Introductory Robot Programming

ENPM809Y

Lecture 01 – Course Overview and Introduction to C++

Zeid Kootbally
zeidk@umd.edu

Fall 2020

Overview I

Overview II

Conventions

- Slides created with Beamer LaTeX.
- C++ code displayed as follows with the LaTeX minted package:

```cpp
#include <iostream>

int main(){
    std::cout << "Hello" << std::endl;

    return 0;
}
```

- To create/open projects: ➤ CLion:Project:file.cpp
- *Best practice* will be provided.

Affiliations and Interests

- Affiliations:
  - National Institute of Standards and Technology (NIST), MD
    - Guest Researcher
  - University of Southern California, CA
    - Senior Research Associate
  - Contact: zeidk@umd.edu
- Interests – Robot Agility:
  - Industrial robotics, Knowledge representation, Task planning, Artificial Intelligence.
- Teaching: ENPM809Y, ENPM809B, and ENPM809E.

Affiliations and Interests | Robot Agility

- Small and medium manufacturers need robots that are easier to task and retask and are more robust to execution failure during operations.
  - Support for small lot productions.
  - Able to detect and recover from failures to avoid downtime.

Affiliations and Interests | Current Efforts

- Hardware agility.
  - How can different hardware configurations affect a robot's ability to accomplish a variety of tasks?
- Software agility.
  - How well can a robot adapt/respond to task failures?
  - How well can a robot re-plan when a new goal is provided to it?
  - How can we allow for interchangeability of robots without the need for reprogramming?
  - How well can a robot respond to changing environmental conditions?

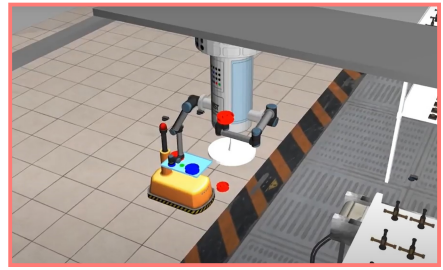
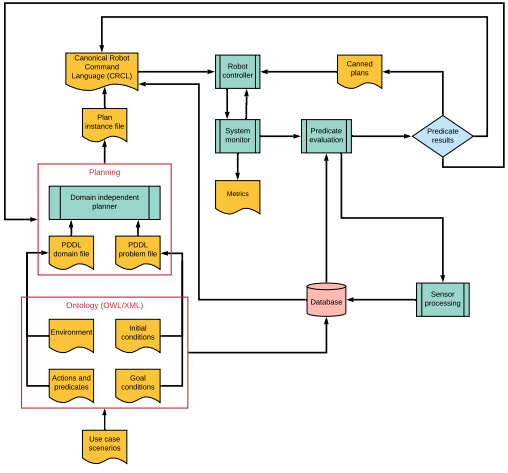


Figure: ARIAC 2020

Affiliations and Interests | Current Efforts | Big Picture



Affiliations and Interests | Current Efforts | IEEE Efforts

- NIST is developing definitions of agility, means of testing agility of a system, and underlying supporting technologies.
  - NIST has worked with IEEE to develop a standard industrial robot ontology [1].
  - The IEEE Robotics and Automation Society has started a Study Group to investigate the feasibility of creating standards and performance metrics to measure robot agility.
  - Per invitation by IEEE, NIST organized 4 rounds of the Agile Robotics for Industrial Automation Competition (ARIAC).
    - ARIAC 2017
    - ARIAC 2018
    - ARIAC 2019
    - ARIAC 2020

Class Introductions

- Grader: Sahana Anbazhagan (sahana22@umd.edu)
- Tell us about you.
  - What experience, if any, do you have with robotics?
  - Why are you interested in this class?
  - One interesting fact about yourself.

Class Introductions | System and Software

- Operating System (OS):
  - Windows, Mac OS, or Ubuntu Linux.
  - Ideal: Ubuntu 16.04 or 18.04, especially for ROS lectures.
- List of software:
  - CLion (or any IDE of your choice).
  - Doxygen – Should be installed for next class.
  - The Robot Operating System (ROS) – Should be installed for ROS lectures (Kinetic or Melodic).

Class Introductions | Textbooks (optional)

B. Stroustrup
The C++ Programming Language, 4th Edition
Addison-Wesley Professional; (May 19, 2013)
ISBN 978-0321563842

S.B. Lippman, J. Lajoie, and B.E. Moo
C++ Primer, 5th Edition
Addison-Wesley Professional; (August 16, 2012)
ISBN 978-0321714114

J.M. O'Kane
A Gentle Introduction to ROS
CreateSpace Independent Publishing Platform
ISBN 978-1492143239
Online: https://www.cse.sc.edu/~jokane/agitr/

Course Overview

- Specifically designed for students who have had little to no programming experience in their previous studies to prepare them for other ENPM robotics courses that require programming experience.
- MAINLY focuses on C++ programming and will provide a VERY BRIEF introduction to the Robot Operating System (ROS).
  - This is not a course about ROS (see ENPM808X, ENPM809E, or ENPM809B).
- We will still learn about mobile robotics.

