ROS DAY2
과제 3번 보고서

2023741024
로봇학부 박건후

목차

(1) qnode.hpp 파일 설명

저는 문제에 접근할 때 publish하는 노드와 subscribe하는 노드가 있고 publish하는 노드는 터미널 창에서 실행을 시켜 터미널 창 내에서 계속 spin을 돌며 대기하도록 버튼이라는 이벤트를 발생시키면 그 때 publish하는 방식으로 생각하며 이 문제에 접근했습니다. 그리고 subscribe_node에서는 터미널 창에서 실행되자마자 ui윈도우 창이 열리며 계속 spin을 돌며 메시지가 들어올 때까지 대기하다가 메시지를 받으면 그 메시지 내용을 MainWindow의 slot메서드에 정보를 전달하고 그 정보를 그대로 출력창에다가 출력하는 방식으로 문제를 풀고자 했습니다. 과제 4번을 하고 과제 3번을 하다보니 2개의 클래스를 다중 상속한다거나 서브스크라이버 역할을 하는 클래스의 콜백 메서드에서 시그널을 보내며 MainWindow에서 publish로, subscribe에서 MainWindow로 connect를 통해 양방향으로 정보를 보내는 것으로 확장하는 코드를 짤 수 있었습니다. 그래서 3번 과제를 보실 때 이 점 양해해주시고 오해 없이 보고서를 봐주시면 감사드리겠습니다.

```
class publisher: public QObject, public rclcpp::Node{ //publisher 클래스 선언
    Q_OBJECT    //버튼을 누르면 이 클래스의 connect되어 publish_message메서드가 slot메서
    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub;
    public:
    Ui::MainWindow* ui; //ui에 접근하여 버튼을 눌렀을 때 입력창의 문자열을 가져오기 위함
    publisher();
    public slots: //slots설정 위해 QObject상속 받음
    void publish_message(); //publish하는 메서드. ui->text로 입력창에 적힌 것 읽음
};
```

위 사진은 publisher 클래스의 선언부입니다. 버튼 이벤트를 발생시키면 그것이 ros부문의 publisher까지 버튼이 눌렸다는 사실이 전달되어야 하므로 connect를 활용하고자 했습니다. 버튼이 눌리면 바로 publish를 할 수 있게 만들고 싶었습니다. 그래서 slot메서드로 등록하는 기능을 publisher에서 가능하게 하기 위해 QObject를 상속하고 Node 클래스를 상속받았습니다. 그리고 publish하는 메서드인 publish_message()메서드를 slot메서드로 등록하였습니다. 이 때 MainWindow에서 publisher의 멤버함수를 참조하는 것이기에 public slots으로 해야 접근할 수 있기에 public slots로 설정하고 publish_message()를 작성하였습니다. 하지만 버튼을 눌렀을 때 어떤 문자열이 적힌 것까지는 publish_message에게 전달하기 쉽지 않아보여서 이전에 썼던 방법대로, publish_message()에서 ui를 참조하여 입력창에 적힌 문자열 정보를 가져올 수 있도록 ui 포인터를 publisher 클래스에 public으로 선언하여 MainWindow클래스에서 publisher클래스의 ui포인터를 MainWindow 클래스의 ui포인터로 초기화하여 같은 윈도우 창을 가리키도록 하였습니다.

이번에 publish_node가 subscribe_node에게 전달해야 할 정보의 타입은 std_msgs::msg::String으로 전달해야 하기에 publish를 해줄 포인터를 인스턴스화할 때 해당 타입을 한 것입니다.

```
class subscribe_node: public QObject, public rclcpp::Node{
  Q_OBJECT //이거 써야 함
  rclcpp::Subscription<std_msgs::msg::String>::SharedPtr sub;
  void topic_callback(const std_msgs::msg::String::SharedPtr msg); //
  signals:
  void signal_func(QString str); //signals 설정하기 위해 QObject상속받음


  public:
  subscribe_node();
};
```

위 사진은 subscribe_node 클래스의 선언부입니다. 여기서는 메시지를 받으면 콜백 메서드를 호출하고 콜백 메서드가 끝나면 해당 정보는 소멸되기에 콜백 메서드에서 받은 메시지의 정보를 MainWindow에 넘겨서 이 정보를 출력창에 출력할 수 있도록 작성하였습니다. 그렇게 하기 위해서는 connect 기능을 사용해야 했고, 콜백함수가 호출되었을 때, 거기서 시그널을 보내야 MainWindow의 slot이 호출되며 시그널과 슬롯 메서드 간에 매개변수가 일치해야 매개변수를 통해서 정보를 보낼 수 있었습니다. 그래서 signal메서드를 사용해야 했기에 이렇게 QObject를 상속받게 하였고 그리고 Node를 상속받게 했습니다.

