

---

# 프로젝트 계획 및 설계 보고서

---

## C-source 편집기

작성일	2022.11.03	제출일	2022.11.06
과 목	어드벤처디자인 3분반	전 공	컴퓨터공학
담당교수	김경수	조 명	1조
조 장	송제용	조 원	김현진
조 원	안현진	조 원	정지수

# 목

# 차

<b>1. 문제 이해</b>	5
1-1. 문제 정의	5
1-2. 프로젝트 목표	5
<b>2. 개발 과정 고려사항</b>	6
2-1. 개발 방법론	6
2-2. 구현 언어 및 고려사항	6
<b>3. 사용자 요구사항 분석 및 설계</b>	7
3-1. 기능적 요구사항	7
3-2. 비기능적 요구사항	7
3-3. Usecase 다이어그램	8
3-4. 시퀀스 다이어그램	9

<b>4. 시스템 설계</b>	11
4-1. 프로그램 전체 기능 및 목표 명세	11
4-2. 프로그램을 구성하는 모듈	12
4-3. MVC 패턴의 이해 및 클래스 다이어그램	13
4-4. 정리	16
<b>5. 프로그램 모듈 설계</b>	17
5-1. 모듈 사용 목적 기능 및 제약 조건 명세	17
5-2. 모듈 입출력 및 실행예시 명세	18
5-3. 모듈의 알고리즘(suedo code) 및 주요 변수 기술	21
5-4. 정리	33
<b>7. 인터페이스 설계</b>	34
7-1. 프로그램 UI	34
7-2. 인터페이스의 전체적인 구조 도식화 및 설명	36
7-3. 인터페이스의 입출력 구조	37
7-4. 사용자 인터페이스와 직접적으로 통신하는 내부	38

8. 논의	39
8-1. 추가적인 아이디어가 필요한 기능	39
8-2. 테스트 및 유지보수 계획	40
9. 추진 일정 및 역할분담	42

# 1. 문제 이해

## 1-1. 문제 정의

객체 지향 언어를 이용해 C 소스 편집기를 제작한다. C 소스 편집기는 일반 텍스트 편집기에서 필요한 기능이 더해 C언어의 문법 요소를 검사하는 기능을 가진다. 이때 검사하는 문법적 요소는 괄호 짝 맞추기, 주석 처리 정도가 있으며 추가적인 기능으로 키워드 강조가 있을 것이다. 자세한 요구사항 분석은 3번 목차의 요구사항 분석 및 설계에서 자세히 할 예정이다.

우리가 만들 프로그램에 대한 예시는 대표적으로 “Gedit” 라는 리눅스 GUI 환경에서 사용하는 텍스트 에디터가 있으며 이번 프로젝트에서 만들 프로그램과 매우 흡사하다.

## 1-2. 프로젝트 목표

프로젝트 목표로써는 다음과 같다.

1. 객체 지향 언어를 사용하여 C 소스 편집기를 제작한다.
2. 텍스트 편집기에 더해 C언어의 문법 요소를 체크할 수 있게 한다.
3. 프로그램은 기능적으로 컴파일러 요소 중 프론트엔드에 해당하는 부분, 즉 어휘 분석과 Scanning단계를 수행한다.
4. 위에 말한 일련의 과정을 GUI로 구현한다.

여기서 3번 항목에 대한 추가적인 설명을 덧붙이자면 컴파일러의 구성요소는 프론트엔드, 최적화기, 백엔드 세 부분으로 구성되어 있다. 이 3가지 부분 중에 우리가 구현하는 부분은 컴파일러의 프론트엔드에 해당하는 부분이며 언급한 대로 어휘 분석과 Scanning단계를 구현하는 것과도 같다. 소스코드의 내부를 모두 토큰 단위로 분리하여 프로그램의 문법 규칙에 맞는지 확인하는 동작을 수행하며 프로그램의 의미를 변경시키거나 하지는 않는다.

흔히 우리가 리눅스에서 Gedit과 Gcc컴파일러를 같이 썼던 경험을 해봤을 텐데 그 이유 역시 이 이유에서 나오는데 Gedit은 컴파일러의 프론트엔드 부분만 제공하기 때문에 바이트들로 이루어진 실행 파일을 만들기 위해서는 Gcc 컴파일러를 사용하여야 하는 것이다.

프로젝트 진행 전, 개발 영역에 대해 정확히 짚고 넘어갈 필요가 있다고 생각하여 이 부분에 대해 구체적으로 자료를 조사하였다.

## 2. 개발 과정 고려사항

### 2-1. 개발 방법론

우리가 이번 프로젝트를 위해 채택한 개발 방법론은 통합 프로세스 모델인 **UP(Unified Process)**이다. 소프트웨어를 반복/전진적으로 개발하는 대표적인 개발 방법론이며 도입, 정련, 구현, 전이 4단계가 반복적으로 이루어지며 개발을 한다.

UP를 채택한 이유는 객체 지향 분석 및 설계(OOA/D)기반의 프로젝트에서 가장 많이 사용되는 개발 방법론이라고 할 수 있다. 문제 정의에서 언급했듯이 객체 지향 언어를 사용하여 개발하는 것이 프로젝트 목표 중 하나이기 때문에 이번 프로젝트에 제일 적합하다고 생각했고 실제로 기본 기능을 구현한 후에 추가 기능을 덧붙여 나가며 프로젝트를 완성 시켜나갈 예정이다.

### 2-2. 구현 언어 및 고려사항

구현 언어는 **자바**를 선택하였으며 GUI 라이브러리로 자바의 **Swing**을 선택하여 기본적인 메모장 인터페이스를 구현할 예정이다. 통합 개발환경은 대표적인 자바 IDE인 **IntelliJ**를 선택하였다. 추가적으로 이번과는 다르게 체계적인 협업을 위해서 **Git**을 사용하여 버전 관리를 하고 **Notion**을 통해 일정 관리 및 프로젝트 산출물을 정리 및 공유할 예정이다. 회의는 **Discord**로 프로젝트 간 지속적으로 진행할 계획이다.



# 3. 사용자 요구사항 분석 및 설계

## 3-1. 기능적 요구사항

기능적 요구사항은 사용자의 입장에서 필요한 기능들을 의미하며 기능적 요구사항들은 하나의 Use-case가 된다.

우리는 문제 정의에 따라 필수요소와 추가요소로 기능적 요구사항을 분리하여 생각해보았다. 이에 따라 정리한 요구사항은 다음과 같다.

### - 필수요소

- 1) 사용자는 키보드를 통해 텍스트를 입력할 수 있어야 함.
- 2) 사용자는 새 파일을 생성하거나 파일 열기, 닫기, 저장, 종료를 수행할 수 있어야 한다.
- 3) 입력 커서의 위치를 방향키로 이동할 수 있어야 한다.
- 4) 입력된 텍스트를 지울 수 있어야 한다.
- 5) 입력된 괄호 쌍은 같은 색으로 표시되어야 한다.
- 6) 주석은 다른 색깔로 화면에 표시되어야 한다.

### - 추가요소

- 1) 마우스 또는 키보드를 통해 영역을 선택 후 이동, 삭제, 복사를 수행할 수 있어야 한다.
- 2) 앞으로 가기, 뒤로 가기를 수행할 수 있어야 한다.
- 3) 단어를 검색할 수 있어야 한다.
- 4) 구현된 동작들에 대해 단축키가 지정되어야 한다.
- 5) 자동 저장이 되어야 한다.
- 6) 괄호와 주석 처리 이외에 키워드, 식별자, 상수, 기호 등을 구분하여 색을 다르게 표시한다.

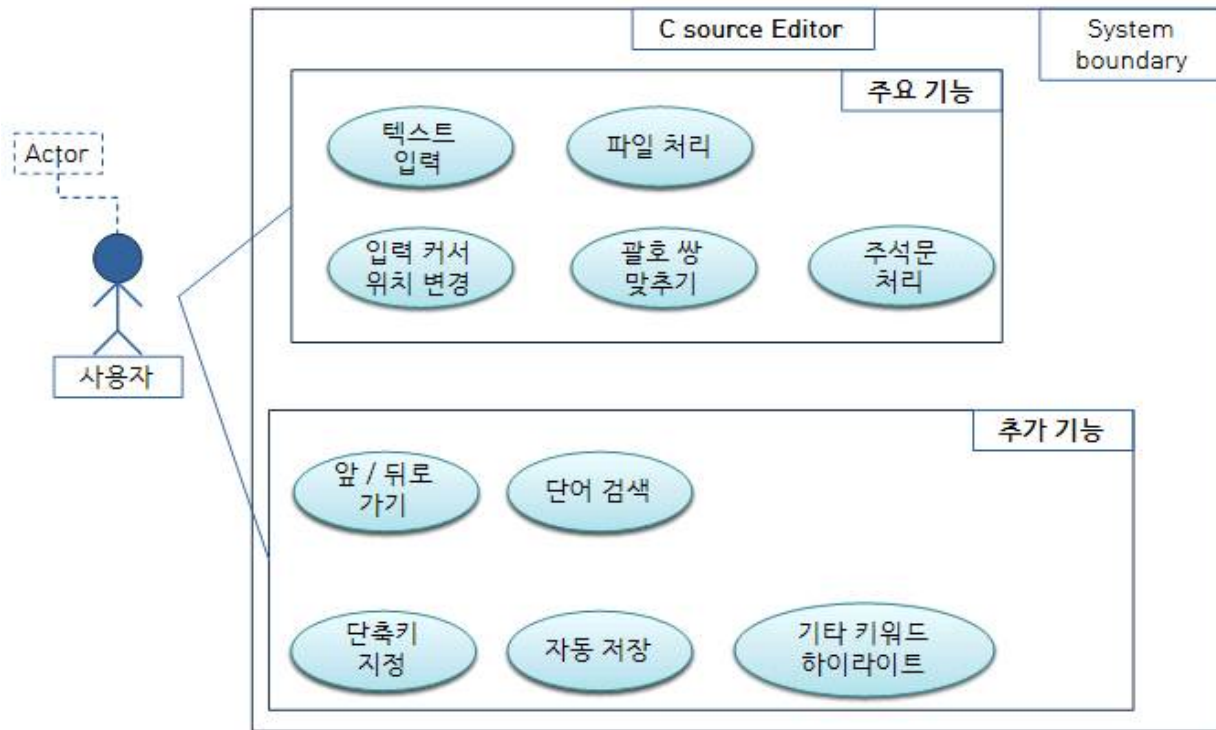
## 3-2. 비기능적 요구사항

비기능적 요구사항은 기능적 요구사항 이외에 프로그램의 성능, UI 등에 관한 내용이 포함되어 있으며 이들은 프로그램의 구조에 직접적인 영향을 미친다. 문제를 토대로 분석한 비기능적 요구사항은 다음과 같다.

- 1) UI의 배치는 기존의 소프트웨어(메모장, gedit)와 유사하여야 한다.
- 2) 저장 시 사용자가 입력한 정보에 대한 누락이 없어야 된다.
- 3) 각각의 동작들이 수행될 때 지연시간이 없어야 한다.
- 4) 각각의 기능들은 충돌이 발생하지 않고 병렬적으로 진행될 수 있어야 한다.

### 3-3. Usecase 다이어그램

앞에서 참고했던 기능적 요구사항을 바탕으로 Usecase 다이어그램을 그리면 다음과 같다.



<C-source 편집기 유즈케이스 다이어그램>

#### Usecase 시나리오

각각의 Usecase에 대해 시나리오를 작성하면 다음과 같다.

##### Usecase 시나리오 : 텍스트 입력

1. 사용자가 키보드로 텍스트를 입력한다.
2. 프로그램이 입력 텍스트를 화면에 즉시 출력해준다.

##### Usecase 시나리오 : 파일 처리

1. 사용자가 프로그램의 메뉴바를 통해 동작을 선택한다.
2. 프로그램은 사용자의 입력에 따라 동작을 수행한다.

##### Usecase 시나리오 : 괄호 쌍 맞춤

##### 성공 시나리오

1. 사용자가 키보드로 텍스트를 입력한다.
2. 프로그램이 입력 텍스트를 읽고 괄호의 여는 부분과 닫는 부분을 읽는다.
3. 괄호 쌍이 올바르게 존재한다면 화면에서 다른 색상으로 보여준다.



