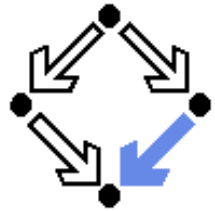
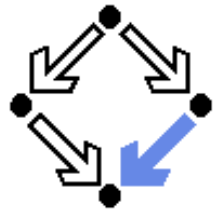
Input and Output

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at





1. Overview

2. Text Input/Output

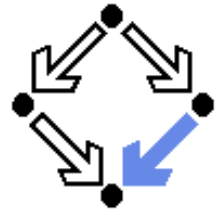
3. Error Handling

4. File Streams

5. Binary Input/Output

6. File System Operations

C Standard Input/Output



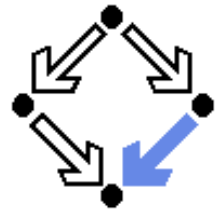
Also in C++, the standard I/O functions of C can be used.

```
#include <cstdio>
using namespace std;

int main() {
  char day[20]; int hour; int min;
  printf("Day:    "); scanf("%19s", day);
  printf("Hour:   "); scanf("%d", &hour);
  printf("Minute: "); scanf("%d", &min);
  printf("%s, %.2d:%.2d\n", day, hour, min);
}
```

```
Day:    Monday
Hour:   6
Minute: 5
Monday, 06:05
```

Low-level and unsafe, C++ provides better alternatives.

C++ I/O Streams



C++ I/O is based on the concept of "streams".

- Stream: an abstraction of an input/output device.
  - A sequence of characters (bytes) flowing from/to a source/sink.
- Stream sources/sinks may be physical devices.

  Console, keyboard, disk file, ...
  - Characters written/read are physically input/output.

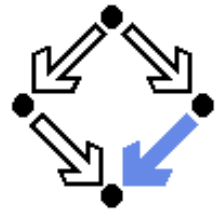
    Class `fstream` (file streams).
- Stream sources/sinks may be abstract devices.

  Strings, buffers, ...
  - Characters written/read are transferred to/from data object.

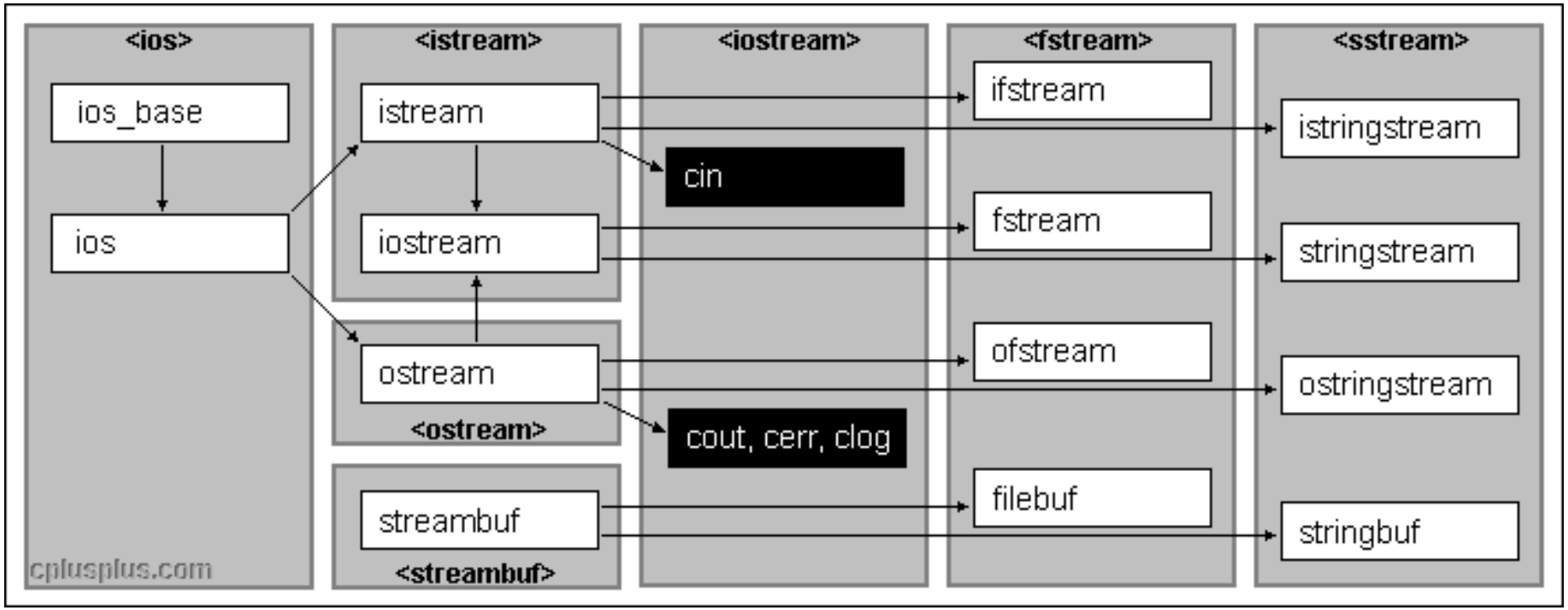
    Class `stringstream` (string streams).

By this abstraction, the same program can operate without significant modification on a multitude of different devices.

C++ Stream Library

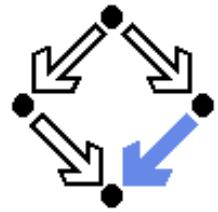


cplusplus.com: “C++ Reference”.

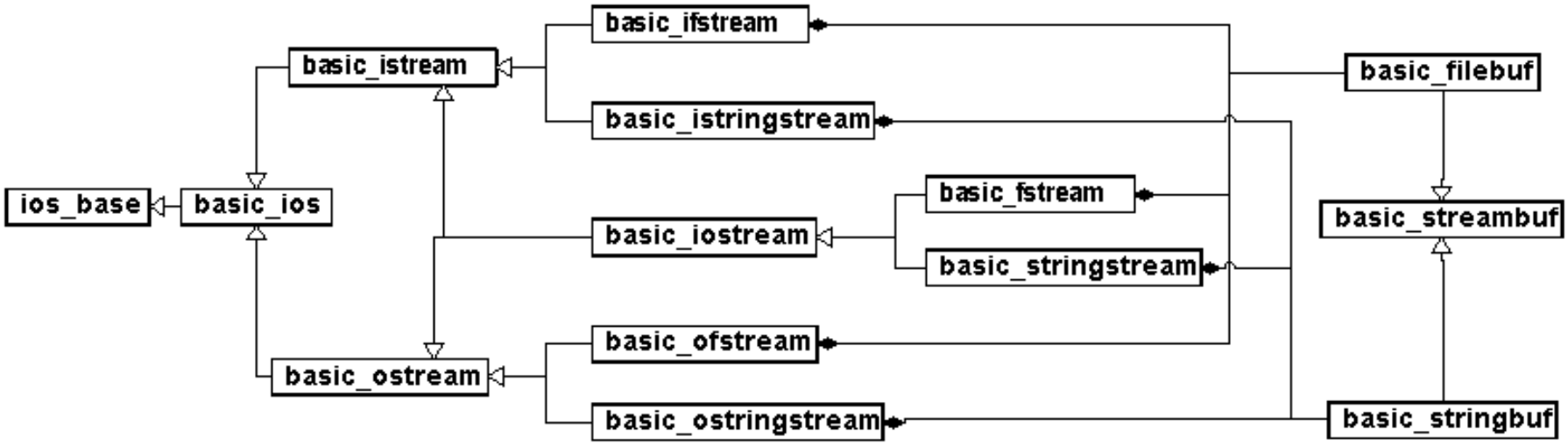


These classes are instantiations of corresponding basic templates.

