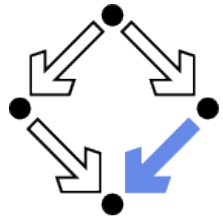
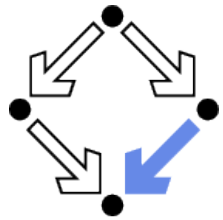
Object-Oriented Programming in C++

(Course "Programming 2")

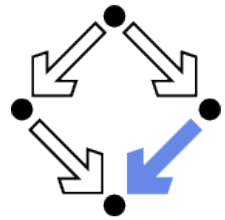
Wolfgang Schreiner
Wolfgang.Schreiner@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
https://www.risc.jku.at





Overview

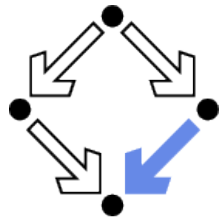


A continuation of the course "Programming 1" in the last semester.

- Last semester: (mostly) procedural ("imperative") prog. in C++.
  - Focus on organization of control flow.
  - Passive entities ("data") processed by active entities ("functions").
  - Programs organized as sets of functions.
- This semester: (mostly) object-oriented programming in C++.
  - Focus on organization of data.
  - "Classes" combine data and functions to "objects".
  - Programs organized as sets of classes.

Modern approach to "programming in the large".

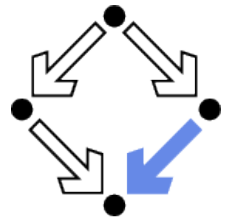
Example



```
struct Date {
  int day;
  int month;
}

static void print(Date d)
{
  cout << d.day << "."
    << d.month << ".";
}

Date d;
d.day = 24;
d.month = 12;
print(d);
```

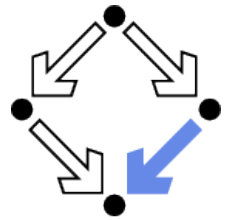
```
class Date {
  private:
    int day;
    int month;
  public:
    Date(int d, int m) {
      day = d;
      month = m;
    }
    void print() {
      cout << day << "."
        << month << ".";
    }
}

Date d(24, 12);
d.print();
```

Topics



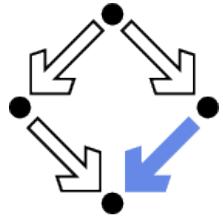
- Classes: combining data and functions.
  - Classes versus records ("structs").
  - Construction, destruction, assignments.
  - Static members versus non-static members.
- Inheritance: building class hierarchies.
  - Classes and subclasses.
  - Virtual functions and overriding.
  - Abstract classes, interfaces, frameworks.
- Templates: type-generic ("polymorphic") programming.
  - Function templates.
  - Class templates.
- The C++ Standard Library: reusing existing functionality.
  - Basic features (I/O, numerics, etc).
  - Containers, iterators, and algorithms.

Organization



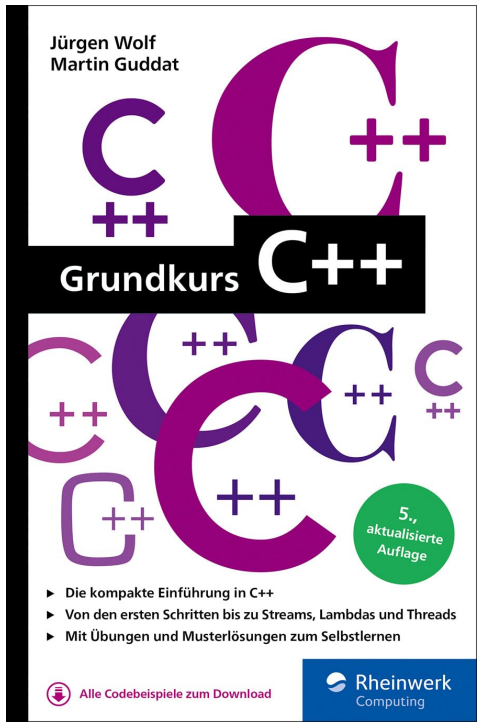
- Lecture: concepts and examples.
  - Slides, blackboard, online demonstrations.
- Assignments: moderate programming exercises.
  - 6 assignments are given.
    - Results to be submitted within 2 weeks.
  - Teaching Assistants: Gabriel Eckertsberger and Thomas Michlmayr.
    - See the course site for the date of the meeting offered every week.
- Grades: based on final exam and assignments.
  - Final exam (50%): concepts and small programming tasks.
  - Assignments (50%): best 5 results are evaluated.
  - Both exam and assignments have to be positive.
- Literature: no lecture notes, get a C++ textbook!
  - C++ is complex and involves many details.
  - Lecture notes could not provide a reasonable substitute.

