

cpp&qt_day3

hw_1 과제 보고서

목차

1. 과제 1번 코드 설명

-mainwindow.h

-main.cpp

-mainwindow.cpp

1. 과제 1번 코드 설명

```
1  #ifndef MYWIDGET_H
2  #define MYWIDGET_H
3  #include <QWidget>
4  #include "ui_mainwindow.h"
5
6  class MyWidget : public QMainWindow
7  {
8      Q_OBJECT
9  public:
10     explicit MyWidget(QWidget *parent = nullptr);
11
12 protected:
13     void paintEvent(QPaintEvent *event) override;
14
15 public slots:
16     void setAngle1(int angle);
17     void setAngle2(int angle);
18     void setAngle3(int angle);
19     void onTimerTimeout();
20     void onTimerTimeout1();
21     void onTimerTimeout2();
22     void loadFile();
23     void saveFile();
24
25 private slots:
26     void on_pushButton_clicked();
27
28     void on_pushButton_2_clicked();
29
30     void on_pushButton_3_clicked();
31
32     void on_pushButton_4_clicked();
33
34     void on_pushButton_5_clicked();
35
36 private:
37     Ui::MainWindow *ui;
38     int angle1 = 30;
39     int angle2 = 45;
40     int angle3 = 45;
41     int count = 1;
42     int count1 = 1;
43     int count2 = 1;
44     QTimer *timer; // QTimer 멤버 변수
45     QTimer *timer1;
46     QTimer *timer2;
47     int direction1 = 1;
48     int direction2 = 1;
49     int direction3 = 1;
50
51 };
52
53 #endif
54
```

위 사진은 mainwindow.h 코드를 캡처한 사진이다. #ifndef MYWIDGET_H

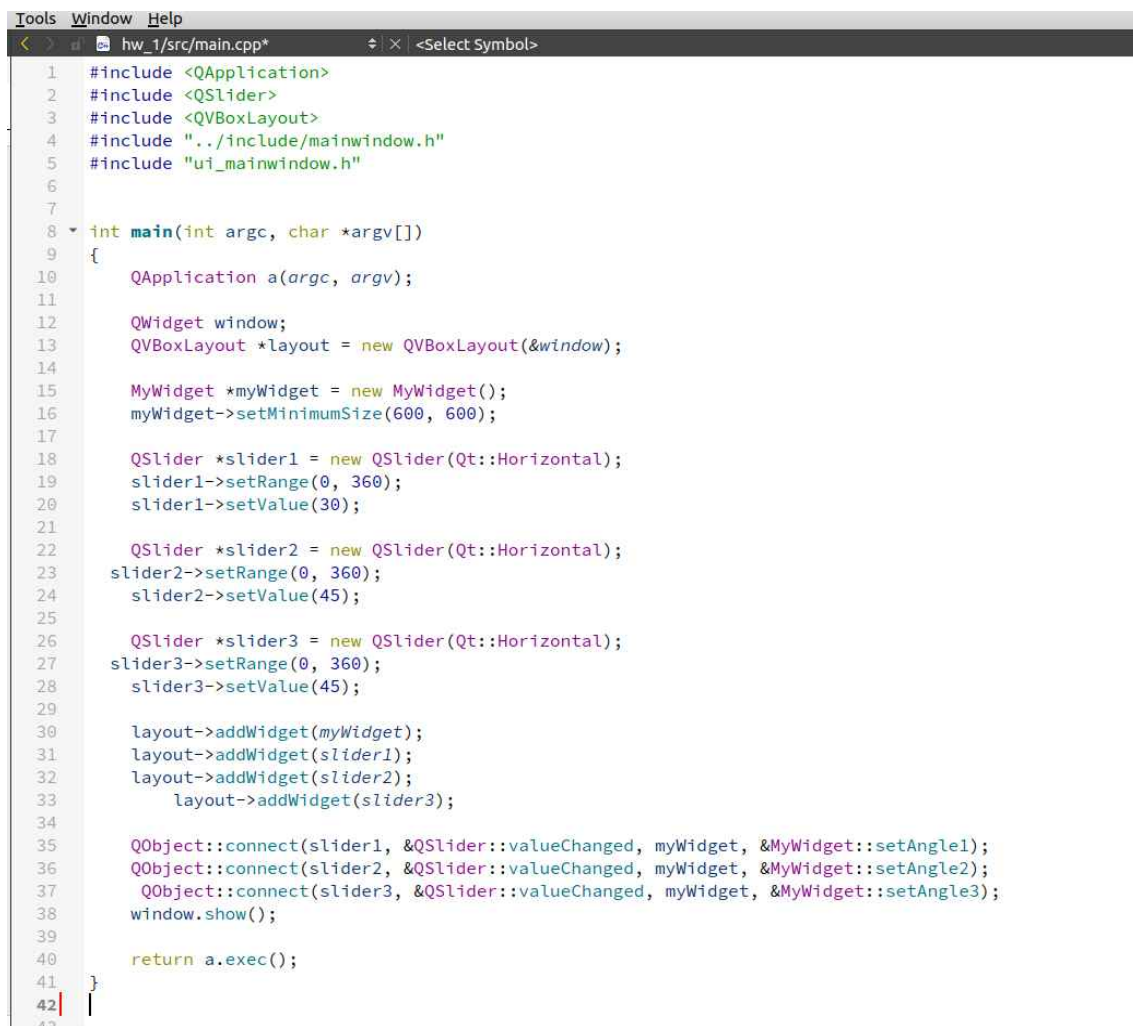
#define MYWIDGET_H 는 헤더 가드라고 불리우며 헤더 파일이 중복 포함이 되는 것을 막기 위해 사용하였다 .