C++ Stream Template Library



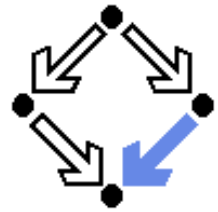
Ray Lischner, “C++ in a Nutshell”.




```
typedef basic_istream<char> istream;
typedef basic_ostream<char> ostream;
typedef basic_iostream<char> iostream;
```

The basic templates are parameterized over the character type.

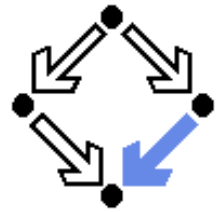
Structure of the Library



- Class `iostream`: reading and writing character sequences.
  - Header `<iostream>`.
  - Inherits from both `istream` and `ostream`.
- Class `fstream`: reading and writing from/to files.
  - Header `<fstream>`.
  - Inherits from `iostream`.
  - Classes `ifstream`/`ofstream` with restricted interfaces.
- Class `stringstream`: reading and writing from/to strings
  - Header `<sstream>`.
  - Inherits from `iostream`.
  - Classes `istringstream`/`ostringstream` with restricted interfaces.
- Objects `cin` and `cout`/`cerr`/`clog`.
  - Standard streams for input and for output/errors/logging.

Wide character counterparts are prefixed by "w" (`wiostream`, `wcin`, ...).

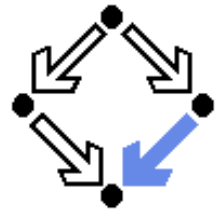
I/O Flavors of the Library



Is the character sequence interpreted by the I/O stream or not?

- Text input/output ("formatted I/O")
  - Character sequence is in some way interpreted.
    - E.g. for conversion of character sequence from/to another datatype.
  - Conversion controlled by the stream's locale (formatting preferences).
    - Adjusted by e.g. applying stream manipulators.
  - I/O operators << and >>.
- Binary input/output ("unformatted I/O")
  - Sequence of characters is transferred without interpretation.
    - Characters are considered as plain bytes.
  - Input functions e.g. `get` and `read`.
  - Output functions e.g. `put` and `write`.

We will investigate both flavors.



1. Overview

2. Text Input/Output

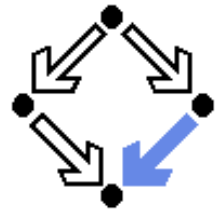
3. Error Handling

4. File Streams

5. Binary Input/Output

6. File System Operations

I/O Operators



For formatted I/O, the shift operators are overloaded as member functions of the corresponding stream classes.

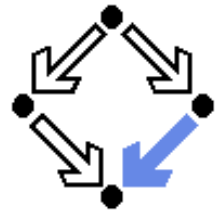
- Writing: `operator<<`
  - *out* `<<` *val*: write value *val* to output stream *out*.
- Reading: `operator>>`
  - *in* `>>` *var*: read value from input stream *in* into variable *var*.
- Typical: chaining of multiple operations on a stream.
  - Write multiple values to an output stream:

    ```
    out << val1 << val2 << ...;
    ```
  - Read multiple values from an input stream:

    ```
    in >> var1 >> var2 >> ...;
    ```

The operators can be also overloaded for user-defined datatypes.

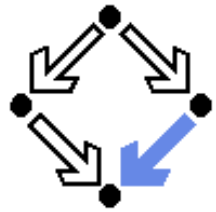
Making a Class Writable to a Stream



D. Ryan Stephens et al, "C++ Cookbook".

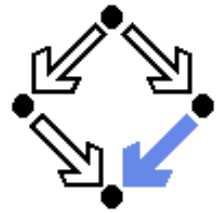
```
#include <iostream>
#include <string>
using namespace std;
...

int main( ) {
   Employee emp;
   string first = "William";
   string last = "Shatner";
   emp.setFirstName(first);
   emp.setLastName(last);

   Employer empr;
   string name = "Enterprise";
   empr.setName(name);
   emp.setEmployer(empr);

   cout << emp;  // Write to the stream
}
```

Making a Class Writable to a Stream



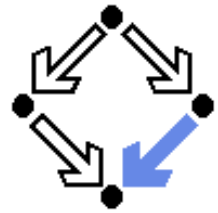
```
class Employer {
   friend ostream& operator<<                  // This has to be a friend
     (ostream& out, const Employer& empr); // so it can access non-
public:                                        // public members
   Employer( ) {}
  ~Employer( ) {}
   void setName(const string& name) {name_ = name;}
private:
   string name_;
};

// Allow us to send Employer objects to an ostream...
ostream& operator<<(ostream& out, const Employer& empr) {
   out << empr.name_ << endl;
   return(out);
}
```

Making a Class Writable to a Stream



```
class Employee {
   friend ostream& operator<< (ostream& out, const Employee& obj);
public:
   Employee( ) : empr_(NULL) {}
  ~Employee( ) {if (empr_) delete empr_;}
   void setFirstName(const string& name) {firstName_ = name;}
   void setLastName(const string& name) {lastName_ = name;}
   void setEmployer(Employer& empr) {empr_ = &empr;}
   const Employer* getEmployer( ) const {return(empr_);}
private:
   string firstName_;
   string lastName_;
   Employer* empr_;
};

// Allow us to send Employee objects to an ostream...
ostream& operator<<(ostream& out, const Employee& emp) {
   out << emp.firstName_ << endl;
   out << emp.lastName_ << endl;
   if (emp.empr_) out << *emp.empr_ << endl;
   return(out);
}
```

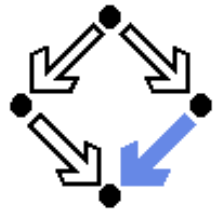
Making a Class Readable from a Stream



D. Ryan Stephens et al, "C++ Cookbook" (modified).

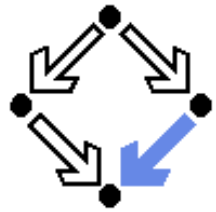
```
#include <iostream>
#include <istream>
#include <string>

using namespace std;

...

int main( ) {
   Employee emp; Employee emp2;

   string first = "William"; string last = "Shatner";
   emp.setFirstName(first); emp.setLastName(last);

   cout << emp;
   cin >> emp2;
   cout << emp2;
}
```

Making a Class Readable from a Stream



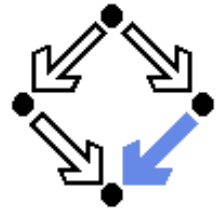
```
class Employee {
   friend ostream& operator<<                // These have to be friends
      (ostream& out, const Employee& emp); // so they can access
   friend istream& operator>>                // nonpublic members
      (istream& in, Employee& emp);

public:
   Employee( ) {}
  ~Employee( ) {}

   void setFirstName(const string& name) {firstName_ = name;}
   void setLastName(const string& name) {lastName_ = name;}

private:
   string firstName_;
   string lastName_;
};
```

Making a Class Readable from a Stream



```
// Send an Employee object to an ostream...
ostream& operator<<(ostream& out, const Employee& emp) {

   out << emp.firstName_ << endl;
   out << emp.lastName_ << endl;

   return(out);
}

// Read an Employee object from a stream
istream& operator>>(istream& in, Employee& emp) {

   in >> emp.firstName_;
   in >> emp.lastName_;

   return(in);
}
```

I/O Manipulators

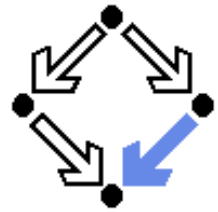


Manipulators can be inserted into a stream to influence its behavior.

```
out << val1 << endl << val2; // insert new line
in >> var1 >> ws >> var2; // eat white space
```

- <iostream>: input/output manipulators.
  - endl: insert new line and flush.
  - ends: insert null character and flush.
  - flush: flush stream buffer.
  - ws: eat white space.
- <ios>: formatting flags manipulators.
  - dec/hex/oct: use decimal/hexadecimal/octal number base.
  - skipws/noskipws: (do not) skip white space.
  - ...
- <iomanip>: parameterized manipulators.
  - setprecision(n): set output precision to n digits.
  - setw(n): set field width of output to n characters.
  - ...

