Informatics

GianAndrea Müller

May 3, 2018

Contents

1 How to...

1.1 ... use this summary

This summary is an overview of the functionality of C++ in connection with the informatics course for mechanical engineers. It covers the content of the lectures but also contains additional information.
To emphasize the connection to the lecture all chapters containing purely additional information are marked in blue.

1.2 ... correct compilation errors

Read error messages, review basic syntax, look for the additional semicolon.

1.3 ... correct runtime errors

Use a debugger.

1.4 ... approach problems

1. Define your problem.
2. Find your algorithm.
3. Code feature.
4. Compile.
5. `goto` 3.

1.5 ... find more information

- Comprehensive Tutorial
- User friendly documentation
- Extensive technical documentation

2 Positional Notation

2.1 Binary numbers

2.1.1 Floating point numbers

2.2 Hexadecimal numbers

3 Syntax

3.1 Basic program

```
#include <iostream>
//#include "local_header_file.h"

/*
 * Function declarations (and definitions)
 */

int main(int argc, char ** argv)
{
    /*
     * Function calls
     */
    std::cout << "Hello World!" << std::endl;
    return 0;
}

/*
 * Function definitions
 */
```

3.2 Identifiers

A valid identifier, i.e. the name of a variable is:

- an arbitrarily long sequence of digits, underscores, lowercase and uppercase Latin letters.
- not starting with a digit.
- not starting with two or more underscores.
- not starting with an underscore followed by a capital letter.
- not a keyword of cpp.

More information on Identifiers.

3.3 Comments

C++ allows masking code such that it is not interpreted as part of the program. This enables documenting the program. There are different possibilities:

```
// normal comment

/*
multi
line
comment
*/
```

Both versions can be nested:

```
///*comment in a comment*/

/*
cout<<"Hello World!<<endl; //comment in a comment
*/
```

3.4 Data Types

3.4.1 Primitive Types

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void

3.4.2 Type modifiers

There exist a number of type modifiers:

Modifier	Effect
signed	variable interpreted as signed
unsigned	variable interpreted as unsigned
short	half number of allocated bits if possible
long	double number of allocated bits if possible

Based on the primitive types and their modifiers the spectrum of available types can be established. Their sizes differ depending on compiler and environment.

Modifier	Typical Bit Width	Typical Range
char	1byte	-127 to 127
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4byte	-2'147'483'648 to 2'147'483'647
unsigned int	4bytes	0 to 4'294'967'295
signed int	4bytes	-2'147'483'648 to 2'147'483'647
short int	2bytes	-32'768 to 32'767
unsigned short int	2bytes	0 to 65'535
signed short int	2bytes	-32'768 to 32'767
long int	4bytes	-2'147'483'648 to 2'147'483'647
signed long int	4bytes	-2'147'483'648 to 2'147'483'647
unsigned long int	4bytes	0 to 4'294'967'295
float	4bytes	+/- 3.4e +/- 38 (7 digits)
double	8bytes	+/- 1.7e +/- 308 (15 digits)
long double	8bytes	+/- 1.7e +/- 308 (15 digits)

3.4.3 Find type sizes on your system

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

int main() {
    cout<< "Size: char : "<<sizeof(char)<<endl;
    cout<< "Size: int : "<<sizeof(int)<<endl;
    cout<< "Size: short int : "<<sizeof(short int)<<endl;
    cout<< "Size: long int : "<<sizeof(long int)<<endl;
    cout<< "Size: float : "<<sizeof(float)<<endl;
    cout<< "Size: double : "<<sizeof(double)<<endl;

