UNIVERSITÉ PARIS 1
PANTHÉON SORBONNE

INTRODUCTION TO MODERN C++

LECTURE 3

Rémi Géraud

February 11, 2016

École Normale Supérieure de Paris

A SHORT REMINDER

What you learned so far:

- Don’t make mistakes (semicolons, spaces, etc.)
- Don’t use too big or too small numbers
- Don’t mix types (type safety)

A SHORT REMINDER

What you learned so far:

- How to write, compile and run simple programs

```
int main() {
    // Your code
}
```

- How do translate complicated math as C++ commands

If you *don't* know how to do that, catch up!

Lecture 3
Control flow and Algorithms

TABLE OF CONTENTS

WHAT IS CONTROL FLOW?

What is control flow?

Computers are good as simple, repetitive tasks.

We already saw how to ask for simple tasks.

Control flow instructions enable us to make them repetitive.

What is control flow?

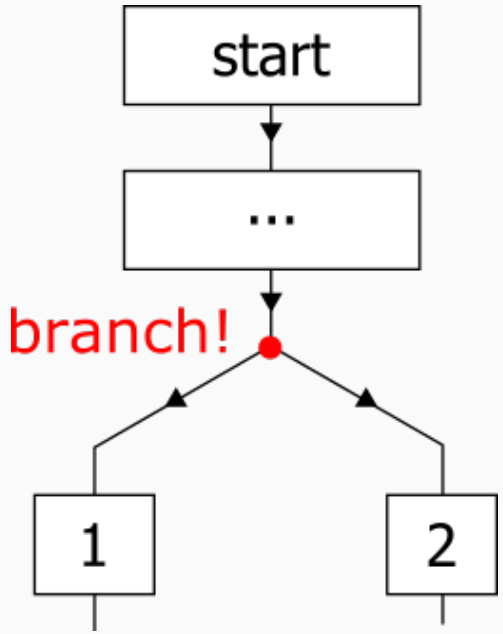
Control flow instructions are essentially of two kinds:

1. **Repeat** (or “loop”)
2. **Branch** (or “conditionals”)

Together with simple instructions, this unleashes the full power of computation.

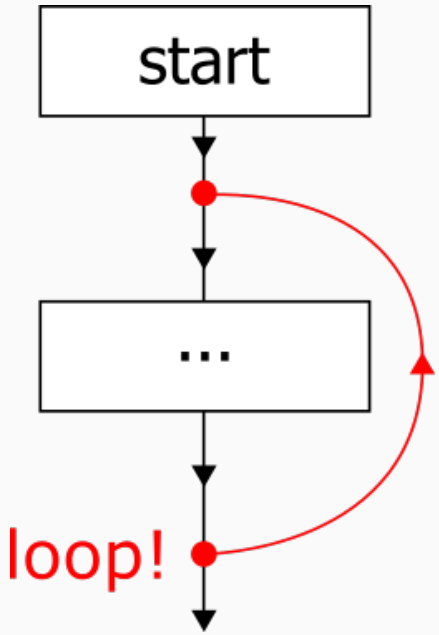
WHAT IS CONTROL FLOW?

A geometric interpretation of branching:



WHAT IS CONTROL FLOW?

A geometric interpretation of looping:



What is control flow?

With control flow instructions C++ is a Turing-complete language.

Anything that can be computed,
can in theory be computed with C++

(Church-Turing Hypothesis)

BRANCHING

BRANCHING: CONDITIONAL STATEMENTS

The simplest form of branching is provided by the `if` statement :

```
if (boolean) {
    // Do something
}
```

Where **`boolean`** is a Boolean value.

BRANCHING: CONDITIONAL STATEMENTS

A simple example :

```
int x;
std::cin >> x;
if (x > 1000) {
    std::cout << "x is larger than 1000" << std::endl;
}
std::cout "The number is " << x << std::endl;
```

BRANCHING: CONDITIONAL STATEMENTS

We can create additional branches;

```
if (boolean1) {
    // Do something
}
else if (boolean2) {
    // Do something
}
else {
    // Do some other thing
}
```

Where `boolean1` and `boolean2` are Boolean values.

