EMC 2013
C++ and ROS

Sjoerd van den Dries

Eindhoven University of Technology

Department of Mechanical Engineering

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

September 11, 2013

C++

- We will use C++ as programming language
- One of the two core ROS languages
  - Packages roscpp and roslib
- C++ is object-oriented C
  - “C with Classes”
  - Encapsulate data and functionality within objects
- Many tutorials available, *e.g.*:
  - http://www.cplusplus.com/doc/tutorial
  - MIT’s Introduction to C++

- C++ is a compiled language

C++

- C++ is a compiled language
- You write source code
  - A set of *'human-readable'* instructions

C++

- C++ is a compiled language
- You write source code
  - A set of *'human-readable'* instructions
- which is compiled by a compiler into

C++

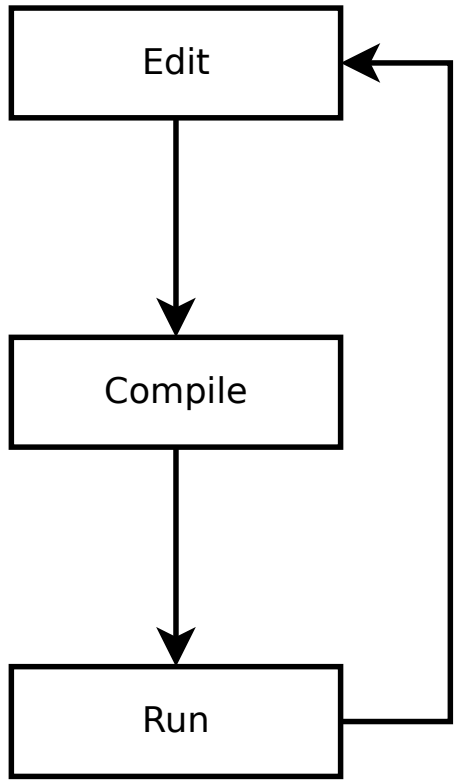
- C++ is a compiled language
- You write source code
  - A set of *'human-readable'* instructions
- which is compiled by a compiler into
- binary code
  - A set of *'machine-readable'* instructions

C++

- C++ is a compiled language
- You write source code
  - A set of *'human-readable'* instructions
- which is compiled by a compiler into
- binary code
  - A set of *'machine-readable'* instructions
- Advantage: fast at run-time
- Disadvantage: slow at design-time

C++

- C++ is a compiled language
- You write source code
  - A set of *'human-readable'* instructions
- which is compiled by a compiler into
- binary code
  - A set of *'machine-readable'* instructions
- Advantage: fast at run-time
- Disadvantage: slow at design-time



Hello World

hello.cpp

```
#include <iostream>

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

Hello World

hello.cpp

```
#include <iostream>

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

- Compile: `g++ hello.cpp -o hello`

Hello World

hello.cpp

```
#include <iostream>

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

- Compile: `g++ hello.cpp -o hello`
- Run: `./hello`

Hello World

hello.cpp

```cpp
#include <iostream>

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

- Compile: `g++ hello.cpp -o hello`
- Run: `./hello`

```
Hello World
```

Hello World

hello.cpp

```cpp
#include <iostream>

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

Hello World

hello.cpp

```cpp
#include <iostream>

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

- Execution starts with main()

Hello World

hello.cpp

```cpp
#include <iostream>

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

- Execution starts with main()
- Executes instructions one by one
  - Each separated by a semi-colon ;

Hello World

hello.cpp

```cpp
#include <iostream>

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

- Execution starts with main()
- Executes instructions one by one
  - Each separated by a semi-colon ;

Hello World

hello.cpp

```cpp
#include <iostream>

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

- Execution starts with main()
- Executes instructions one by one
  - Each separated by a semi-colon ;

Hello World

hello.cpp

```cpp
#include <iostream>

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

- Execution starts with main()
- Executes instructions one by one
  - Each separated by a semi-colon ;
- Stops after last instruction

Hello World

hello.cpp

```cpp
#include <iostream>

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

- Execution starts with main()
- Executes instructions one by one
  - Each separated by a semi-colon ;
- Stops after last instruction
- Flow of control

Hello World

hello.cpp

```cpp
#include <iostream>

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

Hello World

hello.cpp

```
#include <iostream>

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

- #include includes other libraries
  - iostream defines std::cout and std::endl

Hello World

hello.cpp

```cpp
#include <iostream>

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

- #include includes other libraries
  - iostream defines std::cout and std::endl
  - #include in fact just copies the code into your program

Hello World

hello.cpp

```
#include <iostream>

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

- #include includes other libraries
  - iostream defines std::cout and std::endl
  - #include in fact just copies the code into your program
- std::cout « ... « std::endl prints to screen

Hello World

hello.cpp

```
#include <iostream>

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

- #include includes other libraries
  - iostream defines std::cout and std::endl
  - #include in fact just copies the code into your program
- std::cout « ... « std::endl prints to screen
- main() must return an error code
  - 0 means no errors

Expressions and Variables

Expressions and Variables

- Expression
  - A statement that has a value
  - For example:
    - 3 * 5 + 7
  - is an expressions that evaluates to:
    - 22

Expressions and Variables

- Expression
  - A statement that has a value
  - For example:
    - 3 * 5 + 7
  - is an expressions that evaluates to:
    - 22
- Variable
  - A name for a value such that we can refer to it later

Expressions and Variables

- Expression
  - A statement that has a value
  - For example:
    - 3 * 5 + 7
  - is an expressions that evaluates to:
    - 22
- Variable
  - A name for a value such that we can refer to it later
  - Variables have a name, a type and a value

Expressions and Variables

- Expression
  - A statement that has a value
  - For example:
    - 3 * 5 + 7
  - is an expressions that evaluates to:
    - 22
- Variable
  - A name for a value such that we can refer to it later
  - Variables have a name, a type and a value
  - int i = 3 * 5 + 7