    return 0;
}
```

3.4.4 Type qualifiers

3.4.5 Storage classes

3.4.6 Lvalues and Rvalues

3.4.7 Variable Definition

3.4.8 union

3.4.9 enum

4 Variable Scope

4.1 Local Variables

4.2 Global Variables

5 Literals

5.1 Integer Literals

5.2 Floating-point Literals

5.3 Boolean Literals

5.4 Character Literals

5.5 String Literals

5.6 Defining constants

5.6.1 #define

5.6.2 const

6 Operators

6.1 Arithmetic Operators

6.2 Relational Operators

6.3 Logical Operators

6.4 Bitwise Operators

6.5 Assignment Operators

6.6 Misc Operators

6.7 Operator Precedence and Associativity

P.	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right
2	`a++ a--`	Suffix/postfix increment and decrement	
	`type() type{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`. ->`	Member access	
3	`++a --a`	Prefix increment and decrement	Right-to-left
	`+a -a`	Unary plus and minus	
	`! ~`	Locigal NOT and bitwise NOT	
	`(type)`	C-style cast	
	`*a`	Dereference	
	`&a`	Adress-of	
	`sizeof`	Size-of	
	`new new[]`	Dynamic memory allocation	
	`delete delete[]`	Dynamic memory deallocation	
4	`.* ->*`	Pointer-to-member	Left-to-right
5	`a*b a/b a%b`	Multiplication, division, remainder	
6	`a+b a-b`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`<=>`	Three-way comparison operator	
9	`< <= > >=`	Relational operators	
10	`== !=`	Relational operators	
11	`&`	Bitwise AND	
12	`^`	Bitwise XOR	
13	`|`	Bitwise OR	
14	`&&`	Logical AND	
15	`||`	Logical OR	
16	`a?b:c`	Ternary conditional	Right-to-left
	`throw`	throw operator	
	`=`	Direct assignment	
	`+= -=`	Compound assignments	
	`*= /= %=`		
	`<<= >>=`		
	`&= ^= |=`		
17	`,`	Comma	Left-to-right

6.7.1 How to use this table

```
1 cout<<a&&b;     //(cout<<a)&&b;
2
3 *p++            //*(p++);
4
5 a = b = c = d; //a = (b =(c = d)));
6
7 a + b - c;      //(a + b) - c;
8
9 delete +++*p;   //delete(++(*p))
```

1. By its precedence `<<` is evaluated before `&&`.
2. By its precedence `++` is evaluated before `*`.
3. Operators with the same precedence are evaluated based on their associativity. For right-to-left associative operators as `=`, the evaluation proceeds from right to left.

   Thus the assignments made in line 5 are in the order of their execution: `c = d;` which returns a reference to `c`, `b = c;` which returns a reference to `b` and `a = b;`.
4. Operators with the same precedence are evaluated based on their associativity. For left-to-right associative operators as `+` and `-` the evaluation proceeds from left to right.
5. `++()`, `*()` and `delete` have the same precedence, and are thus evaluated based on their associativity, which is right-to-left. Therefore `++()` is evaluated after `*()` and `delete` is evaluated last.

7 Loop Types

7.1 while

7.2 for

7.3 do...while

7.4 Loop Control Statements

8 Conditional Statements

8.1 if

8.2 if...else

8.3 switch

8.4 ? : Operator

9 Functions

9.1 Structure

9.2 Declaration and Definition

9.3 Calling a function

9.4 Function Arguments

9.4.1 Call by Value

9.4.2 Call by Reference

9.4.3 Default Values for Parameters

9.5 Recursion

9.6 Inline Functions

10 Arrays

11 vector

12 Strings

12.1 C-Style Character String

12.2 string

13 Pointers

14 References

15 Input/Output

16 struct

17 class

17.1 Class Members

17.2 Class Access Modifiers

17.3 Constructor and Destructor

17.4 Copy Constructor

17.5 friend

17.6 this

17.7 Static Members

18 Inheritance

18.1 Access Control and Inheritance

19 Overloading

19.1 Function overloading

19.2 Operator overloading

19.2.1 Overloadable operators

20 Polymorphism

21 Dynamic Memory

22 Namespaces

23 Templates

24 Preprocessor

25 Signal Handling

26 Standard Template Library

27 Libraries

27.1 iostream

27.2 math

27.3 ctime