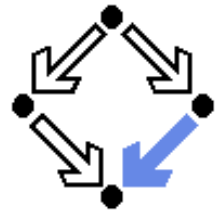
I/O Manipulators



cplusplus.com: “C++ Reference”.

```
Independent flags (switch on):
boolalpha       Alphanumerical bool values (manipulator function)
showbase        Show numerical base prefixes (manipulator function)
showpoint       Show decimal point (manipulator function)
showpos         Show positive signs (manipulator function)
skipws          Skip whitespaces (manipulator function)
unitbuf         Flush buffer after insertions (manipulator function)
uppercase       Generate upper-case letters (manipulator function)

Independent flags (switch off):
noboolalpha     No alphanumerical bool values (manipulator function)
noshowbase      Do not show numerical base prefixes (manipulator function)
noshowpoint     Do not show decimal point (manipulator function)
noshowpos       Do not show positive signs (manipulator function)
noskipws        Do not skip whitespaces (manipulator function)
nounitbuf       Do not force flushes after insertions (manipulator function)
nouppercase     Do not generate upper case letters (manipulator function)
```

Example

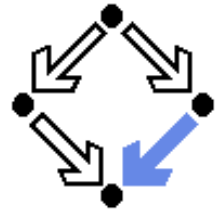


cplusplus.com: “C++ Reference”.

```
#include <iostream>
using namespace std;

int main () {
  char a[10], b[10];

  cin >> noskipws;
  cin >> a >> ws >> b;
  cout << a << "," << b << endl;

  return 0;
}
```

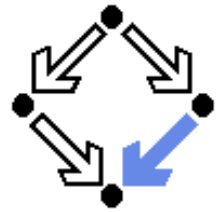
```
Input:
one
   two

Output:
one,two
```

I/O Manipulators



```
Numerical base format flags ("basefield" flags):
dec     Use decimal base (manipulator function)
hex     Use hexadecimal base (manipulator function)
oct     Use octal base (manipulator function)

Floating-point format flags ("floatfield" flags):
fixed           Use fixed-point notation (manipulator function)
scientific      Use scientific notation (manipulator function)

Adustment format flags ("adjustfield" flags):
internal Adjust field by inserting characters at an internal position (m.f.)
left     Adjust output to the left (manipulator function)
right    Adjust output to the right (manipulator function)

Input manipulators
ws      Extract whitespaces (manipulator function)

Output manipulators
endl    Insert newline and flush (manipulator function)
ends    Insert null character (manipulator function)
flush   Flush stream buffer (manipulator function)
```

Example

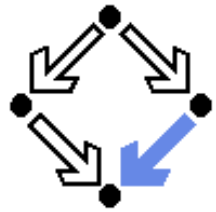


cplusplus.com: “C++ Reference”.

```
#include <iostream>
using namespace std;

int main () {
  int n;
  n=70;
  cout << dec << n << endl;
  cout << hex << n << endl;
  cout << oct << n << endl;
  return 0;
}
```

```
70
46
106
```

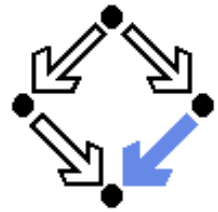
I/O Manipulators



cplusplus.com: “C++ Reference”.

```
Parameterized manipulators

These functions take parameters when used as manipulators. They require the
explicit inclusion of the header file <iomanip>.

setiosflags     Set format flags (manipulator function)
resetiosflags   Reset format flags (manipulator function)
setbase         Set basefield flag (manipulator function)
setfill         Set fill character (manipulator function)
setprecision    Set decimal precision (manipulator function)
setw            Set field width (manipulator function)
```

Example

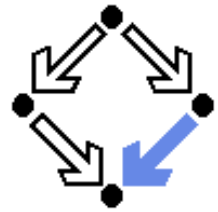


cplusplus.com: “C++ Reference”.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
  double f = 3.14159;
  cout << setprecision (5) << f << endl; // up to 5 digits (both sides of ".")
  cout << setprecision (9) << f << endl; // up to 9 digits (both sides of ".")
  cout << fixed;
  cout << setprecision (5) << f << endl; // exactly 5 digits to the right of "."
  cout << setprecision (9) << f << endl; // exactly 9 digits to the right of "."
  return 0;
}
```

```
3.1416
3.14159
3.14159
3.141590000
```

Example

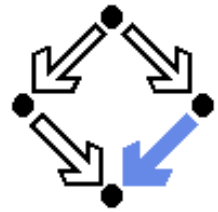


cplusplus.com: “C++ Reference”.

```
#include <iostream>
using namespace std;

int main () {
  int n;
  n=-77;
  cout << setw(6);
  cout << internal << n << endl;
  cout << left << n << endl;
  cout << right << n << endl;
  return 0;
}
```

```
-   77
-77
   -77
```

What is a Manipulator?

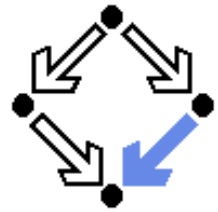


cplusplus.com: “C++ Reference”.

```
ios_base& skipws ( ios_base& str );

Skip whitespaces
Sets the skipws format flag for the str stream.

When the skipws format flag is set, as many whitespace characters as necessary
are read and discarded from the stream until a non-whitespace character is
found before every extraction operation. Tab spaces, carriage returns and
blank spaces are all considered whitespaces.

This flag can be unset with the noskipws manipulator, forcing extraction
operations to consider leading whitepaces as part of the content to be
extracted.

The skipws flag is set in standard streams on initialization.
```

A manipulator is just a function on a stream (and can be directly called).

I/O Operations and Manipulators

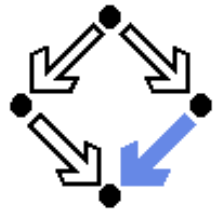


cplusplus.com: “C++ Reference”.

```
istream::operator>>                                    public member function
istream& operator>> (ios_base& ( *pf )(ios_base&));

ostream::operator<<                                    public member function
ostream& operator<< (ios_base& ( *pf )(ios_base&));

This member functions has a pointer to a function as parameter. It is
designed to be used with a manipulator function.  Manipulator functions are
functions specifically designed to be easily used with this operator.
```

The I/O operators are overloaded to accept manipulators and call them on the respective streams.

Streams and Manipulators

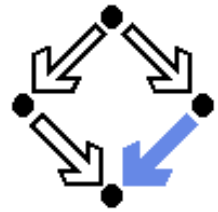


cplusplus.com: “C++ Reference”.

```
ios_base                                                        class

The ios_base class is designed to be a base class for all of the hierarchy of
stream classes, describing the most basic part of a stream, which is common
to all stream objects. Therefore, it is not designed to be directly
instantiated into objects.

The ios_base class is in charge of maintaining internally the following
information of a stream:

Formatting information
  ...
State information
  ...
Other
  ...
```

Manipulators modify the formatting/state information of a stream objects.

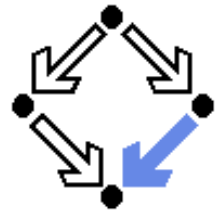
Stream Formatting Information



cplusplus.com: “C++ Reference”.