Literature

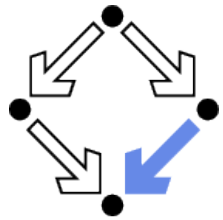


- Jürgen Wolf, Martin Guddat: *Grundkurs C++*, Rheinwerk Computing, 5. Auflage, 2025.
- 18 chapters, 811 pages.
- Chapters 1–10: previous semester.
- Chapters 11–18: this semester.
- EUR 15,40 (Amazon).

Compact C++ introduction in German.



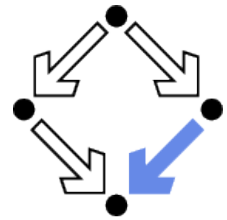
Literature



- Thomas Theis, *Einstieg in C++*, Rheinwerk Computing, 2. Auflage, 2020.
- 17 chapters, 547 pages.
- Chapters 1–9: previous semester.
- Chapters 10–14: this semester.
- EUR 27,92 (Amazon).

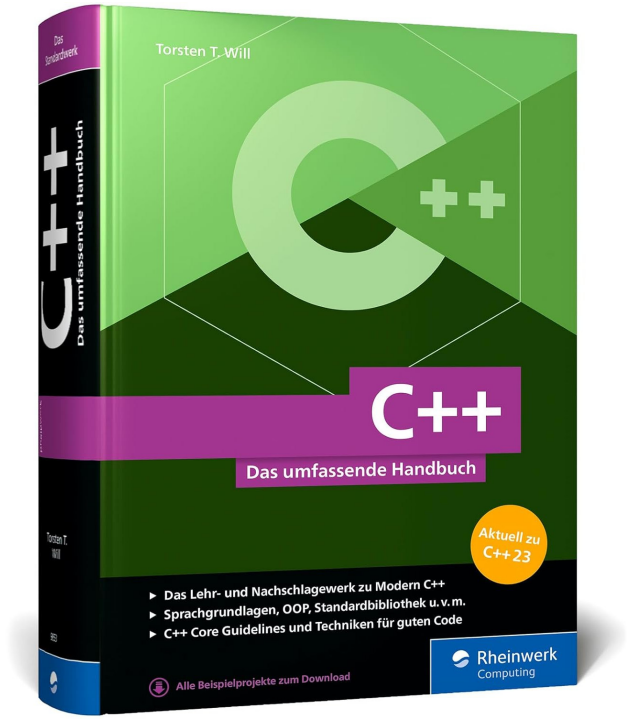
Compact C++ introduction in German.

Literature

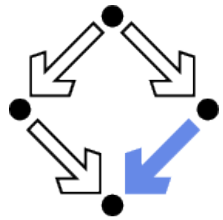


- Torsten T. Will, *C++: Das umfassende Handbuch*, Rheinwerk Computing, 3. Auflage, 2024.
- 29 chapters in 4 parts, 1172 pages.
- Part I: previous semester.
- Parts II-IV (selected): this semester.
- EUR 41,10 (Amazon).

Comprehensive C++ introduction in German.

