**임베디드 시스템 LAB 4 실험보고서**



**팀 4**

**201501509 김주용**

**201601647 노윤표**

**201601659 이인호**

**201601663 장현빈**

**목차**

1. 과제

1.2 목표

2. 과제 수행

2.1 파트 분배

2.2 개발 환경

2.3 기본적인 설계 및 구현

**∙** 하드웨어 구성

**∙** 파일 구성 및 구조

2.4 User Interface

**∙** Screen.c

**∙** 색상 정보 초기화 및 출력

2.5 기본적인 파일 설명

∙ Header.h

∙ Function.h

∙ Var.h

∙ CommonFun.c

∙ DrawFunction.c

**∙** main.c

2.6 구체적인 구현

∙ 전체적인 동작

∙ Line 함수

∙ Rectangle 함수

∙ Oval 함수

∙ Free Draw 함수

**∙** Select 함수

**∙** Erase 함수

**∙** Clear 함수

**∙** Fill 함수

**∙** 기타 등등

3. 결론

**1. 과제 선택**



저희 팀은 Touch – screen을 활용한 그림판 구현 과제를 선택하였습니다.

**1.2 목표**

\* Touch-screen 을 입력장치로 하여 LCD 상에 그림을 그리는 프로그램을 구현

\* 버튼을 선택하고, 스크린을 touch하여 그림이 그려짐

\* Line, Rectangle, Oval은 rubber-band 지원

\* Free draw는 입력된 선을 smooth하게 연결 (점의 연결이 끊어지면 안됨)

\* Select는 그려진 object를 이동. Free draw object도 선택 가능

**2. 과제 수행**

**2.1 파트 분배**

**201501509 김주용** : 본인이 한 일 작성

**201601647 노윤표** : 본인이 한 일 작성

**201601659 이인호** : 본인이 한 일 작성

**201601663 장현빈** : UI 구성, 전체적인 동작 및 FreeDraw, Select, Erase, Clear, Fill 함수 구현

**2.2 개발환경**

**1. Github**



팀원들과 서로 코드를 공유하고, 서로 피드백을 해 주기 위해서 Github를 활용하여 과제를 진행함

Github주소 : <https://github.com/soulsystem00/embedded-project>

위의 주소로 들어가면 전체 코드를 다운받을 수 있음

**2. Linux**

Odroid 연결 및 프로그램 실행을 위해 리눅스에 연결하여 과제를 진행

사용된 리눅스 버전 : 16.04.6 32bit 및 18.04.4 64bit

**3. GCC 버전**

gcc-4.9.real (Ubuntu/Linaro 4.9.2-0ubuntu1~14.04) 4.9.2

**4. 오드로이드 커널 버전**

Linux odroid 3.10.107 #2

SMP PREEMPT Tue Aug 20 06:08:00 PDT 2019 armv7l armv7lx

**2.3 기본적인 설계 및 구현**

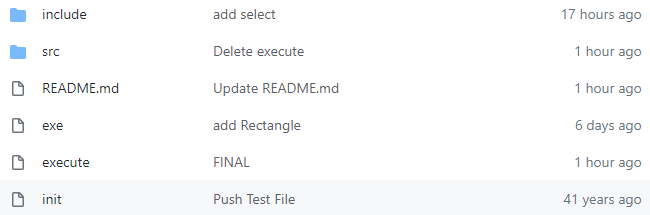
**하드웨어 구성**



추가적인 버튼과 같은 input 없이, 터치스크린만을 사용하기 때문에, Odroid 보드에 직접 TFT LCD를 연결한 형태로 하드웨어를 구성

**파일 구성 및 구조**

- 전체적인 파일 구성



**include** : 코딩에 필요한 헤더파일이 들어있는 디렉토리

**src** : 소스파일이 들어있는 디렉토리

**exe** : 코드의 컴파일 및 프로그램 실행에 관한 명령이 들어있는 파일

**init** : TFT LCD의 초기화에 관한 명령이 들어있는 파일

위의 기본적인 파일 구성을 마친 후 각자 파트를 분배하여 개발을 진행함

**2.4 User Interface**

**Screen.c**

Direction에서 제공된 해당 그림과 유사한 UI를 구현하기 위해 Screen.c 파일에 320 \* 240 크기의 1차원 배열(76,800개의 index)로 구현함





- Screen.c

이러한 형태로 UI를 표현하기 위한 데이터가 저장되어 있으며, 이를 사용하기 위해 소스파일에서 출력을 위한 코드 작성을 진행

**색상 정보 초기화 및 출력**



main.c 에서 screen 배열의 데이터를 사용하기 위한 선언 및 우측 상단의 color를 표현하기 위한 픽셀 정보 초기화

위의 색상 정보는 전역변수로 선언 되어있어 그림을 그릴때도 이용됨

**배열 출력**



헤더 파일에 구현된 Screen 배열을 이용해 터치 스크린에 픽셀을 표현하는 PrintScreen 함수.

