Object-oriented Programming
for Automation & Robotics

Carsten Gutwenger

LS 11 Algorithm Engineering

Lecture 2 • Winter 2011/12 • Oct 18

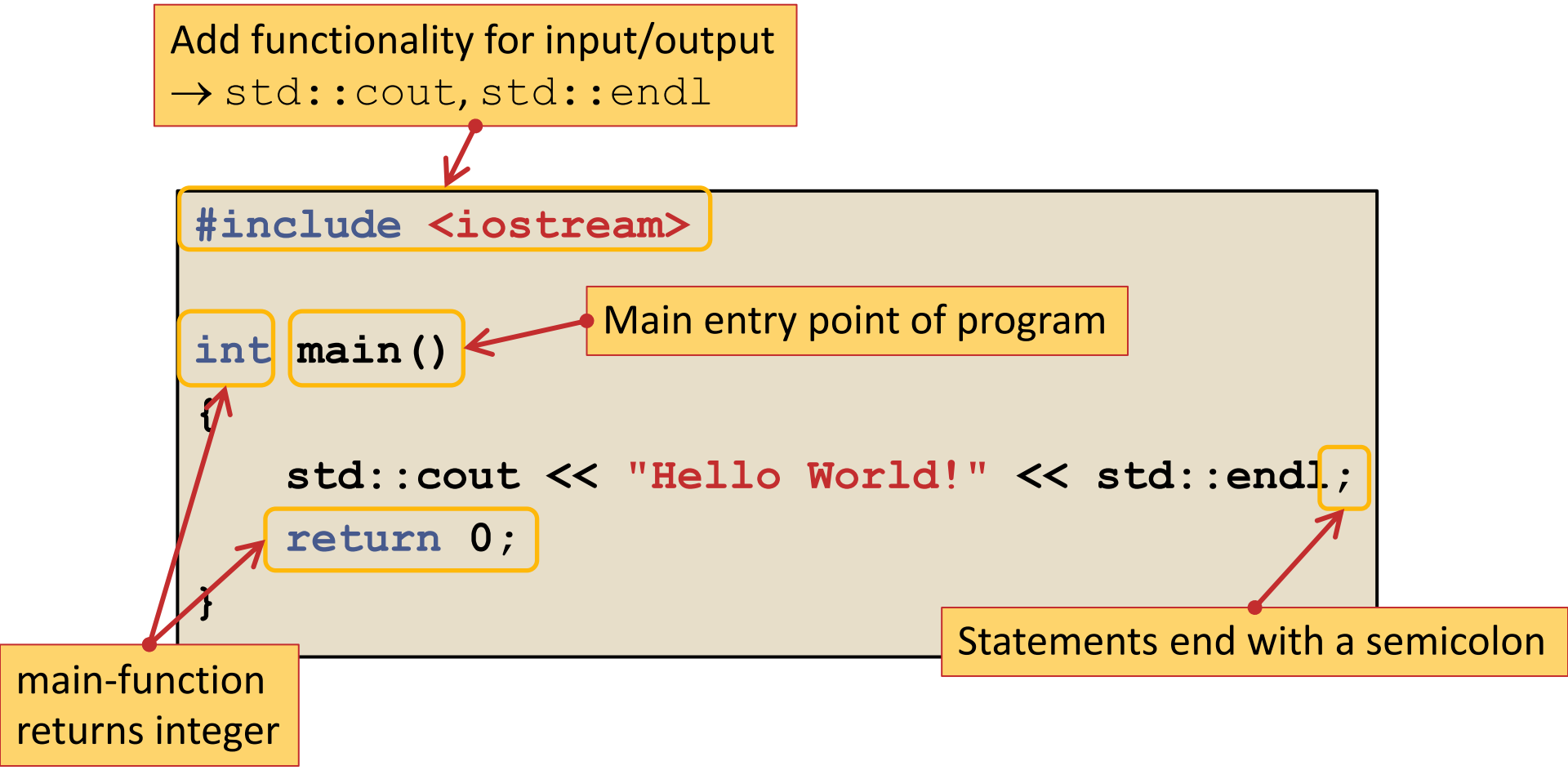
technische universität dortmund

department of computer science

Lessons learned last time…

- Create a project in Visual Studio with the **Win32 Console Application** template.
- Don't forget to check **Empty Project**.
- On the pool computers:
  Create the project in the folder **R:**`Visual Studio…`
  rather than the share `//retina…`
  (this avoids mind-boggling warnings when starting the program).
- Add a source-code file with **Add→New** Item in the **Source Files**' context menu.
- Build the project with **Build Solution**.
- Run the program with **Start without Debugging**.

A closer look at “Hello World”



What means std:: ?

- Consider:

```
std::cout << "Hello World!" << std::endl;
```

- **std** is a namespace (for the whole C++ standard library).
- Namespaces group objects (functions, classes etc.) for avoiding name clashes.
- **::** selects an object from the namespace.
- We can avoid the need to write `std::` with the **using** directive:

```
using namespace std;
cout << "Hello World!" << endl;
```

“Hello World” with using directive

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Using Variables

```
#include <iostream>
using namespace std;
int main() {
    int x;
    x = 7;
    cout << x << " times 2 is " << x*2 << endl;
    return 0;
}
```

- We declare a variable **x** of type **int**. A variable stores a value (of a particular type, here **int**).
- We assign the value 7 to **x** ("**x** gets the value 7").
- We print the value of x, followed by the string **" times 2 is "**, followed by the value of the expression **x*2**.

