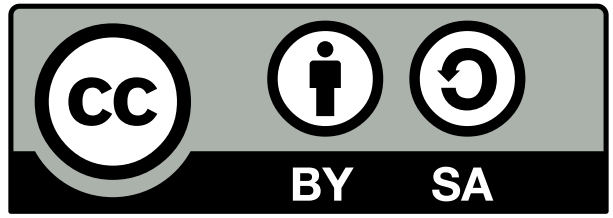
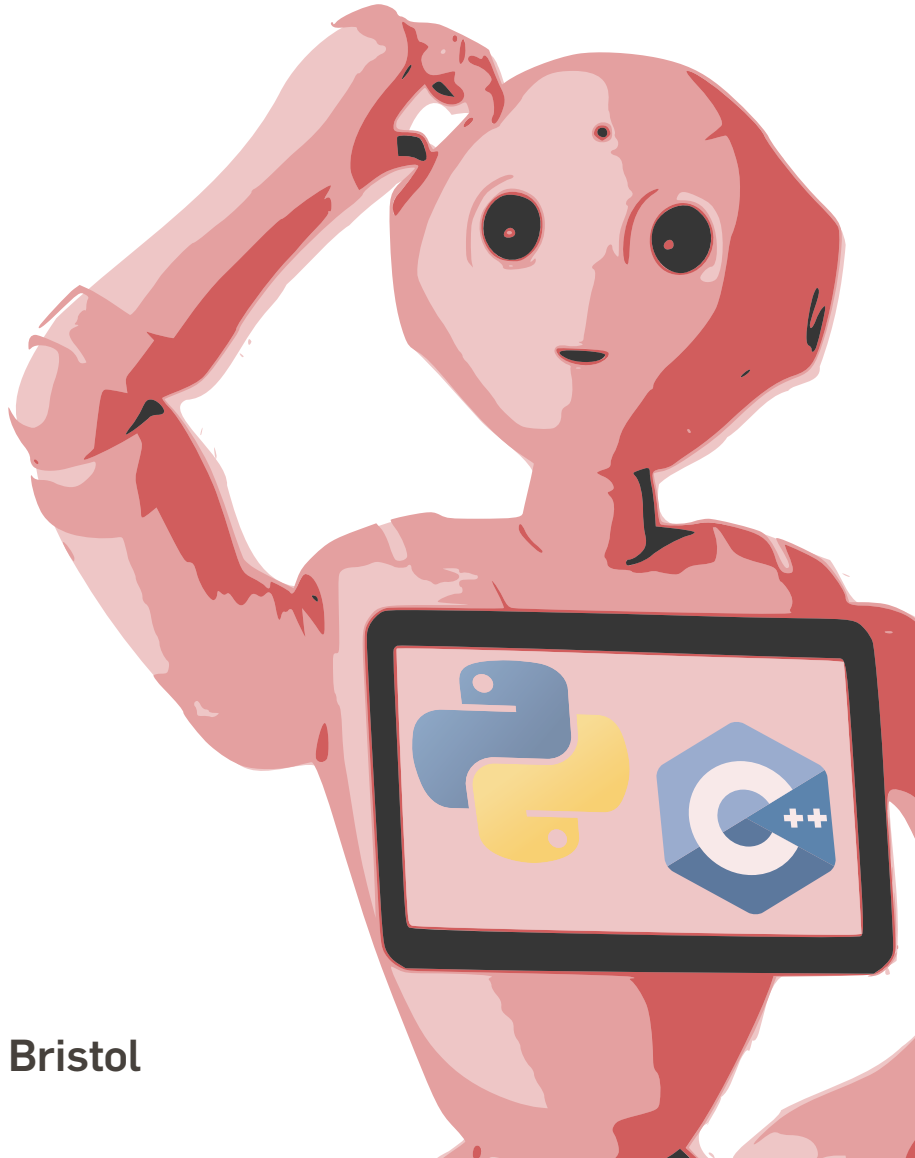


You can download the sources of this presentation here:

github.com/severin-lemaignan/lecture-intro-programming-for-robotics

Python & C++

FARSCOPE workshops



Séverin Lemaignan

Bristol Robotics Lab
University of the West of England/University of Bristol

FIRST OF THREE WORKSHOPS

- Introduction to programming **in Python and C++**
- Software engineering (including things like **git**)
- **ROS**

FIRST OF THREE WORKSHOPS

- Introduction to programming **in Python and C++**
- Software engineering (including things like **git**)
- **ROS**
- At the end: confident computer scientists for robotics
- Mostly hands-on! not lectures

WHERE DO WE START?