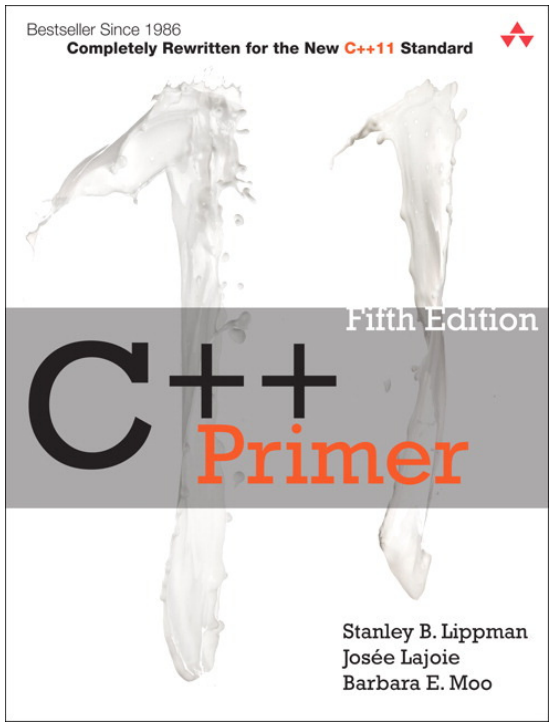


Literature

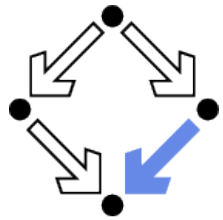


- Stanley B. Lippman, Barbara E. Moo, Josee Lajoie *C++ Primer*, Addison-Wesley, 5th edition, 2012.
- 19 chapters + appendix, 900 pages.
- Chapters 1–6: previous semester.
- Chapters 7–16: this semester.
- EUR 56,09 (Amazon).

Comprehensive C++ introduction in English.

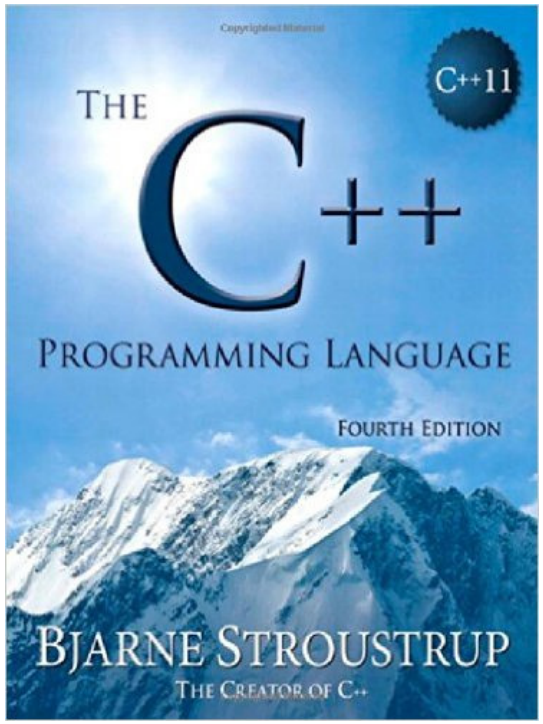


Literature

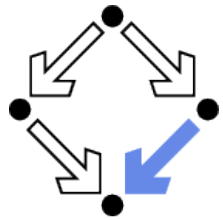


- Bjarne Stroustrup, *The C++ Programming Language*, 4th edition, Addison-Wesley, 2013.
- 44 chapters, 1300 pages.
- Chapters 1–15: previous semester.
- Chapters 16–33: this semester.
- EUR 28,46 (Amazon Kindle).

Not an introduction, but the full reference.



Free Resources

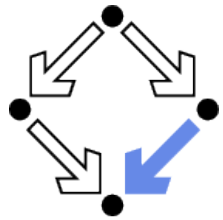


Plenty of free resources in the web.

- C++ language tutorial.
  - 140 pages (PDF), short overview of main language features.
- C++ library reference.
  - A hypertext reference for the C++ standard library.

See the course site for the URLs.

C++ Standards



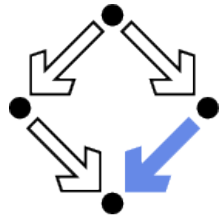
We mainly present the original C++98 standard (revised as C++03) with a few selected features from newer versions.

- Newer: C++11, C++14, C++17, C++20, C++23.
  - C++17 still seems to be most widely used in industry (with C++20 gaining steam).
- Current: C++26.
  - Not yet fully supported in all compilers/IDEs.

`https://github.com/AnthonyCalandra/modern-cpp-features`
`https://en.cppreference.com/w/cpp/compiler_support`

For easy portability, it may be healthy to be a bit conservative.

Pointers

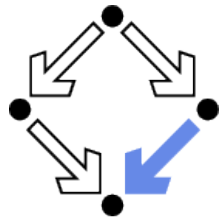


We will mostly present the use of classical pointers rather than "smart pointers".

- Classical ("raw") pointers: `T* p = new T(...); ...; delete p;`
  - Explicit deallocation required to avoid memory leaks.
- Unique pointers: `unique_ptr<T> u(new T(...));`
  - Cannot be copied.
  - Automatic deallocation when pointer falls out of scope.
  - Only for temporary objects (within a function)!
- Shared pointers: `shared_ptr<T> s(new T(...));`
  - Can be copied, number of references to object is tracked.
  - Automatic deallocation when object is not referenced any more.
  - Some runtime overhead, only for non-cyclic data structures!
- Weak pointers: `weak_ptr<T> w = s;`
  - Can be copied from shared pointer, does not increase reference count.
  - Dereferencing: `if (shared_ptr<T> s2 = w.lock()) {..*s2..}`
  - (Mostly) for avoiding cyclic references.

