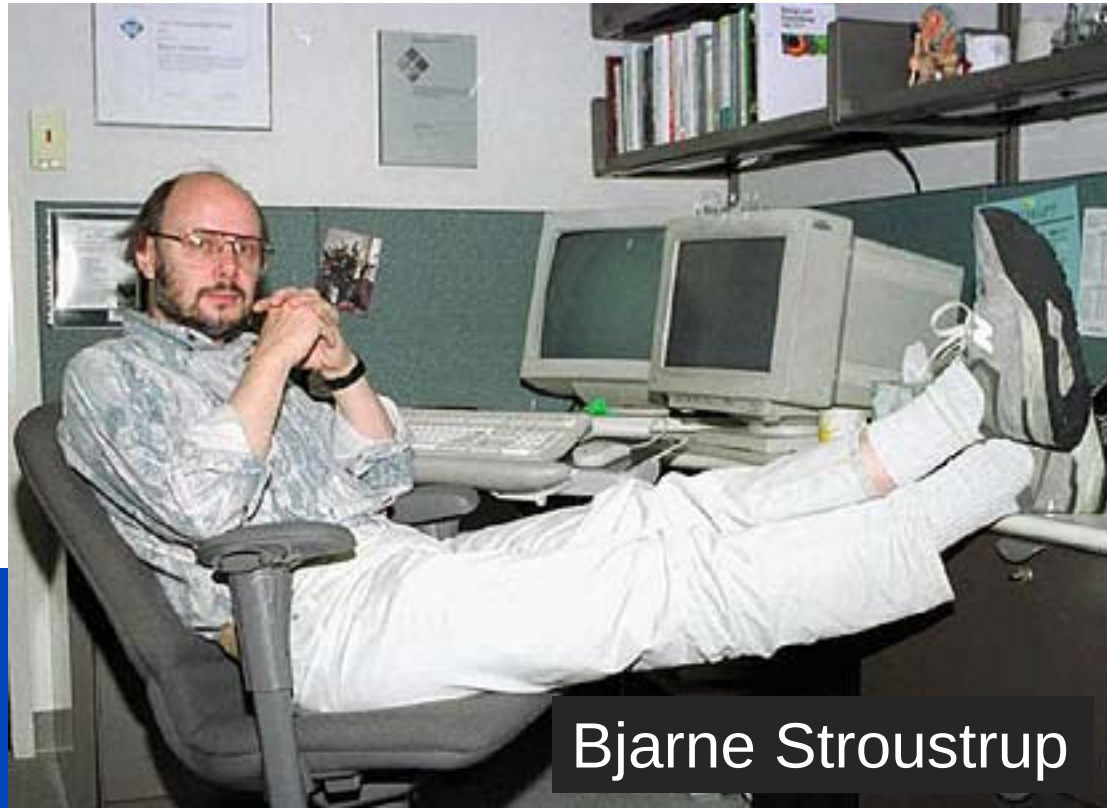
2009/06/17

How source code becomes executable (the “Play” button)

Daniel Maleike



How source code becomes executable

maleike@muhu:~/src/test

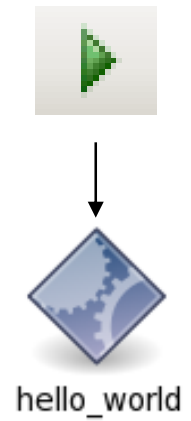
```
 1 class Shape
 2 {
 3
 4   public:
 5
 6     virtual double GetArea() = 0;
 7
 8 #ifdef GRAPHICAL_ENVIRONMENT
 9     virtual void Draw() = 0;
10 #endif
11
12 };
13
                                13,0-1        All
```

maleike@muhu:~/src/test

```
 1 #include "rectangle.h"
 2
 3 #include <iostream>
 4
 5 int main(int argc, char** argv)
 6 {
 7   std::cout << "Hello World." << std::endl;
 8
 9   Rectangle world(152.0, 80.5);
10
11   std::cout << "Your area is " << world->Area() << std::endl;
12
13   return 0;
14 }
15
"~/src/test/main.cpp" 15L, 239C written                8,2           All
```