Go to www.menti.com
and use the code 58 21 00 9

LET’S GET STARTED!

Head to `robomaze.skadge.org`

CURL AND THE TERMINAL

Fire up a terminal window, and try to create a robot:

```
$ curl 'https://robomaze.skadge.org/move/Wall-E/E'
```

PYTHON

FIRST TASK

Clone the `robomaze` project and run one simple example:

```
$ git clone https://github.com/severin-lemaignan/robomaze.git
$ cd robomaze/scripts
$ python random_walk.py Wall-E
```

Then, modify the script to make sure your robot does not die.

SECOND TASK

Write a clever algorithm to get to the end of the maze!

- Breadth first
- Depth first
- Grassfire
- Dijkstra's shortest path
- A*

GRASSFIRE ALGORITHM

```
1   import queue
2
3   Q = queue.Queue()
4
5   def grassfire(M, goal):
6       Q.push(goal)
7       M[goal] = 0 # set goal value to 0
8
9       while not Q.empty(): # loop until map filled
10          a = Q.pop()
11
12          for n in neighbours(a):
13              if not n in M and not is_obstacle(n):
14                  Q.push(n)
15                  M[n] = M[a] + 1
16      return M
```

Q is a queue data structure, a first-in first-out (fifo) list. The queue keeps track of which locations on the map still need to be visited.

SEARCHING GRAPHS - A*

```
1   def astar(start, end):
2
3     cost_to = {} # maps nodes to distance to 'start_node'
4     cost_to[start] = 0
5     come_from = {} # needed to reconstruct shortest path
6
7     nodes_to_visit = [(start,0)] # (node, 'total' cost)
8     visited_nodes = []
9
10    while nodes_to_visit:
11      node,_ = pop_best_node(nodes_to_visit)
12
13      for neighbour in neighbours(node):
14        if cost_to[node] + 1  < cost_to[neighbour]: # new shorter path to v!
15          cost_to[neighbour] = cost_to[node] + 1
16          nodes_to_visit.append((neighbour, cost_to[neighbour] + \
17                                 heuristic(neighbour, goal)))
18          come_from[neighbour] = node
19
20    return come_from
```

C++

COMPILING CODE IN C++

```cpp
/*
 * "Hello, World!": A classic.
 */

#include <iostream>

using namespace std;

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

COMPILING CODE IN C++

```cpp
/*
 * "Hello, World!": A classic.
 */

#include <iostream>

using namespace std;

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

```
$ g++ hello.cpp -ohello
```

COMPILING CODE IN C++

```
/*
 * "Hello, World!": A classic.
 */

#include <iostream>

using namespace std;

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

```
$ g++ hello.cpp -ohello
```

```
$ ./hello
Hello, World!
```

COMPILING CODE IN C++: THE MAIN STAGES

1. Pre-processing
2. Compilation
3. Assembly
4. Linking

These four steps are transparently performed one after the other by your favourite compiler.

COMPILING CODE IN C++: PRE-PROCESSING

```
/*
 * "Hello, World!": A classic.
 */

#include <iostream>

using namespace std;

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

Pre-processor *directives* start with #

→ `#include <iostream>` is replaced by the content of that file.

COMPILING CODE IN C++: COMPILATION

```
$ g++ -S hello.cpp
```

```
main:
.LFB1493:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rsi
        leaq    _ZSt4cout(%rip), %rdi
        call    _ZStlsISt11char_traitsIcEERSt13basic_ostrea
        movq    %rax, %rdx
        movq    _ZSt4endlIcSt11char_traitsIcEERSt13basic_os
        movq    %rax, %rsi
        movq    %rdx, %rdi
        call    _ZNSolsEPFRSoS_E@PLT
```

COMPILING CODE IN C++: ASSEMBLY

```
$ g++ -s hello.cpp
$ hexdump a.out
0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000010 0003 003e 0001 0000 07b0 0000 0000 0000
0000020 0040 0000 0000 0000 1128 0000 0000 0000
0000030 0000 0000 0040 0038 0009 0040 001b 001a
0000040 0006 0000 0005 0000 0040 0000 0000 0000
0000050 0040 0000 0000 0000 0040 0000 0000 0000
0000060 01f8 0000 0000 0000 01f8 0000 0000 0000
0000070 0008 0000 0000 0000 0003 0000 0004 0000
0000080 0238 0000 0000 0000 0238 0000 0000 0000
0000090 0238 0000 0000 0000 001c 0000 0000 0000
00000a0 001c 0000 0000 0000 0001 0000 0000 0000
00000b0 0001 0000 0005 0000 0000 0000 0000 0000
00000c0 0000 0000 0000 0000 0000 0000 0000 0000
00000d0 0b78 0000 0000 0000 0b78 0000 0000 0000
00000e0 0000 0020 0000 0000 0001 0000 0006 0000
...
```

