ME454
Dynamics System Programming

TA Session 4. C++ Basic (2)

Class Objectives

- C++ practice (class and pointer)
  - List implementation 2 (linked list)
- Dynamics simulation practice
  - Multibody object with links and joints
  - Simulation state monitoring with RQT

Directory Structure (Important!)

Please check if the highlighted directories are present.

Please check if your files are properly set as follows.

You may copy the files using filesystem GUI (Nautilus, Windows, …) instead of cp.

```
your home directory
▸ repos
  ▸ ME454_2024
▸ cpp_ws
  ▸ cpp_practice1
  ▸ cpp_practice2
  ▸ Homework2 (TBA)
▸ ros2_ws
  ▸ src
    ▸ ball_throwing
    ▸ pendulum_movement
```

```
In Ubuntu command prompt

cd ~/repos/ME454_2024
// Updates the repository (from previous week)
git pull

// Copies C++ practice files
cp -r ~/repos/ME454_2024/cpp_practice2 ~/cpp_ws

// Copies ball_throwing practice package
cp -r ~/repos/ME454_2024/pendulum_movement ~/ros2_ws/src
```

C++ Practice:
Linked List Implementation with Pointer

Pointer

- Address (position) of a variable in computer memory
- Usually represented with a hexadecimal value (base of 16)
- Referencing : operator & is used to get the pointer of the variable
- Dereferencing : operator * is used to get the variable from the pointer

```
int var1 = 3;
int var2 = 24;
int var3 = 17;

cout << &var1 << endl; // result : 0xf8ac
cout << &var2 << endl; // result : 0xf8a8
cout << &var3 << endl; // result : 0xf8a4

int *p_var1 = &var1; // referencing
int v_var1 = *p_var1; // dereferencing
```

"star p_var is integer"

dereferencing with *

Address	Value	
0xf8a4	17	var3
0xf8a8	24	var2
0xf8ac	3	var1

referencing with &

Function and Pointer

- Call-by-reference Vs. Call-by-value
  - Function argument as a pointer is the address outside the function memory.
  - Function argument as a value is copied to function memory
- The member of struct/class pointer can be accessed with the operator ->
  - The member of the struct/class can be accessed with the operator .

```
void swap1(int n1, int n2); // swaps two integer arguments
void swap2(int *p_n1, int *p_n2); // swaps two integer values from their pointer arguments

int a = 1, b = 2;

swap1(a, b)
cout << a << “, ” << b << endl; // result : 1, 2
swap2(&a, &b)
cout << a << “, ” << b << endl; // result : 2, 1

MyList my_list; // a list structure
my_list.len = 0;
MyList *p_my_list = &my_list; // pointer of the list structure
cout << p_my_list->len << endl; // result : 0
```