maleike@muhu:~/src/test

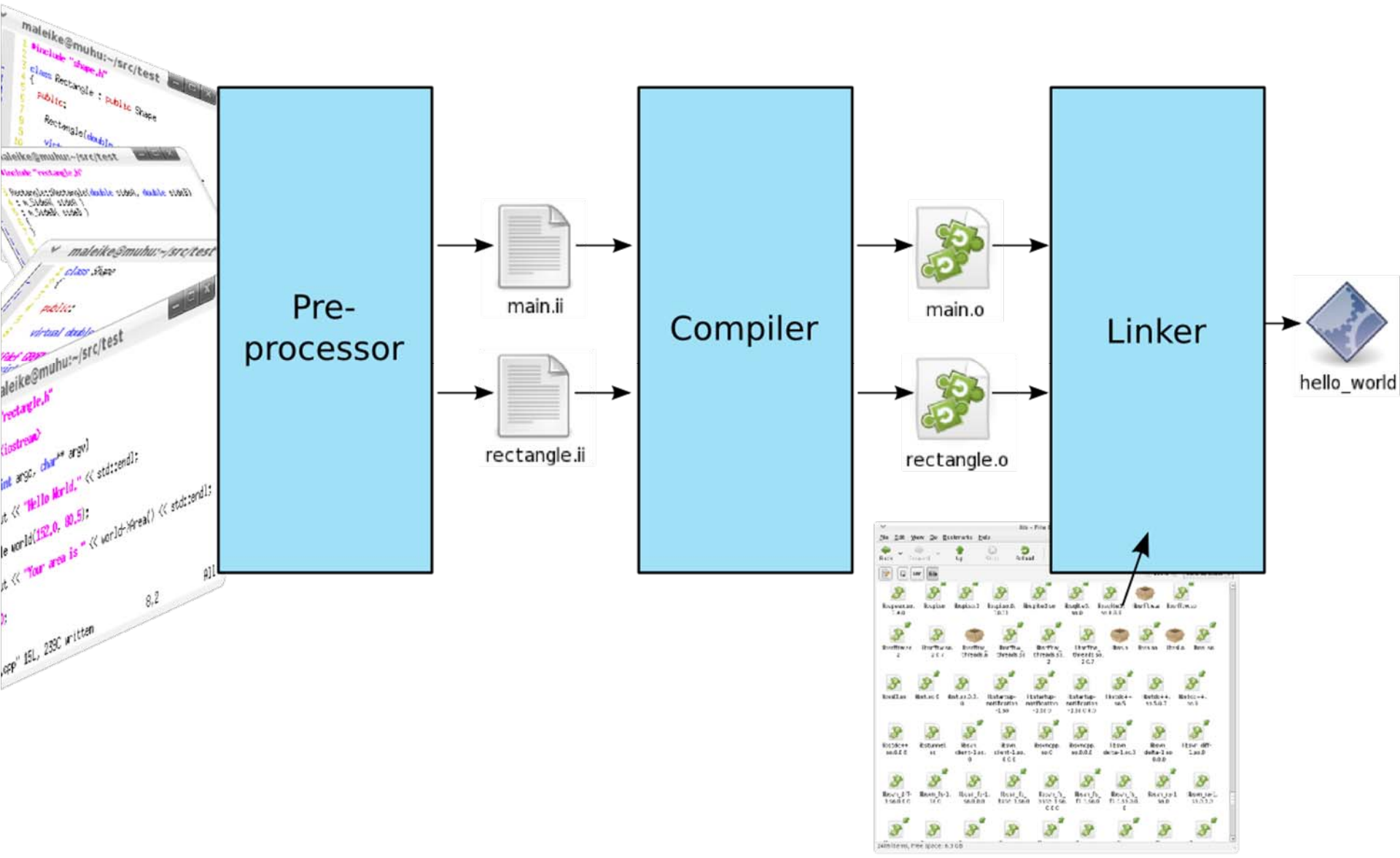
```
 1 #include "shape.h"
 2
 3 class Rectangle : public Shape
 4 {
 5   public:
 6
 7     Rectangle(double sideA, double sideB);
 8
 9     virtual double GetArea();
10
11 #ifdef GRAPHICAL_ENVIRONMENT
12     virtual void Draw();
13 #endif
14
15   protected:
16
17     double m_Side;
18 };
19
<t/rectangle.h" 19L, 238C written 19,0-1        All
```



