UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

INTRODUCTION TO MODERN C++

LECTURE 2

Rémi Géraud

February 4, 2016

École Normale Supérieure de Paris

REMARK ON SETUP

If you had trouble installing VirtualBox or Ubuntu you can:

- Go to `http://cpp.sh` to test code (tick C++14 and all warnings)
- Ask for help by sending an mail
- Get a pre-installed VirtualBox image on a USB stick

Lecture 2
Types, Statements, Arithmetic

TABLE OF CONTENTS

TYPES, BUILT-INS, REPRESENTATION

WHAT IS A TYPE?

WHAT IS A TYPE?

A 27 

WHAT IS A TYPE?

A **type** is a label that we attach to data.

Examples:

- 27 is a **number**, more precisely a **positive integer**
- A is a **letter**, more precisely a **capital letter**

What are types useful for? I

Types tell us how to deal with **operations**.

Examples:

- $27 + 27$
- A + A
- cabbage + carrot, A + 3, A * A

Note: In C++, **you can't mix different types**.

This is called **type safety**.

Cabbage with cabbage, carrots with carrots.

What are types useful for? II

Types tell us how to **interpret data**.

Examples:

- 1000001 read as a number is 65
- 1000001 read as a letter is A
- 1000001 read as a carrot is ?

BUILT-IN TYPES

C++ comes equipped with several default types known as built-ins:

- `int` for "integer": 45, 8088, -78
- `bool` for "Boolean": `true`, `false`
- `float` for "floating-point number": 3.14159, -24.3
- `char` for "character": 'A', '\n', '0'
- `void` for the empty type (no possible value).

That is the complete list.

DECLARATIONS AND STATIC TYPING

C++ is **statically typed**, which means that:

- You **must** explicitly provide the type of everything (**declaration**)
- You **can't change it** after.

Example declaration: `int myinteger;`

Note: identifiers must start with a letter, and contain only letters, digits, and underscores

```
my1, 1dt, my-variable, HelloWorld, lol_123, float
```

INTEGER AND FLOAT ARITHMETIC

INTEGER AND FLOAT ASSIGNMENT

Let's start with numbers. First, you must declare variables:

```
int myinteger;
float myfloat;
int a, b, c, d, e, f;
```

Assignment is done with the equal sign (**statement**):

```
myinteger = 443;
myfloat = 23.4;
myfloat = 1e14;
```

Reminder: You can use this to check

```
std::cout << myinteger << std::endl;
```

BASIC OPERATIONS I

Basic operations on `int` or `float`:

- Addition with +
- Subtraction with -
- Multiplication with *
- Division with / Caution!

USS Yorktown (DDG-48/CG-48) — death by division

BASIC OPERATIONS II

What does this code do?

```
#include <iostream>

int main() {
  int myint;
  std::cout << "Enter a number and press Enter: ";
  std::cin >> myint;
  std::cout << myint << std::endl
            << myint*2 << std::endl
            << myint*myint << std::endl;
}
```

Test it with some values: 27, -10, 16777215, 16777216

The previous example illustrates the following fundamental fact:

$$\texttt{int} \neq \mathbb{Z}, \mathbb{N}$$
$$\texttt{float} \neq \mathbb{R}$$

We will see more about that in a subsequent lecture.

BASIC OPERATIONS IV

Just a remark: What happens if I do this?

```
int x = 123.999;
```

BASIC OPERATIONS IV

Just a remark: What happens if I do this?

```
int x = 123.999;
```

Answer: I will get 123 and maybe a warning.

This is called **implicit conversion** and it's **bad practice**.
Try `int x = 1e89;` and see what happens.

Take away: Respect type safety

If you should convert, do it explicitly

BOOLEAN AND BINARY OPERATIONS

BOOLEAN ALGEBRA I

A Boolean algebra encodes classical propositional logic:

```
AND, OR, NOT, XOR, ...
```

For instance, let P be any proposition, then:

- NOT (NOT P) = P
- P AND (NOT P) = False
- P OR (NOT P) = True

BOOLEAN ALGEBRA II

The Boolean operations are expressed in C++ with the following symbols:

- AND with &&
- OR with ||
- NOT with !

Example: “NOT (P AND Q)” is written in C++ as follows:

```
!(P && Q)
```

Other example: `((x == 4) || (x < 1))`

BOOLEAN ALGEBRA III

What does this code do?

```
#include <iostream>

int main() {
  int  age;
  bool adult;
  std::cout << "How old are you? ";
  std::cin >> age;
  adult = (age >= 18);

  if (!adult) {
    std::cout << "Try again in "
              << 18 - age
              << " years!"
              << std::endl;
  }
}
```

BINARY OPERATIONS

The binary operations act like the Boolean operations, but bitwise.
Let:

```
x = 65 = 1000001
y = 42 = 0101010
```

Then

- `x & y = 00000000 = 0` (binary AND)
- `x | y = 01101011 = 107` (binary OR)
- `x ∧ y = 01101011 = 107` (binary XOR)

A WORD OF CAUTION

Do not be confused!

- `=` (assignment) vs. `==` (equality test)
- `&` (binary AND) vs. `&&` (boolean AND)
- `|` (binary OR) vs. `||` (boolean OR)

The first one in particular is deadly.

MORE TYPES, HIGHER-ORDER TYPES

More types, higher-order types

Not going into too many details now, we will meet other types:

- Standard library types, e.g. `std::string`
- Dependent types, e.g. `std::vector<int>`
- Higher-order types, e.g. types of functions

Remark : Type theory is a very active research area.

SPECIAL FUNCTIONS

SPECIAL FUNCTIONS

Addition and multiplications are nice, but what about $\cos(\pi/27)$?

```
#include <cmath>
```

Then you can just use

```
float myvalue = cos(M_PI/27);
```

More interesting computations in the homework (Black-Scholes option pricing).

BRAIN TEASER: UNDEFINED BEHAVIOUR

Question: What happens with this code and why?

```
#include <iostream>

int main() {
  int myint;
  float myfloat;

  std::cout << myint << std::endl
            << myfloat << std::endl;
}
```

Questions?

Lab Session

Math, arrays, vectors, and the Quake 3 trick