**색상 정보 출력**



우측 상단의 color 값을 표현하기 위해 직접 좌표를 이용해 픽셀 값을 수정하는 FillinitColor 함수

(8개 색상에 대한 좌표에 대해 모두 구현이 되어 있음, 코드는 동일하기 때문에 생략)



UI 디자인 구현을 완료 한 모습

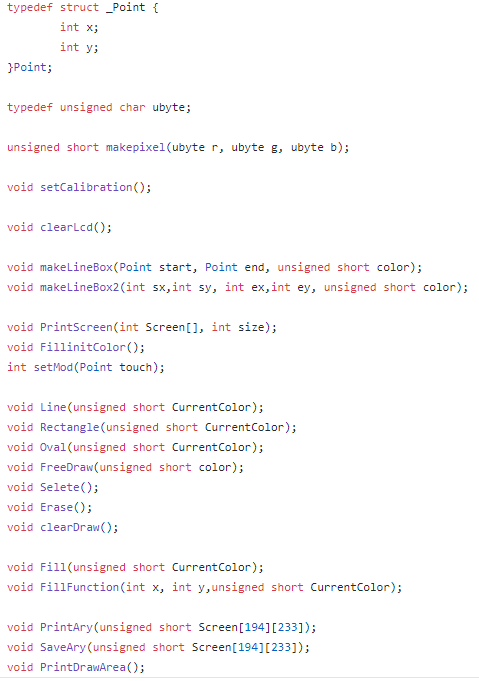
**2.5 기본적인 파일 설명**

필요한 모든 변수들과 함수들을 따로 정의하여 파일을 만듬.

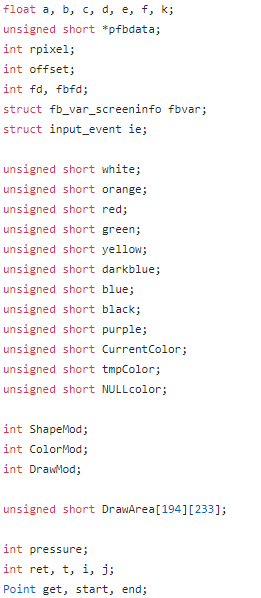
**Header.h** - 프로젝트에서 사용할 헤더들



**Function.h** – 프로젝트에서 사용할 함수들을 선언한 파일



**Var.h** – 프로젝트에서 사용할 전역변수들을 사용한 파일



**CommonFun.c** – 그리기 위해 사용되는 7개의 함수(Line, Rectangle, Oval, FreeDraw, Select, Erase, Clear)이외의 것들을 구현해 놓은 파일

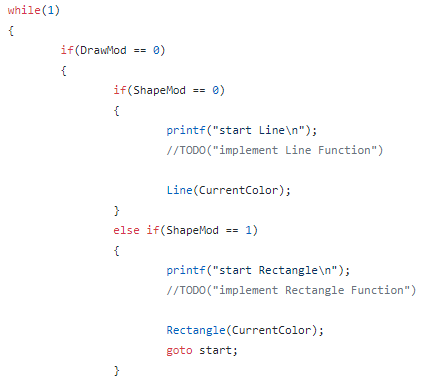
ex)



CommonFun.c에서 구현된 makepixel 함수와 setCalibration 함수.

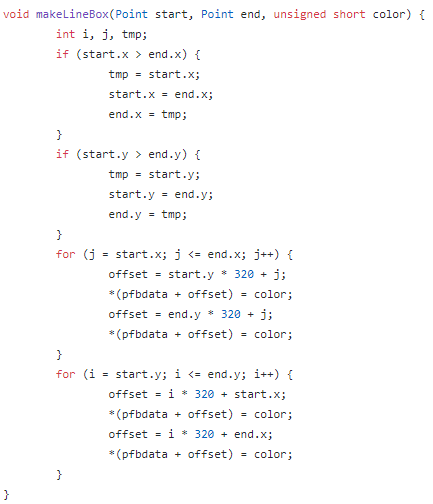


CommonFun.c에서 구현된 clearLcd 함수

**main.c –** 프로그램의 전체적인 동작에 관한 코드를 작성해 놓은 파일 - main.c에서 구현된 동작의 일부분

**DrawFunction.c -** 그리기 위해 사용되는 7개 함수들(Line, Rectangle, Oval, FreeDraw, Select, Erase, Clear)을 구현해 놓은 파일

ex)



DrawFunction.c에서 구현된 makeLineBox 함수

**2.6 구체적인 구현**

**전체적인 동작**

전체적인 동작 UI의 동작과 그리기 알고리즘에 관한 항목이다.

전체적인 동작 UI의 동작은 아래의 알고리즘을 바탕으로 동작한다.

1. 그리기 모드 선택

2. 그리기 구역 터치 시 그리기 모드로 진입

3. 그리기

4. 그리기 구역 밖에 터치 시 그리기 모드를 빠져나감

위의 알고리즘을 반복하면서 그림판 프로그램은 동작한다.