COMPILING CODE IN C++: LINKING

The linker copies (and re-arrange) the machine code of the static dependencies (***static libraries***) into the executable.

That's what the `-l` flag is used for:

```
$ g++ cool_app.cpp -ocool_app -lcv_core -lcv_highgui -lcv_videoproc
```

(more about libraries next time)

COMPILED VS NOT COMPILED

Multiple **execution models**:

- Compiled languages (eg C, C++, Ada...)

COMPILED VS NOT COMPILED

Multiple **execution models**:

- Compiled languages (eg C, C++, Ada...)
- Interpreted languages (eg: ...?)

COMPILED VS NOT COMPILED

Multiple **execution models**:

- Compiled languages (eg C, C++, Ada...)
- Interpreted languages (eg: ...?)
- $\rightarrow$ most 'interpreted languages' are actually 'JITed'

Just-In-Time compilation: the interpreter generates an efficient **intermediate representation** (commonly called **bytecode**) that is executed (and often stored).

Very common execution model: Python, C#, Javascript...

BACK TO ROBOMAZE AND C++

Let's compile a first example:

```
$ cd robomaze/src
$ g++ request.cpp -orequest
```

EASY! MISSING LIBRARY!

```
request.cpp:4:10: fatal error: cpprest/http_client.h: No such file or directory
    4 | #include <cpprest/http_client.h>
      |          ^~~~~~~~~~~~~~~~~~~~~~~
compilation terminated.
```