maleike@muhu:~/src/test

```
 1 #include "rectangle.h"
 2
 3 Rectangle::Rectangle(double sideA, double sideB)
 4 : m_SideA( sideA )
 5 : m_SideB( sideB )
 6 {
 7 }
 8
 9 double Rectangle::GetArea()
10 {
11   return m_SideA * m_SideB;
12 }
13
14 #ifdef GRAPHICAL_ENVIRONMENT
15 void Rectangle::Draw()
16 {
17   // later
18 }
19 #endif
                                13,0-1        All
```

How source code becomes executable



The Preprocessor

Simple text substitution before compile-time

- `#include <filename>`
- `#define/#undef condition`
- `#ifdef/#ifndef condition, #else, #endif`
- `#define RADTODEG(x) ((x) * 57.29578)`
- Stringification: `#x` expands to "`<exp. of x>`"
- Concatenation: `x ## y` concatenates
  the expansions of x and y

The Preprocessor

Simple text substitution be

- #include <filename
- #define/#undef con

maleike@muhu:~/src/test

```
 1 #include "rectangle.h"
 2
 3 Rectangle::Rectangle(double sideA, double sideB)
 4 : m_SideA( sideA )
 5 : m_SideB( sideB )
 6 {
 7 }
 8
 9 double Rectangle::GetArea()
10 {
11   return m_SideA * m_SideB;
12 }
13
14 #ifdef GRAPHICAL_ENVIRONMENT
15 void Rectangle::Draw()
16 {
17   // later
18 }
19 #endif
~
                                13,0-1        All
```

```
 1 # 1 "<built-in>"
 2 # 1 "<command line>"
 3 # 1 "rectangle.cpp"
 4 # 1 "rectangle.h" 1
 5 # 1 "shape.h" 1
 6 class Shape
 7 {
 8
 9   public:
10
11     virtual double GetArea() = 0;
12
13
14     virtual void Draw() = 0;
15
16
17 };
18 # 2 "rectangle.h" 2
19
20 class Rectangle : public Shape
21 {
22   public:
23
24     Rectangle(double sideA, double sideB);
25
26     virtual double GetArea();
27
28
29     virtual void Draw();
30
31
32   protected:
33
34     double m_SideA;
35     double m_SideB;
36 };
37 # 2 "rectangle.cpp" 2
38
39 Rectangle::Rectangle(double sideA, double sideB)
40 : m_SideA( sideA ),
41   m_SideB( sideB )
42 {
43 }
44
45 double Rectangle::GetArea()
46 {
47   return m_SideA * m_SideB;
48 }
49
50
51 void Rectangle::Draw()
52 {
53
54 }
55
                                9,0-1         All
```

Qt: moc, uic

moc

- produce meta-objects, needed for signals and slots in Qt

uic

- create code which can create the designed form at run-time

Useful links:

http://doc.trolltech.com/4.5/moc.html
http://doc.trolltech.com/4.5/designer-using-a-ui-file.html