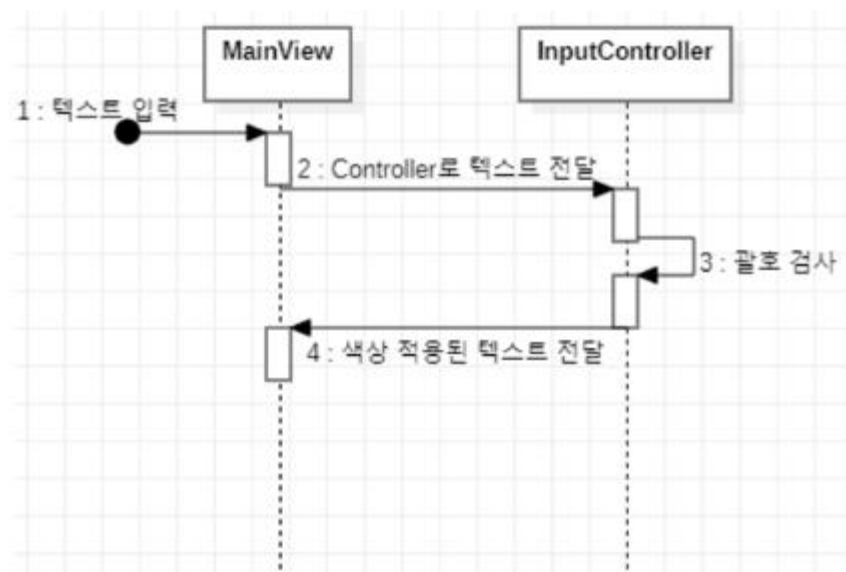
## 실패 시나리오

2-1.괄호가 존재하지 않는 경우,텍스트 색상을 변경하지 않는다.

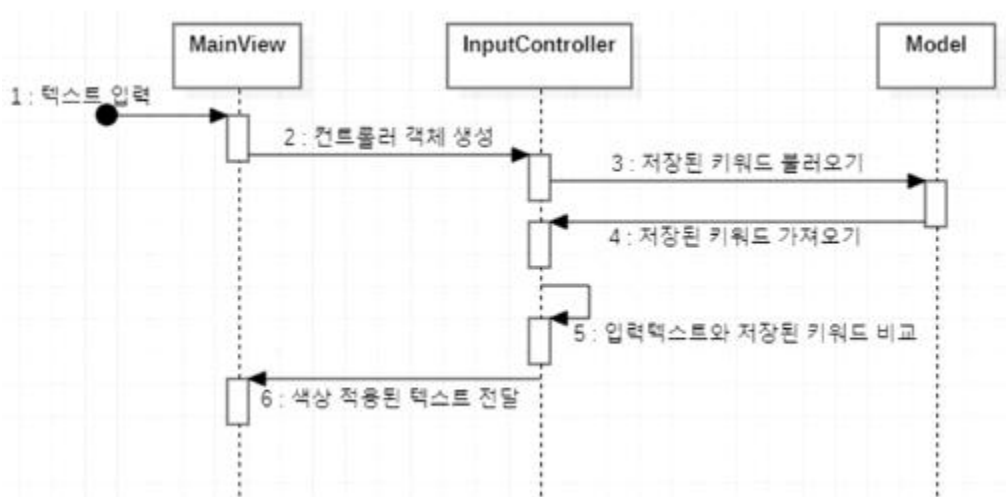
3-1.괄호가 존재하지만 짝이 맞지 않는 경우,색상값을 변경하지 않는다.

## 3-4. 시퀀스 다이어그램

하나의 Usecase에 대해서 시퀀스 다이어그램을 통해서 객체 간 호출 시기 및 동작 시기를 분석할 수 있다.  
주요 기능인 괄호 검사와 키워드 강조에 대한 시퀀스 다이어그램을 살펴보면 다음과 같다.



<시퀀스 다이어그램 : 괄호 검사>



<시퀀스 다이어그램 : 키워드 강조>

키워드 강조의 시퀀스 다이어그램을 확인해보면 전형적인 MVC패턴의 구조를 확인해볼 수 있다. View를 통해서 사용자에게 입력을 받으면 controller는 Model에서 저장된 값을 가져와서 비교하는 방식으로 생각을 해보았으며 MVC에 대해선 추후에 자세히 설명할 예정이다.

## 4. 시스템 설계

앞서 우리 문제의 기초적인 특징들을 살펴보았다. 이를 바탕으로 구현할 프로그램의 전반적인 시스템을 설계하고 모듈들을 명세한다. 또한 이번 과제를 관통하는 MVC 디자인 패턴에 대해서도 알아볼 예정이다.

### 4-1. 프로그램의 전체 기능 및 목표 명세

자바를 사용하여 C언어 소스코드 편집기를 구현한다. 간단한 문법 검사(괄호 검사 등)을 포함하여 각 키워드에 색상을 부여하는 등 IDE의 기능을 일부 구현하는 것이라 생각하는 것이 이해가 빠를 것이다. 다음은 프로그램의 세부 기능 요소들이다.

최종결과물에 반드시 포함되어야 하는 **필수요소**이다.

- 기본 메뉴: 새 파일, 열기, 닫기, 저장, 종료
- 이동기능: 화살표 키 및 이동키 이용
- 편집 기능: 기본 입력 및 수정, DEL
- C 문법 검사 기능: 괄호 짝 맞추기(괄호만 다른 색으로 표시). 주석문 표시

다음은 과제 설명 문서에 명시된 **추가요소**이다.

- 블록 처리: 입력을 받아 일정영역 선택 후 이동, 삭제, 복사, 붙여넣기
- Undo-Redo
- 단어 검색, 바꾸기
- 자동 저장
- 단축 키 기능
- C 문법 검사 추가 기능: 키워드, 식별자, 상수 등을 구분하여 색을 다르게 표현

### 4-2. 프로그램을 구성하는 모듈

프로그램의 기능들을 하나씩 담당하는 모듈들을 기획한다. 이는 다음장의 프로그램 모듈 설계 파트에서 자세하게 다룰 예정이므로 간단하게 보도록 한다.

프로그램을 제작할 때 필수적인 기능들을 담은 모듈은 세 가지이다. 이 세 가지 모듈들은 MVC 디자인 패턴의 컴포넌트에 포함하여 작동하며 각 모듈은 하나의 컨셉을 가지고 작동한다. 모듈명은 한글로 작성하였으며 이는 다음장에 세세히 살펴본다.

## 1. C언어 문법 색상 지정 모듈

- 키워드 색상 지정
- 헤더명 색상 지정
- 식별자 색상 지정
- 상수 색상 지정
- 주식 및 괄호 색상 지정

\*에러일 경우 특이한 색을 지정해서 알려준다.

먼저 문법의 색상을 정해주는 모듈이다. 시중에서 사용하는 텍스트 에디터나 IDE에선 코드를 작성하기 편리하고 직관적으로 살펴보기 용이하도록 각 키워드 값과 개발자가 지정한 상수들에 색상을 부여한다.

## 2. 파일(메뉴) 이벤트 모듈

- 파일 만들기
- 파일 불러오기
- 파일 저장(자동 저장)
- 파일 종료

\*단축키 지원 가능

두 번째로는 존재하는 파일을 가져오거나 저장 등등 파일에 대한 전반적인 처리들을 담당하는 파일 이벤트 모듈이다. 이 모듈에서 핵심적인 부분은 자동 저장인데 이는 모듈 설계 파트에서 자세하게 다루도록 한다.

### 3. 편집 이벤트 모듈

- 단일 검색
- 다중 검색
- 단일 바꾸기
- 다중 바꾸기

\*단축키 지원 가능

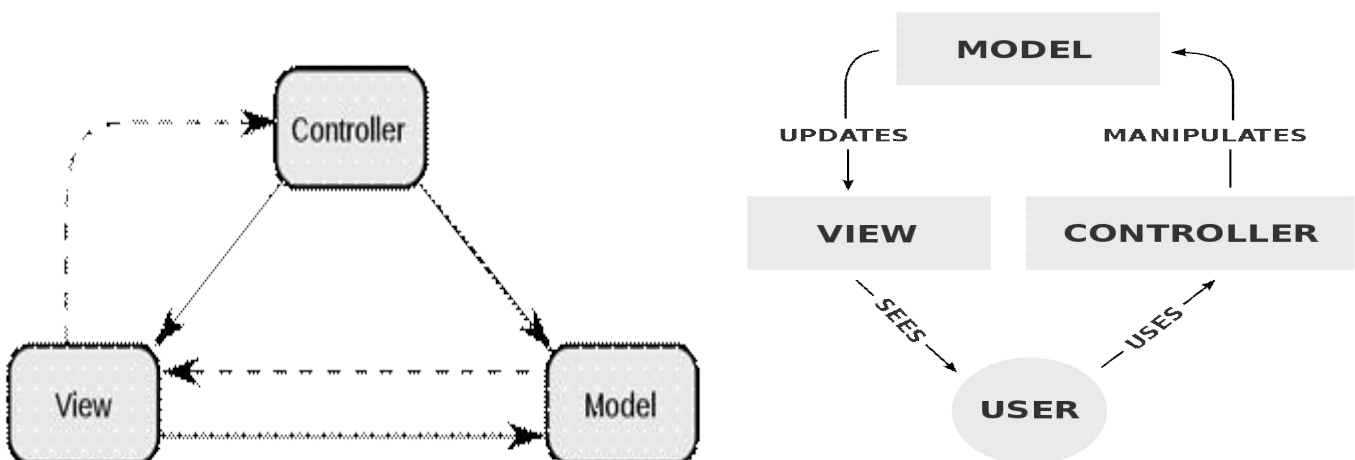
마지막 모듈인 편집 이벤트 모듈이다. 사용자가 필요한 키워드들을 검색하고 이를 수정할 수 있는 기능들을 제공한다. 단수와 복수 단위로 제공된다.

이 세가지 모듈들은 모두 MVC 디자인 패턴에 의거하여 작동한다. 이는 바로 밑에 자세히 기술되어있는 파트를 살펴보도록 하자.

## 4-3. MVC 패턴의 이해 및 클래스 다이어그램

프로그램 모듈들의 작동을 제약하고 약속을 가진 디자인 패턴이다. 리뷰어가 알아 볼 수 없는 코드, 즉 스파게티 코드를 작성하면 서비스를 배포한 후에 있을 유지보수가 쉽지 않다. 이를 대비하고 좀 더 쉽고 편리하게 수정하여 시간을 절약하는 약속이 바로 로직을 분리하여 관리하는 디자인 패턴 중 하나인 MVC 패턴이 되겠다.

MVC 패턴은 총 세 가지의 컴포넌트로 구성되어 있는데 Model, View, Controller의 약자를 딴 이름이다.



위 그림처럼 Controller를 조작하여 Model을 통하여 데이터를 가져오고 이의 대한 결과물을 View로 표현하여 사용자에게 보여지는 역할을 한다. 그럼 이 세가지 컴포넌트의 역할과 규칙을 살펴봐야 한다.

## 1. Model

데이터를 가지고 있다. 우리 과제를 예로 들면 전처리문(#include, #define 등)과 헤더파일, 데이터 타입명, 사용자가 직접 선언한 상수명을 이 모델이 모조리 알고 있다는 뜻이다. 쉽게 말해 데이터 베이스라고 이해하면 되겠다. 다음은 모델에 대한 규칙이다.

**모델에는 사용자가 편집하길 원하는 모든 데이터를 가지고 있어야 한다.** 즉 모델은 모든 정보를 알고 있어야 한다는 뜻이다.

**뷰나 컨트롤러에 대해 알고 있으면 안 된다.** 모델은 데이터 그 자체만을 가지고 있어야 한다는 의미이며 이는 각 컴포넌트가 각자의 역할을 수행해야 하며 다르게 말하면 유지보수에 유리함을 꾀하는 규칙이기도 하다.

**변경이 일어나면 변경통지를 해야 한다.** 변경이 일어나면 이벤트를 발생시켜 누군가에게 전달 해야 하기 때문이다.

## 2. View

문자 그대로 사용자 인터페이스 요소를 나타낸다. 모델의 데이터와 컨트롤러의 수정을 거친 결과물을 출력을 담당하는 컴포넌트이다. 우리 과제에선 GUI 요소 라고 보면 된다. 다음은 뷰에 대한 규칙들을 나열하였다.