EASY! MISSING LIBRARY!

```
request.cpp:4:10: fatal error: cpprest/http_client.h: No such file or directory
    4 | #include <cpprest/http_client.h>
      |          ^~~~~~~~~~~~~~~~~~~~~~~
compilation terminated.
```

```
$ sudo apt install libcpprest-dev
$ g++ request.cpp -orequest
```

```
ttp6client11http_client7requestERKNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEESA_RKN4pplx18cancellation_tokenE]+0xf9):
p::client::http_client::request(web::http::http_request, pplx::cancellation_token const&)'
/tmp/cc1OwGQX.o: In function `web::uri_builder& web::uri_builder::append_query<std::__cxx11::basic_string<char, std::char_trait
>(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, std::__cxx11::basic_string<char, std:
or<char> > const&, bool)':
controller.cpp:(.text._ZN3web11uri_builder12append_queryINSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEEERS0_RKS7_RKT_b[_
yINSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEEERS0_RKS7_RKT_b]+0x3d): undefined reference to `web::uri_builder::append
:basic_string<char, std::char_traits<char>, std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits<ch
t&)'
controller.cpp:(.text._ZN3web11uri_builder12append_queryINSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEEERS0_RKS7_RKT_b[_
yINSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEEERS0_RKS7_RKT_b]+0x61): undefined reference to `web::uri_builder::append
11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits
onst&)'
/tmp/cc1OwGQX.o: In function `pplx::details::_ResultHolder<web::json::value>::Get()':
controller.cpp:(.text._ZN4pplx7details13_ResultHolderIN3web4json5valueEE3GetEv[_ZN4pplx7details13_ResultHolderIN3web4json5value
ence to `web::json::value::value(web::json::value const&)'
/tmp/cc1OwGQX.o: In function `void __gnu_cxx::new_allocator<web::http::details::_http_request>::construct<web::http::details::_
c_string<char, std::char_traits<char>, std::allocator<char> > >(web::http::details::_http_request*, std::__cxx11::basic_string<
d::allocator<char> >&&)':
controller.cpp:(.text._ZN9__gnu_cxx13new_allocatorIN3web4http7details13_http_requestEE9constructIS4_JNSt7__cxx1112basic_stringI
_DpOT0_[_ZN9__gnu_cxx13new_allocatorIN3web4http7details13_http_requestEE9constructIS4_JNSt7__cxx1112basic_stringIcSt11char_trai
: undefined reference to `web::http::details::_http_request::_http_request(std::__cxx11::basic_string<char, std::char_traits<ch
/tmp/cc1OwGQX.o: In function `pplx::details::_ResultHolder<web::json::value>::_ResultHolder()':
controller.cpp:(.text._ZN4pplx7details13_ResultHolderIN3web4json5valueEEC2Ev[_ZN4pplx7details13_ResultHolderIN3web4json5valueEE
 to `web::json::value::value()'
/tmp/cc1OwGQX.o: In function `pplx::details::_ResultHolder<web::json::value>::Set(web::json::value const&)':
controller.cpp:(.text._ZN4pplx7details13_ResultHolderIN3web4json5valueEE3SetERKS4_[_ZN4pplx7details13_ResultHolderIN3web4json5v
```

```
s-lemaignan@escher:~/data/teaching/brl-msc-intro-programming/online-robots/src (master)$ g++ controller.cpp -ocontroller
/tmp/cc1OwGQX.o: In function `main':
controller.cpp:(.text+0x289): undefined reference to `web::uri::uri(char const*)'
controller.cpp:(.text+0x2a2): undefined reference to `web::http::client::http_client::http_client(web::uri const&)'
controller.cpp:(.text+0x2c7): undefined reference to `web::uri::uri(char const*)'
controller.cpp:(.text+0x4eb): undefined reference to `web::uri_builder::to_string[abi:cxx11]() const'
controller.cpp:(.text+0x549): undefined reference to `web::uri_builder::to_string[abi:cxx11]() const'
controller.cpp:(.text+0x572): undefined reference to `web::http::methods::GET[abi:cxx11]'
controller.cpp:(.text+0x681): undefined reference to `web::json::operator<<(std::ostream&, web::json::value const&)'
controller.cpp:(.text+0x6c0): undefined reference to `web::json::value::operator[](unsigned long)'
controller.cpp:(.text+0x6cb): undefined reference to `web::json::operator<<(std::ostream&, web::json::value const&)'
controller.cpp:(.text+0x74b): undefined reference to `web::http::client::http_client::~http_client()'
controller.cpp:(.text+0x90c): undefined reference to `web::http::client::http_client::~http_client()'
/tmp/cc1OwGQX.o: In function `__static_initialization_and_destruction_0(int, int)':
controller.cpp:(.text+0x979): undefined reference to `boost::system::generic_category()'
controller.cpp:(.text+0x985): undefined reference to `boost::system::generic_category()'
controller.cpp:(.text+0x991): undefined reference to `boost::system::system_category()'
/tmp/cc1OwGQX.o: In function `pplx::details::_CancellationTokenRegistration::_Invoke()':
controller.cpp:(.text._ZN4pplx7details30_CancellationTokenRegistration7_InvokeEv[_ZN4pplx7details30_CancellationTokenRegistratio
ference to `pplx::details::platform::GetCurrentThreadId()'
/tmp/cc1OwGQX.o: In function `pplx::details::_CancellationTokenState::_DeregisterCallback(pplx::details::_CancellationTokenRegis
controller.cpp:(.text._ZN4pplx7details23_CancellationTokenState19_DeregisterCallbackEPNS0_30_CancellationTokenRegistrationE[_ZN4
nState19_DeregisterCallbackEPNS0_30_CancellationTokenRegistrationE]+0x151): undefined reference to `pplx::details::platform::Get
/tmp/cc1OwGQX.o: In function `pplx::details::_TaskCollectionImpl::_RunTask(void (*)(void*), void*, pplx::details::_TaskInliningM
controller.cpp:(.text._ZN4pplx7details19_TaskCollectionImpl8_RunTaskEPFvPvES2_NS0_17_TaskInliningModeE[_ZN4pplx7details19_TaskCo
NS0_17_TaskInliningModeE]+0x40): undefined reference to `pplx::get_ambient_scheduler()'
/tmp/cc1OwGQX.o: In function `pplx::task_options::task_options()':
```

BACK TO ROBOMAZE AND C++

Can you figure out what is missing?

