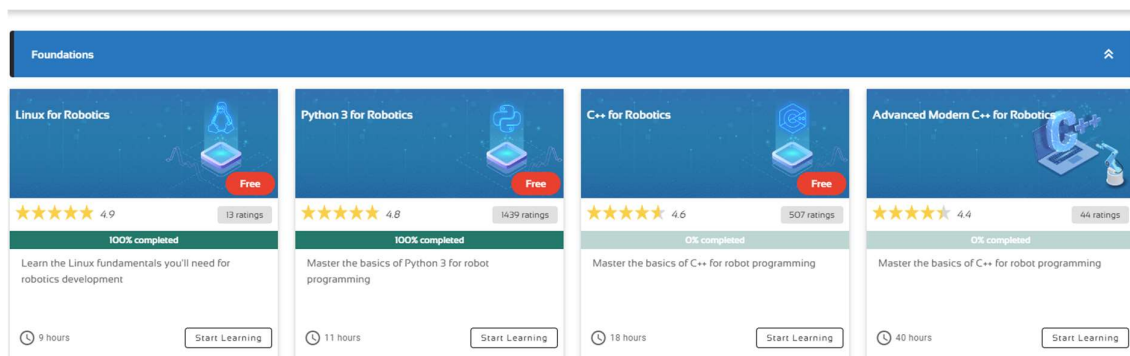


## UTS SEMESTER GANJIL 2023/2024

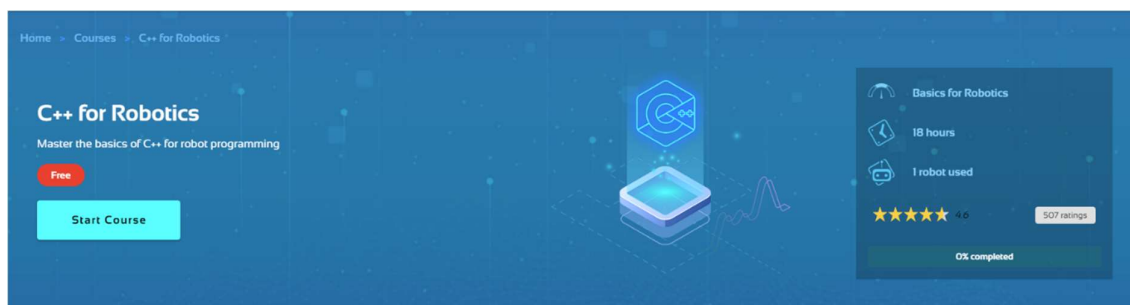
Nama : Mochamad Phillia Wibowo  
NIM : 1103204191  
Kelas : TK-44-G7  
Matakuliah : Robotika dan Sistem Cerdas

### Technical Report C++ For Robotics | C++ Basics

- The Construct/Constructsim  
The Construct adalah platform e-learning untuk ROS dan Robotika, yang membantu para engineer untuk mengembangkan keterampilan robotika mereka dengan kurikulum berskala penuh dan kursus-kursus langsung. Di dalam the construct ini terdapat banyak kursus yang di bagi menjadi beberapa kategori. Pada pembahasan kali ini, kategori yang diambil adalah “Foundations” yang di dalam nya banyak kursus Bahasa pemrograman untuk robotika. Disini saya akan membahas tutorial pada kursus C++ For Robotics.



Banyak kursus dalam kategori Foundations



Tampilan awal pada kursus C++ for Robotics

Kursus ini dirancang khusus untuk individu yang bercita-cita menjadi Developer ROS, suatu bidang yang menuntut pemahaman mendalam tentang bahasa pemrograman C++. Dalam perjalanan pembelajaran ini, kita akan belajar untuk menguasai konsep-konsep

kunci yang esensial dalam C++, dengan fokus pada aplikasinya dalam lingkungan Robot Operating System (ROS).

Pada pembelajaran dimulai dengan cara mengkompilasi program C++, suatu keterampilan dasar yang fundamental dalam pengembangan perangkat lunak robot. Selanjutnya, kursus akan membimbing untuk memahami bagaimana menyimpan data ke dalam variabel dan mengoperasikan data tersebut dengan efisien. Penguasaan konsep ini menjadi pondasi penting dalam pemrograman robot, terutama ketika berurusan dengan data sensor atau perintah kontrol.

Salah satu aspek kunci dalam kursus ini adalah pembelajaran mengenai perubahan perilaku berdasarkan kondisi. Disini akan belajar bagaimana membuat keputusan dalam kode berdasarkan kondisi-kondisi tertentu, suatu kemampuan yang sangat relevan dalam pengembangan perangkat lunak kendali robot.

Selanjutnya, akan diperkenalkan pada pembuatan fungsi yang dapat dipanggil dari berbagai bagian dalam kode. Hal ini memungkinkan pembuatan kode yang modular dan lebih mudah dipelihara, suatu praktik yang sangat penting dalam pengembangan perangkat lunak yang kompleks.

Kemudian, kursus akan membahas penggunaan array dan pointer dengan benar. Keahlian ini akan memberikan kemampuan untuk mengelola dan mengakses data dalam skala yang lebih besar, yang seringkali ditemui dalam konteks pengembangan robot.

Seiring dengan itu, Pada proses pembelajaran akan diajak untuk memahami konsep pengenkapsulasian kode ke dalam kelas. Dengan memahami konsep ini, kita dapat menciptakan kode yang bersih, terstruktur, dan mudah diintegrasikan ke dalam proyek-proyek ROS.

Dengan merangkum, kursus ini membekali peserta dengan keterampilan esensial dalam C++ yang diperlukan untuk menjadi seorang Developer ROS yang kompeten. Dengan mengeksplorasi berbagai konsep ini, peserta akan dapat membangun aplikasi robotik yang kuat, efisien, dan dapat diandalkan.

- ROSbot 2.0



## 1. Overview Robot

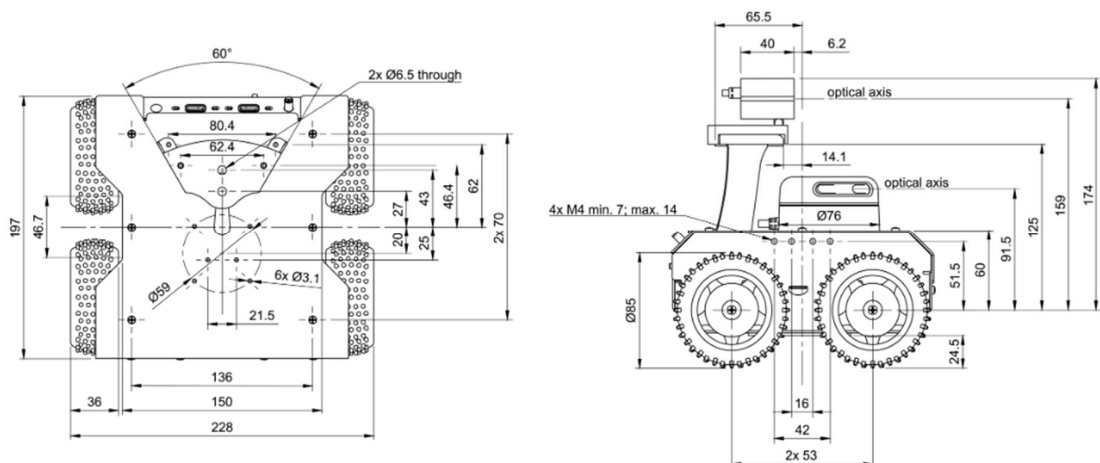
ROSbot adalah platform robot otonom dan sumber terbuka berdasarkan ROS yang dikembangkan oleh Husarion. Dengan simulasi robot ROSbot, pengguna dapat melakukan hal serupa tanpa risiko merusak versi aslinya. Pengguna dapat menggunakannya untuk menguji algoritma apa pun, seperti Navigasi Robot, Perencanaan Gerak, Deteksi Objek. Setelah pengguna memiliki kode yang berfungsi dalam simulasi, pengguna dapat langsung menjalankannya di robot ROSbot yang sebenarnya, dengan sedikit atau tanpa perubahan

ROSbot adalah platform robot seluler otonom penggerak 4x4 bertenaga ROS yang dilengkapi dengan LIDAR, kamera RGB-D, IMU, encoder, sensor jarak yang tersedia dalam tiga versi: "2R", "2 PRO" dan "2".

ROSbot adalah platform robot yang terjangkau untuk pengembangan robot otonom yang cepat. Ini bisa menjadi basis untuk robot layanan khusus, robot inspeksi, dan robot yang bekerja dalam kawanan. Semua versi terintegrasi:

- Platform bergerak 4-roda yang berisi motor DC dengan encoder dan rangka aluminium
- Kamera Orbbec Astra RGBD
- Sensor inersia MPU 9250 atau BNO055 (akselerometer + giro)
- panel belakang yang menyediakan antarmuka untuk modul tambahan

## 2. Specification Robot



Tabel Spesifikasi Robot

Attribute	Description
Dimensions with camera and LiDAR	198 x 228 x 218 mm / 7.80 x 8.98 x 8.58 in [L x W x H]
Dimensions without camera	198 x 228 x 144 mm / 7.80 x 8.98 x 5.67 in [L x W x H]
Dimensions without camera and LiDAR	198 x 228 x 103 mm / 7.80 x 8.98 x 4.06 in [L x W x H]
Weight	2,84 kg / 100 oz (with camera and LiDAR), 2,45 kg / 86 oz (without camera and LiDAR)
Wheel diameter / Clearance / Wheelbase	85 mm / 24 mm / 106 mm
Chassis material	Powder-coated aluminum plate, 1.5 mm thick
Maximum translational velocity	1.0 m/s
Maximum rotational velocity	420 deg/s (7.33 rad/s)
Maximum load capacity	Up to 5 kg / 176 oz *not in continuous work
Battery life	1.5h - 5h