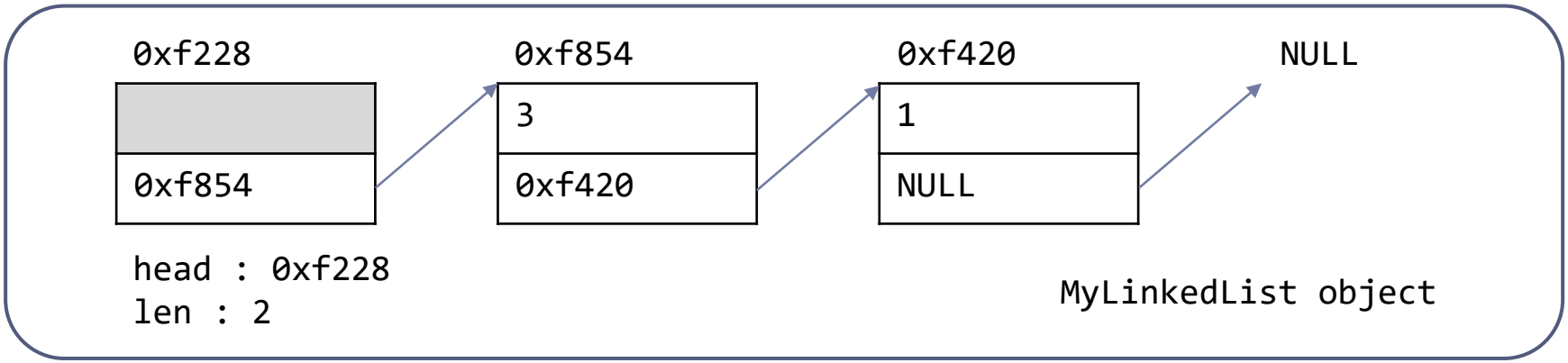
Linked List Class

- In the linked list, items are connected with referencing (pointer).
  - A linked list starts with a head node.
  - A node in a linked list contains an item and a pointer to the next node.

- Implementation
  - A linked list as an object `MyLinkedList`
  - Nodes as structures `Node`

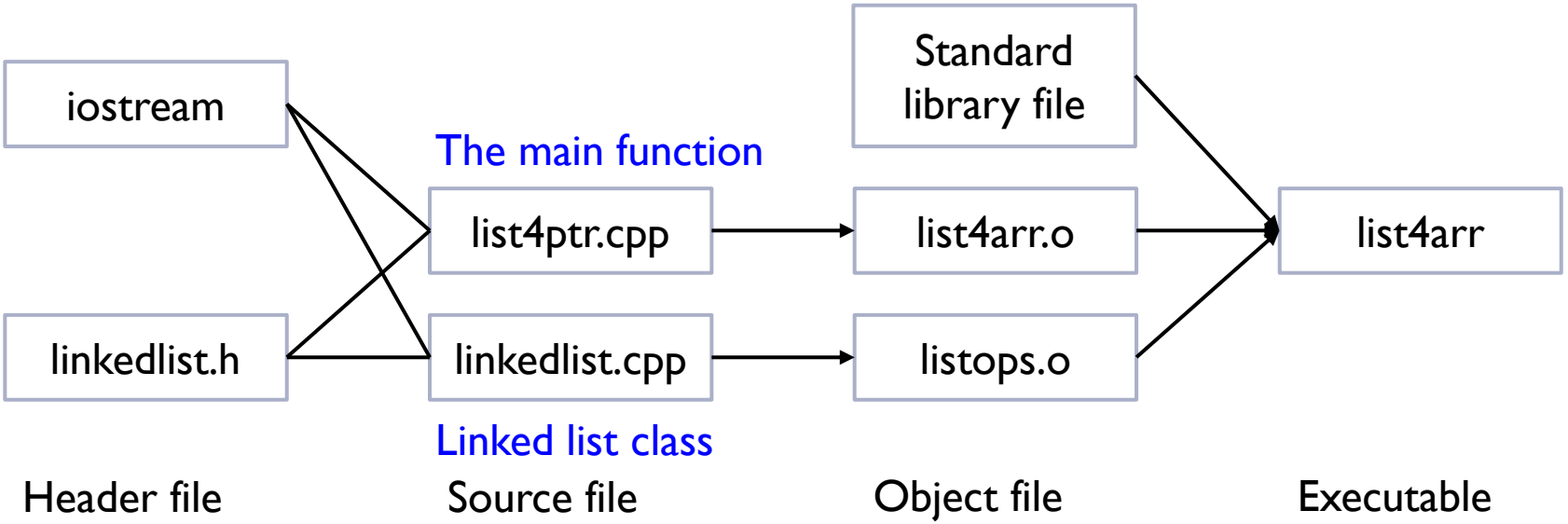
Node struct

int item
Node *next



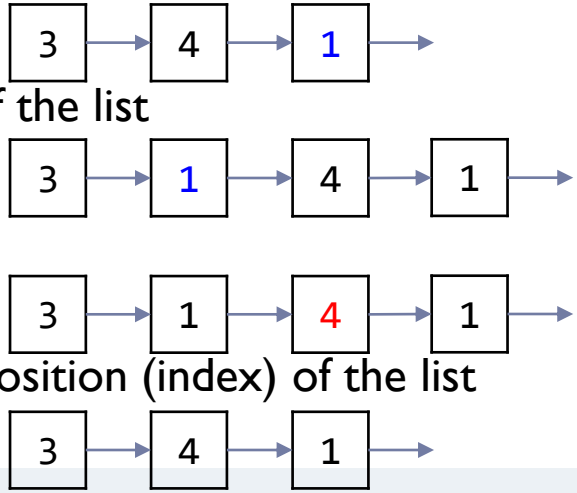
Files for list4ptr

- Let's implement a linked list with pointers.
  - At, Append, Insert, Max, and Pop



