Working with Files

A file is a collection of related data stored in a particular area on the disk.

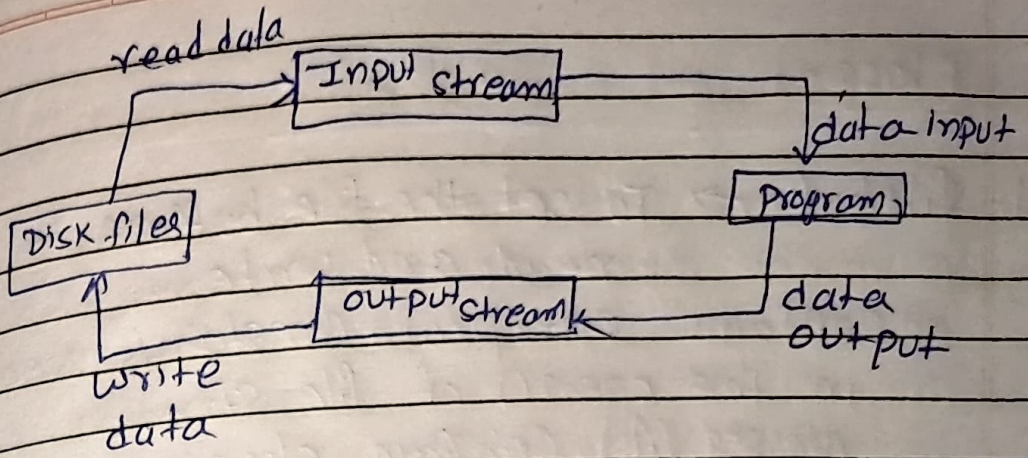
Programs can be designed to perform the read and write operations on these files.

1. Data transfer b/w the Console unit and the program.
2. Data transfer b/w the program and a disk file.

The I/O System of C++ handles file operations which are very much similar to the console Input and output operations.

The stream that supplies data to the program is known as input stream.

The stream that receives data from the program is known as output stream.



The I/O stream of C++ contains a set of classes that define the file handling methods. These include ifstream, ofstream and fstream. These classes are derived from fstream base and from the corresponding iostream class.

These classes, designed to manage the disk files, are declared in fstream.

Opening and closing a File

If we want to use a disk file, we need to decide the following things about the file and its intended use:

1. Suitable name for the file.
2. Data type and structure.
3. Purpose
4. Opening method.

Class

1. filebuf → To set the file buffers to read and write.
   contains openprot constants used in the open() of file stream classes. Also contain close() and open() as members.

2. fstreambase → Provides operations ~~to~~ common to the file streams.
   Serves as a base for fstream, ifstream and ofstream class.
   Contains open() and close() functions.

3. ifstream → Provides input operations.
   Contains open() with default input mode. Inherits the functions get(), getline(), read(), seekg(), tellg() functions from istream.

4. ofstream → provides output operations.
   Contains open() with default output mode
   Inherits put(), seekp(), tellp(), write() functions from ostream.

5. fstream ⇒ Inherits all the functions from istream, ostream classes through iostream.

* The filename is a string of characters that make up a valid filename for operating System.

A file can be opened in two ways -

1. using the constructor

```
ofstream outfile("file-name");
                    // output only.

ifstream infile("file-name");
                    //input only
infile.close()  or outfile.close()
```

2. using the member function open() of the class

```
file-stream-class stream-object;
stream-object.open("filename");
```

Eg:
```
ofstream outfile;
outfile.open("file");
```

Detecting end-of-file

Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.

```
while(fin)
```

An ifstream object 'fin' returns a value of '0' if any errors occurs in the file operation including the end-of-file condition. Thus the while loop terminates when fin returns a value of zero on reaching the end of file condition.

```
if(fin.eof()!=0)
{
    exit(1);
}
```

eof() is a member function of ios class. It returns a non-zero value if the EOF condition is encountered and a zero, otherwise.

More about Open(): File modes.