qt 디자이너로 만든 ui파일을 사용하기 위해 각 헤더 파일을 선언해주었다. 클래스 mywidgetd은 public 뒤에 있는 QMainWindow를 상속받아서 메인 윈도우 형태의 위젯을 출력해준다.

Q_object는 메타오브시스템을 사용하기 위한 매크로로써 사용되며 후에 버튼과 상호작용을 위해 사용할 connect ()를 할 수 있게 한다.

생성자를 설정해주고 그리기 이벤트 오버라이드를 호출하여 화면을 update할 시 위젯이 다시 그려지도록 하였다. 그 다음 슬롯에서는 각 로봇팔의 관절 각도를 조절하는 슬롯 함수를 선언하였고 관절이 자동으로 회전시키기 위한 타이머 함수를 호출하였다.

또한 파일 입출력을 위한 `void loadFile(); void saveFile();`을 선언하였다. UI에서 버튼이 클릭되었을 때 상호작용이 있는 슬롯 함수를 선언하였다. `Ui::MainWindow *ui;` 는 `mainwindow.ui`에 정의되어 있는 모든 버튼 또는 슬라이더를 접근 가능하게 하기 위한 포인터이다. 마지막으로 그 아래에는 각도를 설정하는 변수, 버튼 클릭을 카운트하는 변수, 타이머, 움직임 방향을 나타내는 변수를 차례대로 설정해주었다. `#endif`를 하여 헤더 가드를 종료하는 것 또한 잊지 않는다.



```
1  #include <QApplication>
2  #include <QSlider>
3  #include <QVBoxLayout>
4  #include "../include/mainwindow.h"
5  #include "ui_mainwindow.h"
6
7
8  int main(int argc, char *argv[])
9  {
10     QApplication a(argc, argv);
11
12     QWidget window;
13     QVBoxLayout *layout = new QVBoxLayout(&window);
14
15     MyWidget *myWidget = new MyWidget();
16     myWidget->setMinimumSize(600, 600);
17
18     QSlider *slider1 = new QSlider(Qt::Horizontal);
19     slider1->setRange(0, 360);
20     slider1->setValue(30);
21
22     QSlider *slider2 = new QSlider(Qt::Horizontal);
23     slider2->setRange(0, 360);
24     slider2->setValue(45);
25
26     QSlider *slider3 = new QSlider(Qt::Horizontal);
27     slider3->setRange(0, 360);
28     slider3->setValue(45);
29
30     layout->addWidget(myWidget);
31     layout->addWidget(slider1);
32     layout->addWidget(slider2);
33     layout->addWidget(slider3);
34
35     QObject::connect(slider1, &QSlider::valueChanged, myWidget, &MyWidget::setAngle1);
36     QObject::connect(slider2, &QSlider::valueChanged, myWidget, &MyWidget::setAngle2);
37     QObject::connect(slider3, &QSlider::valueChanged, myWidget, &MyWidget::setAngle3);
38     window.show();
39
40     return a.exec();
41 }
42
```

