
프로젝트 최종 보고서

C-source 편집기

작성일	2022.11.13	제출일	2022.11.20
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

0. Abstact	4
1. Introduction	5
1-1. 문제의 개요 및 정의와 목표	5
2. Related works	6
2-1. 프로젝트 주제와 연관된 기반 이론	6
2-2. 구현을 위해 필요한 주요 데이터	8
2-3. 사용할 주요 자료구조 설계	10
3. Methodology and Implementation	14
3-1. 모듈의 알고리즘 (pseudo code)	14
3-2. 구현에 활용된 핵심 아이디어 (idea)	15
3-3. 프로그램 전체 아키텍처 및 설계도	20
4. Results	23
4-1. 구현 내용 분석	23
4-2. 설계 구성요소 및 제한요소 평가	35

5. Discussion	36
5-1. 구현 결과의 특성과 이유	36
5-2. 문제 발생 시 원인과 해결 방법	43
6. Conclusion	51
6-1. 향후 발전 및 보안 계획	51
6-2. 결과물 활용 가능성	54
6-3. 기술/경제/사회 파급효과 및 기대효과	55
6-4. 테스트 및 유지보수 프로세스	56
7. Ending	58
7-1. 프로젝트를 마치며	58
7-2. 진행일정	58
7-3. 역할분담	59

0. Abstract

요 약 프로그래머들은 각자가 원하는 프로그래밍 언어를 선택하여 주어진 문제들을 해결한다. 이러한 작업들을 조금 더 간소화 하기 위해 통합 개발 환경(Integrated Development Environment, IDE) 이나 텍스트 에디터를 사용한다. 이들은 프로그래머가 작성한 코드들의 문법의 오류나 예외를 알려주고 컴파일과 디버깅을 한 곳에 집중하게 할 수 있는 장점을 가지고 있다. 따라서 이러한 개발툴이 개발 속도를 향상시키고 프로그래머에게 편의성을 제공한다. 본 프로젝트에서는 객체지향언어를 MVC 디자인 패턴에 의거해 C언어 코드의 괄호 짝 맞추기, 주석문 처리 등 간단한 문법과 파일 불러오기, 이동 기능, 단어 검색 등과 같은 텍스트 편집기의 기능들을 가진 C 소스 편집기를 제작하고 개발 과정 제시와 팀원들의 느낀 점, 결과물의 확장등과 같은 논의를 제안한다. 이는 시중에 존재하는 IDE나 텍스트 에디터에 비해 보잘 것 없는 기능을 가진 편집기이지만 이번 과정이 협업의 중요성을 다시 한 번 일깨우는 계기가 되었다는 게 팀원들의 생각이다.

키워드 : C-언어, MVC 디자인 패턴, 텍스트 에디터

Abstract programmers solves a given problem by selecting the programming language that they want. Using Integrated Development Environment (IDE) or a text editor makes them more comfortable to these tasks. The tools have the advantage of informing the programmer of grammatical errors and exceptions to the code and concentrating compiling and debugging in one place. Thus, such development tools increase development speed and provide convenience to programmers. Based on MVC design patterns, C source editor with simple grammar such as C language code parenthesis matching, annotation processing, reading files, moving functions, word retrieval, etc., is produced. This is an editor that has a boring function compared to IDE and text editors in the market, but the team members think that this process was an opportunity to realize the importance of cooperation again.

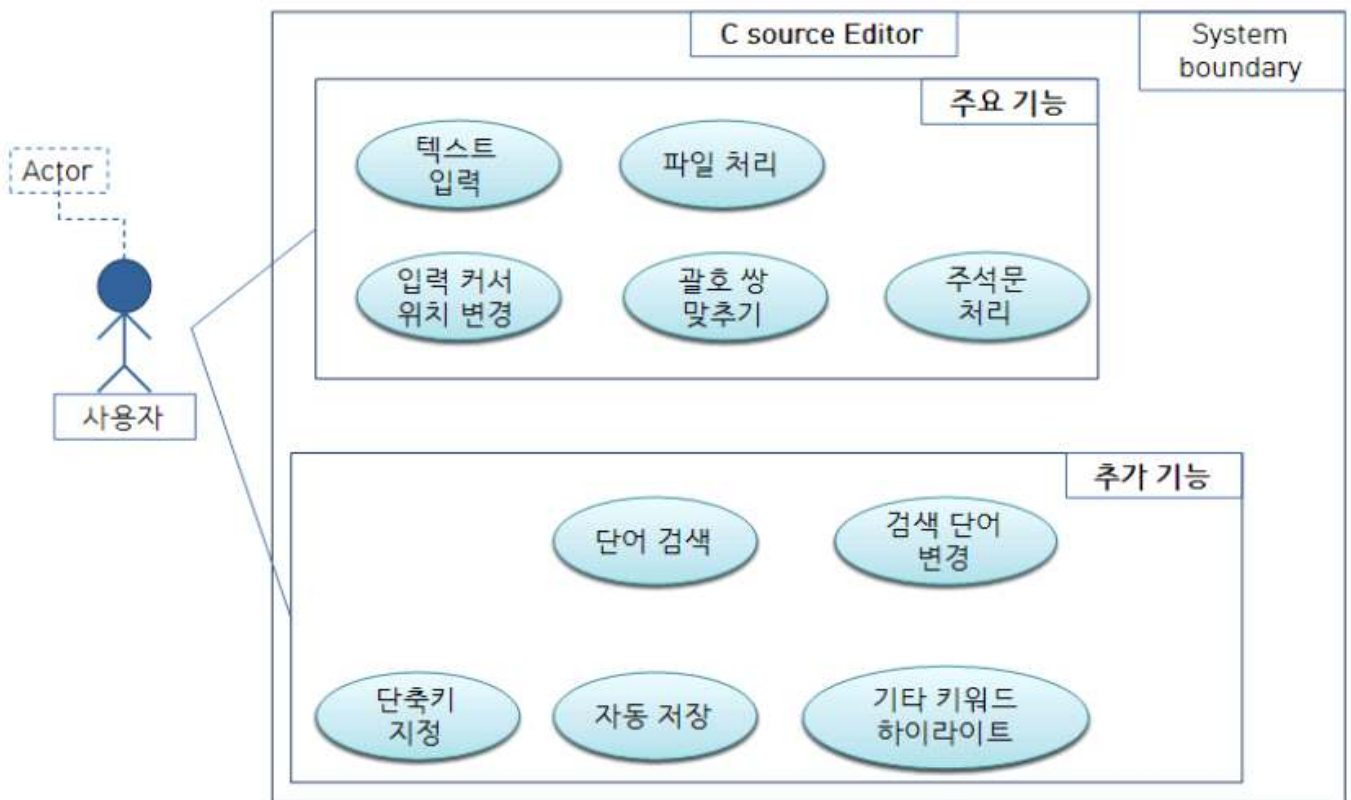
Key words : C language, MVC design patterns, text editor

1. Introduction

1-1. 문제의 개요와 정의 및 목표

우리가 선택한 프로젝트의 주제는 객체 지향 언어를 이용하여 C 소스 편집기를 제작하는 것이다. C소스 편집기는 일반 텍스트 편집기에서 필요한 기능을 더해 C언어의 문법 요소를 체크하는 기능을 가진다. 우리가 개발해야 하는 영역을 아는 것은 굉장히 중요한데 우리가 구현할 영역은 컴파일러의 프론트엔드 부분에 해당한다. 컴파일러는 크게 프론트엔드와 백엔드로 구성되어있는데 그중 프론트엔드에서는 하이레벨 언어로 작성된 소스코드를 분석하는 단계를 수행한다. 이러한 단계를 어휘 분석이나 스캐닝이라고 하며 소스 파일의 토큰, 즉 소스코드를 구성하는 텍스트 심볼을 의미한다.

문제의 구현요소는 필수 요소와 추가 요소로 구분되어 있는데 이를 통하여 작성한 Usecase 다이어그램을 살펴보면 다음과 같다.



2. Related works

2-1. 프로젝트 주제와 연관된 기반 이론

C언어 편집기를 제작을 위해서는 C언어의 문법 규칙을 잘 이해해야할 필요가있다. 사용자가 입력한 값을 C언어 문법 규칙을 활용해서 해당 문법이 정확하다면 색을 지정해주어야한다. 그래서 우리는 C언어 문법 규칙에 대해서 공부를 하였으며 이 부분에서는 C언어 문법규칙에 대한 이론을 주로 다루겠다.

먼저 변수의 이름 생성 규칙이다.

1. 변수의 이름은 영문자(대소문자), 숫자, 언더스코어(_)로만 구성된다.
2. 변수의 이름은 숫자로 시작될 수 없다.
3. 변수의 이름 사이에는 공백을 포함할 수 없다.
4. 변수의 이름으로 C언어에서 미리 정의된 키워드(keyword)는 사용할 수 없다.

아래는 이해를 도와줄 사진이다.

```
int main()
{
    int !song = 0;
    int 123song = 0;
    int if = 0;
}
```

또한 변수의 선언에서도 지켜주어야할 규칙이있다. C언어에서는 변수를 사용하기 전에 반드시 먼저 해당 변수를 저장하기 위한 메모리 공간을 할당받아야 한다. 이렇게 해당 변수만을 위한 메모리 공간을 할당받는 행위를 변수의 선언이라고 부른다. 만약 선언되지 않은 변수를 사용하려고 하면, C 컴파일러는 오류를 발생시킨다. 즉 변수를 선언한 이후부터 변수가 사용가능하다는 것이다. 아래는 이해를 도와줄 예시이다.

```
어벤디_고급레벨_테스트용
1  #include<stdio.h>
2  int main()
3  {
4      song;
5      int song = 0;
6      song;
7  }
```

심볼릭 상수는 #define을 이용하여 선언할 수 있다. 아래는 이해를 도와줄 사진이다.

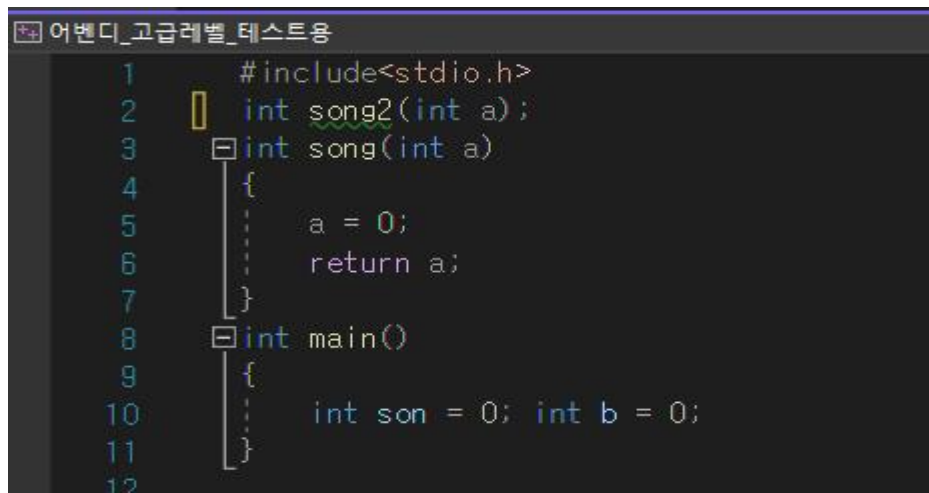
```
#define MAX 10; // #define 선행처리 지시자를 이용한 매크로 심볼릭 상수
```

심볼릭 상수는 변수와 마찬가지로 이름을 가지고 있는 상수이다. 심볼릭 상수 또한 앞에서 기술한 대로 이름 생성 규칙을 지켜야한다. 아래는 이해를 도와줄 사진이다.

```
#define !song = 0;
#define 123song = 0;
#define !.f= 0;
```

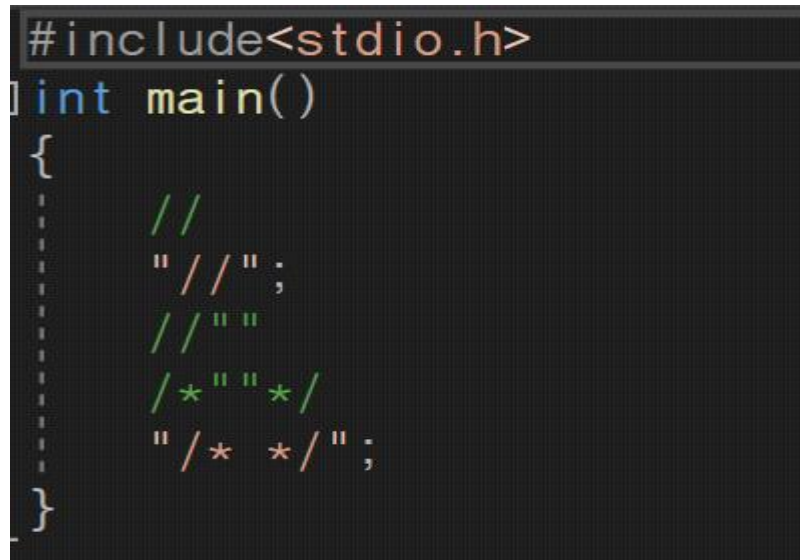
하지만 실제 테스트해보니 사진에서 보는것과 같이 심볼릭 상수는 특별하게 키워드로 이름을 지정해도 오류가 출력되지않는 점을 알 수 있다.

위에서 기술한 이론의 정보만으로는 구현을 하면서 애매한 부분들이 많았다. 그래서 잘 모르겠고 애매한 부분들은 사진과 같이 직접 비주얼 스튜디오를 실행하여 테스트를 하였다. 테스트를 하여 알게된 이론들을 설명해주겠다.



```
1 #include<stdio.h>
2 int song2(int a);
3 int song(int a)
4 {
5     /*
6      *
7      */
8     a = 0;
9     return a;
10 }
11 int main()
12 {
13     /*
14      *
15      */
16     int son = 0; int b = 0;
```

위 사진에서 보는 것과같이 함수의 선언에서는 ;이 필요하나 함수의 정의부분에서는 ;이 필요없는 것을 알 수 있다. 또한 세미콜론을 이용하여 한라인에서도 부분을 나눌 수 있다.



```
#include<stdio.h>
int main()
{
    /*
     *
     */
    int son = 0; int b = 0;
```

위에 사진을 보면 알 수 있듯이 괄호의 경우 뒤에 주석과 따옴표가 나와도 상관없다. 하지만 주석의 경우 뒤에 따옴표가 나오면 주석처리해준다. 또한 따옴표의 경우 뒤에 주석이 나오면 문자열처리해준다. 우리는 사용자가 입력한 괄호, 주석, 따옴표 순서를 고려하여 색을 지정해줄 필요가있다. 이 이 밖에도 C언어에는 많은 자잘한 규칙들이 존재한다. 해당 규칙들을 파악하여 색상 지정을 해야 한다.

2-2. 구현을 위해 필요한 주요 데이터

C언어 편집기를 제작을 위해서는 색깔을 지정해줄 필요가있는데 색깔지정을 위해서는 특별한 취급을 받는 이름을 미리 저장해놓고 처리해야한다. 그래서 우리는 MVC패턴에맞게 해당 데이터를 모델에 저장할 예정이며 이 부분에서는 모델에 관한 이야기를 주로 하겠다.

키워드는 고유한 의미를 가지는 예약어이다. 그래서 우리는 C언어에서 사용되는 키워드들의 데이터가 필요하다. 해당 키워드들의 정보는 검색하여 찾은 뒤 직접 입력해주었다. 일단 C언어에서는 32개의 키워드가 기본적으로 있다. 아래 사진이 32개의 키워드이다. 해당 키워드는 미국 표준 협회 ANSI에서 지정한 키워드라고한다.

auto	beak	case	char	const	continue	default
do	double	else	enum	extern	float	for
goto	if	int	long	register	return	short
signed	sizeof	static	struct	switch	typedef	union
unsigend	void	volatile	while			

하지만 우리는 사용자의 편의성을 위해서 색을 변경해줬으면 좋겠는 단어들을 추가로 키워드모델의 데이터로 지정하였다. 우리가 지정한 키워드는 다음과 같다.

```
Keyword = {"#include", "#define", "struct", "union", "enum", "for", "while", "return", "false", "true", "sizeof", "typedef", "#pragma", "do", "static", "default", "const", "switch", "case", "break", "continue", "if", "else", "goto", "register", "extern", "volatile"};
```

우리가 추가로 지정한 키워드 데이터는 "false", "true", "#include", "#define", "#pragma"가 있다. 해당 데이터들도 색을 변경해주면 좋겠다싶어서 추가로 넣어주었다. 이러한 데이터들을 담은 곳이 바로 KeywordModel이다.

또한 현재 우리가 구현한 키워드 모델에서는 데이터타입 즉 기본타입들이 전부 빠져있는 것을 볼 수 있다. 기본 타입이란 해당 데이터가 메모리에 어떻게 저장되고, 프로그램에서 어떻게 처리되어야 하는지를 명시적으로 알려주는 역할을 한다. 따라서 C언어는 여러 형태의 타입을 미리 작성하여 제공하고 있는데, 이것을 기본 타입이라고 한다. 이러한 기본 타입은 크게 정수형, 실수형, 그리고 문자형 타입으로 나눌 수 있다. 이러한 기본타입은 식별자 및 함수의 처리의 편의성을 위해서 DatatypeModel에 지정해주었다. DatatypeModel에는 데이터타입 즉 기본타입들의 이름들을 담고 있다. 아래는 이해를 도와줄 실제 구현 코드이다.

```
datatype = {"int", "float", "char", "long", "short", "unsigned", "bool", "double", "signed", "void"};
```

해당 데이터들은 키워드와는 구분하는 것이 구현에 있어서 편하여 분리하였다.

그리고 헤더파일의 이름들 또한 필요하다. 해당 데이터는 검색으로는 찾기 힘들어서 Visual Studio를 실행하여서 C언어 헤더파일의 위치를 담고있는 폴더의 위치를 찾아주었다. C언어 헤더파일의 위치는 다음과 같다.

