Opening and closing a file.

I/O objects are attached to specific files with the following.

- ifstream myin(filename, ios::in); // myin is now a file stream for input attached to a specific file, filename
- ofstream myout(filename, ios::out); //myout is now a file stream for output attached to a specific file, filename
- ofstream myout(filename, ios::app); //to append at end
- fstream myio(filename, ios::in | ios::out); //for input & output

We can also establish a file I/O without it being attached to any file. The open member function is used to later attach an object to a file – for example,

```
ifstream myin; //file input object
myin.open(filename); //attach to filename
if( !myin) //open failed?
…
…
myin.close();
```

The member function open() has prototype

```
void open (const char * filename, openmode mode);
```

where filename is a string of characters representing the name of the file to be opened and openmode is a combination of the following flags:

ios::in	Open file for reading
ios::out	Open file for writing
ios::ate	Initial position: end of file
ios::app	Every output is appended at the end of file
ios::trunc	If the file already existed it is erased
ios::binary	Binary mode

These flags can be combined using bitwise operator OR: |. For example, if we want to open the file "test.bin" in binary mode to add data we could do it by the following call to open:

```
ofstream file;
file.open ("test.bin", ios::out | ios::app | ios::binary);
```

The member function open() of classes ofstream, ifstream and fstream include a default mode (that varies from one to the other):

class	Default mode
ofstream	ios::out | ios::trunc
ifstream	ios::in
fstream	ios::in | ios::out

The default value is only applied if the function is called without the mode parameter. If the function is called with any value in that parameter the default mode is not used.

Since frequently the first task that is performed on an object of classes ofstream, ifstream and fstream is to open a file, we could combine the two previous lines in the object declaration as follows:

```
ofstream file ("test.bin", ios::out | ios::app | ios::binary);
```

The I/O operators << and >> are overloaded to serve as output and input operators, respectively, for fstreams. For example, to do output:

```
cout<< var;
myout << var;
```

Similarly input can be done with:

```
cin>> c;
myin>> c;
```

Several built-in member functions for I/O can be used to perform I/O on characters, a whole line or sequence of bytes:

```
obj.put(char c)
obj.get(char& c)
int c = obj.get()
obj.read
obj.write
```

fstream Prototype

* fstream();
* fstream(const char* szName, int nMode, int nProt = filebuf::openprot);

nMode File mode

- ios::app: The function performs a seek to the end of file. When new bytes are written to the file, they are always appended to the end, even if the position is moved with the ostream::seekp function.
- ios::ate: The function performs a seek to the end of file. When the first new byte is written to the file, it is appended to the end, but when subsequent bytes are written, they are written to the current position.
- ios::in: The file is opened for input. The original file (if it exists) will not be truncated.
- ios::out: The file is opened for output.
- ios::trunc: If the file already exists, its contents are discarded. This mode is implied if ios::out is specified, and ios::ate, ios::app, and ios:in are not specified.
- ios::nocreate: If the file does not already exist, the function fails.
- ios::noreplace: If the file already exists, the function fails.
- ios::binary: Opens the file in binary mode (the default is text mode).

nProt is the file protection specification.

Defaults to the static integer filebuf::openprot, which is equivalent to the operating system default, filebuf::sh_compat, under MS-DOS. The possible nProt values are as follows:

- filebuf::sh_compat -- Compatibility share mode (MS-DOS only).
- filebuf::sh_none -- Exclusive mode — no sharing.
- filebuf::sh_read -- Read sharing allowed.
- filebuf::sh_write -- Write sharing allowed.
- The filebuf::sh_read and filebuf::sh_write modes can be combined with the logical OR (||) operator.