ROS DAY1
2번 과제

2023741024
로봇학부 박건후

목차

(1) rclcpp의 hpp
(2) rclcpp의 cpp
(3) rclpy의 publish_py
(4) rclpy의 subscribe_py

(1) rclcpp의 hpp 파일

```
#pragma once
#include <rclcpp/rclcpp.hpp>
#include <std_msgs/msg/string.hpp>
#include <std_msgs/msg/int32.hpp>
#include <std_msgs/msg/float32.hpp>
class MyCppNode : public rclcpp::Node{
 public:
     MyCppNode();
 private:
     rclcpp::TimerBase::SharedPtr timer_;
     void timer_callback();
     rclcpp::Publisher<std_msgs::msg::String>::SharedPtr mycpp_publisher_;
     rclcpp::Publisher<std_msgs::msg::Int32>::SharedPtr mycpp_publisher_int;
     rclcpp::Publisher<std_msgs::msg::Float32>::SharedPtr mycpp_publisher_float;
     int count_ =1;



};
class subscribe_node: public rclcpp::Node{
    rclcpp::Subscription<std_msgs::msg::String>::SharedPtr mycpp_subscriber_;
    rclcpp::Subscription<std_msgs::msg::Int32>::SharedPtr mycpp_subscriber_int;
    rclcpp::Subscription<std_msgs::msg::Float32>::SharedPtr mycpp_subscriber_float;
    void topic_callback_str(const std_msgs::msg::String::SharedPtr msg);
    void topic_callback_int(const std_msgs::msg::Int32::SharedPtr msg);
    void topic_callback_float(const std_msgs::msg::Float32::SharedPtr msg);
    public:
    subscribe_node();
};
```

위 사진은 스트링, 정수, 실수 3 종류의 데이터를 보내기 위해 3개의 추가적인 헤더파일을 선언한 것입니다. String을 사용하기 위해, Int32를 사용하기 위해, float32를 사용하기 위해 선언하였습니다.

publish역할을 할 MyCppNode를 클래스로 설정하였고, subscribe역할을 할 subscribe_node를 클래스로 생성하였습니다. 모두 노드의 특성을 물려받기 위해 rclcpp의 Node클래스를 상속시켰습니다. 각각의 클래스에서 각기 다른 타입의 값을 동시에 통신해야 하기에 각기 다른 타입으로 템플릿된 변수를 만들었습니다. 그리고 MyCppNode에는 예제에 있던 코드처럼 1초마다 타이머를 돌게 하여 1초마다 publish를 할 timer_callback()메서드를 그대로 만들어 사용하였습니다. 반면, subscribe_node에서는 서로 다른 토픽에서 전달된 값들을 각각 캐치하기 위해 서로 다른 메서드들을 만들었고 이를 SharedPtr 멤버변수를 생성할 때 bind를 해당 메서드들로 함으로써 토픽을 받았다면 해당 메서드를 호출하여 터미널에 각각의 문자열을 출력하게 하였습니다.

(2) rclcpp의 cpp

```cpp
#include "pkg_hw2.hpp"
#include <chrono>
using namespace std::chrono_literals; //chrono_literals의 부분을 쉽게 사용하기 위해 선언

    MyCppNode:: MyCppNode():Node("publish_node"){
      RCLCPP_INFO(this->get_logger(), "Hello, ROS 2 C++ Node!"); //처음에 한번만 출력
      //각자 토픽 이름 생성하고 퍼블리셔 생성
      mycpp_publisher_ = this->create_publisher<std_msgs::msg::String>("topicname", 10);
      mycpp_publisher_int= this->create_publisher<std_msgs::msg::Int32>("int_topic", 10);
      mycpp_publisher_float= this->create_publisher<std_msgs::msg::Float32>("float_topic", 10
      timer_ = this->create_wall_timer(1s, std::bind(&MyCppNode::timer_callback, this)); //타이
    }
```

hpp파일을 인클루드 하고 chrono 헤더 파일을 인클루드 했습니다. chrono 헤더 파일을 사용하게 된 이유는 타이머의 시간 간격을 설정할 때 1초라는 것을 create_wall_timer에서 설정해야 했습니다.

rclcpp::create_wall_timer(std::chrono::duration<DurationRepT, DurationT> period,

위 사진은 create_wall_timer 함수를 ros2사이트에서 검색한 것에 대한 일부분입니다. 위 사진을 보시면 chrono를 사용함을 알 수 있습니다. 그리고 강의자료 예제 코드를 보면 1s로 전달을 했었습니다. 이 1s를 사용하는 방법을 알기 위해 인터넷 검색을 하여 https://bongbong-89.tistory.com/53 url에서 아래 사진에 대한 부분을 참고했습니다.

```cpp
#include <iostream>
#include <chrono>
#include <string>

using namespace std::chrono_literals;
using namespace std::string_literals;

int main() {
    auto duration = 10s;
    auto str = "example"s;

    std::cout << "Duration: " << duration.count() << " seconds" << std::endl;
    std::cout << "String: " << str << std::endl;

    return 0;
}
```

```cpp
#include "pkg_hw2.hpp"
#include <chrono>
using namespace std::chrono_literals; //chrono_literals의 부분을 쉽게 사용하기 위해 선언

    MyCppNode:: MyCppNode():Node("publish_node"){
      RCLCPP_INFO(this->get_logger(), "Hello, ROS 2 C++ Node!"); //처음에 한번만 출력
      //각자 토픽 이름 생성하고 퍼블리셔 생성
      mycpp_publisher_ = this->create_publisher<std_msgs::msg::String>("topicname", 10);
      mycpp_publisher_int= this->create_publisher<std_msgs::msg::Int32>("int_topic", 10);
      mycpp_publisher_float= this->create_publisher<std_msgs::msg::Float32>("float_topic", 10
      timer_ = this->create_wall_timer(1s, std::bind(&MyCppNode::timer_callback, this)); //타이
    }
```

아래의 부분은 해당 클래스의 생성자 부분입니다. 우선 Node클래스를 상속받았으므로 Node라는 클래스가 부모클래스이기에 이를 생성자를 통해 부모 인스턴스를 생성해줘야 합니다. 이 때 Node생성자에는 노드 이름을 입력할 수 있습니다. 그래서 노드 이름을 publish_node로 지어서 작성해줬습니다. 이 부분은 CMakeList의 내용에도 중요한 영향을 미치므로 꼭 기억해야 합니다. 노드란 프로그램을 의미하고, 노드명은 파일명과는 아무런 상관이 없기 때문입니다. 그리고 또한 여러 실행할 파일명들을 묶어 노드를 구성하는 것이고 이는 CMakeList의 add_executable에 이를 작성해줘야 합니다.