그러면 1번부터 살펴보자

**1. 그리기 모드 선택**

그리기 모드 선택은 setMod함수에 의해 일어나고 화면의 각 버튼들을 터치하면서 선택할 수 있다.

아래는 setMod함수의 일부분이다.

**if ((touch.x >= 4 && touch.x <= 75) && (touch.y >= 34 && touch.y <= 55))**

**{**

**printf("sets mod Line\n"); ShapeMod = 0; DrawMod = 0;**

**}**

위의 코드를 보면 touch 좌표의 값에 따라 ShapeMod와 DrawMod 변수를 설정하는 것을 알 수있다.

즉, 화면의 표시된 버튼을 터치함에 따라 각 변수들이 변하면서 모드를 설정하게 된다.

**if (ShapeMod == 0)**

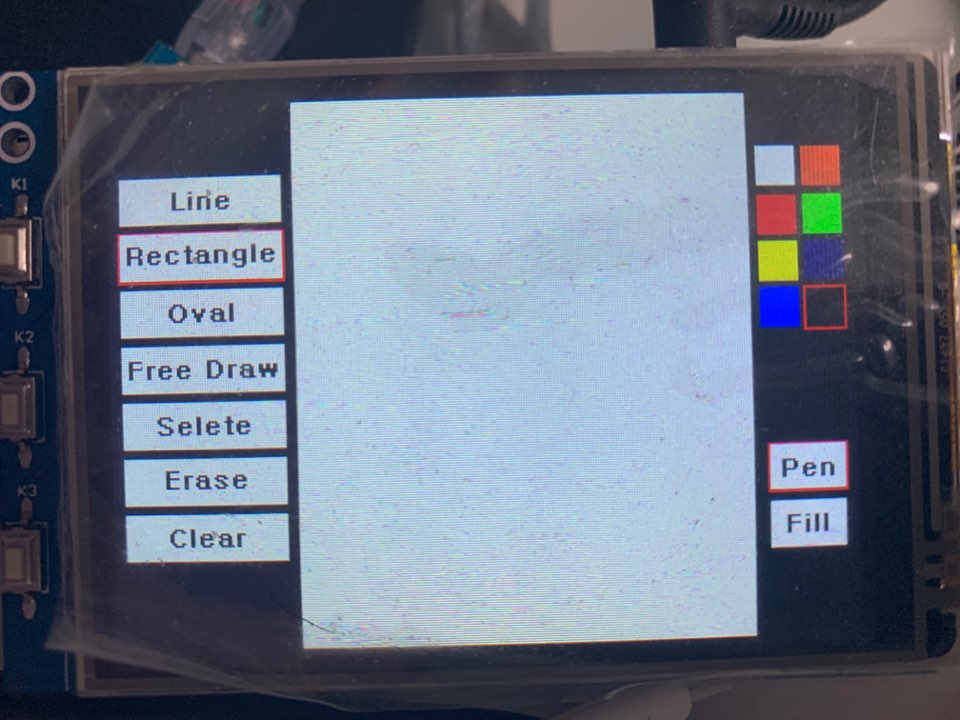
**{**

**ShapeStart.x = 4; ShapeStart.y = 34; ShapeEnd.x = 75; ShapeEnd.y = 55;**

**}**

위의 코드는 자신이 선택한 모드를 화면에 표시해주기 위해 각 버튼의 좌표 값을 저장하는 코드이다.

이렇게 저장한 좌표들을 바탕으로 makeLineBox를 이용해 화면에 자신이 선택한 모드를 표시해주게 된다.



위의 화면을 보면 Rectangle, Pen, 검정색상이 선택된 것을 확인할 수 있다.

즉, 그리기 모드뿐만 아니라 색상선택 펜 선택까지 setMod에서 설정이 된다.