The Compiler

- Translates one (complete) piece of source code into one piece of binary code

- Binary code may depend on external code (symbols)
  - variables/functions declared but not defined need to be defined somewhere else before execution
  - jumps/function calls to unknown addresses
  - from an early C++ compiler: name mangling

```
# 1 "rectangle.cpp"
# 1 "rectangle.h" 1
# 1 "shape.h" 1
class Shape
{

  public:

    virtual double GetArea() = 0;


    virtual void Draw() = 0;


};
# 2 "rectangle.h" 2

class Rectangle : public Shape
{
  public:

    Rectangle(double sideA, double sideB);

    virtual double GetArea();


    virtual void Draw();


  protected:

    double m_SideA;
    double m_SideB;
};
# 2 "rectangle.cpp" 2

Rectangle::Rectangle(double sideA, double sideB)
: m_SideA( sideA ),
  m_SideB( sideB )
{
}

double Rectangle::GetArea()
{
  return m_SideA * m_SideB;
}


void Rectangle::Draw()
{

}
                                        9,0-1         All
```

lete) piece of source code into one

end on external code (symbols)

s declared but not defined need to be
re else before execution

ls to unknown addresses

compiler: name mangling

```
maleike@muhu:~/src/test
[maleike@muhu test]$ nm --defined rectangle.o
00000000 W _ZN5ShapeC2Ev
000000a4 T _ZN9Rectangle4DrawEv
00000090 T _ZN9Rectangle7GetAreaEv
00000048 T _ZN9RectangleC1Edd
00000000 T _ZN9RectangleC2Edd
00000000 V _ZTI5Shape
00000000 V _ZTI9Rectangle
00000000 V _ZTS5Shape
00000000 V _ZTS9Rectangle
00000000 V _ZTV5Shape
00000000 V _ZTV9Rectangle
```

```
# 1 "rectangle.cpp"
# 1 "rectangle.h" 1
# 1 "shape.h" 1
class Shape
{

  public:

    virtual double GetArea() = 0;


    virtual void Draw() = 0;


};
# 2 "rectangle.h" 2

class Rectangle : public Shape
{
  public:

    Rectangle(double sideA, double sideB);

    virtual double GetArea();


    virtual void Draw();


  protected:

    double m_SideA;
    double m_SideB;
};
# 2 "rectangle.cpp" 2

Rectangle::Rectangle(double sideA, double sideB)
: m_SideA( sideA ),
  m_SideB( sideB )
{
}

double Rectangle::GetArea()
{
  return m_SideA * m_SideB;
}


void Rectangle::Draw()
{

}
                                        9,0-1         All
```

lete) piece of source code into one

end on external code (symbols)

s declared but not defined need to be
re else before execution

ls to unknown addresses

compiler: name mangling

```
maleike@muhu:~/src/test
[maleike@muhu test]$ nm -C --defined rectangle.o
00000000 W Shape::Shape()
000000a4 T Rectangle::Draw()
00000090 T Rectangle::GetArea()
00000048 T Rectangle::Rectangle(double, double)
00000000 T Rectangle::Rectangle(double, double)
00000000 V typeinfo for Shape
00000000 V typeinfo for Rectangle
00000000 V typeinfo name for Shape
00000000 V typeinfo name for Rectangle
00000000 V vtable for Shape
00000000 V vtable for Rectangle
```

```
# 1 "rectangle.cpp"
# 1 "rectangle.h" 1
# 1 "shape.h" 1
class Shape
{

  public:

    virtual double GetArea() = 0;


    virtual void Draw() = 0;


};
# 2 "rectangle.h" 2

class Rectangle : public Shape
{
  public:

    Rectangle(double sideA, double sideB);

    virtual double GetArea();


    virtual void Draw();


  protected:

    double m_SideA;
    double m_SideB;
};
# 2 "rectangle.cpp" 2

Rectangle::Rectangle(double sideA, double sideB)
: m_SideA( sideA ),
  m_SideB( sideB )
{
}

double Rectangle::GetArea()
{
  return m_SideA * m_SideB;
}


void Rectangle::Draw()
{

}
                                        9,0-1         All
```