C:\Program Files (x86)\Windows Kits\10\Include\10.0.19041.0\um\AccCtrl.h

내 PC > Windows (C:) > Program Files (x86) > Windows Kits > 10 > Include > 10.0.19041.0 >

이름	수정한 날짜	유형	크기
cppwint	2022-09-24 오후 8:32	파일 폴더	
shared	2022-09-24 오후 8:33	파일 폴더	
ucrt	2022-09-24 오후 8:32	파일 폴더	
um	2022-09-24 오후 8:33	파일 폴더	
wint	2022-09-24 오후 8:33	파일 폴더	

해당폴더전체의 파일명들을 다 추출해내니 너무 많았다. 그래서 우리가 주로 사용하는 헤더파일들이 담겨져있는 ucrt폴더의 파일명들만 파이썬을 활용하여 추출했다. 아래는 파이썬으로 추출해낸 ucrt폴더의 파일들이다.

```
C:\Users\md8\Desktop\코딩> cd C:\Users\md8\Desktop\코딩 ; & C:\Users\md8\AppData\Local\Programs\Python\Python311\python.exe C:\Users\md8\.vscode\extensions\ms-python.python-2022.18.2\python-files-lib\python-debugpy-adaptor\...\debugpy\launcher '2647' '-' 'c:\Users\md8\Desktop\코딩\여벤디_고급레벨_헤더파일명.py'
['<assert.h>', '<complex.h>', '<conio.h>', '<corecrt.h>', '<corecrt_io.h>', '<corecrt_malloc.h>', '<corecrt_math.h>', '<corecrt_math_defines.h>', '<corecrt_memcpy_s.h>', '<corecrt_memory.h>', '<corecrt_search.h>', '<corecrt_share.h>', '<corecrt_startup.h>', '<corecrt_stdio_config.h>', '<corecrt_terminate.h>', '<corecrt_wconio.h>', '<corecrt_wctype.h>', '<corecrt_wdirect.h>', '<corecrt_wio.h>', '<corecrt_wprocess.h>', '<corecrt_wstdio.h>', '<corecrt_wstdlib.h>', '<corecrt_wstring.h>', '<corecrt_wtime.h>', '<crtdbg.h>', '<ctype.h>', '<direct.h>', '<dos.h>', '<errno.h>', '<fcntl.h>', '<fcntl.h>', '<float.h>', '<fpieee.h>', '<inttypes.h>', '<io.h>', '<locale.h>', '<malloc.h>', '<math.h>', '<mbctype.h>', '<mbstring.h>', '<memory.h>', '<minmax.h>', '<new.h>', '<process.h>', '<safeint.h>', '<safeint_internal.h>', '<search.h>', '<share.h>', '<signal.h>', '<stddef.h>', '<stdio.h>', '<stdlib.h>', '<string.h>', '<tchar.h>', '<tgmath.h>', '<time.h>', '<uchar.h>', '<wchar.h>', '<wctype.h>', '<locking.h>', '<stat.h>', '<timeb.h>', '<types.h>', '<utime.h>']
```

그래서 우리는 헤더파일데이터들을 지정할 수 있었다. 우리가 지정한 헤더파일데이터는 다음과 같다.

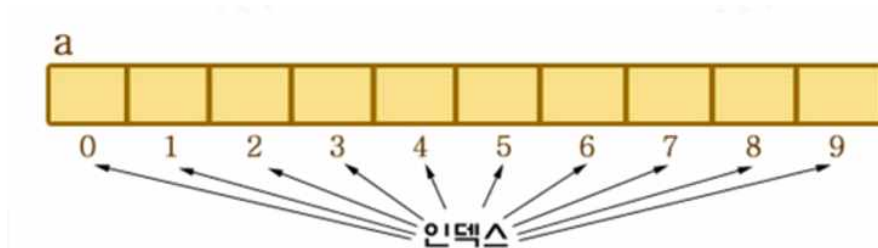
```
Header = { "<assert.h>", "<complex.h>", "<conio.h>", "<corecrt.h>", "<corecrt_io.h>", "<corecrt_malloc.h>", "<corecrt_math.h>", "<corecrt_math_defines.h>", "<corecrt_memcpy_s.h>", "<corecrt_memory.h>", "<corecrt_search.h>", "<corecrt_share.h>", "<corecrt_startup.h>", "<corecrt_stdio_config.h>", "<corecrt_terminate.h>", "<corecrt_wconio.h>", "<corecrt_wctype.h>", "<corecrt_wdirect.h>", "<corecrt_wio.h>", "<corecrt_wprocess.h>", "<corecrt_wstdio.h>", "<corecrt_wstdlib.h>", "<corecrt_wstring.h>", "<corecrt_wtime.h>", "<crtdbg.h>", "<ctype.h>", "<direct.h>", "<dos.h>", "<errno.h>", "<fcntl.h>", "<fcntl.h>", "<float.h>", "<fpieee.h>", "<inttypes.h>", "<io.h>", "<locale.h>", "<malloc.h>", "<math.h>", "<mbctype.h>", "<mbstring.h>", "<memory.h>", "<minmax.h>", "<new.h>", "<process.h>", "<safeint.h>", "<safeint_internal.h>", "<search.h>", "<share.h>", "<signal.h>", "<stddef.h>", "<stdio.h>", "<stdlib.h>", "<string.h>", "<tchar.h>", "<tgmath.h>", "<time.h>", "<uchar.h>", "<wchar.h>", "<wctype.h>", "<locking.h>", "<stat.h>", "<timeb.h>", "<types.h>", "<utime.h>" }
```

해당 데이터들을 담은 곳이 바로 HeaderModel이다.

2-3. 사용할 주요 자료구조 설계

-배열

우리는 사용자가 입력한 텍스트를 들고와서 C언어 문법에 맞게 색을 지정해주어야한다. 그래서 사용자가 입력한 데이터를 들고와 색 지정을 위해 문자열을 문자형으로 분리하여 저장하여야한다. 그래서 우리는 필요한 자료구조로 배열을 선정했다.



배열은 연속된 메모리 공간에 순차적으로 저장된 데이터 모음이다. 우리가 배열을 선정한 이유는 첫 번째로는 자바 toCharArray() 메소드의 존재 때문이다. 우리는 사용자가 입력한 문자열을 문자형으로 분리하여 저장한 뒤 각각의 문자형 데이터들을 검사하고 색 지정을 해주어야한다. 그래서 다른 어려운 자료구조를 사용하지 않고 자바 toCharArray() 메소드를 이용하여 문자열을 문자형 배열로 바꾸준다면 오류가 발생할 일도 없고 안전할 것이라고 생각했다. 또한 두 번째로는 우리는 사용자가 입력한 문자열값만 들고오므로 데이터의 크기가 정해져 있다. 그래서 데이터의 추가적인 삽입이 필요가 없어 데이터의 크기를 바꿀 일이 없다. 그래서 다른 자료구조를 이용하지 않아도 되고 배열이 효율적이라고 판단했다.

우리는 앞서 설명한 사용자가 입력한 문자열 값을 문자형으로 분리하여 저장하는 배열말고도 색 지정을 위해 색정보 값들을 저장해주는 String형 배열또한 사용한다. 이 String형 배열을 이용하여 색을 지정하여줄 때 배열의 장점이 나타나는데 바로 실제 메모리 상에서 물리적으로 데이터가 순차적으로 저장되기 때문에 데이터에 순서가 있으며, index가 존재하여 indexing 및 slicing이 가능하기 때문이다.

```
UserTextString = UserText.getText().replaceAll(System.getProperty("line.separator"), replacement: "\n");
```

```
UserTextCharacter = UserTextString.toCharArray(); //주석, 괄호, 따옴표, 세미콜론을 위한 문자형배열로 분리
UserTextCharacterColor = new String[UserTextCharacter.length]; //각각의 색정보들을 저장할 문자열배열
```

위 사진을 우리가 실제 구현한 소스코드 중 일부이다. 이를 이용하여 추가적인 설명을 돕겠다. 사진에서 보는 것과 같이 사용자가 입력한 문자열을 읽어온다. 또한 읽어온 문자열을 이용하여 제일 처음에 설명한 문자열을 문자형으로 분리하여 저장하는 배열을 생성한다. 그리고 그 배열의 크기를 이용하여 색정보 값을 저장하는 배열을 생성한다. 하지만 만약 색정보값을 저장하기 위해 생성한 배열이 배열이 아니라 순서가 정해져있어 순차적으로밖에 읽어올 수밖에 없는 자료구조라고 가정해보겠다. 그렇다면 후에 색을 지정해줄 때 색을 지정해줄 문자열을 위해서 다시 처음부터 자료구조를 읽어야할 것이다. 그래서 굉장히 비효율적인 구조로 색을 지정해줄 것이다.

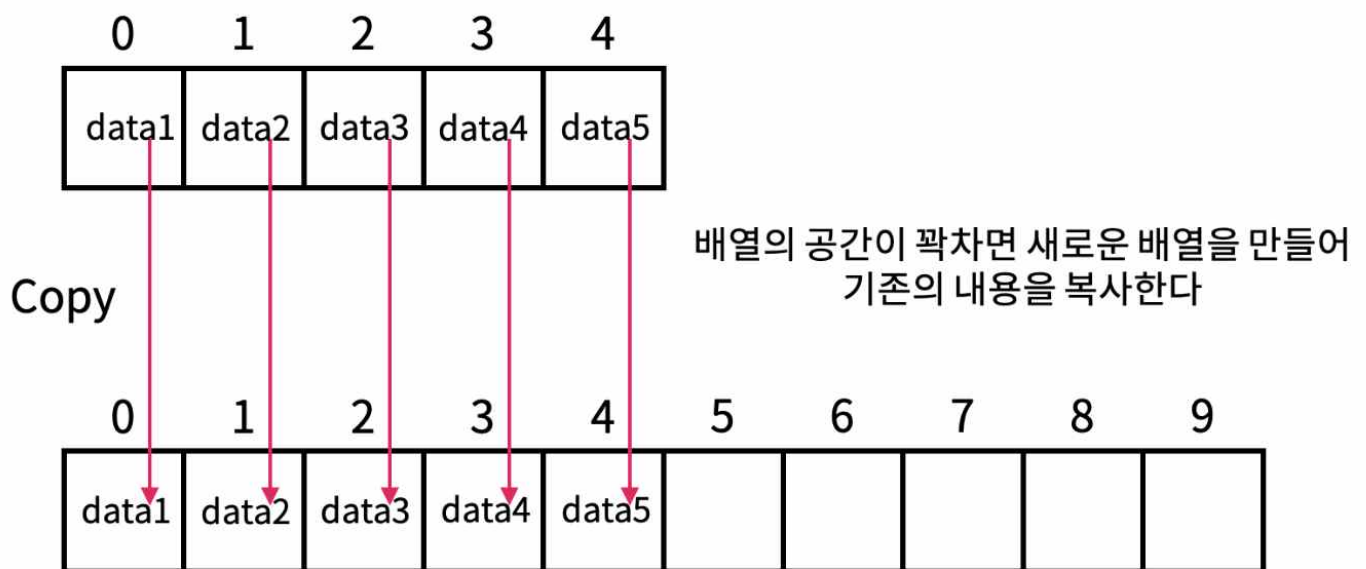
```
for (int SearchKeywordColor = Keywordindex; SearchKeywordColor < Keywordindex + ModelKeywordValue.length(); SearchKeywordColor++) {
    UserTextCharacterColor[SearchKeywordColor] = "CYAN"; //키워드명의 검색위치 ~ 키워드명의 문자열길이까지 색 지정
}
```

하지만 우리는 배열을 사용함으로써 위에 사진과 같이 index를 사용해 특정 요소를 리스트로부터 효율적으로 읽어올 수 있는 indexing 기술과 요소에 특정한 부분을 따로 분리해 조작하는 slicing 기술이 가능한 배열을 이용한다면 효율적으로 구현할 수 있다고 생각했다. 또한 속도에 있어서도 배열은 기본위치 + 오프셋(요소 크기 * 인덱스) 연산으로 모든 요소에 접근 가능하기 때문에 배열의 각 요소에 접근하는 시간은 $O(1)$ 로 모두 동일하다. 그래서 빠르게 색정보를 저장하고싶은 위치로 이동하여 색을 지정하는 기능을 수행할 수 있기 때문에 자료구조로 배열을 선정하였다.

```
UserTextWord = new ArrayList<String>(); //토큰화한 데이터를 담기위한 String형 가변배열
```

또한 우리는 가변배열인 ArrayList를 사용하였다. 앞에서 설명한 배열들은 Array인 일반배열로 구현하였고 공백을 기준으로 토큰화하여 저장하는 배열은 ArrayList인 가변배열을 이용하여 구현하였다.

일반 배열은 앞에서 기술한 것과 같이 사용자가 입력한 문자열을 문자형으로 분리하여 저장할 때 사용하였고 가변배열은 사용자가 입력한 문자열을 공백을 기준으로 분리하여 토큰화하여 저장할 때 사용한다. 추가로 이해를 돕기위해 각각의 배열들이 어디에 사용된지도 기술하자면 일반 배열은 주로 주석, 따옴표, 괄호, 세미콜론의 색 지정을 위해서 사용하고 가변배열은 키워드, 헤더파일명, Define상수, 식별자, 함수의 색 지정을 위해 사용한다. 서로 다른 배열을 이용해 색을 지정하는 이유에대해서는 후에 'Discussion -구현 결과의 특성과 이유'에 기술하겠다. 이 부분에서 왜 가변배열을 사용한지를 주로 다루겠다. 가변배열은 지금부터 ArrayList라고 부르겠다. ArrayList는 자바의 List 인터페이스를 상속받은 여러 클래스 중 하나이다. 일반배열은 크기가 고정되어있지만 ArrayList는 사이즈가 동적인 배열이다. 즉 ArrayList는 크기가 가변적으로 변한다. 아래는 이해를 도와줄 예시사진이다.



위에사진처럼 ArrayList의 공간이 꽉차면 새로운 배열을 만들어 값을 복사하여 저장한다. 그래서 동적인 배열이라는 것이다. 그리고 ArrayList가 값을 삽입하는 방식에 대해서 추가적으로 설명하자면 만약 새로운 값을 배열에 넣어주려고할 때 배열의 크기가 가득차있다면 기존의 용량 + 기존 용량/2 만큼 크기가 늘어난 배열에 기존 값들을 복사해준다. 그리고 새로운 값을 넣어준다고 한다.

우리가 ArrayList를 사용한 이유는 문자열의 공백 기준 토큰화 때문이다. 자바에서는 공백기준으로 토큰화를 하기위해서는 "StringTokenizer"라는 클래스를 이용할 필요가있다. 아래는 이해를 도와줄 사진이다.

```
StringTokenizer userTextTokenizer = new StringTokenizer(UserText.getText()); //공백을 기준으로 토큰화
while (userTextTokenizer.hasMoreTokens()) {
    UserTextWord.add(userTextTokenizer.nextToken()); //키워드, 헤더파일명, 매크로, 식별자, 함수를 위한 공백을 기준으로 분리한 문자열을 가변배열에 저장
}
```

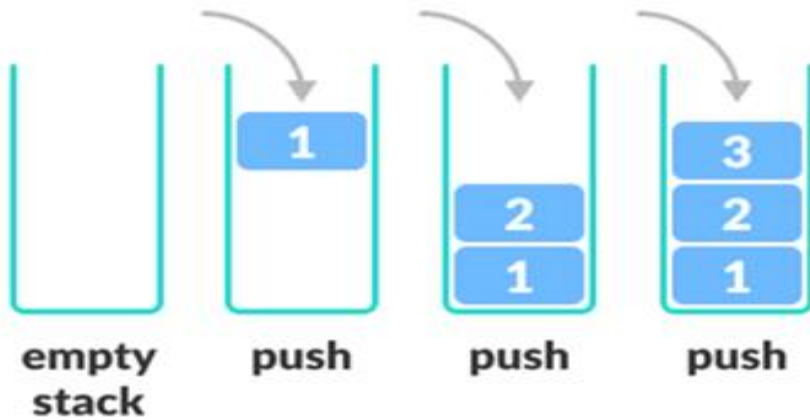
StringTokenizer를 이용하여 토큰을 분리를 한 뒤 while문을 활용해서 다음 단어들을 계속해서 삽입해준다. 참고로 hasMoreTokens()은 다음에 읽어 들일 token이 있으면 true, 없으면 false를 return한다. 그래서 가변배열을 이용하여 구현하였고 가변배열으로 구현하였기 때문에 크기를 계속해서 추가로 늘려주어 오류가 발생할 일도 없다.

그렇다면 의문점이 생길 것이다. 바로 ArrayList을 사용하지않고 연결리스트를 사용하면 되지않냐 라고 말이다. 우리가 연결리스트 즉 LinkedList를 사용하지않고 ArrayList를 사용한 이유는 ArrayList는 무작위 접근이 가능하지만 LinkedList에서는 순차적인 접근만이 가능하기 때문이다. 그래서 LinkedList는 자료를 검색하는 프로그램에서는 적합하지않다. 우리는 앞에서 기술한대로 index를 사용해 특정 요소를 리스트로부터 효율적으로 읽어올 수 있는 indexing 기술과 요소에 특정한 부분을 따로 분리해 조작하는 slicing 기술이 가능해야한다. 아래는 이해를 도와줄 실제구현 코드이다.

```
DefineValue = UserTextWord.get(Wordindex); //해당 단어 저장
```

위 사진처럼 우리는 index를 사용하여 데이터값을 들고와야한다. 하지만 만약 순차적으로 접근해야 한다면 굉장히 비효율적일 것이다. 또한 ArrayList는 n개의 자료를 저장할 때 자료들을 하나의 연속적인 묶음으로 묶어 자료를저장하는 반면 LinkedList는 n개의 자료를 저장할 때 자료들을 저장 공간에 불연속적인 단위로 저장하게 된다. 그렇기 때문에 LinkedList는 메모리 이곳저곳에 산재해 저장되어있는 노드들을 접근하는데에는 ArrayList보다는 긴 지연시간이 소모된다. 배열과 마찬가지로 ArrayList는의 속도에 있어서도 배열은 기본위치 + 오프셋(요소 크기 * 인덱스) 연산으로 모든 요소에 접근 가능하기 때문에 배열의 각 요소에 접근하는 시간은 O(1)로 모두 동일하다. 그래서 우리는 데이터의 효율적인 접근을 위해 LinkedList보다는 ArrayList가 효율적이라고 생각했고 ArrayList를 필요한 자료구조로 선정하였다.

-스택



또한 우리는 앞서 계획 및 설계보고서에 기술한대로 하나의 자료구조를 더 사용해야한다. 바로 Stack이다. 스택은 LIFO(Last In First Out) 구조를 가진다. 제일 마지막에 Push된 데이터가 가장 먼저 Pop된다는 뜻이다. 우리는 Stack을 활용하여 주석, 괄호, 따옴표의 검사를 해주어야한다. 주석, 괄호, 따옴표의 순서를 고려하는 부분은 후에 Discussion에서 기술할 것이고 여기서는 왜 Stack을 사용해야하는지에대해 주로 다루어보겠다. 일단 기본적인 알고리즘은 다음과 같다.