When using smart pointers, be sure to understand their properties/restrictions.

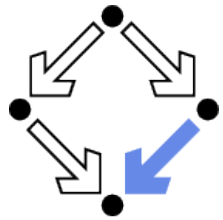
Moodle Course



Central point for electronic communication.

- Enrol as a participant in the Moodle course.
  - All forum messages will be sent as emails to registered participants.
- Submit assignments via Moodle.
  - No email submissions are accepted.
- Post questions in the "Questions and Answers" forum.
  - Answered by Wolfgang Schreiner or by one of the tutors.

https://www.risc.jku.at/people/schreine/courses/ss2026/prog2



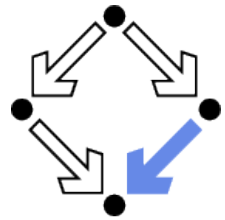
1. Overview

2. **Development**

3. Graphical Output

4. Text Input and Output

C++ Development



Free integrated development environments (IDEs) for C++.

- Eclipse IDE for C/C++ Developers

  `https://www.eclipse.org/downloads/packages/`

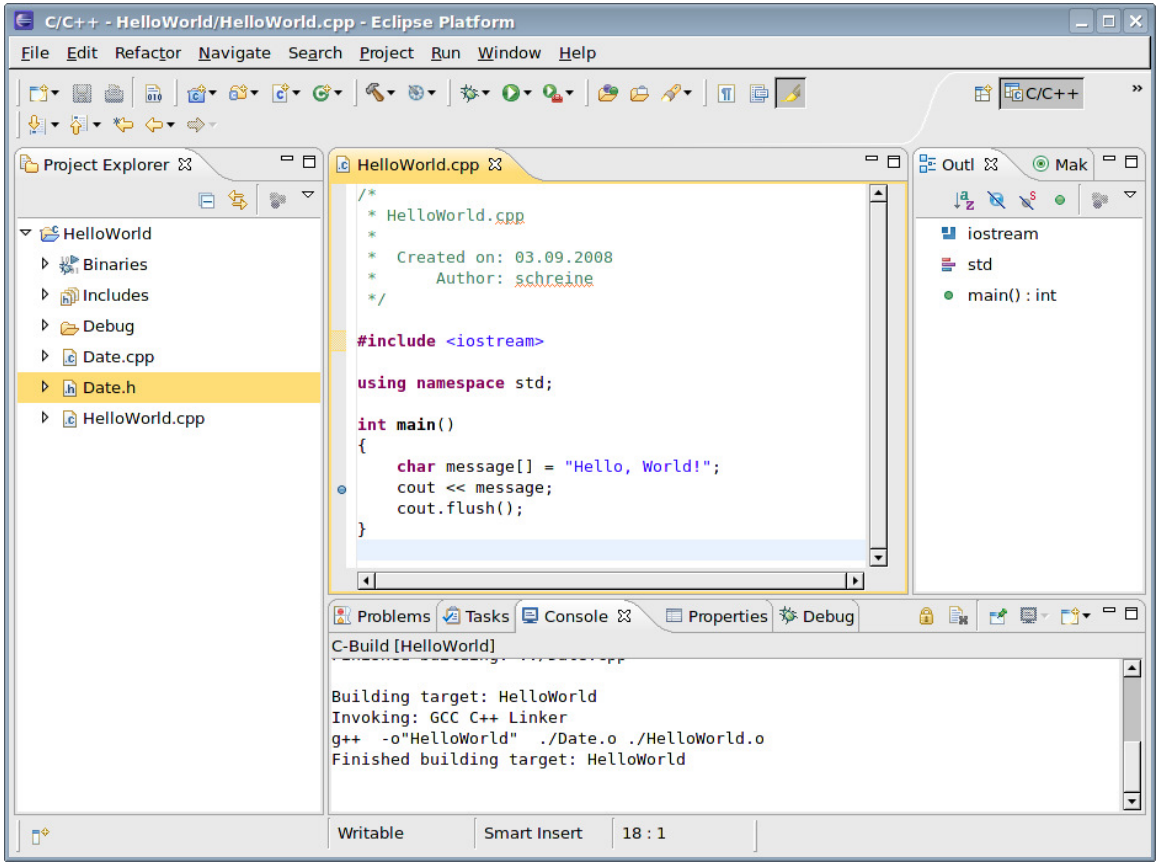
  - GNU/Linux and MacOS X: The GNU C++ Compiler.
    - Debian: `apt-get install g++`
  - Windows: MSYS2 and MinGW-w64.