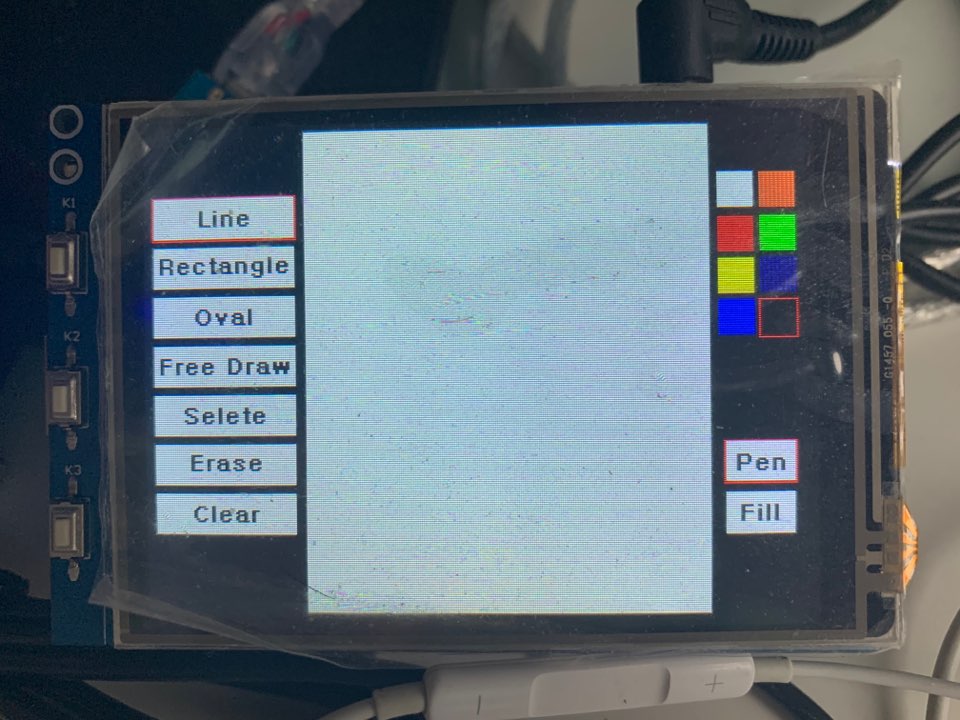
**2. 그리기 구역 터치 시 그리기 모드로 진입**

그리기 모드를 설정했다면 이제 그림을 그릴 차례이다.

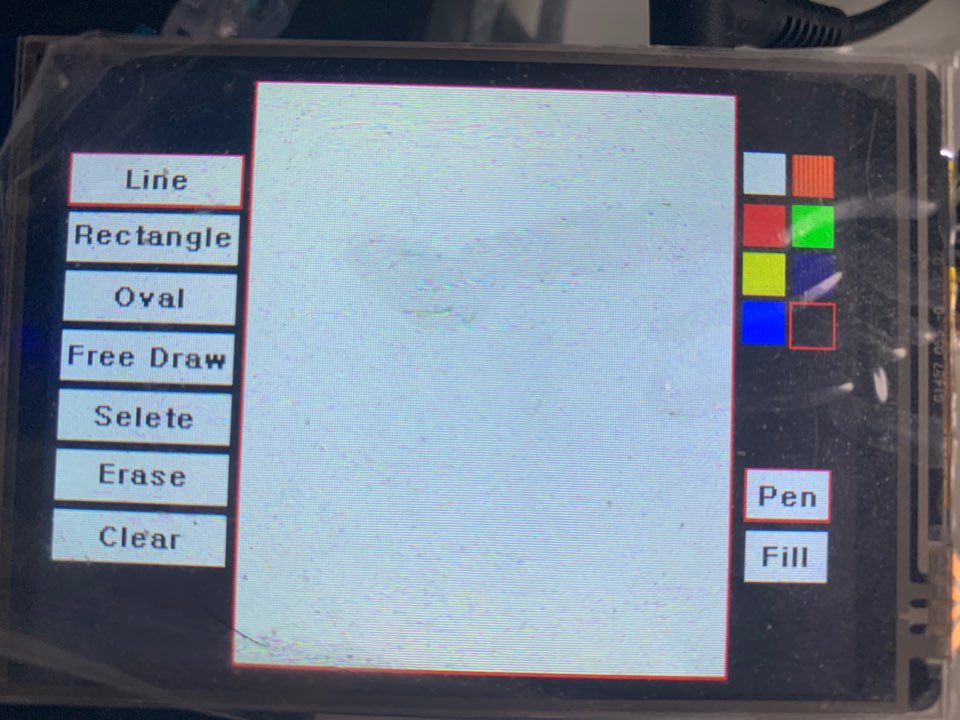
그림을 그리기 위해선 그리기 구역을 터치해 그리기 모드로 진입해야 한다.

그리기 모드로 진입했을시에는 위의 버튼과 마찬가지로 그리기 구역 주변에 빨간색 테두리가 생기게 된다.

그리기 모드 진입 전 ▼



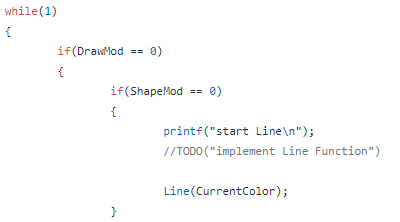
그리기 모드 진입 후 ▼



**3. 그리기**

그리기 모드에 진입했다면 이제 그림을 그릴 차례이다.

그림은 위에서 설정한 그리기 모드에 따라 그림이 그려지게 된다.



위의 코드는 위 동작 중 일부를 캡쳐한 것이다.

위의 코드를 보면 DrawMod가 0이고 ShapeMod가 0이면 Line 함수를 부르도록 되어있다

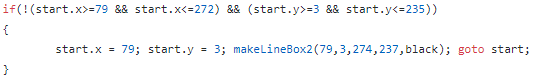
여기서 DrawMod가 0이라는 것은 펜모드에서 Pen이 선택되었다는 것이고

ShapeMod가 0이라는 것은 Line을 그리겠다는 것을 의미한다.