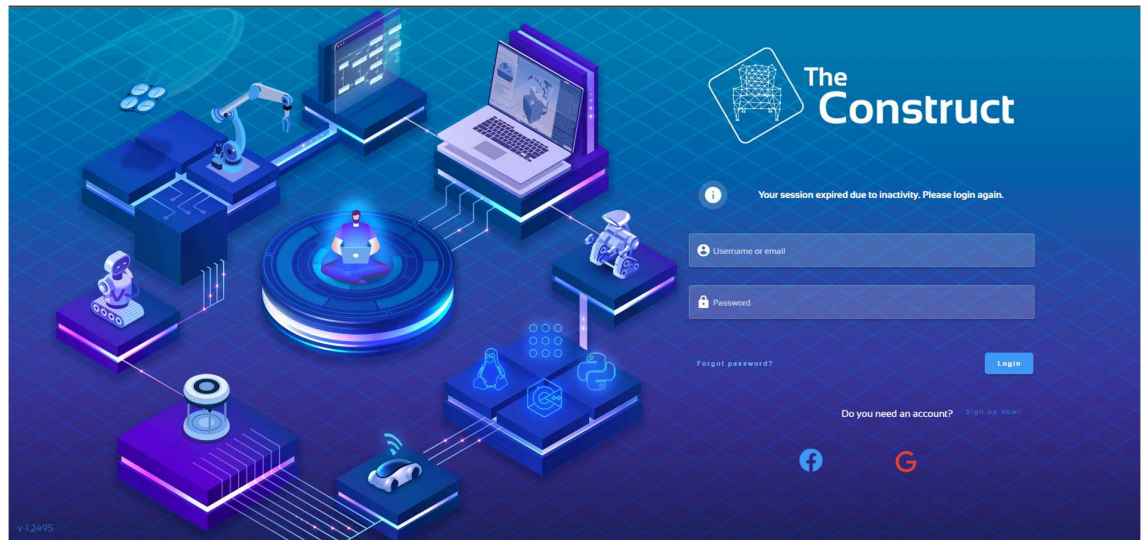
### 3. Component Robot

Tabel Component Description

Component	Quantity	Description
SBC	1	Asus Tinker Board with 2 GB RAM, Rockchip RK 3288 with 4x 1.80 GHz as CPU and a ARM Mali-T764 MP2 as a GPU and 32 GB MicroSD. The SBC runs on Ubuntu-based OS, customized to use ROS.
LIDAR	1	RpLidar A2, 360 degree and up to 8m range
IMU Sensors	1	Powerful 9-Axis Accel/Gyro/Magnetometer sensor with MPU-9250, or Intelligent 9-axis absolute orientation sensor BNO055

- Pembahasan Tutorial C++ For Robotics | C++ Basics

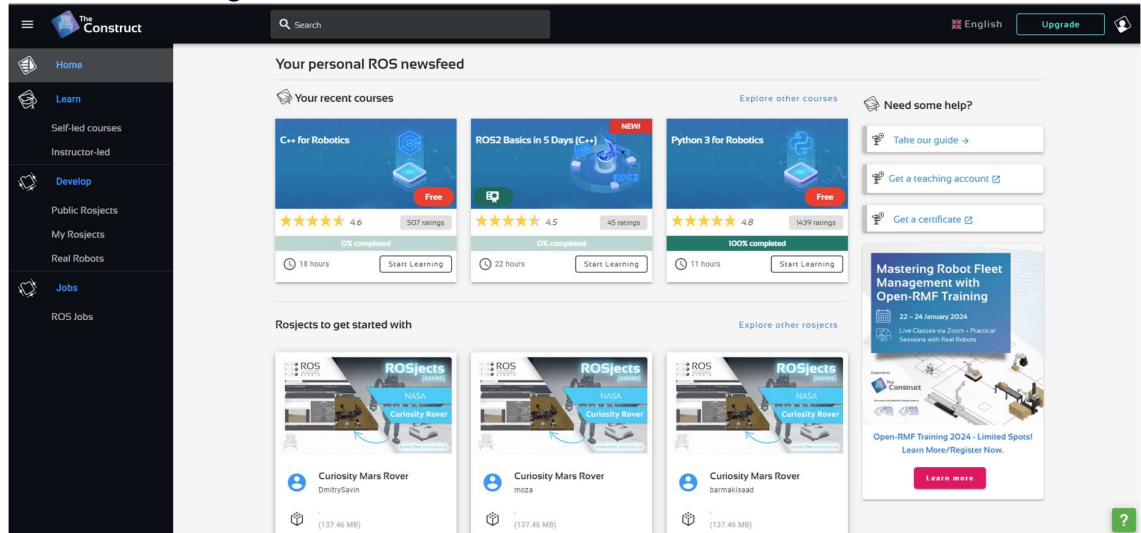
## 1. Part One : Login The Construct/The Constructsim



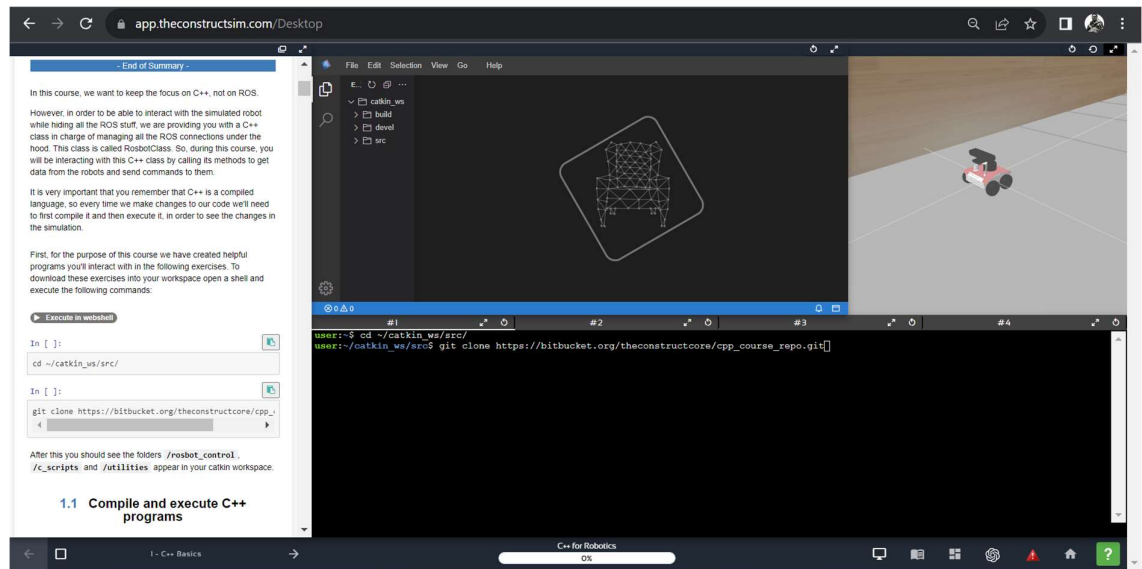
- Masukkan Username/email dan Password apabila sudah memiliki akun.
- Bagi yang belum memiliki akun bisa mengklik Sign up now.
- Atau bisa juga menggunakan akun pribadi Facebook atau Google.

## 2. Part Two : Go to the Courses C++ For Robotics

Klik Start Learning



### 3. Part 3 : Setup Your Workspace



Secara rinci, beberapa elemen dalam perintah tersebut dapat dijelaskan sebagai berikut:

- **git clone**: Perintah untuk mengkloning repositori Git.
- **https://bitbucket.org/theconstructcore/cpp\_course\_repo.git**: URL repositori Git yang akan di-clone. Dalam kasus ini, repositori tersebut terletak di Bitbucket pada alamat tersebut.
- **Direktori lokal**: Direktori di komputer pengguna tempat repositori Git akan di-clone. Jika tidak disertakan, Git akan membuat direktori baru dengan nama repositori.

Tujuan dari perintah tersebut adalah untuk mendapatkan salinan lengkap dari repositori yang terdapat di Bitbucket ke dalam sistem lokal pengguna. Setelah eksekusi perintah ini, pengguna akan memiliki salinan penuh dari proyek yang dapat diedit dan dikelola pada komputernya sendiri. Perintah ini umumnya digunakan saat seseorang ingin berkolaborasi, mengedit, atau mengembangkan proyek yang sudah ada di repositori Git tertentu.

#### 4. Part Four : Compile and execute C++ programs

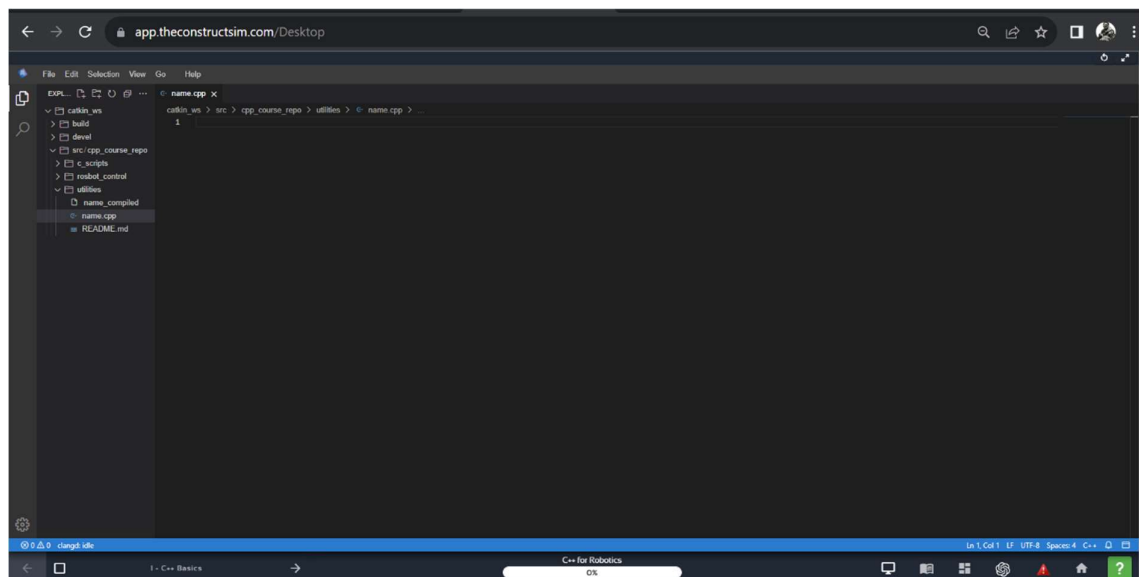
```
#!
user:~/catkin_ws/src$ cd ~/catkin_ws/src/cpp_course_repo/utilities/
user:~/catkin_ws/src/cpp_course_repo/utilities$ touch name.cpp
```

Perintah `cd ~/catkin_ws/src/cpp_course_repo/utilities/` digunakan untuk berpindah direktori ke lokasi tertentu dalam struktur direktori pada sistem file.