Expressions and Variables


```
#include <iostream>

int main() {
    int i;
    i = 3;
    int j = i * i;
    std::cout << i << "^2 = " << j << std::endl;
    return 0;
}
```

Expressions and Variables


```
#include <iostream>

int main() {
    int i;
    i = 3;
    int j = i * i;
    std::cout << i << "^2 = " << j << std::endl;
    return 0;
}
```

- Declare variable with name i and type int (integer)

Expressions and Variables


```
#include <iostream>

int main() {
    int i;
    i = 3;
    int j = i * i;
    std::cout << i << "^2 = " << j << std::endl;
    return 0;
}
```

- Declare variable with name i and type int (integer)
- Assign to i the value 3

Expressions and Variables


```
#include <iostream>

int main() {
    int i;
    i = 3;
    int j = i * i;
    std::cout << i << "^2 = " << j << std::endl;
    return 0;
}
```

- Declare variable with name i and type int (integer)
- Assign to i the value 3
- Declare variable j, evaluate $i \cdot i$ and assign the outcome to j

Expressions and Variables


```
#include <iostream>

int main() {
    int i;
    i = 3;
    int j = i * i;
    std::cout << i << "^2 = " << j << std::endl;
    return 0;
}
```

- Declare variable with name i and type int (integer)
- Assign to i the value 3
- Declare variable j, evaluate i · i and assign the outcome to j
- Outcome: 3^2 = 9

Conditionals

Conditionals

- Instructions are executed one-by-one

Conditionals

- Instructions are executed one-by-one
- However, we can decide to only execute some instructions if a condition is met
  - condition = boolean expression
  - *something that evaluates to true or false*

Conditionals

- Instructions are executed one-by-one
- However, we can decide to only execute some instructions if a condition is met
  - condition = boolean expression
  - *something that evaluates to true or false*
  - `i == 3`
  - `i < 7`
  - `i >= j`

Conditionals

- Instructions are executed one-by-one
- However, we can decide to only execute some instructions if a condition is met
  - condition = boolean expression
  - *something that evaluates to true or false*
  - `i == 3`
  - `i < 7`
  - `i >= j`

```
if (CONDITION) {
    // only happens if CONDITION is true
} else {
    // only happens if CONDITION is false
}
```

Conditionals


```
#include <iostream>

int main() {
    int i = 7;
    if (i > 5) {
        std::cout << "Bigger than 5" << std::endl;
    } else {
        std::cout << "5 or smaller" << std::endl;
    }
    return 0;
}
```

Conditionals


```cpp
#include <iostream>

int main() {
    int i = 7;
    if (i > 5) {
        std::cout << "Bigger than 5" << std::endl;
    } else {
        std::cout << "5 or smaller" << std::endl;
    }
    return 0;
}
```

Conditionals


```cpp
#include <iostream>

int main() {
    int i = 7;
    if (i > 5) {
        std::cout << "Bigger than 5" << std::endl;
    } else {
        std::cout << "5 or smaller" << std::endl;
    }
    return 0;
}
```

Conditionals


```
#include <iostream>

int main() {
    int i = 7;
    if (i > 5) {
        std::cout << "Bigger than 5" << std::endl;
    } else {
        std::cout << "5 or smaller" << std::endl;
    }
    return 0;
}
```

Conditionals


```
#include <iostream>

int main() {
    int i = 7;
    if (i > 5) {
        std::cout << "Bigger than 5" << std::endl;
    } else {
        std::cout << "5 or smaller" << std::endl;
    }
    return 0;
}
```

Loops - While

Loops - While

- Loops:
  - Can be used to repeat a set of instructions

Loops - While

- Loops:
  - Can be used to repeat a set of instructions
  - while some condition is true

Loops - While

- Loops:
  - Can be used to repeat a set of instructions
  - while some condition is true

```
#include <iostream>

int main() {
    int i = 0;
    while (i < 10) {
        std::cout << i << std::endl;
        i++; /* This means: i = i + 1 */
    }
    return 0;
}
```

Loops - While

- Loops:
  - Can be used to repeat a set of instructions
  - while some condition is true

```
#include <iostream>

int main() {
    int i = 0;
    while (i < 10) {
        std::cout << i << std::endl;
        i++; /* This means: i = i + 1 */
    }
    return 0;
}
```

Loops - While

- Loops:
  - Can be used to repeat a set of instructions
  - while some condition is true

```
#include <iostream>

int main() {
    int i = 0;
    while (i < 10) {
        std::cout << i << std::endl;
        i++; /* This means: i = i + 1 */
    }
    return 0;
}
```

Loops - While

- Loops:
  - Can be used to repeat a set of instructions
  - while some condition is true

```
#include <iostream>

int main() {
    int i = 0;
    while (i < 10) {
        std::cout << i << std::endl;
        i++; /* This means: i = i + 1 */
    }
    return 0;
}
```

Loops - While

- Loops:
  - Can be used to repeat a set of instructions
  - while some condition is true

```
#include <iostream>

int main() {
    int i = 0;
    while (i < 10) {
        std::cout << i << std::endl;
        i++; /* This means: i = i + 1 */
    }
    return 0;
}
```

Loops - While

- Loops:
  - Can be used to repeat a set of instructions
  - while some condition is true

```
#include <iostream>

int main() {
    int i = 0;
    while (i < 10) {
        std::cout << i << std::endl;
        i++; /* This means: i = i + 1 */
    }
    return 0;
}
```

Loops - While

- Loops:
  - Can be used to repeat a set of instructions
  - while some condition is true

```
#include <iostream>

int main() {
    int i = 0;
    while (i < 10) {
        std::cout << i << std::endl;
        i++; /* This means: i = i + 1 */
    }
    return 0;
}
```