- ① 열린 괄호 및 열린주석 및 열린 따옴표 탐지 시 해당 값을 순서대로 기록
- ② 닫힘 괄호 및 닫힘주석 및 닫힘 따옴표 탐지 시 바로 직전에 탐지된 괄호 또는 주석과 짝이 맞는지 확인 짝이 맞으면 바로 직전에 탐지된 열린 괄호 또는 열린 주석은 더 이상 고려하지 않음
- ③ 만약 짝이 맞지않으면 닫힘 괄호 및 닫힘주석 및 닫힘 따옴표는 오류표시

해당 알고리즘을 보면 입력순서의 역순으로 자료를 참조해야한다. 더 이해하기 쉽게 설명하자면 가장 최근에 등장한 괄호의 짝이 새로 등장한 괄호와 일치해야한다. 즉, 나중에 저장된 데이터를 먼저 꺼내서 확인해야 한다. 그렇기 때문에 제일 마지막에 Push된 데이터가 가장 먼저 Pop되는 스택을 유용하게 사용하여 구현할 수 있다.

앞에서 이야기하였지만 나는 스택을 단순하게 짝을 맞추기 위한 용도로 사용하지 않았다. 주석, 괄호, 따옴표의 순서를 고려하는 것에도 사용하였고 괄호의 위치값을 저장하기위한 용도로도 사용하였다. 이에대해서는 ‘Discussion - 주석, 따옴표, 괄호, 세미콜론 색상지정 (MarkEdit 메소드)’에서 다루었다.

3. Methodology and Implementation

설계 보고서에 모듈 사용 목적, 기능 및 제약 조건, 알고리즘등 을 기술하였음. 보고서의 가독성을 위해 이미 기술한 부분은 생략하겠음.

3-1. 모듈의 알고리즘(suedo code)

모듈의 알고리즘 또한 이미 설계보고서에 기술하였음. 하지만 실제 구현을 하면서 추가로 구현한 기능이 있어 그 기능의 알고리즘을 추가로 기술하겠음.

1. 세미콜론 색상 지정 및 오류 출력

INPUT: 사용자가 키보드로 입력한 텍스트 값

OUTPUT: 세미콜론이 필요하나 없을 경우 오류 출력

```
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
Wn_Index ← Wn의 위치값을 저장하는 변수
semcolonindex ← 한라인에 세미콜론이2개 나왔을 경우 처리를 위한 변수
semcoloncheck ← 한줄에 세미콜론이 있거나 세미콜론이 없어도 되는지 체크 하는 변수

for i←0 to i<char_list_size {
    semcoloncheck ← 0
    if(char_list [i] = 'Wn') {
        for j←Wn_Index to j<i {
            if(char_list[j]가 주석이거나 키워드거나 헤더파일이라면)
                semcoloncheck ← 1
            if(char_list[j]가 함수의 정의부분이라면)
                semcoloncheck ← 1
        if(semcoloncheck == 0) {
            semcolonindex ← Wn_Index
            for j←Wn_Index to j<i {
                if(char_list[j] = ';' ) {
                    file_contents[j]의 색 세미콜론 색으로 지정
                    semcolonindex ← j + 1 } }
            for j←semcolonindex to j<i {
                file_contents[j]의 색 오류로 색으로 지정 }
            Wn_Index ← j + 1 } } }
```

알고리즘 핵심 내용 설명

```
if(char_list [i] = 'Wn') {
    for j←Wn_Index to j<i {
        if(char_list[j]가 주석이거나 키워드거나 헤더파일이라면)
            semcoloncheck ← 1
        if(char_list[j]가 함수의 정의부분이라면)
            semcoloncheck ← 1
```

=> 개행이 나왔을 때 그 한줄을 기준으로 문자 값중에 주석이거나 키워드거나 헤더파일거나 함수의 정의부분이라면 세미콜론이 필요없다고 해주는 코드 (한줄을 결정하는 법: n_Index ~ \n의 인덱스 값)

```

if(semcoloncheck == 0) {
    semcolonindex ← Wn_Index
    for j←Wn_Index to j<i {
        if(char_list[j] = ';' ) {
            file_contents[j]의 색 세미콜론 색으로 지정
            semcolonindex ← j + 1 } }
    for j←semcolonindex to j<i {
        file_contents[j]의 색 오류로 색으로 지정 }

```

=> 만약 위에 세미콜론 검사에서 만족되지 않았다면 세미콜론이 있는지 검사해야함 세미콜론의 위치를 이용하여 세미콜론의 위치+1 로 에러처리 해줄 시작 위치 변경 후 에러처리

```
Wn_Index ← j + 1 } } }
```

=> 다음 한줄을 위해 n_Index 값 변경

3-2. 구현에 활용된 핵심 아이디어

최종보고서이니 기능을 구현한 알고리즘 및 핵심 아이디어들을 간략하게 기술하겠음.

1. C언어 문법 색상 지정 (InputController)

핵심 아이디어

```
public class InputController implements Runnable {
```

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

```

KeywordEdit(); //키워드 색 지정
IncludeEdit(); //헤더파일명 색지정
DefineEdit(); // #Define 상수 색 지정
DataTypeEdit(); //데이터 타입 색 지정 및 식별자, 함수 색 지정
MarkEdit(); //주석, 괄호, 따옴표, 세미콜론 색 지정

```

-각각의 색상지정해주는 메소드를 실행

```

for (int ColorSetindex = 0; ColorSetindex < UserTextCharacter.length; ColorSetindex++) { //색 정보를 저장하고있는 문자열배열을 이용하여 색 지정하기
    if (UserTextCharacterColor[ColorSetindex].equals("CYAN"))
        userTextDocument.setCharacterAttributes(ColorSetindex, length: 1, CyanColor, replace: true);
}

```

-메소드 실행 후 색상 정보를 저장하고있는 문자열 배열을 이용하여 색 지정

-키워드 색상지정 (KeywordEdit 메소드)

핵심 알고리즘

```

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 } } }

```

핵심 아이디어

```
getKeyword = KeyWord.getKeyword();
```

-키워드의 데이터는 Model에서 받아온다.

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

-헤더파일 색상지정 (IncludeEdit 메소드)

핵심 알고리즘

```
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 } } }
```

핵심 아이디어

-#include 뒤에 헤더파일이 나오는 점을 이용한다.

```
getHeader = Header.getHeader();
```

-헤더파일의 데이터는 Model에서 받아온다.

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

-#define 상수 색상지정 (DefineEdit 메소드)

핵심 알고리즘

```
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 } } }
```

핵심 아이디어

- #define 뒤에 상수명이 나오는 점을 이용한다.

- 사용자가 입력한 단어가 저장한 상수명과 같다면 색상을 변경해준다. 이 부분은 순차적으로 구현하여 상수명의 선언 이후에만 색상지정을 해주게 구현하였다.

-데이터타입 및 식별자명, 함수명 색상지정 (DataTypeEdit 메소드)

핵심 알고리즘

```
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 } } }
```

핵심 아이디어

```
getDatatype = Datatype.getDatatype();
```

- 기본타입 즉 데이터타입의 데이터는 Model에서 받아온다.

- 기본타입 뒤에 식별자명과 함수명이 나오는 점을 이용한다.

- 사용자가 입력한 단어가 저장한 식별자명 또는 함수명과 같다면 색상을 변경해준다. 이 부분은 순차적으로 구현하여 식별자명 또는 함수명의 선언 이후에만 색상지정을 해주게 구현하였다.

-주석, 따옴표, 괄호, 세미콜론 색상지정 (MarkEdit 메소드)

핵심 알고리즘은 가독성을 위해 생략하겠음. 주석, 따옴표, 괄호의 알고리즘은 계획 및 설계 보고서에 기술하였고 세미콜론 알고리즘은 위에 모듈의 알고리즘에서 기술하였음.

핵심 아이디어

- ① 열린 괄호 및 열린주석 및 열린 따옴표 탐지 시 해당 값을 순서대로 기록
 - ② 닫힘 괄호 및 닫힘주석 및 닫힘 따옴표 탐지 시 바로 직전에 탐지된 괄호 또는 주석과 짝이 맞는지 확인 짝이 맞으면 바로 직전에 탐지된 열린 괄호 또는 열린 주석은 더 이상 고려하지 않음
 - ③ 만약 짝이 맞지않으면 닫힘 괄호 및 닫힘주석 및 닫힘 따옴표는 오류표시
- 스택을 이용하여 괄호, 따옴표, 주석의 순서를 고려하여 색상 지정을 해준다. (후에 Discussion에서 자세하게 기술하겠음)
 - 스택을 이용하여 괄호의 위치를 저장하고 해당 위치를 이용하여 색상지정을 해준다. (후에 Discussion에서 자세하게 기술하겠음)

2. C언어 문법 색상 지정을 위해 사용할 데이터 (Model)

모델의 종류는 키워드, 헤더, 데이터타입 3종류로 구성되지만 동일한 형태이기 때문에 한번만 설명하겠음.

DataType모델 구현코드

```
1 usage
private final String[] datatype = {"int", "float", "char", "long", "short", "unsigned", "bool", "double", "signed", "void"};

1 usage 1 joon6093
public String[] getDatatype() { return datatype; }
```

InputController에서 사용하는 코드

```
import Model.DatatypeModel;

getDatatype = Datatype.getDatatype();
```

핵심 아이디어

- Related works -구현을 위해 필요한 주요데이터 부분에서 각각의 데이터를 뽑아온 아이디어를 기술하였음.
- InputController에 해당 class를 import하여 Model의 메소드를 사용함으로써 데이터 접근

3. 파일 관련 모듈(FileController)

-파일 불러오기

핵심 알고리즘:

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

핵심 아이디어:

파일의 내용이 끝날 때 까지 계속 불러와서 저장한다.

-파일 저장하기

핵심 알고리즘:

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

핵심 아이디어:

파일의 내용을 변수에 담은 뒤 해당 변수를 활용하여 저장한다.

-파일 자동저장

핵심 알고리즘:

```
sleep(일정시간)  
파일저장기능(path)
```

핵심 아이디어:

- 쓰레드로 구현
- 파일 저장이 완료된 후에만 실행 하여야 함.

4. 편집 관련 모듈(EditController)

핵심 아이디어:

```
f(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

핵심 알고리즘:

```
if( file_contents에 data가 포함되어있다면 ) {  
    for i←data_index to i<data_index + data_length {  
        file_contents[i] 하이라이트 }  
}
```

핵심 아이디어:

사용자가 검색한 단어가 포함되어있다면 찾아서 하이라이트 해준다.

-다중 검색 기능

핵심 알고리즘:

