Streams - Object input and output in C++

Dr. Donald Davendra *Ph.D.*

Department of Computing Science, FEI VSB-TU Ostrava

Streams - Object input and output in C++ Basic concepts

The **character** is an elementary text information, as given in the ASCII table - 8 bit code, character data types are char, string:
- Printable characters from 32 to 127,
- Control escape sequence,
- Characters of national alphabets.

16-bit Unicode, character data types wchar_t, string wstrng.
Word - a sequence of characters. Words are divided from each other by punctuation marks and indenting.
line of text - words ending sequence symbol (s) transition to a new line.
Length of line? String class creates the necessary space.
At the level of OS option to redirect input and output - programs that read from standard input and write to standard output are called **filters**.

Text Input / Output:

1. To end the input from keyboard, Win32 `ctrl-z`, Unix `ctrl-d`.
2. Standard input and output reads in one line at a time. The program can read input until the moment you end the row by pressing **enter**.
3. `iostream` header.

Classes for input and output stream

Classes for input and output stream - text and binary data stream
Header used is `iostream`, and the **class** is `ios`
Classes for input and output are based on templates - the same interface for input and output text, regardless of its encoding.
tool: classes and objects use the input current *i*, as input and output, whereas files streaming use *f* as file.
Type **char**, Template: **basic** + class name:

class	stream description
streambuf	buffered
ios	input and output
istream	input
ostream	output
fstream	file
ifstream	input file
ofstream	output file

Classes for standard I/O and file streams.

Overloaded operators for input / output:

1. For output current (*put to, insertion*),
2. For input stream (*get from, extraction*).

Objects standard text input and output

Standard streams for C++ object-oriented I / O.

description	stream	std. equipment
cin	std. input	keyboard
cout	std. output	screen
cerr	std. error output	screen
clog	error output line by line	screen

Output formatting

Operator <<
Left operand - an object of type **ostream**
right operand - any type (for which the output is defined).

```
cout << "Ahoj!" << endl;
int i = 5; double d = 123.456;
cout << "i =" << i << "\td=" << d << endl;
```

Format flags for text streams

name	description
skipws	Sets the skipws format flag for the str stream.
left /right	Sets the adjustfield format flag for the str stream to left/right.
internal	Sets the adjustfield format flag for the str stream to internal.
dec	Sets the basefield format flag for the str stream to dec.
oct	Sets the basefield format flag for the str stream to oct.
hex	Sets the basefield format flag for the str stream to hex.
showbase	Sets the showbase format flag for the str stream.
showpoint	Sets the showpoint format flag for the str stream.
uppercase	Sets the uppercase format flag for the str stream.
showpos	Sets the showpos format flag for the str stream.
scientific	Sets the floatfield format flag for the str stream to scientific
fixed	Sets the floatfield format flag for the str stream to fixed.
unitbuf	Sets the unitbuf "format" flag for the str stream.
stdio	used for stdout a stderr
boolalpha	Sets the boolalpha format flag for the str stream.
adjustfield	internal parameterl, left right
basefield	dec, oct, hex
floatfield	fixed a scientific

Formatting output using status flags formatting status flags defined in the `ios` class in the class namespace, hence eg `ios :: hex`.
Enumerated data type for constants of format state flags `fmtflags` is defined in the `ios` class.
You can combine `bit` — or simply add - values to powers of two.
methods `setf ()` to set flags or `unsetf ()` for their removal.
`cout` object has access for other methods more than just setting and the status format flags. There are the particular methods `put()` and `write()`.
Also these methods allows output to a text stream, which, however, unlike the overloaded operator <<, the value is not influenced by formatting symptoms.

```
/*********************
 * file os-tflg.cpp
 *********************/

#include <iostream>

using namespace std;

int main() {
        cout << 123 << '\t' << 123.456e15 << '\t'
          << 0xfe << '\t' << true << endl;

        cout.setf(ios::hex | ios::showbase | ios::bool

        cout.unsetf(ios::dec);
        // needed by some compilers
        cout << 123 << '\t' << 123.456e15 << '\t'
          << 0xfe << '\t' << true << endl;
```

```
123       1.23456e+017     254      1
0x7b      1.23456e+017     0xfe     true
```

Formatting output with manipulators

handlers:

1. Special operators similar functions;
2. Used as its argument a reference to the stream, which also returned;
3. May be part of the command output.