시그널 함수에서는 문자열을 MainWindow에 전달해야 하기 때문에 QString 타입의 매개변수를 설정하였습니다.

```
class QNode : public QThread
{
  Q_OBJECT
public:
  QNode();
  ~QNode();
  std::shared_ptr<subscribe_node> sub_ptr;
  std::shared_ptr<publisher> pub_ptr;  //main_window에서 p
protected:
  void run();

private:

Q_SIGNALS:
  void rosShutDown();
};
```

QNode 클래스의 선언부입니다. MainWindow에서는 pub_ptr로 ui 포인터를 초기화시켜줘야하고 또 publish_message()라는 슬롯 메서드를 connect로 참조할 때도 pub_ptr이 필요하기에 선언하였습니다. sub_ptr 또한 connect로 연결할 때 필요했습니다. 이렇게 qt-QNode-ros 부문 간의 연결을 완료해주었습니다.

(2) qnode.cpp 파일 설정

```
QNode::QNode()
{
  int argc = 0;
  char** argv = NULL;
  rclcpp::init(argc, argv);
  sub_ptr = std::make_shared<subscribe_node>();  //subscribe_node 객체 생성
  pub_ptr = std::make_shared<publisher>();       //publisher 객체 생성
  this->start();
}

QNode::~QNode()
{
  if (rclcpp::ok())
  {
    rclcpp::shutdown();
  }
}

void QNode::run()
{
  rclcpp::WallRate loop_rate(20);
  while (rclcpp::ok())
  {
    rclcpp::spin_some(sub_ptr);  //여기서 subscribe_node가 대기함
    loop_rate.sleep();
  }
  rclcpp::shutdown();
  Q_EMIT rosShutDown();
}
```

QNode 클래스 구현부분입니다. make_shared를 통해 각각의 클래스의 객체를 생성하였습니다. 그리고 publish_node가 터미널 창에서 대기하고 subscribe_node가 ui 윈도우 창을 띄우고 대기하는 것이므로 QNode::run에서의 spin은 subscriber_node의 스마트 포인터가 들어가야 합니다.

```
publisher::publisher():Node("publish_node"){ // publisher 클래스 생성자, 노드 이름 설정
    pub = this->create_publisher<std_msgs::msg::String>("hw3_pkg",10); //토픽 이름 설정
}
void publisher::publish_message(){

    std_msgs::msg::String str;
    QString s = ui->input->text();   //ui에서 받아와서 저장
    str.data = s.toStdString();      //문자열로 변환해서 보낼 변수에 저장
    pub->publish(str);
}
subscribe_node::subscribe_node():Node("subscribe_node"){

    sub = this->create_subscription<std_msgs::msg::String>("hw3_pkg",10,std::bind(&subs
}
void subscribe_node:: topic_callback( const std_msgs::msg::String::SharedPtr msg){
   emit signal_func(QString::fromStdString(msg->data)); //callback에서 emit함 메시지는 콜백

}
```

publisher 클래스의 구현부입니다. 생성자에서는 기존에 하던대로 publish_node로 노드명을 생성해주고 토픽명으로 hw3_pkg로 설정하였습니다. 그리고 publish_message에서는 ui포인터를 통해 입력창의 문자열을 읽어와서 저장한 후 QString의 문자열을 받아오는 메서드를 통해 data멤버에 저장할 수 있었습니다. 그리고 이렇게 저장된 문자열을 publish하도록 했습니다.

subscribe_node의 생성자에서는 노드 이름을 정하고 토픽명을 일치시켰습니다. 그리고 콜백 메서드를 정의할 때는, 정보를 가리키는 msg매개변수로 문자열을 가져오기 위해 msg->data를 사용하였고 전달할 때 시그널에서 전달할 때 QString으로 전달해야 하므로 문자열을 QString으로 바꾸는 유틸리티 함수를 사용하였습니다. 그것을 시그널 함수 인수에 넣어 시그널을 발생시킴과 동시에 해당 QString 정보를 넘길 수 있었습니다.

