Institute of Automation

C++ Basics and Applications in technical Systems

Lecture 1 - Fundamental Terms

Institute of Automation
University of Bremen

26th October 2012 / Bremen

WiSe 2012/2013

VAK 01-036

Universität Bremen

Institute of Automation

Overview

Universität Bremen

Supervisors

Institute of Automation

Each participant will be assigned to one of the following supervisors:

Henning Kampe (HK)

- Room M1090
- Mailbox 260
- kampe@iat.uni-bremen.de

Adrian Leu (AL)

- Room N1240
- Mailbox 081
- aleu@iat.uni-bremen.de

Christos Fragkopoulos (CF)

- Room N1310
- Mailbox 272
- Fragkopoulos@iat.uni-bremen.de

Universität Bremen

Lecture schedule

Institute of Automation

Time schedule

HK **26. Oct.** - Introduction / Simple Program / Datatypes ...

HK **02. Nov.** - Flow control / User-Defined Data types ...

CF **09. Nov.** - Simple IO / Functions/ Modular Design ...

CF **16. Nov.** - C++ Pointer

CF **23. Nov.** - Object oriented Programming / Constructors

AL **30. Nov.** - UML / Inheritance / Design principles

AL **07. Dec.** - Namespace / Operators

AL **14. Dec.** - Polymorphism / Template Classes / Exceptions

HK **11. Jan.** - Design pattern examples

Universität Bremen

Important dates

Submission of exercises

1-3 **16. Nov.** - Deadline for submission of Exercise I, 13:00

4-6 **07. Dec.** - Deadline for submission of Exercise II, 13:00

For admission to final exam you need at least 50% of every exercise sheet.

Final project

1-9 **15. Feb.** - Deadline for submission of final project, 13:00

Final exam

1-9 **06. Feb.** - Final exam, 10:00-12:00, H3

Rating and final mark

Institute of Automation

Rating of exercises and final project

- completeness of the delivered solution
- functionality errors
- compliance with programming guidelines

Final mark

50% **final project work**

50% **final exam**

Universität Bremen

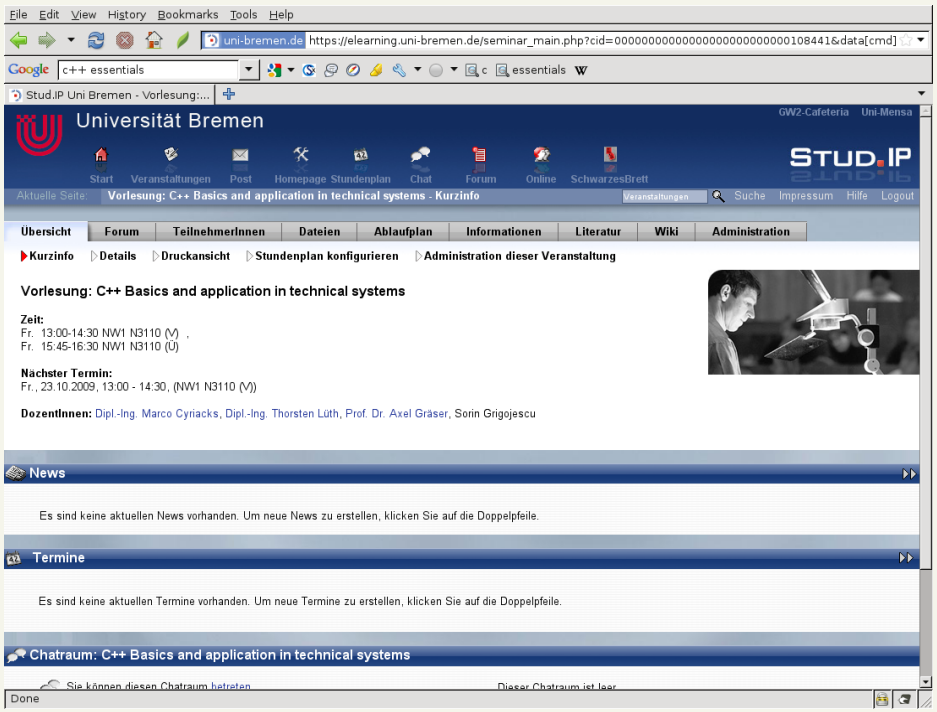
e-Learning

Institute of Automation

For communication purpose we use the e-Learning system `http://www.elearning.uni-bremen.de`.