Formatting output with manipulators

manipulator	description	property
boolalpha	log. values textually	I/O
dec	decimal	I/O
endl	newline	O
ends	terminates line. and empties memory	O
fixed	sets a fixed flag	O
flush	empty the cache	O
hex	hexadecimal	I/O
internal	sets a flag	O
left	decimal system	I/O
noboolalpha	cancel flag	I/O
noshowbase	cancel flag	O
noshowpoint	cancel flag	O
noshowpos	cancel flag	O
noskipws	cancel flag	I
nounitbuf	cancel flag	O
nouppercase	cancel flag	O
oct	octel	I/O

manipulator	description	property
resetiosflags(fmtflags f)	resets flags	I/O
right	sets a flag	O
scientific	sets a flag	O
setbase(int base)	e.g (0, 8, 10, 16)	I/O
setfill(int ch)	fills the characters	O
setiosflags(fmtflags f)	sets the flags	I/O
setprecision(int p)	output formatted to precision	O
setw(int w)	width of the field	O
showbase	sets a flag	O
showpoint	sets a flag	O
showpos	sets a flag	O
skipws	sets a flag	I
unitbuf	sets a flag	O
uppercase	sets a flag uppercase	O
ws	skip introductory whitespace	I

```
/********************
 * file os-tman.cpp
 ********************/

#include <iostream>
#include <iomanip>

using namespace std;

int main() {
         cout << setiosflags(ios::showbase)
              << hex << 123 << '\t'
              << setprecision(3) << setw(10)
              << 123.456e15
              << '\t' << oct << 0xfe << endl;

 return 0;
} // void main()
```

```
    0x7b     1.23e+017       0376
```

Overloading the I/O operators for custom classes

```
ostream & operator << (ostream & os, class id);
istream & operator >> (istream & os, class & id);
```

The need to transfer the reference to the stream and return the same reference is seemingly pointless. But only until you realize that calling the operator functions and finishing it definitely changes the status of the stream - either we wrote something in it, or we read something from it. Therefore, after we have outlined the activities of the operator's command to leave a status link for transferring the data stream.
Writing object id in the stream does not require further attention. Reading an object id of the current requires attention - we mentioned object operator functions to load, so pass it by reference.

```
/ soubor stro-cls.cpp
#include <cmath>
#include <iostream>
#include <iomanip>

using namespace std;

const double M_PI = 3.14159265358979323846L;

class polar {
 double r, fi;
public:
 polar(double a, double b) {r = a; fi = b;};
 double get_r(void) {return r;};
 double get_fi(void) {return fi;};
};

ostream& operator<<(ostream& os, polar& po){
 os << setw(8) << setprecision(4) << "r ="
    << po.get_r()
    << "\tfi =" << po.get_fi() << endl;
 return os;
}
```

```
// ************* main() ******************
void main() {
 polar x(1.0, M_PI / 2.0);
 cout << x;      // operator << (cout, x)
} // void main()
```

```
       r =1       fi =1.571
```