The general form of the function open() with two arguments is:

stream-object.open("filename", mode);

ios::in → for ifstream (reading only)
ios::out → ofstream (writing only)
[defined in class ios]

File mode parameters-

ios::app	→ append to EOF
ios::ate	→ Go to EOF opening
ios::binary	→ Binary File
ios::in	→ open for reading only
ios::nocreate	→ open fails if the file doesn't exist.
ios::noreplace	→ open fails if the file already exists.
ios::out	→ open file for writing only.
ios::trunc	→ Delete the contents of file if it exists

The mode can combine two or more parameters using the bitwise OR operator.

File pointers and their Manipulations

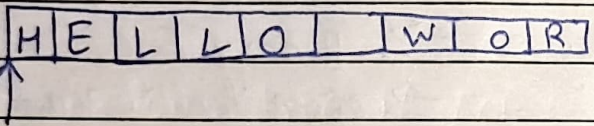
Each file has two associated pointers known as file pointers.
One of them is called input pointer or get pointer and the other is called output or put pointer.

We can use these pointers to move through the files while reading or writing.

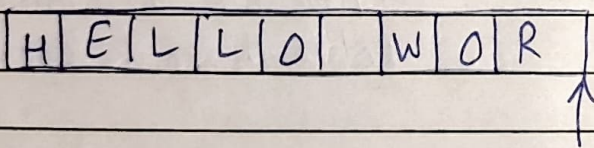
Default Actions

file.

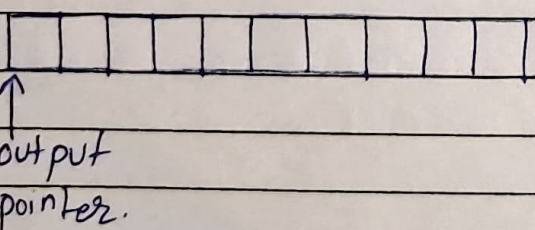
reading only.



open in append mode.



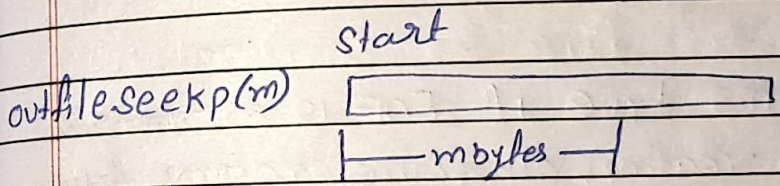
writing only



Functions for Manipulation of file pointers

- seekg() → Moves get pointer to a specified location.
- seekp() → Moves put pointer to a specified location.
- tellg() → Gives the current position of get pointer.
- tellp() → gives the current position of put pointer.

Specifying the offset:




```
seekg(offset, refposition);
seekp(offset, refposition);
```

refposition takes one of the following three constants —

- ios::beg → start of file
- ios::cur → current position of pointer
- ios::end → End of the file.

'offset' parameter represents the number of bytes the file pointer is to be moved from the location.

Sequential input and output operations

put() and get() are designed for handling a single character at a time.
write() and read() are designed to write and read blocks of binary data.

Reading and writing a class object

Error handling Functions

eof() → Returns true if EOF is encountered while reading; otherwise returns false.

fail() → Returns true when an input or output operation has failed.

bad() → Returns true if an invalid operation is attempted.

good() → Returns true if no error has occurred.

Command-Line Arguments

C++ too supports a feature that facilitates the arguments to the main() function. These arguments are supplied at the time of invoking the program. They are typically used to pass the names of data files. E.g;

C> exam data results.

The command-line arguments are typed by the user and delimited by a space.

The first arguments is always the filename (command name) and contains the program to be excuted.

The main() functions which we have been using up to now without any arguments can take two arguments as shown

int main(int argc, char *argv[])

argc known as argument Counter represents the number of arguments in Command line. The argv known as argument vector is an array of char type pointers that points to the command line arguments.

For the c> exam data results the value of argc would be 3 and the argv would be:

argv[0] ---> exam
argv[1] ---> data
argv[2] ---> results.

argv[0] always represents the command or name that invokes the program.

```
infile.open(argv[1]); // open data file
                         for reading.

~~if~~
outfile.open(argv[2]);
  // open results file for writing.
```