contents

- lecture notes
- exercises
- additional information



Universität Bremen

Development environment

Institute of Automation

The lecture is based on the following free development tools:

`http://gcc.gnu.org`

`http://qt.nokia.com/products/developer-tools/`

GNU Toolchain

- GNU g++ compiler
- GNU make
- ...

Qt Creator

- Editor
- Debuger
- ...

All tools we use within the lecture are available for at least Linux and Microsoft Windows systems.

Universität Bremen

Development system as Boot-Stick

Institute of Automation

We created a complete development system based on the debian linux distribution. During the lecture you can get the system on USB-Stick.

Download

To create your own boot stick you can download the complete image file from

`http://www2.iat.uni-bremen.de/~C++2008/CppBootStickImage.img`

The image has a size of $\approx 8GB$.

Universität Bremen

C++ literature

Institute of Automation

non-free

- „The C++ Programming Language“ - [Str00, Str09] - in english and german available
- „C++ Einführung und professionelle Programmierung“ - [Bre07]
- „Problem Solving with C++“ - [Sav11]

free PDF version available

- „Thinking C++“ - [Eck01, Eck03]
- „C++ Essentials“ - [Hek05]

Full references on the next slides.

Universität Bremen

Literature I


BREYMANN, Ulrich:
C++: Einführung und professionelle Programmierung.
Hanser Fachbuchverlag, 2007. –
ISBN 978–3446410237


ECKEL, Bruce:
Thinking in C++: Introduction to Standard C++.
http://mindview.net : Prentice Hall, 2001. –
ISBN 978–0139798092


ECKEL, Bruce:
Thinking in C++: Practical Programming.
http://mindview.net : Prentice Hall, 2003. –
ISBN 978–0130353139

Literature II


HEKMAT, Sharam:
C++ Essentials.
http://www.pragsoft.com : PragSoft Corporation, 2005


SAVITCH, Walter:
Problem Solving with C++.
Addison-Wesley, 2011. –
ISBN 978–0132162739


STROUSTRUP, Bjarne:
The C++ Programming Language.
Amsterdam : Addison-Wesley Longman, 2000. –
ISBN 978–0201700732

Literature III

STROUSTRUP, Bjarne:
Die C++-Programmiersprache.
München : Addison-Wesley, 2009. –
ISBN 978–3827328236