Input from a text stream

Input from a text stream is connected to the operator >>.
The left operand must be an object of type **istream**, right operand can be of any type for which the input is defined.

```
int i;
double d ;

cin >> i >> d;

istream& operator>>(istream& is , polar& po){

        double a, b;
        is >> a >> b;
        po = polar(a, b);

return is ;
}
```

```
// soubor stri-cls.cpp

#include <cmath>
#include <iostream>
#include <iomanip>

using namespace std;

const double M_PI = 3.14159265358979323846L;

class polar {
 double r, fi;
public:
 polar() : r(0.0), fi(0.0) {};
 polar(double a) : r(a) {
   fi = 0.0;};
 polar(double a, double b)
   : r(a), fi(b) {};
 polar operator+ (polar b);
 double get_r(void) {return r;};
 double get_fi(void) {return fi;};
};
```

```
ostream& operator<<(ostream& os, polar& po){
 os << setw(8) << setprecision(4) << "r ="
    << po.get_r()
    << "\tfi =" << po.get_fi() << endl;
 return os;
}
istream& operator>>(istream& is, polar& po){
 double a, b;
 is >> a >> b;
 po = polar(a, b);
 return is;
}
polar polar::operator+(polar b){//only I.and II.quadrant
 double abs, fi, u;
 fi = this->fi - (this->fi - b.fi) / 2.0;
 if (this->fi < b.fi)
    u = M_PI + (this->fi - b.fi);
 else
    u = M_PI - (this->fi - b.fi);
 abs = sqrt(this->r * this->r + b.r * b.r
       - 2.0 * this->r * b.r * cos(u));
 return polar(abs, fi);
}
```

```
void main() {
 polar x,
       y(1.0, M_PI / 2.0);
 cin >> x;
 x = x + y;
 cout << x; // operator << (cout, x)
} // void main()
```

```
   1 .5
        r =1.72    fi =1.035
```

To enter additional methods are defined: for example, **get()** and **read()**. The last byte is loaded can be undone using the **putback()**. See the next incoming character without reading it can be accomplished using the **peek()** method. Note, that the current text can be input and output using the window buffer.

```
// soubor stri-misc.cpp

#include <iostream>
#include <iomanip>

using namespace std;

int main() {
 bool b;
 cout << "enter 'true' or 'false'" << endl;
 cin >> boolalpha >> b;
 cout << b << '\t' << boolalpha << b << endl;

 int d, o;
 cout << "enter two octal numbers"
      << endl;
 cin >> oct >> d >> o;

 char s[10];
 cout << "enter string" << endl;
 cin >> setw(sizeof(s)) >> s;//s "will not overflow"

 cout << endl << "d =" << d << "\to =" << o
      << endl << "s = " << s << endl;


} // void main()
```

```
enter 'true' or 'false'
false
0 false
enter two octal numbers
0123 0765
enter string
Hello!

d =83 o =501
s = Hello!
```

Files

The **file** is a sequence of characters (bytes) terminated with a special combination that has the contents of the file that are not the end of file, EOF symbolically.
The text file contains lines of text. The binary file contains valuesin the same way as they are stored in computer memory.
external file name - file name at the operating system;
internal file name - identifying file in the program in C++, it most frequently the file called through he object, through which we work with the file.
Each **stream** can be opened and closed. When you open the stream we provide connection between the internal and the external file name.
When you open mode determine our approach to the data in the stream
Closing the current stream enables the OS updates the directory information to the current state of the file that was associated with the stream.

Object text input and output from / to files

- Consistent with standard text input and output (in open streams):

```
double f;
Input current >> f;
Output current << "has been loaded:" << f;
```

- uses the header `#include <fstream>`
and the methods are:
- `ofstream` for output
- `ifstream` for input.

opening a file-stream constructor:
`ifstream input_current (file.in);`
`ofstream output_current (file.out);`
or `open()`
`close().`
unsuccessful opening of current-NULL:

```
if (input_current == 0)
         cerr << "error opening input"
                  "file <file.in> ";
else
         ...
```

Operating modes are set using **ios::**

mode	description
app	connects the data is always written to the end of file
ate	opens and sets the end of file
in	when opening the set reading mode (default for ifstream)
out	sets the mode when opening the notation (default for ofstream)
binary	opens the file in binary mode
trunc	cancel the contents of the file, if it exists

When working with a binary stream is an essential to use block transfer data.

random access - Items have a binary file known size can calculate their position and to read or write to the specified position.

put(), **get()** working with unformatted one byte (character),

read(), **write()**: arguments address and number of bytes.