Linked List Methods

- At: returns an item in the specific position (index) of the list (already implemented)
  - `my_list.append(1)`
- Append: adds an item to the end of the list (already implemented)
  - `my_list.append(1)`
- Insert: adds an item to the specific position (index) of the list
  - `my_list.insert(1, 1)`
- Max: returns the biggest item in the list
  - `my_list.max()`
- Pop: removes and returns an item from the specific position (index) of the list
  - `my_list.pop(1)`




```
linkedlist.h

int at(int index);
int append(int item);
int insert(int item, int index);
int max();
int pop(int index);
```

Dynamic Memory Allocation

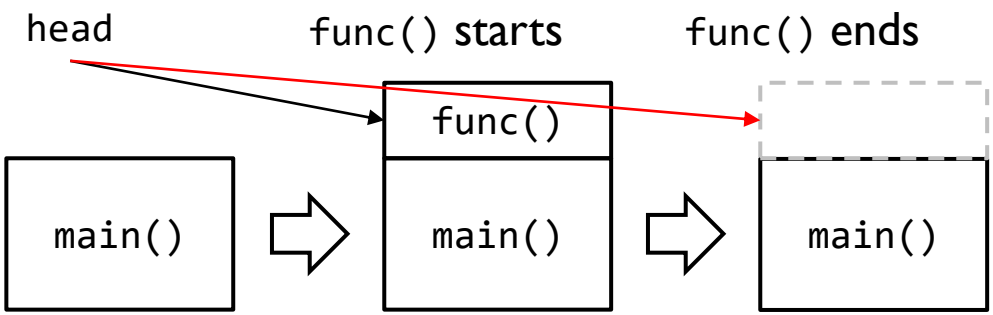
- Last week's `MyList` used an array with a fixed memory size.
  - `int items[LIST_CAPACITY]; // LIST_CAPACITY is 64`

- In `MyLinkedList`, there are functions that add memory to the objects.
  - Constructor `MyLinkedList()`, methods `append()`, `insert()`
  - How about making a `Node` variable in the function, and adding its pointer?
    - Memory for a function (stack memory) is released after the function.

linkedlist.cpp (wrong code)

```
MyLinkedList::MyLinkedList()
{
    len = 0;
    Node new_node;
    head = &new_node;
    head->item = -1;
    head->next = NULL;
}
```

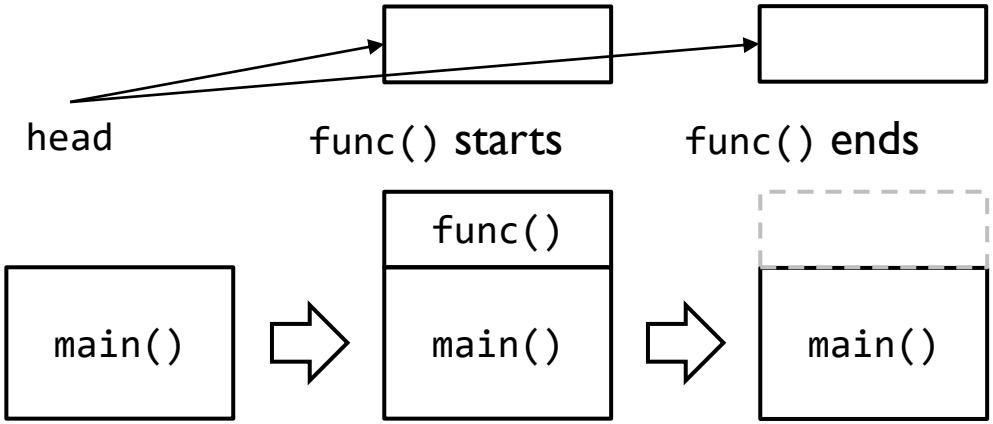
```
gpark@DESKTOP-3QBQKP6:~/cpp_ws/list4ptr$ ./list4ptr
세그멘테이션 오류 Segmentation fault
```