Tools:
Linux: `nm, c++filt`
Windows: `undname`

end on external code (symbols)

s declared but not defined need to be
re else before execution

ls to unknown addresses

compiler: name mangling

```
maleike@muhu:~/src/test
[maleike@muhu test]$ nm -C --defined rectangle.o
00000000 W Shape::Shape()
000000a4 T Rectangle::Draw()
00000090 T Rectangle::GetArea()
00000048 T Rectangle::Rectangle(double, double)
00000000 T Rectangle::Rectangle(double, double)
00000000 V typeinfo for Shape
00000000 V typeinfo for Rectangle
00000000 V typeinfo name for Shape
00000000 V typeinfo name for Rectangle
00000000 V vtable for Shape
00000000 V vtable for Rectangle
```

The Linker

- Combines binary modules into executable or library
  - relocates code (local addresses)
  - resolves symbol dependencies (sooner or later)

- Time of final relocation/symbol resolution
  - static linking: compile time (big files, fast execution)
  - dynamic linking: run-time (small files, reuse, sharing)

dkfz.

maleike@muhu:~/src/test

```
[maleike@muhu test]$ nm -C --defined rectangle.o
00000000 W Shape::Shape()
000000a4 T Rectangle::Draw()
00000090 T Rectangle::GetArea()
00000048 T Rectangle::Rectangle(double, double)
00000000 T Rectangle::Rectangle(double, double)
00000000 V typeinfo for Shape
00000000 V typeinfo for Rectangle
00000000 V typeinfo name for Shape
00000000 V typeinfo name for Rectangle
00000000 V vtable for Shape
00000000 V vtable for Rectangle
```

ules into executable or library

cal addresses)

lencies (sooner or later)

rectangle.o

/Symbol resolution

- static linking: compile tim iles, fast execution)

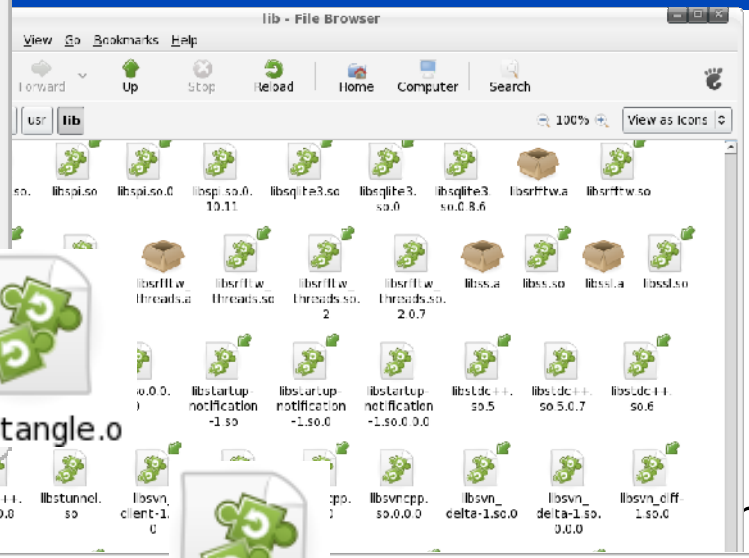
main.o

maleike@muhu t

```
[maleike@muhu test]$ nm -C main.o
00000046 t global constructors keyed to main
00000000 t __static_initialization_and_destruction_0(int, int)
         U Rectangle::GetArea()
         U Rectangle::Rectangle(double, double)
         U std::ostream::operator<<(std::ostream& (*)(std::ostream&))
         U std::ostream::operator<<(double)
         U std::ios_base::Init::Init()
         U std::ios_base::Init::~Init()
         U std::cout
         U std::basic_ostream<char, std::char_traits<char> >& std::endl<char, std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >
00000000 b std::__ioinit
         U std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char>
         U __cxa_atexit
         U __dso_handle
         U __gxx_personality_v0
0000005e t __tcf_0
00000072 T main
```

```
maleike@muhu:~/src/test
[maleike@muhu test]$ nm -C --defined rectangle.o
00000000 W Shape::Shape()
000000a4 T Rectangle::Draw()
00000090 T Rectangle::GetArea()
00000048 T Rectangle::Rectangle(double, double)
00000000 T Rectangle::Rectangle(double, double)
00000000 V typeinfo for Shape
00000000 V typeinfo for Rectangle
00000000 V typeinfo name for Shape
00000000 V typeinfo name for Rectangle
00000000 V vtable for Shape
00000000 V vtable for Rectangle
```