Structure of a simple source file

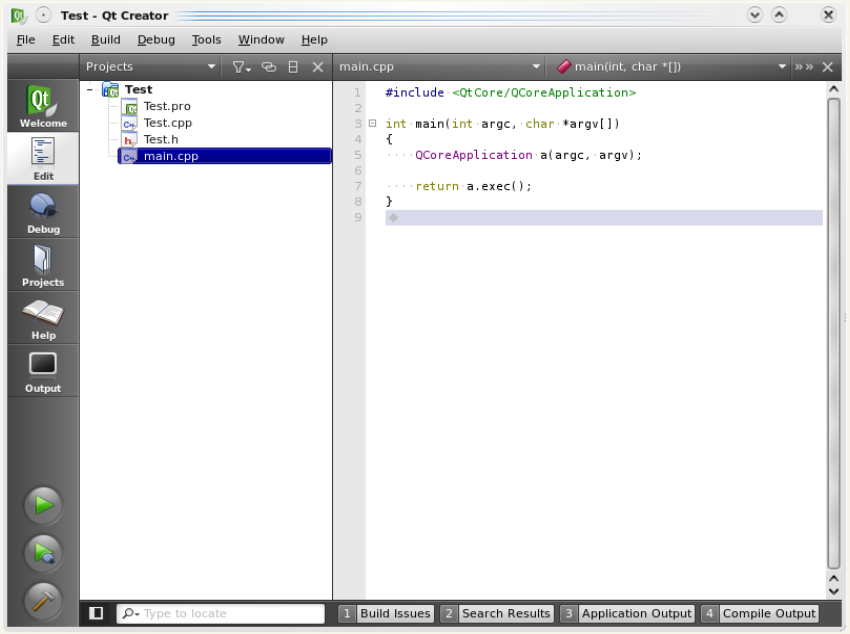
```
1  #include <iostream>                          // Preprocessor instruction

1  using std::cout;                             // Compiler instruction
2  using std::cin;                              // Compiler instruction

1  int main() {                                       // Main function

1    int nSum;                                  // Variable declaration
2    int nA;                                    // Variable declaration
3    int nB;                                    // Variable declaration

1    cout << "Input a and b:";                        // Output message
2    cin >> nA >> nB;                             // Input values
3    nSum = nA + nB;                          // Calculation of sum
4    cout << "Sum = " << nSum;                        // Output result

1    return 0;                          // return value to operating system

1  }
```

Qt Creator

The Qt Creator is an IDE (Integrated Development Environment) which has an editor with direct access to a compiler and linker. It integrates the Qt framework easily into projects and takes care of the makefile.

Features

- Plattform independent
- Free for non-commercial usage
- Automatic makefile generation
- Integrated project manager



Introduction to Qt Creator: New project

Steps to create a new console using project in Qt Creator:

- Menu: File -> New...
- Select “Qt4 Console Application“
- Press “OK“
- Input name of project
- Optional: select folder to create project in
- Press “Next >“
- Optional: select Qt modules to integrate
- Press “Next >“
- Press “Finish“

The created console application has a generated “main.cpp“ file with a default frame for using Qt in the project. Its content is optional and can be replaced.

Introduction to Qt Creator: New class

Steps to create a new class in Qt Creator:

- Menu: File -> New...
- Select “C++ Class“
- Press “OK“
- Input name of class
- Optional: select base class and change file names
- Press “Next >“
- Press “Finish“

Introduction to Qt Creator: Hints (1)

Institute of Automation

The Qt Creator does not support a full terminal in its GUI. Console applications started from the creator display their output in a window of the creator. This window does not support cin. To handle such programs they must be executed in an actual terminal. Also under linux the correct terminal program must be specified in the creator settings.

Universität Bremen

Introduction to Qt Creator: Hints (2)

Institute of Automation

Set program execution to be in terminal in Qt Creator:

- In the left icon bar: press “Projects“
- In the area “Run Settings“: press “Show details“
- Check “Run in terminal“

Set terminal in Qt Creator for linux:

- Menu: Tools -> Options...
- Select “Environment“ -> “General“
- Set text for “Terminal:“ to “/usr/bin/xterm -e“
- Press “OK“

Universität Bremen

Function declaration

```
1  #include <iostream>
2  using std::cout;
3  using std::cin;

1  int Square(int nA);          // Declaration of function (signature)

1  int main() {
2    int nSquare;
3    int nValueA;
4    cout << "Input value A:";
5    cin >> nValueA;

1    nSquare = Square(nValueA);                    // Call to function

1    cout << "Result= " << nSquare;
2    return 0;
3  }

1  int Square(int nA) {                          // Definition of function
2    return nA * nA;
3  }
```

In general

See also the IAT programming guidelines uploaded to the eLearning (`http://www.elearning.uni-bremen.de`).

Naming conventions

- Sequence of symbols (letters, numerals and underscore „_“)
- Upper- and lower-case letters are distinguished (case sensitive)
- Beginning with a letter
- No blanks inside a name
- Keywords like **int**, **main**, **for**, ... are not allowed
- In principle no restriction in length (compiler dependent)

Variable names

See also the IAT programming guidelines uploaded to the eLearning (`http://www.elearning.uni-bremen.de`).