Dynamic Memory Allocation

- Commands `new` and `delete` manages dynamic memory (heap memory).
  - Allocate a memory space for structs/objects with `new`
  - Free the memory space with `delete`
    - Memory leaks will occur without deleting all the memory allocated.

```
linkedlist.cpp (fixed code)

MyLinkedList::MyLinkedList()
{
    len = 0;
    head = new Node;
    head->item = -1;
    head->next = NULL;
}

MyLinkedList::~MyLinkedList()
{
    ...
    delete head;
}
```



Other Notices

- The contents of MyList should be non-negative integers:
  - Ranging from 0 to 2,147,483,647 (maximum value of 32-bit int)

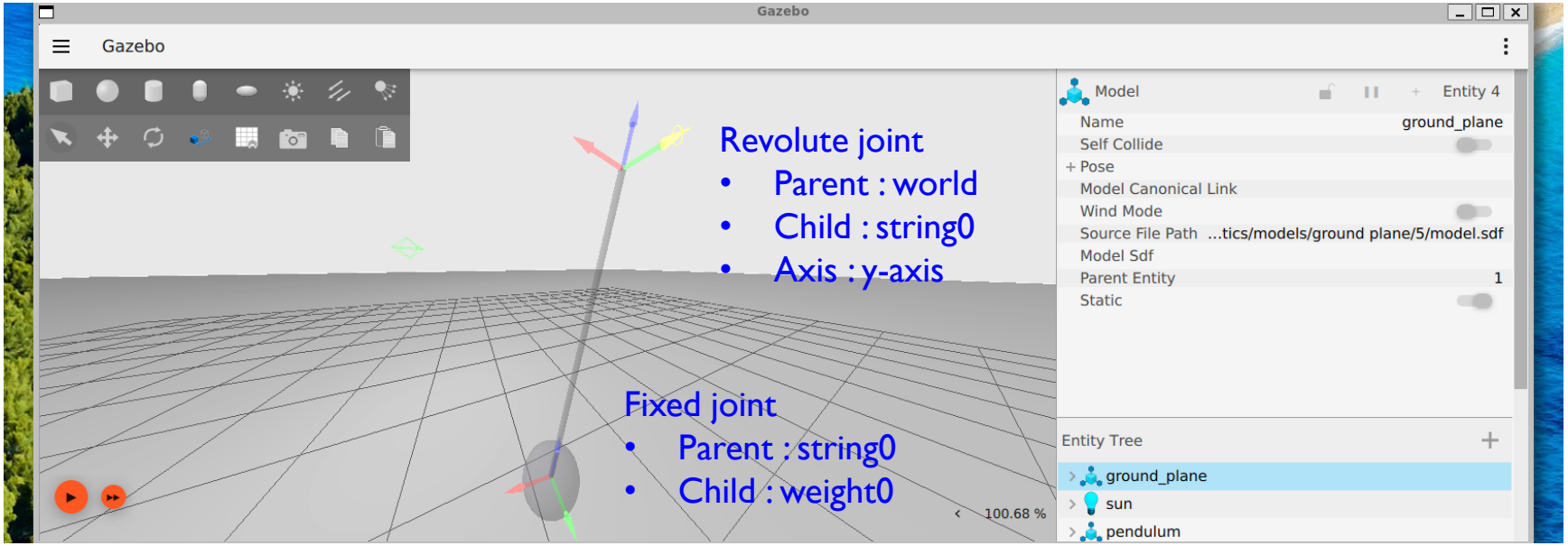
- With the Boolean variable `interact`, you can select from two modes
  - User interaction mode: demonstration for standard console input and output
  - Pre-set command mode: may be modified to test the functions

- TAs will check if the functions are implemented with the evaluation code.
  - `./list4ptr_eval`
  - It is okay to refer to the source `list4ptr_eval.cpp`, but please don't modify it.
  - Don't forget to build the program using `make` after code modification.
  - Please read carefully the description comment in the source code.

