Lecture Nine

C++ File I/O System

Ref: Herbert Schildt, Teach Yourself C++, Third Edⁿ (Chapter 9)

© Dr. M. Mahfuzul Islam
Professor, Dept. of CSE, BUET

File I/O Basics

➢ For file I/O, the header file <fstream> need to be included. It defines several classes, including ifstream, ofstream and fstream. This classes are derived from istream and ostream, which are derived from ios. So, all I/O operations are applicable to file I/O.

➢ There are three types of streams: input stream, output stream and input/output stream. To create specific stream, following declarations are used:

```
ifstream in;   //decleration of input stream
ofstream out;   //decleration of output stream
fstream io;   //decleration of input/output stream
```

➢After defining streams, streams need to associate with file by using open() function. The prototypes are as follows:

```
void ifstream::open(const char *filename, openmode mode =ios::in);
void ofstream::open(const char *filename, openmode mode =ios::out | ios::trunc);
void fstream::open(const char *filename, openmode mode =ios::in | ios::out);
```

File I/O Basics

- **openmode is an enumeration defined in ios. The values of openmode are as follows:**

ios::app	All output are appended to the end of the file.
ios::ate	Seek to the end-of-file.
ios::binary	Causes a file to be opened in binary mode.
ios::in	Specify that file is capable of input.
ios::out	Specify that file is capable of output.
ios::trunc	Causes the contents of a pre-existing file destroyed and the file to be truncated to zero length.

- **The following fragments open an output file called *test*:**

```
ofstream mystream;
mystream.open("test");
or
ofstream mystream.open("test");
```

Constructor is run to open the file.

File I/O Basics

- **After opening a file, need to be confirmed:**

```
if (!mystream){
    cout << "cannot open file.\n";
}
or
if (!mystream.is_open()){
    cout << "File is not opened.\n";
}
```

- **Member function close() is used to close a file.**

```
mystream.close();
```

- **To check end of file, the member function eof() is used.**

```
mystream.eof();
```

File I/O Basics

```
#include <iostream>
#include <fstream>
using namespace std;

int main( int argc, char *argv[]){
   if (argc != 3 ){
       cout << “Usage: Convert <input> <output>\n”;
       return 1;
   }

   ifstream fin.open( argv[1] );
   ostream fout.open( argv[2] );

    if (!fout ){
        cout << “Cannot open output file.\n”;
        return 1;
    }

    if (!fin ){
       cout << “Cannot open input file.\n”;
       return 1;
   }
```

```
     char ch;

     fin.unsetf(ios::skipws);
     while(!fin.eof()){
          fin >> ch;
          fout << ch;
      }

      fin.close();
      fout.close();

      return 0;
}
```

Unformatted Binary File I/O

- Binary files access character by character.
- The following functions are used to access unformatted file I/O.

Function Prototype	Purpose
istream &get(char &ch);	Reads a single character from the associated stream and put that value in *ch*.
ostream &put(char ch);	Writes *ch* to the stream and returns a reference to the stream.
istream &read(char *buf, streamsize num);	Reads *num* bytes from the invoking stream and puts them in the buffer pointed to by *buf*.
Ostream &write(const char *buf, streamsize num);	Writes *num* bytes to the associated stream from the buffer pointed to by *buf*.
Stream gcount();	Number of characters read by the last unformatted input operation

Unformatted Binary File I/O

Some examples:

```
ifstream in( argv[1], ios::in | ios::binary );
ofstream out( argv[2], ios::out | ios::binary );
in.eof();
in.get(ch);
out.put(ch);
double num;
char str[] = "This is a test.";
out.write((char *) &num, sizeof(double));
out.write(str, strlen(str));
in.read((char *) &num, sizeof(double));
in.read(str, 14);
str[14] = '\0';
in.gcount();
```

➢**Consider the statement:**

```
out.write((char *) &num, sizeof(double));
```

Since *num* is double type, it is necessary to convert it into character before writing into a file.

➢**The type cast inside read() is necessary because C++ will not automatically convert a pointer of one type to another.**

More Unformatted I/O Functions

➢**There are several different ways, the get() function is overloaded.**

```
istream &get( char *buf, streamsize num );
istream &get( char *buf, streamsize num, char delim );
int get();
```

✓ **The first form reads characters into the array pointed to by *buf* until either *num* -1 characters have been read, a new line is found or the end of the file is encountered. New line character is not extracted and remains in the stream.**

✓ **The second form reads characters into the array pointed to by *buf* until either *num* -1 characters have been read, the character specified by *delim* is found or the end of the file is encountered.**

✓ **The third form returns the next character from the stream.**

More Unformatted I/O Functions

- **The prototypes of getline() function are as follows:**

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

✓ **The first form reads characters into the array pointed to by *buf* until either *num* -1 characters have been read, a new line is found or the end of the file is encountered. New line character is extracted but it is not put into *buf*.**

✓ **The second form reads characters into the array pointed to by *buf* until either *num* -1 characters have been read, the character specified by *delim* is found or the end of the file is encountered.**

More Unformatted I/O Functions

Function Prototype	**Purpose**
int peek();	Returns next character in the input stream without removing it from the stream. It returns EOF if end of file is encountered.
istream &putback(char c);	Returns the last character from a stream to the stream .
ostream flush();	✓ Information is stored in internal buffer until the buffer is full before writing them to physical device. ✓ flush() function force the information to be written into the physical device before buffer is full.