**모델이 가지고 있는 데이터를 따로 저장해서는 안된다.** 즉 어떠한 결과물을 출력하는 역할만 수행하여야 하며 그에 대한 필요한 정보는 저장하지 않는다.

**다른 컴포넌트(View, Controller)에 대해 무지하여야 한다.** 전 규칙과 그 목적이 같다. 뷰는 결과물을 출력하는 목적 하나만으로 설계된 컴포넌트이므로 다른 역할을 가지고 있어선 안 된다는 뜻이다.

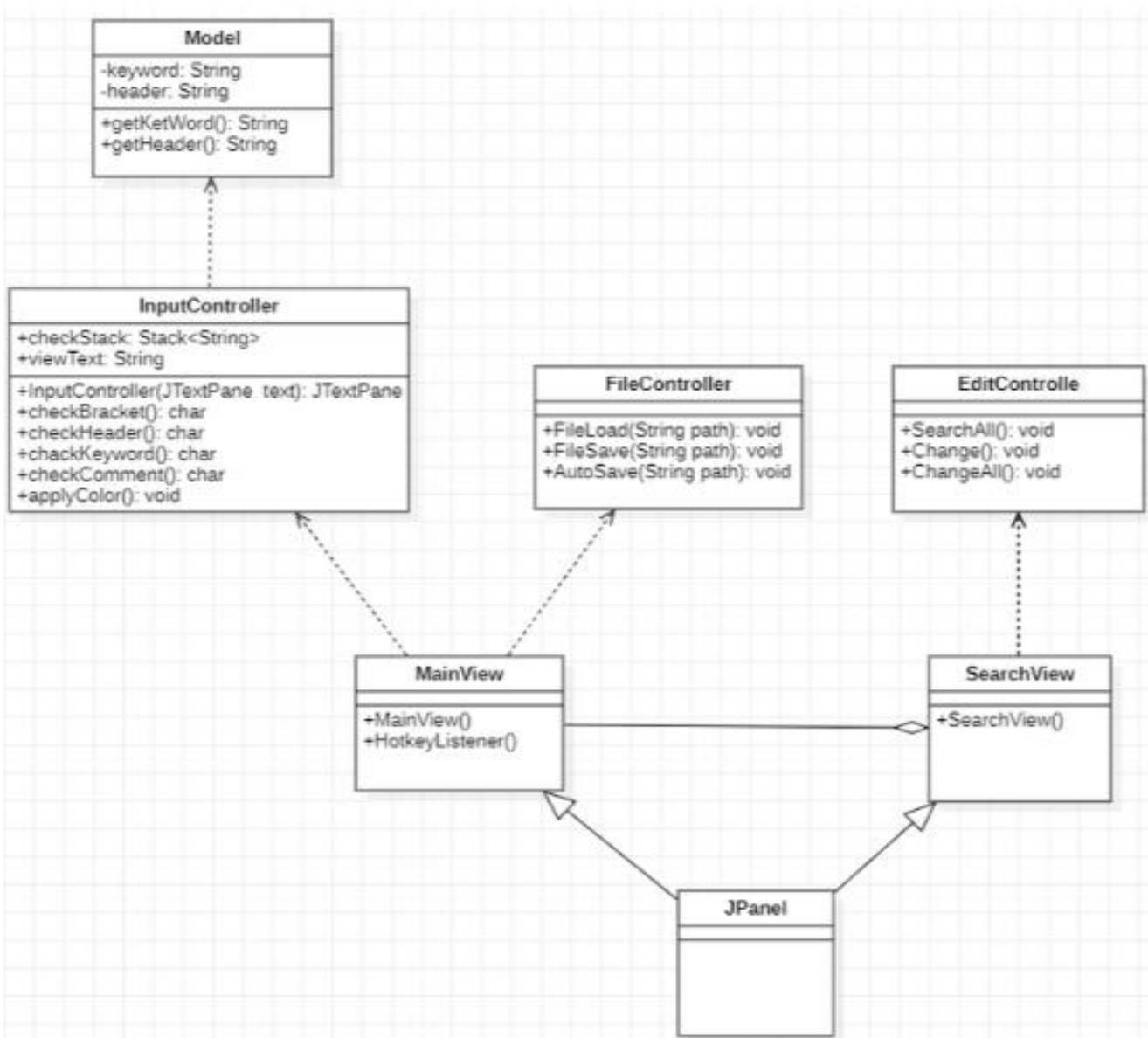
**변경이 일어나면 변경통지를 해야 한다.** 이는 뷰에서 변경이 일어나면 이에 대한 이벤트를 모델에 전달하여 모델을 변경하여야 하는 데에 그 목적이 있다.

### 3. Controller

데이터와 사용자 인터페이스를 연결하는 컴포넌트이다. 즉 모델의 데이터를 사용자 요청에 맞게 가공하여 뷰에 전달하는 역할을 하는 만큼 세가지 컴포넌트 중 가장 핵심이라고도 볼 수 있다. 컨트롤러에 대한 규칙은 다음과 같다.

**컨트롤러는 모델이나 뷰에 대해 알고 있어야 한다.** 나머지 두 컴포넌트는 서로의 존재를 인지하지 못 하지만, 컨트롤러는 이 둘을 중재를 목적으로 이 둘을 알고 있어야 한다.

**그리고 모델과 뷰의 변경을 모니터링 해야 한다.** 변경 통지를 받으면 이를 해석하여 각각의 구성요소에 인지시켜야 한다. 모델과 뷰의 마지막 규칙과 일맥상통한다.



<C-source 편집기 클래스 다이어그램>

MVC패턴을 바탕으로 C-source 편집기에 대해서 클래스 다이어그램을 구성해보았다. 우선 View는 일반적인 메모장 인터페이스의 UI, MainView와 검색기능을 사용했을 때 나타나는 별도의 창, SearchView를 생각하여 총 두 개를 생각하였다. 이들은 각각 자바 스윙의 라이브러리를 상속받아 만들 예정이다. 그리고 간단한 문법검사 및 키워드 강조에 대한 구현은 InputController에서 할 예정이며 파일 처리에 관한 내용은 FileController에서 할 예정이다.

이 두 개의 Controller는 모두 MainView에서 데이터 입력 및 이벤트를 입력받아 동작하게 된다. 그리고 EditController는 SearchView에서 입력을 받아 동작하도록 만들 예정이며 이 기능은 흔히 편집기에서 찾기 기능을 사용하였을 때 해당 단어에 대해 바꾸기 및 모두 바꾸기를 본 따 만들 예정이다. EditController에서는 별도의 내부데이터를 사용하지 않아 Model과의 연결점은 없으며 InputController의 경우 키워드들을 내부데이터에서 관리해야 하므로 Model에서 이에 관한 정의를 하고 사용하게 된다.

## 4-4. 정리

시스템 설계에선 프로그램의 최종 목표를 분석하고 이를 위해 로직을 세워 모듈의 초안을 제작한다. 이것을 MVC 패턴을 기반으로 구성하고 최종적으로 과제에서 요구하는 목적을 포함한 프로그램을 구현할 예정이다. 다음 장에선 구성한 모듈을 구체화하는 프로그램 모듈 설계를 살펴보고자 한다.



## 5. 프로그램 모듈 설계

시스템 설계에서 초안을 만든 모듈을 이제 구체화하는 작업을 거친다. 자세한 모듈의 입출력과 제약조건, 주요 변수와 그 목적을 명세하고 이에 대한 알고리즘을 작성한다. 설계의 꽃이라고도 볼 수 있다. 유지 보수를 위해선 프로그램 설계 부분을 먼저 건드려야 하기에 세부적인 설계가 필요하다.

### 5-1. 모듈 사용 목적, 기능 및 제약조건 명세

시스템 설계에서 제작한 모듈을 본격적으로 쓰임새와 그 기능을 명세한다. 첫 번째로 목적, 기능과 제약조건을 살펴보자.

#### 1. C언어 문법 색상 지정 (InputController)

실행 및 제약조건:

MVC 패턴에 따라서 색상지정을 위해 필요한 데이터는 Model에서 받아온다.

MVC 패턴에 따라서 사용자에게 보여주는 화면은 View에서 출력한다.

사용 목적과 기능:

사용자가 키보드로 텍스트를 입력하면 C언어 문법에 기준 해서 색상을 지정해준 뒤 출력해준다.

#### 2. 파일 이벤트 (FileController)

실행 및 제약조건:

MVC 패턴에 따라서 사용자에게 보여주는 화면은 View에서 출력한다.

사용 목적과 기능:

사용자가 GUI로 구현되어있는 파일 버튼을 클릭하거나 단축키 기능을 이용하면 원하는 기능을 실행하여준다.

#### 3. 편집 이벤트 (EditController)

실행 및 제약조건:

MVC 패턴에 따라서 사용자에게 보여주는 화면은 View에서 출력한다.

사용 목적과 기능:

사용자가 GUI로 구현되어있는 편집 버튼을 클릭하거나 단축키 기능을 이용하면 원하는 기능을 실행하여준다.

#### 4. 편집 창 (EditView)

실행 및 제약조건:

MVC 패턴에 따라서 사용자의 입력에 대한 처리는 Controller에서 해야한다.

사용 목적과 기능:

편집 창에 대한 사용자의 입력에 대한 변경이 일어나면 변경 통지를 InputController, FileController에 해준다.

#### 5. 메인 화면 (MainView)

실행 및 제약조건:

MVC 패턴에 따라서 사용자의 입력에 대한 처리는 Controller에서 해야한다.

사용 목적과 기능:

메인화면에 대한 사용자의 입력에 대한 변경이 일어나면 변경 통지를 EditController에 해준다.

#### 6. C언어 문법 색상 지정을 위해 사용할 데이터 (Model)

실행 및 제약조건:

MVC 패턴에 따라서 정보에 대한 처리는 Controller에서 해야한다.

사용 목적과 기능:

C언어 문법 색상 지정을 위한 정보를 담고 있다.