Loops - For

Loops - For

- For-loops:
  - More advanced loop control

Loops - For

- For-loops:
  - More advanced loop control
  - Init, condition and increment in 'one line'

Loops - For

- For-loops:
  - More advanced loop control
  - Init, condition and increment in 'one line'
  - for(INIT; CONDITION; INCREMENT)

Loops - For

- For-loops:
  - More advanced loop control
  - Init, condition and increment in 'one line'
  - for(INIT; CONDITION; INCREMENT)

```
#include <iostream>

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

Loops - For

- For-loops:
  - More advanced loop control
  - Init, condition and increment in 'one line'
  - for(INIT; CONDITION; INCREMENT)

```
#include <iostream>

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

Loops - For

- For-loops:
  - More advanced loop control
  - Init, condition and increment in 'one line'
  - for(INIT; CONDITION; INCREMENT)

```
#include <iostream>

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

Loops - For

- For-loops:
  - More advanced loop control
  - Init, condition and increment in 'one line'
  - for(INIT; CONDITION; INCREMENT)

```
#include <iostream>

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

Loops - For

- For-loops:
  - More advanced loop control
  - Init, condition and increment in 'one line'
  - for(INIT; CONDITION; INCREMENT)

```
#include <iostream>

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

Loops - For

- For-loops:
  - More advanced loop control
  - Init, condition and increment in 'one line'
  - for(INIT; CONDITION; INCREMENT)

```
#include <iostream>

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

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction
- The body { ... } of the function tells what the function *'does'*

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction
- The body { ... } of the function tells what the function *'does'*
- The function can be called

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction
- The body { . . . } of the function tells what the function *'does'*
- The function can be called
- Flow of control . . .

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction
- The body { . . . } of the function tells what the function *'does'*
- The function can be called
- Flow of control . . .

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction
- The body { . . . } of the function tells what the function *'does'*
- The function can be called
- Flow of control . . .

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction
- The body { . . . } of the function tells what the function *'does'*
- The function can be called
- Flow of control . . .

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction
- The body { . . . } of the function tells what the function *'does'*
- The function can be called
- Flow of control . . .

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction
- The body { . . . } of the function tells what the function *'does'*
- The function can be called
- Flow of control . . .

Functions


```
#include <iostream>

void niceFunction() {
    int i = 3;
    std::cout << i << "^2 = " << i * i << std::endl;
}

int main() {
    niceFunction();
    return 0;
}
```

- Define a function with name niceFunction
- The body { . . . } of the function tells what the function *'does'*
- The function can be called
- Flow of control . . .

Functions


```
#include <iostream>

int niceFunction(int x, int y) {
    return x * x + y * y;
}

int main() {
    int i = 3;
    int j = niceFunction(i, 4);
    std::cout << j << std::endl;
    return 0;
}
```

- A function can also take input and return output:

Functions


```
#include <iostream>

int niceFunction(int x, int y) {
    return x * x + y * y;
}

int main() {
    int i = 3;
    int j = niceFunction(i, 4);
    std::cout << j << std::endl;
    return 0;
}
```

- A function can also take input and return output:

Functions


```
#include <iostream>

int niceFunction(int x, int y) {
    return x * x + y * y;
}

int main() {
    int i = 3;
    int j = niceFunction(i, 4);
    std::cout << j << std::endl;
    return 0;
}
```

- A function can also take input and return output:
  - Input arguments

Functions


```
#include <iostream>

int niceFunction(int x, int y) {
    return x * x + y * y;
}

int main() {
    int i = 3;
    int j = niceFunction(i, 4);
    std::cout << j << std::endl;
    return 0;
}
```

- A function can also take input and return output:
  - Input arguments
  - Return value

Functions


```
#include <iostream>

int niceFunction(int x, int y) {
    return x * x + y * y;
}

int main() {
    int i = 3;
    int j = niceFunction(i, 4);
    std::cout << j << std::endl;
    return 0;
}
```

- A function can also take input and return output:
  - Input arguments
  - Return value
  - Return type

Recap

- So far, we have seen:
  - How to write, compile and run a program
  - Variables
  - Conditionals
  - Loops
  - Functions

Recap

- So far, we have seen:
  - How to write, compile and run a program
  - Variables
  - Conditionals
  - Loops
  - Functions
- Enough to write a procedural program

Recap

- So far, we have seen:
  - How to write, compile and run a program
  - Variables
  - Conditionals
  - Loops
  - Functions
- Enough to write a procedural program
  - Step-by-step instructions
  - re-use of code
  - Advanced control flow

C++ Data Types

C++ has many built-in data types:

int	Integer	{. . . , -2, -1, 0, 1, . . . }

C++ Data Types

C++ has many built-in data types:

int	Integer	{. . . , -2, -1, 0, 1, . . . }
unsigned int	Positive integer	{0, 1, . . . }

C++ Data Types

C++ has many built-in data types:

int	Integer	{. . . , -2, -1, 0, 1, . . . }
unsigned int	Positive integer	{0, 1, . . . }
bool	Boolean	0 or 1

C++ Data Types

C++ has many built-in data types:

int	Integer	{. . . , -2, -1, 0, 1, . . . }
unsigned int	Positive integer	{0, 1, . . . }
bool	Boolean	0 or 1
float	Floating point	*e.g.*, -2.6, 1.0, 3.1415

C++ Data Types

C++ has many built-in data types:

int	Integer	{. . . , -2, -1, 0, 1, . . . }
unsigned int	Positive integer	{0, 1, . . . }
bool	Boolean	0 or 1
float	Floating point	*e.g.*, -2.6, 1.0, 3.1415
double	Double floating point	

C++ Data Types

C++ has many built-in data types:

int	Integer	{. . . , -2, -1, 0, 1, . . . }
unsigned int	Positive integer	{0, 1, . . . }
bool	Boolean	0 or 1
float	Floating point	*e.g.*, -2.6, 1.0, 3.1415
double	Double floating point	
char	Character	*e.g.*, 'a', 'B', '?'