Dynamics Simulation Practice: MultiBody Object with Joints and Links

Running Gazebo with ROS Launch

```
// Ubuntu command to start Gazebo simulation

// Build a ROS package & Source the workspace
cd ~/ros2_ws
colcon build
. install/setup.bash

// Launch Gazebo simulation in rigidbody package
// ros2 launch [package_name] [launch_file_name]
ros2 launch pendulum_movement launch.py
```



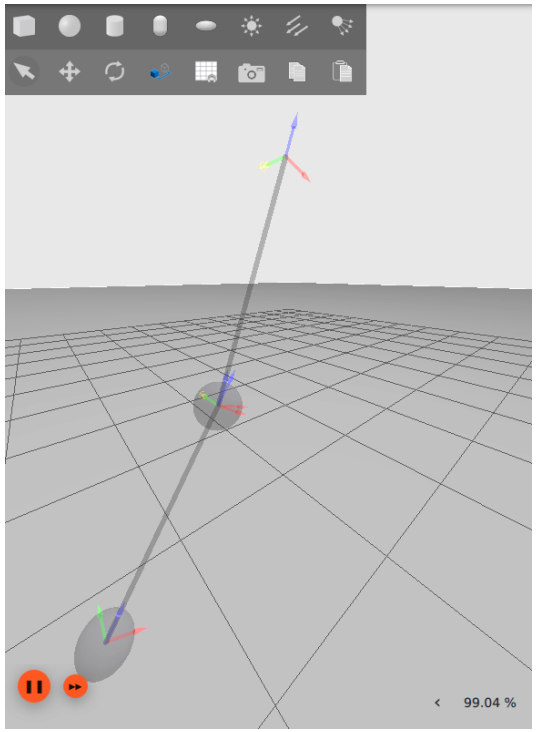
Double Pendulum Model

- The model is specified in a separate model description file
  - `~/ros2_ws/models/pendulum/pendulum.sdf`

- The model file was modified as following
  - `~/ros2_ws/models/pendulum/pendulum_double.sdf`

  - Add two links
    - Cylinder link `string1`
    - Sphere link `weight1`

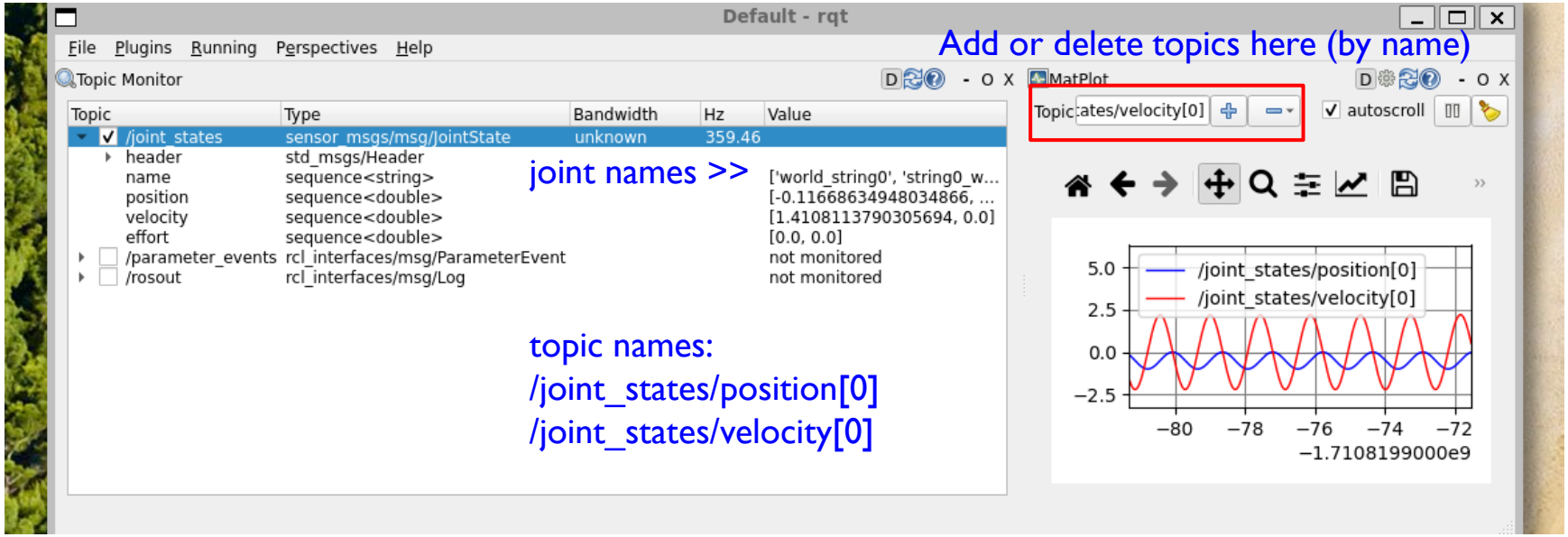
  - Add two joints
    - Revolute joint `weight0_string1`
    - Fixed joint `string1_weight1`