```
while(1) {  
    if( data_index >= data_last_index ) {  
        break } }  
}
```

핵심 아이디어:

무한반복을 이용해서 사용자가 검색한 단어를 모두 하이라이트하고 마지막 단어라면 탈출한다.

-단일 바꾸기 기능

핵심 알고리즘:

file_contents에서 change_data 해당하는 첫 번째 값 변경

핵심 아이디어:

사용자가 검색기능을 수행한 후 바꾸기 기능을 수행하면 하이라이트 된 단어를 바꾸어 준다.

-다중 바꾸기 기능

핵심 알고리즘:

ifile_contents에서 change_data 해당하는 모든 값 변경

핵심 아이디어:

사용자가 다중검색기능을 수행한 후 다중바꾸기 기능을 수행하면 하이라이트 된 단어를 모두 바꾸어 준다.

3-3. 프로그램 전체 아키텍처 및 설계도

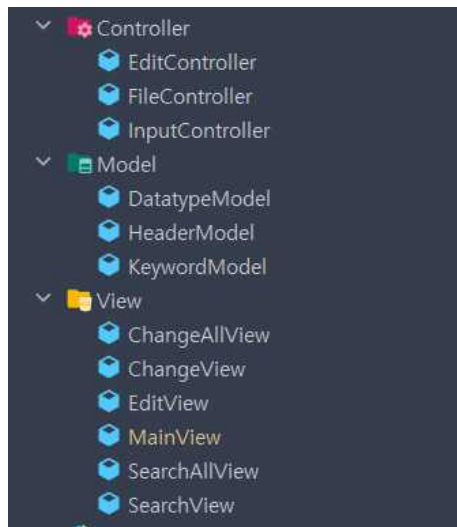
-클래스 다이어그램



우리가 제작한 프로그램에 대한 최종적인 클래스 다이어그램은 다음과 같다. MVC 패턴을 활용하여 프로젝트 설계를 진행하였으며 구현 단계에서 요구 사항이 구체화 됨에 따라 추가된 클래스들이 생겨났다.

우선 Model에서는 더욱 세분화하여 3개의 모델을 사용하기로 하였으며 Controller는 전과 동일하게 3개를 사용하였다. View는 MainView를 포함하여 5개의 View로 늘어났다. 이러한 점을 통해서 확인할 수 있는 MVC패턴의 장점으로 유지보수와 확장에 용이하다는 점이다. 모델-뷰 분리 원칙으로 프로젝트를 제작함으로써 모델과 뷰의 커플링을 줄여서

소스코드의 변경을 최소화할 수 있다. 예를 들어 모델의 개수가 2개에서 3개로 늘어나더라도 뷰에서는 이에 대해 알 필요가 없다. 왜냐하면, 컨트롤러에 의해 분리되어있기 때문이다. 이론적으로만 알고 있던 내용에 대해서 직접 구현을 하며 더욱 와닿게 느낄 수 있었다.

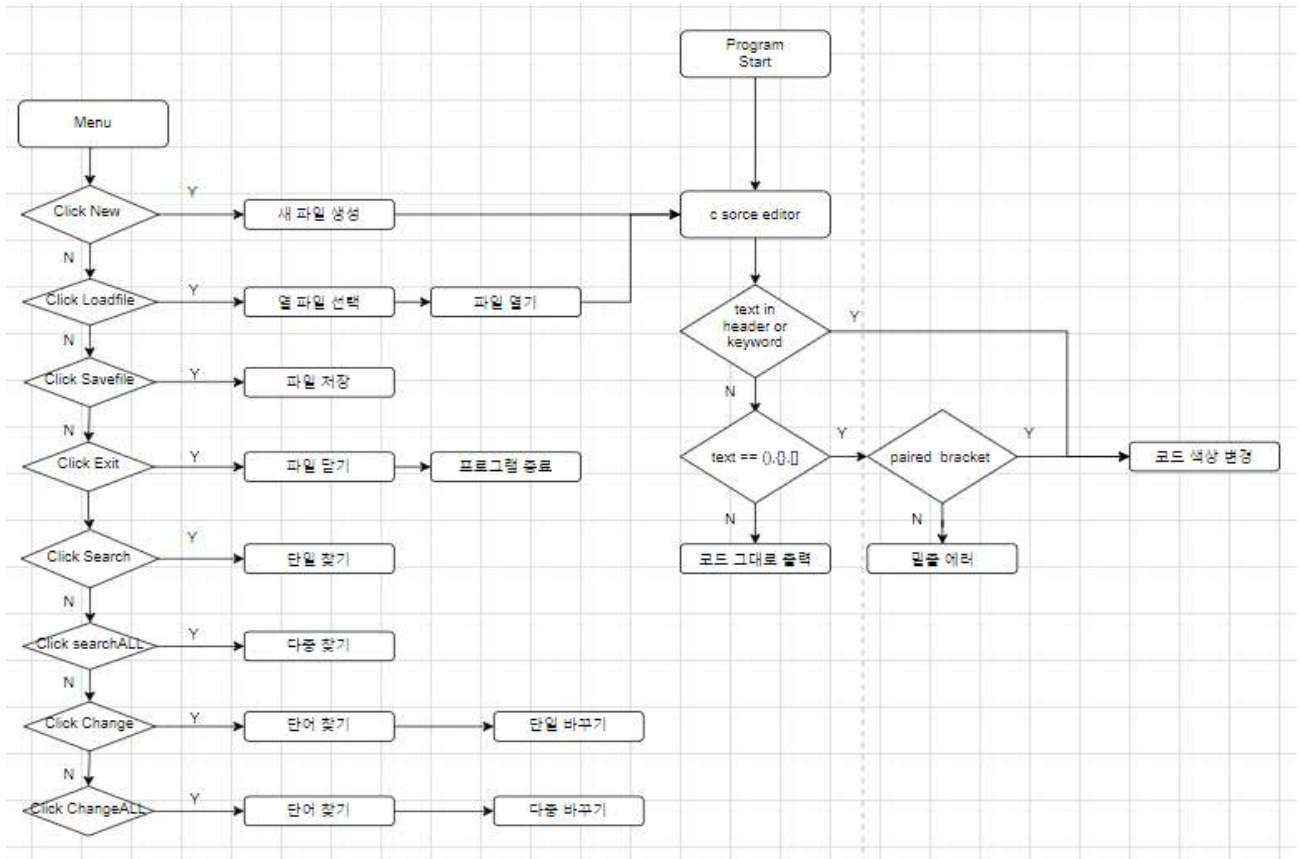


실제 소스코드의 계층을 확인해보면 Model, Controller, View 3개의 계층으로 나누어져 있는 것을 확인해볼 수 있다.

아래는 프로그램의 실행을 나타낸 것이다.

설계보고서 이후 코드 리팩토링 등의 이유로 전체적인 인터페이스의 흐름이 일부 변경되었다. 주요 기능들의 실행흐름도 및 설계도를 첨부하여 프로그램의 핵심 기능들이 어떤 흐름으로 진행되는지 확인할 수 있다.

-인터페이스 플로우 차트



위는 프로그램의 실행흐름을 인터페이스 플로우차트로 나타낸 것이다.

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, EDIT) 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 밀줄 에러를 발생시킨다. 앞서 말한 위 모든 경우가 아닌 경우는 코드를 그대로 출력하여 사용자에게 보여주도록 한다.

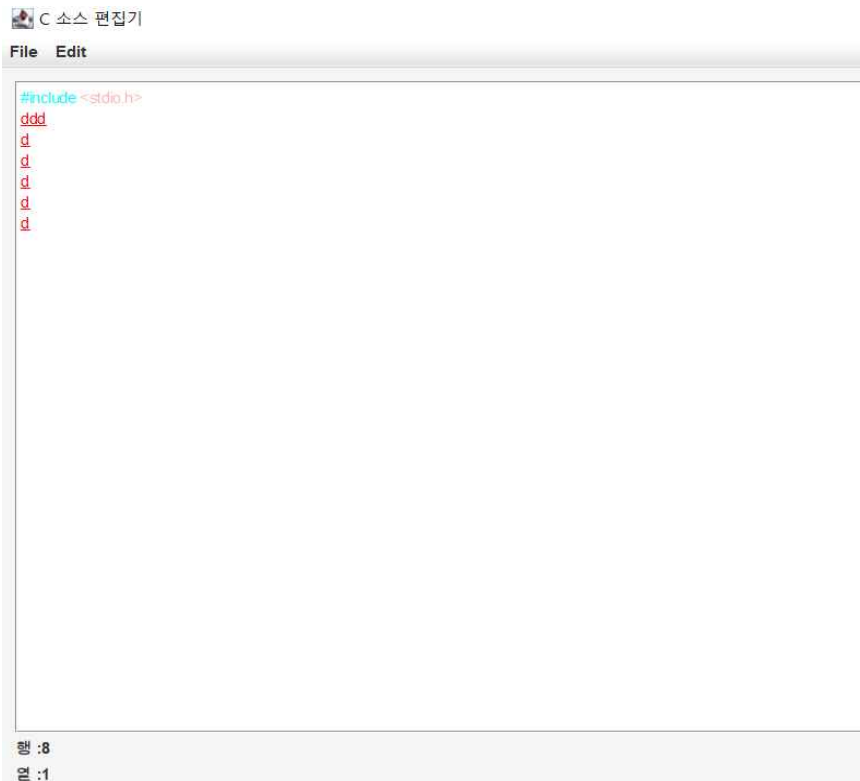
메뉴는 크게 2가지로 나눌 수 있다. 인터페이스 플로우 차트의 Menu의 조건문을 참고하여 보면 첫 번째 메뉴인 파일 메뉴 창은 내부에 새 파일 생성, 파일 불러오기, 파일 저장, 프로그램 종료를 할 수 있는 기능을 가지고 있고, 각 기능의 실행은 클릭을 통해 입력받는다.

이전 설계보고서에 기술하지 못했던 두 번째 메뉴인 편집 메뉴도 내부에 단일 찾기, 다중 찾기, 단일 바꾸기, 다중 바꾸기 기능을 가지고 있고, 각 기능의 실행은 클릭을 통해 입력받게 된다. 이렇게 메뉴에서 실행명령을 입력받으면 각 기능에 따라 메인 뷰인 소스를 입력받는 창을 검사하여 찾기 및 바꾸기 기능을 수행하거나, 파일을 저장 및 열기, 프로그램 종료 등의 기능을 수행하는 식으로 프로그램이 동작하게 된다.

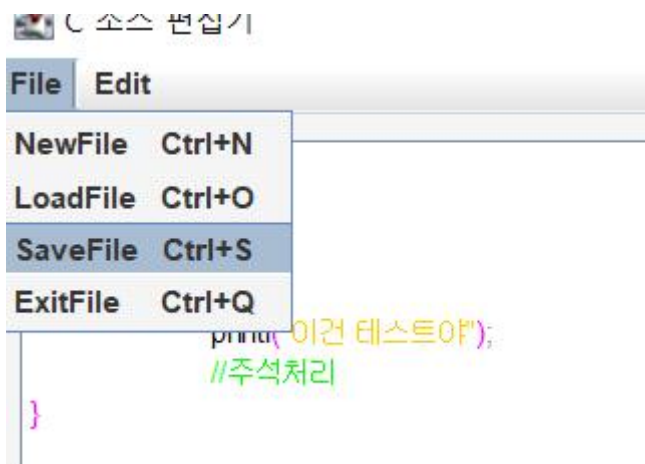
4. Results

4-1. 구현 내용 분석

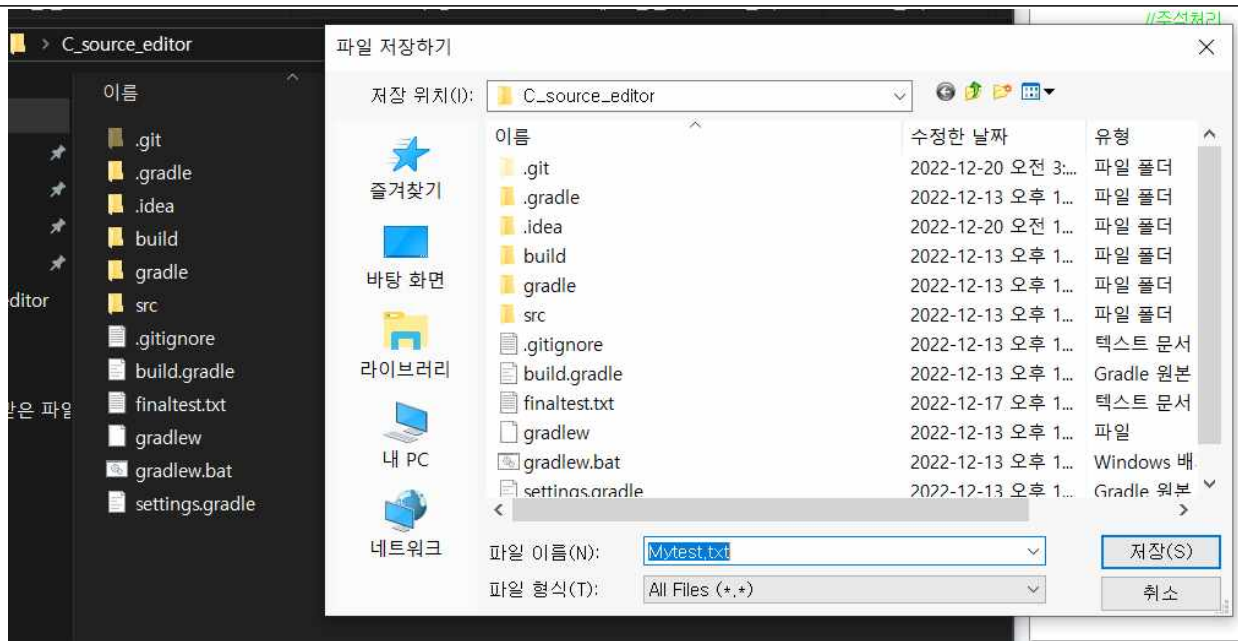
메인 뷰 (MainView)



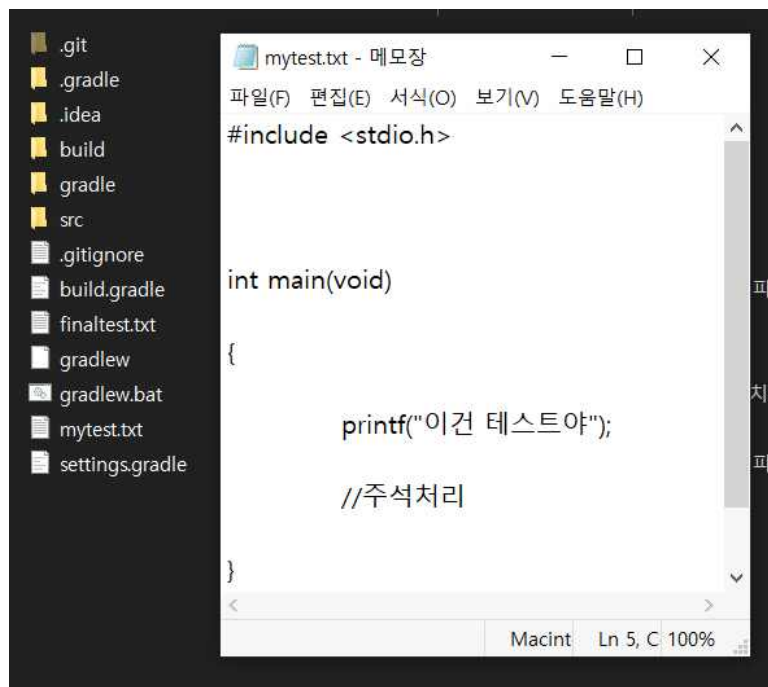
커서의 위치에 따라 행과 열이 정상적으로 잘 출력 되는 것을 확인해볼 수 있다.



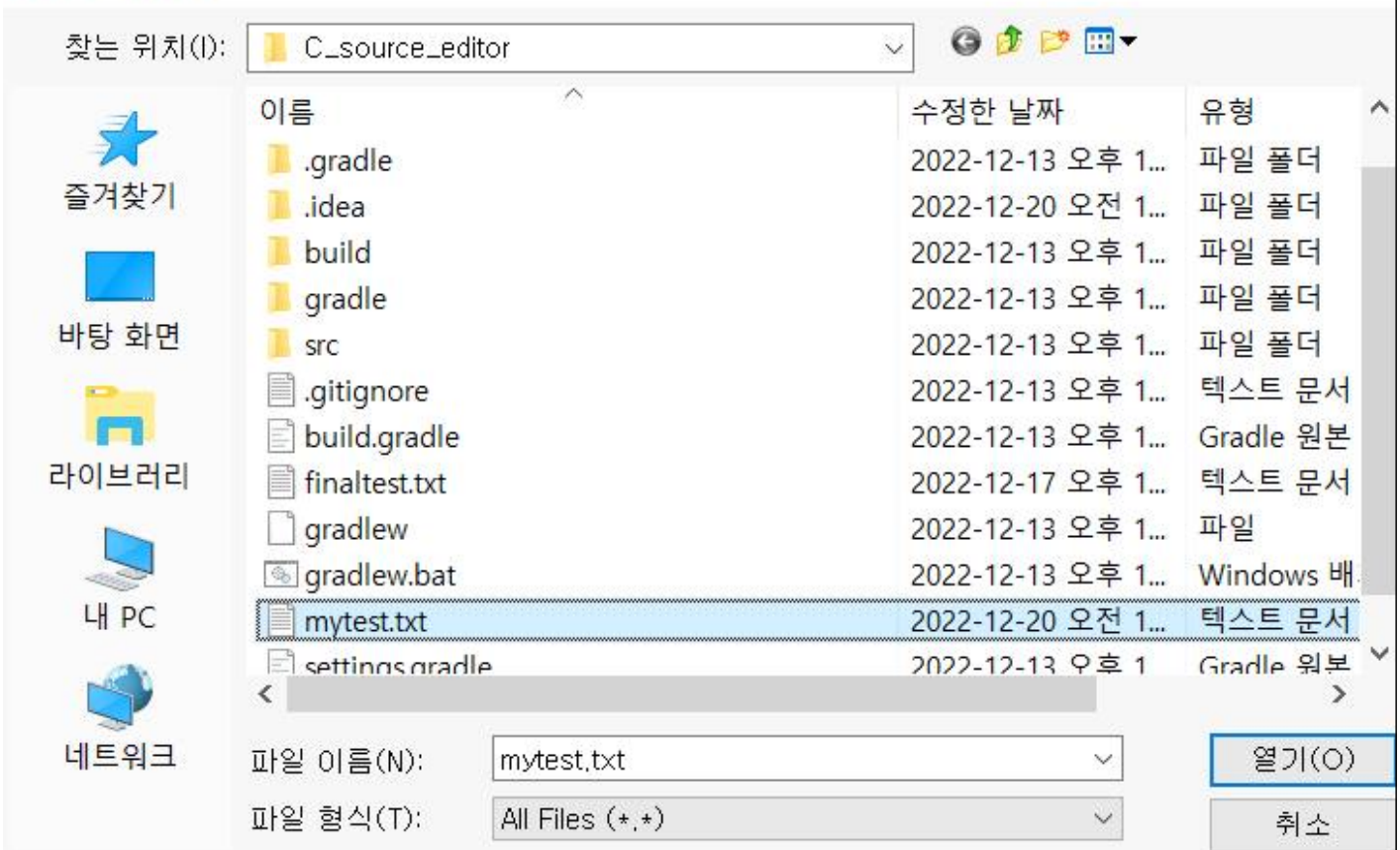
작성한 텍스트에 대해서 SaveFile 메뉴를 선택하거나 Ctrl+S를 누른다면 파일 다이얼로그 창이 뜨며 파일을 저장할 수 있는 창이 나온다.



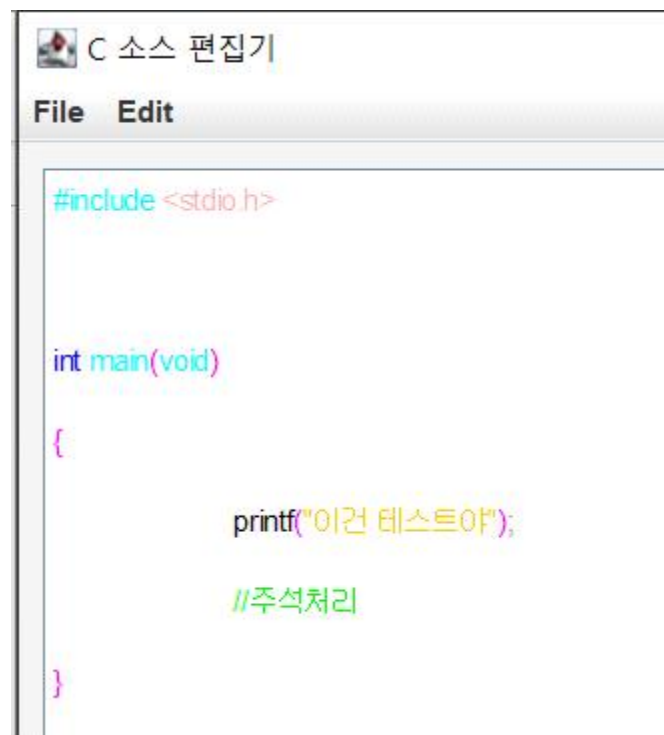
파일을 저장한다면 파일이 정상적으로 저장된다.



파일 불러오기

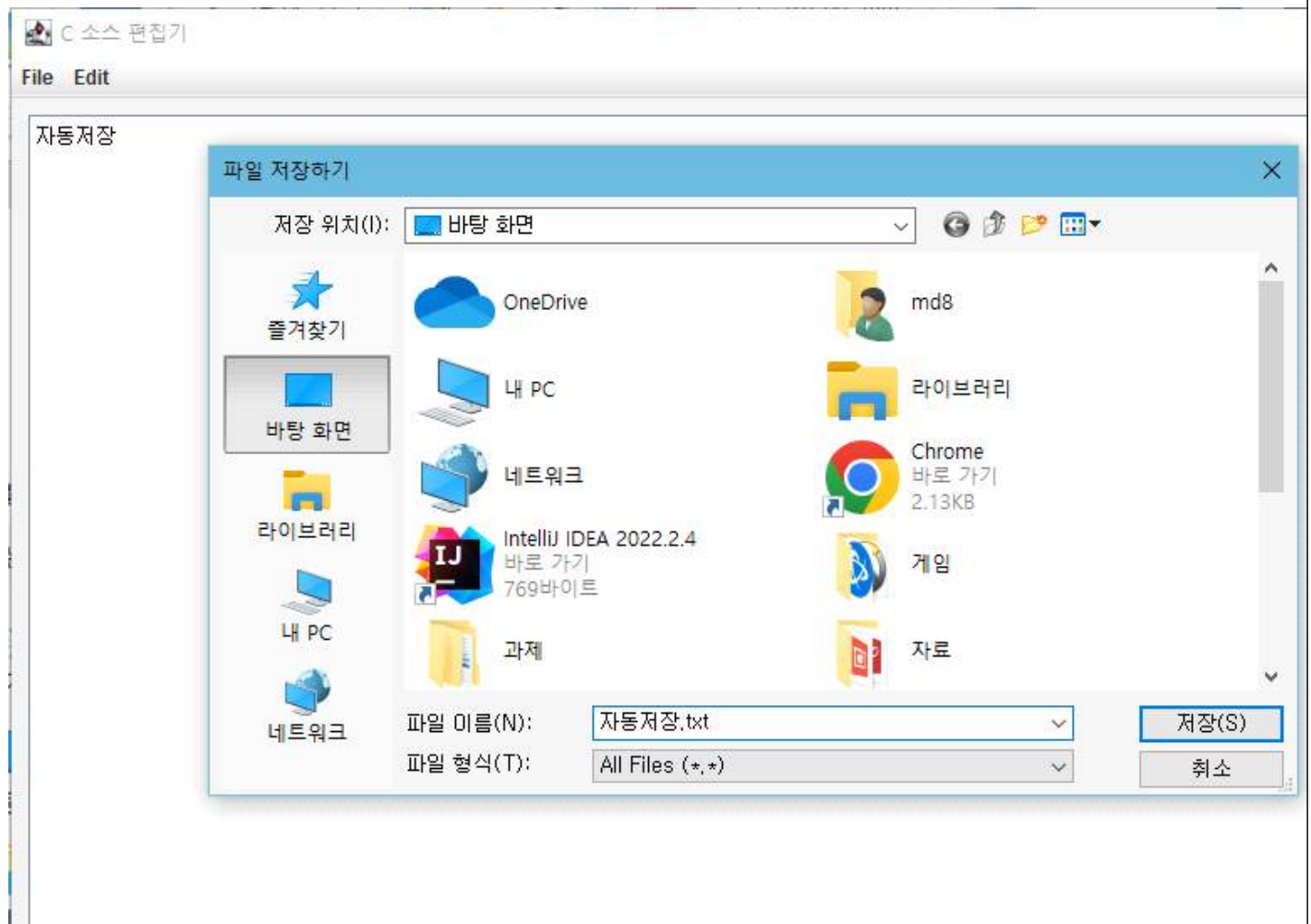


메모장에서 LoadFile을 하면 파일 다이얼로그 창에서 파일을 읽어올 수 있다.

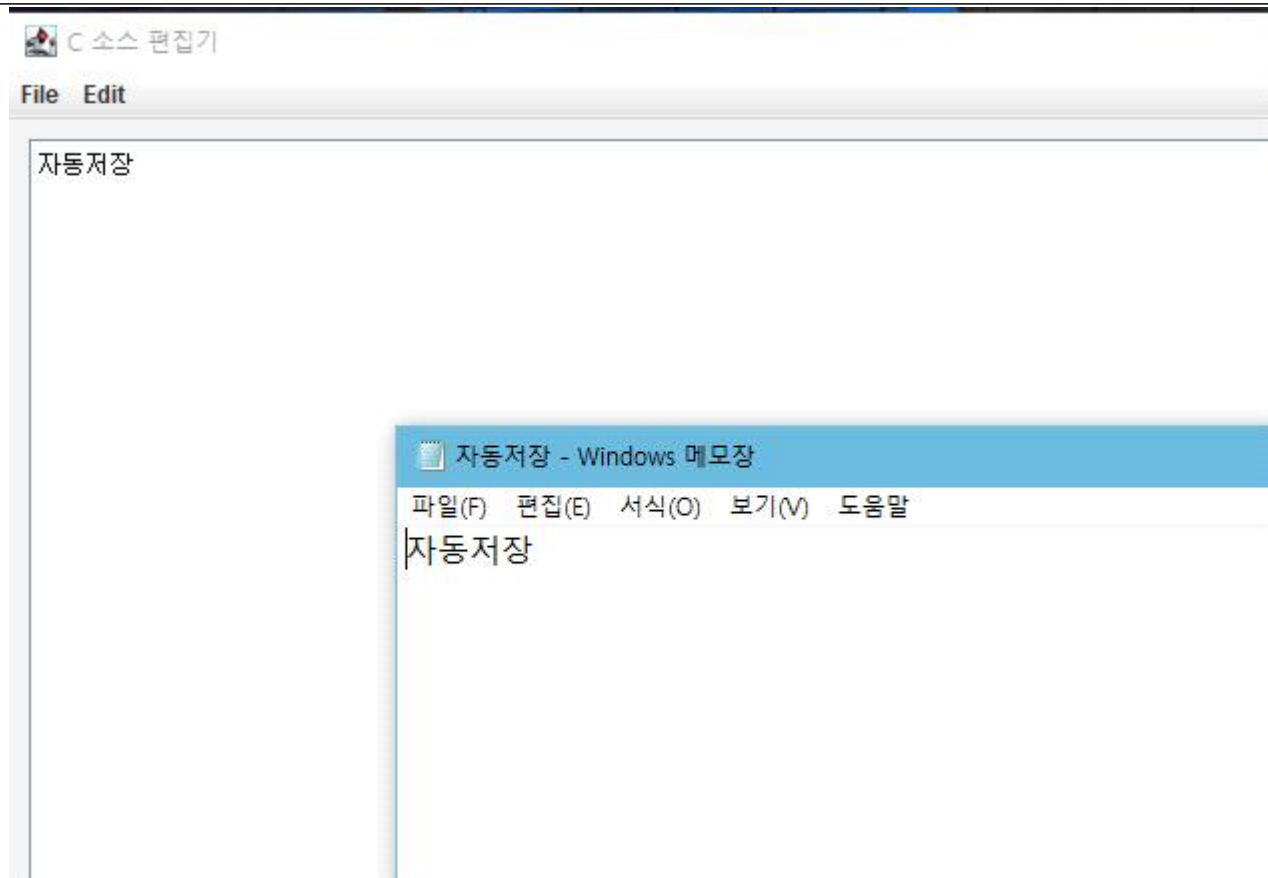


정상적으로 파일을 읽어온 것을 확인할 수 있다.

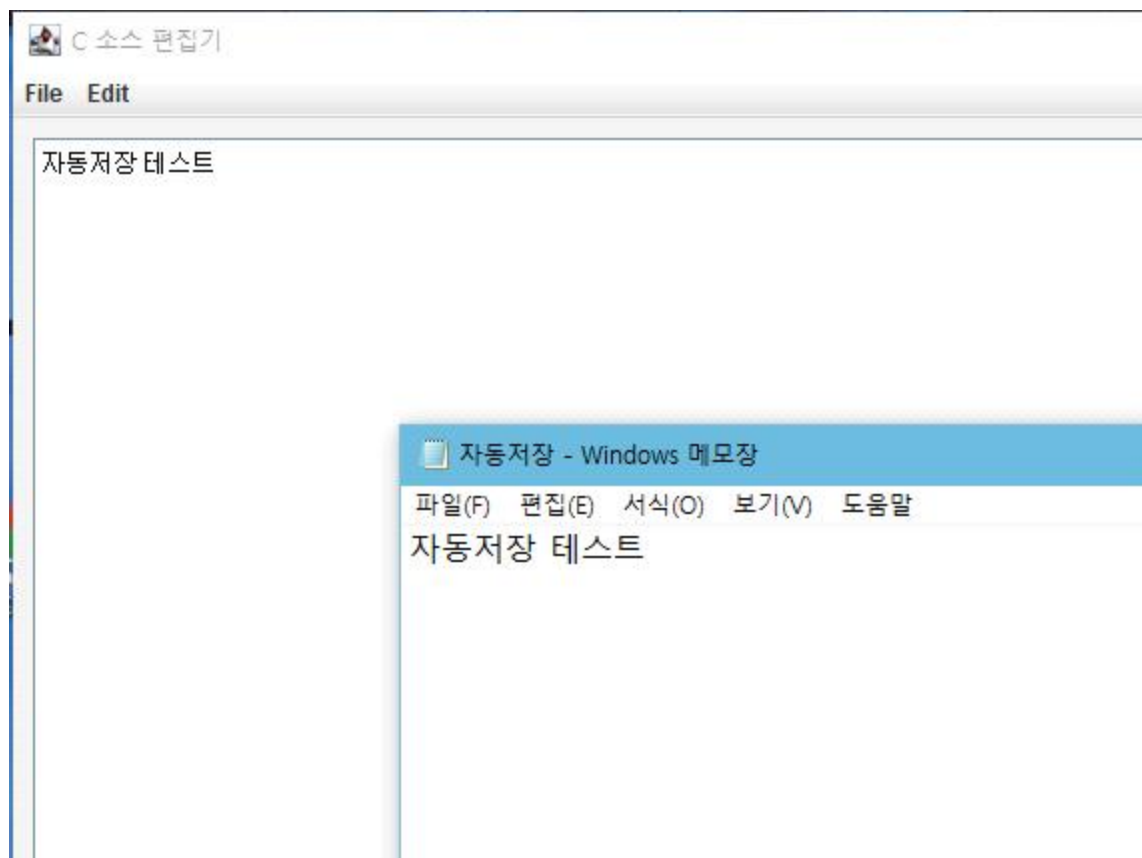
자동저장기능



‘자동저장.txt’라는 이름으로 자동저장기능을 테스트한다.

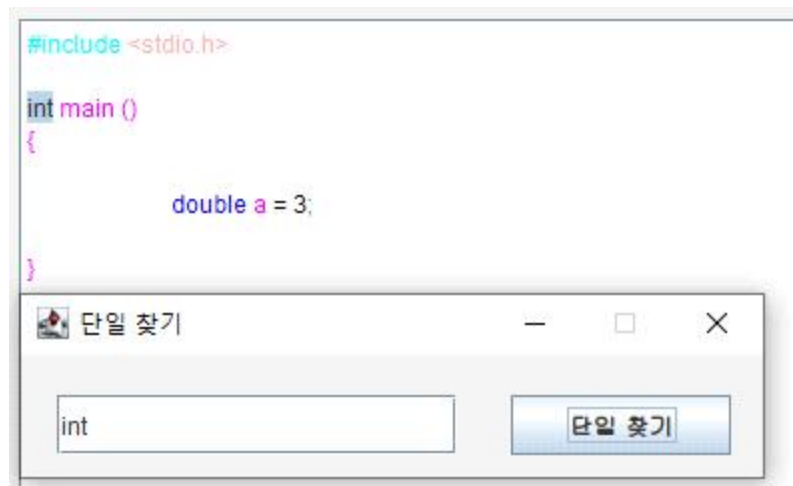


‘자동저장’으로 저장된 모습이다.

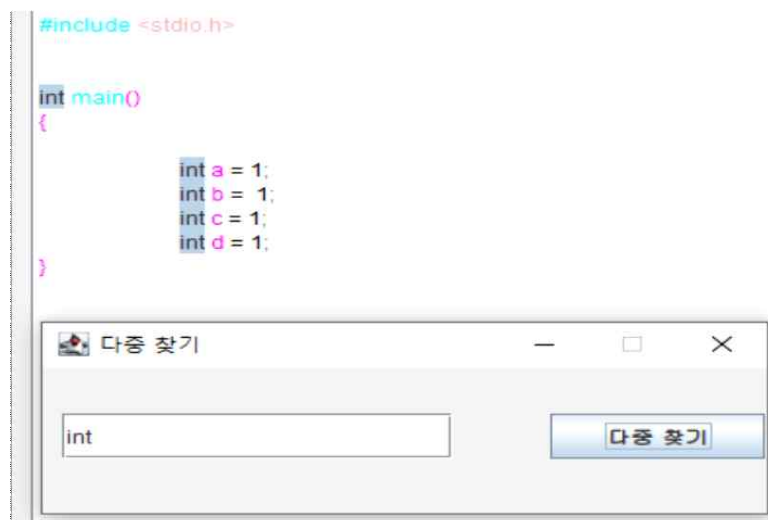


‘자동저장 테스트’로 바꾸고 일정시간을 기다리면 자동으로 ‘자동저장 테스트’라고 저장해주는 모습이다. 자동저장기능이 잘 수행되는 것을 알 수 있다.

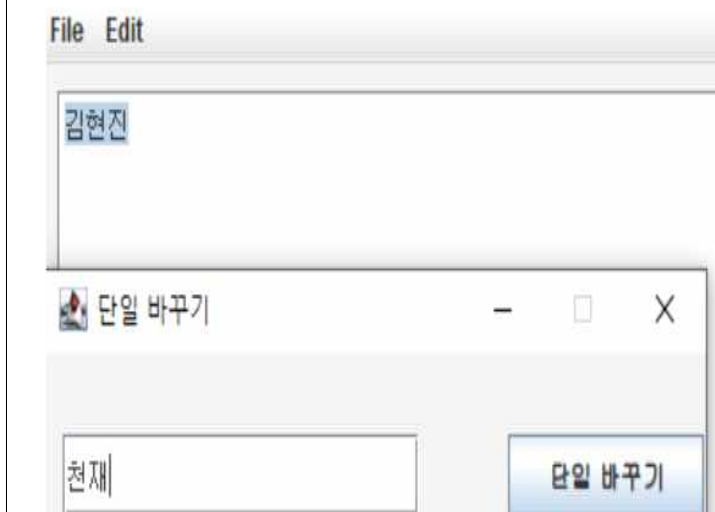
텍스트 찾기 및 바꾸기 (EditController & EditView)

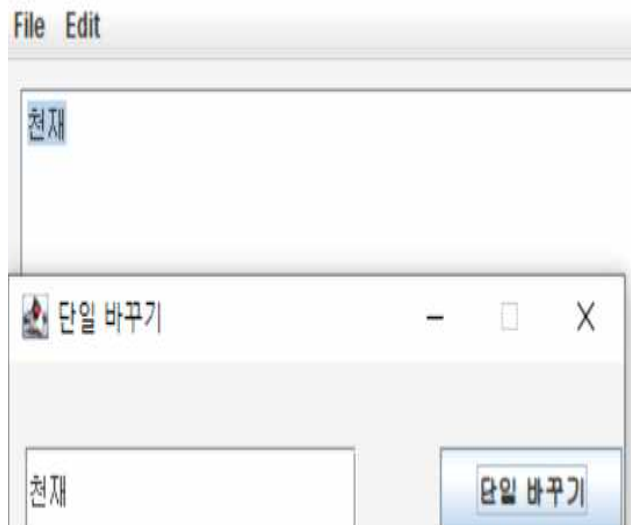


단일 찾기 - 말 그대로 단어를 하나씩 찾아주는 기능이다.



다중 찾기 - 조건이 맞는 모든 단어들을 하이라이팅해준다.





단일 바꾸기 - 무리 없이 구현 완료하였다.



다중바꾸기 - 역시 별 다른 문제 없이 구현하였다.

C언어 문법 색상 지정 (InputController)

-키워드 색상지정 (KeywordEdit 메소드)