- **cd**: Singkatan dari "change directory," perintah ini digunakan untuk pindah dari satu direktori ke direktori lainnya.
- **~/catkin\_ws/src/cpp\_course\_repo/utilities/**: Ini adalah path lengkap ke direktori yang ingin diakses. Dalam hal ini, dimulai dari direktori home (~), kemudian ke direktori **catkin\_ws**, lalu **src**, dan akhirnya **cpp\_course\_repo/utilities/**.

Jadi, setelah menjalankan perintah ini, lokasi kerja (current working directory) pada terminal akan berpindah ke `~/catkin_ws/src/cpp_course_repo/utilities/`. Ini berarti semua perintah atau operasi yang dilakukan dalam terminal setelah perintah ini akan berlaku di dalam direktori tersebut.

Perintah **touch name.cpp** digunakan untuk membuat file baru dengan nama "name.cpp". Di sini, "touch" adalah perintah yang biasanya digunakan untuk memperbarui timestamp dari suatu file. Jadi, setelah menjalankan perintah ini, sistem akan memiliki file baru dengan nama "name.cpp" dalam direktori tempat perintah ini dijalankan.



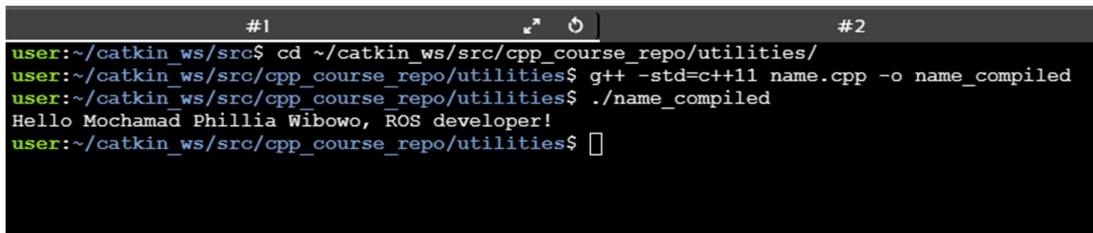
Pada file name.cpp yang tadi telah dibuat, tuliskan code dibawah ini.

```
#include <iostream>

int main() {
    printf("Hello Mochamad Phillia Wibowo, ROS developer!  \n");

    return 0;
}
```

Kode tersebut secara umum adalah program C++ yang sederhana untuk mencetak pesan sambutan. Jika ingin menjalankannya, pastikan pengguna sudah menyimpannya dalam file dengan ekstensi ".cpp" dan kemudian mengkompilasi serta menjalankannya menggunakan compiler C++, seperti g++:



```
#1 #2
user:~/catkin_ws/src$ cd ~/catkin_ws/src/cpp_course_repo/utilities/
user:~/catkin_ws/src/cpp_course_repo/utilities$ g++ -std=c++11 name.cpp -o name_compiled
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
Hello Mochamad Phillia Wibowo, ROS developer!
user:~/catkin_ws/src/cpp_course_repo/utilities$
```

- **g++:** Ini adalah perintah untuk mengkompilasi program C++ menggunakan compiler GNU C++.
- **-std=c++11:** Opsi ini menentukan bahwa compiler harus mengikuti standar C++11. Standar ini merujuk pada versi standar C++ yang diterbitkan pada tahun 2011
- **name.cpp:** Ini adalah nama file sumber yang akan dikompilasi.
- **-o name\_compiled:** Opsi ini menentukan nama output dari proses kompilasi. Dalam hal ini, outputnya akan menjadi file bernama "name\_compiled" (biasanya ini adalah file biner atau eksekutabel).

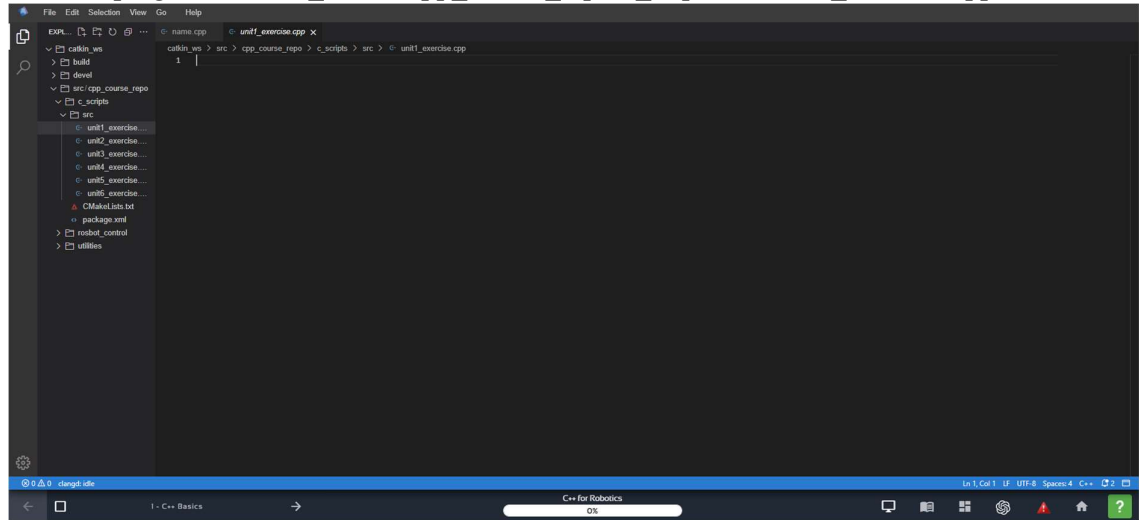
Jadi, setelah menjalankan perintah ini, jika tidak ada kesalahan dalam kode, pengguna akan mendapatkan file bernama "name\_compiled" yang dapat dijalankan untuk menjalankan program C++ yang telah ditulis sebelumnya. Yaitu "Hello Mochamad Phillia Wibowo, ROS developer!" ke layar.



## 5. Part Five : Compile and execute C++ programs in ROS

Pengguna akan melihat bagaimana program C++ dapat dikompilasi dan dieksekusi serta bagaimana program tersebut berinteraksi dengan robot ROS dalam simulasi

Pertama pergi ke : catkin\_ws/src/cpp\_course\_repo/c\_scripts/src/unit1\_exercise.cpp



Kemudian ketikkan code dibawah ini di dalam file unit1\_exercise.cpp

```
#include "rosbot_control/rosbot_class.h"
#include <ros/ros.h>

using namespace std;

int main(int argc, char **argv) {
    ros::init(argc, argv, "rosbot_node");

    RosbotClass rosbot;
    rosbot.move();

    float coordinate = rosbot.get_position(1);

    ROS_INFO_STREAM(coordinate);

    return 0;
}
```

Penjelasan Code :

- `#include "rosbot_control/rosbot_class.h"`  
`#include <ros/ros.h>`
  - Baris ini mengimpor header file yang berisi definisi kelas dan fungsi yang diperlukan untuk mengendalikan robot. Selanjutnya, "ros/ros.h" adalah header file dari ROS yang menyediakan fungsi-fungsi dasar untuk berinteraksi dengan ROS.

- `using namespace std;`
  - Baris ini menggunakan namespace "std", yang berarti bahwa pengguna dapat menggunakan anggota-anggota dari namespace standar C++ tanpa menuliskan "std::" sebelumnya.
- `int main(int argc, char **argv) {`  
`ros::init(argc, argv, "robot_node");`
  - Fungsi utama (**main()**) dimulai di sini. **ros::init** digunakan untuk menginisialisasi ROS. Argument **argc** dan **argv** adalah argumen dari baris perintah saat program dijalankan. "robot\_node" adalah nama node ROS yang diberikan kepada program ini.
- `RosbotClass robot;`  
`robot.move();`
  - Dua baris ini membuat objek dari kelas "RosbotClass" dan memanggil metode "move()" pada objek tersebut. Presumtif, kelas "RosbotClass" berisi implementasi untuk menggerakkan robot.
- `float coordinate = robot.get_position(1);`
  - Baris ini mendapatkan posisi dari robot menggunakan metode "get\_position()" dari objek "robot". Nilai 1 yang diberikan adalah suatu argumen yang mengidentifikasi sumbu atau dimensi tertentu dari posisi robot.
- `ROS_INFO_STREAM(coordinate);`
  - Baris ini menggunakan fungsi **ROS\_INFO\_STREAM** dari ROS untuk mencetak informasi ke konsol ROS. Dalam hal ini, itu mencetak nilai koordinat yang telah diperoleh sebelumnya.
- `return 0;`  
`}`
  - Program diakhiri dengan pernyataan **return 0;**, menandakan bahwa program telah berakhir dengan sukses.

