C++ File I/O

Constructors

Syntax:

```
  fstream( const char *filename, openmode mode );
  ifstream( const char *filename, openmode mode );
  ofstream( const char *filename, openmode mode );
```

The fstream, ifstream, and ofstream objects are used to do file I/O. The optional *mode*, please refer to the open function. The optional *filename* specifies the file to be opened and associated with the stream. For example, the following code reads input data and appends the result to an output file.

```
    ifstream fin( "/tmp/data.txt" );
    ofstream fout( "/tmp/results.txt", ios::app );
    while( fin >> temp )
      fout << temp + 2 << endl;
    fin.close();
    fout.close();
```

Input and output file streams can be used in a similar manner to C++ predefined I/O streams, **cin** and **cout**.

close

Syntax:

```
  void close();
```

The close() function closes the associated file stream.

eof

Syntax:

```
  bool eof();
```

The function eof() returns **true** if the end of the associated input file has been reached, **false** otherwise. For example:

```
    char ch;
    ifstream fin( "temp.txt" );
    while(true) {
      fin >> ch;
      if(fin.eof()) break;
      cout << ch;
    }
    fin.close();
```

flush

Syntax:

```
  ostream &flush();
```

The flush() function causes the buffer for the current output stream to be actually written out to the attached device. This function is useful for printing out debugging information, because sometimes programs abort before they have a chance to write their output buffers to the screen. Judicious use of flush() can ensure that all of your debugging statements actually get printed.

get

Syntax:

```
  int get();
  istream &get( char &ch );
  istream &get( char *buffer, streamsize num );
  istream &get( char *buffer, streamsize num, char delim );
  istream &get( streambuf &buffer );
  istream &get( streambuf &buffer, char delim );
```

The get() function is used with input streams, and either:

- reads a character and returns that value,
- reads a character and stores it as *ch*,
- reads characters into *buffer* until *num* - 1 characters have been read, or EOF or newline encountered,
- reads characters into *buffer* until *num* - 1 characters have been read, or EOF or the *delim* character encountered (*delim* is not read until next time),
- reads characters into *buffer* until a newline or EOF is encountered,
- or reads characters into *buffer* until a newline, EOF, or *delim* character is encountered (again, *delim* isn't read until the next get()).

For example, the following code displays the contents of temp.txt, character by character:

```
char ch;
ifstream fin( "temp.txt" );
while( fin.get(ch) )
  cout << ch;
fin.close();
```

getline

Syntax:

```
istream &getline( char *buffer, streamsize num );
istream &getline( char *buffer, streamsize num, char delim );
```

The getline() function is used with input streams, and reads characters into *buffer* until either:

- *num* - 1 characters have been read,
- a newline is encountered,
- an EOF is encountered,
- or, optionally, until the character *delim* is read. The *delim* character is not put into *buffer*.

---

open

Syntax:

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

The function open() is used with file streams. It opens *filename* and associates it with the current stream. The optional *mode* can be:

Mode	Meaning
ios::app	append output
ios::ate	seek to EOF when opened
ios::binary	open the file in binary mode
ios::in	open the file for reading
ios::out	open the file for writing
ios::trunc	overwrite the existing file

If open() fails, the resulting stream will evaluate to false when used in a Boolean expression. For example:

```
ifstream inputStream("file.txt");
if( !inputStream ) {
  cerr << "Error opening input stream" << endl;
  return;
}
```

---

put

Syntax:

```
ostream &put( char ch );
```

The function put() is used with output streams, and writes the character *ch* to the stream.

---

read

Syntax:

```
  istream &read( char *buffer, streamsize num );
```

The function read() is used with input streams, and reads *num* bytes from the stream before placing them in *buffer*. If EOF is encountered, read() stops, leaving however many bytes it put into *buffer* as they are. For example:

```
    struct {
      int height;
      int width;
    } rectangle;

    input_file.read( (char *)(&rectangle), sizeof(rectangle) );
    if( input_file.bad() ) {
      cerr << "Error reading data" << endl;
      exit( 0 );
    }
```

---

write

Syntax:

```
  ostream &write( const char *buffer, streamsize num );
```

The write() function is used with output streams, and writes *num* bytes from *buffer* to the current output stream.

---

seekp

Syntax:

```
  ostream &seekp( streamoff off, ios_base::seekdir way );
```

The seekp() function is used to set the position where the next character is to be inserted into the output stream. This position is specified by the offset value *off* and the base object *way* where *off* is relative to. It may take any of the following constant values as *way*:

value	offset is relative to...
ios_base::beg	beginning of the stream
ios_base::cur	current position in the stream
ios_base::end	end of the stream

For example, you can use `foo.seekg (0, ios::beg)` to go back to the beginning of the file for get.

---

seekg

Syntax:

```
  istream& seekg ( streamoff off, ios_base::seekdir way );
```

The *seekg()* function is used to set the position of the next character to be extracted from the input stream. This position is specified by the offset value *off* and the base object *way*. You may refer to the table in *seekp()* section for the constant values of *way*.

For example, you can use `foo.seekp (0, ios::beg)` to go back to the beginning of the file for put.

---

tellp

Syntax:

```
  streampos tellp();
```

The *tellp()* function is used to return the position of the current character in the output stream.

---

tellg

Syntax:

```
  streampos tellg();
```

The *tellg()* function is used to returns the position of the current character in the input stream.