위 사진은 `main.cpp` 파일을 캡처한 사진이다. 위젯을 사용하기 위한 헤더파일을 모두 선언해주었다. `QApplication a(argc, argv);`는 이벤트

루프와 GUI 처리를 담당할 객체이다. 그 후 빈 윈도우를 생성하고 수직 박스 레이아웃을 토대로 다른 위젯을 출력한다. 출력할 위젯의 최소 크기를 600x600으로 지정하였다. 로봇팔의 각 축에서 각도를 조절할 슬라이더를 setRange로 0에서 360의 범위 그리고 초기 각도를 30도로 지정해둔 상태로 생성하였다. 위와 같은 방식으로 슬라이더를 2개 더 생성하여 3축으로 움직이는 로봇팔을 구현해보았다. 이렇게 생성한 위젯들은 layout -> 명령으로 레이아웃에 배치된다. 슬라이더를 움직일 시에 축의 각도가 변경되도록 하기 위해 connect로 슬라이더 값이 변경되면 setangle 함수를 호출하였다. 마지막으로 return a.exec(); 하여 타이머나 선 출력하기 등 이벤트를 처리한다.

```
Tools Window Help
hw_1/src/mainwindow.cpp* MyWidget::loadFile() -> void
55
56 painter.translate(100, 0); // 팔 1끝으로 이동
57 painter.rotate(angle2);
58 painter.drawLine(0, 0, 80, 0); // 팔 2
59
60 painter.translate(80, 0); // 팔 2끝으로 이동
61 painter.rotate(angle3);
62 painter.drawLine(0, 0, 60, 0); // 팔 3
63
64 painter.restore();
65
66
67 void MyWidget::setAngle1(int angle)
68 {
69     angle1 = qBound(0, angle, 360);
70     update();
71 }
72
73 void MyWidget::setAngle2(int angle)
74 {
75     angle2 = qBound(0, angle, 360);
76     update();
77 }
78
79 void MyWidget::setAngle3(int angle)
80 {
81     angle3 = qBound(0, angle, 360);
82     update();
83 }
84
85 // 버튼 클릭 시 호출되는 함수
86 void MyWidget::on_pushButton_clicked()
87 {
88     count++;
89
90     if (count % 4 == 2) {
91         if (!timer->isActive()) {
92             timer->start(); // 타이머 시작
93         }
94     }
95
96     else if (count % 4 == 0) {
97         if (timer->isActive()) {
98             timer->stop(); // 타이머 정지
99         }
100     }
101
102     else if (count > 4 && count % 4 == 2) {
103         if (!timer->isActive()) {
104             timer->start();
105         }
106     }
107
108     update();
109 }
```