Paket `c_scripts` ini sudah memiliki instruksi untuk mengkompilasi program yang baru saja kita edit ketika mengkompilasi seluruh ruang kerja. Untuk melihat perubahannya, kita perlu mengkompilasi ruang kerja catkin dengan perintah berikut:

```
In [ ]:
cd ~/catkin_ws

In [ ]:
catkin_make

In [ ]:
source devel/setup.bash
```

```

user:~/catkin_ws$ catkin_make
Base path: /home/user/catkin_ws
Source space: /home/user/catkin_ws/src
Build space: /home/user/catkin_ws/build
Devel space: /home/user/catkin_ws/devel
Install space: /home/user/catkin_ws/install
####
### Running command: "cmake /home/user/catkin_ws/src -DCATKIN_DEVEL_PREFIX=/home/user/catkin_ws/devel -DCMAKE_INSTALL_PREFIX=/home/u
ser/catkin_ws/install -G Unix Makefiles" in "/home/user/catkin_ws/build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/user/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /home/user/catkin_ws/devel;/home/simulations/public_sim_ws/devel;/opt/ros/kinetic
-- This workspace overlays: /home/user/catkin_ws/devel;/home/simulations/public_sim_ws/devel;/opt/ros/kinetic
-- Using PYTHON_EXECUTABLE: /usr/bin/python
-- Using Debian Python package layout
-- Using empy: /usr/bin/empy
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/user/catkin_ws/build/test_results
-- Found gtest sources under '/usr/src/gmock': gtests will be built
-- Found gmock sources under '/usr/src/gmock': gmock will be built
-- Using Python nosetests: /usr/local/bin/nosetests-2.7
-- catkin 0.7.20
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-----
-- ~~~ traversing 2 packages in topological order:
-- ~~~ - rosbob_control
-- ~~~ - c_scripts
-----
-- +++ processing catkin package: 'rosbob_control'
-- ==> add_subdirectory(cpp_course_repo/rosbob_control)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- +++ processing catkin package: 'c_scripts'
-- ==> add_subdirectory(cpp_course_repo/c_scripts)
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/catkin_ws/build
####
### Running command: "make -j8 -l8" in "/home/user/catkin_ws/build"
####
Scanning dependencies of target sensor_msgs_generate_messages_cpp
Scanning dependencies of target geometry_msgs_generate_messages_py

```

- **cd ~/catkin\_ws**: Perintah ini digunakan untuk berpindah ke direktori Catkin workspace. Di sini, **~/catkin\_ws** adalah path ke workspace tersebut.
- **catkin\_make**: Ini adalah perintah untuk melakukan proses pembuatan (build) proyek ROS di dalam workspace. Perintah ini menghasilkan atau memperbarui binari dan library yang diperlukan untuk proyek pengguna. Hasilnya biasanya disimpan di direktori **devel** di dalam workspace.
- **source devel/setup.bash**: Perintah ini digunakan untuk menjalankan script **setup.bash** yang terdapat di dalam direktori **devel** di dalam workspace. Ini penting karena script tersebut mengatur beberapa variabel lingkungan (environment variables) yang dibutuhkan agar ROS dapat mengenali paket-paket dan library yang telah pengguna bangun.

Jika kompilasi berhasil, maka langkah selanjutnya yaitu menjalankan program dengan melakukan:

```

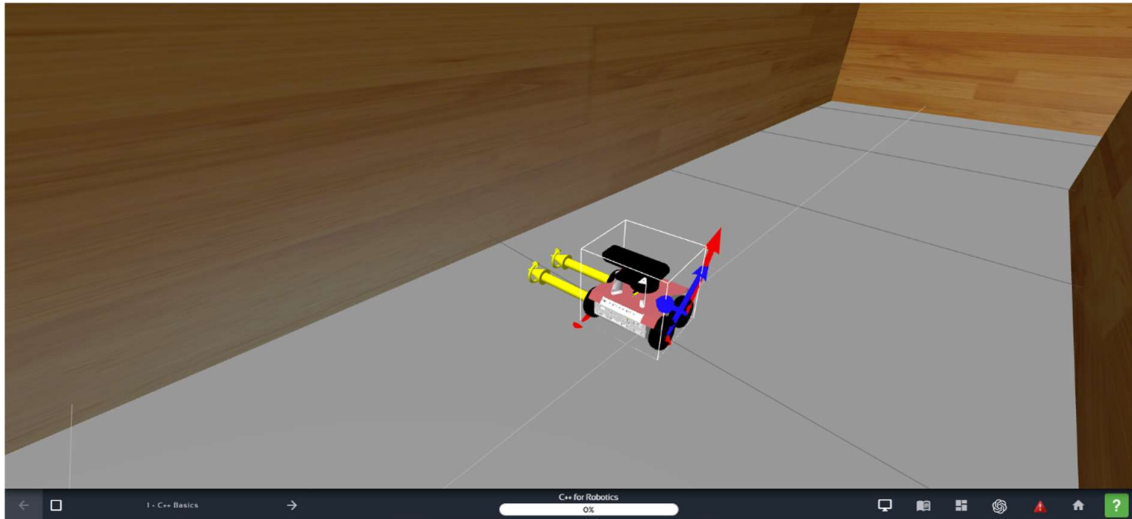
In [ ]:
rosrun c_scripts unit1_exercise

```

Perintah **rosrun c\_scripts unit1\_exercise** digunakan untuk menjalankan sebuah node ROS yang ada dalam paket **c\_scripts** dengan nama **unit1\_exercise**. Saat menjalankan perintah ini, ROS akan mencari node dengan nama tersebut dalam lingkungan yang telah diatur, dan kemudian menjalankannya.

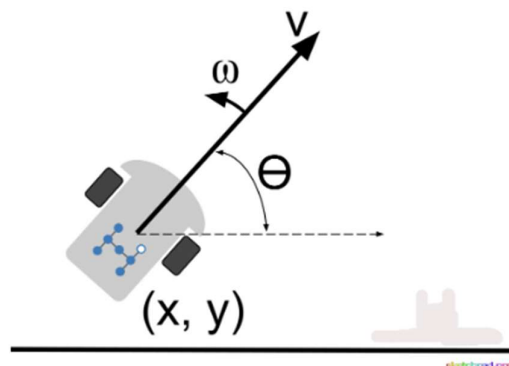
Output :

```
user:~/catkin_ws$ roslaunch c_scripts unit1_exercise
[ INFO] [1700148450.265357315]: Initializing node .....
[ INFO] [1700148454.272238300, 4066.036000000]: 0.892034
user:~/catkin_ws$
```



Pada simulasi robot akan bergerak, Robot ini dapat bergerak berkat motor yang terpasang di setiap rodanya. Robot ini memiliki komputer terpasang yang dapat menyimpan informasi dan menyampaikan perintah kita ke roda motor. Selain itu, roda-roda tersebut memiliki sistem pengukuran yang disebut encoder yang menyimpan berapa kali roda-roda tersebut berputar. Dengan data ini, komputer onboard dapat memperkirakan kecepatan robot, dan dengan menggunakan waktu yang terukur, ia juga dapat memperkirakan data Odometry.

Odometry adalah nilai posisi robot saat ini, yang berarti koordinat  $x$ ,  $y$ ,  $z$  di ruang angkasa, juga orientasinya menurut sumbu dunia, dan kecepatannya. Karena ini adalah robot beroda, maka ia dapat bergerak maju dan mundur, sehingga memiliki kecepatan linier  $v$ , dan dapat berbelok ke kiri atau ke kanan, sehingga memiliki kecepatan sudut  $w$ .



ROS memudahkan untuk mendapatkan data odometri ini dengan Subscriber, sehingga kita dapat menggunakannya dengan program kita. Dalam hal ini, metode kita `get_position()` memiliki sebuah daftar dengan koordinat x,y,z dan kita dapat memilih dengan parameter passing yang mana yang kita butuhkan.

## 6. Part Six :Variables

Variabel dapat dilihat sebagai sebuah wadah yang menyimpan sejumlah data: bisa berupa angka, teks, atau tipe data yang lebih kompleks. Ketika program kita sedang dieksekusi, variabel dapat diakses atau bahkan diubah, yang berarti nilai baru akan diberikan ke variabel tersebut. Dalam bahasa C++, variabel harus dideklarasikan terlebih dahulu dan kemudian diberi nilai berupa angka, kalimat, sekumpulan angka, atau lainnya.

Contoh praktik : Dapatkan koordinat x dan y dari robot dengan memanggil dua kali fungsi `get_posisi()` dan cetak keduanya. Buat robot bergerak dengan memanggil fungsi `move()`, yang tidak membutuhkan parameter, dan tidak memberikan hasil. Dapatkan koordinat x dan y yang baru dari robot.

```
#include "rosbot_control/rosbot_class.h"
#include <ros/ros.h>

using namespace std;

int main(int argc, char **argv) {
    ros::init(argc, argv, "rosbot_node");

    RosbotClass rosbot;
    rosbot.move();

    float x_1 = rosbot.get_position(1);
    float y_1 = rosbot.get_position(2);

    ROS_INFO_STREAM(x_1 << " and " << y_1);

    rosbot.move();

    float x_2 = rosbot.get_position(1);
    float y_2 = rosbot.get_position(2);

    ROS_INFO_STREAM(x_2 << " and " << y_2);

    return 0;
}
```

Penjelasan Code :

- `#include "rosbot_control/rosbot_class.h"`  
`#include <ros/ros.h>`
  - Baris ini mengimpor header file yang berisi definisi kelas dan fungsi yang diperlukan untuk mengendalikan robot. Selanjutnya, "ros/ros.h" adalah header file dari ROS yang menyediakan fungsi-fungsi dasar untuk berinteraksi dengan ROS.
- `using namespace std;`
  - Baris ini menggunakan namespace "std", yang berarti bahwa pengguna dapat menggunakan anggota-anggota dari namespace standar C++ tanpa menuliskan "std::" sebelumnya.
- `int main(int argc, char **argv) {`  
`ros::init(argc, argv, "rosbot_node");`
  - Fungsi utama (**main()**) dimulai di sini. **ros::init** digunakan untuk menginisialisasi ROS. Argument **argc** dan **argv** adalah argumen dari baris perintah saat program dijalankan. "rosbot\_node" adalah nama node ROS yang diberikan kepada program ini.
- `RosbotClass rosbot;`  
`rosbot.move();`
  - Dua baris ini membuat objek dari kelas "RosbotClass" dan memanggil metode "move()" pada objek tersebut. Presumtif, kelas "RosbotClass" yang berisi implementasi untuk menggerakkan robot.
- `float x_1 = rosbot.get_position(1);`  
`float y_1 = rosbot.get_position(2);`  
  
`ROS_INFO_STREAM(x_1 << " and " << y_1);`
  - Baris ini mendapatkan posisi robot pada sumbu x (1) dan sumbu y (2) menggunakan metode "get\_position()" dari objek "rosbot". Nilai-nilai ini kemudian dicetak menggunakan **ROS\_INFO\_STREAM**.
- `rosbot.move();`
  - Baris ini memanggil lagi metode "move()", yang berfungsi untuk melakukan pergerakan tambahan pada robot.
- `float x_2 = rosbot.get_position(1);`  
`float y_2 = rosbot.get_position(2);`  
  
`ROS_INFO_STREAM(x_2 << " and " << y_2);`
  - Baris ini kembali mendapatkan posisi robot setelah pergerakan tambahan dan mencetak nilai-nilai tersebut.