그리고 위에서 선택한 색상은 Line함수의 매개변수로 들어가 해당 색상을 그릴 수 있게 해준다.

**4. 그리기 구역 밖에 터치 시 그리기 모드를 빠져나감**

위의 반복문 안에는 아래의 코드를 넣어놓았다.



그리기 구역 밖을 터치하면 코드의 시작지점으로 돌아간다는 뜻이다.

즉 그리기 구역 밖을 터치하면 그리기 모드가 종료되고 1번부터 다시 코드가 반복이 된다는 것이다.

위의 1~4번을 반복하면서 그림판 프로그램은 동작을 하고 그에 맞게 함수를 불러와 올바른 동작을 하게끔 만들어준다.

그림을 그리는 알고리즘은 아래와 같다.

1. 현재 그린 그림을 tmp배열에 저장

2. Draw배열 출력

3. tmp배열 출력

4. 그리기 종료시 tmp배열의 내용을 Draw배열에 추가

Touch Screen으로의 출력은 mmap을 사용하여 바로 출력할 수 있었다.

하지만 다른 그림을 그리면 기존 그림이 지워지는 문제와 Select를 이용한 그림 옮기기를 위해서

배열을 사용하여 출력을 하도록 바꾸었다. 기존 mmap보다는 속도와 반응성이 떨어지지만 여러 문제점들을 손쉽게 해결할 수 있었다.

위의 알고리즘을 이해하기 쉽게 그림으로 표현해 보았다.

🡺 현재 그리고 있는 화면(tmp)

🡺 기존 그렸던 그림이 저장되어 있는 화면(Draw)

🡺 UI 구성 화면

이런 식으로 레이어를 나누어 표현을 하면 새로 그림을 그려도 새로운 그림이 기존 저장되어 있는 화면을 간섭하지 않고 그림을 그릴 수 있게 된다.

또한 rubber-band를 구현함에 있어서도 전체 화면을 간섭할 필요 없이 현재 그리고 있는 화면에 대한 컨트롤만 해주면 되기 때문에 상당히 편리 해졌다고 생각한다.

그리기가 종료되면 tmp에 있는 내용들을 Draw에 옮겨 줌으로써 변경내용을 반영하도록 해주었다.

**Line 함수**

\* 기능 구현에 사용 된 알고리즘

저희의 LCD는 320 \* 240의 좌표로 이루어져 있습니다. 여기서, 직선을 긋는다면 시작점과 끝 점 사이에서 기울기를 구해서 그 기울기에 맞는 좌표들을 모두 찍어야 합니다. 일반적으로 기울기를 구하는 공식은,

(시작점 = (x1, y1) 끝점 = (x2, y2)



이러한 직선의 방정식을 떠올릴 수 있습니다. 하지만, 실수 연산이 모두 정수형으로 형 변환이 되기 때문에 이러한 식으로 직선을 구현한다면, 기울기가 정수인 경우에만 그려질 것입니다. 이를 해결하기 위해

‘ 브레슨 햄 알고리즘 ‘ 을 착안하였습니다.

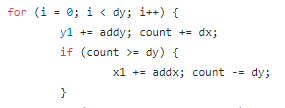


저희는 LCD에 시작점과 끝 점 사이의 최소의 거리들을 모두 점으로 표현하고 싶은 것이 목적입니다. 즉 시각적으로 직선에 가장 가깝게 보이도록 표현해야 합니다. 이를 표현하기 위해서 흔한 표현으로, 위의 사진처럼 직선 렌더링 기법을 사용하기로 결정했습니다.

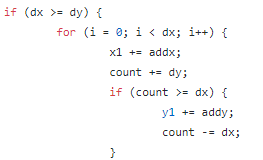
이를 표현하기 위해서는, 기울기가 1보다 작은 경우와, 기울기가 1보다 큰 두 가지 경우로 나뉩니다.



기울기가 1보다 클 때는, x좌표가 증가하는 기분 보단, y좌표가 증가하는 기준으로 점이 찍히게 됩니다. 이를 코드에 적용하면,

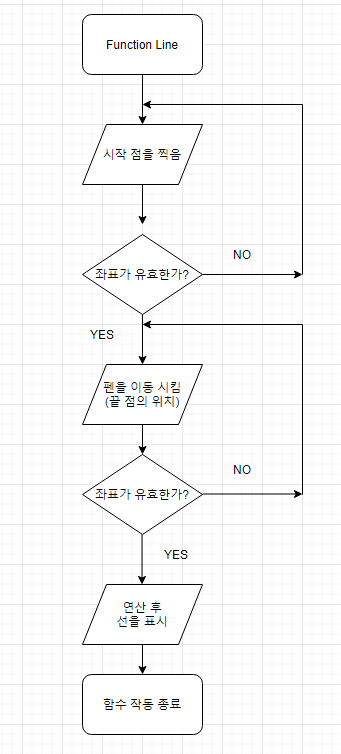


이러한 연산을 하게 됩니다, (y1 = 시작점의 y좌표, dx, dy =증가량, count = 증가량에 따라 x와 y 중 어떤 것을 더할 지 결정하는 변수,) + addx, addy 의 값은 1이며, 시작점 좌표가 끝 점 좌표보다 좌표 값이 낮은 위치에 있을 경우에 부호가 역전됩니다. Y가 증가함에 따라, count 변수에 dx를 계속 더합니다. 만약 count의 누적된 값이 y 증가량을 넘어선다면, x를 증가시키고 count에 dy만큼 뺍니다. 기울기가 1보다 작은 경우는,