Course Overview | Statistics

Letter Grades	2019 (%)	2020 (%)
A+	33.8	4.76
A	49.06	71.43
A-	5.26	23.81
B+	4.165	-
B	5.26	-

Course Overview | Class Structure

- Sometimes: Quiz on previous lecture (refer to syllabus).
- Questions and discussion on the previous lecture (10 min).
- Lecture with examples and exercises.
- Usually one topic per lecture.

Course Overview | Syllabus

- Week 1 – Course Presentation and Introduction to C++ .
- Week 2 – Structure of a C++ Program.
- Week 3 – Program Flow Control.
- Week 4 – Functions.
- Week 5 – Raw Pointers and References.
- Week 6 – Smart Pointers.
- Week 7 – Object Oriented Programming – Part I.
- Week 8 – Object Oriented Programming – Part II.
- Week 9 – Process and Thread.
- Week 10 – Path Planning Algorithm.
- Week 11 – Robot Operating System (ROS) – Part I.
- Week 12 – Robot Operating System (ROS) – Part II.
- Week 13 – Class and Function Templates.
- Week 14 – Final Class – Project Presentation.

Course Overview | Syllabus

Category	Points	Weight (%)
Quiz × 5	100	22.22
Real-world Application × 3	120	26.67
Final Project × 1	230	51.11
C++ code	100	-
Live demo	65	-
Report	65	-
Total	450	100

Course Overview | Syllabus | Quiz

- Quizzes are taken at the beginning of classes.
- Quizzes are hosted on Canvas.
- Quizzes usually last between 15 and 20 minutes.
- Notes and software not allowed.
- Quiz dates are posted on Canvas.
- Types of question:
  - True/False.
  - Essay.
  - Multiple choice.
- An example of quiz can be found on Canvas.

Course Overview | Syllabus | Real World Application (RWA)

- Individual and group RWAs.
- RWAs must be submitted in due time (penalty for late submission).
- RWAs must be submitted as an archive file (zip, tar.gz, rar, . . .)
- Peer reviews required for group RWAs.
  - Rate your teammates on their contribution (1-10).
  - Rate yourself on your contribution (1-10).

Course Overview | Syllabus | Final Project

- Final project is a group project.
- Write a Breadth-first Search algorithm to move two robots from a starting position to a goal position in a maze.
- The robots have no *a priori* knowledge of the walls in the maze and will discover them in real time.
- Your algorithm should be robust enough to handle different mazes.
- The final project mainly focuses on:
  - Object Oriented Programming.
  - Polymorphism and Inheritance.
- Peer reviews needed for the final project: 30 pts for C++ code and 20 pts for report.
- The final project is NOT a ROS-based project

Course Overview | Syllabus | Final Project

- Grading will take into account:
  1. How well your program performs.
  2. How well you documented your code (see slide 27).
  3. How well your code follows the coding guidelines and conventions (see slide 28).
- Group reports should clearly describe the overall approach and the contribution of each member.

Course Overview | Syllabus | Peer Reviews

- Peer reviews required for RWAs (10/40) and final project (50/230).

The C++ Language | Why Learn C++?

- The department offers courses that require a C++ background.
  - ENPM809B.
  - ENPM808X.
  - ...
- Popular
  - Lots of code still written in C++ .
  - Programming language popularity indexes (Tiobe, Pypl).
  - Active community, e.g., Github, stackoverflow.

The C++ Language | Why Learn C++?

- Powerful:
  - Fast, flexible, and portable.
  - Procedural and Object-oriented.
- Good career opportunities:
  - C++ skills always in demand.
  - C++ = Salary ++
- Many robotic projects are written in C++ :
  - A lot of low-level hardware libraries use C++ .
  - C++ allows for real time performance.
  - C++ is a very mature programming language.

The C++ Language | Classical and Modern C++?

Classical

- Early 1970s
  - C programming language
  - Dennis Ritchie
- 1979
  - Bjarne Stroustrup
  - C with Classes
- 1983: Name changed to C++
- 1989: First commercial release
- 1998: C++ 98 Standard
- 2003: C++ 03 Standard

Modern

- 2011: C++11 Standard
  - Lots of new features.
- 2014: C++14 Standard
  - Some changes.
- 2017: C++17 Standard
  - Simplification.
- 2020: C++20 Standard underway
  - New features.