위 사진은 mainwindow.cpp 코드를 캡처한 사진이다.

```

Tools Window Help
hw_1/src/mainwindow.cpp* MyWidget::loadFile()-> void
1 #include "../include/mainwindow.h"
2 #include <QPainter>
3 #include <QtMath>
4 #include <QtGlobal>
5 #include "ui_mainwindow.h"
6 #include <QPushButton>
7 #include <QVBoxLayout>
8 #include <QTimer>
9 #include <QFile>
10 #include <QTextStream>
11 #include <iostream>
12
13 // QMainWindow를 상속
14 MyWidget::MyWidget(QWidget *parent)
15 : QMainWindow(parent),
16   ui(new Ui::MainWindow), count(0)
17 {
18     ui->setupUi(this); // UI 초기화
19
20     // 버튼 클릭 시 슬롯 연결
21     connect(ui->pushButton, &QPushButton::clicked, this, &MyWidget::on_pushButton_clicked);
22     connect(ui->pushButton_2, &QPushButton::clicked, this, &MyWidget::on_pushButton_2_clicked);
23     connect(ui->pushButton_3, &QPushButton::clicked, this, &MyWidget::on_pushButton_3_clicked);
24
25     connect(ui->pushButton_4, &QPushButton::clicked, this, &MyWidget::saveFile);
26     connect(ui->pushButton_5, &QPushButton::clicked, this, &MyWidget::loadFile);
27
28     // QTimer 객체의 멤버 변수
29     timer = new QTimer(this);
30     timer->setInterval(40); // 40ms마다
31     connect(timer, &QTimer::timeout, this, &MyWidget::onTimerTimeout);
32     timer->start();
33
34     timer1 = new QTimer(this);
35     timer1->setInterval(40);
36     connect(timer1, &QTimer::timeout, this, &MyWidget::onTimerTimeout1);
37     timer1->start();
38
39     timer2 = new QTimer(this);
40     timer2->setInterval(40);
41     connect(timer2, &QTimer::timeout, this, &MyWidget::onTimerTimeout2);
42     timer2->start();
43
44 }
45
46 void MyWidget::paintEvent(QPaintEvent *)
47 {
48     QPainter painter(this);
49     painter.setRenderHint(QPainter::Antialiasing);
50
51     painter.save();
52     painter.translate(300, 300); // 팔1 시작점
53     painter.rotate(angle1);
54     painter.drawLine(0, 0, 100, 0); // 팔 1
55 }

```

MyWidget::MyWidget() 생성자 그리고 ui->setupUi(this):로 UI및 프로그램의 초기 작업을 해준다. 그 후 UI에서 버튼을 클릭했을 때 각 역할에 맞는 기능을 수행하기 위한 함수를 connect로 연결해준다.

자동으로 축이 돌아가는 기능을 구현하기 위해 총 3개의 타이머 객체를 생성하였다. 40ms마다 onTimerTimeout() 함수가 실행된다.

paintEvent는 로봇팔을 이루는 선을 그려주는 역할을 하는 함수이다 .

그리기 도구인 QPainter 객체를 생성하고 중심 (300,300)에 길이가 100인 선 하나, 이 팔의 끝 지점에서 길이가 80인 선 하나 , 그리고 이 선 끝 지점에서 마지막 길이가 60인 선을 그려주었다. 자동 회전 버튼을 클릭했을 때 로봇팔이 계속 움직이게 하기 위해 on_pushButton_clicked함수에서 버튼이 눌리는 횟수를 count해주었고 디버깅 과정에서 버튼 클릭을 하면 count가 2씩 늘어나는 문제가 있었

```

109 }
110
111 void MyWidget::onTimerTimeout()
112 {
113     if (count % 3 == 2) {
114         angle1 -= 10; // 반대 방향으로 회전
115         direction1 = 0;
116     }
117     else {
118         angle1 += 10; // 정방향으로 회전
119         direction1 = 1;
120     }
121
122     angle1 = qBound(0, angle1, 360);
123
124     update();
125 }
126
127 void MyWidget::on_pushButton_2_clicked()
128 {
129     count1++;
130
131     if (count1 % 4 == 2) {
132         if (timer1->isActive()) {
133             timer1->start();
134         }
135     }
136     else if (count1 % 4 == 0) {
137         if (timer1->isActive()) {
138             timer1->stop(); // 타이머 정지
139         }
140     }
141     else if (count1 % 4 && count1 % 4 == 2) {
142         if (timer1->isActive()) {
143             timer1->start(); // 타이머 시작
144         }
145     }
146     update();
147 }
148
149 void MyWidget::onTimerTimeout1()
150 {
151     // count가 2일 때는 반대로 회전, 아니면 정상 방향으로 회전
152     if (count1 % 3 == 2) {
153         angle2 -= 10; // 반대 방향으로 회전
154         direction1 = 0;
155     }
156     else {
157         setAngle1(angle1);
158         setAngle2(angle2);
159         setAngle3(angle3);
160
161         if (direction1 == 1) {
162             angle1 += 10; // 시계방향
163         }
164         else {
165             angle1 -= 10; // 반시계방향
166         }
167
168         if (direction2 == 1) {
169             angle2 += 10;
170         }
171         else {
172             angle2 -= 10;
173         }
174
175         if (direction3 == 1) {
176             angle3 += 10;
177         }
178         else {
179             angle3 -= 10;
180         }
181     }
182
183     update();
184 }
185
186 void MyWidget::on_pushButton_4_clicked()
187 {
188     saveFile();
189 }
190
191 void MyWidget::on_pushButton_5_clicked()
192 {
193     loadFile();
194 }
195
196
197
198
199
200