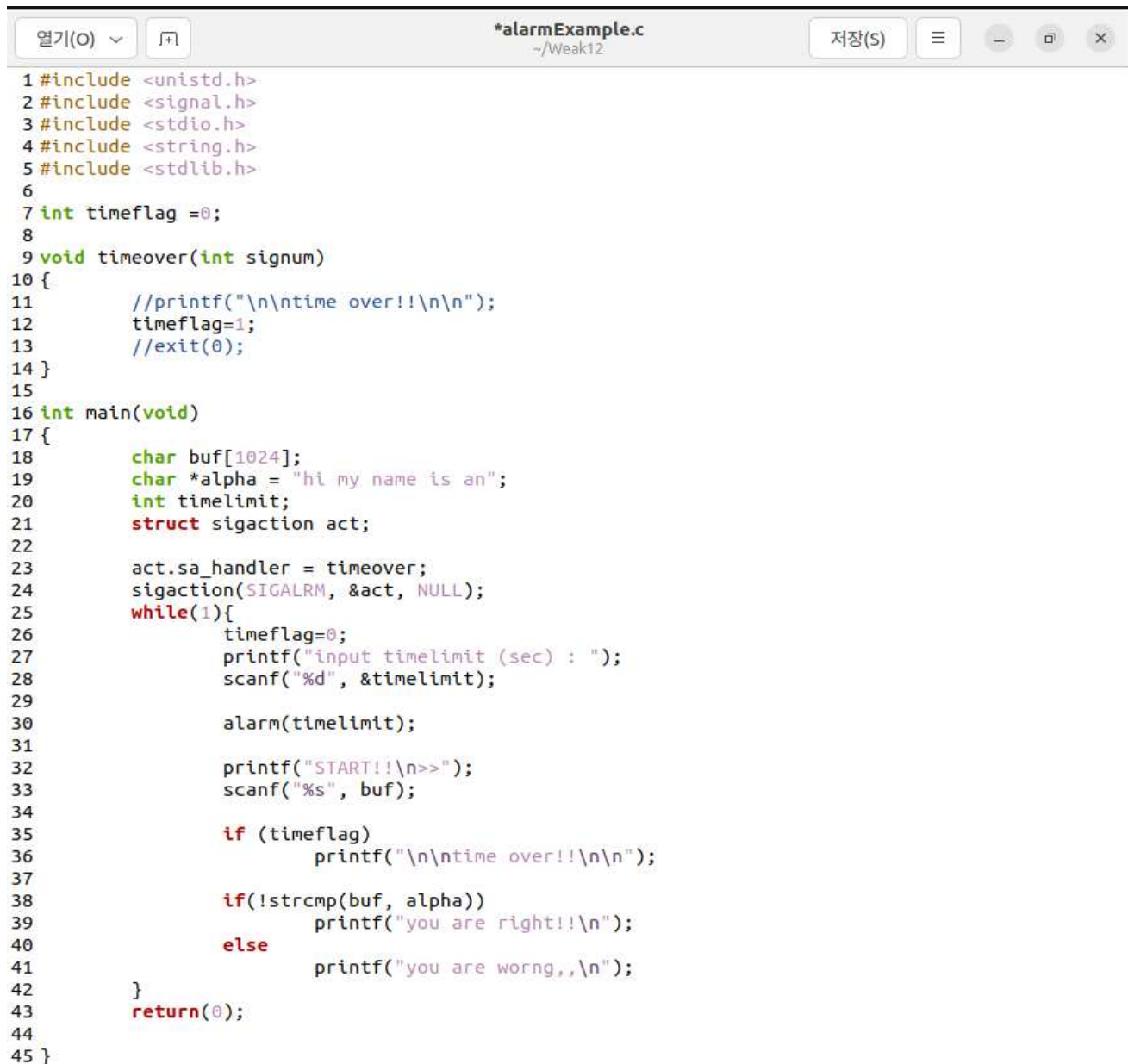
## 5-2. 모듈의 입출력 및 실행예시 명세

다음으로 모듈의 입출력 및 실행예시 명세이다. 우리가 설계한 실행예시 그림은 후에 나오는 인터페이스 설계에서 기술할 예정이다. 여기에서는 우리가 설계하면서 참고한 예시를 명세 하겠다. 또한 Model 과 View에 대한 명세는 Controller에대한 명세로 충분하므로 생략하겠다.

## 1. C언어 문법 색상 지정 모듈

INPUT: 사용자가 키보드로 텍스트를 입력한다

OUTPUT: 프로그램이 입력 텍스트를 C언어에 맞게 색상을 지정해준 뒤 화면에 즉시 출력해준다.



```
1 #include <unistd.h>
2 #include <signal.h>
3 #include <stdio.h>
4 #include <string.h>
5 #include <stdlib.h>
6
7 int timeflag =0;
8
9 void timeover(int signum)
10 {
11     //printf("\n\ntime over!!\n\n");
12     timeflag=1;
13     //exit(0);
14 }
15
16 int main(void)
17 {
18     char buf[1024];
19     char *alpha = "hi my name is an";
20     int timelimit;
21     struct sigaction act;
22
23     act.sa_handler = timeover;
24     sigaction(SIGALRM, &act, NULL);
25     while(1){
26         timeflag=0;
27         printf("input timelimit (sec) : ");
28         scanf("%d", &timelimit);
29
30         alarm(timelimit);
31
32         printf("START!!\n>>");
33         scanf("%s", buf);
34
35         if (timeflag)
36             printf("\n\ntime over!!\n\n");
37
38         if(!strcmp(buf, alpha))
39             printf("you are right!!\n");
40         else
41             printf("you are wrong,,\n");
42     }
43     return(0);
44 }
45 }
```

## 2. 파일 이벤트 모듈

INPUT: 사용자가 GUI로 구현되어있는 파일 버튼을 클릭하거나 단축키 기능을 이용한다.

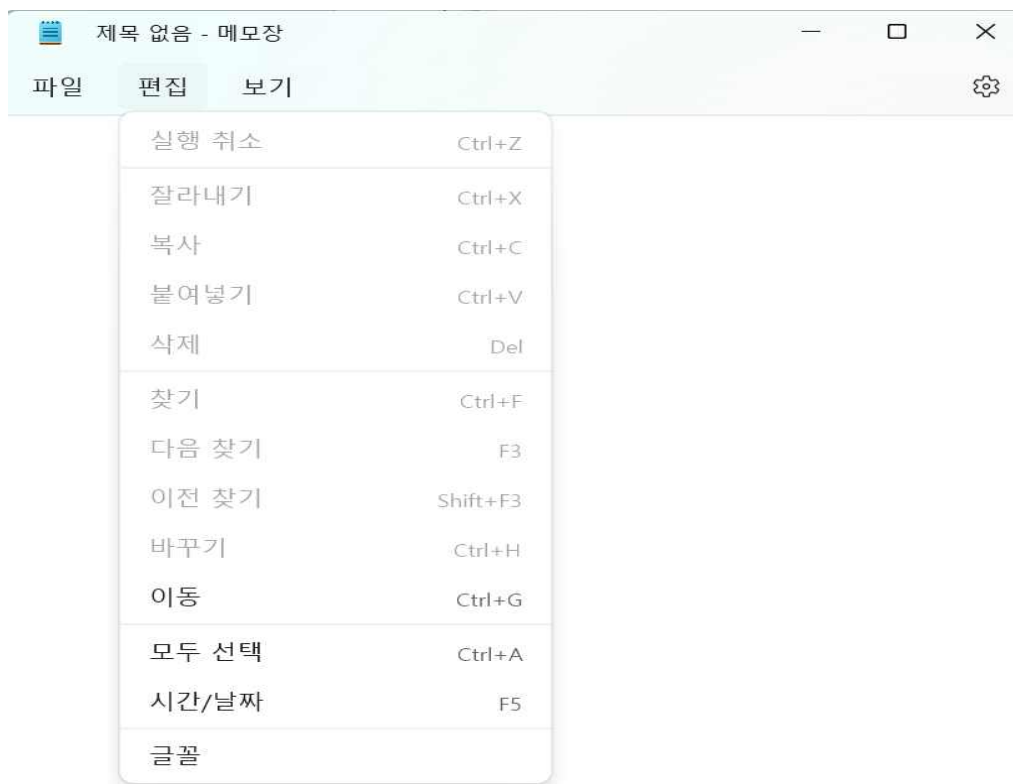
OUTPUT: 사용자가 원하는 기능을 수행한다.(파일 만들기, 파일 불러오기, 파일 저장, 파일 종료)

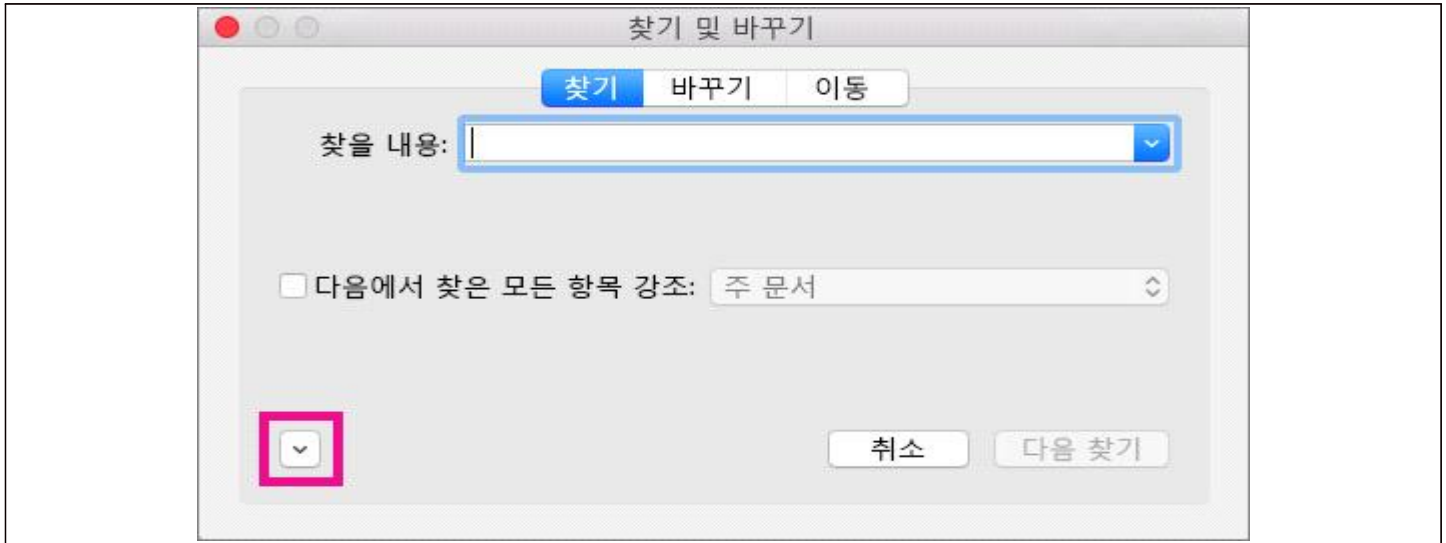


### 3. 편집 이벤트 모듈

INPUT: 사용자가 GUI로 구현되어있는 편집 버튼을 클릭하거나 단축키 기능을 이용한다.

OUTPUT: 사용자가 원하는 기능을 수행한다. (단일 검색, 다중검색, 단일 바꾸기, 다중 바꾸기)





### 5-3. 모듈의 알고리즘 (pseudo code) 및 중요 속성 기술

요구사항 분석, 시스템 설계를 거쳐 모듈의 정보들을 투합해 직접 코드로 옮겨지는 작업이 수행된다. 이 작업을 보다 수월하게 가이드라인을 제공하는 pseudo code를 기술한다. 구현과 가장 밀접한 관계를 지녔으므로 설계 보고서 중 가장 빈번하게 보는 파트가 아닐까 싶다. MVC 구조에서 모델과 뷰 부분은 데이터와 GUI 적인 코드만 있으므로 컨트롤러 부분만 pseudo code로 기술할 예정이다.

이해를 돕기 위해 모델과 뷰의 대략적인 구조를 기술하겠다.

Model	View
<ul style="list-style-type: none"> <li>- Keyword //C언어 문법 색상 지정을 위해 사용할 키워드 데이터</li> <li>- Header //C언어 문법 색상 지정을 위해 사용할 헤더파일명 데이터</li> </ul>	<ul style="list-style-type: none"> <li>-Main_View //프로그램 실행 시 C언어 문법 색상 지정을 해주는 소스코드 입력창과 메뉴버튼, 편집버튼으로 이루어져 있음</li> <li>-Edit_View //편집 버튼 클릭 시 실행되는 편집 창 (검색 및 바꾸기 기능 실행)</li> </ul>

## 1. C언어 문법 색상 지정 모듈

(키워드 색상 지정, 헤더파일 색상지정, 식별자 색상지정, 상수 색상지정, 주석 및 괄호 색상지정)

### 주요속성

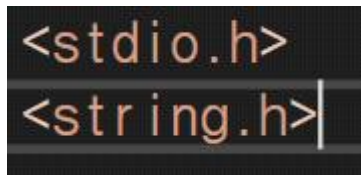
-키워드 , 헤더명 과 같은 데이터는 모델에서 받아옴 (모델:사용자가 편집할 모든 정보를 담고 있음)

-view에서 사용자가 편집기에 기록한 내용을 쉬지 않고 계속 받아와야 하므로 쓰레드로 구현

키워드:프로그래밍할 때 정의된 구문을 의미한다. 즉 특별한 의미를 가지고 있는 단어들이라 생각하면 됨.

char	int	double	float	short
break	if	else	switch	case
long	while	do	static	return
extern	continue	for	void	default
goto	sizeof	const	register	typeof
ifdef	ifndef	case	enum	signed
volatile	unsigned	auto	union	

헤더파일:C언어에서 내부적으로 지원해주는 키워드를 제외한 모든 것



식별자:변수, 상수, 함수, 구조체, 포인터 등에 부여된 이름을 이야기함. 아래 사진에서 num과 alpha가 식별자임

```
int num;  
char alpha;
```

상수: 변하지 않는 데이터 아래 사진에서 song이 상수임

```
#define song 3.14
#include<stdio.h>

int main()
{
    song;
}
```

### 키워드 색 지정하기 기능

word\_list ← 사용자가 입력한 내용을 공백을 기준으로 분리하여 저장한 배열

word\_list\_size ← 공백을 기준으로 분리하여 저장한 배열의 크기

keyword\_list ← 모델에서 받아오는 키워드를 저장한 배열

keyword\_list\_size ← 키워드를 저장한 배열의 크기

file\_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수

file\_contents\_word\_index ← 파일 내용에서 색을 바꿔 주어야하는 키워드의 위치를 담을 변수

```
for i←0 to i<word_list_size {
    for j←0 to j<keyword_list_size {
        if(word_list [i] = keyword_list [j]) {
            file_contents_word_index ← file_contents에서 word_list[i]의 위치
            for k← file_contents_word_index to k<file_contents_word_index +word_list[i]_size {
                file_contents[k]의 색 변경 } } }
```

