWrite);
```

Or, using the MFC exception handling facilities:

```
char* pFileName = "test.dat";
try{ CFile f( pFileName,CFile::modeCreate|
                    CFile::modeWrite);}
catch(CfileException *e){
  //Alert user to the error…
}
```

Example 2 - opening a file

- In MFC, the most common way to open a file is a two-stage process.:

1. Create the file object without specifying a path or permission flags. You usually create a file object by declaring a **CFile** variable on the stack frame.
2. Call the **Open** member function for the file object, supplying a path and permission flags. The return value for **Open** will be nonzero if the file was opened successfully or 0 if the specified file could not be opened.

Example 2 - opening a file

- The following example shows how to create a new file with read/write permission (replacing any previous file with the same path).
- If there are problems, the **Open** call can return a **CFileException** object in its last parameter.
- The **TRACE** macro prints both the file name and a code indicating the reason for failure.
- You can call the **AfxThrowFileException** function if you require more detailed error reporting.


```
char* pszFileName = "myfile.dat";
CFile myFile;
CFileException fileException;

if ( !myFile.Open(  pszFileName,
                    CFile::modeCreate |
                    CFile::modeReadWrite),
                    &fileException ){
     TRACE( "Can't open file %s,
               error = %u\n",
               pszFileName,
               fileException.m_cause );
     }
```


To read from and write to the file

- Use the **Read** and **Write** member functions to read and write data in the file. -or-
- The **Seek** member function is also available for moving to a specific offset within the file.

To read from and write to the file

- **Read** takes a pointer to a buffer and the number of bytes to read and returns the actual number of bytes that were read.
- If the required number of bytes could not be read because end-of-file (EOF) is reached, the actual number of bytes read is returned.
- If any read error occurs, an exception is thrown.
- **Write** is similar to **Read**, but the number of bytes written is not returned. If a write error occurs, including not writing all the bytes specified, an exception is thrown.

If you have a valid **CFile** object,
you can read from it or write to it as follows:

```
char      szBuffer[256]="Any String";
UINT      nActual = 0;
CFile myFile;

myFile.Write(szBuffer,sizeof(szBuffer) );
myFile.Seek( 0, CFile::begin );
nActual = myFile.Read(szBuffer,
                      sizeof(szBuffer));
```

Closing Files

- Once you finish with a file, you must close it. (Otherwise, it may remain locked)
- Use the **Close** member function. This function closes the file-system file and flushes buffers if necessary.
- If you allocated the CFile object on the frame, the object will automatically be closed and then destroyed when it goes out of scope.
- Note that deleting the **CFile** object does not delete the physical file in the file system.