C++ Data Types

C++ has many built-in data types:

int	Integer	{. . . , -2, -1, 0, 1, . . . }
unsigned int	Positive integer	{0, 1, . . . }
bool	Boolean	0 or 1
float	Floating point	*e.g.*, -2.6, 1.0, 3.1415
double	Double floating point	
char	Character	*e.g.*, 'a', 'B', '?'
unsigned char	Byte	{0, . . . , 255}

C++ Data Types

- Built-in data types are nice, but limited

C++ Data Types

- Built-in data types are nice, but limited
  - Let's say we want to determine PICO's nearest laser point

C++ Data Types

- Built-in data types are nice, but limited
  - Let's say we want to determine PICO's nearest laser point
  - Such point can be represented with two variables:
    - double x
    - double y

C++ Data Types

- Built-in data types are nice, but limited
  - Let's say we want to determine PICO's nearest laser point
  - Such point can be represented with two variables:
    - double x
    - double y
  - Cumbersome to pass around

C++ Data Types

- Built-in data types are nice, but limited
  - Let's say we want to determine PICO's nearest laser point
  - Such point can be represented with two variables:
    - double x
    - double y
  - Cumbersome to pass around
  - Not very clear we're talking about a point (semantics)

C++ Data Types

- Built-in data types are nice, but limited
  - Let's say we want to determine PICO's nearest laser point
  - Such point can be represented with two variables:
    - double x
    - double y
  - Cumbersome to pass around
  - Not very clear we're talking about a point (semantics)
  - How to return both values from a function?

C++ Data Types

- Built-in data types are nice, but limited
  - Let's say we want to determine PICO's nearest laser point
  - Such point can be represented with two variables:
    - double x
    - double y
  - Cumbersome to pass around
  - Not very clear we're talking about a point (semantics)
  - How to return both values from a function?

```
...? determineNearestPoint() {
    // Some nice code
    double x = ...
    double y = ...
    return ...?
}
```

Data structures

- Solution: create your own data structure:

Data structures

- Solution: create your own data structure:
  - *“a group of data elements grouped together under one name”*

Data structures

- Solution: create your own data structure:
  - *“a group of data elements grouped together under one name”*
  - Are called structs in C++

Data structures

- Solution: create your own data structure:
  - *"a group of data elements grouped together under one name"*
  - Are called structs in C++

```
struct Point {    // This starts the definition of a struct
    double x;     // This declares x as member of Point
    double y;     // This declares y as member of Point
}
```

Data structures

- Solution: create your own data structure:
  - *"a group of data elements grouped together under one name"*
  - Are called structs in C++

```
struct Point {    // This starts the definition of a struct
    double x;     // This declares x as member of Point
    double y;     // This declares y as member of Point
}
```

- Now we can use Point as data type:

Data structures

- Solution: create your own data structure:
  - *"a group of data elements grouped together under one name"*
  - Are called structs in C++

```
struct Point {    // This starts the definition of a struct
    double x;     // This declares x as member of Point
    double y;     // This declares y as member of Point
}
```

- Now we can use Point as data type:

```
Point determineNearestPoint() {
    // Some nice code
    Point p;     // declare variable p of type Point
    p.x = ...    // set x member of p
    p.y = ...    // set y member of p
    return p;
}
```

Data structures

- Solution: create your own data structure:
  - *"a group of data elements grouped together under one name"*
  - Are called structs in C++

```
struct Point {    // This starts the definition of a struct
    double x;     // This declares x as member of Point
    double y;     // This declares y as member of Point
}
```

- Now we can use Point as data type:

```
void main() {
    Point p = determineNearestPoint();
    std::cout << "x = " << p.x << std::endl;
    std::cout << "y = " << p.y << std::endl;
}
```

Data structures

- We can even use structs as member types
  - This way, we can build more complex data structures:

Data structures

- We can even use structs as member types
  - This way, we can build more complex data structures:

```
struct Line {
    Point p1;    // p1 is member of Line with type Point
    Point p2;    // p2 is member of Line with type Point
}
```

Data structures

- We can even use structs as member types
  - This way, we can build more complex data structures:

```
struct Line {
    Point p1;    // p1 is member of Line with type Point
    Point p2;    // p2 is member of Line with type Point
}
```

```
// ...
Line l;
l.p1.x = 3.0;
l.p1.y = 4.5;
l.p2.x = 6.0;
l.p2.y = 10.3;

Point q = l.p2;

std::cout << q.x << std::endl;
// ...
```

std::vector

- Sometimes we want to store an arbitrary number of values together

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure
  - Can hold any type of data, but

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure
  - Can hold any type of data, but
  - all elements in the vector must have the same type

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure
  - Can hold any type of data, but
  - all elements in the vector must have the same type

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

int main() {
    std::vector<int> v;  /* Declare a vector that holds ints */
    v.push_back(10);     /* Add '10' to the back of the vector */
    v.push_back(20);     /* Add '20' to the back */
    v.push_back(30);     /* Add '30' */

    for(unsigned int i = 0; i < v.size(); i++) {
        std::cout << v[i] << std::endl; /* Print i'th element */
    }
}
```

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure
  - Can hold any type of data, but
  - all elements in the vector must have the same type

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

int main() {
    std::vector<int> v;  /* Declare a vector that holds ints */
    v.push_back(10);     /* Add '10' to the back of the vector */
    v.push_back(20);     /* Add '20' to the back */
    v.push_back(30);     /* Add '30' */

    for(unsigned int i = 0; i < v.size(); i++) {
        std::cout << v[i] << std::endl; /* Print i'th element */
    }
}
```

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure
  - Can hold any type of data, but
  - all elements in the vector must have the same type