(3) main_window.hpp/cpp 파일 설명

```
class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  MainWindow(QWidget* parent = nullptr);
  ~MainWindow();
  QNode* qnode;
  friend class publisher; //publish_message에서 같은 윈도우 창을 가리켜서 입력창에 있는 문

private:
  Ui::MainWindow* ui;
  void closeEvent(QCloseEvent* event);
private slots:
  void set_text(QString str); //subscribe_node의 시그널을 받으면 호출되는 slot 메서드.
};

#endif  // hw3_pkg_MAIN_WINDOW_H
```

MainWindow의 헤더파일 부분입니다. publisher 클래스에서 ui 포인터를 사용하게 되면서 publish_message()에서 ui 포인터를 통해 ui가 가리키는 객체의 멤버를 보다 쉽게 참조하기 위해 friend로 설정하였습니다. publisher 메서드 구현을 할 때 마치 MainWindow에서 ui를 사용하는 것처럼 사용할 수 있도록 하기 위함입니다. 그리고 subscribe_node 클래스에서 시그널이 발생되었을 때 보내진 QString 정보를 받아 이를 ui에 그대로 출력해줄 slot메서드를 선언한 것이 위 사진의 private slots 아래의 set_text 슬롯 메서드입니다.

```
MainWindow::MainWindow(QWidget* parent) : QMainWindow(parent), ui(new Ui::MainWindow)
{
  ui->setupUi(this);
  qnode = new QNode();
  qnode->pub_ptr->ui = ui; //publisher객체의 ui변수에 MainWindow의 ui변수를 대입하여 같은 윈도우 창을 가리키도록 함
  connect(ui->publish, &QPushButton::clicked, qnode->pub_ptr.get(), &publisher::publish_message); //버튼
  connect(qnode->sub_ptr.get(), &subscribe_node::signal_func, this, &MainWindow::set_text); //그리고 콜백
  QObject::connect(qnode, SIGNAL(rosShutDown()), this, SLOT(close()));
}
```

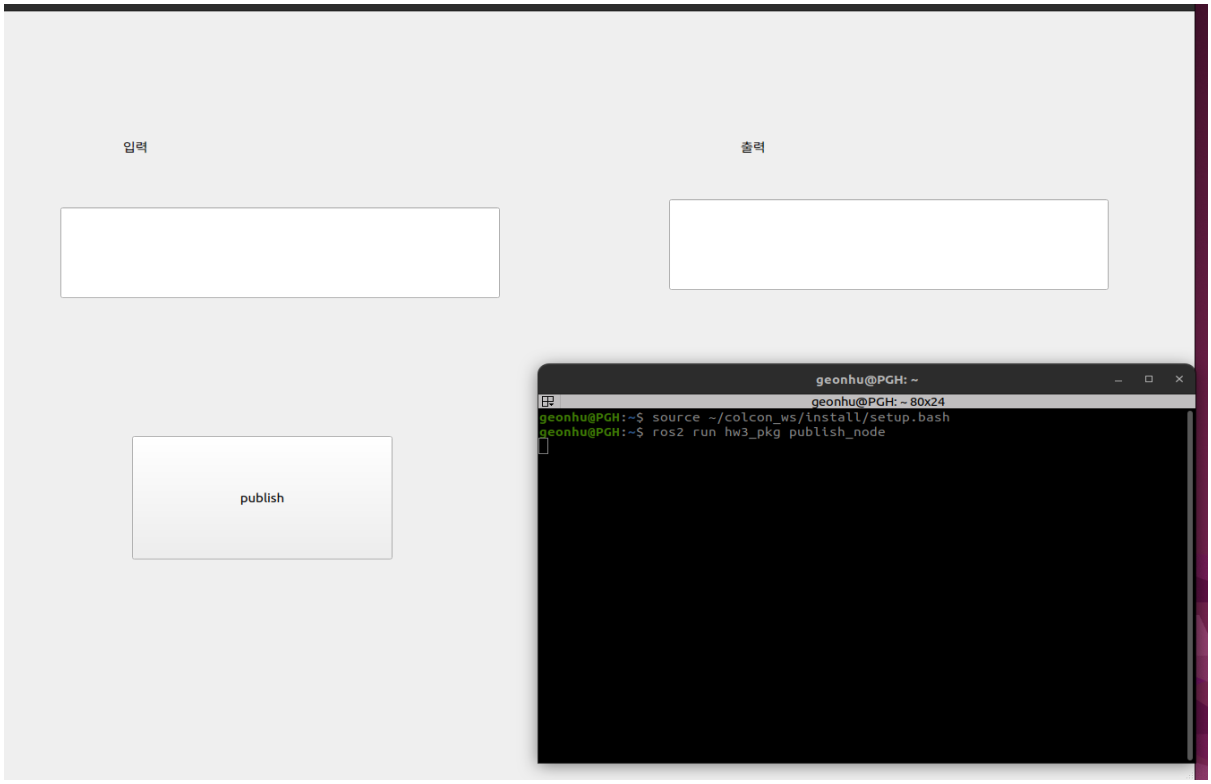
MainWindow의 생성자 구현부입니다. setupUi로 ui의 멤버를 초기화하고 자동으로 connect 되는 것을 자동으로 연결되게 해줍니다. 그리고 QNode 객체를 생성한 하면 QNode 생성자가 호출되고 생성자가 호출되어 make_shared로 각각의 publisher, subscriber_node 객체가 생성되게 됩니다. 그렇기에 QNode 객체를 생성하고 나서는 qnode로 pub_ptr멤버에 접근하여 publisher 객체에 접근하고 그 객체의 ui를 초기화하는 작업을 할 수 있게 됩니다. 그래서 publisher 객체의 ui 포인터를 MainWindow ui로 초기화시켜서 같은 윈도우 창을 가리키도록 설정하였습니다.