Variables

- Variables store values for later use.
- Each variable is identified by a variable name and has a type.
- A variable must be declared before it can be used. Such a declaration has the following form:

```
type name;
```

- type can be any C++ type (e.g. `int`, `bool`, `std::string`).
- A variable name
  - must start with a letter, followed by letters, digits, or underscores;
  - C++ keywords (e.g. `int`, `return`) are not allowed;
  - names are case-sensitive:
    result, Result and RESULT are three different names!
- Example:

```
int x;
```

Assignments

- For storing a value in a variable, you have to assign the value to the variable:

```
x = 7;
```

  From this point on, the variable will have the value 7.

- General form of an assignment:

```
variable = expression;
```

- On the right hand side of the assignment can be a compound expression, e.g.

```
x = (7 + 2) * 3;
```

  The value of the expression is calculated and assigned to x.

- Caution:

  = is the assignment operator, never an equality test!

Printing Data

- Printing text to the console window is done using the std::cout object.
- Everything that shall be printed is send to std::cout using the output operator << .
  - Write text as string literal `" times 2 is "` .
  - Variables and expressions are evaluated, and their value is printed.
- Example:

```
std::cout << x << " times 2 is " << x * 2;
```

prints (if x has value 7):

```
7 times 2 is 14
```

Printing Data

- You can end a line with `std::endl` ("end of line").
  (We assume `using namespace std;` is used.)

```
cout << "This is the first line."  << endl;
cout << "And this the second one." << endl;
```

- This can also be combined:

```
cout << "This is the first line."  << endl
     << "And this the second one." << endl;
```

Reading Data

- Reading data from the console is done using the input object std::cin and the input operator >> .

Reading Data

```
#include <iostream>
using namespace std;
int main() {
    int x;
    cout << " Enter a number: ";
    cin >> x;
    cout << x << " squared is " << x*x << endl;
    return 0;
}
```

- We declare a variable **x** of type **int**.
- We print a message.
- We read a number from the console and store it in x.
- We print something useful.

Operations on Integers

- `int`s can be read with **`std::cin`** and printed with **`std::cout`**.
- Arithmetic operators:
  - Addition: `+`
  - Subtraction: `-`
  - Multiplication: `*`
  - Division: `/`
  - Modulo (remainder after division): `%`
- As usual: Multiplication, division, modulus precede over addition and subtraction
- Use parentheses to explicitly specify precedence.
- Integer division is always rounding down:
  `19 / 10` is 1

Conditional Statements

- The **if** statement allows the program to make decisions.
- That means that some part of the program is executed conditionally, depending on some boolean expression.
- The general form of an **if** statement is:

```
if ( condition )
    statement;
```

- *statement* is executed if *condition* is true; otherwise, *statement* is not executed.
- Example:

```
if (x > 0)
    cout << x << " is positive." << endl;
```

A typical source of errors…

- if refers only to the immediately following statement!
- What happens here?

```
int money;
bool inDebt;
/* money is assigned some value here */
if (money < 0)
   inDebt = true;
   cout << " Your account is in debt!" << endl;
```

- The message “Your account is in debt!” is printed in any case!

Compound Statements

- How to solve this problem? Use a compound statement!

```
int money;
bool inDebt;
/* money is assigned some value here */
if (money < 0) {
   inDebt = true;
   cout << " Your account is in debt!" << endl;
}
```

- Multiple statements can be grouped with curly braces: { }
- We say that we have to make a new block.

Relational Operators

- The following operators can be used to form conditions:
  - Less than or equal: `<=`
  - Less than: `<`
  - Greater than or equal to: `>=`
  - Greater than: `>`
  - Equal: `==`
  - Not equal: `!=`
- You can compare variables with variables, or even expressions with expressions.

```
if (2*a+b > c*c-4)
    ...
```

- **Beware of the difference between equality (==) and assignment (=) !**

`if-else` Statements

- The extended form of the if statement is:

```
if ( condition )
    statement1;
else
    statement2;
```

- *statement1* is executed if condition is true, *statement2* is executed if *condition* is false.
- Example:

```
if ( (a % 2) == 0 )
    cout << a << " is even." << endl;
else
    cout << a << " is odd." << endl;
```

Dangling `else`

- Typical problem: To which **if** does an **else** belong?

```
if ( a == 1 )
    if ( b == 1 )
        a = 20;
else
    b = 20;
```

```
if ( a == 1 )
    if ( b == 1 )
        a = 20;
    else
        b = 20;
```

- Rule: **else** always belongs to the closest preceding **if**.
- Make clear what you mean using a compound statement:

```
if ( a == 1 ) {
    if ( b == 1 )
        a = 20;
} else
    b = 20;
```

while-Loops

- Loops are used for repeating a statement (or a block) several times. We first consider **while**-loops.
- The general form of a **while** statement is:

```
while ( condition )
    statement;
```

- *statement* is executed again and again as long as *condition* is true.
- *condition* is formed in the same way as for **if** statements.
- If *condition* is already false from the start, *statement* is never executed.

Example: Printing the numbers from 1 to 100

```
#include <iostream>

using namespace std;

int main() {
    int counter = 1;

    while (counter <= 100) {
       cout << counter << endl;
       counter = counter + 1;
    }

    return 0;
}
```

Preparations for next week

- Read about loops
  - do-while-loops
  - for-loops
  - break and continue
- C++-Strings (class std::string)