### 헤더파일 색 지정하기 기능

#### 주요속성

#### -헤더파일은 항상 #include 뒤에 나옴

word\_list ← 사용자가 입력한 내용을 공백을 기준으로 분리하여 저장한 배열

word\_list\_size ← 공백을 기준으로 분리하여 저장한 배열의 크기

header\_file\_list← 모델에서 받아오는 헤더파일명을 저장한 배열

header\_file\_list\_size ← 헤더파일명을 저장한 배열의 크기

file\_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수

file\_contents\_word\_index ← 파일 내용에서 색을 바꿔 주어야하는 헤더파일의 위치를 담을 변수

```
for i←0 to i<word_list_size {
    if(word_list [i-1] = #include) {
        for j←0 to j<header_file_list_size {
            if(word_list [i] = header_file_list[j]) {
                file_contents_word_index ← file_contents에서 word_list[i]의 위치
                for k← file_contents_word_index to k<file_contents_word_index +word_list[i]_size {
                    file_contents[k]의 색 변경 } } }
```

### 상수 및 식별자 색 지정하기 기능

#### 주요속성

-상수 및 식별자의 색을 지정해주려면 **#define** 또는 데이터 타입 키워드 다음에 나오는 단어를 저장하고 있어야함.

word\_list ← 사용자가 입력한 내용을 공백을 기준으로 분리하여 저장한 배열

word\_list\_size ← 공백을 기준으로 분리하여 저장한 배열의 크기

alias\_list← #define 또는 데이터 타입 키워드 다음에 나오는 단어를 저장하는 배열

alias\_list\_size ← alias\_list 배열의 크기

file\_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수

file\_contents\_word\_index ← 파일 내용에서 색을 바꿔 주어야하는 상수 및 식별자의 위치를 담을 변수

```
for i←0 to i<word_list_size {
    if(word_list [i-1] = #define || word_list [i-1] = 데이터 타입 키워드 ) {
        file_contents_word_index ← file_contents에서 word_list[i]의 위치
        alias_list=alias_list+word[i]
    }
}

for i←0 to i<word_list_size {
    for j←0 to j<alias_list_size {
        if(word_list [i] = alias_list[j] ) {
            file_contents_word_index ← file_contents에서 word_list[i]의 위치
            for k← file_contents_word_index to k<file_contents_word_index +word_list[i]_size {
                file_contents[k]의 색 변경 } } }
```

## 주석 및 괄호 및 다음표 색상 변경 기능

### 필요한 자료구조 설계

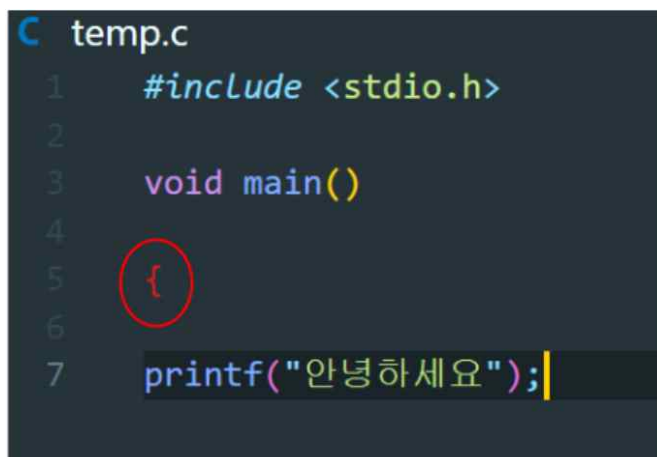
① 열림 괄호 및 주석 탐지 시 해당 괄호를 순서대로 기록

② 닫힘 괄호 및 주석 탐지 시 바로 직전에 탐지된 괄호 또는 주석과 짝이 맞는지 확인 짝이 맞으면 바로 직전에 탐지된 열림 괄호 또는 열림 주석은 더 이상 고려하지 않음

-> 입력순서의 역순으로 자료를 참조해야하므로 STACK를 사용해야함.

### 주요속성

-짝이 맞지 않는 경우 다른 색으로 알려주어야 함.



- { , // , /\* , " 의 나오는 순서에 따라 모든 경우들을 고려해주어 스택에 넣어주어야함.



```
#include<stdio.h>

int main()
{
    //
    " // " ;
    // " "
    / * " " * /
    " / * * / " ;
}
```

괄호의 경우 뒤에 주석과 따옴표가 나와도 상관없다.  
 주석의 경우 뒤에 따옴표가 나오면 주석처리해준다.  
 따옴표의 경우 뒤에 주석이 나오면 문자열처리해준다.

#### -//(한줄 주석)의 처리 알고리즘

char\_list ← 사용자가 입력한 내용을 한 문자씩 분리하여 저장한 배열  
 char\_list\_size ← 한 문자씩 분리하여 저장한 배열의 크기  
 file\_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수  
 file\_contents\_word\_index ← 파일 내용에서 색을 바꿔 주어야하는 주석의 위치를 담을 변수  
 stack.push ←스택에 데이터를 입력한다.  
 stack.pop ←스택에서 가장 최근에 넣은 데이터를 제거한다.  
 stack.peek ←스택에서 가장 최근에 넣은 데이터를 인출한다.  
 for i←0 to i<char\_list\_size - 1 {  
 if(char\_list [i] = '/' && word\_list [i+1] = '/' ) {  
 stack.push('//')  
 for j←i to j<char\_list\_size {  
 if(char\_list[j] != '\n')  
 file\_contents[j]의 색변경  
 else break }  
  
 if(stack.peek == '//') {  
 stack.pop }  
 }  
 }

#### -/\* \*/ (열고 닫는 주석)의 처리 알고리즘

char\_list ← 사용자가 입력한 내용을 한 문자씩 분리하여 저장한 배열  
 char\_list\_size ← 한 문자씩 분리하여 저장한 배열의 크기  
 file\_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수  
 file\_contents\_word\_index ← 파일 내용에서 색을 바꿔 주어야하는 주석의 위치를 담을 변수  
 stack.push ←스택에 데이터를 입력한다.  
 stack.pop ←스택에서 가장 최근에 넣은 데이터를 제거한다.  
 stack.peek ←스택에서 가장 최근에 넣은 데이터를 인출한다.  
 for i←0 to i<char\_list\_size -1 {  
 if(char\_list [i] = '/' && word\_list [i+1] = '\*' ) {  
 stack.push('/\*')  
 for j←i to j<char\_list\_size {

```

if(char_list[j] != '*' || char_list[j+1] != '/')
    file_contents[j]의 색 빨간색으로 변경
else break }

```

```

if(stack.peek == '/') {
    if(char_list[i] == '*' && char_list[i+1] == '/')
        for j←stack.peek한 위치 to j<i+2 {
            file_contents[j]의 색 초록색으로 변경
        }
    stack.pop }

```

```

if(char_list [i] = '*' && word_list [i+1] = '/' ) {
    if(stack.peek != '/')
        file_contents[i],file_contents[i+1]의 색 빨간색으로 변경 }}

```

### - " (따옴표)의 처리 알고리즘

char\_list ← 사용자가 입력한 내용을 한 문자씩 분리하여 저장한 배열  
char\_list\_size ← 한 문자씩 분리하여 저장한 배열의 크기  
file\_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수  
file\_contents\_word\_index ← 파일 내용에서 색을 바꿔 주어야하는 주석의 위치를 담을 변수  
stack.push ←스택에 데이터를 입력한다.  
stack.pop ←스택에서 가장 최근에 넣은 데이터를 제거한다.  
stack.peek ←스택에서 가장 최근에 넣은 데이터를 인출한다.  
for i←0 to i<char\_list\_size {  
 if(char\_list [i] = "") {  
 stack.push("")  
 for j←i to j<char\_list\_size {  
 if(char\_list[j] != "" && char\_list[j] != '/n')  
 file\_contents[j]의 색 빨간색으로 변경  
 else break }  
  
 if(stack.peek == "") {  
 if(char\_list[i] == "" || char\_list[i] == 'wn')  
 for j←stack.peek한 위치 to j<i {  
 file\_contents[j]의 색 주황색으로 변경  
 }  
 stack.pop }

### -괄호( '{', '[', '(' ) 처리 알고리즘

char\_list ← 사용자가 입력한 내용을 한 문자씩 분리하여 저장한 배열  
char\_list\_size ← 한 문자씩 분리하여 저장한 배열의 크기  
file\_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수  
file\_contents\_word\_index ← 파일 내용에서 색을 바꿔 주어야하는 주석의 위치를 담을 변수  
stack.push ←스택에 데이터를 입력한다.  
stack.pop ←스택에서 가장 최근에 넣은 데이터를 제거한다.  
stack.peek ←스택에서 가장 최근에 넣은 데이터를 인출한다.  
for i←0 to i<char\_list\_size {  
 if(char\_list [i] = '{' || char\_list [i] = '[' || char\_list [i] = '(' ) {  
 stack.push(char\_list [i])  
 file\_contents[i]의 색 빨간색으로 변경  
  
 if(stack.peek == '{' || stack.peek == '[' || stack.peek == '(' ) {  
 if(char\_list[i] == '}' || char\_list[i] == ']' || char\_list[i] == ')' )  
 file\_contents[j] , file\_contents[stack.peek한 위치]의 색 괄호의 색으로 변경  
 stack.pop }  
  
 elif(char\_list[i] == '}' || char\_list[i] == ']' || char\_list[i] == ')' ) {

```
file_contents[i]의 색 빨간색으로 변경 }}
```

## -각 알고리즘의 핵심 부분

### 키워드 색 지정하기 기능

```
if(word_list [i] = header_file_list[j]) {  
    for k← file_contents_word_index to k<file_contents_word_index +word_list[i]_size {  
        file_contents[k]의 색 변경 } } }
```

->사용자가 입력한 단어가 키워드(break, if, else ..)라면 색을 변경해준다.

### 헤더파일 색 지정하기 기능

```
for i←0 to i<word_list_size {  
    if(word_list [i-1] = #include) {  
->#include 뒤에 헤더파일이 나오는 점을 이용한다.  
if(word_list [i] = header_file_list[j]) {  
    for k← file_contents_word_index to k<file_contents_word_index +word_list[i]_size {  
        file_contents[k]의 색 변경 } } }
```

->사용자가 입력한 단어가 헤더파일(stdio.h, string.h)라면 색을 변경해준다.

### 상수 및 식별자 색 지정하기 기능

```
if(word_list [i-1] = #define || word_list [i-1] = 데이터 타입 키워드 ) {  
    file_contents_word_index ← file_contents에서 word_list[i]의 위치  
->#define 또는 데이터 타입 키워드뒤에 나오는 단어를 저장한다.  
if(word_list [i] = alias_list[j] ) {  
    file_contents_word_index ← file_contents에서 word_list[i]의 위치  
    for k← file_contents_word_index to k<file_contents_word_index +word_list[i]_size {  
        file_contents[k]의 색 변경 } } }
```

->사용자가 입력한 단어가 저장한 단어와 같다면 색상을 변경해준다.