그리고 connect를 설정해주었는데, 버튼이 눌리면 버튼이 눌렸다는 사실을 publish_message()에 알려야 publish를 할 수 있게 되는데 그러기 위해 connect 기능을 사용하고자 했었습니다. 그래서 버튼이 눌렸다는 시그널이 발생했을 때 publish객체 쪽의 publish_message를 호출하도록 코드를 짰고, 반대로 subscribe_node에서 시그널을 발생될 때에는 QString 정보를 받아와서 이를 출력창에 출력하도록 한 것입니다.

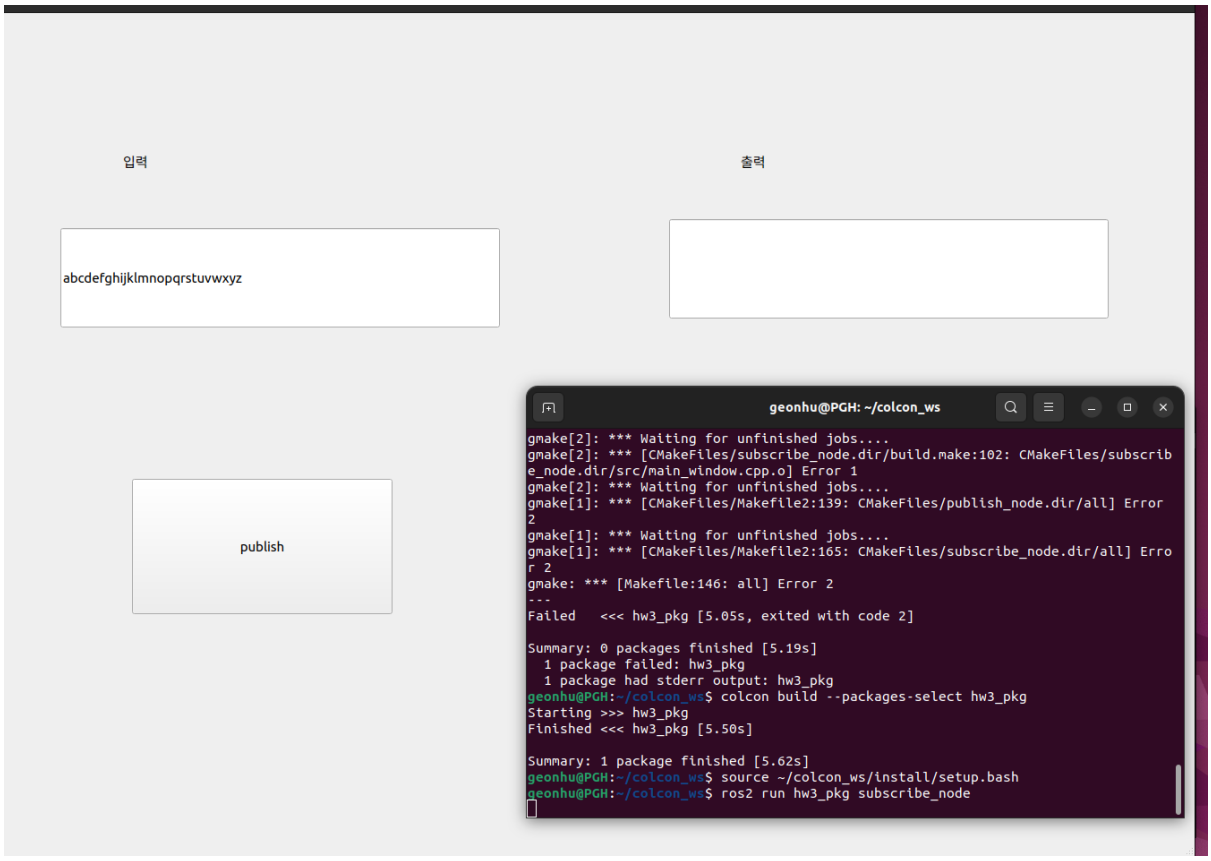
이 때 스마트 포인터를 그대로 넣으면 타입 에러가 나게 되기 때문에 스마트 포인터가 갖고 있는 객체의 실제 포인터를 반환하는 get메서드를 사용하였습니다.

```
}
void MainWindow::set_text(QString str){
  ui->output->setText(str); //문자열 출력하는 부분
}
```