```
else if  
if  
while (true)
```

-키워드 색을 지정해줄 때 잘 지정해주는 모습을 볼 수 있고 괄호를 사용하여도 색상 지정을 잘해주는 모습을 볼 수있음.

-헤더파일 색상지정 (IncludeEdit 메소드)

```
#include<stdio.h>  
#include <stdio.h>  
#include<string.h>  
#include<stdlib.h>  
#include <stdio.h>|
```

-헤더파일의 색을 지정해줄 때 띄어쓰기를 하지 않아도 색상을 지정을 잘해주는 모습을 볼 수 있음.

-#define 상수 색상지정 (DefineEdit 메소드)

```
SONG ;  
#define SONG 10  
  
SONG ;  
song ;  
(SONG) ;  
  
#define !SONG 20  
#define 123SONG 20  
  
!SONG ;
```

-#define 상수를 지정해줄 때 선언전에 사용할 경우 색상지정을 해주지않는 모습과 대소문자 구분을 해주는 모습을 볼수있음. 또한 괄호를 사용해도 색상지정을 잘해주는 모습을 볼 수있음.

-상수명 선언시에 규칙을 지키지않으면 오류로 표시해주는 모습을 볼수있음, 또한 오류처리된 상수명은 사용불가능한 모습을 볼 수있음.

-데이터타입 및 식별자명, 함수명 색상지정 (DataTypeEdit 메소드)

```
void song ( int a ) ;  
int song2 ( int b )  
{  
    b = 4 ;  
    return b ;  
}  
int main ()  
{  
    abc ;  
    int abc = 0 ;  
    abc ;  
    abcd ;  
    double int abcd = 0 ;  
    abcd ;  
    song (abc) ;  
    song2 (abc) ;  
    song3 (abc) ;  
    int !abc ;  
    int if ;  
    int else ;  
    char 123song ;  
    if ;  
    !abc ;  
}  
int song3 ( int c )  
{  
    c = 4 ;  
    return c ;  
}
```


<세미콜론>

```
#include <stdio.h>
int main ()
{
    int b;
    int a;
    int song; int c = 0;
}
void SONG ( int abc )
void SONG2 ( int abcd );
void SONG3 ( int abc )
{
    abc = 0;
    abc = 1;
}
```

-세미콜론 처리가 필요없는 라인은 세미콜론 처리를 안해주어도 오류출력을 안해주고 세미콜론이 필요한 라인은 세미콜론이 없을 시 오류 출력을 해주는 모습을 볼 수 있음.

-한 라인에서도 세미콜론을 통해 구분하여 정상 출력 또는 오류 출력을 구분해주는 모습을 볼 수있음.

-함수의 선언부분에서는 세미콜론이 없으면 오류가 나오나 함수의 정의부분에서는 세미콜론이 없어도 정상적인 출력이 되는 모습을 볼 수 있음.

<괄호 따옴표 순서>

```
//asd " asd asd "
/* "asd asd //asd asd */
/* //asd asd asd " asd asd "

"asd asd //asd asd /* */" ;
"asd asd //asd asd sa /*asd asd
```

-앞에서 이야기했던대로 주석과 따옴표의 순서를 고려하여 잘 처리해주는 모습을 볼 수있음. 주석의 경우 뒤에 따옴표가 나오면 주석처리해주어야하고 따옴표의 경우 뒤에 주석이 나오면 문자열처리해주어야한다.

<최종 점검>

```
#include<stdio.h>
#include <string.h>
void song1 ( int a );
int main ()
{
    abc;
    int abc = 0;
    song1 ( abc );
    song2 ( abc );
    //asdasdasdaasdd
    "asdasd";
    /*asdasdasd
    asdasd */
    if ( abc == 1 )
        printf("안녕하세요");

    else
        printf("안녕");

    int !asdasd;
    !asdasd;
    (((())))({{{}}})!!!;
}
void song1 ( int a )
{
    a = 0;
}
char song2 ( char c )
{
    c = 'a';
    return c;
}
```

-모든 기능이 잘 작동하는 모습을 볼 수있다

4-2. 설계 구성요소 및 제한요소 평가

설계 구성요소				설계 제한요소		
목표설정	분석	구현/제작	시험/평가	성능	규격/표준	테스트
○	○	▲	○	○	○	○

지난 중간레벨에서와는 달리 이번 고급 레벨 문제에서는 구현에 시간을 많이 투자하였다. 또한 git 사용이 살짝 미숙하여 각 팀원들의 소스코드를 합치는데에 있어서 시간이 살짝 걸렸었다. 또한 시간적인 관계로 구현/제작 파트가 미흡하였는데 이는 다음 표를 보면 알 수 있다.

기본 기능 구현 여부		
기능 분류	기능 상세	구현 여부
기본 기능	새 파일, 열기, 닫기, 저장, 종료 FileController	○
이동 기능	화살표 및 이동 키 이동 MainView	○
편집 기능	기본 입력 및 수정 MainView	○
C 문법 검사 기능	괄호 짝 맞추기, 주석문 표시 InputController	○

추가 기능 구현 여부		
기능 분류	기능 상세	구현 여부
Undo/Redo(Ctrl + Z)	키보드 입력	X
단어 검색 및 바꾸기	EditController	○
C 문법 검사 추가 기능	InputController	○

표에서 볼 수 있듯 구현 단계에선 필수 기능들은 모두 완료하였고 추가 기능 중 Undo/Redo 부분에서 구현에 실패하였는데 이는 다음 장에서 확인할 수 있다.

5. Discussion

소스코드가 길어 보고서에 담기에는 무리가 있다고 판단하여 해당 소스코드는 압축된 파일에 주석을 달아 첨부하였다. 또한 프로그램코드에 주석을 달아 프로그램 코드를 이해하는 것에는 무리가 없다고 생각하나 추가로 설명해주면 좋을 것같은 부분들이 있어 이 부분에 대해 추가적인 설명을 기술하겠음.

5-1. 구현 결과의 특성과 이유

1. C언어 문법 색상 지정 (InputController)

```
new Thread( task: this).start();
```

특성: InputController은 쓰레드를 사용한다.

이유: 사용자가 입력한 값을 계속해서 변경해줘야하므로 쓰레드를 사용하여 구현하였다.

```
UserTextWord = new ArrayList<String>(); //트큰화한 데이터를 담기위한 String형 가변배열 자세한 설명은 보고서에 기술하였음
StringTokenizer userTextTokenizer = new StringTokenizer(UserText.getText()); //공백을 기준으로 토큰화
while (userTextTokenizer.hasMoreTokens()) {
    UserTextWord.add(userTextTokenizer.nextToken()); //키워드, 헤더파일명, 매크로, 식별자, 함수를 위한 공백을 기준으로 분리한 문자열을 가변배열에 저장
}
```

특성: 키워드, 헤더파일, 식별자, 함수, #define 상수는공백을 기준으로 띄운 것으로 구분한다

이유: C언어는 int a, 와 같이 공백을 기준으로 한다. 그래서 공백을 기준으로 구분하는 것이 정확도와 효율성 측면에서 좋다고 생각해서 공백을 기준으로 나누어 구현하였다.