위와 같은 연산을 합니다. 이러한 연산을 이용해, UI에 Line 함수가 정상적으로 작동하도록, 함수의 틀은 아래와 같이 구성하였습니다.

\* 함수 구조



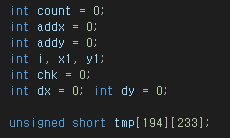
해당 flow chart를 기반으로, 그리는 공간에 유효한 좌표에 시작 점을 찍을 때 까지 반복을 하는 while문과,

그리는 공간에 유효한 좌표에 끝 점을 찍을 때 까지 반복을 하는 while문으로, 총 두개의 반복문을 가진 구조를 구현하기로 하였습니다.

\* 함수 코드 설계



우선 현재 색을 매개변수로 받아서 작동을 하는 Line 함수의 선언부 입니다.(가독성을 위해 VS로 캡쳐하였습니다.)



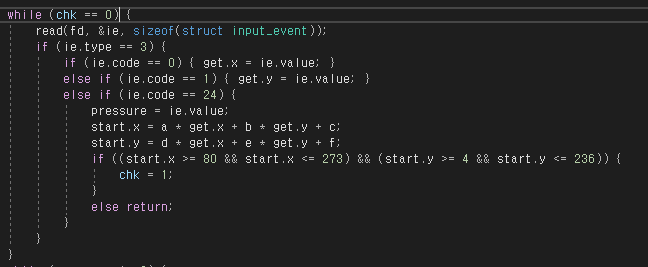
함수 작동을 위해 사용되는 지역변수들입니다.

Count : 앞서 설명 드린 증가량에 따라 x와 y 중 어느 것을 증가 시킬 지 판별하기 위한 조건변수

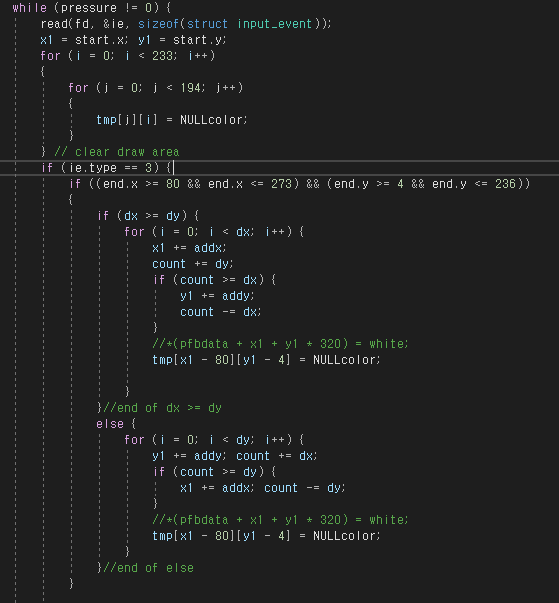
Addx, addy : count에 맞추어 x나 y를 증가시킬 때 더하는 변수.

Chk : 시작 점이 찍혔는지, 안 찍혔는지 판별하기 위한 조건변수

Tmp 배열 : 그리는 공간의 좌표와 해당 배열 값들을 매핑 시키기 위해 선언합니다.

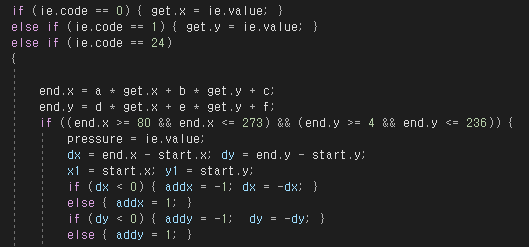


함수의 시작부분에서, chk 값을 조건으로 걸어서 첫 시작 좌표를 찍을 때 까지 반복합니다. 또, 그 좌표가 DrawArea (80,4) ~ (273,236) 에 들어갈 경우에만, chk 값을 바꿔서 해당 반복문을 빠져나가게 합니다.