노드를 생성하였다면 각각의 정보를 퍼블리시 해줄 퍼블리시 객체들에 대한 SharedPtr을 생성해줄 차례입니다. 기본적인 shared_ptr을 생성할 때도 make_shared라는 헬퍼 함수가 있듯이 여기에서도 그런 역할을 하는 함수가 있는 것 같습니다. 강의자료 예제를 참고하여 해당 함수를 작성하여 각각의 포인터를 생성했습니다. 이 때 같은 토픽명으로 생성할 경우에는 한꺼번에 같은 토픽명으로 퍼블리시할 경우 subscribe하는 과정에서 값이 깨지거나 문제가 생길 수 있습니다. 그래서 각각의 토픽명을 정하고 다른 스트림을 통해 전달하고자 했습니다. 토픽명은 create_publisher헬퍼함수의 매개변수에서 작성하면 됩니다. 그리고 2번째 매개변수에서는 큐의 크기를 나타낸 것인데, 퍼블리시를 하고 제때 처리되지 못한 경우 이를 큐에 저장하게 됩니다. 이 때 큐의 크기를 설정하는 것을 해당 매개변수에서 진행합니다.

그리고 create_wall_timer라는 함수를 통해 1초마다 정의한 timer_callback메서드가 호출되도록 설정하는 부분입니다. 해당 timer_callback메서드는 멤버함수이기에 bind를 할 때 매개변수를 받는 것이 아니더라도 첫 매개변수로 꼭 this를 명시해야 합니다. 또한 bind를 할 때 함수명을 작성하다보니, 함수 객체의 부류인 람다나 function<>클래스나 함수 포인터, 함수 객체 등이 올 수 있습니다.

```cpp
void MyCppNode:: timer_callback(){ //1초마다 호출되는 콜백함수
   //각기 메시지 담을 그릇 생성
  auto msg = std_msgs::msg::String();
  auto num = std_msgs::msg::Int32();
  auto fl= std_msgs::msg::Float32();
  //메시지 담기
  msg.data= "Hello World: "+ std::to_string(count_++);
  num.data = 3456+count_;
  fl.data = 3.4567+count_;
  //터미널 출력
  RCLCPP_INFO(this->get_logger(), "Published message: '%s', int: '%d', float: '%f'", msg.
  //각각 퍼블리시
  mycpp_publisher_ -> publish(msg);
  mycpp_publisher_int->publish(num);
  mycpp_publisher_float->publish(fl);
}
```

이번에는 timer_callback 메서드입니다. 각각의 메시지를 담을 객체를 생성하고 해당 데이터를 담았습니다, 처음에는 실제 const char* 문자열, int, float를 담으면 되는 것으로 접근하였지만 안 된다는 것을 알고 위 사진처럼 헤더파일을 선언하고 사용했습니다. 터미널에 출력하기 위해 강의자료 예제에 있던 RCLCPP_INFO를 사용하여 출력하였고 각각에 대한 정보를 1초마다 토픽으로 전달해야 하기에 각각의 포인터로 publish하는 부분입니다.

```
//각각의 받는 메서드 정의: 터미널 출력함. 메시지를 받았고 이를 출력하는 것, SharedPtr로 전달 받은 객체 가리킴
void subscribe_node:: topic_callback_str(const std_msgs::msg::String::SharedPtr msg){

  RCLCPP_INFO(this->get_logger(), "Received message: '%s'", msg->data.c_str());
}
void subscribe_node:: topic_callback_int( const std_msgs::msg::Int32::SharedPtr num){

  RCLCPP_INFO(this->get_logger(), "Received message: '%d'",  num->data);
}
void subscribe_node:: topic_callback_float( const std_msgs::msg::Float32::SharedPtr fl){

  RCLCPP_INFO(this->get_logger(), "Received message: '%f'", fl->data);
}
//노드 이름을 subscribe_node로 지정, 각자 토픽 이름 생성하고 서브스크라이버 포인터 생성
subscribe_node:: subscribe_node():Node("subscribe_node"){
   mycpp_subscriber_ = this->create_subscription<std_msgs::msg::String>("topicname", 10,
   mycpp_subscriber_int =  this->create_subscription<std_msgs::msg::Int32>("int_topic",
   mycpp_subscriber_float =  this->create_subscription<std_msgs::msg::Float32>("float_to
}
```

이제 subscribe_node의 구현 부분입니다. 각각의 토픽에서 값을 전달받았다면 bind된 메서드를 호출하게 되는데 각각의 bind된 메서드를 구현한 부분입니다. topic_callback_~ 메서드 모두 자신이 메시지를 완벽히 받고 받은 값을 터미널에 출력해야 잘 받았다는 것을 확인할 수 있기에 터미널에 출력하는 것을 작성하였습니다.