```
Formatting information

* format flags: a set of internal indicators describing how certain
  input/output operations shall be interpreted or generated. The state of these
  indicators can be obtained or modified by calling the members flags, setf
  and unsetf.

* field width: describes the width of the next element to be output. This
  value can be obtained/modified by calling the member function width.

* display precision: describes the decimal precision to be used to output
  floating-point values. This value can be obtained/modified by calling member pre

* locale object: describes the localization properties to be considered when
  formatting i/o operations. The locale object used can be obtained calling
  member getloc and modified using imbue.
```

Format Flags



ios_base::setf public member function

```
fmtflags setf ( fmtflags fmtfl );
fmtflags setf ( fmtflags fmtfl, fmtflags mask );
```

The first syntax sets the stream's format flags whose bits are set in fmtfl leaving unchanged the rest.

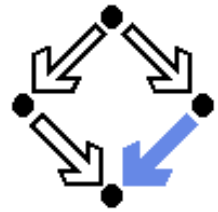
The second syntax sets the flags whose bits are set in both fmtfl and mask, and clears the format flags whose bits are set in mask but not in fmtf. ...

The first syntax of setf is generally used to set unary format flags: boolalpha, showbase, showpoint, showpos, skipws, unitbuf and uppercase, which can also be unset directly with unsetf.

On the other hand, the second syntax is generally used to set a value for one of the selective flags, using one of the field bitmasks as the mask argument:

fmtfl	mask
left, right or internal	adjustfield
dec, oct or hex	basefield
scientific or fixed	floatfield

Example



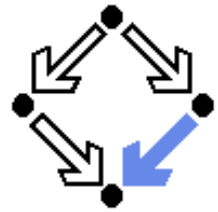
cplusplus.com: “C++ Reference”.

```
#include <iostream>
using namespace std;

int main () {
  cout.setf ( ios::hex, ios::basefield );        // set hex as the basefield
  cout.setf ( ios::showbase );                   // activate showbase
  cout << 100 << endl;
  cout.setf ( 0, ios::showbase );                // deactivate showbase
  cout << 100 << endl;
  return 0;
}
```

```
0x64
64
```

Set flags directly on stream object.

Example

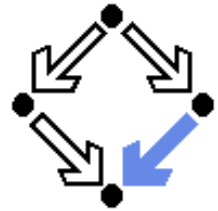


cplusplus.com: “C++ Reference”.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
  cout << hex << setiosflags (ios_base::showbase | ios_base::uppercase);
  cout << 77 << endl;
  return 0;
}

0X4D
```

Use stream manipulator for setting flags.

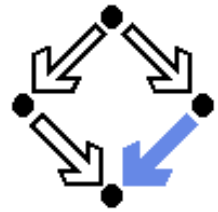
Example



D. Ryan Stephens et al: “C++ Cookbook”.

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

int main( ) {
   ios_base::fmtflags flags = cout.flags( ); // save format flags
   string first, last, citystate;
   int width = 20;

   first = "Richard"; last  = "Stevens"; citystate = "Tucson, AZ";

   cout << left                          // Left-justify in each field
        << setw(width) << first          // Then, repeatedly set the width
        << setw(width) << last           // and write some data
        << setw(width) << citystate << endl;

   cout.flags(flags);                          // restore format flags
}
```

String Streams

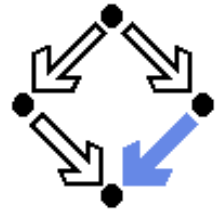


cplusplus.com: “C++ Reference”

```
stringstream                                          class (header <sstream>)
Input/output string stream class

stringstream provides an interface to manipulate strings as if they were
input/output streams.

The objects of this class maintain internally a pointer to a stringbuf object
that can be obtained/modified by calling member rdbuf. This streambuf-derived
object controls a sequence of characters (string) that can be
obtained/modified by calling member str.
```

We can use I/O operations to construct strings and parse them.

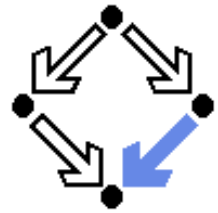
Example



cplusplus.com: “C++ Reference” (modified).

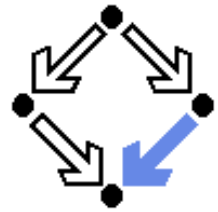
```
#include <iostream>
#include <sstream>
using namespace std;

int main () {
  int val;
  stringstream ss;

  ss << "120 42 377 6 5 2000";

  for (int n=0; n<6; n++)
  {
    ss >> val;
    cout << val*2 << endl;
  }

  return 0;
}
```

Example: Writing a Rational Number



Ray Lischner, "C++ in a Nutshell".

```
// Print a rational number as two integers separated by a slash.
// Use a string stream so the two numbers are written without padding,
// and the overall formatted string is then padded to the desired width.
template<typename T, typename charT, typename traits>
::std::basic_ostream<charT, traits>&
operator<<(::std::basic_ostream<charT, traits>& out,
           const rational<T>& r)
{
  // Use the same flags, locale, etc. to write the
  // numerator and denominator to a string stream.
  ::std::basic_ostringstream<charT, traits> s;
  s.flags(out.flags());
  s.imbue(out.getloc());
  s.precision(out.precision());
  s << r.numerator() << '/' << r.denominator();
  // Write the string to out, padding the entire fraction as needed.
  out << s.str();
  return out;
}
```



1. Overview

2. Text Input/Output

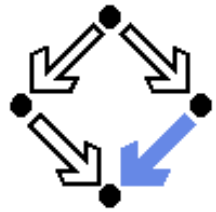
3. Error Handling

4. File Streams

5. Binary Input/Output

6. File System Operations

Stream State Information

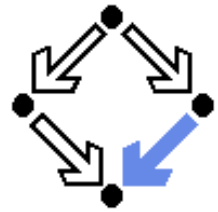


cplusplus.com: “C++ Reference”.

```
State information

* error state: internal indicator reflecting the integrity and current error
  state of the stream. The current object may be obtained by calling
  ios::rdstate and can be modified by calling ios::clear and
  ios::setstate. Individual values may be obtained by calling ios::good,
  ios::eof, ios::fail and ios::bad.

* exception mask: internal exception status indicator. Its value can be
  retrieved/modified by calling member ios::exceptions.
```

It is important to check the success of an I/O operation.

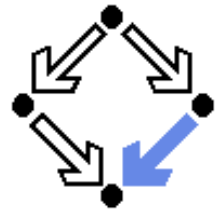
Error State



cplusplus.com: “C++ Reference”.

```
ios_base::iostate                                          public member type
Bitmask type to represent stream error state flags.

All stream objects keep internally information on the state of the object that
can be retrieved as an element of this type by calling member function
ios::rdstate or set by calling ios::setstate.

The values passed and retrieved by these functions can be any valid combination
(using the boolean or operator, "|") of the following member constants:

flag value   indicates
eofbit       End-Of-File reached while performing an extracting operation
               on an input stream.
failbit      The last input operation failed because of an error related
               to the internal logic of the operation itself.
badbit       Error due to the failure of an I/O operation on the stream buffer.
goodbit      No error. Represents the absence of all the above (value zero).
```

Reading and Writing State Information

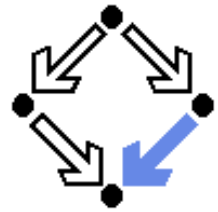


cplusplus.com: “C++ Reference”.