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

int main() {
    std::vector<int> v;  /* Declare a vector that holds ints */
    v.push_back(10);     /* Add '10' to the back of the vector */
    v.push_back(20);     /* Add '20' to the back */
    v.push_back(30);     /* Add '30' */

    for(unsigned int i = 0; i < v.size(); i++) {
        std::cout << v[i] << std::endl; /* Print i'th element */
    }
}
```

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure
  - Can hold any type of data, but
  - all elements in the vector must have the same type

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

int main() {
    std::vector<int> v;  /* Declare a vector that holds ints */
    v.push_back(10);     /* Add '10' to the back of the vector */
    v.push_back(20);     /* Add '20' to the back */
    v.push_back(30);     /* Add '30' */

    for(unsigned int i = 0; i < v.size(); i++) {
        std::cout << v[i] << std::endl; /* Print i'th element */
    }
}
```

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure
  - Can hold any type of data, but
  - all elements in the vector must have the same type

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

int main() {
    std::vector<int> v;  /* Declare a vector that holds ints */
    v.push_back(10);     /* Add '10' to the back of the vector */
    v.push_back(20);     /* Add '20' to the back */
    v.push_back(30);     /* Add '30' */

    for(unsigned int i = 0; i < v.size(); i++) {
        std::cout << v[i] << std::endl; /* Print i'th element */
    }
}
```

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure
  - Can hold any type of data, but
  - all elements in the vector must have the same type

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

int main() {
    std::vector<int> v;  /* Declare a vector that holds ints */
    v.push_back(10);     /* Add '10' to the back of the vector */
    v.push_back(20);     /* Add '20' to the back */
    v.push_back(30);     /* Add '30' */

    for(unsigned int i = 0; i < v.size(); i++) {
        std::cout << v[i] << std::endl; /* Print i'th element */
    }
}
```

std::vector

- Sometimes we want to store an arbitrary number of values together
- Solution: use the std::vector data structure
  - Can hold any type of data, but
  - all elements in the vector must have the same type

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

int main() {
    std::vector<int> v;  /* Declare a vector that holds ints */
    v.push_back(10);     /* Add '10' to the back of the vector */
    v.push_back(20);     /* Add '20' to the back */
    v.push_back(30);     /* Add '30' */

    for(unsigned int i = 0; i < v.size(); i++) {
        std::cout << v[i] << std::endl; /* Print i'th element */
    }
}
```

C++ Standard Library

- std::vector is a part of the C++ Standard Library

C++ Standard Library

- std::vector is a part of the C++ Standard Library
- This library contains many useful data structures, functions, etc...

C++ Standard Library

- std::vector is a part of the C++ Standard Library
- This library contains many useful data structures, functions, etc...
- Uses the namespace **std**:
  - std::vector means vector in the namespace std

C++ Standard Library

- std::vector is a part of the C++ Standard Library
- This library contains many useful data structures, functions, etc...
- Uses the namespace **std**:
  - std::vector means vector in the namespace std
  - std::cout
  - std::endl
  - ...

C++ Standard Library

- std::vector is a part of the C++ Standard Library
- This library contains many useful data structures, functions, etc...
- Uses the namespace **std**:
  - std::vector means vector in the namespace std
  - std::cout
  - std::endl
  - ...
- http://www.cplusplus.com/reference/

std::string

- std::string

std::string

- std::string
  - Represents text:
    - A sequence (string) of characters

std::string

- std::string
  - Represents text:
    - A sequence (string) of characters

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

int main() {
    std::string s1 = "Hello";
    std::string s2 = "World";
    std::string s3 = s1 + " " + s2; /* concatenate strings */

    std::cout << s3 << std::endl;
    std::cout << s3.size() << std::endl; /* print size */
}
```

std::string

- std::string
  - Represents text:
    - A sequence (string) of characters

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

int main() {
    std::string s1 = "Hello";
    std::string s2 = "World";
    std::string s3 = s1 + " " + s2; /* concatenate strings */

    std::cout << s3 << std::endl;
    std::cout << s3.size() << std::endl; /* print size */
}
```

std::string

- std::string
  - Represents text:
    - A sequence (string) of characters

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

int main() {
    std::string s1 = "Hello";
    std::string s2 = "World";
    std::string s3 = s1 + " " + s2; /* concatenate strings */

    std::cout << s3 << std::endl;
    std::cout << s3.size() << std::endl; /* print size */
}
```

std::string

- std::string
  - Represents text:
    - A sequence (string) of characters

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

int main() {
    std::string s1 = "Hello";
    std::string s2 = "World";
    std::string s3 = s1 + " " + s2; /* concatenate strings */

    std::cout << s3 << std::endl;
    std::cout << s3.size() << std::endl; /* print size */
}
```

std::string

- std::string
  - Represents text:
    - A sequence (string) of characters

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

int main() {
    std::string s1 = "Hello";
    std::string s2 = "World";
    std::string s3 = s1 + " " + s2; /* concatenate strings */

    std::cout << s3 << std::endl;
    std::cout << s3.size() << std::endl; /* print size */
}
```

std::string

- std::string
  - Represents text:
    - A sequence (string) of characters

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

int main() {
    std::string s1 = "Hello";
    std::string s2 = "World";
    std::string s3 = s1 + " " + s2; /* concatenate strings */

    std::cout << s3 << std::endl;
    std::cout << s3.size() << std::endl; /* print size */
}
```

std::map

- std::map

std::map

- std::map
  - Creates mapping between two data types

std::map

- std::map
  - Creates mapping between two data types
  - Can think of it as: $K \rightarrow V$

std::map

- std::map
  - Creates mapping between two data types
  - Can think of it as: $K \rightarrow V$
  - Declaration: std::map‹KeyType, ValueType›

std::map

- std::map
  - Creates mapping between two data types
  - Can think of it as: $K \rightarrow V$
  - Declaration: std::map‹KeyType, ValueType›