그리고 아래 부분은 생성자입니다. MyCppNode 클래스와 마찬가지로 노드명을 작성하고 해당 토픽이름이 맞아야 통신이 가능하기에 MyCppNode에서 작성한 각각의 토픽명을 대응시켜 작성하였습니다. 이번에는 subscribe의 SharedPtr 타입을 생성해주는 함수로 create_subscription을 사용하여 만들었습니다.

```
10, std::bind(&subscribe_node::topic_callback_str, this, std::placeholders::_1));
c", 10, std::bind(&subscribe_node::topic_callback_int, this, std::placeholders::_1));
t_topic", 10, std::bind(&subscribe_node::topic_callback_float, this, std::placeholders::_1
```

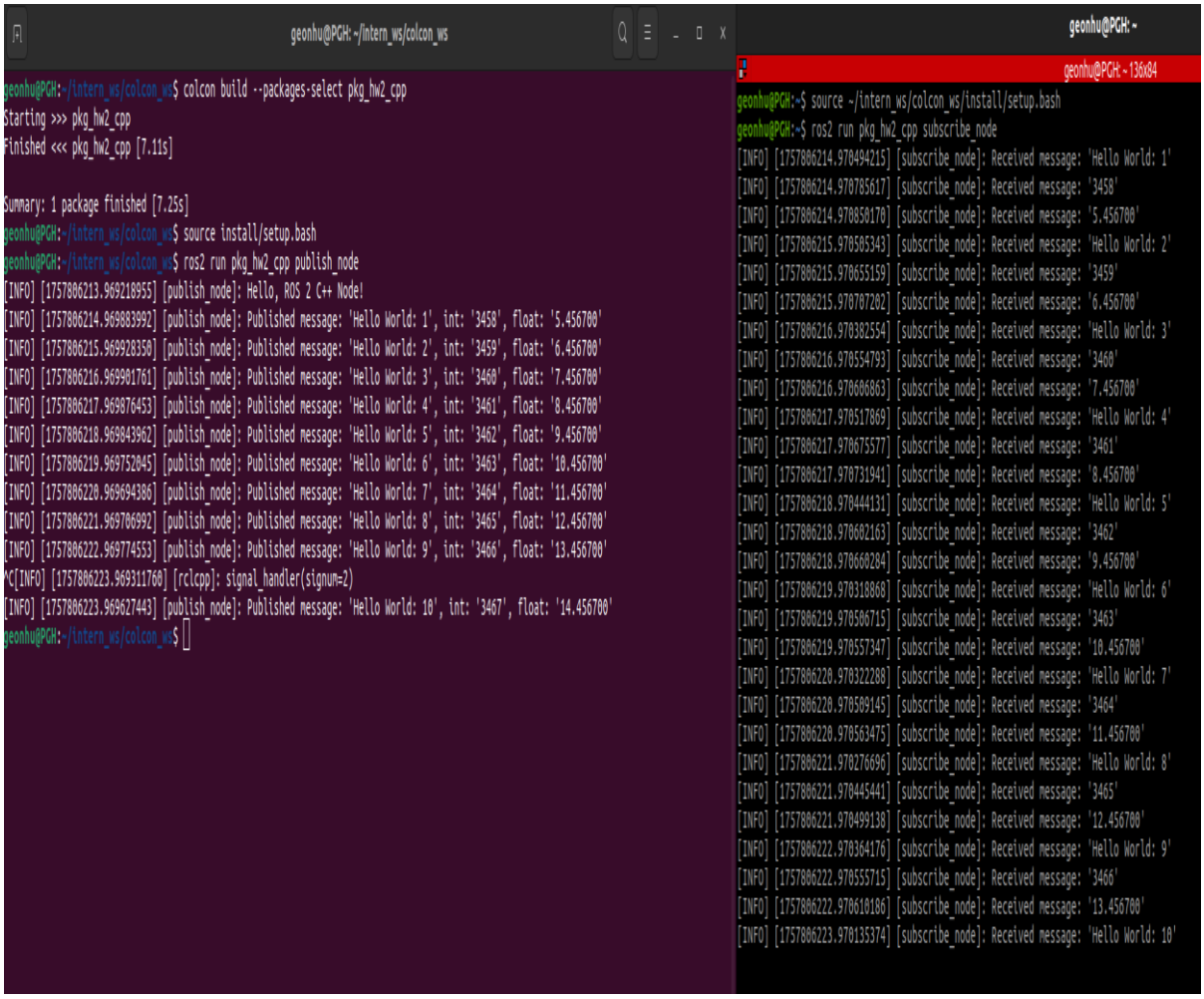
위 사진은 create_subscription을 사용한 것의 매개변수 뒷 부분입니다. bind를 했음을 보여드리려 했습니다. 각각의 subscribe 객체가 생성될텐데, 각각의 subscribe 객체가 해당 토픽명에서 퍼블리시된 메시지를 받았을 때 동작을 해야 하고 그 동작을 하기 위해 메서드를 bind한 것입니다. bind할 때는 멤버변수이기에 this를 넣고, 포인터를 통해 전달받은 메시지를 가리켜야 하기에 각 메서드의 매개변수를 그대로 넣어줘야 합니다. 그래서 std::placeholders::_1을 사용하였습니다.

```cpp
#include "pkg_hw2.hpp"
int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<MyCppNode>(); //퍼블리셔 노드 생성
    rclcpp::spin(node); //spin하여 계속 프로그램이 돌아가도록 함
    rclcpp::shutdown();
    return 0;
}
```