    `https://www.msys2.org/`
- Microsoft Visual Studio Community

  `https://www.visualstudio.com/downloads`

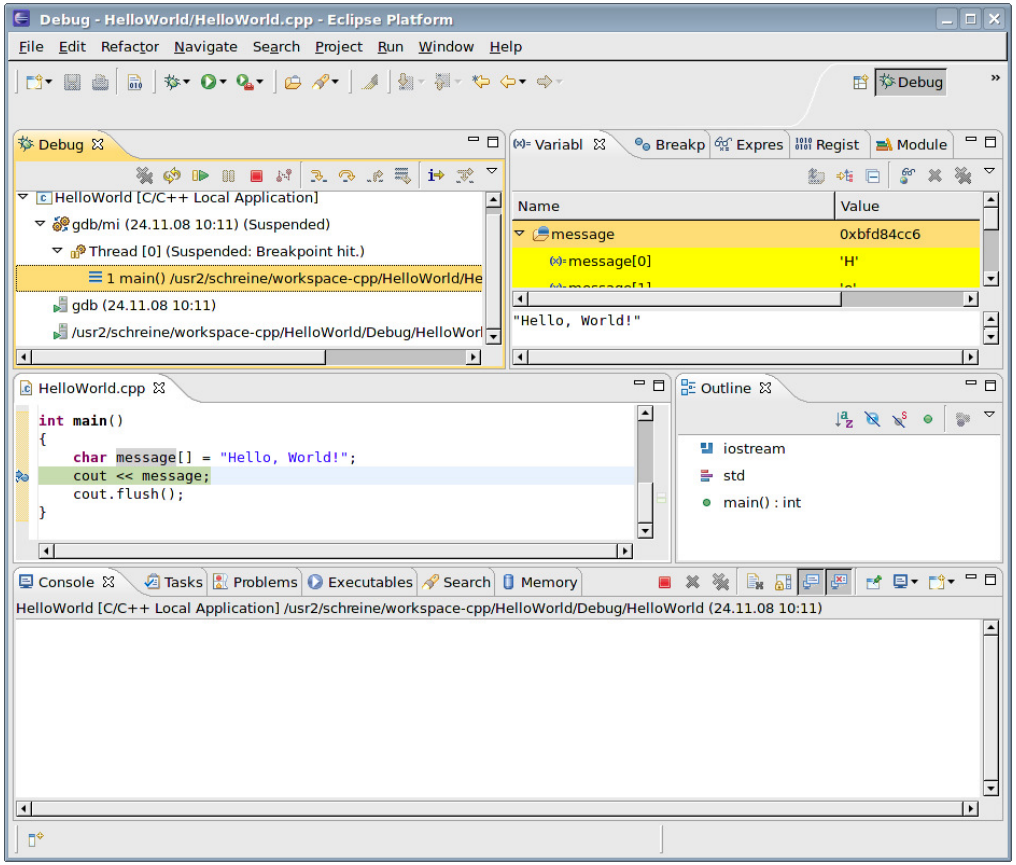
A decent IDE makes program development much more productive.

Eclipse IDE for C/C++ Developers

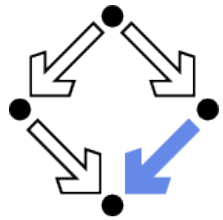


Good support for development and for correcting compilation errors.

Eclipse Debug View



Learn to use a debugger for correcting runtime errors!



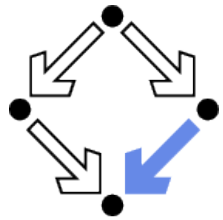
1. Overview

2. Development

3. **Graphical Output**

4. Text Input and Output

Graphical Output

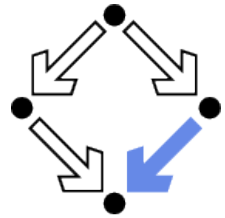


Not in the C++ standard but system-dependent.