The C++ Language | Working with C++

- Editor to write your program (.cpp and .h files)
- Program to tell the computer EXACTLY what to do (instructions)
- Compiler to translate from source to binary
- Linker to link together our code with other libraries before creating the executable program
- Testing and debugging

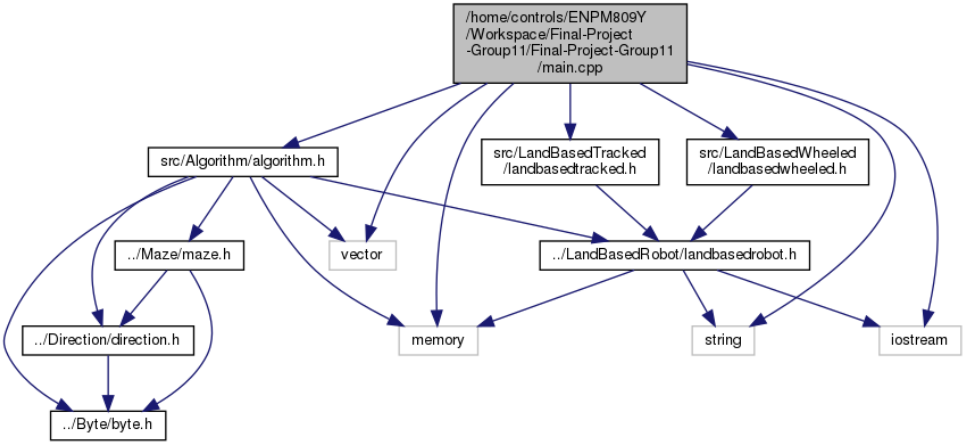
The C++ Language | Style Guides and Coding Conventions

- When working as a professional programmer, you will likely be required to conform to your employer's coding conventions.
- You should get used to following a specification.
- In this course we will use the C++ Style Guides provided by Google.
  - As we progress and learn new concepts, you should refer to these C++ style guides and coding conventions.

The C++ Language | Documentation with Doxygen

- Documenting your code is very important.
  - It will help you understand what you wrote a few weeks back.
  - It will help anybody that works with you on the same code or anybody that comes after you.
- Doxygen is the *de facto* standard tool for generating documentation from annotated C++ sources.
  - It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages.
  - Doxygen can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

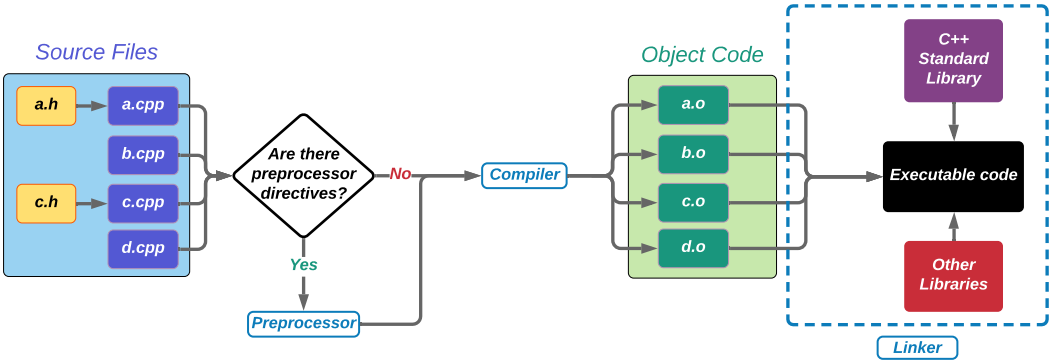
The C++ Language | Documentation with Doxygen



The C++ Language | Programming Language

- C++ is a programming language.
  - A programming language allows us to give instructions to a computer in a language the computer understands.
  - Just as many human-based languages exist, there are an array of computer programming languages that programmers can use to communicate with a computer.
- Source code: high-level code for humans.
- Object code or binary: Portion of the language that a computer can understand.
- Translating source code into binary is known as "compiling".
- Examples of C++ compilers: GCC (GNU project), Visual C++ , C++ Builder, Turbo C++ , . . .

The C++ Language | Programming Language



The C++ Language | Integrated Development Environment

- An Integrated Development Environment (IDE) is a software that provides:
  - Editor.
  - Compiler.
  - Linker.
  - Debugger.
- Keeps everything in sync.
- IDE used in this course is CLion.
  1. Overview of the CLion interface.
  2. Test debugger.

The C++ Language | The Command-line Interface

- Why use the command-line interface?
  - You do not like working with IDEs.
  - Your system does not have enough memory to handle an IDE.
  - You want to study what is happening behind the scenes.
- What do you need?
  - Text editor (not a Word Processor).
  - A compiler (see compiler options)
  - A terminal.
- `g++ file.cpp` Compile and create an executable file a.out (default target name).
- `g++ -E file.cpp` Generate preprocessed code.
- `g++ -S file.cpp` Generate a file.s assembly source file.
- `g++ -c file.cpp` Generate only the object file file.o.
- `g++ -o myexe file.cpp` Compile and link file.cpp to generate the executable myexe.