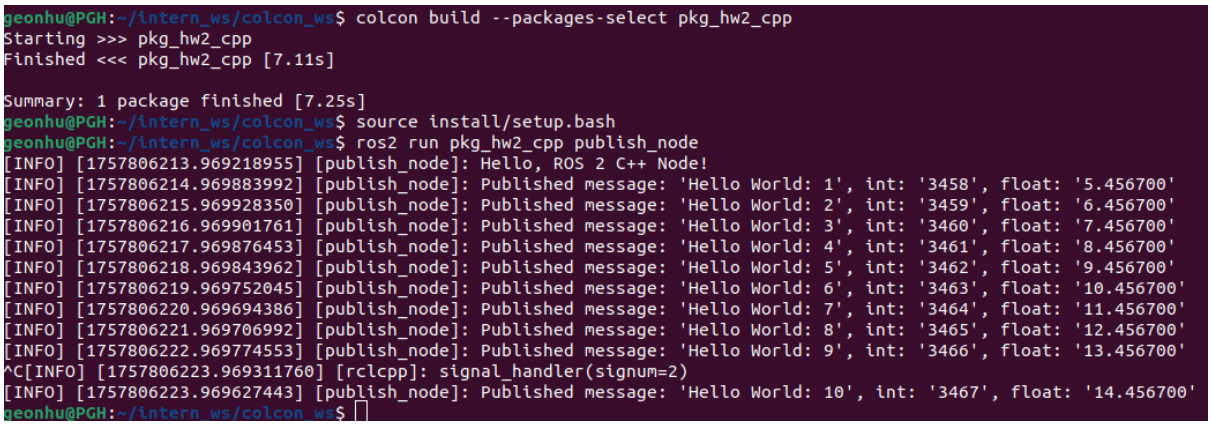
각각의 터미널에서 서로 다른 노드를 실행시켜야 하고 각각의 노드를 실행시키기 위해서는 각각의 main이 있어야 합니다. 현재 위의 사진에서는, publish에 대한 main으로, publish를 하는 클래스에 대한 객체를 생성하기 위해 make_shared로 shared_ptr을 생성하여 해당 객체를 생성 및 가리키게 하는 것입니다. 그리고 spin()을 통해 해당 프로그램이 계속 돌아가게 하여 1초마다 터미널에 출력 및 퍼블리시하게 하였습니다.

```cpp
#include "pkg_hw2.hpp"

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    // 여기서 퍼블리셔 대신 서브스크라이버 노드를 실행
    auto node = std::make_shared<subscribe_node>();
    rclcpp::spin(node); //spin하여 계속 대기하도록 함
    rclcpp::shutdown();

    return 0;
}
```

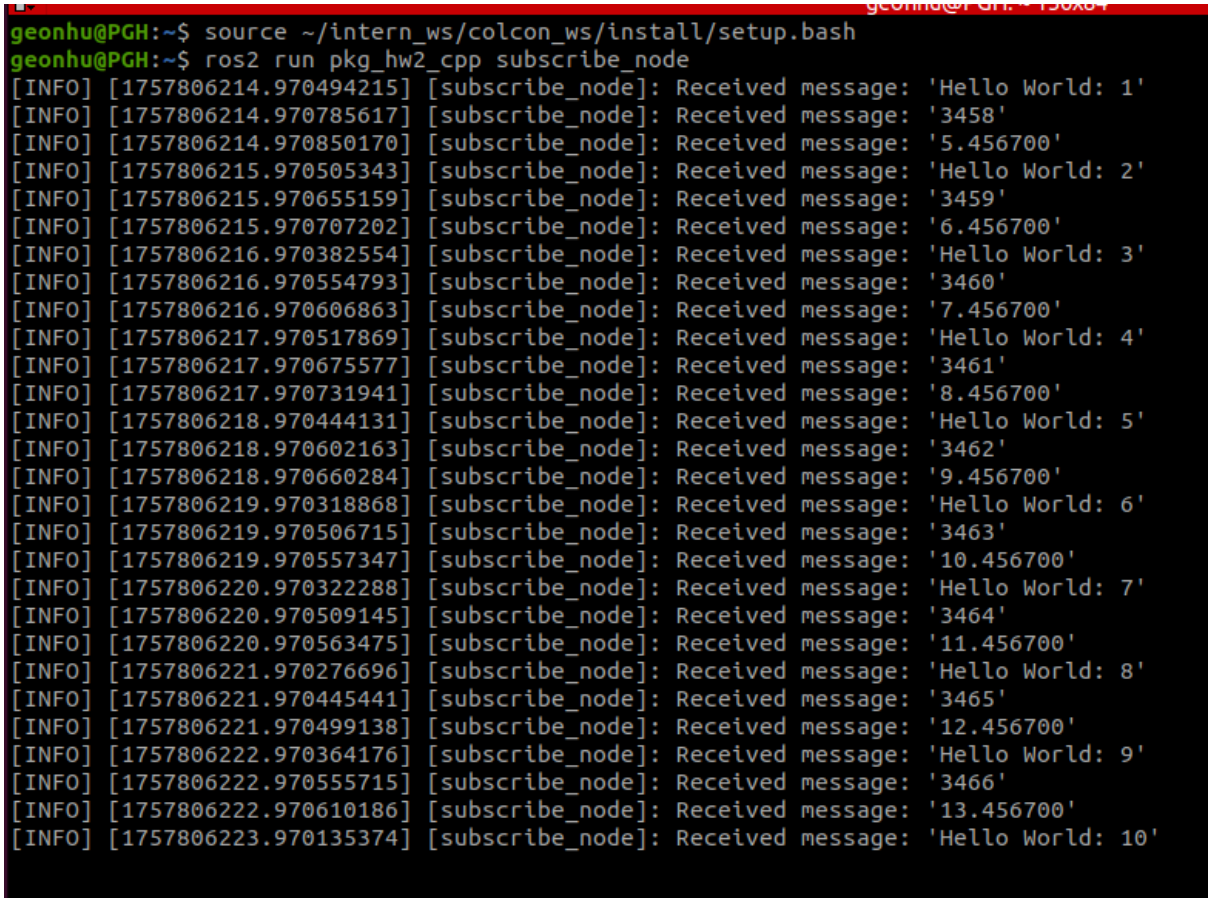
이번에는 subscribe에 대한 main으로, publish에 대한 main과 같은 방식입니다.



위 사진은 하나의 화면에서 2개의 터미널 창을 띄워서 cpp패키지 내에서 publish_node의 노드와 subscribe_node노드끼리의 통신하는 것을 나타낸 사진입니다.