```
ios::...                                                public member functions

iostate rdstate ( ) const;
   Returns the current internal error state flags of the stream.
void setstate ( iostate state );
   Modifies the current error state by combining the error state flags passed
   as argument with those currently set for the stream.
void clear ( iostate state = goodbit );
   Sets a new value for the error control state. All the bits in the control
   state are replaced by the new ones.
```

Better use the higher-level alternatives.

Checking State Information

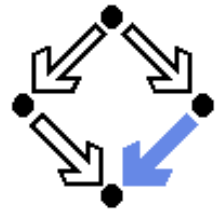


cplusplus.com: “C++ Reference”.

```
ios::...                                                public member functions

bool good ( ) const;
   The function returns true if none of the stream's error flags are set.
bool eof ( ) const;
   The function returns true if the eofbit stream's error flag has been set.
bool fail ( ) const;
   The function returns true if either the failbit or the badbit is set .
bool bad ( ) const;
   The function returns true if the badbit stream's error flag is set.

// Automatic conversion of error status to bool
// (actual implementation is a bit different)
operator bool() const { !fail(); } // conversion operator
```

Higher-level ways of checking the status of a stream.

Checking State Information

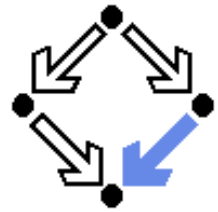


cplusplus.com: “C++ Reference”.

```
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ifstream is;
  is.open ("test.txt");
  if ((is.rdstate() & ifstream::failbit ) != 0) // alternative:  if (is.fail())
    cerr << "Error opening 'test.txt'\n";       // even simpler: if (!is)
  return 0;
}
```

Especially important for file input/output (see later).

Reading a Sequence of Values



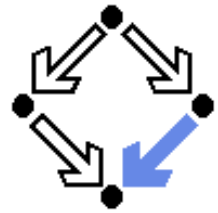
A common pattern for reading a sequence of values.

```
while (true)
{
  int input;
  cin >> input;
  if (!cin) break;
  ... // process input
}
```

- After last item has been read, `eofbit` is set on input stream.
  - `cin.eof()` yields "true".
- On next attempt on reading item, `failbit` is set on input stream.
  - `cin.fail()` yields "true".
  - `!cin` yields "false".

Check status of input stream after every read operation!

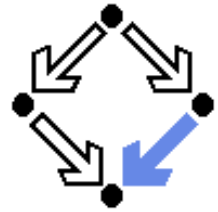
Example: Reading a Rational Number



Ray Lischner, “C++ in a Nutshell”.

```
// Read a rational number, e.g. "2/3" or "-14/19".
template<typename T, typename charT, typename traits>
::std::basic_istream<charT, traits>&
operator>>(::std::basic_istream<charT, traits>& in, rational<T>& r)
{
  rational<T>::numerator_type n;
  rational<T>::denominator_type d;
  char c;

  if (! (in >> n)) return in;
  if (! (in >> c)) return in;
  if (c != '/') {
     in.setstate(::std::ios_base::failbit);
     return in;
  }
  if (! (in >> d)) return in;
  r.set(n, d);
  return in;
}
```

I/O Errors and Exceptions



Rather than setting error bits, streams may also raise exceptions.

- Every stream has an exception mask.
  - Set of failure conditions for which exceptions are thrown.
  - By default, the exception mask is empty.
- Public member function `ios::exceptions`
  - Overloaded for setting/getting a stream's exception mask.
- *s*`.exceptions(`*mask*`);`
  - Sets exception mask of *s* to *mask*.

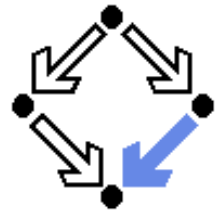
```
cin.exceptions(ios_base::badbit | ios_base::failbit);
```

- `iostate mask =` *s*`.exceptions();`
  - Gets exception mask of *s* to *mask*.

```
iostate mask = cin.exceptions();
cin.exceptions(mask | ios_base::badbit);
```

May considerably simplify the handling of I/O errors.

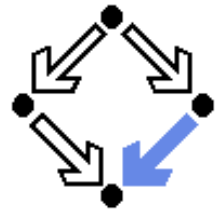
Example: Handling I/O Errors



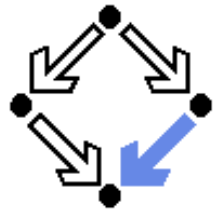
Ray Lischner, "C++ in a Nutshell" (modified).

```
#include <exception>
#include <iostream>
#include <string>

using namespace std;

int main() {
  try {
    cin.exceptions(ios_base::badbit);  // badbit (but not failbit)
    cout.exceptions(ios_base::badbit); //   triggers exception
    while (true) {
      string word;
      cin >> word;
      if (!cin) break;                 // failbit is checked
      cout << word << endl;
    }
  }
  ...
}
```

Example: Handling I/O Errors (Contd)



```
#include <exception>
#include <iostream>
#include <string>

using namespace std;

int main() {
  try {
    ...
  } catch(ios_base::failure& ex) {
    cerr << "I/O error: " << ex.what() << '\n';
    return 1;
  } catch(exception& ex) {
    cerr << "Fatal error: " << ex.what() << '\n';
    return 2;
  } catch(...) {
    cerr << "Total disaster.\n";
    return 3;
  }
}
```



1. Overview

2. Text Input/Output

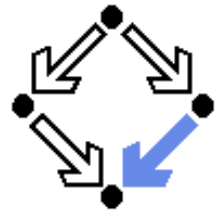
3. Error Handling

4. File Streams

5. Binary Input/Output

6. File System Operations

File Streams

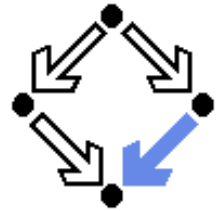


cplusplus.com: “C++ Reference”.

```
fstream
Input/output file stream class

fstream provides an interface to read and write data from files as
input/output streams.

The objects of this class maintain internally a pointer to a filebuf object
that can be obtained by calling member rdbuf.

The file to be associated with the stream can be specified either as a
parameter in the constructor or by calling member open.

After all necessary operations on a file have been performed, it can be closed
(or disassociated) by calling member close. Once closed, the same file stream
object may be used to open another file.

The member function is_open can be used to determine whether the stream object
is currently associated with a file.
```

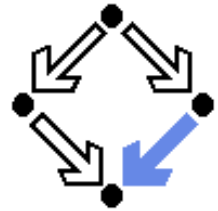
File streams represent the contents of files (not the file system data).

Examples



```
#include <fstream>
#include <iostream>
using namespace std;

// read from file in.txt, let constructor open file
int main () {
  fstream in ("in.txt", fstream::in);
  if (!in) return;
  // input operations (in >> var)
  in.close();
  return 0;
}

// append to file out.txt, call open() for opening file
int main () {
  fstream out;
  out.open("out.txt", fstream::out | fstream::app);
  if (!out) return;
  // output operations (out << value)
  out.close();
  return 0;
}
```

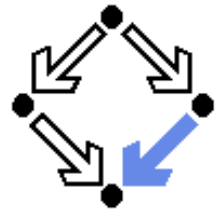
Constructing a File Stream



cplusplus.com: “C++ Reference”.

```
fstream::fstream  constructor member

fstream ( );
explicit fstream ( const char * filename,
                   ios_base::openmode mode = ios_base::in | ios_base::out );

Construct object and optionally open file

Constructs an object of the fstream class. This implies the initialization of
the associated filebuf object and the call to the constructor of its base
class with the filebuf object as parameter.

Additionally, when the second constructor version is used, the stream is
associated with a physical file as if a call to the member function open with
the same parameters was made.