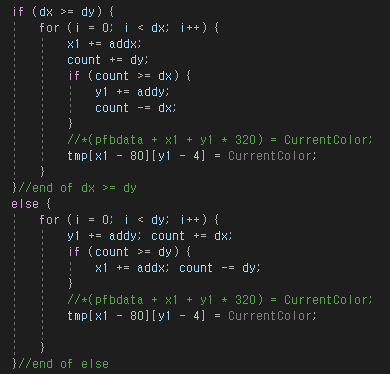


시작 점이 조건에 맞게 찍혔다면, 그 다음에는 터치 펜을 스크린에서 뗄 때 까지(pressure가 0이 될 때 까지) 반복을 하는 반복문에 들어가게 됩니다. 첫 점을 찍은 이후, 스크린에 pressure가 계속 들어오고, 좌표가 계속 읽히기 때문에 과제 direction의 rubber-band를 이에 맞춰 구현 할 수 있다고 생각하였습니다.

끝 점 처리의 시작부분에서는, 먼저 스크린의 draw area와 매핑되는 tmp 배열의 원소들을 초기화합니다. Rubber – band 방식을 하게 되면, 펜을 움직일 때 마다 그려져 있던 선을 지우고, 다시 그리는 작업을 반복해야 하기 때문입니다.



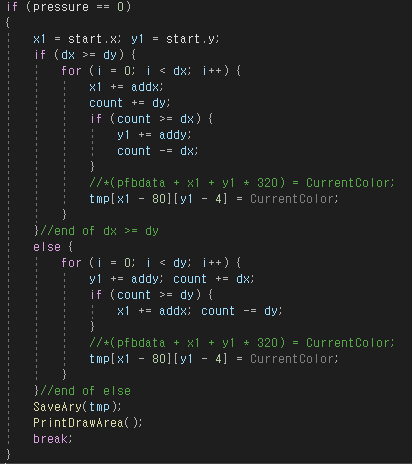
지우는 과정을 완료한 이후, 펜에 입력되는 정보들을 가져와서 변수에 대입합니다. 이 때, 예외 처리가 들어가게 되는데, 만약 끝 점의 좌표 값이 시작 점의 좌표보다 값이 작다면, 증가량은 음수를 띄게 될 겁니다. 그렇다면, 이에 따라 계산을 다르게 해 주어야 하는데, 저는 계산을 다르게 하지 않고, 증가량과 addx, addy 자체의 부호를 역전시켜 동일한 계산을 해도 문제가 없도록 처리하였습니다.



변수에 값들을 선언 하고, 부호에 대한 예외 처리 이후, 선을 그리기 위해 계산을 합니다. CurrentColor 라는 색상 정보를 매개변수로 받았기 때문에, 해당하는 x좌표와 y좌표를 tmp 배열의 인덱스에 매핑시켜서, 해당 좌표는 CurrentColor 라는 정보를 넣게 됩니다. 그 이후,



해당 함수로 사용자가 보는 Draw area에 시각적으로 표현을 해줍니다. 그리고 이 과정은 펜을 뗄 때 까지(Pressure 값이 0이 될 때 까지) 지우고 그리기를 반복합니다. -> rubber band 구현



펜을 뗐다면, 마지막으로 읽은 좌표가 남아있으니 그 좌표를 이용해 다시 계산을 해서, tmp 배열에 저장 한 다음, 출력을 하게 됩니다. 이를 수행 하고 Line 함수는 종료됩니다.

**Rectangle 함수**

**Oval**

**Free Draw**

Free Draw의 기본적인 알고리즘은 아래와 같다.

1. 터치가 이루어짐

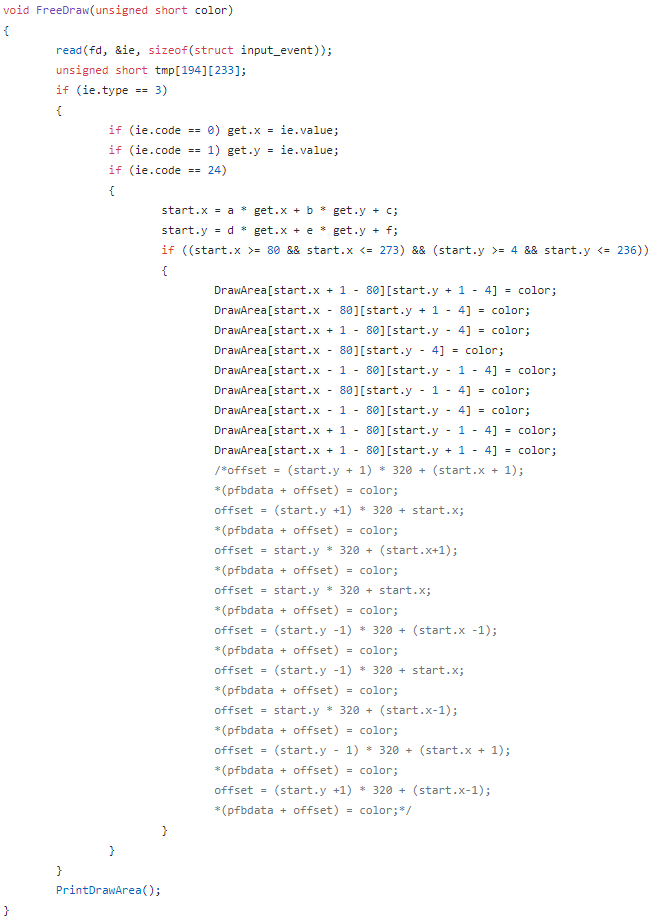
2. 해당 좌표를 기준으로 3\*3만큼의 크기만큼 배열에 저장

3. 배열의 출력

Free Draw의 구성은 간단하다.