위 사진은 왼쪽의 터미널 창을 확대한 것으로 publish_node가 Hello World와 count 변수값을 문자열로 바꿔 1씩 증가하는 문자열을 띄우게 하며 int: %d, float: %f 으로 변환기호를 넣어서 count 변수값을 사용하여 매번 1씩 증가된 값을 보내는 과정입니다.



이 사진은 오른쪽 터미널 창을 확대한 부분으로 subscribe_node로써 받은 값을 잘 받고 있는 것을 나타낸 사진입니다.

(3) rclpy의 publish_py

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
from std_msgs.msg import Int32
from std_msgs.msg import Float32
```

publish하는 publish.py파일의 import하는 부분입니다. rclpy의 node클래스를 import하고 위의 cpp로 작성할 때처럼 3개 이상의 타입의 메시지를 통신해야 하기에 String, Int32, Float32를 import하였습니다.

```
class PyTalker(Node):
    def __init__(self):
        super().__init__('py_talker') #Node 인스턴스 생성 및 노드 네임 설정

        #퍼블리셔 할 멤버 생성
        self.pub_string = self.create_publisher(String, 'topicname', 10)
        self.pub_int = self.create_publisher(Int32, 'int_topic', 10)
        self.pub_fl = self.create_publisher(Float32, 'float_topic', 10)
        self.timer = self.create_timer(1.0, self.on_timer) #1초마다 on_timer 호출되도록 설정
        self.count = 0
        self.get_logger().info('PyTalker up') #터미널에 한 번만 초기에 출력
```

publish역할을 해줄 클래스 정의하는 부분입니다. super()로 상속받은 Node클래스에 대해 __init__()을 통해 부모 인스턴스를 생성하고 cpp파일에서 했었던 것처럼 노드명을 적어줬습니다.

그리고 publish해줄 멤버를 선언, 정의를 해야 합니다. 그래서 create_publisher를 통해 생성해줬습니다. 이를 작성할 때 강의자료에 있던 create_publisher를 참고하여 작성하였습니다. 2번째 매개변수를 작성할 때는 토픽명을 작성하게 되는데 과제 2번에서 하나의 패키지 내에서 통신하는 것뿐만 아니라 cpp 패키지와도 통신해야 하기 때문에 토픽명을 일치시켜야 합니다. 그래서 topicname, int_topic, float_topic으로 각각 대응시켜 일치되도록 작성하였습니다.

```
self.timer = self.create_timer(1.0, self.on_timer) #1초마다 on_timer 호출되도록 설정
self.count = 0
self.get_logger().info('PyTalker up') #터미널에 한 번만 초기에 출력
```

publish할 멤버를 작성한 후에는 전과 같이 1초마다 터미널에 출력하면서 publish를 하기 위해 예제 코드를 참고하여 작성하였습니다. 그리고 publish를 시작했다는 것을 표시하기 위해 초기에 PyTalker up이라는 메시지를 출력하도록 했습니다.

```
def on_timer(self):
    #메시지 생성
    msg = String()
    num = Int32()
    fl= Float32()
    msg.data = f'Hello World_py: {self.count}'
    num.data = 9000
    fl.data = 0.89
    self.count += 1
    #터미널에 매번 출력
    self.get_logger().info(f'Published: str: {msg.data}, num: {num.data}, float: {fl.data}')
    #메시지 발행
    self.pub_string.publish(msg)
    self.pub_fl.publish(fl)
    self.pub_int.publish(num)
```

1초마다 터미널에 출력하고 publish해야 합니다. 그래서 바인드된 on_timer메서드를 이에 맞게 정의하였습니다. 각각의 타입의 정보를 저장하기 위해 String, Int32, Float32 객체를 생성하여 값을 넣고, 터미널에 출력하고, publish 역할을 하는 멤버들을 통해 각각을 publish해줬습니다.

```
def main():
    rclpy.init()
    node = PyTalker() #객체 생성
    rclpy.spin(node) #노드가 종료될 때까지 돌아감
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main() #위에 정의한 main 함수 호출
```

이 부분은 main을 정의한 부분입니다. 그리고 main을 호출함으로써 해당 파일만으로 클래스를 정의하고 main을 호출할 수 있는 것입니다. main함수에서는 객체를 생성하고 이를 계속 돌아가게 하였습니다.

```
if __name__ == '__main__':
    main() #위에 정의한 main 함수 호출
```

이 부분에서는 예제 코드를 참고한 것이지만 이해가 가지 않아 따로 인터넷 검색을 해보았습니다. https://wikidocs.net/195615 이 url을 통해 알게 된 사실로는 __name__이라는 것은 모듈의 이름을 담고 있는 내장 변수인데, 현재 publish.py라는 파일, 모듈이 실행되면 __name__에 "__main__"이라는 문자열이 할당된다고 합니다. 그래서 이 모듈이 실행될 때 __name__의 값이 바뀌면서 이 if문이 참이 되어 main을 호출하여 PyTalker의 퍼블리시 하는 것이 1초마다 진행될 수 있는 것입니다.

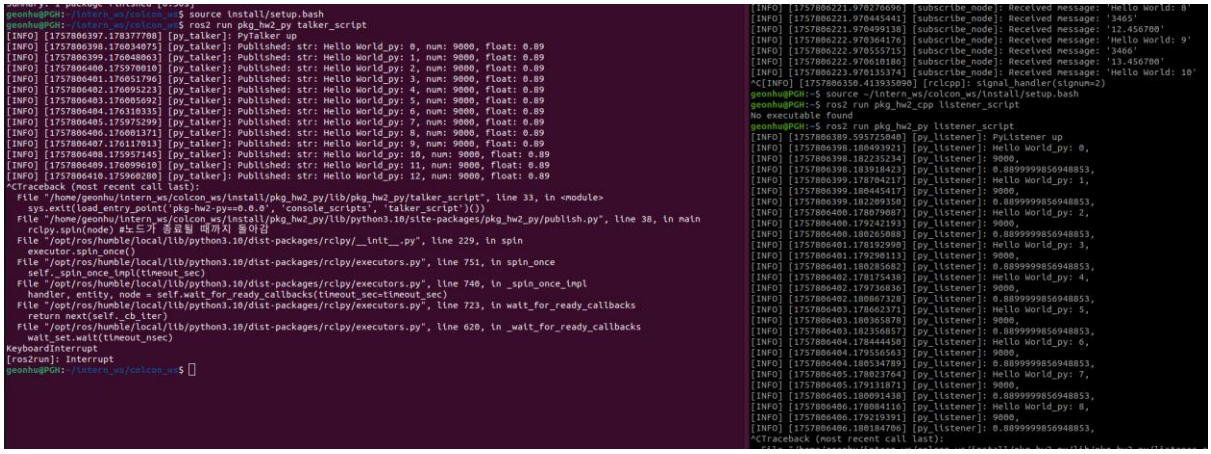
(4) rclpy의 subscribe_py

