SFU CMPT-212 2008-1 Topic: Streams

Ján Maňuch

E-mail: jmanuch@sfu.ca

Sunday 6th April, 2008

Streams

types:

- *input* (`istream`)— we can read from the stream
- *output* (`ostream`) — we can write to the stream
- *i/o* (`iostream`) — allows read-and-write access to the stream

types:

- *console input and output* (`#include <iostream>`) — `cin`, `cout`, `cerr` and `clog`
- *file input and output* (`#include <fstream>`)
- *string streams* (`#include <sstream>`)
- *C-style string streams* (`#include <strstream>`)

Buffers

- each stream has a buffer (an i/o stream has 2 buffers, one for reading, one for writing)
- to force sending all information (from a buffer) to an *output* stream, you can use `flush()` member function, or send manipulator `flush` (similar to `endl` or `ends`) to the stream:
  `cout <<10<<flush;`

Formatting

- using *member functions*:

```
1 cout.width(10);    // only next item (default: 0)
2 cout.precision(5); // number of digits (default: 6)
3 // number of decimal digits (in fixed or scientific)
4 cout.fill('*');    // (default ' ')
```

- using `setf()` and `unsetf()` member function and `ios_base` constants:

```
1  fmtflags oldf=cout.setf(ios_base::showpoint);
2  // ios_base::boolapha - display bools as true/false
3  // ios_base::showbase - display base prefixes
4  // ios_base::showpoint - display '.'
5  // ios_base::uppercase - display 'E' instead of 'e'
6  // ios_base::showpos - display '+'
7  cout.setf(oldf);
8  //         flag value:    field mask:
9  cout.setf(ios_base::hex, ios_base::basefield);
10 cout.unsetf(ios_base::basefield); // revert to default
```

flag value	**field mask**
`dec`, `oct` or `hex`	`basefield`
`scientific` or `fixed` (show trailing zeros)	`floatfield`
`left`, `right` or `internal`	`adjustfield`

- using *manipulators* — send them directly to stream:
  - *standard manipulators*:

```
1 cout <<endl<<boolalpha<<(1==2)<<flush;
```

    other manipulators:
    `boolalpha`, `noboolalpha`, `showbase`, `noshowbase`,
    `showpoint`, `noshowpoint`, `uppercase`, `nouppercase`,
    `showpos`, `noshowpos`
    `dec`, `hex`, `oct`
    `fixed`, `scientific`
    `left`, `right`, `internal`
  - the `<iomanip>` *header file manipulators:*
    `setw(10)` — set width (valid only for the next output)
    `setprecision(5)`, `setfill('*')`
    Example:

```
1 cout <<setprecision(2)<<setw(5)<<setfill('*')<<3.14;
2 // prints "**3.1"
```

Stream states

- `eof()` — returns `true` if end-of-file reached
- `bad()` — returns `true` if the stream is corrupted (read error, etc..)
- `fail()` — returns `true` if stream failed to read or write the expected character
- `good()` — returns `true` if everything is ok;
  Note. a stream is automatically converted to `bool` with value `good()`:

```
1 while (cin) { ... }
```

- it's not possible to read from or write to the stream, if it's not in the "good" state
- `clear()` resets the stream to the "good" state

Random access

- `tellg()` and `seekg()` — read and set the "get"-position for the input stream
- `tellp()` and `seekp()` — read and set the "put"-position for the output stream

C-string streams

- *deprecated* (not anymore supported by C++ standard) $\Longrightarrow$ use `string` streams instead
- the header file `<strstream>`
- used c-style string (`char*`) as a buffer (can be specified in the constructor or can be obtained with `str()` member function **Note.** `str()` freezes the buffer)
- allows random access; 3 types: `istrstream` (input), `ostrstream` (output), `strstream` (i/o)

String streams

- the header file `<sstream>`
- uses a `string` as a buffer
- allows random access
- 3 types: `istringstream` (input), `ostringstream` (output), `stringstream` (i/o)
- `str()` member function — returns the buffer (`string`) of the stream
- useful when converting any type to string

Example:

```
1  stringstream ss();
2  ss << "test " << 1023;
3  string t;
4  ss >> t; // put "test" into t
5  ss << t; // ss now contains: "test 1023test"
6  int number;
7  ss >> number; // put 1023 into number
8  cout << ss.str() << endl; // extract string
9  cout << ss.str().c_str() <<endl; // c-string
10
11 ss.seekg(0); // start reading from the begining
12 cout << "One char: '"<<ss.get()<<"'"<<endl; // 't'
13 ss.seekp(ss.tellg()); // set writing pos. to reading pos.
14 ss << 5.7; // overwrite content
15 cout << ss.str() << endl; // "t5.7 1023 test"
```

Example: str.cpp

File streams

- the header file `<fstream>`
- write and/or reads data from/to a file
- allows random access
- all 3 types: `ifstream` (input), `ofstream` (output), `fstream` (i/o)
- `open("data.txt")` — associate the stream with a file `data.txt` (or the file name can be specified in the constructor)
- `close()` — close connection

Examples:

```
1 ifstream fi;
2 fi.open("data.txt");
3 int i;
4 fi >> i; // read an integer from the file
5 fi.close();
```

```
1 ofstream fo("data1.txt",
2   ios_base::out | ios_base::app);
3 // connect to data.txt, no open() needed
4 // append output to the end of file
5 fo <<"Hello" << x << endl;
```

Note:

- default state of `ofstream` is `ios_base::out | ios_base::trunc` — deletes the content of the file when connecting!
- state `ios_base::binary` — data read and written in binary form (more space efficient)
- setting position to end of file: `fo.seekp(0, ios_base::end);`
- for more efficient reading, use block operations (`read` and `write`):

  Example: count.cpp