The new input/output classes

The iostream classes implement most of the features of AT&T version 2.0 iostream library classes, and most of the features of the ANSI X3J16 library draft (which is based on the AT&T design). The iostream classes replace all of the old stream classes in previous versions of libg++. It is not totally compatible, so you will probably need to change your code in places.

The streambuf layer

The lower level abstraction is the **streambuf** layer. A **streambuf** (or one of the classes derived from it) implements a character source and/or sink, usually with buffering.

Classes derived from **streambuf** include:

- A **filebuf** is used for reading and writing from files.
- A **strstreambuf** can read and write from a string in main memory. The string buffer will be re-allocated as needed it, unless it is ``frozen''.
- An **indirectbuf** just forwards all read/write requests to some other buffer.
- A **procbuf** can read from or write to a Unix process.
- A **parsebuf** has some useful features for scanning text: It keeps track of line and column numbers, and it guarantees to remember at least the current line (with the linefeeds at either end), so you can arbitrarily backup within that time. WARNING: The interface is likely to change.

- An **edit_streambuf** reads and writes into a region of an **edit_buffer** called an **edit_string**. Emacs-like marks are supported, and sub-strings are first-class functions. WARNING: The interface is almost certain to change.

The istream and ostream classes

The stream layer provides an efficient, easy-to-use, and type-secure interface between C++ and an underlying **streambuf**.

Most C++ textbooks will at least give an overview of the stream classes. Some libg++ specifics:

istream::get(char* s, int maxlength, char terminator='\n') Behaves as described by Stroustrup. It reads at most maxlength characters into s, stopping when the terminator is read, and pushing the terminator back into the input stream.

istream::getline(char* s, int maxlength, char terminator = '\n') Behaves like get, except that the terminator is read (and not pushed back), though it does not become part of the string.

istream::gets(char ss, char terminator = '\n')** Reads in a line (as in get) of unknown length, and places it in a free-store allocated spot and attaches it to **ss**. The programmer must take responsibility for deleting ***ss** when it is no longer needed.

ostream::form(const char* format...) Outputs **printf**-formated data.

The SFile class

SFile (short for structure file) is provided both as a demonstration of how to build derived classes from **iostream**, and as a useful class for processing files containing fixed-record-length binary data. They are created with constructors with one additional argument declaring the size (in bytes, i.e., **sizeof** units) of the records. **get** will input one record, **put** will output one, and the **[]** operator, as in **f[i]**, will position to the i'th record. If the file is being used mainly for random access, it is often a good idea to eliminate internal buffering via **setbuf** or **raw**. Here is an example:

```
class record
{
  friend class SFile;
  char c; int i; double d;     // or anything at all
};

void demo()
{
  record r;
  SFile recfile("mydatafile", sizeof(record), ios::in|ios::out);
  recfile.raw();
  for (int i = 0; i < 10; ++i)  // ... write some out
  {
    r = something();
    recfile.put(&r);            // use '&r' for proper coercion
  }
  for (i = 9; i >= 0; --i)      // now use them in reverse order
  {
    recfile[i].get(&r);
    do_something_with(r);
  }
}
```

The PlotFile Class

Class **PlotFile** is a simple derived class of **ofstream** that may be used to produce files in Unix plot format. Public functions have names corresponding to those in the **plot**(5) manual entry.

C standard I/O

There is a complete implementation of the ANSI C stdio library that is built on *top* of the iostream facilities. Specifically, the type **FILE** is the same as the **streambuf** class. Also, the standard files are identical to the standard streams: **stdin == cin.rdbuf()**. This means that you don't have to synchronize C++ output with C output. It also means that C programs can use some of the specialized sub-classes of streambuf.

The stdio library (**libstdio++**) is not normally installed, because of some difficulties when used with the C libraries version of stdio. The stdio library provides binary compatibility with traditional implementation. Unfortunately, it takes a fair amount of care to avoid duplicate definitions when linking with both **libstdio++** and the C library.