```
class PyListener(Node): #Node 상속, subscribe 구현
    def __init__(self):
        super().__init__('py_listener') #Node 인스턴스 생성 및 노드 네임 설정
        #subscribe 할 멤버 생성 및 콜백함수 설정
        self.sub_str = self.create_subscription(String, 'topicname', self.on_msg, 10)
        self.sub_int = self.create_subscription(Int32, 'int_topic', self.on_msg, 10)
        self.sub_float = self.create_subscription(Float32, 'float_topic', self.on_msg, 10)
        self.get_logger().info('PyListener up') #터미널에 한 번만 초기에 출력

    def on_msg(self, msg): #메시지 수신 시 호출되는 콜백함수
        self.get_logger().info("{0}, ".format(msg.data))
```

이번에는 subscriber.py 파일의 내용입니다. 이 파일에서도 기존 import한 것들을 똑같이 import해줬습니다. 그리고 subscribe역할을 해줄 PyListener이라는 클래스를 만들어줬습니다. 그리고 노드명을 py_listener로 해주었고, create_subscription을 사용하여 토픽명을 cpp파일의 토픽명과 일치시켜줬습니다. 그리고 subscribe역할을 하는 멤버를 생성할 때는 멤버가 토픽을 받았을 때마다 호출해줄 콜백함수를 항상 지정해줘야 합니다. 콜백함수로 on_msg라는 메서드를 만들어 지정해줬습니다. 파이썬에서는 자료형을 적어주지 않기 때문에 받는 인수에 따라 동적으로 처리해줍니다. 그래서 타입에 대한 메서드를 나눠 작성하지 않고 on_msg라는 메서드 하나로 모든 타입을 처리할 수 있었습니다. 터미널에 출력해줄 때는 받은 객체의 데이터를 출력하도록 해줬습니다.

```python
# main 정의
def main():
    rclpy.init()
    node = PyListener() #객체 생성
    rclpy.spin(node) #노드가 종료될 때까지 돌아감
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main() #위에 정의한 main 함수 호출
```

main을 정의한 부분입니다. PyListener라는 객체를 생성하고 프로그램이 멈추지 않고 계속 돌아가도록 spin을 넣어줬습니다. 그리고 publish.py처럼 if절을 만들어 main()을 바로 실행되도록 만들어줬습니다.



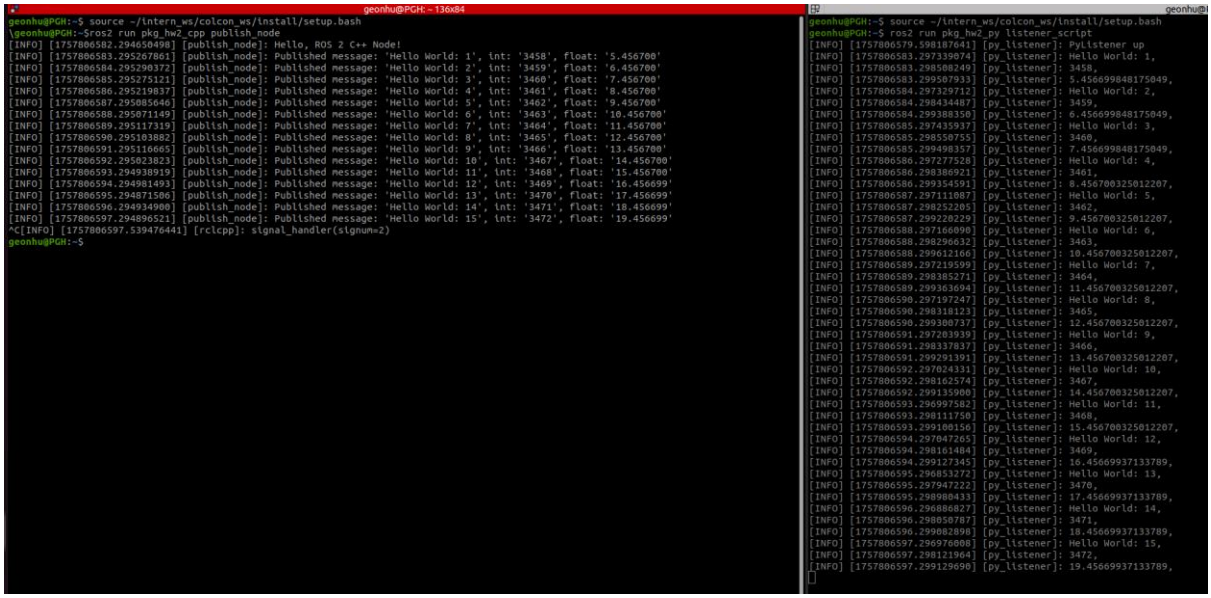
이 사진은 파이썬 패키지 내에서 py파일끼리 하는 통신을 나타낸 사진입니다.