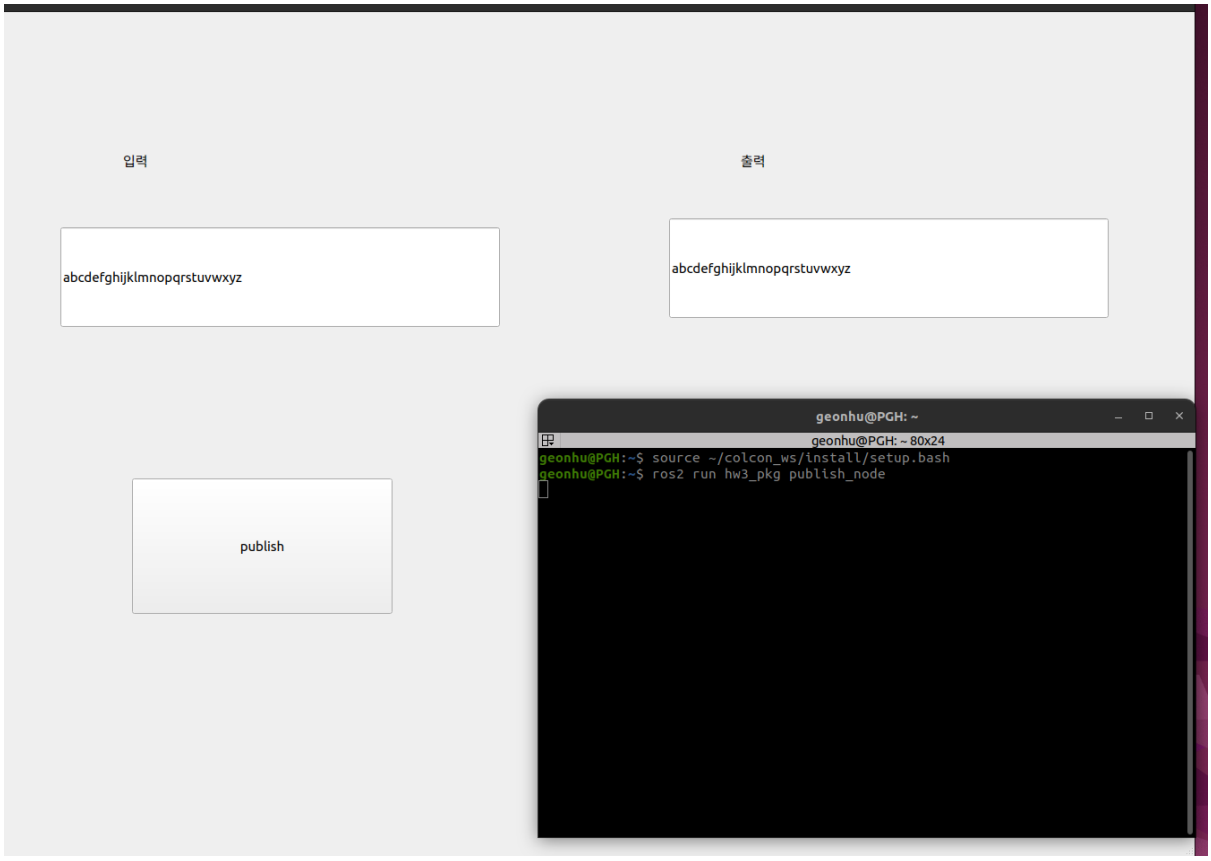
마지막으로 subscribe_node의 시그널로부터 QString 정보를 받아와 출력창에 setText를 통해 출력하는 부분입니다.



현재 각각의 2개의 노드를 실행시킨 상황입니다.



그리고 알파벳을 모두 나열하여 입력하였습니다



publish라는 버튼을 누르면 출력부분에 같은 문자열이 출력됨을 확인할 수 있습니다.