- `return 0;`  
}
- Program diakhiri dengan pernyataan **return 0;**, menandakan bahwa program telah berakhir dengan sukses.

Selanjutnya apabila sudah mengisi code di file cpp maka selanjutnya adalah compile dan execute program.

```
user@:~$ cd catkin_ws
user@catkin_ws$ catkin_make
Base path: /home/user/catkin_ws
Source space: /home/user/catkin_ws/src
Build space: /home/user/catkin_ws/build
Devel space: /home/user/catkin_ws/devel
Install space: /home/user/catkin_ws/install

### Running command: "make cmake_check_build_system" in "/home/user/catkin_ws/build"
###
### Running command: "make -j8 -l8" in "/home/user/catkin_ws/build"
###
[ 0%] Built target geometry_msgs_generate_messages_py
[ 0%] Built target actionlib_msgs_generate_messages_eus
[ 0%] Built target nav_msgs_generate_messages_py
[ 0%] Built target nav_msgs_generate_messages_nodejs
[ 0%] Built target nav_msgs_generate_messages_lisp
[ 0%] Built target roscpp_generate_messages_py
[ 0%] Built target roscpp_generate_messages_eus
[ 0%] Built target actionlib_msgs_generate_messages_py
[ 0%] Built target geometry_msgs_generate_messages_eus
[ 0%] Built target roscpp_generate_messages_cpp
[ 0%] Built target roscpp_generate_messages_cpp
[ 0%] Built target std_msgs_generate_messages_cpp
[ 0%] Built target sensor_msgs_generate_messages_nodejs
[ 0%] Built target geometry_msgs_generate_messages_eus
[ 0%] Built target roscpp_generate_messages_eus
[ 0%] Built target roscpp_generate_messages_py
[ 0%] Built target roscpp_generate_messages_lisp
[ 0%] Built target roscpp_generate_messages_nodejs
[ 0%] Built target actionlib_msgs_generate_messages_cpp
[ 0%] Built target roscpp_generate_messages_nodejs
[ 0%] Built target nav_msgs_generate_messages_cpp
[ 0%] Built target std_msgs_generate_messages_nodejs
[ 0%] Built target roscpp_generate_messages_lisp
[ 0%] Built target std_msgs_generate_messages_eus
[ 0%] Built target std_msgs_generate_messages_py
[ 0%] Built target nav_msgs_generate_messages_eus
[ 0%] Built target geometry_msgs_generate_messages_lisp
[ 0%] Built target std_msgs_generate_messages_lisp
[ 12%] Built target robot_control
```

Output :

```
user@~/catkin_ws$ rosrn c scripts unit1_exercise
[ INFO] [1700149062.146300552]: Initializing node .....
[ INFO] [1700149066.384959899, 4624.557000000]: 1.95514 and -0.0317848
[ INFO] [1700149068.615817638, 4626.557000000]: 3.01529 and -0.108287
user@~/catkin_ws$
```

- **[ INFO] [1700149062.146300552]: Initializing node** : Ini adalah pesan informasi (INFO) dari ROS, memberi tahu bahwa node sedang diinisialisasi. Waktu dalam kurung kurawal menunjukkan timestamp atau waktu sistem di mana pesan ini dibuat.
- **[ INFO] [1700149066.384959899, 4624.557000000]: 1.95514 and -0.0317848** : Ini adalah pesan informasi lainnya. Angka dalam kurung kurawal menunjukkan timestamp atau waktu sistem ketika pesan ini dibuat. Pada kondisi ini, tampaknya node telah mengeksekusi suatu operasi atau fungsi yang memberikan nilai 1.95514 untuk sumbu x dan -0.0317848 untuk sumbu y.
- **[ INFO] [1700149068.615817638, 4626.557000000]: 3.01529 and -0.108287** : Pesan informasi lainnya dengan timestamp atau waktu sistem yang berbeda. Kembali, node telah melakukan operasi atau fungsi yang memberikan nilai 3.01529 untuk sumbu x dan -0.108287 untuk sumbu y.



Output mencerminkan aktivitas dari node termasuk inisialisasi, pergerakan robot, dan pengambilan posisi robot pada sumbu x dan y setelah beberapa operasi tertentu.

## 7. Part Seven : Data Types

Pada sebelumnya pengguna sudah dapat membuat dan memanipulasi variabel dengan mudah, tetapi C++ bisa sangat sulit untuk digunakan jika tidak memahami tipe data mana yang harus kita gunakan dalam setiap kasus. Ada berbagai macam tipe data dalam C++.

- **Booleans**  
Boolean dapat diekspresikan dalam C++ sebagai bool, dan merupakan tipe data logika yang dapat mengambil nilai benar atau salah.
- **Numbers: integers, doubles and floats**  
Di dalam tipe data angka, kita dapat membagi menjadi beberapa jenis angka. Untuk saat ini, mari kita bedakan menjadi bilangan bulat (int), bilangan ganda (double), dan bilangan mengambang (float).

Yang pertama adalah nilai bilangan bulat, misalnya 3. Jika kita memiliki angka dengan desimal, kita akan menggunakan float atau double, tergantung pada berapa desimal presisi yang kita butuhkan. Float dapat memiliki hingga 7 desimal, dan double hingga 15 desimal.

```
In [ ]:
int a = 3;           // This is an integer
float b = 0.23;      // This is a float
double c = 0.225678391734; // This is a double
```



- Strings

Di dalam tipe data teks, kita dapat membaginya menjadi dua jenis yang berbeda: karakter (char) dan string (string). Dalam bahasa ini, semuanya harus direpresentasikan di dalam tanda kutip ganda (" ").

Yang pertama adalah urutan karakter yang harus memiliki panjang yang tetap, misalnya "hello". Dalam hal ini kita harus menginisialisasi variabel sebagai karakter dengan panjang 6, satu lagi dari huruf yang dimilikinya.

```
char d[6] = "hello"
```

Array karakter **d** yang di deklarasikan memiliki panjang 6, bukan karena jumlah karakter kata "hello" adalah 6, tetapi karena array tersebut membutuhkan satu elemen tambahan untuk menampung karakter null-terminator ('\0'). Karakter null-terminator ditambahkan secara otomatis pada akhir string untuk menandai akhir dari string tersebut.

```
string e = "developer";
```

String dapat disubskrip atau diindeks. Karakter pertama dari sebuah string memiliki indeks 0.

Untuk code lengkap menampilkan hello developer

```
/*Header files*/
#include <iostream>
#include <list>
#include <map>
#include <string>

using namespace std;

int main() {

/*Deklarasi dan Inisialisasi String dan Array*/
    char d[6] = "hello";

    string e = "developer";

/*Mencetak Karakter Pertama dari Array dan String*/
    cout << d[0] << endl;
    cout << e[4] << endl;

    string f = ",";
    string g = "!";

/*Menggabungkan String*/
```

```

    cout << d + f + e + g << endl;

    return 0;
}

```

Output :

```

user:~/catkin_ws/src/cpp_course_repo/utilities$ g++ -std=c++11 name.cpp -o name_compiled
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
h
l
hello, developer!
user:~/catkin_ws/src/cpp_course_repo/utilities$ 

```

`cout << d[0] << endl;` : akan menampilkan index 0 pada variabel d, yaitu 'h','e','l','l','o'. maka index 0 sama dengan h.

`cout << e[4] << endl;` : akan menampilkan index 4 pada variabel e, yaitu 'd','e',' ','v',' ','e','l',' ','o',' ','p',' ','e','r'. maka index 4 sama dengan l.

`string f = " ,";`

`string g = "!";`

`cout << d + f + e + g << endl;` : untuk menggabungkan variabel string. Maka output menjadi hello, developer!