### //(한줄 주석)의 처리 알고리즘

```
if(char_list [i] = '/' && word_list [i+1] = '/' ) {  
    stack.push("//")  
->//을 만나면 스택에 PUSH해준다.  
for j←i to j<char_list_size {  
    if(char_list[j] != '\n')  
        file_contents[j]의 색변경  
->만약 줄바꿈을 만나지만 않는다면 색을 변경하여준다.  
if(stack.peek == '//') {  
    stack.pop }  
->만약 STACK의 가장 최근에 삽입한 값이 //라면 pop해준다.
```

### /\* \*/ (열고 닫는 주석)의 처리 알고리즘

```
if(char_list [i] = '/' && word_list [i+1] = '*' ) {  
    stack.push('/*')  
->만약/*을 만난다면 스택에 PUSH해준다.  
for j←i to j<char_list_size {  
    if(char_list[j] != '*' || char_list[j+1] != '/')  
        file_contents[j]의 색 빨간색으로 변경  
->만약 */를 만나지않는다면 오류니 빨간색으로 표시해준다.  
if(char_list[i] == '*' && char_list[i+1] == '/')  
    for j←stack.peek한 위치 to j<i+2 {  
        file_contents[j]의 색 초록색으로 변경  
->만약 */를 만난다면 pop해주고 주석의 색인 초록색으로 색을 변경해준다.  
if(char_list [i] = '*' && word_list [i+1] = '/' ) {  
    if(stack.peek != '/*')
```

file\_contents[i],file\_contents[i+1]의 색 빨간색으로 변경 }}

->만약 \*/를 만났으나 스택의 젤 최근 값이 \*/이 아닐 경우 /\*으로 열리지 않은 주석이므로 \*/을 빨간색으로 에러 처리

### - " (따옴표)의 처리 알고리즘

```
if(char_list [i] == '"') {  
    stack.push('"')
```

->만약 " 을 만난다면 스택에 PUSH해준다.

```
for j←i to j<char_list_size {  
    if(char_list[j] != '"' && char_list[j] != '/n')  
        file_contents[j]의 색 빨간색으로 변경
```

->만약 "를 만나지않는다면 오류니 빨간색으로 표시해준다. 또한 줄바꿈이 나올 때 까지만 빨간색으로 표시해주어야 한다.

```
if(char_list[i] == '"' || char_list[i] == '\n')
```

```
for j←stack.peek한 위치 to j<i {  
    file_contents[j]의 색 주황색으로 변경
```

->만약 " 를 만난다면 pop해주고 따옴표의 색인 주황색으로 색을 변경해준다.

### -괄호( '{' , '[' , '(' ) 처리 알고리즘

```
if(char_list [i] == '{' || char_list [i] == '[' || char_list [i] == '(' ) {  
    stack.push(char_list [i])  
    file_contents[i]의 색 빨간색으로 변경
```

->만약 열린 괄호를 만난다면 스택에 PUSH해준다.

```
if(stack.peek == '{' || stack.peek == '[' || stack.peek == '(' ) {  
    if(char_list[i] == '}' || char_list[i] == ']' || char_list[i] == ')')  
        file_contents[j] , file_contents[stack.peek한 위치]의 색 괄호의 색으로 변경
```

->만약 열린괄호가 가장 최근에 스택에 넣은 값이고 닫힌 괄호를 만난다면 pop해주고 괄호의 색으로 색을 변경해 준다.

```
elif(char_list[i] == '}' || char_list[i] == ']' || char_list[i] == ')') {  
    file_contents[i]의 색 빨간색으로 변경 }}
```

->만약 반대로 닫힌괄호만을 만난다면 열린괄호가 없는 오류이므로 빨간색으로 표시해준다.

## 2. 파일 이벤트 모듈

(파일 만들기, 파일 불러오기, 파일 저장하기, 파일 종료하기)

```
if(사용자가 메뉴에서 나가기 버튼 클릭)  
    시스템 탈출
```

```
else if(사용자가 새 파일 만들기 버튼 클릭)  
    if(편집기에 내용이 남아 있다면)  
        if(사용자가 내용 저장을 원한다면)  
            파일 저장 기능 실행  
            편집기 초기화  
    else  
        편집기 초기화
```

```
else if(사용자가 파일 불러오기 버튼 클릭)  
    if(편집기에 내용이 남아있다면)  
        if(사용자가 내용 저장을 원한다면)  
            파일 저장 기능 실행  
            파일 불러오기 기능 실행  
    else  
        파일 불러오기 기능 실행
```

```
else if(사용자가 파일 저장 버튼 클릭)
    파일 저장 기능 실행
    자동저장 기능 실행
```

### 파일 저장하기 기능

```
path ← 사용자가 저장을 원하는 경로를 저장하는 변수
file_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수
data ← 입력한 파일 내용을 담는 변수
new_file ← path를 이용해서 새로운 파일 생성

data ← file_contents
new_file .write(data)
new_file .close
```

### 파일 불러오기 기능

```
line ← 사용자가 지정한 파일 내용을 한 줄씩 읽어오는 변수
data ← 읽어온 파일 내용을 담는 변수

while( line.next_line != null ) {
    data ← data + "\n"; }
display("data ")
```

### 파일 자동저장 기능

#### 주요속성

- 쓰레드로 구현
- 파일 저장이 완료된 후에만 실행 하여야 함.

```
path ← 사용자가 저장을 원하는 경로를 저장하는 변수

sleep(일정시간)
파일저장기능(path)
```

-각 알고리즘의 핵심 부분

파일 이벤트 모듈은 주로 파일 입출력에 관한 알고리즘이므로 핵심부분 기술은 생략하겠다.

## 3. 편집 이벤트 모듈

### (단일 검색, 다중 검색, 단일 바꾸기, 다중 바꾸기)

```
if(사용자가 편집에서 단일 검색 버튼 클릭)
    단일 검색 기능 수행

else if(사용자가 편집에서 다중 검색 버튼 클릭)
    다중 검색 기능 수행

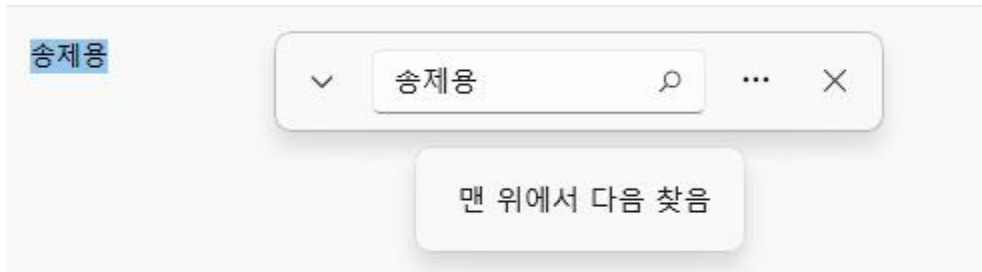
else if(사용자가 편집에서 단일 바꾸기 버튼 클릭)
    단일 바꾸기 기능 수행

else if(사용자가 편집에서 다중 바꾸기 버튼 클릭)
    다중 바꾸기 기능 수행
```

## 단일 검색 기능

### 주요 속성

-단일 검색 기능 실행 시 검색한 해당 단어를 표시해줌



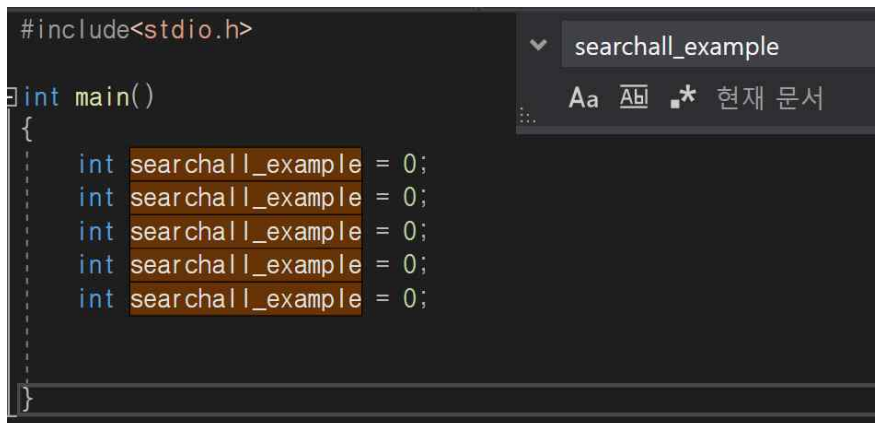
```
data ← 검색 기능 수행을 위해 사용자가 입력한 검색단어
data_length ← 검색 단어의 길이
file_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수
data_index ← file_contents에서 data의 위치 값을 담는 변수
data_last_index ← file_contents에서 data의 마지막 위치 값을 담는 변수
```

```
if( file_contents에 data가 포함되어있다면 ) {
    for i←data_index to i<data_index + data_length {
        file_contents[i] 다른 색으로 표시 }
    data_index ← data_index.next
    if( data_index >= data_last_index ) {
        data_index ← 0 }
else
    display("데이터가 없음")
```

## 다중 검색 기능

### 주요 속성

-다중 검색 기능 실행 시 검색한 해당 모든 단어를 표시해줌



```
data ← 검색 기능 수행을 위해 사용자가 입력한 검색단어
data_length ← 검색 단어의 길이
file_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수
data_index ← file_contents에서 data의 위치 값을 담는 변수
data_last_index ← file_contents에서 data의 마지막 위치값을 담는 변수
```

```
if( file_contents에 data가 포함되어있다면 ) {
    while(1) {
        for i←data_index to i<data_index + data_length {
            file_contents[i] 다른 색으로 표시 }
        data_index ← data_index.next
        if( data_index >= data_last_index ) {
```

```

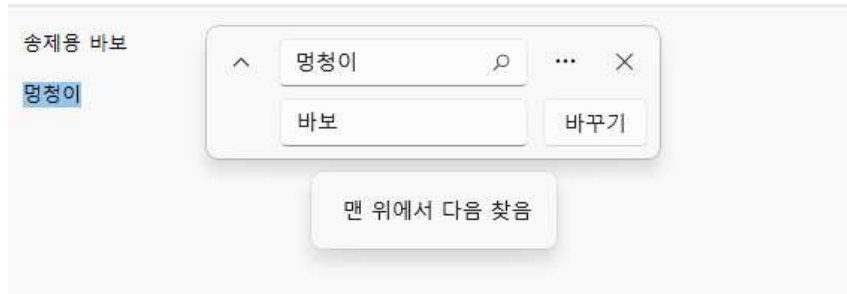
        break }
    }
else
    display("데이터가 없음")

```

## 단일 바꾸기 기능

### 주요속성

-단일 바꾸기 기능 실행 시 해당 위치를 표시해주고 바꾸기를 실행함



```

data ← 바꾸기 기능 수행을 위해 사용자가 입력한 검색단어
change_data ← 바꾸기 기능 수행을 위해 사용자가 입력한 바꿀 단어
data_length ← 검색 단어의 길이
file_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수
data_index ← file_contents에서 data의 첫 번째 위치 값을 담는 변수
serch_check ← 검색기능 수행 후 바꾸기 기능 수행을 위한 변수

```

```

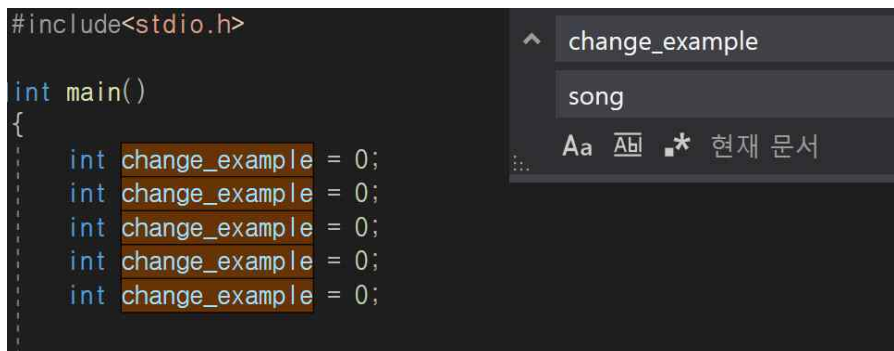
if(serch_check != 1) {
    if( file_contents에 data가 포함되어있다면 ) {
        for i←data_index to i<data_index + data_length {
            file_contents[i] 다른 색으로 표시 }
        serch_check ← 1
    }
    else
        display("데이터가 없음")
}
else {
    file_contents에서 change_data 해당하는 첫 번째 값 변경 //자바 ReplaceFirst 사용 예정
    serch_check ← 0
}

```

## 다중 바꾸기 기능

### 주요속성

-다중 바꾸기 기능 실행 시 해당 단어의 모든 위치를 표시해주고 바꾸기를 실행함



```

data ← 검색 기능 수행을 위해 사용자가 입력한 검색단어

```

```

change_data ← 바꾸기 기능 수행을 위해 사용자가 입력한 바꿀 단어
data_length ← 검색 단어의 길이
file_contents ← 사용자가 입력한 파일 내용을 읽어오는 변수
data_index ← file_contents에서 data의 위치 값을 담는 변수
data_last_index ← file_contents에서 data의 마지막 위치값을 담는 변수
serch_check ← 검색기능 수행 후 바꾸기 기능 수행을 위한 변수

if(serch_check != 1) {
  if( file_contents에 data가 포함되어있다면) {
    while(1) {
      for i←data_index to i<data_index + data_length {
        file_contents[i] 다른 색으로 표시 }
      data_index ← data_index.next
      if( data_index >= data_last_index ) {
        break } }
    serch_check ← 1
  } else
    display("데이터가 없음")
} else {
  file_contents에서 change_data 해당하는 모든 값 변경 //자바 ReplaceALL 사용 예정
  serch_check ← 0
}

```

## -각 알고리즘의 핵심 부분

### 단일검색기능

```

if( file_contents에 data가 포함되어있다면) {
  for i←data_index to i<data_index + data_length {
    file_contents[i] 다른 색으로 표시 }
  ->사용자가 검색한 단어가 포함되어있다면 찾아서 다른 색으로 표시한다.