```
#include <iostream>
 #include <map>

int main() {
    std::map<std::string, int> student_nr;
    student_nr["Pete Johnson"] = 47823;
    student_nr["John Snow"] = 49529;

    std::cout << m["John Snow"] << std::endl;
}
```

std::map

- std::map
  - Creates mapping between two data types
  - Can think of it as: $K \rightarrow V$
  - Declaration: std::map‹KeyType, ValueType›

```
#include <iostream>
 #include <map>

int main() {
    std::map<std::string, int> student_nr;
    student_nr["Pete Johnson"] = 47823;
    student_nr["John Snow"] = 49529;

    std::cout << m["John Snow"] << std::endl;
}
```

std::map

- std::map
  - Creates mapping between two data types
  - Can think of it as: $K \rightarrow V$
  - Declaration: std::map‹KeyType, ValueType›

```
#include <iostream>
 #include <map>

int main() {
    std::map<std::string, int> student_nr;
    student_nr["Pete Johnson"] = 47823;
    student_nr["John Snow"] = 49529;

    std::cout << m["John Snow"] << std::endl;
}
```

std::map

- std::map
  - Creates mapping between two data types
  - Can think of it as: $K \rightarrow V$
  - Declaration: std::map‹KeyType, ValueType›

```
#include <iostream>
 #include <map>

int main() {
    std::map<std::string, int> student_nr;
    student_nr["Pete Johnson"] = 47823;
    student_nr["John Snow"] = 49529;

    std::cout << m["John Snow"] << std::endl;
}
```

std::map

- std::map
  - Creates mapping between two data types
  - Can think of it as: $K \rightarrow V$
  - Declaration: std::map‹KeyType, ValueType›

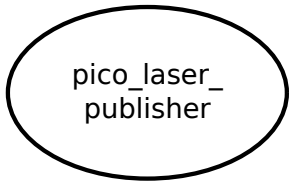
```
#include <iostream>
 #include <map>

int main() {
    std::map<std::string, int> student_nr;
    student_nr["Pete Johnson"] = 47823;
    student_nr["John Snow"] = 49529;