BACK TO ROBOMAZE AND C++

Can you figure out what is missing?

```
#include <iostream>
#include <string>

#include <cpprest/http_client.h>

using namespace std;
using namespace web;                        // Common features like URIs.
using namespace web::http;                  // Common HTTP functionality
using namespace web::http::client;          // HTTP client features
//...
```

BACK TO ROBOMAZE AND C++

Can you figure out what is missing?

```
#include <iostream>
#include <string>

#include <cpprest/http_client.h>

using namespace std;
using namespace web;                        // Common features like URIs.
using namespace web::http;                  // Common HTTP functionality
using namespace web::http::client;          // HTTP client features
//...
```

```
$ sudo apt install libcpprest-dev
$ g++ request.cpp -lcpprest -orequest
```

BACK TO ROBOMAZE AND C++

Can you figure out what is missing?

```
#include <iostream>
#include <string>

#include <cpprest/http_client.h>

using namespace std;
using namespace web;                        // Common features like URIs.
using namespace web::http;                  // Common HTTP functionality
using namespace web::http::client;          // HTTP client features
//...
```

```
$ sudo apt install libcpprest-dev
$ g++ request.cpp -lcpprest -orequest
```

```
$ g++ request.cpp -lcpprest -lboost_system -lcrypto -lpthread -orequest
$ ./request Wall-E S
```

STRUCTURE OF THE C++ PROGRAM

```cpp
#include <iostream>
#include <string>
#include <cpprest/http_client.h>

using namespace std;
using namespace web::http::client;

int main(int argc, char* argv[])
{
    // Create http_client to send the request.
    http_client client(U("https://robomaze.skadge.org/"));

    // Build request URI and start the request.
    uri_builder builder(U("move/" + string(argv[1]) + "/" + string(argv[2])));
    http_response response = client.request(methods::GET, builder.to_string()).get();

    cout << "Received response status code:" << response.status_code() << endl;

    // extract the JSON response
    if (response.status_code() == 200) {
        auto json_response = response.extract_json(true).get();
        cout << json_response << endl;
        cout << "Was move successful? " << json_response[0] << endl;
    }
    else {
        cout << "Error!" << endl;
    }

    return 0;
}
```

THIRD TASK

Path planning in C++, of course!

...hum...

LET'S ORGANISE THE WORK

→ Isolate the A* algorithm from the `main`.

Three templates in `src/`:

- `controller.cpp`: our 'main' (that contains the ***entry point*** of our program)

LET'S ORGANISE THE WORK

→ Isolate the A* algorithm from the `main`.

Three templates in `src/`:

- `controller.cpp`: our 'main' (that contains the ***entry point*** of our program)
- `astar.cpp`: the algorithm itself

LET'S ORGANISE THE WORK

→ Isolate the A* algorithm from the `main`.

Three templates in `src/`:

- `controller.cpp`: our 'main' (that contains the ***entry point*** of our program)
- `astar.cpp`: the algorithm itself
- `astar.h`: the ***header*** describing what functionalities `astar.cpp` provides (eg, its *API*)

ASTAR.H

```
typedef std::tuple<int, int> Position;
typedef std::tuple<Position, int> Node; // {position, score}
typedef std::vector<Node> NodesList;
typedef std::tuple<bool, bool, bool, bool> Obstacle;

const int MAZE_SIZE = 100;
const Position START_POS {1, 1};
const Position END_GOAL {98, 98};

class AStar
{
public:
    AStar();
    std::string getNextMove(Obstacle obstacle);

private:
    // contains our maze (the portion we have already explored, anyway)
    // 'true' means 'there is a wall'
    std::array<bool, MAZE_SIZE * MAZE_SIZE> maze;

    Position current_position;

    Position planNextPosition();
    unsigned int heuristic(Position node, Position goal);
};
```

ASTAR.CPP

```cpp
#include "astar.h"

using namespace std;

AStar::AStar() :
    current_position(START_POS) // initialiase current_pos to our start position
{
    maze.fill(false); // maze is initially unknown -> assume no obstacles
}

Position AStar::planNextPosition()
{
    // todo!
}

unsigned int AStar::heuristic(Position node, Position goal)
{
    // todo!
}

std::string AStar::getNextMove(Obstacle obstacles)
{
    // todo!
    return "S";
}
```

CONTROLLER.CPP

```cpp
#include <iostream>
#include <string>

#include <cpprest/http_client.h>

#include "astar.h"
//...

int main(int argc, char* argv[])
{
    //...
    auto astar = AStar();
    //...

    string next_move("E");
    while(true) {
        uri_builder builder(U("/move/" + string(argv[1]) + "/" + next_move));
        http_response response = client.request(methods::GET, builder.to_string()).get();

        //...
        next_move = astar.getNextMove(response);
    }
}
```

YOUR TURN!

Starting from these templates, implement, compile and run a path finding algorithm in C++