# Disk File I/O with Streams

## Stream Classes

- A *stream* is a general name given to a flow of data. In C++ a stream is represented by an object of a particular class.

## The ios Class

- **The ios class is the granddaddy of all the stream classes, and contains the majority of the features you need to operate C++ streams.**
- **The istream Class**
- **The istream class, which is derived from ios, performs input-specific activities, or extraction.**

lists the functions you'll most commonly use from the `istream` class.

| *Function* | *Purpose* |
|---|---|
| `>>` | Formatted extraction for all basic (and overloaded) types. |
| `get(ch);` | Extract one character into `ch`. |
| `get(str)` | Extract characters into array `str`, until '`\n`'. |
| `get(str, MAX)` | Extract up to `MAX` characters into array. |
| `get(str, DELIM)` | Extract characters into array `str` until specified delimiter (typically '`\n`'). Leave delimiting char in stream. |
| `get(str, MAX, DELIM)` | Extract characters into array `str` until `MAX` characters or the `DELIM` character. Leave delimiting char in stream. |
| `getline(str, MAX, DELIM)` | Extract characters into array `str`, until `MAX` characters or the `DELIM` character. Extract delimiting character. |
| `putback(ch)` | Insert last character read back into input stream. |
| `ignore(MAX, DELIM)` | Extract and discard up to `MAX` characters until (and including) the specified delimiter (typically '`\n`'). |
| `peek(ch)` | Read one character, leave it in stream. |
| `count = gcount()` | Return number of characters read by a (immediately preceding) call to `get()`, `getline()`, or `read()`. |
| `read(str, MAX)` | For files—extract up to `MAX` characters into `str`, until `EOF`. |
| `seekg()` | Set distance (in bytes) of file pointer from start of file. |
| `seekg(pos, seek_dir)` | Set distance (in bytes) of file pointer from specified place in file. `seek_dir` can be `ios::beg`, `ios::cur`, `ios::end`. |
| `pos = tellg(pos)` | Return position (in bytes) of file pointer from start of file. |

## The ostream Class

- The ostream class handles output or insertion activities.

| *Function* | *Purpose* |
|---|---|
| `<<` | Formatted insertion for all basic (and overloaded) types. |
| `put(ch)` | Insert character `ch` into stream. |
| `flush()` | Flush buffer contents and insert newline. |
| `write(str, SIZE)` | Insert `SIZE` characters from array `str` into file. |
| `seekp(position)` | Set distance in bytes of file pointer from start of file. |
| `seekp(position, seek_dir)` | Set distance in bytes of file pointer, from specified place in file. `seek_dir` can be `ios::beg`, `ios::cur`, or `ios::end`. |
| `pos = tellp()` | Return position of file pointer, in bytes. |

- Working with disk files requires another set of classes: ifstream for input, fstream for both input and output, and ofstream for output. Objects of these classes can be associated with disk files, and we can use their member functions to read and write to the files.

# Writing Data

```
// formato.cpp
// writes formatted output to a file, using <<
#include <fstream>                  //for file I/O
#include <iostream>
#include <string>
using namespace std;

int main()
   {
   char ch = 'x';
   int j = 77;
   double d = 6.02;
   string str1 = "Kafka";           //strings without
   string str2 = "Proust";          //   embedded spaces

   ofstream outfile("fdata.txt"); //create ofstream object

   outfile << ch                    //insert (write) data
           << j
           << ' '                   //needs space between numbers
           << d
           << str1
           << ' '                   //needs spaces between strings
           << str2;
   cout << "File written\n";
   return 0;
   }
```

- Here we define an object called outfile to be a member of the ofstream class. At the same
- time, we initialize it to the file FDATA.TXT. This initialization sets aside various resources for thefile, and accesses or *opens* the file of that name on the disk. If the file doesn’t exist, it is created.

- The outfile object acts much as cout did in previous programs, so we can use the insertion operator (<<) to output

  variables of any basic type to the file.
- When the program terminates, the outfile object goes out of scope. This calls its destructor,which closes the file, so we don’t need to close the file explicitly.

## Reading Data

- We can read the file generated by FORMATO by using an ifstream object, initialized to the

  name of the file.

  The file is automatically opened when the object is created. We can then read from it using the extraction (>>) operator.

```
// formati.cpp
// reads formatted output from a file, using >>
#include <fstream>                  //for file I/O
#include <iostream>
#include <string>
using namespace std;

int main()
   {
   char ch;
   int j;
   double d;
   string str1;
   string str2;


   ifstream infile("fdata.txt");   //create ifstream object
                                   //extract (read) data from it
   infile >> ch >> j >> d >> str1 >> str2;

   cout << ch << endl              //display the data
        << j << endl
        << d << endl
        << str1 << endl
        << str2 << endl;
   return 0;
   }
```

- Here the ifstream object, which we name infile, acts much the way cin did in previous programs.
- Provided that we have formatted the data correctly when inserting it into the file, there’s
- no trouble extracting it, storing it in the appropriate variables, and displaying its contents. The
- program’s output looks like this:
- x
- 77
- 6.02
- Kafka
- Proust