RQT: ROS GUI Tool

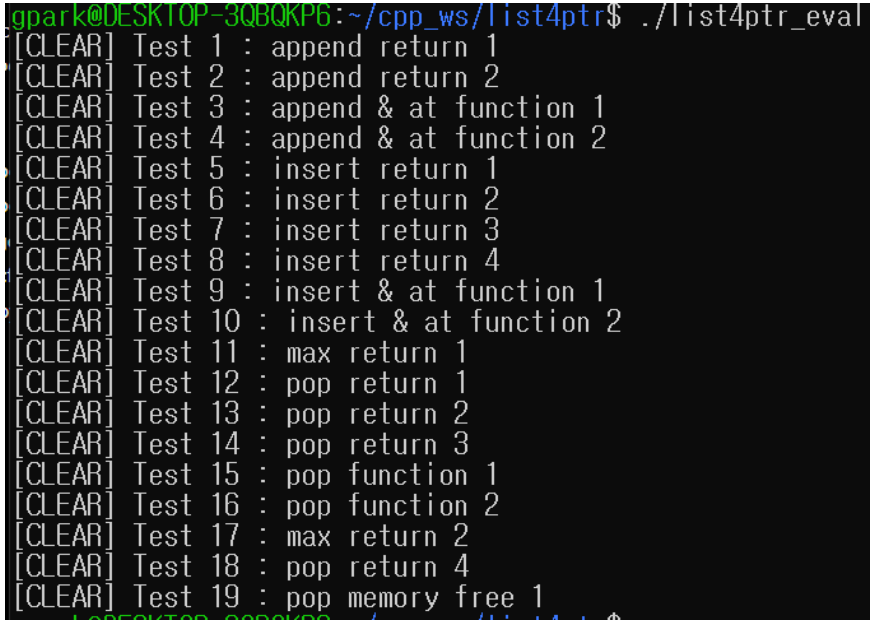
```
// Ubuntu command to start ROS GUI tool

rqt
rqt --clear-config // reset the previous configuration (monitor, plot, and so on)
```