- We provide a small portable library for drawing pictures.
  - Based on the CImg library `https://cimg.eu`
  - Works for GNU/Linux, Microsoft Windows, MacOS X.
- Library consists of three source files:
  - `Drawing.h`
  - `Drawing.cpp`
  - `CImg.h`
- Download files from course site and use for your programs.
  - `Drawing.h` must be included in sources.
  - `Drawing.cpp` (includes `CImg.h`) has to be compiled and linked to executable program.
- Usage is demonstrated in file `Main.cpp`.

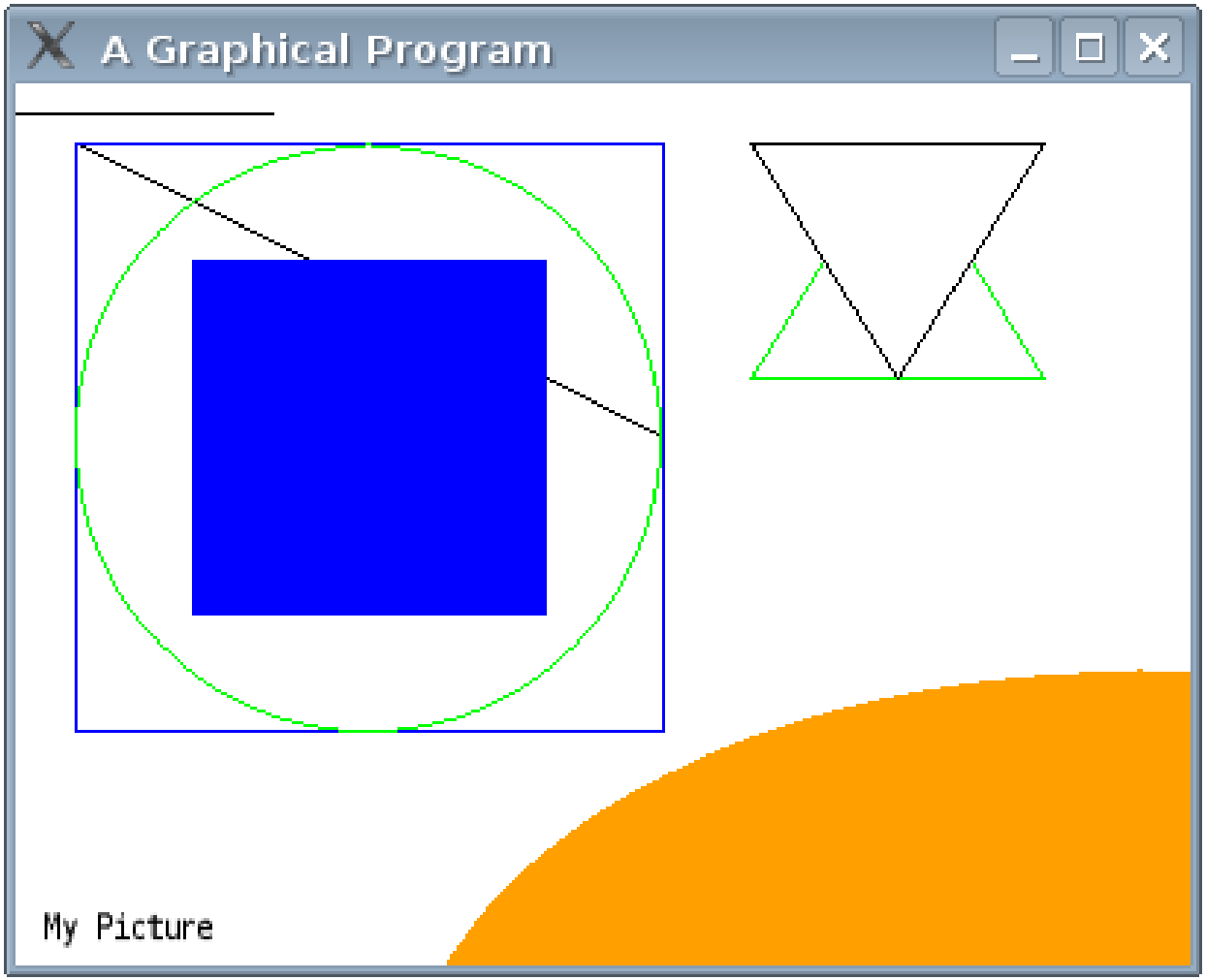
See the course site for more information.