```
UserTextCharacter = UserTextString.toCharArray(); //주석, 괄호, 따옴표, 세미콜론을 위한 문자형배열으로 분리
```

특성: 주석, 따옴표, 괄호는 위에 키워드, 헤더파일, 식별자와는 다르게 문자 하나하나를 기준으로 띄운 것을 구분한다

이유: 사용자가 주석 및 따옴표, 괄호는 공백을 두지않고 사용할 때가 많다. 그래서 정확성을 위해 문자를 각각 나누어서 구현하였다.

-키워드 색상지정 (KeywordEdit 메소드)

```
for (int Wordindex = 0; Wordindex < UserTextWord.size(); Wordindex++) { //사용자가 입력한 텍스트를 공백단위로 분리한 데이터들
    for (int ModelKeywordindex = 0; ModelKeywordindex < getKeyword.length; ModelKeywordindex++) { //키워드 모델의 데이터들
```

특성: 사용자가 입력한 텍스트를 공백단위로 분리한 데이터와 모델의 데이터를 비교하여 색을 지정해준다.

이유: C언어에서의 키워드는 공백을 기준으로 구분하는 것이 많아 효율적인 구현을 위해서이다.

-헤더파일 색상지정 (IncludeEdit 메소드)

```
for (int WordIndex = 0; WordIndex < UserTextWord.size(); WordIndex++) {
    for (int HeaderIndex = 0; HeaderIndex < getHeader.length; HeaderIndex++) { //헤더파일명 모델의 데이터들
        if (UserTextWord.get(WordIndex).equals("#include" + getHeader[HeaderIndex])) { //#include<stdio.h>와 같이 붙여쓰는 경우를 처리
            //...
        } else if (WordIndex != 0 && UserTextWord.get(WordIndex - 1).equals("#include")) { //#include <stdio.h> 와 같이 띄어쓰는 경우 처리 이 경우 #include뒤에 나오니 0에서는 처리해주면 오류 발생
            //...
        }
    }
}
```

특성: 헤더파일명의 경우에는 띄어쓰는경우와 띄어쓰지않는 경우 모두 처리해주어야한다.

이유: 사용자마다 띄어쓰는사람이 있고 띄어쓰지않는 사람이있다. 사용자의 편의성을 위해서이다.

```
if (WordIndex != 0 && UserTextWord.get(WordIndex - 1).equals("#include")) {
```

특성: 헤더파일명은 #include 뒤에 나오는 점을 이용하여 구현하였다.

-#define 상수 색상지정 (DefineEdit 메소드)

```
for (int Wordindex = 1; Wordindex < UserTextWord.size(); Wordindex++) { //무조건 #define 뒤에 나오니 1부터 시작
    if (UserTextWord.get(Wordindex - 1).equals("#define")) { //만약 앞에 단어가 #define이라면
```

특성: #define 상수는 무조건 #define 뒤에 나오는 점을 이용하여서 구현하였다.

-데이터타입 및 식별자명, 함수명 색상지정 (DataTypeEdit 메소드)

```
for (int Wordindex = 0; Wordindex < UserTextWord.size(); Wordindex++) { //여기서는 이해하기 쉽게 설명하자면 int, char 과 같은 데이터타입의 색 지정임.
    For (int ModelDataindex = 0; ModelDataindex < getDatatype.length; ModelDataindex++) {
```

특성: 사용자가 입력한 텍스트를 공백단위로 분리한 데이터와 모델의 데이터를 비교하여 색을 지정해준다.

이유: C언어에서의 기본타입은 공백을 기준으로 구분하는 것이 많아 효율적인 구현을 위해서이다.

```
for (int Wordindex = 1; Wordindex < UserTextWord.size(); Wordindex++) { //여기에서는 식별자 와 함수의 색 지정임.
    if (!(Arrays.asList(getDatatype).contains(UserTextWord.get(Wordindex)))) { //double int와 같이 데이터타입명이 연속으로 쓰일 경우를 방지하여 현재 단어가 데이터타입이 아닐경우
        if ((Arrays.asList(getDatatype).contains(UserTextWord.get(Wordindex - 1)))) { //현재 단어의 전 단어가 데이터 타입일 경우
```

특성: 식별자명 및 함수명은 무조건 데이터타입 뒤에 나오는 점을 이용하였다. 또한 데이터타입명이 연속으로 쓰일 경우를 방지해야한다.

-주석, 따옴표, 괄호, 세미콜론 색상지정 (MarkEdit 메소드)

```
Stack<Character> MarkEditStack = new Stack<>(); //주석, 따옴표, 괄호의 색 지정을 위한 Stack 위 3가지는 누가 먼저나오냐에따라서 처리방식이 달라짐 그래서 하나의 스택에 일괄적으로 처리해야함
```

특성: 주석, 따옴표, 괄호의 순서고려를 위해 하나의 스택만을 사용한다.

이유: 여러 개의 스택을 사용한다면 순서고려가 불가능하다.

(문제 발생 원인과 해결 방법에서 보다 자세히 다룸)

```
Stack<Integer> brackePosStack = new Stack<>(); //괄호의 색지정과 오류색 지정을 위해서는 열린괄호의 위치를 알고있어야함.
```

특성: 괄호는 따로 자신만의 열린괄호위치를 담아줄 스택이 필요하다.

이유: 괄호는 소괄호, 중괄호, 대괄호를 구분할 필요가있다. 또한 주석과는 다르게 열린괄호만 여러번 나올 수가 있기 때문에 변수 하나로는 위치저장을 할 수 없다. 만약 변수로만 구현한다면 (괄호의 종류) x (열린괄호의 개수)만큼의 변수가 필요할 것이다. 효율적인 구현을 위해 스택을 사용하는 것은 필수적이다.

또한 열린괄호의 위치를 담는 이유는 후에 닫힘괄호를 만났을 때 열린괄호의 색 지정을 위해서이다.

(문제 발생 원인과 해결 방법에서 보다 자세히 다룸)

```

#include <stdio.h>
int main ()
{
    int b;
    int a;
    int song; int c = 0;
}
void SONG ( int abc )
void SONG2 ( int abcd );
void SONG3 ( int abc )
{
    abc = 0;
    abc = 1;
}

```

특성: 세미콜론처리를 해줄 때 세미콜론이 필요한 라인과 필요하지 않은 라인을 구분해야한다. 또한 한라인에서도 세미콜론을 통하여 구분해야한다.

이유: C언어에서는 키워드, 주석, 헤더파일의 경우에는 세미콜론을 사용하지않아도 오류처리가 되지않는다. 또한 함수의 선언에서도 오류처리가 되지않는다. 또한 C언어에서는 한라인에서 ; 의 위치를 통해서도 오류를 출력해주거나 해주지않는다. 사용자의 편의성을 위해서 최대한 동일하게 구현해야한다.

(문제 발생 원인과 해결 방법에서 보다 자세히 다룸)

2. 텍스트에 찾기 및 바꾸기 (EditController & EditView) (소스코드에 주석처리함. 자세한 사항은 소스코드를 방문하길 권함)

```

public class EditController extends JFrame implements ActionListener {
    3개 사용 위치
    private static SearchView SearchView;
    3개 사용 위치
    private static ChangeView ChangeView;
    3개 사용 위치
    private static ChangeAllView ChangeAllView;
    3개 사용 위치
    private static SearchAllView SearchAllView;
}

```

특성: EditController의 클래스. 총 4개의 멤버 객체(뷰)를 받아서 사용한다.

이유: 메인뷰의 찾기와 바꾸기의 요청을 처리하기 위해 4개의 뷰들을 사용한다.


```

public void actionPerformed(ActionEvent e)
{
    if (e.getActionCommand().equals("Search    Ctrl+F"))
    {
        SearchView = new SearchView();
    }
    else if (e.getActionCommand().equals("SearchALL    Ctrl+G"))
    {
        SearchAllView = new SearchAllView();
    }
    else if (e.getActionCommand().equals("Change    Ctrl+R"))
    {
        ChangeView = new ChangeView();
    }
    else if (e.getActionCommand().equals("ChangeALL    Ctrl+T"))
    {
        ChangeAllView = new ChangeAllView();
    }
    else if (e.getActionCommand().equals("remove highlight"))
    {
        if(MyActionListener.text_highlight != null)
        {
            MyActionListener.text_highlight.removeAllHighlights();
        }
    }
}

```

특성: MainView의 해당 기능 버튼 클릭의 입력을 처리하는 메소드이다.

```

public static class MyKeyListener extends KeyAdapter {
    @ badasskim *
    public void keyPressed(KeyEvent e) {
        if (e.isControlDown() && e.getKeyCode() == KeyEvent.VK_F) {
            SearchView = new SearchView();
        } else if (e.isControlDown() && e.getKeyCode() == KeyEvent.VK_G) {
            SearchAllView = new SearchAllView();
        } else if (e.isControlDown() && e.getKeyCode() == KeyEvent.VK_R) {
            ChangeView = new ChangeView();
        } else if (e.isControlDown() && e.getKeyCode() == KeyEvent.VK_T) {
            ChangeAllView = new ChangeAllView();
        }
    }
}

```

특성: 단축키를 지정해 해당 단축키를 누르면 각 뷰 객체를 생성하는 내부 클래스이다.

```
public static class MyActionListener implements ActionListener
{
```

특성: SearchView, SearchAllView, ChangeView, ChangeAllView 에 존재하는 내부 버튼들의 액션을 담당하는 클래스이다. 즉 이 내부 클래스 안에 액션을 담당하는 메소드가 있는데 이 메소드들 속 하나하나 살펴보도록 하자.

```
JButton currentButton = (JButton)e.getSource();
```

이 JButton이 현재 이벤트가 요청된 버튼을 의미한다.

1. 단일 찾기

```
if(currentButton.getText().equals("단일 찾기"))
{
    text_highlight.removeAllHighlights(); // 먼저 존재하는 하이라이트 제거
    viewText = textPane.getText().replace( target "\r\n", replacement "\n"); /// 텍스트권의 전체 문자열, \r\n 처리된 개행을 \n으로 변경
    find_text = SearchView.tf1.getText(); // 찾으려는 단어

    if(viewText.contains(find_text)) // 단어가 존재한다면
    {
        word_count = 1; // 단어 갯수
        offset = viewText.indexOf(find_text, offset); // 뷰텍스트의 단어 위치
        count = 0; // 단어 전체를 안 돌았을 때
        try { text_highlight.addHighlight(offset, p1: offset + find_text.length(), DefaultHighlighter.DefaultPainter); } // 단어 하이라이트
        catch (BadLocationException CanNotSearch) { CanNotSearch.printStackTrace(); }
        offset = offset + find_text.length(); // 현재 작업 위치
        if(offset > viewText.lastIndexOf(find_text)) {last_offset = offset; count = 1; offset = 0;} // 단어가 마지막이라면 맨 위쪽으로 초기화
    }
    else // 단어가 존재하지 않는다면
    {
        word_count=0;
        JOptionPane.showMessageDialog( parentComponent null, message: "더 이상 찾는 단어가 없습니다.", title: "message", JOptionPane.WARNING_MESSAGE); // 에러메세지 출력
    }
}
```

먼저 단일 찾기이다. 하이라이팅 된 단어들이 있다면 이를 삭제하고 SearchView에 입력된 텍스트를 읽어온다. 이를 find_text에 저장하고 MainView(viewText)와 대조하여 존재여부를 확인한다. 존재하면 각 멤버 변수들을 조정하고 해당 위치의 단어를 하이라이팅한다. offset은 나중에 바꾸기 기능에 사용된다.

```
if(offset > viewText.lastIndexOf(find_text)) {last_offset = offset; count = 1; offset = 0;}
```

또한 마지막 if문은 단일찾기의 단어가 여러개일 때, 맨 마지막의 단어를 하이라이팅 하고 다시 찾기 버튼을 누를시, 뷰텍스트의 첫 번째 단어로 다시 가는 역할을 한다. 이는 나중에 바꾸기 기능에서 조건문으로 분기 해야 하는 이유이기도 하다.

만약 존재하지 않는다면 word_coun는 0으로 정의하고 찾는 단어가 없다고 GUI 메시지를 출력한다.

2. 다중 찾기

```
else if(currentButton.getText().equals("다중 찾기"))
{
    text_highlight.removeAllHighlights(); // 하이라이팅 제거
    viewText = textPane.getText().replace( target: "\r\n", replacement: "\n"); // 텍스트판의 전체 문자열, \r\n 처리된 개행을 \n으로 변경
    find_text = SearchAllView.tf1.getText(); // 찾으려는 단어

    if(viewText.contains(find_text)) // 존재한다면
    {
        max = viewText.lastIndexOf(find_text); // 해당 단어의 마지막으로 있는 위치
        for(int k=0; k < max; k++)
        {
            fi = viewText.indexOf(find_text, fi); // 뷰텍스트의 위치
            try
            {
                text_highlight.addHighlight(fi, p1: fi+find_text.length(), DefaultHighlighter.DefaultPainter); // 하이라이팅
            }
            catch (BadLocationException CanNotSearch)
            {
                CanNotSearch.printStackTrace();
            }
            fi = fi + find_text.length(); // 다음 단어 위치 변경
            if(fi ≥ max + 1) break; // 마지막을 넘었으면 반복문 종료
        }
    }
}
```

반복문을 사용한다. 단일 찾기와 똑같이 메인뷰에 있는 텍스트를 가져오고 찾으려는 단어의 맨 마지막 위치를 파싱한다. 그리고 이를 기반으로 반복문을 돌려 각 단어에 `fi + find_text.length()`만큼 하이라이팅을 진행하고 종료조건인 `max`값보다 `fi` 값이 더 크면 종료한다.

3. 단일 바꾸기

```
else if(currentButton.getText().equals("단일 바꾸기"))
{
    if(word_count ≠ 0) // 단어가 존재할 때(하이라이팅)
    {
        conversion_word = ChangeView.tf1.getText(); // 바꾸려는 단어
        if(count == 1) // 마지막에서 첫번째로 다시 돌아온 경우가 아니라면
        {
            textPane.select( selectionStart: last_offset-find_text.length(), last_offset); // 바꿀 단어 선택
            textPane.replaceSelection(conversion_word); // 변경
            try {
                text_highlight.addHighlight( p0: last_offset - find_text.length(), p1: last_offset - find_text.length() + conversion_
                // 변경된 단어 하이라이팅
            } catch (BadLocationException e1) {
                e1.printStackTrace();
            }
            count=0;
        }
    }
}
```

바꾸기이다. 여기서 조건문이 들어가는데 count == 1 이냐 아니냐의 차이다. 이 차이는 즉 검색한 단어를 viewText에 한 번씩 돌고 맨 처음의 단어를 하이라이팅 하였는가의 여부를 따지는 것이다.

만약 맨 처음의 단어로 되돌아 온 것이 아니라면 textPane의 텍스트를 해당 단어의 위치만큼 선택하고 이를 대체한다.

```
else // 마지막에서 첫번째로 다시 돌아온 경우(첫번째 단어)
{
    textPane.select(selectionStart: offset-find_text.length(), offset); // 바꿀 단어 선택
    textPane.replaceSelection(conversion_word); // 변경
    try {
        text_highlight.addHighlight(p0: offset-find_text.length(), p1: offset-find_text.length()+ conversion_word.length());
        // 변경된 단어 하이라이팅
    } catch (BadLocationException e1) {
        e1.printStackTrace();
    }
}
}

word_count=0; // 변경할 단어 없음
```

만약 되돌아 온 후 첫 번째 문자라면 오프셋의 위치가 달라졌으므로 이에 맞게 단어 위치를 선택한 후 변경한다. 그리고 이 작업들이 마무리되면 word_count 변수를 0으로 변경하여 더 이상 처리할 단어가 없음을 알려준다.

4. 다중 바꾸기

```
else if(currentButton.getText().equals("다중 바꾸기"))
{
    int cpos=0;
    String cur, after;
    conversion_word = ChangeAllView.tf1.getText();
    cur = textPane.getText(); // 현재 텍스트팬 적힌 문자열
    after = cur.replaceAll(find_text, conversion_word); // 변환할 문자열
    textPane.selectAll(); // 텍스트팬 전체 선택
    textPane.replaceSelection(after); // 바뀐 문자열로 교체
    int max2;
    max2 = after.lastIndexOf(conversion_word);
    for(int k=0; k<after.length(); k++) // 바뀐 단어 하나씩 하이라이팅.
    {
        cpos = after.indexOf(conversion_word, cpos);
        try {
            text_highlight.addHighlight(cpos, p1: cpos+ conversion_word.length(), DefaultHighlighter.DefaultPainter);
        } catch (BadLocationException ble) {
        }
        cpos = cpos+ conversion_word.length();
        if(cpos >= max2 + 1)
            break;
    }
}
```

다중바꾸기이다. 텍스트팬에 적힌 모든 정보를 바뀐 단어로 변환된 정보를 after란 변수에 삽입하고 이를 텍스트팬에 다시 교체한다. 마지막으로 바뀐 단어의 하이라이팅을 해주는 작업을 걸친다.

5-2. 문제 발생 원인과 해결 방법

1. C언어 문법 색상 지정 (InputController)

-공통적인 부분

문제 발생 원인:

단어의 색을 지정해주기위해서는 위치값이 필요한데 이 위치값을 구하기 위해서 indexOf를 사용하였으나 만약 사용자가 해당단어를 포함하는 단어를 사용할 경우 내가 원하는 위치가아닌 해당단어를 포함하는 단어에 색을 지정해준다.

```
else if (KeywordValue.contains(ModelKeywordValue)) { //만약 위에 모든 경우가 아니고 사용자가 키워드의 명을 포함하는 문자열을 작성하였을 경우 색을 지정해주면 안됨.  
    Keywordindex = UserTextString.indexOf(KeywordValue, Keywordindex); //그렇다고 해서 검색위치를 업데이트해주지않으면 후에 키워드명을 포함하는 문자열의 색을 지정해주는 버그가 발생.  
    Keywordindex = Keywordindex + KeywordValue.length();
```

해결방법: Contains를 이용하여 포함할 경우에는 검색 위치를 업데이트하여 다음 검색을 할때에는 해당 단어를 지나치게 하였다.

문제발생 원인: 개행문자는 운영체제마다 다르다. (윈도우:'\r\n' , 맥:'\r' , 유닉스:'\n') 이런 방식은, 서로 다른 종류의 OS에서 동작하는 프로그램에서 문제가 발생할 수 있다.

```
UserTextString = UserText.getText().replaceAll(System.getProperty("line.separator"), replacement: "\n");
```

해결방법: "line.separator" 은 프로그램이 실행되는 OS의 개행 문자를 리턴한다, 해당 메서드를 사용하여 어떤 OS에서 실행해도 해당 OS의 개행문자를 리턴하여 해당 개행문자를 '\n'으로 바꾸어 주어 해결하였다.

-키워드 색상지정 (KeywordEdit 메소드)

문제 발생 원인:

사용자의 편의성을 위해서 괄호를 사용해도 키워드의 색을 지정해주어야한다.

```
else if (KeywordValue.equals("(" + ModelKeywordValue + ")") || KeywordValue.equals("{ " + ModelKeywordValue + "}") || KeywordValue.equals("[ " + ModelKeywordValue + "]")) {  
    Keywordindex = UserTextString.indexOf(KeywordValue, Keywordindex);  
    for (int SeachKeyWordColor = Keywordindex + 1; SeachKeyWordColor < Keywordindex + ModelKeywordValue.length() + 1; SeachKeyWordColor++) {  
        UserTextCharacterColor[SeachKeyWordColor] = "CYAN";  
    }  
}
```

해결방법:

괄호의 경우에는 경우가 그렇게 많지 않다. 수작업으로 해결하였다.

-헤더파일 색상지정 (IncludeEdit 메소드)

문제 발생 원인:

사용자의 편의성을 위해서 헤더파일명의 경우에는 띄어쓰는경우와 띄어쓰지않는 경우 모두 처리해주어야 한다.