Some examples:

```
char ch = in.peek();  \\ see what type of character is next
in.putback(*p) ;      \\ return character to stream
*p ='\o';
```

Random Access File

- **Random access to a file can be performed by seekg() and seekp() functions.**
- **There are two pointers associated with C++: get pointer and put pointer.**
  - **The get pointer specifies where in the file the next input operation will occur.**
  - **The put pointer specifies where in the file the next output operation will occur.**

Random Access File

Function Prototype	**Purpose**
pos_type tellg();	Returns current position of get pointer. pos_type is an integer type defined by ios and capable of holding the largest value that defines a file pointer.
Pos_type tellp();	Returns current position of put pointer.
istream &seekg(off_type offset, seekdir origin);	Moves the current get pointer *offset* number of bytes from the origin. off_type is an integer type defined by ios and capable of holding the largest value that offset can be. seekdir is an enumeration defined by ios that has three values: ios::beg seek from beginning; ios::cur seek from current location; ios::end seek from end
istream &seekg(pos_type position);	
ostream &seekp(off_type offset, seekdir origin);	Moves the current put pointer *offset* number of bytes from the origin.
ostream &seekp(pos_type position);	

Checking File I/O Status

- **I/O status is stored in the error flags.**
- **The following member functions are used for checking I/O status:**

Function Prototype	**Purpose**
iostate rdstate();	iostate is an enumeration defined by ios that includes goodbit No error occurred eofbit end of file has been encountered failbit A nonfatal I/O error has occurred badbit A fatal I/O error has occured
bool bad();	Returns true if bad bit is set.
bool eof();	Returns true if end of file is encountered.
bool fail();	Returns true if failbit is set.
bool good();	Returns true if there are no errors.
void clear(iostate flags = ios::goodbit);	If flags is goodbit (as it is bydefault), all error flags are cleared.

Checking File I/O Status

```
#include <iostream>
#include <fstream>
using namespace std;

void checkstatus(ifstream &in);

int main( int argc, char *argv[]){
    if ( argc != 2 ){
          cout << “Usage: Convert <input>
                          <output>\n”;
          return 1;
    }

    ifstream in( argv[1] );
    if ( !in ){
         cout << “Cannot open input file.\n”;
         return 1;
    }

    char c;
    while(in.get(c)){
         cout << c;
         checkstatus(in);
    }
```

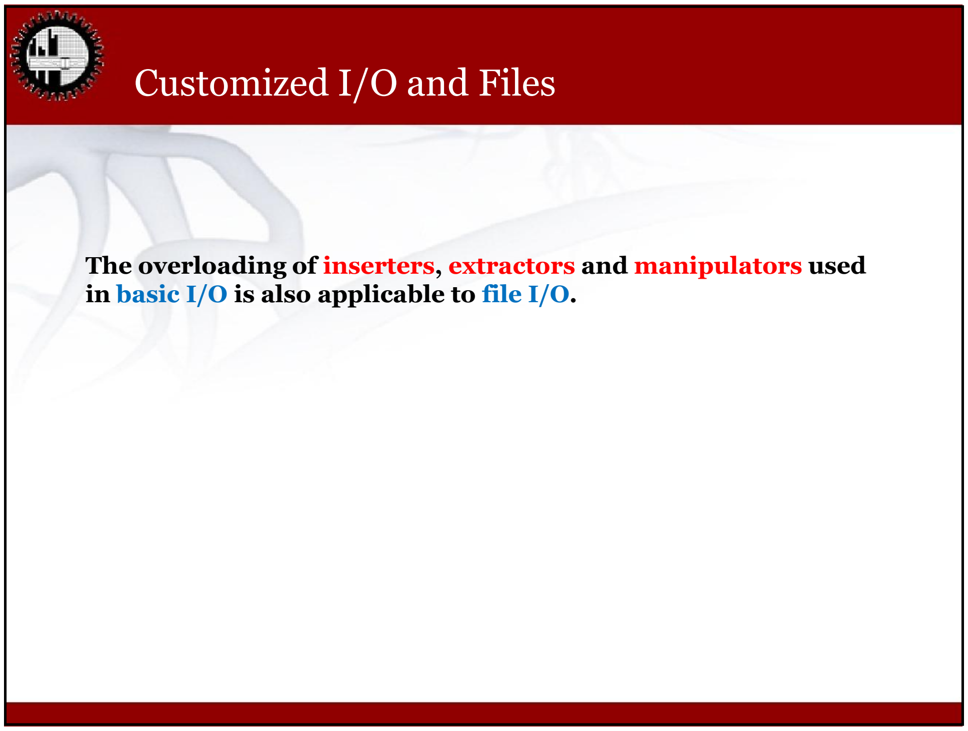
```
    checkstatus(in);
    in.close();

    return 0;
}
```

```
void checkstatus ( ifstream &in){
    ios::iostate i;

    i = in.rdstate();

    if ( i & ios::eofbit) cout << “End of file has
            encountered.\n”;
    else if ( i & ios::failbit) cout << “Nonfatal I/O
            error.\n”;
    else if ( i & ios::badbit) cout << “Fatal I/O
             error.\n”;
}
```



Customized I/O and Files

The overloading of inserters, extractors and manipulators used in basic I/O is also applicable to file I/O.