```

는데 이에 대한 문제 원인을 찾지 못하여 4로 나눈 나머지로 움직임의 여부를 결정하였다. 이 값이 2이면 타이머가 시작되어 정방향(시계방향)으로 움직인다. 또 다시 버튼을 클릭하여 count가 4의 배수를 만족하

```

Tools Window Help
hw_1/src/mainwindow.cpp*  MyWidget::loadFile() -> void

220 void MyWidget::saveFile()
221 {
222     QFile file("arm_status.txt"); // 파일명 설정
223     if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
224         QTextStream out(&file);
225         out << "Angle1: " << angle1 << "\n";
226         out << "Angle2: " << angle2 << "\n";
227         out << "Angle3: " << angle3 << "\n";
228         out << "Direction1: " << direction1 << "\n"; // 회전 방향
229         out << "Direction2: " << direction2 << "\n";
230         out << "Direction3: " << direction3 << "\n";
231         file.close();
232         std::cout << "State saved to arm_status.txt\n";
233     }
234     else {
235         std::cout << "Error";
236     }
237 }
238
239 //파일에서 불러오는 함수
240 void MyWidget::loadFile()
241 {
242     QFile file("arm_status.txt"); // 파일명
243     if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {
244         QTextStream in(&file);
245         QString line;
246
247         // 파일에서 각도 및 방향 읽기
248         while (!in.atEnd()) {
249             line = in.readLine();
250             if (line.startsWith("Angle1: ")) angle1 = line.mid(8).toInt();
251             else if (line.startsWith("Angle2: ")) angle2 = line.mid(8).toInt();
252             else if (line.startsWith("Angle3: ")) angle3 = line.mid(8).toInt();
253             else if (line.startsWith("Direction1: ")) direction1 = line.mid(12).toInt();
254             else if (line.startsWith("Direction2: ")) direction2 = line.mid(12).toInt();
255             else if (line.startsWith("Direction3: ")) direction3 = line.mid(12).toInt();
256
257         }
258
259         setAngle1(angle1);
260         setAngle2(angle2);
261         setAngle3(angle3);
262
263         if (direction1 == 1) {
264             angle1 += 10; // 시계방향
265         }
266         else {
267             angle1 -= 10; // 반시계방향
268         }
269
270         if (direction2 == 1) {
271             angle2 += 10;
272         }
273         else {
274

```

면 타이머가 종료된다. 역방향 (반시계 방향)으로 회전하는 것을 구현하기 위해 다시 버튼을 클릭하면 angle을 지속해서 더하는 방식이 아니라 빼는 방식을 택하였다.

파일입출력 기능을 활용해 먼저 save 버튼을 누르면 arm_status.txt 열어 3개의 축의 현재의 각도 및 정방향인지 역방향인지 여부를 결정하는 direction 값을 그 안에 저장한다. load 버튼을 누른다면 이와 연결된 loadFile 함수에서 위에서 정보를 저장한 파일을 읽기 모드로 열고 끝에 도달할 때까지 while문을 돌며 direction과 angle 값을 읽는다. 그 후 읽은 값을 UI에도 적용하여 direction 값에 따라 각도가 save 한 정보대로 변경되도록 하였다.