```

if (UserTextWord.get(WordIndex).equals("#include" + getHeader[HeaderIndex])) { // #include <stdio.h>와 같이 붙여쓰는 경우를 처리
    Headerindex = UserTextString.indexOf(UserTextWord.get(WordIndex), Headerindex); // 붙여쓰는 경우 위에서 키워드명으로 인식할 못해주어서 여기서 색을 지정해주어야함.
    for (int IncludeColor = Headerindex; IncludeColor < Headerindex + "include".length(); IncludeColor++) {
        //System.out.println(k);
        UserTextCharacterColor[IncludeColor] = "CYAN";
    }
    Headerindex = Headerindex + "include".length(); // 검색 위치 업데이트
    for (int HeaderColor = Headerindex; HeaderColor < Headerindex + getHeader[HeaderIndex].length(); HeaderColor++) { // 헤더파일명의 색 지정
        //System.out.println(k);
        UserTextCharacterColor[HeaderColor] = "PINK";
    }
    Headerindex = Headerindex + getHeader[HeaderIndex].length(); // 검색 위치 업데이트
}

```

```

//System.out.println("test2");
} else if (WordIndex != 0 && UserTextWord.get(WordIndex - 1).equals("#include")) { // #include <stdio.h> 와 같이 띄어쓰는 경우 처리 이 경우 #include뒤에 나오니 0에서는 처리해주면 오류 발생
    if (UserTextWord.get(WordIndex).equals(getHeader[HeaderIndex])) { // 헤더파일명과 일치하는 경우
        Headerindex = UserTextString.indexOf(str "include", Headerindex); //정밀도를 위한 #include 부터 검색
        Headerindex = Headerindex + "include".length();
        Headerindex = UserTextString.indexOf(getHeader[HeaderIndex], Headerindex); //헤더파일명을 검색한다.
        //System.out.println("test2"+Headerindex);
        for (int HeaderColor = Headerindex; HeaderColor < Headerindex + getHeader[HeaderIndex].length(); HeaderColor++) { //헤더파일 색 지정
            //System.out.println(k);
            UserTextCharacterColor[HeaderColor] = "PINK";
        }
        Headerindex = Headerindex + getHeader[HeaderIndex].length();
    }
}

```

해결방법: 각각의 경우를 나누어서 해결하였다. 띄어쓰지않는 경우에는 위에 키워드 색상지정에서 색지정을 안해주니 여기에서 해주어야한다.

-#define 상수 색상지정 (DefineEdit 메소드)

문제 발생 원인: 괄호를 사용해도 상수의 색을 지정해주어야한다.

```

else if (SearchConstantValue.equals("(" + DefineValue + ")") || SearchConstantValue.equals("{ " + DefineValue + "}") || SearchConstantValue.equals("[ " + DefineValue + "]")) {
    DefineValueindex = UserTextString.indexOf(SearchConstantValue, DefineValueindex);
    for (int SearchConstantColor = DefineValueindex + 1; SearchConstantColor < DefineValueindex + DefineValue.length() + 1; SearchConstantColor++) {
        UserTextCharacterColor[SearchConstantColor] = "MAGENTA";
    }
    DefineValueindex = DefineValueindex + SearchConstantValue.length();
}

```

해결방법:

괄호의 경우에는 경우가 그렇게 많지 않다. 수작업으로 해결하였다.

문제 발생 원인:

상수명의 처음 단어가 특수문자가 오거나 숫자가 오면 에러표시를 해주어야한다

```

!(0<DefinecheckFirsttint && 10>DefinecheckFirsttint) && DefinecheckFirst != '!' && DefinecheckFirst != '@' && DefinecheckFirst != '#'

```

```

} else { //C언어 규칙에 맞지않는 상수선언은 오류처리
    for (int j = Defineindex; j < UserTextCharacter.length; j++) {
        if (UserTextCharacter[j] == '\n') { //개행까지 에러로 색 지정
            break;
        }
        UserTextCharacterColor[j] = "ERROR";
    }
    Defineindex = Defineindex + DefineValue.length(); //오류더라도 검색위치 업데이트를 해주어야함. 이를 안해줄씨 마찬가지로 한칸씩 밀리는 오류 발생 가능성 있음.
}

```

해결방법: 마찬가지로 수작업으로 찾아내고 조건에 부합할 시에 개행까지 에러로 색을 지정해주어서 해결 해주었다.

문제 발생 원인: 상수는 선언 뒤에 나오는 것만 색상 처리 해줘야한다.


```

Defineindex = Defineindex + DefineValue.length();
DefineValueindex = Defineindex; //해당 위치부터 후에 나오는 동일한 이름을 가진 상수를 처리해주어야함. 이를 안해줄시 선언 전에 나오는 동일한 명을 가진 상수도 색 지정해주는 버그가 발생
for (int SearchConstant = Wordindex + 1; SearchConstant < UserTextWord.size(); SearchConstant++) { //선언 위치 이후부터 검색
    SearchConstantValue = UserTextWord.get(SearchConstant);
    if (SearchConstantValue.equals(DefineValue)) { //후에 나오는 단어가 위에 선언값의 상수명과 같다면

```

해결방법: 선언할때의 위치값을 이용하여 이후에 나오는 상수명만 색을 지정하여준다. 즉 순차적으로 처리하여준다.

문제 발생 원인: 상수의 색 지정에서 현재 우리 알고리즘은 공백을 기준으로 분리하여 사용자가 ;을 띄어쓰기 하지 않고 붙일 경우 ;까지 이름 처리 된다. 사용자가 ;을 제일 뒤에 붙일 경우 알려줘야한다.

```

if (DefinecheckLast != ';' )

```

```

} else { //C언어 규칙에 맞지않는 상수선언은 오류처리
    for (int j = Defineindex; j < UserTextCharacter.length; j++) {
        if (UserTextCharacter[j] == '\n') { //개행까지 에러로 색 지정
            break;
        }
        UserTextCharacterColor[j] = "ERROR";
    }
    Defineindex = Defineindex + DefineValue.length(); //오류더라도 검색위치 업데이트를 해주어야함. 이를 안해줄씨 마찬가지로 한칸씩 밀리는 오류 발생 가능성 있음.

```

해결방법: 마찬가지로 수작업으로 찾아내고 개행까지 에러로 색상을 지정하여준다.

-데이터타입 및 식별자명, 함수명 색상지정 (DataTypeEdit 메소드)

문제 발생 원인: 괄호를 사용해도 식별자 및 함수의 색을 지정해주어야한다.

```

else if (DataValue.equals("(" + ModelDataValue + ")") || DataValue.equals("{ " + ModelDataValue + "}") || DataValue.equals("[ " + ModelDataValue + "]")) {
    Dataindex = UserTextString.indexOf(DataValue, Dataindex);
    for (int SearchKeywordColor = Dataindex + 1; SearchKeywordColor < Dataindex + ModelDataValue.length() + 1; SearchKeywordColor++) {
        UserTextCharacterColor[SearchKeywordColor] = "BLUE";
    }
}

} else if (SearchDataValue.equals("(" + DataTypeValue + ")") || SearchDataValue.equals("{ " + DataTypeValue + "}") || SearchDataValue.equals("[ " + DataTypeValue + "]")) {
    DataTypeValueindex = UserTextString.indexOf(SearchDataValue, DataTypeValueindex);
    for (int SearchDatatypeColor = DataTypeValueindex + 1; SearchDatatypeColor < DataTypeValueindex + DataTypeValue.length() + 1; SearchDatatypeColor++) {
        UserTextCharacterColor[SearchDatatypeColor] = "MAGENTA";
    }
    DataTypeValueindex = DataTypeValueindex + SearchDataValue.length();
}

```

해결방법: 괄호의 경우에는 경우가 그렇게 많지 않다. 수작업으로 해결하였다.

문제 발생 원인: 식별자명 및 함수명의 처음 단어가 특수문자가 오거나 숫자가 오면 에러표시를 해주어야한다. 또한 식별자명이나 함수명은 키워드명이 될 수 없다.

```

if (!(Arrays.asList(getKeyword).contains(DataTypeValue)) && DataTypecheckLast
    !(0<DataTypecheckFirstint && 10>DataTypecheckFirstint) && DataTypecheckFirst != '!' && DataTypecheckFirst != '@' && DataTypecheckFirst != '#')

```

```

else {
    for (int j = DataTypeindex; j < UserTextCharacter.length; j++) {
        if (UserTextCharacter[j] == '\n') {
            break;
        }
        UserTextCharacterColor[j] = "ERROR";
    }
    DataTypeindex = DataTypeindex + DataTypeValue.length();
}

```

해결방법:

마찬가지로 수작업으로 찾아내고 조건에 부합할 시에 개행까지 에러로 색을 지정해주어서 해결해주었다.

문제 발생 원인:

식별자 및 함수는 선언 뒤에 나오는 것만 색상 처리 해줘야한다.

```

DataTypeindex = DataTypeindex + DataTypeValue.length();
DataTypeValueindex = DataTypeindex; //해당 위치부터 후에 나오는 동일한 이름을 가진 식별자와 함수를 처리해주어야함.
//System.out.println("test3: " + DataTypeindex);
for (int SearchConstant = Wordindex + 1; SearchConstant < UserTextWord.size(); SearchConstant++) { //우
    SearchDataValue = UserTextWord.get(SearchConstant);
    if (SearchDataValue.equals(DataTypeValue)) {

```

해결방법:

선언할때의 위치값을이용하여 이후에 나오는 상수명만 색을 지정하여준다. 즉 순차적으로 처리하여준다.

문제 발생 원인:

식별자명 및 함수명의 색 지정에서 현재 우리 알고리즘은 공백을 기준으로 분리하여 사용자가 ;을 찍어쓰기 하지 않고 붙일 경우 ;까지 이름 처리 된다. 사용자가 ;을 제일 뒤에 붙일 경우 알려줘야한다. 또한 식별자의 경우에는 괄호와 같이 사용하는 경우가 많다. 이를 알려주어야한다.

```

DataTypecheckLast != ';' && DataTypecheckLast != '(' && DataTypecheckLast != ')'

```

```

else {
    for (int j = DataTypeindex; j < UserTextCharacter.length; j++) {
        if (UserTextCharacter[j] == '\n') {
            break;
        }
        UserTextCharacterColor[j] = "ERROR";
    }
    DataTypeindex = DataTypeindex + DataTypeValue.length();
}

```

해결방법:

마찬가지로 수작업으로 찾아내고 개행까지 에러로 색상을 지정하여준다.

-주석, 따옴표, 괄호, 세미콜론 색상지정 (MarkEdit 메소드)

문제 발생 원인:

주석, 따옴표, 괄호의 순서를 고려하여 색상지정을 해주어야한다.

①

```
Stack<Character> MarkEditStack
```

②

```
if (MarkEditStack.size() != 0) {
```

③

```
if ((MarkEditStack.peek() != '"') && (MarkEditStack.peek() != '*') && (MarkEditStack.peek() != '+')) {
```

④

```
MarkEditStack.push
```

해결방법:

주석, 따옴표, 괄호의 순서고려는 pop은 push이후의 일이니 push만 생각하면 된다. 모든 push의 기본적인 알고리즘은 다음과 같다.

①의 사진과 같이 하나의 스택을 활용하여 구현하였고 만약 ②의 사진과 같이 스택의 크기가 0이 아닌경우에만 peek를 해줄 수 있으므로 해당 조건을 붙이고 ③ 만약 peek한 값 즉 가장 최근에 삽입 된 값이 따옴표나 열린주석('*')이나 한줄주석('+')이 아닐경우에만 ④사진과 같이 push해준다.

②의 사진과 반대로 스택의 사이즈가 0일경우에는 push만 해주면 되고 주석, 괄호, 따옴표의 순서를 고려할 필요가 없다.

문제 발생 원인:

호는 소괄호, 중괄호, 대괄호를 구분할 필요가있다. 또한 주석과는 다르게 열린괄호만 여러번 나올 수가 있기 때문에 변수 하나로는 위치저장을 할 수 없다.

①

```
Stack<Integer> brackePosStack = new Stack<>();
```

②

```
MarkEditStack.push(BracketItem); //대괄호를 푸쉬  
brackePosStack.push(OpenBracketPos); //대괄호의 위치를 푸쉬
```

③

```
MarkEditStack.pop();  
UserTextCharacterColor[brackePosStack.peek()] = UserTextCharacterColor[characterIndex] = "MAGENTA";  
brackePosStack.pop();
```

해결방법:

①의 사진과 같이 열린괄호의 위치를 저장하기 위해 int형 Stack를 추가로 선언하여준다. 만약 push해야 할 상황이온다면 ②의 사진과 같이 괄호의 종류는 공동으로 사용하는 스택에 삽입하여주고 열린 괄호의

위치는 ①의 스택에 삽입하여준다. pop해야할 상황이온다면 공동으로 사용하는 스택은 pop해주고 ①의 스택을 활용하여 짝이 맞는 열린괄호의 위치를 가지고 온다. 해당 위치를 이용하여 색을 지정하여주고 pop해준다.

문제 발생 원인:

세미콜론의 필요 유무를 구분해야한다.

①

```
if (UserTextCharacterColor[characterColor].equals("GREEN") || UserTextCharacterColor[characterColor].equals("PINK") || UserTextCharacterColor[characterColor].equals("CYAN"))
```

②

```
if (UserTextCharacterColor[characterColor].equals("BLUE")) { //만약 데이터타입이고
    for (int Bracescheck = characterColor; Bracescheck < characterIndex; Bracescheck++) { //해당 데이터 타입 이후에 나온 문자가 범위는 한 라인
        if (UserTextCharacter[Bracescheck] == '(') { //여는괄호라면 위치저장
            OpenBracesindex=Bracescheck;
        }
        if (UserTextCharacter[Bracescheck] == ')') { //닫는괄호라면 위치저장
            CloseBracesindex=Bracescheck;
        }
    }
    if(OpenBracesindex < CloseBracesindex){ //int song( int a ) 와 같이 함수의 정의부분에서는 여는 괄호가 닫는괄호보다 먼저 나와야함
        for (int Bracescheck = CloseBracesindex+1; Bracescheck < UserTextCharacter.length; Bracescheck++) { //함수를 정의할때 해당라인에 여는 대괄호가
            if (UserTextCharacter[Bracescheck] == '{') { //여는 대괄호를 찾았을 경우에는
                Exception = 1; //마찬가지로 세미콜론이 없어도 됨
                OpenBracescheck = 1; //또한 뒤에 나오는 정의부 밑에 대괄호가 나올 경우를 처리해줄 필요가 없음.
                //System.out.println("test1");
                break;
            }
            else if (UserTextCharacter[Bracescheck] == ' ' || UserTextCharacter[Bracescheck] == '\n' || UserTextCharacter[Bracescheck] == '\t')
                //System.out.println("test2");
            }
            else { //하지만 여는괄호,공백,개행의 경우가 아닌경우에는 이상한 함수정의이거나 함수 선언임 그래서 세미콜론이 필요할 for문 break
                break;
            }
        }
    }
}
```

③

```
if (OpenBracescheck == 0) { //위에 함수 정의에서의 해당 라인에 대한 세미콜론 예외가 없었을 경우에만 실행하면 됨
    if (UserTextCharacter[characterColor] == '{') { //여는 대괄호일경우
        //System.out.println("test1");
        Exception = 1; //일단 세미콜론예외
        for (int Bracescheck = SemicolonIndex; Bracescheck < characterIndex; Bracescheck++) { //해당 여는 대괄호가 있는 라인 전부 검사
            //System.out.println(UserTextCharacter[Bracescheck]);
            if (UserTextCharacter[Bracescheck] != '{' && UserTextCharacter[Bracescheck] != ' ' && UserTextCharacter[Bracescheck] != '\t')
                Exception = 0; //해당 경우에는 세미콜론 예외처리를 해주면 안됨
                break;
            }
        }
    }
}
if (UserTextCharacter[characterColor] == '}'){ //이하 동일
    //System.out.println("test2");
    Exception = 1;
    for (int Bracescheck = SemicolonIndex; Bracescheck < characterIndex; Bracescheck++) {
        //System.out.println(UserTextCharacter[Bracescheck]);
        if (UserTextCharacter[Bracescheck] != '}' && UserTextCharacter[Bracescheck] != ' ' && UserTextCharacter[Bracescheck] != '\t') {
            Exception = 0;
            break;
        }
    }
}
```


④