```
geonhu@PGH:~/intern_ws/colcon_ws$ source install/setup.bash
geonhu@PGH:~/intern_ws/colcon_ws$ ros2 run pkg_hw2_py talker_script
[INFO] [1757806397.178377708] [py_talker]: PyTalker up
[INFO] [1757806398.176034075] [py_talker]: Published: str: Hello World_py: 0, num: 9000, float: 0.89
[INFO] [1757806399.176048063] [py_talker]: Published: str: Hello World_py: 1, num: 9000, float: 0.89
[INFO] [1757806400.175970010] [py_talker]: Published: str: Hello World_py: 2, num: 9000, float: 0.89
[INFO] [1757806401.176051796] [py_talker]: Published: str: Hello World_py: 3, num: 9000, float: 0.89
[INFO] [1757806402.176095223] [py_talker]: Published: str: Hello World_py: 4, num: 9000, float: 0.89
[INFO] [1757806403.176005692] [py_talker]: Published: str: Hello World_py: 5, num: 9000, float: 0.89
[INFO] [1757806404.176310335] [py_talker]: Published: str: Hello World_py: 6, num: 9000, float: 0.89
[INFO] [1757806405.175975299] [py_talker]: Published: str: Hello World_py: 7, num: 9000, float: 0.89
[INFO] [1757806406.176001371] [py_talker]: Published: str: Hello World_py: 8, num: 9000, float: 0.89
[INFO] [1757806407.176117013] [py_talker]: Published: str: Hello World_py: 9, num: 9000, float: 0.89
[INFO] [1757806408.175957145] [py_talker]: Published: str: Hello World_py: 10, num: 9000, float: 0.89
[INFO] [1757806409.176099610] [py_talker]: Published: str: Hello World_py: 11, num: 9000, float: 0.89
[INFO] [1757806410.175960280] [py_talker]: Published: str: Hello World_py: 12, num: 9000, float: 0.89
^CTraceback (most recent call last):
  File "/home/geonhu/intern_ws/colcon_ws/install/pkg_hw2_py/lib/pkg_hw2_py/talker_script", line 33, in <module>
```

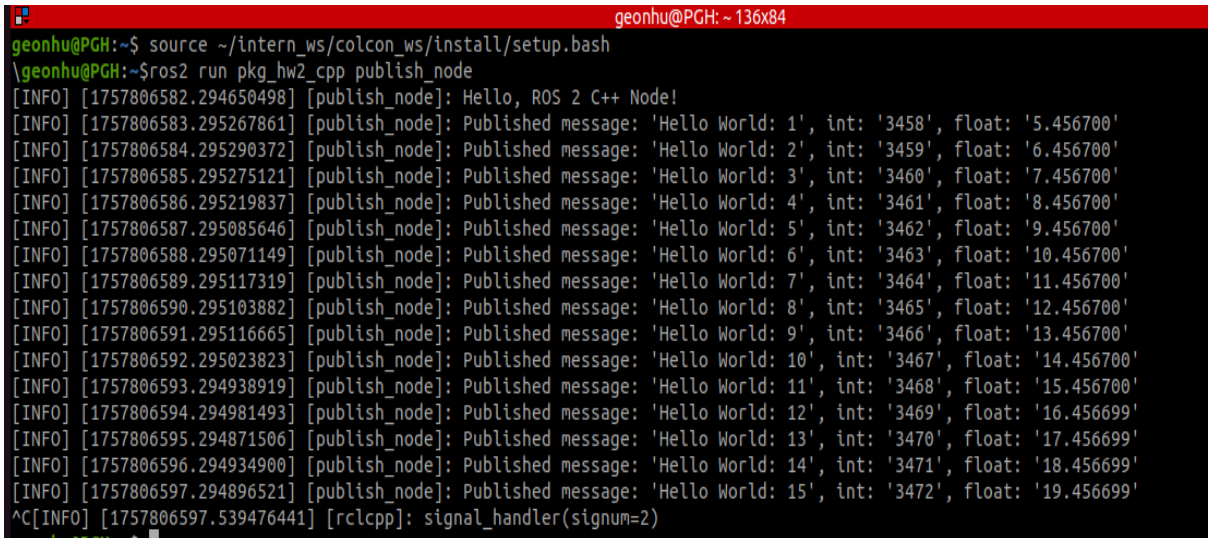
이번에는 Hello World_py: 에 count 변수를 통해 값이 증가된 문자열을 출력하도록 하였습니다. 그리고 정수값, float값 모두 출력하도록 하였습니다.