```

### 다중검색기능

```

while(1) {
  if( data_index >= data_last_index ) {
    break } }
  ->무한반복을 이용해서 사용자가 검색한 단어를 모두 표시하고 마지막 단어라면 탈출한다.

```

### 단일바꾸기기능

```

if( file_contents에 data가 포함되어있다면) {
  for i←data_index to i<data_index + data_length {
    file_contents[i] 다른 색으로 표시 }
  ->사용자가 바꾸기 기능을 수행할 해당 단어의 위치를 사용자에게 알려준다.

```

```

file_contents에서 change_data 해당하는 첫 번째 값 변경 //자바 ReplaceFirst 사용 예정
->해당단어를 바꾸어준다.

```

### 다중바꾸기기능

```

while(1) {
  if( data_index >= data_last_index ) {
    break } }
  ->무한반복을 이용해서 사용자가 바꾸기 기능을 수행할 해당 단어의 위치를 모두 표시한다.

```

```

file_contents에서 change_data 해당하는 모든 값 변경 //자바 ReplaceALL 사용 예정
->해당단어를 모두 바꾸어준다.

```



## 5-4. 정리

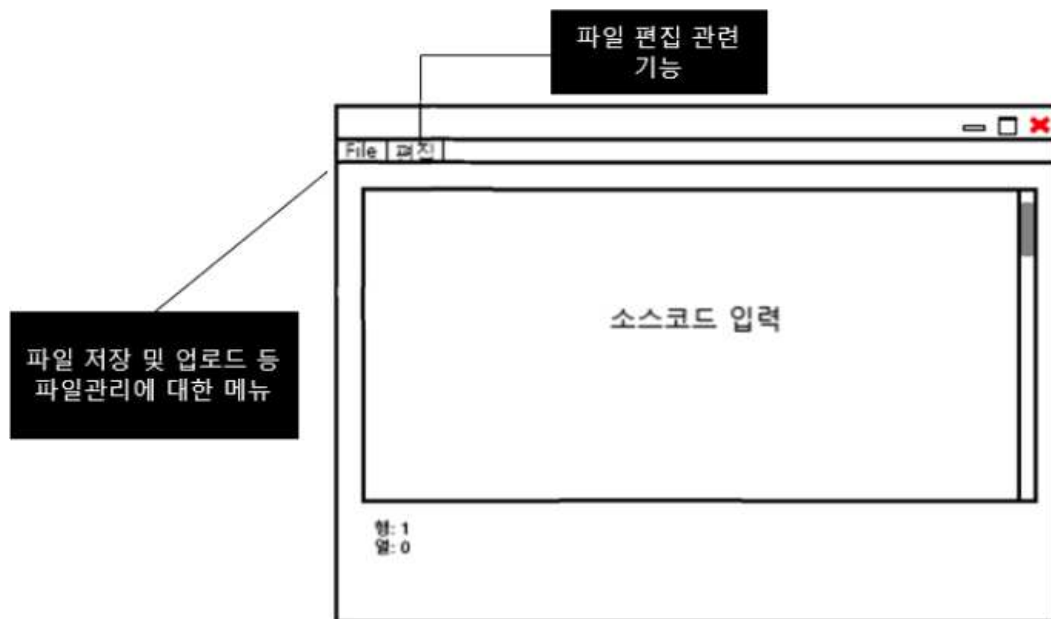
지금까지 프로그램 설계, 즉 시스템 설계에서 명시한 모듈들의 입출력과 기능, 제약조건, 주요 변수 및 슈도코드를 명세하고 이를 설명하였다. 가장 핵심적인 부분인 만큼 정교하고 자세한 설계가 필요하여 가장 많은 시간을 투자하지 않았나 싶다. 다음 파트에선 각 모듈들 사이의 정보처리, 신호를 주고받는 접점, 즉 인터페이스 설계를 기술하려고 한다.

# 7. 인터페이스 설계

프로그램 실행 시 사용자 측면에서 볼 수 있는 UI와 인터페이스의 실행 흐름을 살펴볼 수 있는 인터페이스 구조도, 사용자와 상호작용 하는 과정에서 사용되는 모듈에 대해 살펴본다.

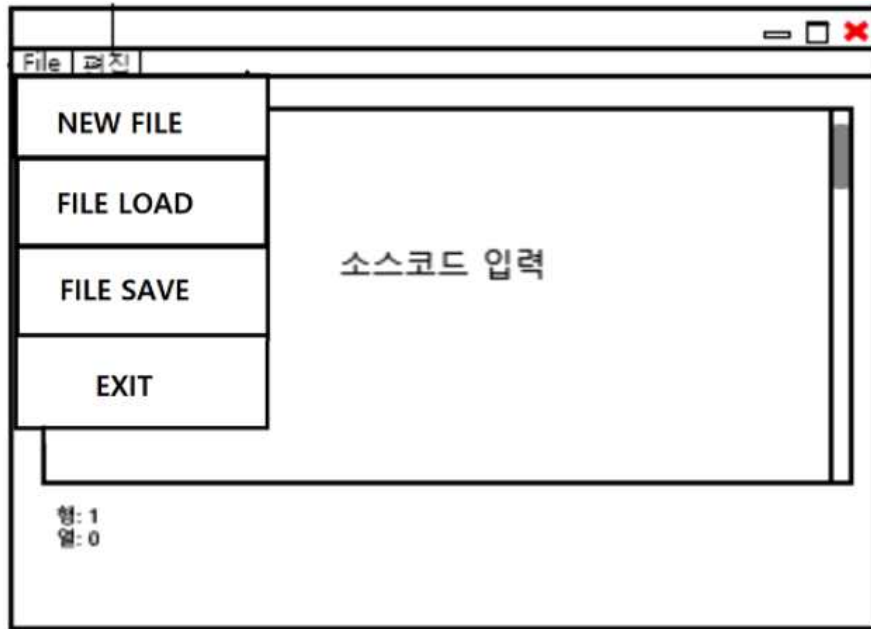
## 7-1. 프로그램 UI

< Main UI >



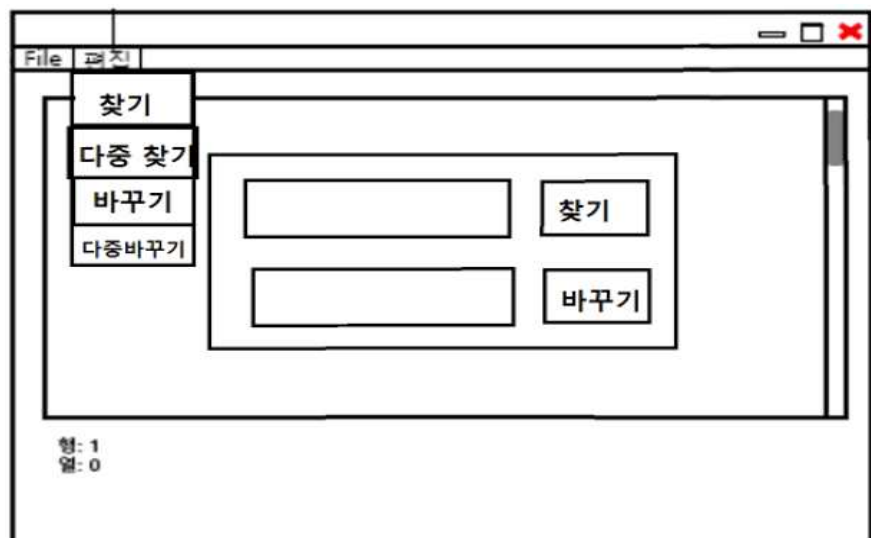
- 프로그램이 실행되면 가장 먼저 사용자에게 보여지는 창으로, 소스코드를 입력할 수 있는 창과 왼쪽 상단에 FILE 메뉴와 편집 메뉴가 자리하고 있는 것을 확인할 수 있다.
- 소스코드 입력을 하는 부분은 보여지는 화면보다 코드의 길이가 늘어날 시 스크롤바가 생성되어 전체적인 코드를 확인할 수 있고, 위 창에서 C 언어 문법 검사를 한 결과가 사용자에게 반환되게 된다.

### < File UI >



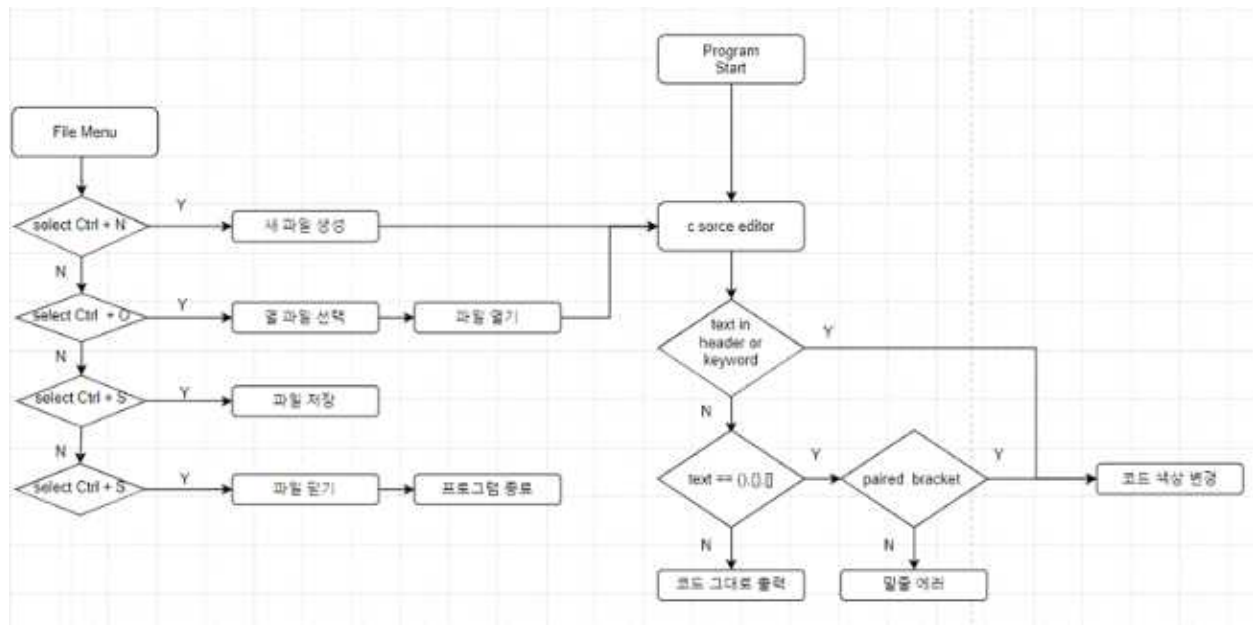
- 왼쪽 상단의 FILE 메뉴를 클릭하였을 때 사용자에게 보여지는 화면이다.
- File 메뉴는 파일 관리에 대한 메뉴로 새 파일 생성, 파일 열기, 파일 저장, 프로그램 종료를 할 수 있는 기능을 가지고 있다.

### < 편집 UI >



- 편집 UI를 클릭하였을 때 사용자에게 보여지는 화면이다.
- 편집 메뉴는 검색 기능에 관한 메뉴로 단일/다중 단어 찾기와 바꾸기를 수행하는 기능을 가지고 있다.