- Lists

Dalam C++, daftar adalah urutan variabel yang memiliki tipe yang sama. Untuk menginisiasinya, kita harus menentukan tipe variabel yang ada di dalamnya. Sebagai contoh, untuk sebuah daftar nilai bilangan bulat dan nilai string:

```

list<int> numbers_list({1,10,100,1000});
list<string> vocals_list( {"a","e","i","o","u"} );

```

Ketidaknyamanan dengan daftar dalam C++ adalah bahwa daftar tidak mudah dicetak seperti dalam bahasa lain. Dalam hal ini kita perlu mengulang semua item dalam daftar untuk mencetaknya satu per satu.

```

for (int val : numbers_list)           // Loop
    cout << val << " ";               // Print function

for (string val : vocals_list)          // Loop
    cout << val << " ";              // Print function

```

Daftar sangat berguna, karena menempati ruang memori yang dapat dimodifikasi. Daftar ini memiliki fungsi bawaan untuk, misalnya, menambahkan item baru di awal daftar, atau di akhir daftar:

Untuk Code Lengkap

```
/*Header files*/
#include <iostream>
#include <list>
#include <map>
#include <string>

using namespace std;

int main() {

/*Mendeklarasikan dan Menginisialisasi List*/
    list<int> numbers_list({1, 10, 100, 1000});
    list<string> vocals_list({"a", "e", "i", "o", "u"});

/*Looping dan Mencetak List Angka:*/
    for (int val : numbers_list)
        cout << val << " ";

/*Looping dan Mencetak List Huruf Vokal*/
    for (string val : vocals_list)
        cout << val << " ";

    return 0;
}
```

Output :

```
user:~/catkin_ws/src/cpp_course_repo/utilities$ g++ -std=c++11 name.cpp -o name_compiled
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
1 10 100 1000 a e i o u user:~/catkin_ws/src/cpp_course_repo/utilities$
```

`g++ -std=c++11 name.cpp -o name_compiled` : digunakan untuk mengkompilasi program C++ yang terdapat dalam file "name.cpp" dengan menggunakan compiler GNU C++ (g++), dengan menentukan bahwa program mengikuti standar C++11.

`./name_compiled` : Untuk melihat hasil eksekusi dari program.

Pesan output yang di berikan, yaitu "1 10 100 1000 a e i o u", menunjukkan bahwa program telah berjalan dengan sukses dan berhasil mencetak elemen-elemen dari list **numbers\_list** dan **vocals\_list** ke layar.

- Dictionaries

Kamus dalam C++ disebut peta, dan merupakan sebuah wadah nilai yang diindeks oleh kunci. Ini berarti kamus ini menyimpan dua jenis informasi: kunci dan nilai.

Sebagai contoh, jika kita ingin menyimpan nama-nama karakter serial TV dan juga berapa banyak episode yang muncul, kita tidak memerlukan daftar dengan nama-nama dan daftar dengan jumlah episode, kita hanya memerlukan kamus di mana kuncinya adalah nama-nama dan nilainya adalah jumlah episode:

```
{ "Dolores": 30, "Maeve": 27, "Theresa":6, "Clementine":11 }
```

Untuk menginisialisasinya, kita perlu memanggil map, dan menentukan tipe data dari kunci dan nilai:

```
map<string, int> girls_dictionary;
```

Di sini kita membuat kamus bernama girls\_dictionary, di mana kuncinya berupa string dan nilainya berupa bilangan bulat. Untuk memasukkan data ke dalam kamus ini, kita dapat memanggil setiap kunci dan memberikan nilai, satu per satu:

```
girls_dictionary["Dolores"] = 30;
girls_dictionary["Maeve"] = 27;
girls_dictionary["Theresa"] = 6;
girls_dictionary["Clementine"] = 11;

for (auto item : girls_dictionary)
    cout << item.first << " appears in " << item.second << " episodes\n";
```

Untuk code lengkap nya

```
/*Header files*/
#include <iostream>
#include <list>
#include <map>
#include <string>

using namespace std;

int main() {

/*Membuat dan Menginisialisasi Map*/
    map<string, int> girls_dictionary;
```

```

girls_dictionary["Dolores"] = 30;
girls_dictionary["Maeve"] = 27;
girls_dictionary["Theresa"] = 6;
girls_dictionary["Clementine"] = 11;

/*Looping dan Mencetak Isi Map*/
for (auto item : girls_dictionary)
    cout << item.first << " appears in " << item.second << " episodes\n";

return 0;
}

```

Program ini menggunakan beberapa header file standar C++ dan STL. `<iostream>` untuk fungsi input dan output, `<list>` dan `<map>` untuk menggunakan struktur data **list** dan **map**, dan `<string>` untuk bekerja dengan string.

Di sini, program membuat objek **girls\_dictionary** yang merupakan map dengan kunci bertipe string dan nilai bertipe int. Kemudian, ditambahkan beberapa pasangan kunci-nilai ke dalam map menggunakan operator `[]`.

Menggunakan loop **for** dan iterasi dengan variabel **item**, program ini mencetak setiap pasangan kunci-nilai dari map ke konsol. **item.first** memberikan kunci (nama karakter), dan **item.second** memberikan nilai (jumlah episode).

Output :

```

user:~/catkin_ws/src/cpp_course_repo/utilities$ g++ -std=c++11 name.cpp -o name_compiled
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
Clementine appears in 11 episodes
Dolores appears in 30 episodes
Maeve appears in 27 episodes
Theresa appears in 6 episodes
user:~/catkin_ws/src/cpp_course_repo/utilities$ 

```

Program ini menciptakan map **girls\_dictionary**, mengisi dengan beberapa pasangan kunci-nilai, dan kemudian mencetak isi map tersebut ke layar. Outputnya akan menunjukkan nama karakter dan jumlah episode di mana karakter tersebut muncul.

Contoh praktik : membuat sebuah kamus yang akan menyimpan posisi x dari robot dari waktu ke waktu, dan mencetaknya di dalam shell. Pertama, modifikasi program unit1\_latihan.cpp dari latihan sebelumnya untuk mendapatkan koordinat x dari robot dengan memanggil metode get\_posisi(), dan juga mendapatkan waktu simulasi dengan memanggil metode get\_waktu(). Perhatikan bahwa get\_position() dan get\_time() adalah metode dari RosbotClass dan pengguna perlu mengetik rosbob.get\_time() dan bukan hanya get\_time() yang akan pengguna tulis untuk memanggil fungsi biasa. Kemudian, buatlah robot bergerak dengan memanggil metode move() yang juga merupakan bagian dari kelas RosbotClass. Ulangi langkah 1: ambil posisi x dan cap waktu.

Alih-alih mencetaknya di dalam shell, Pengguna akan menginisialisasi sebuah kamus dan menyimpan waktu yang diperoleh sebagai sebuah kunci, dan posisi x sebagai sebuah nilai. Lakukan untuk semua nilai x yang diperoleh, 1 kali, 2 kali, .., sebanyak yang diinginkan. Cetak kamus dengan kode yang disediakan di bagian Kamus.

Isi code di file unit1\_exercise.cpp

```
/*Header Files*/
#include "rosbot_control/rosbot_class.h"
#include <iostream>
#include <list>
#include <map>
#include <ros/ros.h>
#include <string>

using namespace std;

int main(int argc, char **argv) {

    /*Inisialisasi ROS*/
    ros::init(argc, argv, "rosbot_node");

    /*Membuat Objek RosbotClass dan Pergerakan*/
    RosbotClass rosbob;
    rosbob.move();

    /*Mendapatkan Posisi dan Waktu Pertama*/
    float x_0 = rosbob.get_position(1);
    double t_0 = rosbob.get_time();

    /*Mencetak Posisi dan Waktu Pertama ke Konsol*/
    ROS_INFO_STREAM(x_0 << " and " << t_0);
    rosbob.move();

    /*Pergerakan Lagi dan Mendapatkan Posisi dan Waktu Kedua*/
    float x_1 = rosbob.get_position(1);
```

```

double t_1 = rosbob.get_time();
ROS_INFO_STREAM(x_1 << " and " << t_1);

/*Membuat dan Mengisi Map x_t_dictionary*/
map<double, float> x_t_dictionary;
x_t_dictionary[t_0] = x_0;
x_t_dictionary[t_1] = x_1;

/*Looping dan Mencetak Isi Map*/
for (auto item : x_t_dictionary) {
    ROS_INFO_STREAM("Time " << item.first << ", position " << item.second
                    << " \n");
}

/*Mengakhiri Program*/
return 0;
}

```

Penjelasan Code :

- `#include "robbob_control/robbob_class.h"`  
`#include <iostream>`  
`#include <list>`  
`#include <map>`  
`#include <ros/ros.h>`  
`#include <string>`
  - Program ini menggunakan beberapa header file standar C++, termasuk header file ROS, dan juga file header **robbob\_class.h** yang berisi definisi dari kelas **RobbotClass**.
- `ros::init(argc, argv, "robbob_node");`
  - Fungsi **ros::init** digunakan untuk menginisialisasi ROS. Argument **argc** dan **argv** adalah argumen dari baris perintah saat program dijalankan. "robbob\_node" adalah nama node ROS yang diberikan kepada program ini.
- `RobbotClass robbob;`  
`robbob.move();`
  - Dua baris ini membuat objek dari kelas **RobbotClass** dan memanggil metode **move()** pada objek tersebut, yang berfungsi untuk menggerakkan robot.
- `float x_0 = robbob.get_position(1);`  
`double t_0 = robbob.get_time();`
  - Dua baris ini mendapatkan posisi dan waktu pertama dari robot menggunakan metode **get\_position()** dan **get\_time()** dari objek **robbob**.

- `ROS_INFO_STREAM(x_0 << " and " << t_0);`
  - Menggunakan `ROS_INFO_STREAM`, program mencetak posisi dan waktu pertama ke konsol ROS.
- `robot.move();`  
`float x_1 = robot.get_position(1);`  
`double t_1 = robot.get_time();`
  - Baris ini memanggil lagi metode **move()** untuk pergerakan tambahan dan kemudian mendapatkan posisi dan waktu kedua dari robot.
- `ROS_INFO_STREAM(x_1 << " and " << t_1);`
  - Menggunakan **`ROS_INFO_STREAM`**, program mencetak posisi dan waktu kedua ke konsol ROS.
- `map<double, float> x_t_dictionary;`  
`x_t_dictionary[t_0] = x_0;`  
`x_t_dictionary[t_1] = x_1;`
  - Program ini membuat objek map **`x_t_dictionary`** yang memiliki kunci bertipe **double** (waktu) dan nilai bertipe **float** (posisi). Kemudian, program mengisi map ini dengan pasangan kunci-nilai yang merepresentasikan waktu dan posisi dari dua pengukuran yang dilakukan sebelumnya.
- `for (auto item : x_t_dictionary) {`  
`ROS_INFO_STREAM("Time " << item.first << ", position " << item.second`  
`<< " \n");`  
`}`
  - Menggunakan loop **for** dan iterasi dengan variabel **item**, program ini mencetak setiap pasangan waktu dan posisi dari map ke konsol ROS.
- `return 0;`
  - Mengindikasikan bahwa program telah berakhir dengan sukses.