Example (file `Main.cpp`)

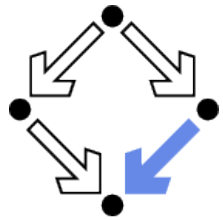


```
#include "Drawing.h"
using namespace compsys;

const unsigned int BLACK = 0x000000;
const unsigned int WHITE = 0xFFFFFF;
...

int main() {
  beginDrawing(400, 300, "A Graphical Program");
  drawLine(20, 20, 220, 120);
  drawRectangle(20, 20, 200, 200, BLUE);
  fillRectangle(60, 60, 120, 120, BLUE);
  drawEllipse(21, 21, 198, 198, GREEN);
  fillEllipse (getWidth()/3, getHeight()*2/3, 500, 300, ORANGE);
  int xs[] = { 300, 350, 250 }; int ys[] = { 20, 100, 100 };
  drawPolygon(3, xs, ys, GREEN);
  int xs0[] = { 300, 350, 250 }; int ys0[] = { 100, 20, 20 };
  fillPolygon(3, xs0, ys0, WHITE, BLACK);
  drawText(10, 280, "My Picture");
  for (int i=0; i<getWidth(); i++) drawPoint(i, 10);
  endDrawing();
}
```

Example

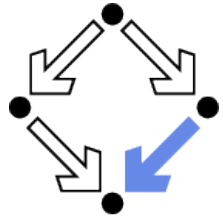


Interface (file `Drawing.h`)



```
void beginDrawing(int width, int height, const char *title,
                  unsigned int color = 0xFFFFFF, bool flush = true);
void endDrawing();
void flush();
int getWidth();
int getHeight();
void drawPoint(int x, int y, unsigned int color = 0);
void drawLine(int x0, int y0, int x1, int y1, unsigned int color = 0);
void drawRectangle(int x, int y, int w, int h, unsigned int color = 0);
void fillRectangle(int x, int y, int w, int h,
                   unsigned int fcolor = 0, unsigned int ocolor = NO_COLOR);
void drawEllipse(int x, int y, int w, int h, unsigned int color = 0);
void fillEllipse(int x, int y, int w, int h,
                 unsigned int fcolor = 0, unsigned int ocolor = NO_COLOR);
void drawPolygon(int n, int* xs, int *ys, unsigned int color = 0);
void fillPolygon(int n, int* xs, int *ys,
                 unsigned int fcolor = 0, unsigned int ocolor = NO_COLOR);
void drawText(int x, int y, const char *text,
              int size = 14, unsigned int color = 0);
```

See “Drawing.h” for the detailed specification of the interface.



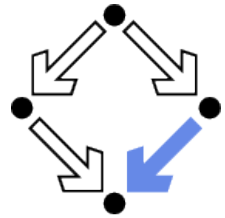
1. Overview

2. Development

3. Graphical Output

4. **Text Input and Output**

C Standard Input/Output



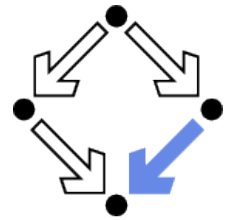
Also in C++, the standard I/O functions of C can be used.

```
#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.

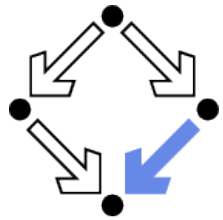
    Class `fstream` (file streams).
- Stream sources/sinks may be abstract devices.

  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.

I/O Operators



The shift operators are overloaded for all kinds of streams.

- 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:

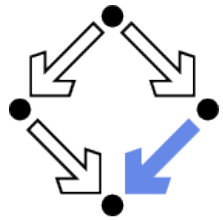
    ```
    out << val1 << val2 << ...;
    ```

  - Read multiple values from an input stream:

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

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

Reading a Sequence of Values



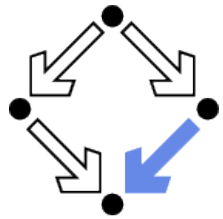
A common pattern for reading a sequence of values.

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

- The conversion of a stream to a boolean indicates the success of the last operation on that stream.
  - If `(!cin)` yields "true", the last operation on `cin` has failed (e.g. because of an unsuccessful attempt to read from the stream).

Check the status of input stream after every read operation!

I/O Manipulators

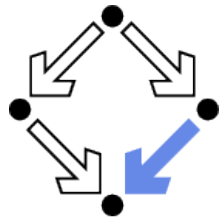


Manipulators can be inserted into a stream to influence its behavior.