Example

- `int iValue;`
- `const int iVALUE;`
- `std::string sMyString;`
- `bool bSuccess;`
- `std::string fileName;`

Function and method names

See also the IAT programming guidelines uploaded to the eLearning (`http://www.elearning.uni-bremen.de`).

Example

- `void SetData(int iData);`
- `int GetData(void) const;`
- `void PerformComputation(void);`

Expressions

Institute of Automation

Definition

An **expression** consists of one ore more **operators** and one or more **operands**. **Operands** are fixed values or variables. The result of an **expression** is a value. Each **Operand** belongs to a specific data-type.

Example

- **Operators**: `=`, `+`, `-`, `*`, `/`, **...**
- **Operands**: `1`, `2`, **...**, `1230`, `"a string"`, **...**
- `iResult = iValue + 13;`
- `iResult = 24 + 13 + iValue + 16 - 14;`
- `sLibName = sBaseName + ".lib";`

Universität Bremen

Data-types

Institute of Automation

The following data-types will be discussed in this lecture:

simple data-tyes

- Integer, real or complex numbers
- Logical data-type (boolean)
- References

More data-types and user defined data-types are discussed later in one of the following lectures.

Universität Bremen

Integer numbers: Different types

Institute of Automation

short

```
short iValue;
unsigned short iValue;
```

int

```
int iValue;
unsigned int iValue;
```

long

```
long iValue;
unsigned long iValue;
```

Values can be assigend to variables of type `int` using different systems:

Representation of constant numbers

Decimal number: `12345`
Octal number: `0377`
Hexadecimal number: `0xAFFE`

Universität Bremen

Integer numbers: Size

Institute of Automation

Number of bits

`short iValue;`	$\Rightarrow$	16
`int iValue;`	$\Rightarrow$	32 (or 16 depending on compiler)
`long iValue;`	$\Rightarrow$	64 (or 32 depending on compiler)

Example

Range example for 16 bits:

$$-2^{15} \cdots +2^{15} - 1 \quad = \quad -32768 \cdots 32767$$

Universität Bremen

Integer numbers: Precision

Assumption of 16 bits for `short`:

```
1  #include <iostream>                    // Preprocessor instructions
2  using std::cout;

1  int main()
2  {

1    short iA = 50;
2    short iB = 1000;
3    short iC;

1    iC = iA * iB;
2    cout << iC;                  // Output -15536 (instead of 50000)

1    return 0;

1  }
```

Integer numbers: Numeric limits

Institute of Automation

To check numeric limits of integer values you can include the header file `<limits>` and use the following methods:

Example

```
int iIntMin = std::numeric_limits<int>::min();
int iIntMax = std::numeric_limits<int>::max();
```

To check the size in bytes of a specific type (not only integer) use the function `sizeof()`:

Example

```
int iIntBytes = sizeof(int);
int iBoolBytes = sizeof(bool);
```

Universität Bremen

Integer numbers: Operators

Institute of Automation

Selection of arithmetic operators

`+, -, ++, --, *, /, %, *=, -=, ...`

Selection of relational operators

`<, >, <=, >=, ==, !=, ...`

Selection of bit-operators

`<<, >>, &, ^, |, ~, <<=, >>=, &=, ^=, |=, ...`

Example

- `int iValue = 6;`
- `iValue += 3;`
- `iValue++;`
- `--iValue;`

Universität Bremen

Real numbers: Different types

Institute of Automation

Data-types, bits, numer range, precision

Data-type	Bits	Number range	Precision
`float`	32	$\pm 3.4e^{-38} \cdots \pm 3.4e^{38}$	7
`double`	64	$\pm 1.7e^{-308} \cdots \pm 1.7e^{308}$	15
`long double`	80	$\pm 3.4e^{-4932} \cdots \pm 3.4e^{4932}$	19

Example for the number of bits

	`float`	`double`	`long double`
sign	1	1	1
mantissa	23	52	64
sign exponent	1	1	1
exponent	7	10	14
sum	32	64	80