```
geonhu@PGH:~$ ros2 run pkg_hw2_py listener_script
[INFO] [1757806389.595725040] [py_listener]: PyListener up
[INFO] [1757806398.180493921] [py_listener]: Hello World_py: 0,
[INFO] [1757806398.182235234] [py_listener]: 9000,
[INFO] [1757806398.183918423] [py_listener]: 0.8899999856948853,
[INFO] [1757806399.178704217] [py_listener]: Hello World_py: 1,
[INFO] [1757806399.180445417] [py_listener]: 9000,
[INFO] [1757806399.182209350] [py_listener]: 0.8899999856948853,
[INFO] [1757806400.178079087] [py_listener]: Hello World_py: 2,
[INFO] [1757806400.179242193] [py_listener]: 9000,
[INFO] [1757806400.180265088] [py_listener]: 0.8899999856948853,
[INFO] [1757806401.178192990] [py_listener]: Hello World_py: 3,
[INFO] [1757806401.179290113] [py_listener]: 9000,
[INFO] [1757806401.180285682] [py_listener]: 0.8899999856948853,
[INFO] [1757806402.178175438] [py_listener]: Hello World_py: 4,
[INFO] [1757806402.179736836] [py_listener]: 9000,
[INFO] [1757806402.180867328] [py_listener]: 0.8899999856948853,
[INFO] [1757806403.178662371] [py_listener]: Hello World_py: 5,
[INFO] [1757806403.180365878] [py_listener]: 9000,
[INFO] [1757806403.182356857] [py_listener]: 0.8899999856948853,
[INFO] [1757806404.178444450] [py_listener]: Hello World_py: 6,
[INFO] [1757806404.179556563] [py_listener]: 9000,
[INFO] [1757806404.180534789] [py_listener]: 0.8899999856948853,
[INFO] [1757806405.178023764] [py_listener]: Hello World_py: 7,
[INFO] [1757806405.179131871] [py_listener]: 9000,
[INFO] [1757806405.180091438] [py_listener]: 0.8899999856948853,
[INFO] [1757806406.178084116] [py_listener]: Hello World_py: 8,
[INFO] [1757806406.179219391] [py_listener]: 9000,
[INFO] [1757806406.180184706] [py_listener]: 0.8899999856948853,
^CTraceback (most recent call last):
  File "/home/geonhu/intern_ws/colcon_ws/install/pkg_hw2_py/lib/pkg_hw2_py/lis
    sys.exit(load_entry_point('pkg-hw2-py==0.0.0', 'console_scripts', 'listen
  File "/home/geonhu/intern_ws/colcon_ws/install/pkg_hw2_py/lib/python3.10/si
```

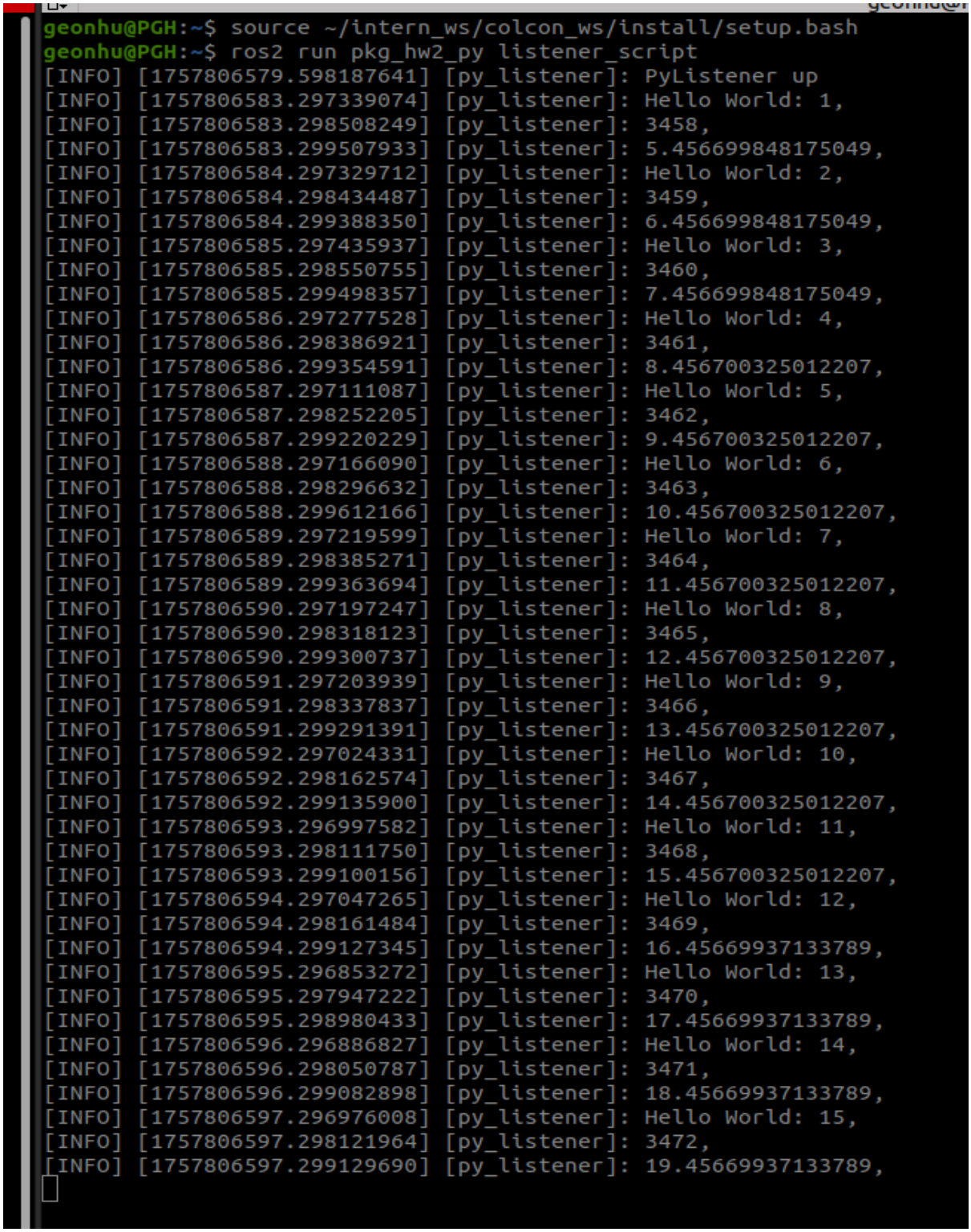
이 부분은 터미널의 오른쪽 창으로써, 퍼블리시가 보낸 것을 잘 읽고 있는 것을 나타낸 사진입니다.



이 부분은 하나의 화면에 서로 다른 터미널 창을 띄워 cpp와 py인 서로 다른 패키지에서 통신하는 것을 나타낸 부분입니다.



기존의 publish_node가 보내던 방식으로 잘 보내고 있고



py의 subscribe 역할을 하는 노드가 PyListener Up을 생성자 생성 당시 출력하고 그 다음부터는 전달 받은 대로 출력하며 정상적으로 통신이 된다는 것을 확인할 수 있습니다.