C++ Input and Output

* the standard C++ library has a collection of classes that can be used for input and output
* most of these classes are based on a stream abstraction, the input or output device is viewed as a stream of characters
* so far we have used two basic output streams, cout and cerr, these streams are objects of the class ostream
* the ostream class has a << operator that is used to output a value onto the stream, the ostream class defines this operator for most of the standard C++ data types
* as we have seen before we can extend the range of this operator by defining our own << operator for any of the classes that we define
* for input we have one predefined object, cin, which is an object of the class istream
* this class uses the >> operator to input a value, it defines the >> operator for most of the standard C++ data types
* for each of the classes that we define, we can produce a version of the >> operator to input values of that class
* the >> operator normally returns the istream that it was called on, if it can't read an item of the expect type it returns zero
* this could happen if the end of file has been reached, or the next item on the input stream doesn't match the type of value that >> is expecting
* we can use this fact to write input loops, for example

```
int i;

while(TRUE) {
  if(cin >> i)
    cout << i;
  else break;
}
```

* this loop will copy integers from cin to cout until the end of file is reached or a non-integer appears on cin
* **we could convert this into a template function that copies any type of value from an istream to an ostream**

```
template void iocopy(T z, istream& is, ostream& os) {
  while(is >> z) os << z;
}
```

* even though we don't really need to pass z to this function, we need to have it as a parameter, otherwise the iocopy function won't compile
* the standard input operations skip white space the same way that the C input functions do, for most applications this is okay, we don't care about the white space between values
* but it we want to copy a text file exactly the way it is, including the whitespace characters this approach doesn't work

* the istream call defines a get function that handles this problem, there are two versions of this function
* the get(char&) version of this function reads a single character from the istream and places it in its parameter, this function returns NULL when the end of file is reached
* the other version of this function has the following declaration:

```
get(char* p, int n, char='\n')
```

* this procedure reads one or more characters into the buffer pointed to by p
* the parameter n gives the size of the buffer, **get** will always terminate the character string that is read with a 0, so there is room for at most n-1 characters in this buffer
* the get operation terminates when either an end of file is encountered, or n-1 characters have been read, or the character that is passed as the third parameter is encountered
* this terminating character isn't placed in p, and isn't read by the get operation, to determine whether get has overflowed the buffer, you must check whether the next character in the stream is the terminating character

```
#include <strstream.h>   // useful for read/write arrays
#include <iostream.h>    // cin and cout -- also cerr
#include <fstream.h>     // reading/writing files
```

Stream States

* each stream has a state, this state is defined in the ios class that both istream and ostream derive from
* there are many functions for manipulating the state of a stream, but there are only four that you really need to know about for most I/O programming:
* **cin.**eof() - returns true if the end of file has been seen
* **cin.**fail() - returns true if the next operation will fail
* **cin.**bad() - returns true if the stream has been corrupted
* **cin.**good() - returns true if the next operation might succeed

Formatted I/O

* the C printf and scanf functions provided formatted I/O, some of this is handled by the **>>** and **<<** operators, but other parts, such as field width, aren't
* formatting is handled by the ios class, this class handles most of the interaction between the stream classes and their character buffers
* we will briefly look at some of the formatting functions that are available
* the **flush()** function is used to transfer all the characters in an output buffer to the device it is connected to

* this function is mainly used for streams connected to a display screen, so information is on the screen when the user needs to see it
* the **tie()** function is used to tie together an input and an output stream, in an interactive application we often send information to the screen, or make requests and expect the user to respond to them, in this case the ostream needs to be flushed before each input operation
* the tie() function is called on an istream and it's parameter is the ostream that is to be flushed before each input operation
* the following call is automatically made for us:

```
cin.tie(&cout)
```

* the width of an output field can be controlled by calling the width() function, the parameter to this call is the minimum width of the output field
* the next output operation that requires formatting will produce an output field of at least this width, if the value needs more characters the field will be wider, otherwise it will be padded with a fill character
* the fill character is specified using the fill() procedure, the parameter to this procedure is a char giving the character to be used for padding, by default this character is the blank

```
cout << width(5) << i << width(4) << j << flush;
```

File Positioning

* both the istream and ostream classes have functions for positioning the character pointer within the stream, these functions only work if the stream is connected to a disk file
* for the ostream class we have the following functions:

ostream& seekp(streampos);
ostream& seekp(streamoff, seek_dir); streampos tellp()

* the streampos type is an absolute position within the stream and the streamoff type is an offset from a reference position
* the possible reference positions (values of type seek_dir) are:

* **beg** - beginning of the file
* **cur** - current position in the file
* **end**- end of the file

* these functions work in essentially the same way as the stdio functions
* for istreams we have the following functions:

istream& seekg(streampos);
istream& seekg(streamoff, seek_dir); streampos tellg();

* these functions operate in the same way as the corresponding ostream functions
* they have been assigned different names since a stream can be used for both input and output, therefore, a value of stream class could have an istream and ostream side and both could be active at the same time
* by having two sets of functions the position we are reading from could be different from the position we are writing to, even in the same file

The fstream Classes

* the fstream classes are designed for reading and writing files, there are three related classes:

ifstream - input from a file
ofstream - output to a file
fstream - both input and output to a file

* when an ifstream file object is created it automatically assumes that the file will be opened for input, the first parameter to its constructor is the name of the file

* when an ofstream file object is created it automatically assumes that the file will be opened for output, the first parameter to its constructor is the name of the file
* the destructors for both of these objects will close their files, or you can explicitly close the file by calling the close() function on the object
* the constructors for these classes can take a second parameter that specifies how the file is opened, the values for these flags are specified in the ios class
* some of the possible values are:

- in - open for reading
- out - open for writing
- ate - open and seek to end of file
- app - append
- trunc - truncate file to zero length
- nocreate - fail if file does not exist
- noreplace - fail if the file exists

* if we wanted to open a file for both reading and writing we can do the following:

```
fstream index("database.index", ios::in | ios::out);
```

String Steams

* the istrstream and ostrstream classes can be used to read and write from character arrays, these classes operate in the same way as the sscanf and sprintf functions in stdio

* the first parameter to their constructors is a pointer to the character array, and the second parameter is the size of the array. The array must be allocated before the constructor is called

* in the case of an ostrstream, an end-of-file condition is signalled when the end of the string is reached. An ostrstream will not overwrite the end of the character array that it is passed

* the istrstream class expects a NULL terminated character string and extracts characters from this string as it reads. Again an end of file will be signalled when the end of the string is reached

* all the istream and ostream functions can be performed on these classes

```
#include <strstream.h>      // useful for read/write arrays
#include <iostream.h>       // cin and cout -- also cerr
#include <fstream.h>        // reading/writing files
```