4SC020 Mobile Robot Control 2024: Best practices for C++ and Git

26TH OF APRIL 2024

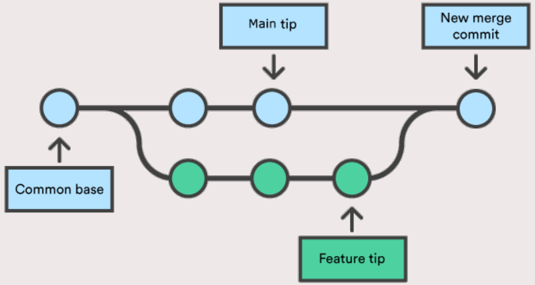
Koen de Vos

Mechanical Engineering, Robotics

EINDHOVEN UNIVERSITY OF TECHNOLOGY

What is this lecture about?

- Best practices for programming in C++
- The importance of code quality
- Basic Introduction to GIT



Main tip

New merge commit

Common base

Feature tip

C++ Programming

```
#include <iostream>

int main()
{
   std::cout << "Hello MRC Students :)";
   return 0;
}
```

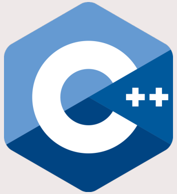
What is C++

C++ is a compiled, statically typed language that offers strong type checking at compile time, ensuring robustness and efficiency in software development.

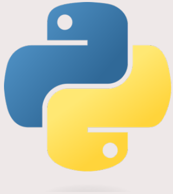
ChatGPT, 2024

But what does this mean?

Why is it different from Matlab/Python/…



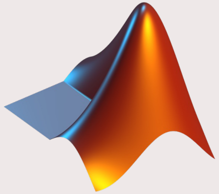
- Compiled
- Statically typed
- Weakly typed



- Interpreted
- Dynamically typed
- Strongly typed

- Duck typing

If it walks like a duck and quacks like a duck, it is a duck.



- Compiled
- Dynamically typed
- Weakly typed

What does the compiler do?

- Parsing
  - The compiler analyzes the code you’ve written, and ensures the syntax is correct

- Optimization
  - Depending on your compiler configuration, the compiler tries to improve performance and efficiency by rearranging and sometimes eliminating instructions

- Code Generation
  - The compiler generates machine code, translating C++ constructs into instructions understood by the processor.

- Error detection
  - It identifies and reports syntax errors, type mismatches, and other issues that could cause the program to behave unexpectedly.

A typical C++ project

- build *Contains the executable after compilation*
  - main
- Config *Contains any configuration files you need*
  - params.json
- include *Contains the header files, and third party libraries*
  - implementation.h
- test *Contains test files, to verify correct implementation*
- src *Contains source files*
  - implementation.cpp
  - main.cpp
- CMakeLists.txt *File to configure Cmake, for compilation*

Header and Source Files

```
#include <iostream>
#include "addition.h"

int main() {
   int result = add(3, 5);
   std::cout << "Result: " << result << std::endl;
   return 0;
}
```

main.cpp

```
#include "addition.h"

int add(int a, int b) {
   return a + b;
}
```

addition.cpp

```
#pragma once

/**
 * @brief Adds two integers.
 *
 * This function takes two integers as input
and returns their sum.
 *
 * @param a The first integer operand.
 * @param b The second integer operand.
 * @return The sum of a and b.
 */
int add(int a, int b);
```

addition.h

C++ features you might want to use

Pass by reference

```cpp
#include <iostream>

void increment(int &num) {
    num++;
}

int main() {
    int number = 5;
    std::cout << "Before increment: " << number << std::endl;
    increment(number);
    std::cout << "After increment: " << number << std::endl;
    return 0;
}
```

C++ features you might want to use

Pass by reference

```cpp
#include <iostream>

int main() {
    const int constant = 10;
    // constant = 20; // This would cause a compilation error
    std::cout << "The value of constant is: " << constant << std::endl;
    return 0;
}
```

C++ features you might want to use

Lambda Expressions

```cpp
#include <iostream>

int main() {
    int x = 10;
    int y = 20;
    auto add = [](int a, int b) { return a + b; };
    std::cout << "Sum of x and y is: " << add(x, y) << std::endl;
    return 0;
}
```

C++ features you might want to use

Range-based for loops

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> numbers = {1, 2, 3, 4, 5};
    for (int num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

C++ features you might want to use

Templates

```cpp
#include <iostream>

template <typename T>
T add(T a, T b) {
    return a + b;
}

int main() {
    int x = 5, y = 10;
    std::cout << "Sum of x and y is: " << add(x, y) << std::endl;

    double a = 3.5, b = 2.5;
    std::cout << "Sum of a and b is: " << add(a, b) << std::endl;
    return 0;
}
```

The C++ standard library

Python/Matlab provide you with a lot of build in features.