If the constructor is not successful in opening the file, the object is still
created although no file is associated to the stream buffer and the stream's
failbit is set (which can be checked with inherited member fail).
```

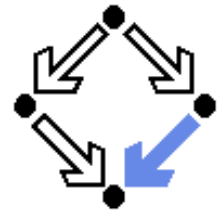
Opening a File Stream



cplusplus.com: “C++ Reference”.

```
fstream::open  public member function

void open ( const char * filename,
            ios_base::openmode mode = ios_base::in | ios_base::out );

Open file

Opens a file whose name is s, associating its content with the stream object
to perform input/output operations on it. The operations allowed and some
operating details depend on parameter mode.

The function effectively calls rdbuf()->open(filename,mode).

If the object already has a file associated (open), the function fails.

On failure, the failbit flag is set (which can be checked with member fail),
and depending on the value set with exceptions an exception may be thrown.
```

File Modes



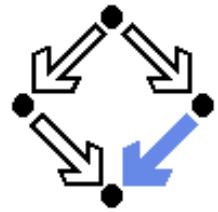
ios_base::openmode public member type

Bitmask type to represent stream opening mode flags.A value of this type can be any valid combination of the following member constants:

flag value	opening mode
app	(append) Set the stream's position indicator to the end of the stream before each output operation.
ate	(at end) Set the stream's position indicator to the end of the stream on opening.
binary	(binary) Consider stream as binary rather than text.
in	(input) Allow input operations on the stream.
out	(output) Allow output operations on the stream.
trunc	(truncate) Any current content is discarded, assuming a length of zero on opening.

These constants are defined in the ios_base class as public members. Therefore, they can be refered to either directly by their name as ios_base members (like ios_base::in) or by using any of their inherited classes or instantiated objects, like for example ios::ate or cout.out.

Closing a File Stream

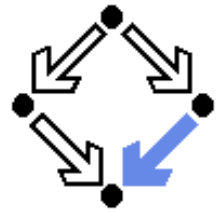


cplusplus.com: “C++ Reference”.

```
fstream::close  public member function

void close ( );

Close file

Closes the file currently associated with the object, disassociating it from
the stream. Any pending output sequence is written to the physical file.

The function effectively calls rdbuf()->close().

The function fails if no file is currently open (associated) with this object.

On failure, the failbit internal state flag is set (which can be checked with
member fail), and depending on the value set with exception an exception may
be thrown.
```

Before a file stream is closed, it is not guaranteed that data are on disk.

Flushing a Stream

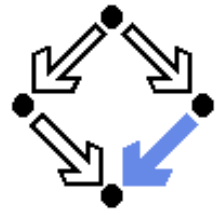


cplusplus.com: “C++ Reference”.

```
ostream::flush                                          public member function

ostream& flush ( );

Flush output stream buffer

Synchronizes the buffer associated with the stream to its controlled output
sequence. This effectively means that all unwritten characters in the buffer
are written to its controlled output sequence as soon as possible ("flushed").

The function only has meaning for buffered streams, in which case it
effectively calls the pubsync member of the streambuf object
(rdbuf()->pubsync()) associated to the stream.

A manipulator exists with the same name and behavior (see flush manipulator).
```

Typically applied in long running programs to make sure that output generated so far is on disk (not all data are lost when computer fails).

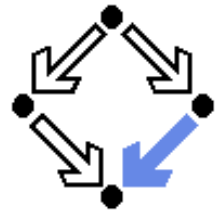
Example



cplusplus.com: “C++ Reference”.

```
#include <fstream>
using namespace std;

int main () {
  fstream outfile ("test.txt", ios::out);

  for (int n=0; n<100; n++) {
    outfile << n << " "; // alternative: outfile << n << " " << flush;
    outfile.flush();
  }

  outfile.close();

  return 0;
}
```

Input/Output File Streams

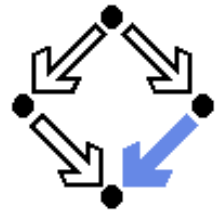


cplusplus.com: “C++ Reference”.

```
ifstream                                                    class

Input file stream
ifstream provides an interface to read data from files as input streams.

ifstream( );
explicit ifstream (const char* filename, ios_base::openmode mode = ios_base::in);
void open(const char* filename, ios_base::openmode mode = ios_base::in);

ofstream                                                    class

Output file stream
ofstream provides an interface to write data to files as output streams.

ofstream( );
explicit ofstream(const char* filename, ios_base::openmode mode = ios_base::out);
void open(const char* filename, ios_base::openmode mode = ios_base::out);
```

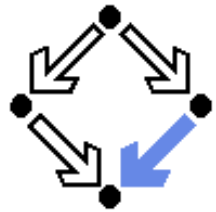
Specialized to support only the corresponding read/write operations.

Copying a File of Integers

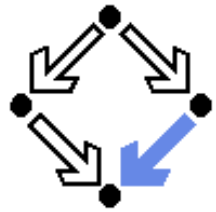


```
#include <iostream>
#include <fstream>
using namespace std;

int intFileCopy(char* inName, char* outName) throw(string);

int main(int argc, char* argv[]) {
  if (argc != 3) {
    cout << "Usage: intCopy <infile> <outfile>" << endl;
    return -1;
  }
  try {
    int n = intFileCopy(argv[1], argv[2]);
    cout << n << " values copied" << endl;
  }
  catch(string& message) {
    cerr << "Error: " << message << endl;
  }
  return 0;
}
```

Copying a File of Integers (Contd)



```
int intFileCopy(char* inName, char* outName) throw(string) {
  ifstream in(inName);
  if (!in) throw string("could not open input file");

  ofstream out(outName);
  if (!out) { in.close(); throw string("could not open output file"); }

  int i = 0;
  while (true) {
    int input;
    in >> input;
    if (!in) break;
    out << input << endl;
    i = i+1;
  }

  in.close();
  out.close();

  return i;
}
```



1. Overview

2. Text Input/Output

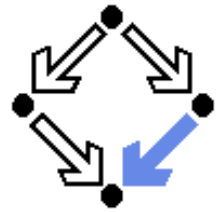
3. Error Handling

4. File Streams

5. **Binary Input/Output**

6. File System Operations

Binary Input: Byte Sequences

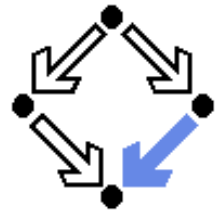


cplusplus.com: “C++ Reference”.

```
istream::get                                          public member function
istream& get ( char* s, streamsize n );

Extracts characters from the stream and stores them as a c-string into the
array beginning at s. Characters are extracted until either (n - 1)
characters have been extracted or the delimiting character '\n' is found. ...
If the delimiting character is found, it is not extracted from the input
sequence and remains as the next character to be extracted. Use getline if
you want this character to be extracted (and discarded).  The ending null
character that signals the end of a c-string is automatically appended at the
end of the content stored in s.

istream::get                                          public member function
istream& get ( char* s, streamsize n, char delim );

Same as above, except that delimiting character is specified in delim.
```

Reading delimited byte sequences into buffers (preserving the delimiter).

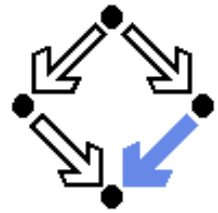
Binary Input: Byte Sequences



```
istream::getline                                    public member function

istream& getline (char* s, streamsize n );
istream& getline (char* s, streamsize n, char delim );

Extracts characters from the input sequence and stores them as a c-string into
the array beginning at s.

Characters are extracted until either (n - 1) characters have been extracted
or the delimiting character is found (which is delim if this parameter is
specified, or '\n' otherwise). The extraction also stops if the end of file
is reached in the input sequence or if an error occurs during the input operation.