## 7-2. 인터페이스의 전체적인 구조 도식화 및 설명



- 현재 프로그램이 객체지향으로 설계되어 플로우 차트로 모든 기능을 나타내기에 어려움이 있으나 설계된 인터페이스를 중심으로 프로그램의 대략적인 구조 및 도식에 대해 설명하도록 하겠다.
- 프로그램이 실행되면 화면 창에는 C소스 에디터 창과 상단의 메뉴 2개(FILE, 편집) 기능이 나타나게 된다.
- C 소스 에디터 창에 C 코드를 입력하면 문법 검사가 진행된다. 입력된 텍스트가 미리 정의해둔 header 또는 keyword에 존재하거나 괄호라면 코드의 색상을 변경하게 된다. 특히 괄호의 경우, 괄호의 짝이 맞는지 확인하여 쌍으로 존재한다면 정의해둔 색상을 입히고, 짝이 맞지 않다면 밑줄 에러를 발생시킨다. 앞서 말한 위 모든 경우가 아닌 경우는 코드를 그대로 출력하여 사용자에게 보여주도록 한다.
- 파일 메뉴 창을 클릭 또는 해당 단축키 입력 시 새 파일 생성, 파일 불러오기, 파일 저장, 프로그램 종료를 수행한다.
- 현재 플로우차트엔 존재하진 않으나 편집 메뉴도 파일 메뉴 창이 수행되는 과정과 동일한 방식으로 각 기능을 수행하게 된다. (단일/다중 검색 및 바꾸기)

## 7-3. 인터페이스의 입출력 구조 명세

### < C 언어 문법 색상 지정 모듈 >

기능	입력 및 출력
C 언어 문법 색상 지정 모듈	입력: 사용자가 키보드로 입력한 텍스트
	출력: 지정된 색으로 변환된 텍스트
키워드 색상 지정	입력: 미리 정의된 키워드 입력
	출력: 색상이 입혀진 키워드
헤더파일 색상 지정	입력: 미리 정의된 헤더파일명 입력
	출력: 색상이 입혀진 헤더파일명
식별자 색상 지정	입력: 미리 정의된 식별자 입력
	출력: 색상이 입혀진 식별자
상수 색상 지정	입력 : 미리 정의된 상수 입력
	출력: 색상이 입혀진 상수
주석 및 괄호 색상 지정	입력: 주석(//, /* */) 및 괄호((), {}, [])
	출력: 색상이 입혀진 주석 및 괄호

### < 파일 처리 모듈 >

기능	입력 및 출력
파일 처리 모듈	입력: GUI 메뉴 버튼을 클릭 or 단축키 입력
	출력: 사용자가 원하는 파일처리 수행
파일 만들기 (New File)	입력: New File 클릭 또는 Ctrl + N
	출력: 기존 내용이 사라진 c 코드 에디터
파일 열기 (File Load)	입력: File Load 클릭 또는 Ctrl + O
	출력: 저장되어 있던 c 코드를 에디터에 출력
파일 저장 (File Save)	입력: File Save 클릭 또는 Ctrl + S
	출력: 에디터에 작성된 c언어 파일 ( ~.c)
프로그램 종료 (Exit)	입력: Exit 클릭 또는 Ctrl + Q
	출력: 종료된 프로그램

### < 편집 처리 모듈 >

기능	입력 및 출력
편집 처리 모듈	입력: GUI 편집 버튼을 클릭 or 단축키 입력
	출력: 사용자가 원하는 편집 기능을 수행
단일 검색	입력: 단일 검색 클릭 또는 Ctrl + F
	출력: 검색한 단일 단어 백그라운드 highlight 표시
다중 검색	입력: 다중 검색 클릭 또는 Ctrl + G
	출력: 검색한 단어 모두 코드 내 백그라운드 highlight 표시
단일 바꾸기	입력: 단일 바꾸기 클릭 또는 Ctrl + R / 바꿀 단어
	출력: 바뀐 단어 출력 및 해당 단어 백그라운드 highlight 표시
다중 바꾸기	입력 : 다중 바꾸기 클릭 또는 Ctrl + T / 바꿀 단어
	출력: 바뀐 단어 출력 및 해당 단어 모두 백그라운드 highlight

## 7-4. 사용자 인터페이스와 직접적으로 통신하는 내부

### - 우리가 정의할 주요 내부 모듈

내부 모듈명	기능 명세
checkBracket()	괄호 짝 맞추기 검사
checkHeader()	입력 단어가 헤더파일인지 검사
checkKeyword()	입력 단어가 키워드인지 검사
checkComment()	입력 단어가 주석문인지 검사
applyColor()	단어에 색을 입히는 기능
change()	찾은 단어를 사용자가 원하는 단어로 변경
changeAll()	찾은 단어 모두 사용자가 원하는 단어로 변경
Search()	찾고 싶은 단어를 찾아주는 기능
SearchAll()	찾고 싶은 단어를 모두 찾아주는 기능

### - UI 관점에서 본 내부모듈 명세

New(Ctrl + N)	: 새로운 파일을 생성하는 기능
LoadFile (Ctrl + O)	: 기존의 파일을 업로드하는 기능
SaveFile (Ctrl + S)	: 현재 파일을 저장하는 기능
Exit (Ctrl + Q)	: 프로그램 종료 기능
Check Bracket()	: 괄호 짝 맞추기 검사
Check Header()	: 헤더파일인지 검사
Check String()	: C 문법 여부 검사
Check Comment()	: 주석문 여부 검사
Check Move()	: 화살표 키 및 이동 키 이용
Check Edit()	: 기본 입력 및 수정, DEL 여부 체크

Search() (Ctrl + F)	: 원하는 단어 검색 기능
SearchAll() (Ctrl + G)	: 원하는 단어 다중 검색 기능
Change() (Ctrl + R)	: 원하는 단어 바꾸기 기능
ChangeAll() (Ctrl + T)	: 원하는 단어 다중 바꾸기 기능

## 8. 논의

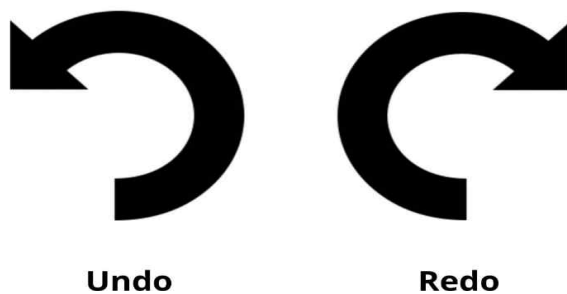
### 8-1. 추가적인 아이디어가 필요한 기능

문법 색상 지정

```
#include<stdio.h>
// " " " // " ;
```

현재 알고리즘은 한 줄을 띄어쓰기 기준으로 나누어 해당 단어가 무엇인지 판단한다. 이 경우 여러 가지 변수들이 생길 수 가있다. 예를 들어 위 사진에서 보이는 것 과같이 C언어에서 헤더파일을 선언하는 `#include<stdio.h>`의 경우를 살펴보면 대부분 띄어쓰기를 하지 않는다. 이 경우에는 띄어쓰기를 하지않아 알고리즘에서는 한 단어로 인식 할 것이고 색상지정을 해주지 않을 것이다. 다행히 C언어에서는 `#include<stdio.h>`와 같이 헤더파일 선언 외에는 키워드를 붙여서 사용하는 것은 없다. 이 경우 예외처리를 해줄 것이다. 또한 괄호, 따옴표, 주석부분알고리즘에서 위 사진과 같이 무엇이 먼저 나오냐에 따라서 처리해줘야 할 것이 달라진다. 이 경우 또한 예외 처리를 해줘야한다. 이번 고급레벨 프로젝트는 여러 가지 변수들을 생각해보고 처리해줄 필요가 있다.

Undo-Redo



Swing의 `JTextComponent`는 기본적으로 Undo/Redo를 지원하기 위한 메카니즘이 준비되어 있다. 그렇다고 해서 디폴트로 텍스트 컴포넌트에 Undo/Redo를 지원하는 것은 아니다. `JTextComponent`에 `Ctrl+Z(undo)`, `Ctrl+Y(redo)` 키 입력을 추가해 구현해볼 예정이다,

블록 처리(마우스나 키보드를 이용해 일정 영역 선택 후 이동)



마우스나 키보드를 이용해 일정 영역을 선택 후 이동하는 기능이다. 문제에서 제시한 추가 기능이  
라 모든 기능이 완료가 된다면 도전해볼 생각이다.

## 8-2. 테스트 및 유지보수 방법 설계

### <테스트>

소프트웨어를 제작하고 보수하는데 있어 테스트는 가히 필수적이라고 할 수 있다. 그에 따라 소프트웨어나 프로그램의 특성에 따라 많은 테스트 방법이 존재하지만, 우리가 채택한 테스트 방법은 “화이트박스 테스트”이다.

### 화이트박스 테스트

화이트 박스 테스트란 소프트웨어 혹은 제품의 내부 구조, 동작을 세밀하게 검사하는 테스트 방식으로 외부에서 요구사항에 따른 예상 결과값을 테스트 하는 것과는 다르게 내부 소스 코드를 테스트하는 기법으로 사용자가 들여다 볼 수 없는 구간의 코드 단위를 테스트 한다.

즉, 정리하면 개발자가 소프트웨어 또는 컴포넌트 등의 로직에 대한 테스트를 수행하기 위해 설계 단계에서 요구된 사항을 확인하는 개발자 관점의 단위테스팅 기법이다.

### 선정이유

1. 우리 프로그램의 기능은 각각의 메소드로 독립적으로 구성될 수 있다.
2. 이에 따라 각 메소드를 고립시켜 각 소스코드의 동작이 원활하게 진행되는지 확인할 수 있다.
3. 팀원들 간의 협업에서 업무분담(테스트)도 원활하게 이루어질 수 있다.
4. 코드 커버리지 테스트가 가능하다.



## 테스트 프레임워크

현재 우리가 프로그램을 작성할 때 사용하는 언어가 JAVA이므로, 테스트 프레임 워크는 JUnit을 사용한다. JUnit은 자바 기반의 단위 테스트 프레임워크로 같은 테스트 코드를 여러번 작성할 필요가 없어 코드 유지보수 및 시간 효율성을 극대화 할 수 있다.

### <유지보수>

또한 다양한 유지보수 종류 중 우리가 선택한 것은 “**완전형 유지보수**”이다. 완전형 유지보수는 코드의 기능 및 효율성을 향상시킬 수 있으며, 사용자의 요구 변화에 따라 시스템의 기능을 변경할 수 있다.

유지보수 프로세스에는 **UP(Unified Process)모델**을 사용하여 진행하게 된다. 특히 우리조가 선택한 개발방법론도 UP를 사용하는데, 이러한 UP 모델을 사용하여 유지보수를 진행하면 반복/점진적으로 유지보수를 하여 효율적인 유지보수가 가능해진다

## 9. 추진 일정 및 역할분담

### 9-1. 추진 일정

(수행 기간 : 2022년 11월 23일 ~ 2022년 12월 22일)					
개발내용 \ 기간(주)	1	2	3	4	담당 팀원
계획 및 설계 발표자료 제작	○				정지수
계획 및 설계 보고서 작성	○	○			김현진
프로그램 구현		○	○		송제용, 안현진
최종 발표자료 제작			○	○	정지수
최종 보고서 작성			○	○	김현진

모든 개발내용은 팀원 함께 다룰 예정이나 중점적으로 담당할 사람을 표에 기술하였다.

담당할 팀원은 프로젝트를 진행하며 추진 일정 및 담당할 포지션은 유동적으로 조정이 가능하며 보고서 마다 각 역할 분담 내용을 다시 기술할 예정이다.

### 9-2. 역할 분담

설계 보고서의 역할분담은 아래와 같이 담당하였다.

안현진	김현진	송제용	정지수
요구사항 분석 - Usecase 분석 - Usecase 다이어그램 - 시퀀스 다이어그램 시스템 설계 - 클래스 다이어그램	시스템 설계 - 프로그램 전체 기능 및 목표 명세 - 프로그램을 구성하는 모듈 - MVC 패턴의 이해 - 보고서 작성	프로그램 모듈 설계 - 모듈의 입출력 명세 - 제약조건 명세 - 알고리즘 설계 논의 - 추가적인 아이디어가 필요한 기능 일정계획	인터페이스 및 유지보수설계 - 인터페이스 설계, 도식화 및 플로우차트 제작 - 내부 모듈 기능 명세 - 테스트 및 유지방법 설계 - ppt 제작