rectangle.o



hello_world

- static linking: co

main.o

```
maleike@muhu
[maleike@muhu test]$ nm -C main.o
00000046 t global constructors keyed to main
00000000 t __static_initialization_and_destruction_0(int, int)
         U Rectangle::GetArea()
         U Rectangle::Rectangle(double, double)
         U std::ostream::operator<<(std::ostream& (*)(std::ostream&))
         U std::ostream::operator<<(double)
         U std::ios_base::Init::Init()
         U std::ios_base::Init::~Init()
         U std::cout
         U std::basic_ostream<char, std::char_traits<char> >& std::endl<char, std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char> >
00000000 b std::__ioinit
         U std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits<char> >(std::basic_ostream<char, std::char_traits<char>
         U __cxa_atexit
         U __dso_handle
         U __gxx_personality_v0
0000005e t __tcf_0
00000072 T main
```

make, Visual Studio, cmake

make/Visual Studio

- Call generators, preprocessor, compiler, and linker
- Provide all the right include paths and library search paths
- Define some special defines for some source code
- All the calls in the right order

cmake

- Generate project information for make/Visual Studio
- A platform independent “make”

How to read error messages

Basics, but highly important

- start with the first message and ignore the rest
  - learn how to find the very first message
  - know your development environment
    - (with MS VisualStudio, get to know MSDN)
- read and understand the full line
  - DO NOT dive into code until you have read the last character of the message
  - This is especially important with template errors

http://mbits/cdash/index.php?project=MITK

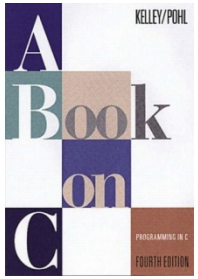
How to read error messages

Possible error sources

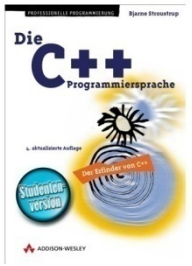
- Compiler (C++ level problems)
  - undeclared variables
  - unknown types (#include missing)
  - type mismatches
- Linker
  - missing libraries
  - missing symbols (details follow)
    - methods declared in header but not implemented
    - implemented but not in project (Cmake)
- Preprocessor

- Tool chain (CMake, Makefile, etc.)

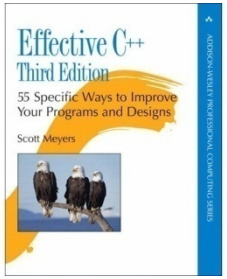
Books



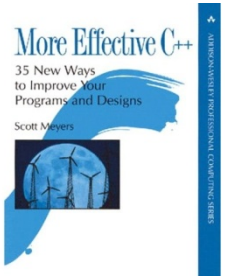
Kelley, A Book on C



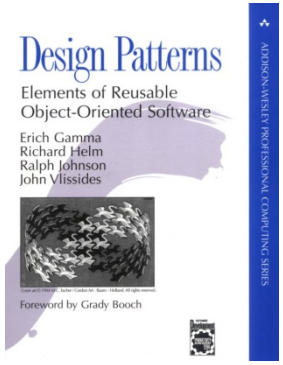
Stroustrup, The C++ Programming Language



Meyers, Effective C++



Meyers, More Effective C++



Gamma, Design Patterns

World Wide Web

Bjarne Stroustrup's C++ Style and Technique FAQ
http://www.research.att.com/~bs/bs_faq2.html

C++ FAQ LITE
http://www.parashift.com/c++-faq-lite/

C/C++ Reference
http://www.cppreference.com/

Google
http://www.google.com