```
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.
  - ...

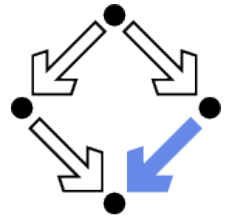
I/O Manipulators



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)
```

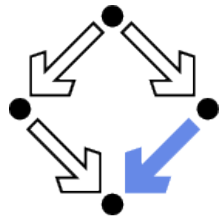
Example



cplusplus.com: “C++ Reference”.

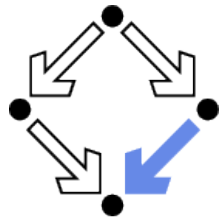
```
#include <iostream>
using namespace std;

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

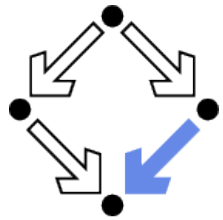


cplusplus.com: “C++ Reference”.

```
#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
```

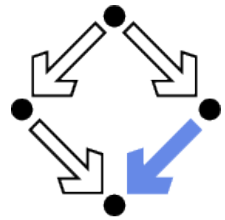
I/O Manipulators



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

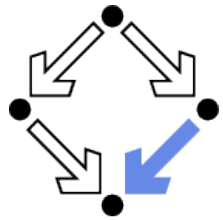


cplusplus.com: “C++ Reference”.

```
#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

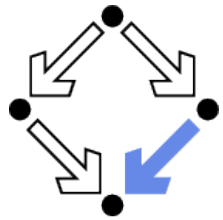


cplusplus.com: “C++ Reference”.

```
#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
```

String Streams

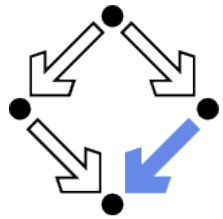


cplusplus.com: “C++ Reference”

```
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.

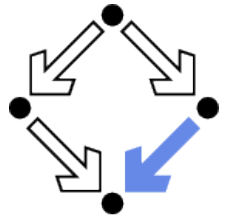
Example



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;
}
```

File Streams

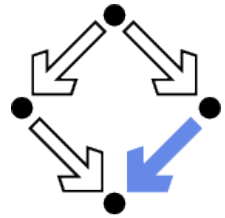


cplusplus.com: “C++ Reference”.

```
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.
```

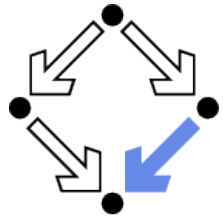
File streams represent the contents of files (not the file system data).

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;
}
```

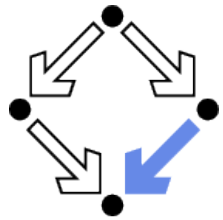
Constructing a File Stream



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).
```

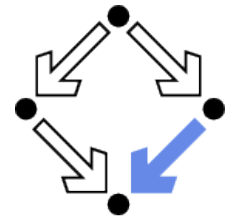
Opening a File Stream



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



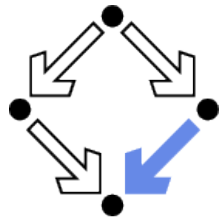
ios_base::openmode public member type

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

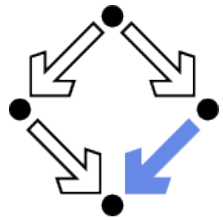


cplusplus.com: “C++ Reference”.

```
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

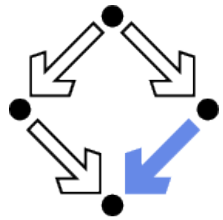


cplusplus.com: “C++ Reference”.

```
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).

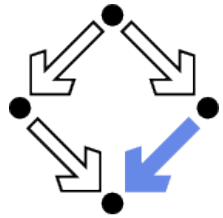
Example



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

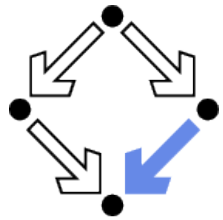


cplusplus.com: “C++ Reference”.

```
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);
```

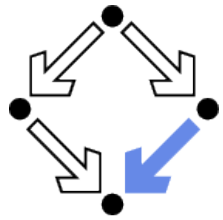
Specialized to support only the corresponding read/write operations.

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;
}
```