Universität Bremen

Real numbers: Representation of constants

Institute of Automation

Description

- sign (+/−)
- decimal point, no comma!
- `e` or `E` for exponent (optional)
- suffix `f`, `F` or `l`, `L` (optional, without suffix `double` is default)

Example

- `123.123e6f`
- `1.8L`
- `1.8E`
- `0.999`
- `1e-03`

Real numbers: Operators

Institute of Automation

Overview

Operator	Example	Meaning
+	+d	Unary plus
−	−d	Unary minus
+	d + 2	Binary plus
−	d − 5	Binary minus
*	5 * d	Multiplication
/	d / 6	Division
=	d = 3 + k	Assignment
*=	d *= 3	$d = d * 3$
/=	d /= 3	$d = d/3$
+=	d += 3	$d = d + 3$
−=	d −= 3	$d = d - 3$
<	d < f	Smaller than
>	d > f	Bigger than
<=	d <= f	Smaller or equal
>=	d >= f	Bigger or equal
==	d == f	Equal
!=	d != f	Unequal

Bremen

Real numbers: Mathematical helper functions

Institute of Automation

Use the header file `<cmath>` to use mathematical helper functions:

```
1  #include <iostream>                                    // Header files
2  #include <cmath>
3  using std::cout;
4  using std::cin;
```

```
1  int main()
2  {
3    float fNumber;
4    cin >> fNumber;                                      // Input number
5    cout << "Root = " << std::sqrt(fNumber);  // Output square root
6    cout << "Sine = " << std::sin(fNumber);             // Output sine
7    cout << "Norm = " << std::fabs(fNumber);            // Output norm
8    return 0;
9  }
```

Universität Bremen

Character values: Possible contents

Institute of Automation

Possible characters (Literals)

- Letters: A b c D z ...
- Numerals: 1 2 3 ...
- Special characters: ! , ; .

Data-type

```
signed char
unsigned char
char
```

Example

```
char myChar = 'a';
```

Universität Bremen

Character values: Excerpt from ASCII-table

Institute of Automation

American **S**tandard **C**ode for **I**nformation **I**nterchange

Value	Char	C++
32		
33	!	!
34	"	\"
39	’	\’
40	(	(
43	+	+
48	0	0
49	1	1

Value	Char	C++
57	9	9
60	<	<
61	=	=
65	A	A
90	Z	Z
92	\	\\
97	a	a
122	z	z

Universität Bremen

Character values: Example

Institute of Automation

```
1  char cC;
2  int iI = 66;
```

```
1  cC = static_cast<char>(iI);                    // type casting int -> char
```

```
1  cout << cC;                                                  // Output: 'B'
2  cC = '1';                                          // cC = character for 1
3  iI = static_cast<int>(cC);                     // type casting char -> int
4  cout << iI;                                                  // Output: 49
```

```
1  cC = '5';
2  iI = cC - '0';                 // Subtraction (Implicit type casting)
3  cout << iI << endl;                                          // Output: 5
```

Universität Bremen

Character values: Operations

Operator	Example	Interpretation
=	`D = 'A'`	Assignment
<	`D < f`	Smaller than
>	`D > f`	Bigger than
<=	`D <= f`	Smaller or equal
>=	`D >= f`	Bigger or equal
==	`D == f`	Equality
!=	`D != f`	Inequality

Character values: Small exercise

Write two programs performing the following conversions:

First program

Ask the user for an integer value, convert it to the corresponding ASCII character and print the result to the screen.

Second program

Ask the user for an character value, convert it to the corresponding ASCII value and print the result to the screen.