Output :

```
user::~$ cd ~/catkin_ws
user::~/catkin_ws$ catkin_make
Base path: /home/user/catkin_ws
Source space: /home/user/catkin_ws/src
Build space: /home/user/catkin_ws/build
Devel space: /home/user/catkin_ws/devel
Install space: /home/user/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/user/catkin_ws/build"
####
####
#### Running command: "make -j8 -l8" in "/home/user/catkin_ws/build"
####
[ 0%] Built target sensor_msgs_generate_messages_eus
[ 0%] Built target geometry_msgs_generate_messages_py
[ 0%] Built target actionlib_msgs_generate_messages_nodejs
[ 0%] Built target sensor_msgs_generate_messages_lisp
[ 0%] Built target actionlib_msgs_generate_messages_eus
[ 0%] Built target sensor_msgs_generate_messages_py
[ 0%] Built target sensor_msgs_generate_messages_cpp
[ 0%] Built target actionlib_msgs_generate_messages_lisp
[ 0%] Built target nav_msgs_generate_messages_py
[ 0%] Built target actionlib_msgs_generate_messages_py
[ 0%] Built target rosgraph_msgs_generate_messages_eus
[ 0%] Built target nav_msgs_generate_messages_nodejs
[ 0%] Built target geometry_msgs_generate_messages_eus
[ 0%] Built target roscpp_generate_messages_py
[ 0%] Built target nav_msgs_generate_messages_lisp
[ 0%] Built target roscpp_generate_messages_cpp
[ 0%] Built target rosgraph_msgs_generate_messages_cpp
[ 0%] Built target std_msgs_generate_messages_cpp
[ 0%] Built target roscpp_generate_messages_eus
[ 0%] Built target geometry_msgs_generate_messages_cpp
[ 0%] Built target sensor_msgs_generate_messages_nodejs
[ 0%] Built target roscpp_generate_messages_nodejs
[ 0%] Built target actionlib_msgs_generate_messages_cpp
[ 0%] Built target rosgraph_msgs_generate_messages_py
[ 0%] Built target rosgraph_msgs_generate_messages_nodejs
[ 0%] Built target nav_msgs_generate_messages_cpp
[ 0%] Built target rosgraph_msgs_generate_messages_lisp
[ 0%] Built target std_msgs_generate_messages_nodejs
[ 0%] Built target roscpp_generate_messages_lisp
[ 0%] Built target nav_msgs_generate_messages_eus
```

```
user::~/catkin_ws$ source devel/setup.bash
user::~/catkin_ws$ rosrunc scripts unit1_exercise
[ INFO] [1700159949.992399993]: Initializing node .....
[ INFO] [1700159954.003616292, 106.854000000]: 0.915115 and 106.853
[ INFO] [1700159956.012270414, 108.854000000]: 1.98343 and 108.854
[ INFO] [1700159956.012332752, 108.854000000]: Time 106.853, position 0.915115
[ INFO] [1700159956.012374083, 108.854000000]: Time 108.854, position 1.98343
user::~/catkin_ws$
```

**cd ~/catkin\_ws** : digunakan untuk pindah ke direktori Catkin workspace.

**catkin\_make** : digunakan untuk membangun (build) proyek ROS di dalam Catkin workspace. Ini mengompilasi semua paket ROS yang ada di dalam workspace tersebut. Proses ini menghasilkan eksekutabel, pustaka, dan file lainnya yang diperlukan untuk menjalankan dan mengintegrasikan paket-paket tersebut.

**source devel/setup.bash** : digunakan untuk mengatur lingkungan kerja ROS setelah pengguna berhasil melakukan kompilasi dengan **catkin\_make**.

**rosrunc scripts unit1\_exercise** : digunakan untuk menjalankan node ROS yang disebut **unit1\_exercise** dari paket **c\_scripts**.

[ INFO] [1700159949.992399993]: **Initializing node .....** : Ini adalah pesan informasi yang memberi tahu bahwa node telah diinisialisasi. Nomor yang panjang seperti **1700159949.992399993** adalah waktu dalam detik sejak epoch (waktu referensi), dan pesan selanjutnya dapat memberikan timestamp yang lebih spesifik.

[ INFO] [1700159954.003616292, 106.854000000]: **0.915115 and 106.853** : Ini adalah pesan informasi yang memberikan beberapa data numerik. Potongan ini sepertinya mencetak posisi (0.915115) dan waktu (106.853) dari suatu peristiwa.

[ INFO] [1700159956.012270414, 108.854000000]: **1.98343 and 108.854** : Pesan informasi lainnya yang sepertinya mencetak posisi (1.98343) dan waktu (108.854) dari peristiwa berikutnya.

[ INFO] [1700159956.012332752, 108.854000000]: **Time 106.853, position 0.915115** : Pesan informasi yang mencetak waktu (106.853) dan posisi (0.915115) dari peristiwa sebelumnya.

[ INFO] [1700159956.012374083, 108.854000000]: **Time 108.854, position 1.98343** : Pesan informasi yang mencetak waktu (108.854) dan posisi (1.98343) dari peristiwa sebelumnya.

Simulasi Robot :



Pada simulasi, node diatas berfungsi untuk melakukan pengukuran atau pemantauan posisi dan waktu, dan mencetak informasi ini ke layar melalui **ROS\_INFO\_STREAM**. Pesan timestamp menunjukkan waktu relatif terhadap epoch, dan kemungkinan besar, node ini terus mengukur dan mencetak informasi tersebut pada interval waktu tertentu. Informasi yang tepat ini bergantung pada implementasi dari node **unit1\_exercise**.

## 8. Part Eight : I/O Functions

- **Print**  
Pada bagian ini, pada dasarnya digunakan untuk menulis ke dalam output standar program. Hal ini sangat berguna untuk berkomunikasi dengan pengguna yang berinteraksi dengan program, untuk memberitahukan apa yang

sedang terjadi, tetapi juga sangat berguna ketika kita ingin men-debug program kita sendiri.

Dalam C++ ada beberapa cara untuk mengimplementasikan sebuah print, tetapi dalam bab ini kita telah melihat dua di antaranya:

- Printf
- cout

Yang pertama adalah fungsi yang membutuhkan tipe variabel yang akan dicetak. Sebagai contoh, jika kita ingin mencetak sebuah bilangan bulat, kita menspesifikasikannya dengan simbol %i, jika sebuah float dengan %f, dan jika sebuah string dengan %s. Dalam kasus khusus pencetakan string, string perlu dikonversi menjadi karakter dengan fungsi c\_str().

#### Code I/O Function with Print

```
#include <iostream>
using namespace std;

int main() {

    int a = 42;
    printf("Value a is %i \n", a); // Print an integer
    float b = 3.1415;
    printf("Value a is %f \n", b); // Print a float
    string word = "Hey you!";
    printf("- %s \n", word.c_str()); // Print a string

    return 0;
}
```

Output :

```
user:~$ cd ~/catkin_ws/src/cpp_course_repo/utilities/
user:~/catkin_ws/src/cpp_course_repo/utilities$ g++ -std=c++11 name.cpp -o name_compiled
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
Value a is 42
Value a is 3.141500
- Hey you!
```

Code I/O Function with cout

```
#include <iostream>
using namespace std;

int main() {

    cout << "Nice!" << endl;
    cout << "Let's get back to work then" << endl;

    return 0;
}
```

Output :

```
user:~/catkin_ws/src/cpp_course_repo/utilities$ g++ -std=c++11 name.cpp -o name_compiled
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
Nice!
Let's get back to work then
user:~/catkin_ws/src/cpp_course_repo/utilities$
```

- Input  
Dalam pemrograman, interaksi dengan pengguna atau lingkungan luar seringkali diperlukan. Fungsi input/output (I/O) memungkinkan program untuk menerima input dari pengguna dan menghasilkan output. Contoh sederhana dalam bahasa C++ adalah menggunakan fungsi **cout** untuk output dan **cin** untuk input. Berikut adalah contoh singkat:

Contoh Program Input umur (integer)

```
//input Umur (integer)
#include <iostream>
using namespace std;

int main() {

    int x;

    /*Menampilkan pesan kepada pengguna*/
    cout << "Apa Kabarmu Mochamad Phillia? ";

    /*Membaca input umur dari pengguna dan menyimpannya dalam variabel x*/
    cin >> x;

    /*Menampilkan pesan yang mencakup umur yang dimasukkan oleh pengguna*/
    cout << "Kamu berumur " << x << " Tahun";

    return 0;
}
```

Output :

```
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
Apa Kabarmu Mochamad Phillia? 21
Kamu berumur 21 Tahunuser:~/catkin_ws/src/cpp_course_repo/utilities$
```

Contoh Program Input nama (string)

```
//Input nama (string)
#include <iostream>
using namespace std;

int main() {

    string name;
    /*Menampilkan pesan kepada pengguna*/
    cout << "Siapa Nama Mu? ";

    /*Membaca input umur dari pengguna dan menyimpannya dalam variabel name*/
    cin >> name;

    /*Menampilkan pesan yang mencakup umur yang dimasukkan oleh pengguna*/
    cout << "Senang bertemu dengan mu " << name << "!";

    return 0;
}
```

Output :

```
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
Siapa Nama Mu? Phillia
Senang bertemu dengan mu Phillia!user:~/catkin_ws/src/cpp_course_repo/utilities$
```

- Namespaces  
C++ memiliki fitur khusus yang disebut dengan namespaces. Fitur ini memungkinkan pengguna untuk mengelompokkan entitas bernama ke dalam satu ruang lingkup. Dengan begitu, mereka hanya akan memiliki arti di dalam namespace tersebut. Selain itu, kita juga bisa mengulang nama variabel, tetapi selalu di dalam namespace yang berbeda.