Method **eof()** returns true when reaching the end of the file.

```
// file strof-b1.cpp
// creates file odmoc.dta

#include <fstream>
#include <cmath>

using namespace std;

const char *name = "odmoc.dta";

int main() {
 ofstream ofs(name, ios::out | ios::binary);
 if (ofs != 0) {
   double f;
   for (int i = 0; i < 100; i++) {
     f = sqrt((float) i+1);
     ofs.write((const char *) &f, sizeof(f));
   } // for (;;)
   ofs.close();
 } // if (ofs != 0)

 return 0;
} // void main()
```

```
 +1.000 +1.414  +1.732  +2.000  +2.236  +2.449  +2.646  +2.828
+3.317  +3.464  +3.606  +3.742  +3.873  +4.000  +4.123  +4.243
+4.583  +4.690  +4.796  +4.899  +5.000  +5.099  +5.196  +5.292
+5.568  +5.657  +5.745  +5.831  +5.916  +6.000  +6.083  +6.164
+6.403  +6.481  +6.557  +6.633  +6.708  +6.782  +6.856  +6.928
+7.141  +7.211  +7.280  +7.348  +7.416  +7.483  +7.550  +7.616
+7.810  +7.874  +7.937  +8.000  +8.062  +8.124  +8.185  +8.246
+8.426  +8.485  +8.544  +8.602  +8.660  +8.718  +8.775  +8.832
+9.000  +9.055  +9.110  +9.165  +9.220  +9.274  +9.327  +9.381
+9.539  +9.592  +9.644  +9.695  +9.747  +9.798  +9.849  +9.899
```

```
// file strif-bs.cpp
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;
const char *name = "odmoc.dta";

int main() {

 ifstream ifs(name, ios::in | ios::binary);

 if (ifs != 0) {
         double f;
         cout<< setw(7) << setprecision(4) <<setiosflags(ios::showpoint|ios::showpos);

         while (true) {
               if (ifs.eof()) { // test of file end
           break;
         } // if (ifs.eof())

         ifs.read((char *) &f, sizeof(f));
         cout << f << '\t';
} // while (true)
         ifs.close();
 } // if (ifs != 0)

 return 0;
} // void main()
```

Open a file stream in binary mode can be read sequentially and randomly. It's the same with writing. For the combination of reading and writing over the same set - two indicators:
get indicator - indicates the location in the file where the next read-pointer is;
put indicator - the place, which will next write.
Assistive interface methods described indicators programmer are called:
tellg(), **tellp()** returns the value of the indicators
seekg(), **seekp()** is set.
pos_type type for reading position in the file. Setting position: number of bytes and relative position.

Constants also intended for reference position in the file, **ios ::**

Identificator	description
beg	shift to current position
cur	shift due early
end	shift due to the end

```
ifs.seekg((i  1)  sizeof(f), ios::beg);
```

```
// soubor strif-bn.cpp

#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

const char *name = "odmoc.dta";

int main() {
 ifstream ifs(name, ios::in | ios::binary);
 if (ifs != 0) {
  double f;
  int i;
  cout<< setw(7) << setprecision(4) << setiosflags(ios::showpoint | ios::showpos);
  cout << "Enter number within range <1, 100>: ";
  cin >> i;

  while ((i > 0) && (i < 101)) {
   ifs.seekg((i - 1) * sizeof(f), ios::beg);
   ifs.read((char *) &f, sizeof(f));
   cout << "square root of " << i
        << "\tis " << f << endl;
   cout << "Enter number within range <1, 100>: ";
   cin >> i;
  } // while()
  cout << endl << "End." << endl;
  ifs.close();
 } // if (ifs != 0)

 return 0;
} // void main()
```

```
Enter number within range <1, 100>: 2
square root of +2 is +1.414
Enter number within range <1, 100>: 4
square root of +4 is +2.000
Enter number within range <1, 100>: 29
square root of +29 is +5.385
Enter number within range <1, 100>: 0

End.
```