If the delimiter is found, it is extracted and discarded, i.e. it is not
stored and the next input operation will begin after it. If you don't want
this character to be extracted, you can use member get instead.

The ending null character that signals the end of a c-string is automatically
appended to s after the data extracted. The number of characters read by this
function can be obtained by calling to the member function gcount.
```

Reading delimited byte sequences as C strings (discarding the delimiter).

Binary Input: Byte Sequences

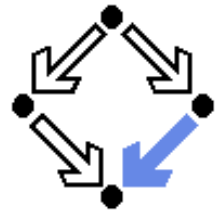


cplusplus.com: “C++ Reference”.

```
getline                                          function (header <string>)

istream& getline ( istream& is, string& str, char delim );
istream& getline ( istream& is, string& str );

Extracts characters from is and stores them into str until a delimitation
character is found.

The delimiter character is delim for the first function version, and '\n'
(newline character) for the second. The extraction also stops if the end of
file is reached in is or if some other error occurs during the input
operation.

If the delimiter is found, it is extracted and discarded, i.e. it is not
stored and the next input operation will begin after it.

Notice that unlike the c-string versions of istream::getline, these string
versions are implemented as global functions instead of members of the class.
```

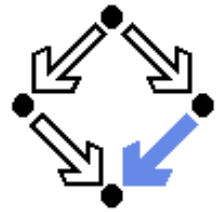
Reading delimited byte sequences as C++ strings (discarding delimiter).

Example: Copying a File Line-Wise



```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

const static int BUF_SIZE = 4096;
int main(int argc, char** argv) {
   ifstream in(argv[1]);
   ofstream out(argv[2]);
   if (!in || !out) return -1;

   while(true) {
      string s;
      getline(in, s);
      if (!in) break;
      out << s << endl;
   }

   in.close( ); out.close( );
   return 0;
}
```

Binary Input/Output: Bytes

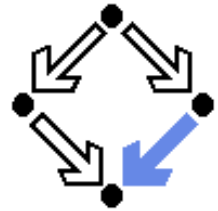


cplusplus.com: “C++ Reference”.

```
istream::get                                              public member function
int get();

Extracts a character from the stream and returns its value (casted to int).

istream::get                                              public member function
istream& get ( char& c );

Extracts a character from the stream and stores it in c.

ostream::put                                              public member function
ostream& put ( char c );

Writes the character c to the output buffer at the current put position and
increases the put pointer to point to the next character.
```

Reading/writing individual bytes from/to input/output stream.

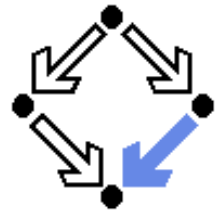
Example: Copying a File Byte-Wise



cplusplus.com: “C++ Reference” (modified)

```
#include <iostream>
#include <fstream>
using namespace std;

int main(int argc, char** argv) {
  ifstream in(argv[1],  ios_base::in | ios_base::binary);
  ofstream out(argv[2], ios_base::out | ios_base::binary);
  if (!in || !out) return -1;

  while(true) {
    char ch = in.get();
    if (!in) break;
    out.put(ch);
  }

  in.close(); out.close();
  return 0;
}
```

Binary Input: Blocks

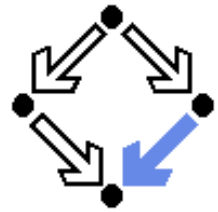


cplusplus.com: “C++ Reference”.

```
istream::read                                          public member function
istream& read ( char* s, streamsize n );

Reads a block of data of n characters and stores it in the array pointed by s.

If the End-of-File is reached before n characters have been read, the array
will contain all the elements read until it, and the failbit and eofbit will
be set (which can be checked with members fail and eof respectively).

Notice that this is an unformatted input function and what is extracted is not
stored as a c-string format, therefore no ending null-character is appended
at the end of the character sequence.

istream::gcount                                        public member function
streamsize  gcount ( ) const;

Returns the number of characters extracted by the last unformatted input
operation performed on the object.
```

Reading byte blocks of fixed size from the input stream.

Binary Output: Blocks

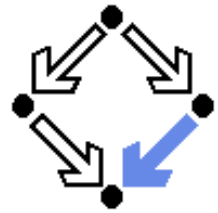


cplusplus.com: “C++ Reference”.

```
ostream::write                                          public member function
ostream& write ( const char* s , streamsize n );

Writes the block of data pointed by s, with a size of n characters, into the
output buffer. The characters are written sequentially until n have been written.

This is an unformatted output function and what is written is not necessarily
a c-string, therefore any null-character found in the array s is copied to
the destination and does not end the writing process.
```

Writing byte blocks of fixed size to the output stream.

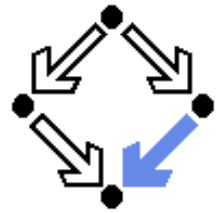
Example: Copying a File Block-Wise



D. Ryan Stephens et al, "C++ Cookbook" (modified).

```
#include <iostream>
#include <fstream>
using namespace std;

const static int BUF_SIZE = 4096;
int main(int argc, char** argv) {
   ifstream in(argv[1],  ios_base::in | ios_base::binary);
   ofstream out(argv[2], ios_base::out | ios_base::binary);
   if (!in || !out) return -1;

   while(true) {
      char buf[BUF_SIZE];
      in.read(&buf[0], BUF_SIZE);       if (!in) return -1;
      if (in.gcount() == 0) break;
      out.write(&buf[0], in.gcount( )); if (!out) return -1;
   }

   in.close( ); out.close( );
   return 0;
}
```

Get Pointer Manipulation

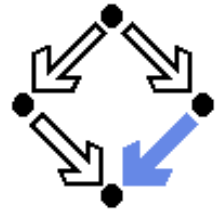


cplusplus.com: “C++ Reference”.

```
The get pointer determines the next location in the input sequence to be read
by the next input operation.

istream::*                                                public member function

streampos tellg ( );

Returns the absolute position of the get pointer.

istream& seekg ( streampos pos );
istream& seekg ( streamoff off, ios_base::seekdir dir );

Set position of the get pointer.

dir         offset is relative to...
ios_base::beg beginning of the stream buffer
ios_base::cur current position in the stream buffer
ios_base::end end of the stream buffer
```

Random read access to the contents of a file stream is possible.

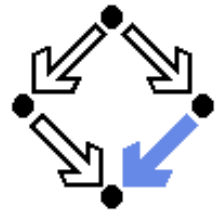
Example



cplusplus.com: “C++ Reference” (modified)

```
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ifstream is("test.txt", ios::binary);

  is.seekg (0, ios::end);
  int length = is.tellg();

  is.seekg (0, ios::beg);
  char* buffer = new char [length];
  is.read (buffer,length);
  is.close();

  cout.write (buffer,length);
  delete[] buffer;
  return 0;
}
```

Put Pointer Manipulation

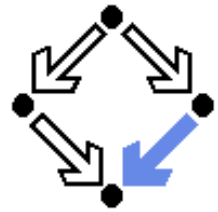


cplusplus.com: “C++ Reference”.