Dalam praktiknya, kita tidak akan mengembangkan salah satunya di sini. Kita hanya akan menjelaskan salah satu yang telah kita gunakan dalam bab ini: std

```
using namespace std;
```

Baris ini memungkinkan pengguna untuk menggunakan semua nama di dalam namespace std, yang sejauh ini adalah:

- String  
Adalah tipe data yang digunakan untuk menyimpan dan memanipulasi teks atau urutan karakter.
- Cout  
Adalah objek output standar dalam C++, digunakan untuk menampilkan (atau mengirim) output ke layar atau perangkat keluaran lainnya.
- Cin  
Adalah objek input standar dalam C++, digunakan untuk menerima input dari pengguna melalui keyboard atau perangkat masukan lainnya.
- Endl  
Adalah manipulator output yang digunakan untuk memindahkan kursor ke baris berikutnya dan membersihkan buffer output.

## 9. Part Nine : Operators

Dalam C++, operator digunakan untuk melakukan operasi pada variabel dan nilai. Kita dapat membagi operator ke dalam kelompok dasar berikut ini:

- Operator Aritmatika

Operator	Name	Example
+	Addition	1 + 1 = 2
-	Substraction	2 - 1 = 1
*	Multiplication	2 * 2 = 4
/	Division	5 / 2 = 2
%	Modulus	5 % 2 = 1

Contoh Code

```
// Arithmetic Operators
#include <iostream>

using namespace std;

int main() {
    int a = 4;
```

```

int b = 3;

cout << a + b << endl;
cout << a - b << endl;
cout << a * b << endl;
cout << a / b << endl;
cout << a % b << endl;

return 0;
}

```

Output :

```

user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
7
1
12
1
1

```

- Operator Penugasan

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

Contoh Code

```

// Assignment Operators
#include <iostream>

using namespace std;

int main() {
    int x = 5;

    cout << (x += 2) << endl;
    cout << (x -= 2) << endl;
    cout << (x *= 2) << endl;
    cout << (x /= 2) << endl;
    cout << (x %= 2) << endl;
}

```

```
    return 0;
}
```

Output :

```
user:~/catkin_ws/src/cpp_course_repo/utilities$ g++ -std=c++11 name.cpp -o name_compiled
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
7
5
10
5
1
user:~/catkin_ws/src/cpp_course_repo/utilities$
```

- Operator Perbandingan

Operator	Means	Same As
==	Equal	5 == 5
!=	Not Equal	4 != 5
>	Greater than	5 > 4
<	Less than	4 < 5
>=	Greater than or equal to	5 >= 4
<=	Less than or equal to	4 <= 5

Contoh Code

```
// Comparison Operators
#include <iostream>

using namespace std;

int main() {
    int y = 9;
    int z = 8;

    cout << (y == z) << endl;
    cout << (y > z) << endl;
    cout << (y < z) << endl;
    cout << (y >= z) << endl;
    cout << (y <= z) << endl;

    return 0;
}
```



Output :

```
user:~/catkin_ws/src/cpp_course_repo/utilities$ g++ -std=c++11 name.cpp -o name_compiled
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
0
1
0
1
0
user:~/catkin_ws/src/cpp_course_repo/utilities$
```

- Operator Logika

Operator	Description
&&	AND
	OR
!	NOT
!=	NOT EQUAL TO
&	BITWISE AND
	BITWISE OR
^	BITWISE XOR
&=	AND EQUAL
=	OR EQUAL
^=	XOR EQUAL

Contoh Code

```
// Logical Operators
#include <iostream>

int main() {
    // Mendeklarasikan dua variabel integer
    int nilai1, nilai2;

    // Meminta pengguna untuk memasukkan dua nilai
    std::cout << "Masukkan nilai pertama: ";
    std::cin >> nilai1;

    std::cout << "Masukkan nilai kedua: ";
    std::cin >> nilai2;

    // Menggunakan operator logika untuk mengevaluasi kondisi
    if (nilai1 > 0 && nilai2 > 0) {
        std::cout << "Keduanya merupakan nilai positif." << std::endl;
    } else if (nilai1 > 0 || nilai2 > 0) {
        std::cout << "Salah satunya merupakan nilai positif." << std::endl;
    } else {
        std::cout << "Keduanya bukan nilai positif." << std::endl;
    }

    return 0;
}
```

Output :

```
user:~/catkin_ws/src/cpp_course_repo/utilities$ g++ -std=c++11 name.cpp -o name_compiled
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
Masukkan nilai pertama: 5
Masukkan nilai kedua: 2
Keduanya merupakan nilai positif.
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
Masukkan nilai pertama: -5
Masukkan nilai kedua: 3
Salah satunya merupakan nilai positif.
user:~/catkin_ws/src/cpp_course_repo/utilities$ ./name_compiled
Masukkan nilai pertama: -9
Masukkan nilai kedua: -6
Keduanya bukan nilai positif.
user:~/catkin_ws/src/cpp_course_repo/utilities$
```

Contoh Praktik : Program ini sekarang harus melakukan tiga aksi, Pertama, dapatkan ukuran waktu dan posisi x dengan fungsi `get_time()` dan `get_position()`, dan simpan di dalam variabel. Kemudian buatlah robot bergerak, dan dapatkan lagi waktu dan posisi x dengan menyimpannya dalam variabel yang berbeda. Kedua, hitung kecepatan rata-rata robot pada periode tersebut, dengan menggunakan rumus  $[v(m/s) = (x_1 - x_0) / (t_1 - t_0)]$ .

Ketiga, cetak (menggunakan `ROS_INFO_STREAM`) nilai yang benar jika kecepatan rata-rata lebih rendah dari 1 m/s

```
// Header Files
#include "rosbot_control/rosbot_class.h"
#include <iostream>
#include <ros/ros.h>

using namespace std;

int main(int argc, char **argv) {

    // Inisialisasi ROS
    ros::init(argc, argv, "rosbot_node");

    // Objek RosbotClass
    RosbotClass rosbot;
    rosbot.move();

    // Mendapatkan Posisi dan Waktu Pertama
    float x_0 = rosbot.get_position(1);
    double t_0 = rosbot.get_time();

    rosbot.move();

    // Pergerakan Lagi dan Mendapatkan Posisi dan Waktu Kedua
    float x_1 = rosbot.get_position(1);
    double t_1 = rosbot.get_time();
```

```

// Menghitung dan Mencetak Kecepatan
float speed = (x_1 - x_0) / (t_1 - t_0);
ROS_INFO_STREAM("Speed is lower than 1 m/s? " << (speed <= 1.0) << "\n");

// Mengakhiri Program
return 0;
}

```

Penjelasan Code :

- `#include "rostopic/rostopic.h"`  
`#include <iostream>`  
`#include <rostopic/rostopic.h>`
  - Program ini menggunakan beberapa header file termasuk header file ROS (`rostopic/rostopic.h`) dan juga file header `rostopic.h` yang berisi definisi dari kelas `RostopicClass`.
- `rostopic::init(argc, argv, "rostopic_node");`

`RostopicClass rostopic;`

`rostopic.move();`

- Fungsi **`rostopic::init`** digunakan untuk menginisialisasi ROS. Argument **`argc`** dan **`argv`** adalah argumen dari baris perintah saat program dijalankan. `"rostopic_node"` adalah nama node ROS yang diberikan kepada program ini.
- Dua baris ini menginisialisasi ROS dan membuat objek dari kelas **`RostopicClass`**, kemudian melakukan gerakan awal dengan memanggil metode **`move()`** pada objek tersebut.
- `float x_0 = rostopic.get_position(1);`  
`double t_0 = rostopic.get_time();`
  - Program ini mendapatkan posisi dan waktu pertama dari robot menggunakan metode **`get_position()`** dan **`get_time()`** dari objek **`rostopic`**.
- `rostopic.move();`  
`float x_1 = rostopic.get_position(1);`  
`double t_1 = rostopic.get_time();`
  - Baris ini memanggil lagi metode **`move()`** untuk pergerakan tambahan dan kemudian mendapatkan posisi dan waktu kedua dari robot.

- $\text{float speed} = (x\_1 - x\_0) / (t\_1 - t\_0);$   
`ROS_INFO_STREAM("Speed is lower than 1 m/s? " << (speed <= 1.0) << "\n");`
  - Program ini menghitung kecepatan dengan membagi selisih posisi dengan selisih waktu. Kemudian, menggunakan **ROS\_INFO\_STREAM**, program mencetak apakah kecepatan tersebut lebih rendah dari atau sama dengan 1 m/s ke layar ROS.
- `return 0;`
  - Mengindikasikan bahwa program telah berakhir dengan sukses.

Output :

```
user:~/catkin_ws$ rosrunc scripts unit1_exercise
[ INFO] [1700165766.483288714]: Initializing node .....
[ INFO] [1700165772.556008113, 133.964000000]: Speed is lower than 1 m/s? 1
user:~/catkin_ws$
```

Output yang diberikan menunjukkan bahwa program berhasil dijalankan dan memberikan informasi bahwa kecepatan robot, berdasarkan perubahan posisi dan waktu yang diukur, lebih rendah dari 1 m/s.

**[ INFO] [1700165766.483288714]: Initializing node ..... :** Pesan ini memberitahu bahwa node telah diinisialisasi. Waktu seperti 1700165766.483288714 adalah timestamp yang memberikan informasi tentang waktu terkait dengan epoch.

**[ INFO] [1700165772.556008113, 133.964000000]: Speed is lower than 1 m/s? 1 :** Pesan ini memberikan hasil perhitungan kecepatan dan memberitahu bahwa kecepatan tersebut lebih rendah dari 1 m/s. Angka 1 di bagian akhir pesan menunjukkan bahwa kondisi ( $\text{speed} \leq 1.0$ ) benar (true).

Simulasi Robot :