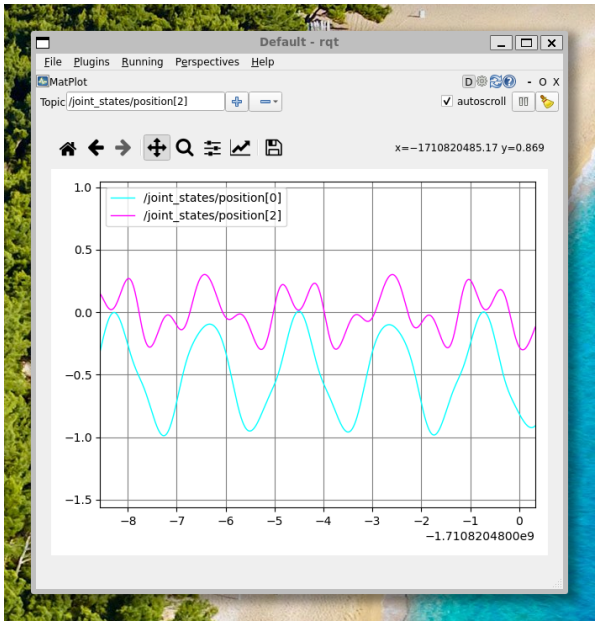
```
// In rqt menu taps

Plugins >> Topics >> Topic Monitor
Plugins >> Visualization >> Plot
```



TODO

- Complete all the tests in `list4ptr_eval` and show the screen to TAs.
- Try running the double pendulum simulation and show the x-t graph of two weights.
  - You don't need to modify the given files.
  - You may refer to them for further tasks. (practice, homework, exams, …)





References for Gazebo-Ros Simulation

How to use Gazebo with Ros2:

- https://gazebosim.org/docs/harmonic/ros_installation#-gazebo-harmonic-with-ros-2-humble-iron-or-rolling-use-with-caution-
- https://gazebosim.org/docs/harmonic/migrating_gazebo_classic_ros2_packages#launch-the-world
- https://gazebosim.org/docs/harmonic/migrating_gazebo_classic_ros2_packages#spawn-model
- https://gazebosim.org/docs/harmonic/migrating_gazebo_classic_ros2_packages#bridge-ros-topics

Package ros_gz repository and examples (launch, model, plugin, …)

- https://github.com/gazebosim/ros_gz
- https://github.com/gazebosim/ros_gz/blob/ros2/ros_gz_sim_demos/launch/joint_states.launch.py
- https://github.com/gazebosim/ros_gz/blob/ros2/ros_gz_sim_demos/worlds/vehicle.sdf

SDF files

- https://gazebosim.org/docs/latest/sdf_worlds

Appendix:
Week 3 review

MyList Structure (Week 3)

- The function argument `p_list` was pointer of MyList structure.
  - We had to modify the structure outside the function memory.
- Since `p_list` is a pointer, we used (`p_list->len`) instead of (`p_list.len`).

listops.h

```
#define LIST_CAPACITY 64

struct MyList
{
    int len;
    int items[LIST_CAPACITY];
};
```

len: 5
items:

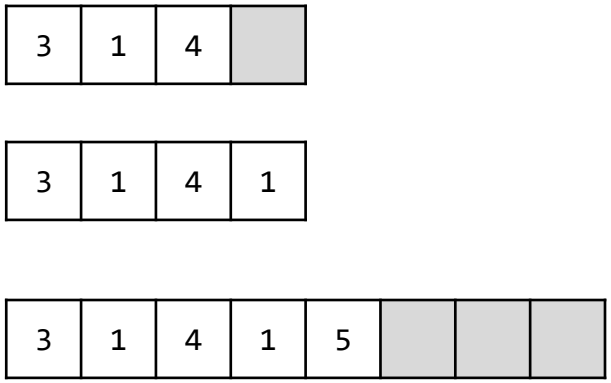
3	1	4	1	5			

listops.cpp

```
int append(MyList *p_list, int item)
{
    // return -1 for the two cases
    if (p_list->len >= LIST_CAPACITY) return -1;
    if (item < 0) return -1;

    // add the element to the list and increase the length
    p_list->items[p_list->len] = item;
    p_list->len++;

    return 0;
}
```

C++ Standard Vector

- Can we use dynamic array in C++?
- Array-based container that can change in size
  - Works similarly with our week 3 practice
- What if the memory is full?
  - It moves to the larger re-allocated memory




```
#include <vector>

// initialization
vector<int> vec = {3, 1, 4};

// append new element
vec.push_back(1);

// print 4th element of the vector
cout << vec.at(3) << endl;
```