Eventough beginners often don’t know about it, so does C++.

Algorithms for: Sorting, Searching, Reversing, …

https://en.cppreference.com/w/cpp/header

The C++ standard library

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    // Create a vector of integers
    std::vector<int> numbers = {5, 2, 8, 1, 9, 3};

    // Sort the vector in ascending order using std::sort
    std::sort(numbers.begin(), numbers.end());
    return 0;
}
```

The C++ standard library

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

int main() {
    // Create a vector of Person objects
    std::vector<Person> people = {
        {"Alice", 25},
        {"Bob", 30},
    };

    // Define a lambda expression as a variable for comparing by age
    auto compareByAge = [](const Person& a, const Person& b) {
        return a.age < b.age; // Sort in ascending order of age
    };

    // Sort the list of people based on age using the lambda variable
    std::sort(people.begin(), people.end(), compareByAge);

    return 0;
}
```

```cpp
// Define the Person class
class Person {
public:
    std::string name;
    int age;

    // Constructor
    Person(const std::string& n, int a) : name(n), age(a) {}
};
```

With a little help it even works on our own custom data-types and classes!!

But how does {feature} work?

Some very good resources for all your C++ related questions:

https://en.cppreference.com/
https://cplusplus.com/reference/

https://www.w3schools.com/cpp/cpp_intro.asp
https://ocw.mit.edu/courses/6-096-introduction-to-c-january-iap-2011/

https://stackoverflow.com/

Furthermore, try asking:

https://google.com
ChatGPT/Bing Chat/….

Code Quality

Important Considerations

1. Whenever possible initialize variables

```
vector<float> laser_beam_readings(10);

for (float reading:laser_beam_readings){
        cout << reading <<" ";
    }
```

```
vector<float> laser_beam_readings(10,100);

for (float reading:laser_beam_readings){
        cout << reading <<" ";
    }
```

Important Considerations

2. Use meaningful variable names

```
vector<float> readLaser(vector<float> &laser_beam_readings, bool &a)

int main(int argc, const char * argv[]) {

    bool f=false;
    readLaser(laser_beam_readings, f)

    if (f){
        cout << "After reading the measurements" << endl;
        for (float reading:laser_beam_readings){
            cout << reading <<" ";
        }
```

Important Considerations

2. Use meaningful variable names

```
vector<float> readLaser(vector<float> &laser_beam_readings, bool &measValid)

int main(int argc, const char * argv[]) {

    bool laserMeasValid=false;
    readLaser(laser_beam_readings,laserMeasValid);

    if (laserMeasValid){
        for (float reading:laser_beam_readings){
            cout << reading <<" ";}
        }
    }
```

Important Considerations

3. Avoid magic numbers

```
getMinValue(laser_beam_readings, minimum_dist_obs);

    if(minimum_dist_obs < 0.6)
    {
        cout << "Stop the robot\n";
    }
```

```
const float safety_distance = 0.6; //cm

if(minimum_dist_obs < safety_distance)
    {
        cout << "Stop the robot\n";
    }
```

Important Considerations

4. Use a single source of definition (preferably a configuration file)

```
struct ConfigParams {
    const float safety_distance = 0.6; // cm;
    const float robot_radius    = 0.57; // cm
    const float robot_max_omega = 0.2; // rad/sec
    const float robot_max_vel   = 0.7; // m/sec
} robot_config;


if(minimum_dist_obs < robot_config.safety_distance)
    {
        cout << "Stop the robot\n";
    }
```

Important Considerations

5. Try to reuse, instead of replicate.

```
#include <cmath>
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int x = 1, y = 2;
    // of (1, 2)
    int m1 = sqrt(x * x + y * y);
    std::cout << m1 << "\n";
    int x = 2, y = 3;
    // magnitude of (2,3)
    int m2 = sqrt(x * x + y * y);
    std::cout << m2 << "\n";
}
```

```
#include <cmath>
#include <stdio.h>
float magnitude(int x, int y)
{
    return sqrt(x * x + y * y);
}
int main(int argc, const char *argv[])
{
    std::cout << magnitude(1, 2);
    std::cout << magnitude(2, 3);
}
```