The C++ Language | Web-based Compiler

- Why use the web-based compiler?
  - You do not have your laptop
  - You want to check something quickly
  - You dream of C++ and want to try something right away
- What do you need?
  - A browser
    - C++ Shell
    - repl.it
    - ideone

The C++ Language | First C++ Program

1. Create a project ➤ CLion:Lecture1:FirstProject for C++17

```
#include <iostream>

int main(){
    std::cout << "Hello World" << std::endl;

    return 0;
}
```

The C++ Language | First C++ Program | Preprocessor Directive

```
#include <iostream>
```

- `#include`: A preprocessor directive used to include a standard or user-defined file in the program and is mostly written at the beginning of any C/C++ program. This directive tells the preprocessor to include the contents of the file specified in the input stream to the compiler and then continue with the rest of the original file.
- `iostream`: Header file which refers to a family of class templates and supporting functions in the C++ Standard Library that implement stream-based input/output capabilities.
- <>: Used to include header files from the C++ Standard Library. To include a user-defined file we usually use "", e.g., `#include "myfile.h"`

The C++ Language | First C++ Program | main Function

```
int main(){}
```

- The `main()` function happens to be the entry point of any (standard-compliant) C++ program and must be defined.
- The compiler arranges for the `main()` function to be called when the program begins execution.
- It has a return type (and in some cases accepts inputs via parameters).
- `main()` may call other functions which may call yet other functions.
- Every C++ program must have exactly one `main()` function.
- The return type of `main()` must be `int` and not `void`.
- `void main()` is not and never has been conformed to C++ , nor has it even been conformed to C (see page 71 of C++ standard).

The C++ Language | First C++ Program

- `std::cout << "Hello" << std::endl;`
  - This is an instruction which prints a string message on the standard output.
  - `std` is a namespace. The `::` operator is the scope operator. It tells the compiler which class/namespace to look in for an identifier. The `endl` is a pre-defined object of ostream class. It is used to insert a new line character and flushes the stream.
  - << is called insertion operator and sends bytes to an output stream object.
  - Semicolon is a command in C++ . The Semicolon lets the compiler know that it has reached the end of a command.

Errors and Warnings | Compiler Errors

- When processing the source code, the compiler checks that rules of the language are respected.
- **Syntax errors**: Something wrong with the structure of your program.
- **Semantic errors**: Syntax is correct but the meaning of the code does not make sense to the compiler
  - Suppose we want to add 2 variables `a` and `b` (`a+b`) with `a` an integer and `b` a string literal.
  - The compiler will throw an error because of variable incompatibility.

Errors and Warnings | Compiler Warnings

- The compiler detected an error with your code that could generate a potential problem.
- A warning is different from an error.
- A warning means that the compiler is still able to generate correct machine code (object files .o or .obj).
- Do not ignore compiler warnings.

```
#include <iostream>

int main(){
    int number;
    int number2;
    std::cout << number << std::endl;
    return 0;
}
```

Errors and Warnings | Linker Errors

- The linker is having trouble linking all the objects file to create an executable.
- Usually there is a library or object file that is missing.
- Fixing linker errors can be tricky.
- Fixing a path to an object file or library can solve the problem sometimes.

Errors and Warnings | Linker Errors

```
#include <iostream>

extern int x;
int main(){
    std::cout << x;
    return 0;
}
```

- `extern int x;` tells the compiler that an object of type `int` called `x` exists somewhere.
- However, the program does not say where to find `x`.

Errors and Warnings | Runtime Errors

- Errors that occur when the program is running.
- Errors difficult to predict when we are writing our program.
- Examples:
  - Divide by 0.
  - File not found.
  - Out of memory.
- Can cause your program to crash.
- Exception handling can help address runtime errors.

Errors and Warnings | Logic Errors

- Mistakes made by the programmer(s).
- Logic errors cause your program to run incorrectly and have unexpected outcome.
- Example: You need to be at least 18 to vote.
- The following code snippet contains a logic error:

```
if (age > 18){
    std::cout << "You can vote";
}
```

  - Correct statement:

```
if (age >= 18){
    std::cout << "You can vote";
}
```

Next Class | 09/10

- Structure of a C++ program.
- In-person lecture at UMD.
- Bring your laptop.
- Stay safe!

References

[1] C. Schlenoff, E. Prestes, R. Madhavan, P. Goncalves, H. Li, S. Balakirsky, T. Kramer, and E. Migueláñez.
An IEEE standard Ontology for Robotics and Automation.
In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1337–1342, 2012.