```
The put pointer determines the next location in the output sequence where the
next output operation is going to take place.

ostream::*                                                  public member function

streampos tellp ( );

Returns the absolute position of the put pointer.

ostream& seekp ( streampos pos );
ostream& seekp ( streamoff off, ios_base::seekdir dir );

Set position of the put pointer

dir          offset is relative to...
ios_base::beg beginning of the stream buffer
ios_base::cur current position in the stream buffer
ios_base::end end of the stream buffer
```

Random write access to the contents of a file stream is possible.

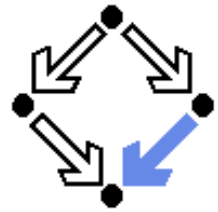
Example



cplusplus.com: “C++ Reference” (modified)

```
#include <fstream>
using namespace std;

int main () {
  long pos;

  ofstream outfile("test.txt");

  outfile.write ("This is an apple",16);
  pos=outfile.tellp();
  outfile.seekp (pos-7);
  outfile.write (" sam",4);

  outfile.close();
  return 0;
}

This is a sample
```

Stream Buffers



All stream I/O is automatically buffered in memory.

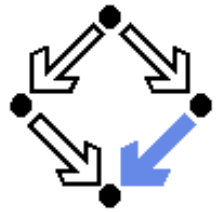
- I/O streams have buffer objects associated to them.

```
ios::rdbuf                                public member function

streambuf* rdbuf ( ) const;               // get stream buffer
streambuf* rdbuf ( streambuf* sb );       // set stream buffer
```

- Copying of streams possible by referring to their buffers.

```
// copy as many characters as possible from/to sb
ostream& operator<< (streambuf* sb );
istream& operator>> (streambuf* sb );

// copy whole content of in to out
out << in.rdbuf(); // first alternative
in >> out.rdbuf(); // second alternative
```

The explicit use of buffers allows the redirection of streams.



1. Overview

2. Text Input/Output

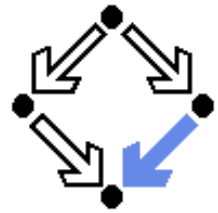
3. Error Handling

4. File Streams

5. Binary Input/Output

6. File System Operations

File System Operations

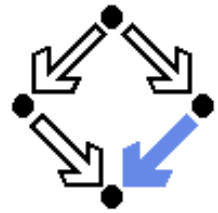


D. Ryan Stephens et al: C++ Cookbook (modified)

```
#include <iostream>
#include <ctime>
#include <sys/types.h>
#include <sys/stat.h>
#include <cerrno>
#include <cstring>

int main(int argc, char** argv ) { // usage: fileinfo <filename>
   struct stat fileInfo;
   if (stat(argv[1], &fileInfo) != 0) return(EXIT_FAILURE);
   if ((fileInfo.st_mode & S_IFMT) == S_IFDIR)
      std::cout << "Type: Directory\n";
   else
      std::cout << "Type: File\n";
   std::cout << "Size:     " << fileInfo.st_size << '\n';
   std::cout << "Device:   " << (char)(fileInfo.st_dev + 'A') << '\n';
   std::cout << "Created:  " << std::ctime(&fileInfo.st_ctime);
   std::cout << "Modified: " << std::ctime(&fileInfo.st_mtime);
}
```

For operating on the file system level, C system calls are needed.

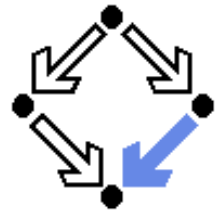
Deleting a File



D. Ryan Stephens et al: C++ Cookbook (modified)

```
#include <iostream>
#include <cstdio>
#include <cerrno>

using namespace std;

// Usage: remove <filename>
int main(int argc, char** argv) {
   if (argc != 2) {
      cerr << "You must supply a file name to remove." << endl;
      return(EXIT_FAILURE);
   }

   if (remove(argv[1]) != 0) {  // remove( ) returns non-0 on error
      cerr << "Error: " << strerror(errno) << endl;
      return(EXIT_FAILURE);
   }
   else
      cout << "File '" << argv[1] << "' removed." << endl;
}
```

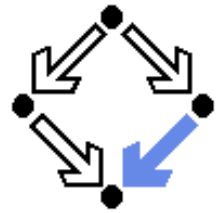
Renaming a File



D. Ryan Stephens et al: C++ Cookbook (modified)

```
#include <iostream>
#include <cstdio>
#include <cerrno>

using namespace std;

// Usage: rename <oldname> <newname>
int main(int argc, char** argv) {
   if (argc != 2) {
      cerr << "You must supply a file name to rename." << endl;
      return(EXIT_FAILURE);
   }

   if (rename(argv[1], argv[2]) != 0) {  // rename( ) returns non-0 on error
      cerr << "Error: " << strerror(errno) << endl;
      return(EXIT_FAILURE);
   }
   else
      cout << "File '" << argv[1] << "' renamed to '" << argv[2] << "'" << endl;
}
```

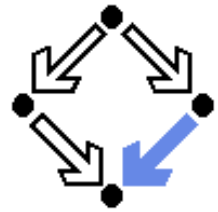
Creating a Temporary File



D. Ryan Stephens et al: C++ Cookbook (modified)

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdio>
using namespace std;

int main( ) {
   ofstream out;
   char* name = NULL;
   do { // must retry because other program might choose same file name
     name = tmpnam(NULL);
     in (!name) return(EXIT_FAILURE);
     out.open(name);
   }
   while(!out);

   out << "Here is some temporary data.";
   out.close( );
   return 0;
}
```

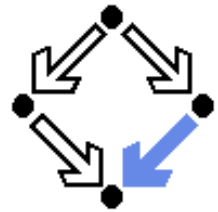
Creating a Directory



D. Ryan Stephens et al: C++ Cookbook (modified)

```
#include <iostream>
#include <direct.h>

// Usage: mkdir <dirname>
int main(int argc, char** argv) {

   if (argc < 2) {
      std::cerr << "Usage: " << argv[0] << " [new dir name]\n";
      return(EXIT_FAILURE);
   }

   if (mkdir(argv[1]) != 0) {  // Create the directory
      std::cerr << "Error: " << strerror(errno);
      return(EXIT_FAILURE);
   }
}
```

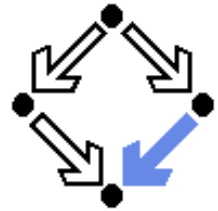
Removing a Directory



D. Ryan Stephens et al: C++ Cookbook (modified)

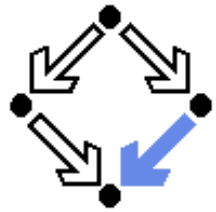
```
#include <iostream>
#include <direct.h>

using namespace std;

// Usage: rmdir <dirname>
int main(int argc, char** argv) {

   if (argc < 2) {
      cerr << "Usage: " << argv[0] << " [dir name]" << endl;
      return(EXIT_FAILURE);
   }

   if (rmdir(argv[1]) != 0) {  // Remove the directory
      cerr << "Error: " << strerror(errno) << endl;;
      return(EXIT_FAILURE);
   }
}
```

Reading the Contents of a Directory



```
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
#include <vector>
#include <string>
#include <iostream>
using namespace std;

vector<string> *listDir (string dir);

// Usage: dir [ <dirname> ]
int main(int argc, char* argv[]) {
    string dir;
    if (argc < 2) dir = "."; else dir = argv[1];

    vector<string> *files = listDir(dir);
    if (files == NULL) return -1;

    for (unsigned int i = 0;i < files->size();i++)
        cout << files->operator[](i) << endl;
    return 0;
}
```

Reading the Contents of a Directory



```
vector<string> *listDir (string dir) {
    DIR *dp = opendir(dir.c_str());
    if(dp == NULL) return NULL;

    vector<string> *files = new vector<string>;
    while (true) {
      struct dirent *dirp = readdir(dp);
      if (dirp == NULL) break;
      files->push_back(string(dirp->d_name));
    }

    closedir(dp);
    return files;
}
```

Type `struct dirent` contains all meta-data about a directory entry.