    std::cout << m["John Snow"] << std::endl;
}
```

ROS Concepts

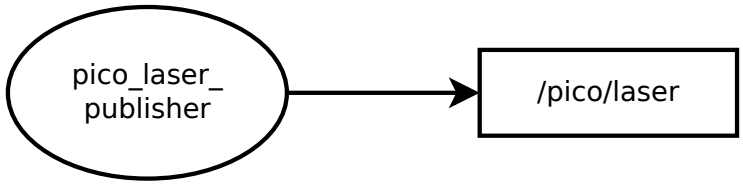
ROS Concepts

- Node: process that performs computation
- Master: provides name registration and lookup
- Messages: nodes communicate with each other by passing messages
- Topics: named buses over which nodes exchange messages

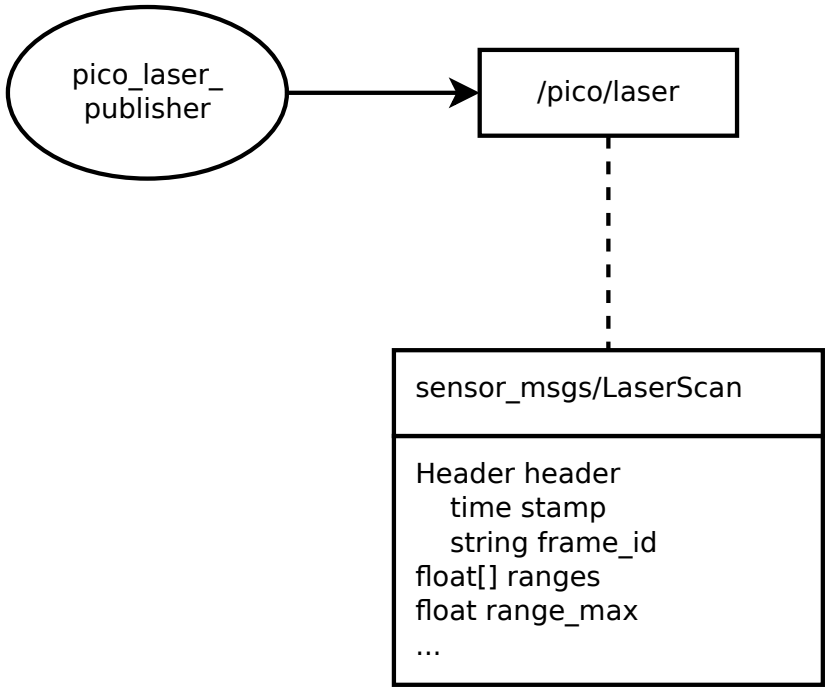
ROS Communication Example



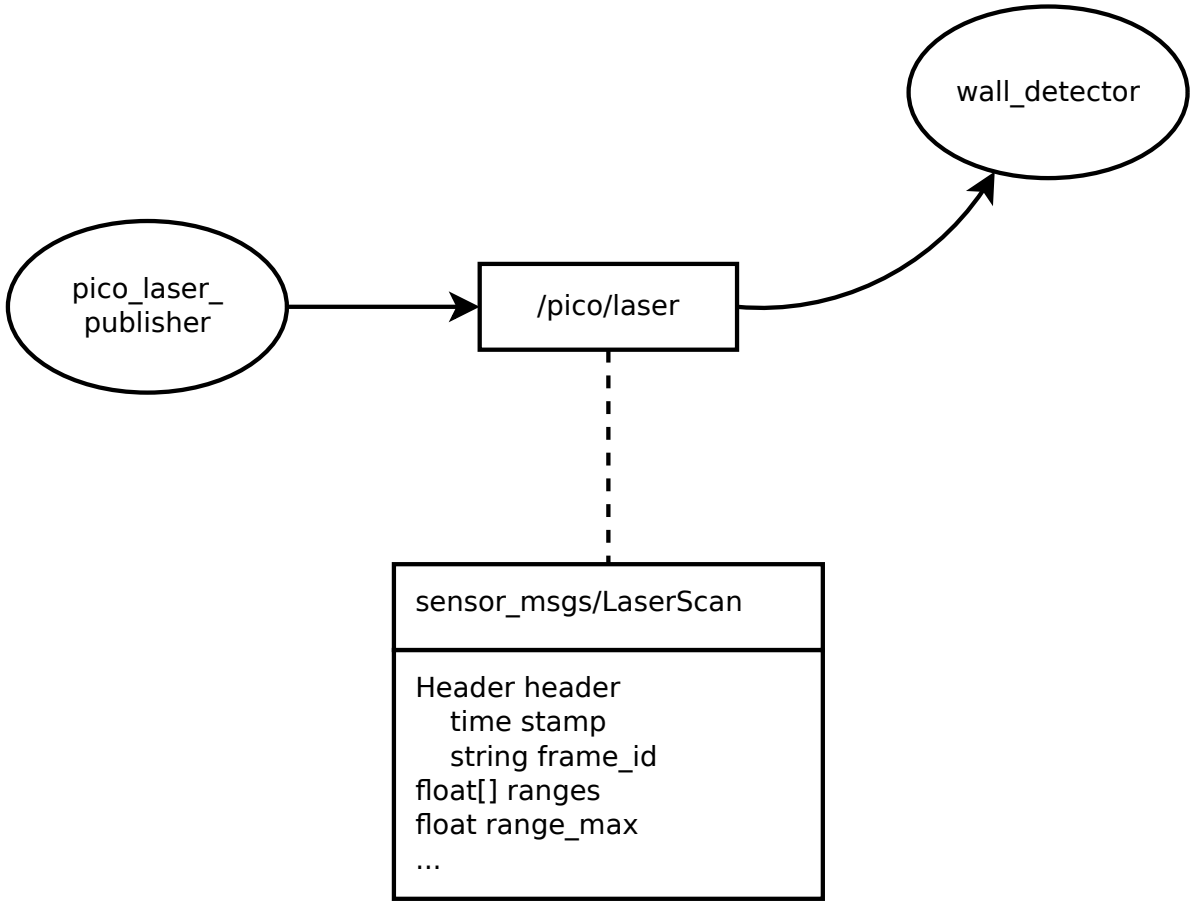
ROS Communication Example



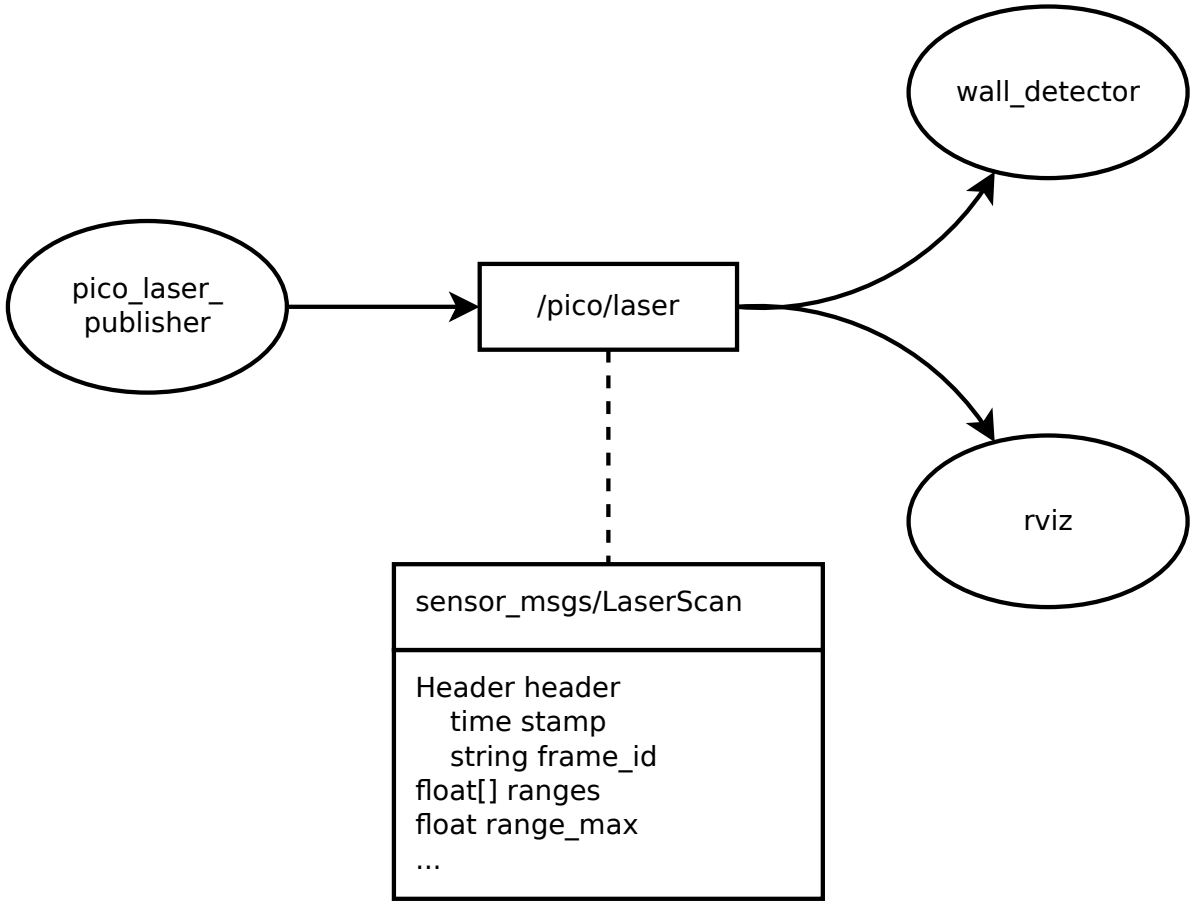
ROS Communication Example



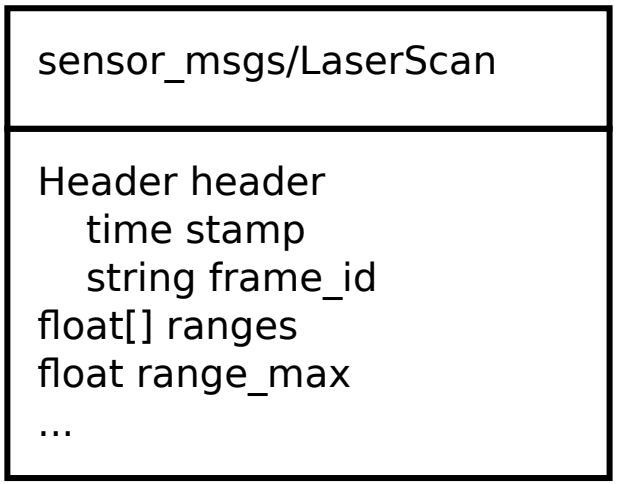
ROS Communication Example



ROS Communication Example



ROS Message Types



ROS Message Types

sensor_msgs/LaserScan

Header header
  time stamp
  string frame_id
float[] ranges
float range_max
...

Is in fact:

```
struct LaserScan {
    Header header;
    float range_max;
    std::vector<float> ranges;
}

struct Header {
    std::string frame_id;
    Time stamp;
}

struct Time {
    int secs;
    int nsecs;
}
```

ROS Message Types

sensor_msgs/LaserScan

Header header
  time stamp
  string frame_id
float[] ranges
float range_max
…

Is in fact:

```
struct LaserScan {
    Header header;
    float range_max;
    std::vector<float> ranges;
}

struct Header {
    std::string frame_id;
    Time stamp;
}

struct Time {
    int secs;
    int nsecs;
}
```

ROS Message Types

sensor_msgs/LaserScan

Header header
  time stamp
  string frame_id
float[] ranges
float range_max
…

Is in fact:

```
struct LaserScan {
    Header header;
    float range_max;
    std::vector<float> ranges;
}

struct Header {
    std::string frame_id;
    Time stamp;
}

struct Time {
    int secs;
    int nsecs;
}
```

ROS and C++

ROS and C++

- To use ROS in your program:

```
#include <ros/ros.h>
```

ROS and C++

- To use ROS in your program:

```
#include <ros/ros.h>
```

- Register your program as a node to the ROS master:

```
ros::init(..., "your_node_name");
```

ROS and C++

- To use ROS in your program:

```
#include <ros/ros.h>
```

- Register your program as a node to the ROS master:

```
ros::init(..., "your_node_name");
```

- Let ROS know you want to listen to a certain topic:

```
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("/pico/laser", 1,
                                  callbackFunction);
```

ROS and C++

- Let ROS know you want to listen to a certain topic:

```
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("/pico/laser", 1,
                                  callbackFunction);
```

ROS and C++

- Let ROS know you want to listen to a certain topic:

```
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("/pico/laser", 1,
                                  callbackFunction);
```

- Start listening to the topics:

```
ros::spin();
```

ROS and C++

- Let ROS know you want to listen to a certain topic:

```
ros::NodeHandle n;
ros::Subscriber sub = n.subscribe("/pico/laser", 1,
                                  callbackFunction);
```

- Start listening to the topics:

```
ros::spin();
```

- This function is called every time the node receives a message:

```
void callbackFunction(sensor_msgs::LaserScan scan) {
    // do something
    std::cout << scan.header.stamp << std::endl;
}
```

ROS C++ Example


```
#include <ros/ros.h>                    // include ROS
#include <sensor_msgs/LaserScan.h>      // include LaserScan
                                        // message type

void callback(sensor_msgs::LaserScan scan) {
    // do something
    std::cout << scan.header.stamp << std::endl;
}

int main(int argc, char** argv) {  // ignore argc and argv
    ros::init(argc, char** argv, "example");  // register node
                                              // to master
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("/pico/laser",
                                      1, LaserScan);

    ros::spin();     // Keep receiving messages
    return 0;
}
```

ROS C++ Example

ROS C++ Example

- The callback function gets the ROS message as input argument:

```
void callback(LaserScan scan) {
    // do something
    std::cout << scan.header.stamp << std::endl;
}
```

ROS C++ Example

- The callback function gets the ROS message as input argument:

```
void callback(LaserScan scan) {
    // do something
    std::cout << scan.header.stamp << std::endl;
}
```

- Remember:

```
struct LaserScan {
    Header header;
    float range_max;
    std::vector<float> ranges;
}
```

ROS C++ Example

- So, we can take a look a the data inside:

ROS C++ Example

- So, we can take a look a the data inside:

```
void callback(sensor_msgs::LaserScan scan) {

    for(unsigned int i = 0; i < scan.ranges.size(); i++) {
        if (scan.ranges[i] < 0.3) {
            std::cout << "HELP!" << std::end;
            // ...
        }
    }

}
```

ROS C++ Example

- So, we can take a look a the data inside:

```
void callback(sensor_msgs::LaserScan scan) {

    for(unsigned int i = 0; i < scan.ranges.size(); i++) {
        if (scan.ranges[i] < 0.3) {
            std::cout << "HELP!" << std::end;
            // ...
        }
    }

}
```

- Try to get this example running
  - Together with the jazz_example package, it's a good start for your project

Recap

- C++ Data Types
- Structs
- C++ Standard Library
  - std::vector
  - std::string
  - std::map
- ROS Message as C++ struct
- ROS C++ Subscriber Example