BRANCHING: SWITCH/CASE STATEMENTS

There exists a variant, which is trickier to use, but equivalent:

```
switch (variable) {
    case (value1):
        // Do something
        break;
    case (value2):
        // Do something
        break;
    default:
        // Do some other thing
}
```

What do the `break` statements mean? What happens if we remove them?

LOOPING

Loop statements: while, do-while

A `while` statement repeats the execution of some code :

```
while (boolean) {
    // Do something
}
```

If `boolean` is false (or become false), the repetition stops.

Loop statements: while, do-while

A simple example

```
int i = 0;
while (i < 100) {
    std::cout << "i = " << i << std::endl;
    i = i + 1;
}
std::cout << "Finished." << std::endl;
```

This prints all numbers 0, 1, ..., 99.

LOOP STATEMENTS: WHILE, DO-WHILE, FOR

Strictly equivalent is the `do-while` construction:

```
int i = 0;

do {
    std::cout << "i = " << i << std::endl;
    i = i + 1;
} while (i < 100);

std::cout << "Finished." << std::endl;
```

Remark: Don't forget the semi-colon!

If we know in advance the number of repetitions, then it is often more practical to use a `for` loop:

```
for (initial ; condition ; iteration ) {
    // Do something
}
```

An example will make things clearer.

LOOP STATEMENTS: WHILE, DO-WHILE, FOR

Example:

```
int i;

for (i = 3 ; i < 20 ; i = i + 4 ) {
    std::cout << i << std::endl;
}
```

This program prints : 3, 7, 11, 15, 19.

Loop statements: while, do-while, for

Note: Very useful special case:

```
for (int i = 0 ; i < 10 ; ++i) {
    std::cout << i << std::endl;
}
```

This program prints : 0, ..., 9.
The statement `++i` is shorthand for `i = i + 1`.

LOOP STATEMENTS: WHILE, DO-WHILE, FOR

Note: Variant for `std::vector`, `std::set` and `std::array`:

```
std::vector<int> x = {3, 1, 4, 1, 5, 9, 2, 6}

for (int xi : x) {
    std::cout << i*i << std::endl;
}
```

This program prints : 9, 1, 16, 1, 25, 81, 4, 36.

ALGORITHMS

You now know enough to start implementing actual algorithms.

We will start with a silly but typical example (more in the lab) :

The Factorial!

ALGORITHMS: THE FACTORIAL

Reminder: If n is a positive integer, then its *factorial* is denoted $n!$ and is equal to

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

For example, $4! = 24$.

ALGORITHMS: THE FACTORIAL

In C++ this can be easily implemented as follows:

```
#include <iostream>

int main() {
    int n;
    int result = 1;
    std::cin >> n ;
    for (int i = 2 ; i < n ; ++i) {
        result = result * i;
    }
    std::cout << n << "! = " << result << std::endl;
}
```

Note: You can't compute the factorial of large numbers, why?

Note 2: This program doesn't check whether $n > 0$

Note 3: What is the value of 0! ?

ALGORITHMS: COLLATZ SEQUENCE

Challenge: Implement the following algorithm in C++.

1. Input a positive integer n
2. If n is odd then multiply it by three and add one
3. If n is even then divide it by two
4. Repeat steps 2 and 3 until $n = 1$.

Remark: No one knows if that algorithm always finishes.

ALGORITHMS: STANDARD LIBRARY

Teaser: Many useful algorithms are already implemented for you in the C++ standard library.

In the lab, we'll see how to use e.g. `std::sort` for fast sorting.

We don't need to reinvent the wheel.

Theorem (Turing)

It cannot be determined automatically whether a generic program terminates or not.

Proof.

Assume $T(P)$ returns 0 if P doesn't terminate, and enters a perpetual loop otherwise. Consider $T(T)$. □

Corollary

Correctness must usually be checked by hand!

Questions?

Lab Session
Sorting, Numerical integration, and Angry Birds