Important Considerations

6. Write sufficient Documentation and comments, but

Do not comment obvious things:

```
// Calculate the distance
float distance = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
// Distance formula

++counter;
// increment counter
```

Important Considerations

6. Write sufficient Documentation and comments, but

Do not disable code with comments, that’s what version control is for

```
// This function is no longer used
/*
int computeManhattanDistance(const Point& p1, const Point& p2)
{
int distance = abs(p1.x -p2.x) + abs(p1.y -p2.y);
return distance;
}
*/
```

Important Considerations

6. Write sufficient Documentation and comments, but

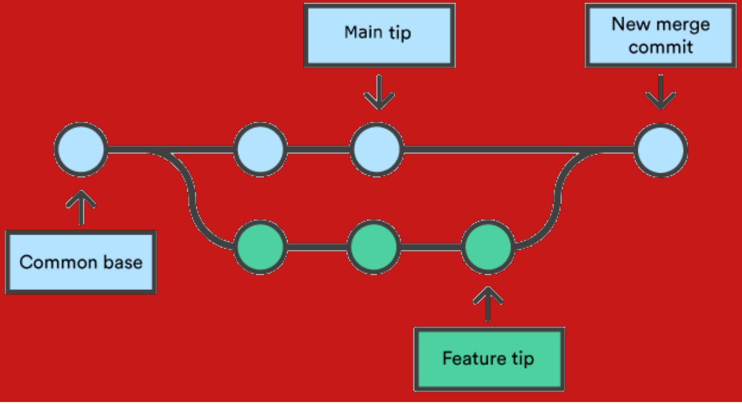
Do:

- Make sure that your comments add value to the code
- Highlight design decisions and assumptions
- Explain always why, not how!
- Try to be as short and expressive as possible

Consider using Docstrings to define the interfaces of your functions **(in Vscode, ctrl+shift+P -> generate doxygen comment)**

```
/**
 * @brief Calculate the sum of two integers.
 *
 * This function takes two integers as input and returns their sum.
 *
 * @param a The first integer.
 * @param b The second integer.
 * @return The sum of the two integers.
 */
int calculateSum(int a, int b) {
    return a + b;
}
```

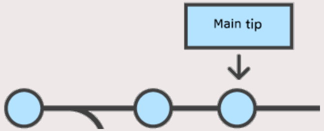
Version Control/GIT



A typical Git workflow

Assuming you have already cloned your repo:

- git pull: make sure you’re up-to-date
- *change some files, fix some bugs*
- git status
- git add {files you changed}
- git commit –m {Some meaningful message}
- git push



A typical Git workflow cont'd

Sometimes you'll need:

git branch (create branches)

git branch new-feature

git checkout (Switch branch)

git checkout new-feature

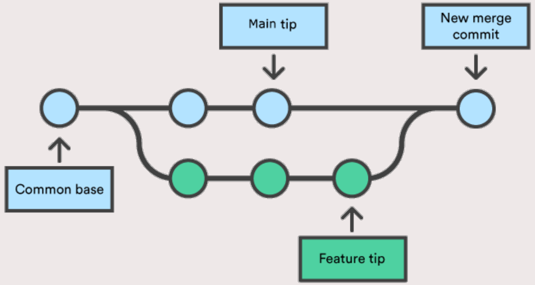
git merge

git merge feature main

git stash (Temporarily store changes)

git stash

git stash pop



Best practices for using GIT?

- Make incremental and small changes
- Each commit should only contain the results of a single task (a feature, a bug fix, a refactor)

- A commit message should be descriptive

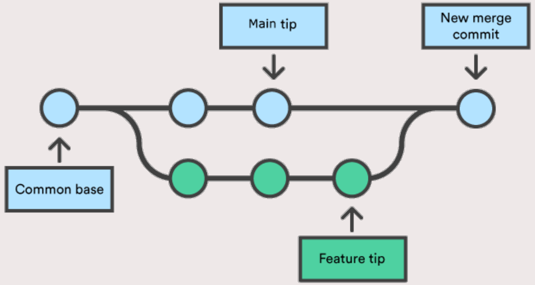
- Develop using branches
- Review each others code, before merging into main/master

- *Use .gitignore*: Don’t push files that are not necessary (build files especially).

- **Most important:** try to Never push broken code to **main/master**

What was this lecture about?

- Best practices for programming in C++
- The importance of code quality
- Basic Introduction to GIT


Main tip
New merge commit
Common base
Feature tip