현재 터치가 되고 있는 좌표를 읽은 후 출력을 해주면 된다.

코드는 아래와 같다.



기존 출력 알고리즘은 tmp에 저장을 해야 했지만 FreeDraw는 출력내용이 바로 화면에 반영이 되어도 상관없기 때문에 tmp에 저장하는 것이 아닌 DrawArea에 곧바로 저장하도록 만들어 주었다.

**Select**

Select함수의 알고리즘은 아래와 같다.

1. 시작좌표 입력받음

2. 종료좌표 입력받음

3. 시작좌표와 종료좌표를 바탕으로 사각형을 그림

4. 시작좌표 입력받음

5. 종료좌표 입력받음

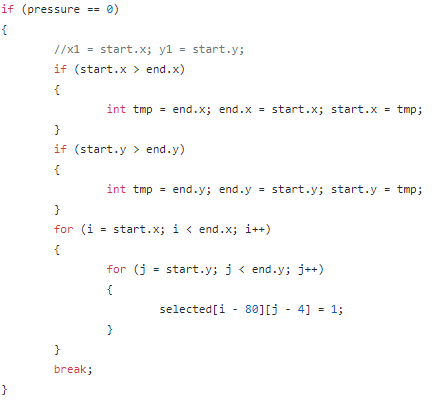
6. 시작좌표와 종료좌표간 차이만큼 사각형 안의 그림을 옮김

시작좌표와 종료좌표를 입력받는 것은 다른 코드와 동일하다. 그렇기 때문에 3번과 6번을 위주로 설명하도록 하겠다.

3. 시작좌표와 종료좌표를 바탕으로 사각형을 그림

시작좌표와 종료좌표를 바탕으로 사각형을 그리는 것은 Ractangle함수와 동일하다.

하지만 Select에서는 하나의 과정이 더 추가된다.



바로 위의 코드가 Rectangle의 함수와 다른 부분이다.

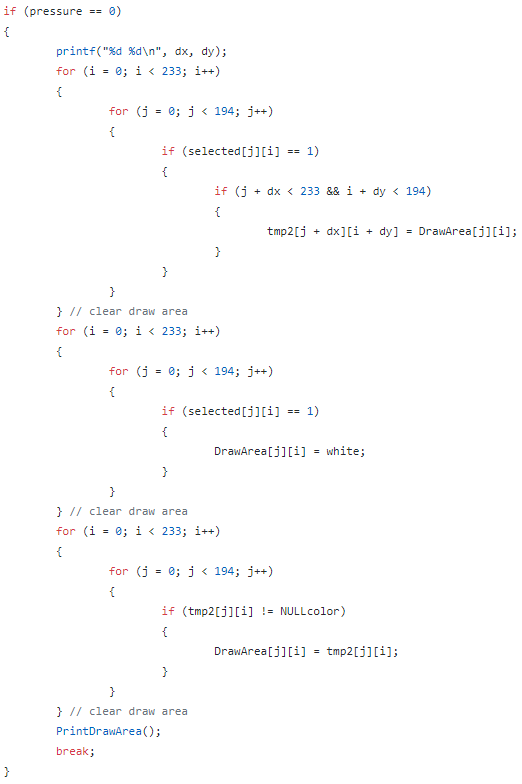
select 함수에는 selected배열이 있다. 이 배열의 역할은 선택된 영역에 대해 flag를 설정해주는 역할을 한다.

즉 선택된 영역 안의 플래그를 모두 1로 설정을 해준다는 이야기이다.

이런 식으로 설정해 놓은 것들은 뒤이어 이동을 시키는 부분에서 사용이 된다.

다시 시작좌표와 종료좌표를 입력 받고 선택된 영역을 이동을 시킨다.

이동시키는 부분의 코드이다.



위의 코드의 주된 내용은 flag로 설정해 두었던 selected배열을 바탕으로 1로 설정되어 있는 부분에 대해 DrawArea의 값을 tmp2에 저장을 한다.

이때 tmp2에 저장하는 인덱스는 앞서 입력받았던 시작좌표와 종료좌표의 차이만큼 변동시킨다.

그렇게 되면 tmp2에는 값들이 이동된 것처럼 저장이 된다.

그리고 나서 Selected배열이 1로 되어있는 곳의 DrawArea부분을 하얀색으로 바꿔주고

DrawArea에 tmp2의 값을 저장해주면 DrawArea에는 선택된 부분이 이동되어 저장이 되고 함수는 종료가 된다.

**Erase**

**Clear**

**Fill**

**기타 등등**

**결론**