Logical data-type: Values and operations

Institute of Automation

Possible values

Data-type: `bool`

Possible values: `true` and `false`

`true` = 1

`false` = 0

Operations

`!`	⇒	Negation
`&&`	⇒	AND
`||`	⇒	OR
`==`	⇒	Equality
`!=`	⇒	Inequality
`=`	⇒	Assignment

Logical data-type: Example

```
1  bool bVar0;
2  bool bVar1;
3  bool bVar2;
```

```
1  bVar0 = bVar1 = bVar2 = true;
2  cout << (bVar1 && bVar2) << endl;                    // Output: 1
3  bVar0 = !bVar1;
4  cout << bVar0 << endl;                               // Output: 0
5  bVar1 = static_cast<bool>(10);                       // type casting
6                                                       // int -> bool => true
7  bVar2 = static_cast<bool>(0);                        // dito => false
8  cout.setf(ios_base::boolalpha);                      // set textformat
9  cout << bVar1 << endl << bVar2 << endl;              // Output: true false
```

Complex numbers: Different types

Institute of Automation

Data-types, bits, numer range, precision

```
std::complex<float> floatComplex;
std::complex<double> doubleComplex;
std::complex<long double> longDoubleComplex;
```

Header

```
#include <complex>
```

Operations

All arithmetic operators as well as check of the equality and inequality of the real numbers.

Complex numbers: Example

Institute of Automation

```
1  #include <iostream>                                    // Header files
2  #include <complex>
3  using std::cout;
4  using std::endl;
5  using std::complex;
```

```
1  int main()
2  {
3    complex<float> comNum1;                  // comp. num. (0.0+0.0i)
4    complex<float> comNum2(4.1,3.4);         // comp. num. (4.1+3.4i)
5    float fReal;                          // Declaration of real num.
6    comNum1 = comNum2;                 // Example: arithmetic operation
7    comNum1 = comNum2 * 5.0f;                    // Suffix f (=float)
8    fReal = comNum1.real();                   // Determine real part
9    cout << comNum2.imag() << endl;             // Output imag. part
10   comNum1 = conj(comNum2);            // helper function for complex
11   return 0;
12 }
```

Universität Bremen

References

Institute of Automation

Definition

A **Reference** to an object is like an alias name in C++. Changes to a reference of a variable will change the varible itself.

```
1  int iInt1;
2  int iInt2;
3  int &iRef = iInt1;                    // Reference on iInt1
4                                         // (Alias for iInt1)
```

```
1  iInt1 = 100;                              // changes iInt1
2  iRef = 10;                                // changes iInt1
3  iRef = iInt2;                      // effect: iInt1 = iInt2
```

Universität Bremen

Constants

Institute of Automation

Keyword

```
const
```

Example

```
const float fPI = 3.1415926;
const int iSIZE = 1000;
const int iINDEX_SIZE = iSIZE - 1;
```

Constant values should always be declared as `const`!

Validity and visibility: Restrictions

Institute of Automation

Restrictions on variable validity

- Names are only valid after the declaration and within the block `{...}` in which they were defined
- Names remain valid in re-created blocks within the block of the declaration

Restrictions on variable visibility

- Visibility of variables of the same name is reduced, e.g. for the visibility area of the internal variables the outside is invisible

Universität Bremen

Validity and visibility: Example

Institute of Automation

```
1   int nVar = 5;                                          // global variable

1   {                                                        // block start

1     int nVar = 3;                                       // local variable
2                     // global variable is still valid but invisible

1     cout << nVar;                                        // Output => 3
2     cout << ::nVar;                                      // Output => 5

1   }                                                        // block end

1   cout << nVar;                                          // Output => 5
```

Universität Bremen

Validity and visibility: Rule

Institute of Automation

Rule

Avoidance of local variables with names which cover variables in an outside range of validity!

Universität Bremen

Exercise

Institute of Automation

Create a program that prints out the number ranges as well as the size in bytes of the following data types:

- Natural numbers (short, unsigned short; int, unsigned int; long, unsigned long)
- Real numbers (float; double; long double)

Hint 1

Don't use knowledge about the internal data representation directly. Use the function `std::numeric_limits<>::min()` (e.g.: `std::numeric_limits<int>::min()`).

Hint 2

Use the function `sizeof()` like `sizeof(int)`.