```
if(ErrorException==0) //만약 위에 세미콜론 예외처리를 못받은 라인의 경우에는 세미콜론이 필요함.
{
    SemicolonCheckindex=SemicolonIndex; //일단 해당라인만 처리해주기 위해서 라인의 시작인덱스 저장
    for (int characterColor = SemicolonIndex; characterColor < characterIndex; characterColor++) { //해당라인 전체 검사
        if(UserTextCharacter[characterColor] ==';'){ //세미콜론을 만났다면
            UserTextCharacterColor[characterColor] = "GRAY";
            SemicolonCheckindex=characterColor+1; //해당 세미콜론의 위치 +1로 에러처리 해줄 시작 위치 변경
        }
    }
    for (int characterColor = SemicolonCheckindex; characterColor < characterIndex; characterColor++) { //세미콜론 위치 +1부터 해당라인 에러처리
        UserTextCharacterColor[characterColor] = "ERROR";
    }
}
```

해결방법:

세미콜론이 필요없는 경우는 ①의 경우인 해당라인에 키워드, 헤더파일, 주석이 있을경우이다. 또다른 경우는 ②와③의 경우인 함수의 정의 부분이다. ①은 간단하니 설명을 생략하도록하고 ②와③을 주로 다루겠다.

②의 경우에는 'int song(int a)'과 같은 함수의 선언부분에대한 처리이다. 여기서 중요한점은 함수를 선언 할때에는 int와 같은 데이터타입이 먼저나오고 괄호가 순서대로 나오는 모습이다. 그래서 만약 데이터타입 이고 괄호의 위치를 저장한뒤 열린괄호가 닫힌괄호보다 먼저 나왔다면 다음 단계로 넘어간다. 다음단계의 중요한 점은 함수의 선언과 정의의 구분이다. 함수의 선언과 같은 경우에는 세미콜론이 필요하다. 하지만 함수의 정의는 세미콜론이 필요하지않다. 구분을 위한 함수의 선언과 함수의 정의의 차이점은 후에 나오는 대괄호의 유무이다. 그래서 후에 세미콜론을 찾는다면 세미콜론이 필요없다고 처리해주는모습이다. 이 부분에서 중요한 부분이 나오는데 바로 사용자가 대괄호 이전에 공백을 추가할 수 도있으니 이 부분 또한 처리해주어야한다.

③의 경우에는 2가지로 나누어지는데 열림대괄호 처리와 닫힘대괄호 처리이다. ②에서 열린괄호의 처리를 해주었을 경우에는 ③의 열림대괄호처리를 해줄 필요가 없다. 각각의 처리를 해주는 알고리즘은 만약 해당라인에 대괄호밖에 없을 경우에는 세미콜론이 필요없다고 처리해주는 것이다. 여기서도 마찬가지로 사용자가 공백을 입력할 수 도 있으니 이를 대비하여 처리해주어야한다.

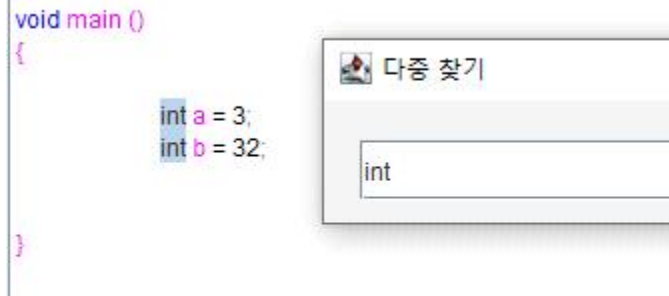
마지막으로 ④는 위에 ①②③에서 예외처리를 받지못하였을 경우에만 실행된다. ①②③에서 예외처리를 받지 못했다는 소리는 세미콜론이 필요한 라인이라는 뜻이다. 그래서 해당라인을 전체 검사하여 세미콜론의 위치+1로 시작위치를 변경하고 해당 시작위치를 이용하여 에러색상을 지정해준다. 세미콜론의 위치 +1을 해주는 이유는 'int a; int b;'와 같이 한 라인에서도 세미콜론을 통해 구분을 해주어야한다. 또한 주의점은 해당 라인에 세미콜론이 없을수도 없기 때문에 해당 경우를 고려하여 에러 색상 지정 시작위치를 해당 라인의 시작점으로 지정해주고 시작해야한다.

2. 텍스트 찾기 및 바꾸기(EditController)

-찾기 및 바꾸기 후 하이라이팅 된 단어의 처리 (MyMouseEvent 내부 클래스)

문제 발생 원인:

텍스트가 적힌 메인뷰에 찾기 기능을 실행 후 조건에 맞는 단어를 파싱하여 하이라이팅한다. 바꾸기 기능 역시 하이라이팅 된 단어를 대상으로 진행하는데, 문제는 찾거나 바꾸기 기능을 실행 한 후 어떠한 입력을 통하여 하이라이팅을 풀어줘야 하는데 이에 대한 처리를 예상치 못 하였다.



해결 방법:

텍스트입력창에 마우스클릭 시 하이라이트를 제거한다. 마우스클릭 이벤트를 메인뷰에서 듣고 컨트롤러에 요청하면 처리하도록 구현하였다.

```
textPane.addMouseListener(new EditController.MyMouseEvent());
```

MainView

```
public static class MyMouseEvent implements MouseListener
{
    // 마우스가 해당 컴포넌트를 클릭했을때.
    @ badasskim
    public void mouseClicked(MouseEvent e) throws NullPointerException
    {
        if(MyActionListener.text_highlight != null)
        {
            MyActionListener.text_highlight.removeAllHighlights();
        }
    }
    @ badasskim
    public void mouseEntered(MouseEvent e) { }
    @ badasskim
    public void mouseExited(MouseEvent e) { }
    @ badasskim
    public void mousePressed(MouseEvent e) { }
    @ badasskim
    public void mouseReleased(MouseEvent e) { }
}
```

EditController

6. Conclusion

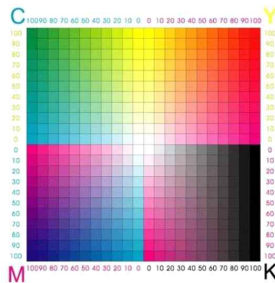
6-1. 향후 발전 및 보완 계획

1. C언어 문법 색상 지정 부분 보완 계획

-띄어쓰기의 의존도 줄이기

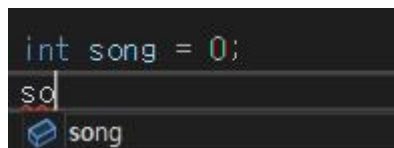
현재 나의 프로그램은 띄어쓰기의 의존도가 크다. 사용자의 편의성을 위해서 오류처리를 많이 해주었지만 Visual Studio와 같이 띄어쓰기를 하지않아도 모든 것을 구분해주고싶다.

-색상의 문제

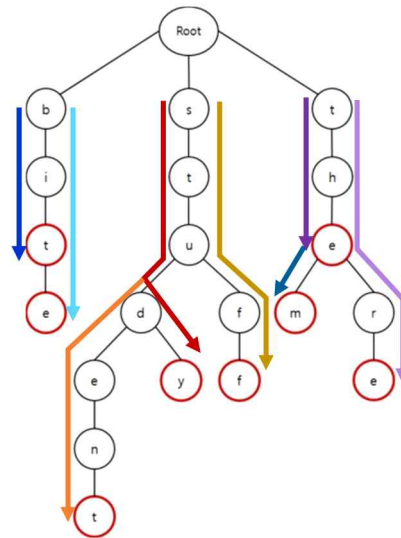


구현을 하면서 java.awt.Color 클래스를 사용하였는데 많은 컬러를 지원해주고있지않다. 또한 편집기의 배경이 흰색이라 Yellow같은 경우에는 잘 보이지않아 사용하지못하였다. 후에 시간이 남는다면 찾아보아서 색상을 좀 다양하게 지원해주고싶다.

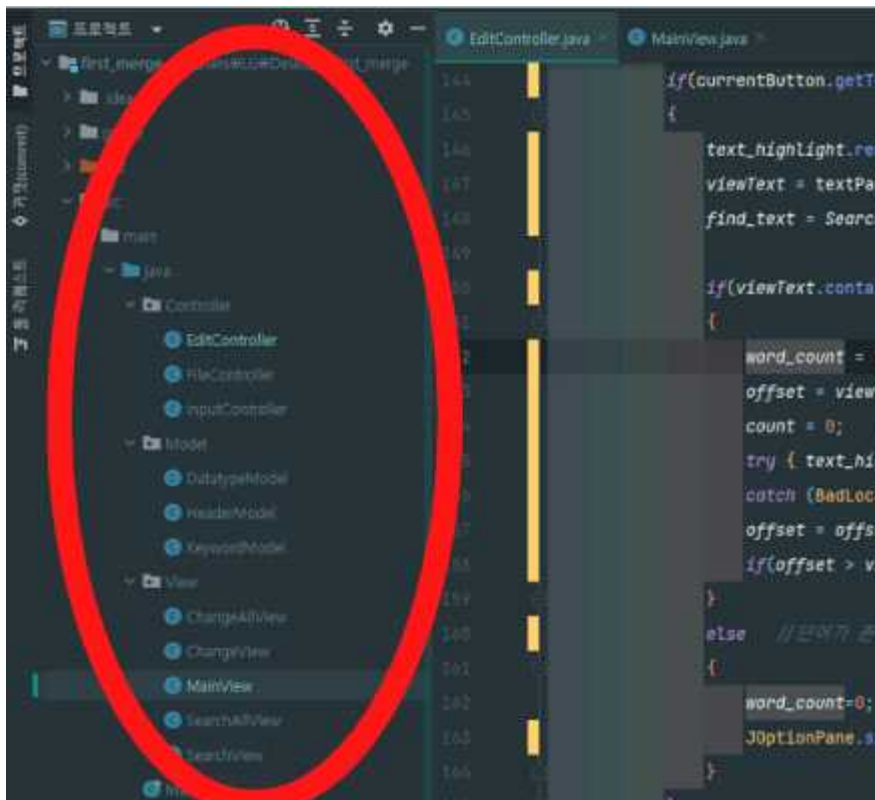
-사용자 편의성의 증가



Visual Studio를 사용해보면 굉장히 편리한 기능이있다. 바로 Tab을 이용한 자동완성이다. 이미 선언한 식별자나 함수의 경우 저장하고있다가 만약 해당 이름의 글자가 나온다면 사용자에게 추천하는 방식으로 구현한다면 굉장히 사용자에게 편리할 것이다. 해당 기능은 우리가 어드벤처디자인 수업을 하며 배운 자료구조인 트라이로도 구현해 볼 수 있다. 어드벤처디자인시간때 배운 트라이로 구현한다면 더 뜻 깊을 것으로 예상된다. 트라이로 구현해보고싶은 마음이 있다.



2. 파일 위치 탐색기



현재 학부생이나 현직 개발자들이 사용하고 있는 IDE나 텍스트 에디터에선 거의 모두 파일위치탐색기로 현재 작업중인 디렉토리를 확인하면서 구현한다. 현재 우리팀이 구현한 프로그램은 파일탐색기를 구현하지 않았지만 추후에 구현해보고 싶은 마음이 계속 생긴다.

3. Undo/Redo

원래는 구현이 되어있던 기능이었으나 코드를 합치면서 InputController와의 충돌로 인해 이번 프로젝트에서는 뺐다. 발생한 문제는 텍스트에 대해서 지정된 색깔만 사라지고 텍스트는 사라지지 않는 것이었다.

The UndoManager makes use of `isSignificant` to determine how many edits should be undone or redone. The UndoManager will undo or redo all insignificant edits (`isSignificant` returns false) between the current edit and the last or next significant edit. `addEdit` and `replaceEdit` can be used to treat multiple edits as a single edit, returning false from `isSignificant` allows for treating can be used to have many smaller edits undone or redone at once. Similar functionality can also be done using the `addEdit` method.

- 오라클 공식문서

이를 해결하기 위해 자바의 UndoManager 인터페이스의 `addEdit`과 `replaceEdit`을 오버라이딩하여 Undo/Redo의 단위를 지정하여 색깔과 텍스트를 모두 지우는 것에 성공하였으나, 쓰레드간에 충돌이 발생하여 프로그램이 제대로 동작하지 않았다.

멀티 쓰레드 환경을 고려하여 쓰레드 간 우선순위를 정하고 특정 조건에서 쓰레드를 wait했다가 notify해서 깨우는 것 까지 모두 고려하였으나 어떠한 에러인지 쓰레드가 종료되도 undo/redo가 되지 않았다. 이 부분에 대해서는 자세한 원인 분석을 토대로 구현을 해볼 예정이다.

4. 다양한 언어 문법 체크기능 지원 계획

숨 코드

언어

C++17

언어 설정

소스 코드 공개

☒ 공개

☐ 비공개

☐ 맞았을 때만 공개

소스 코드

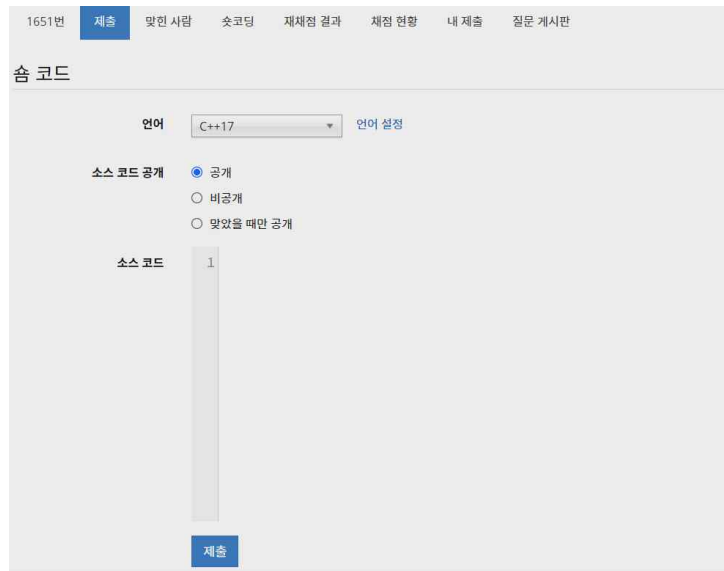
1

제출

결과물 활용가능성에서 제시했던 온라인 코딩테스트 에디터로써 사용되기 위해 현재 C언어 문법 검사만 가능한 C 소스 에디터에서 좀 더 다양한 프로그래밍 언어(Java/C++/Python 등) 문법 검사 기능을 제공할 수 있도록 발전한다면 사용자에게 좀 더 편의성을 제공할 수 있는 프로그램으로 성장할 것으로 예상된다.

6-2. 결과물 활용 가능성

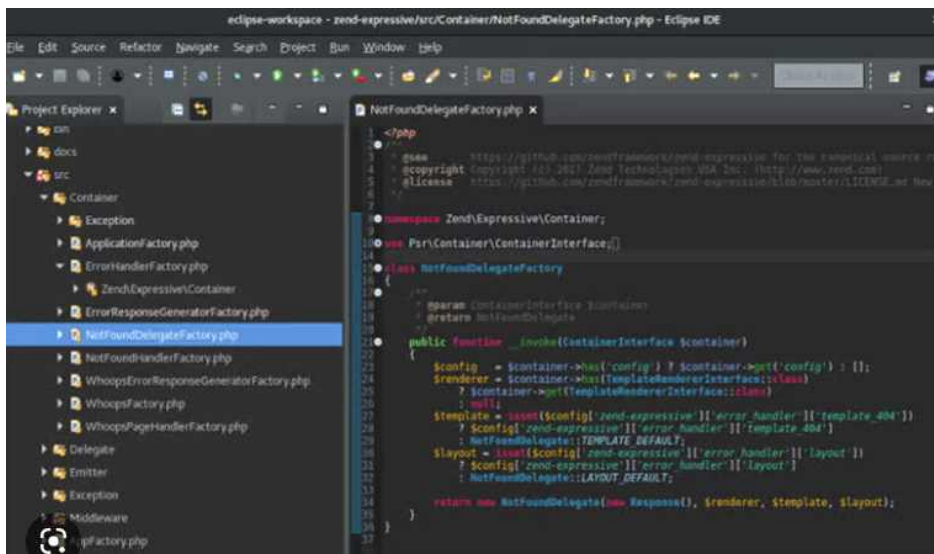
-온라인 코딩테스트 에디터 활용 가능성



깃허브나 타 오픈소스 플랫폼을 통해 .exe파일로 우리가 만든 C 소스 에디터 프로그램을 배포하는 방법 이외에도 백준이나 코드업과 같은 온라인 코딩테스트 사이트 혹은 C언어 코딩도장과 같은 온라인 코딩 학습 사이트 상에서 웹 에디터로 활용할 수 있을 것으로 보인다. 추가적으로 로컬이나 웹 뿐만아니라 안드로이드나 iOS와 같은 핸드폰 기반 OS 상에서도 돌아갈 수 있도록 관련 지식을 쌓아 구현을 하게 된다면 우리 프로그램의 확장성은 더욱 넓어질 것으로 예상된다.

이를 위해 java, python, c++과 같은 다양한 언어에 관해서도 문법 오류 여부를 체크할 수 있도록 보완해야 할 것으로 보인다.

-새로운 IDE 플랫폼으로의 활용 가능성



향후 발전 계획을 토대로 터미널 구현이나 UNDO/REDO, C언어 문법 색상 지정 늘리기와 같은 추가 기능을 구현한다면 새로운 IDE 플랫폼으로 활용이 가능할 것으로 보인다. 이 또한 하나의 언어, 즉 현재 C 언어 에디터로만 사용되는 것이 아닌 다양한 언어에 대한 문법 체크를 실행하여 좀 더 사용자가 폭 넓은 목적으로 사용할 수 있도록 만든다면 Visual Studio와 같은 대표적인 다목적 IDE로 사용되기에 적합할 것으로 예상된다.

6-3. 기술/경제/사회 파급 효과 및 기대효과

-소비자들에게 C 언어 개발 툴 선택지 증가



앞서 소개했던 터미널창, UNDO/REDO, C언어 문법 색상지정 늘리기와 같은 다양한 기능등을 프로그램에 제공하여 C언어 에디터가 필요한 소비자로 하여금 에디터 선택의 폭을 넓힐 수 있다. 좀 더 완성도 있게 프로그램을 발전시킨다면 현재 C언어 에디터로 유명한 VScode, Dev C++과 같은 완성도 높은 C언어 에디터를 배포할 수 있을 것이라 기대된다.

-오픈소스로 공개하여 경제적/사회적 효과 도모



GIT과 같은 다양한 오픈소스 플랫폼을 활용하여 프로그램 배포하여 C언어 에디터가 필요한 소비자들에게 무료로 C언어 에디터 기능을 사용할 수 있게 하여 경제적으로는 위 에디터가 필요한 소비자에게 적절한 기능을 제공하는 C언어 에디터를 무료로 제공할 수 있을 것이라 기대되며, 사회적으로는 경제적 여유가 없는 소비자에게도 학습기회를 제공하여 사람들로 부터 우리 프로그램에 대한 긍정적 인식효과를 기대할 수 있다.

6-4. 테스트 및 유지보수 프로세스

1) 테스트 방법

< 화이트 박스 테스트 >

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.

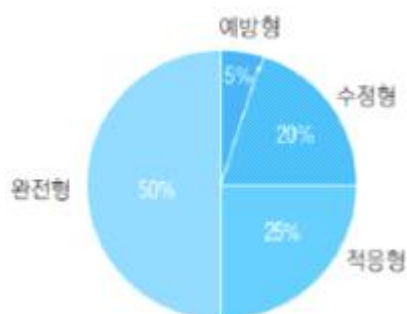
2) 테스트 방법 선정 이유

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.

3) 테스트 프레임 워크

C 소스 에디터 프로그램 작성을 java 언어로 작성을 하여서, 테스트 프레임 워크는 JUnit을 사용하였다. JUnit은 자바 기반의 단위 테스트 프레임워크로 같은 테스트 코드를 여러번 작성할 필요가 없어 코드 유지보수 및 시간 효율성을 극대화 할 수 있었다.

4) 유지보수 및 프로세스



[완전형 유지보수]

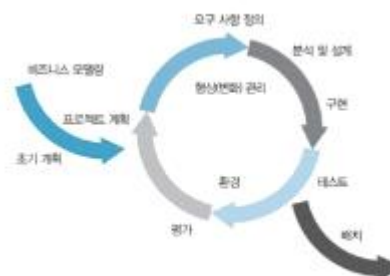


그림 2-19 통합 프로세스(UP) 방법

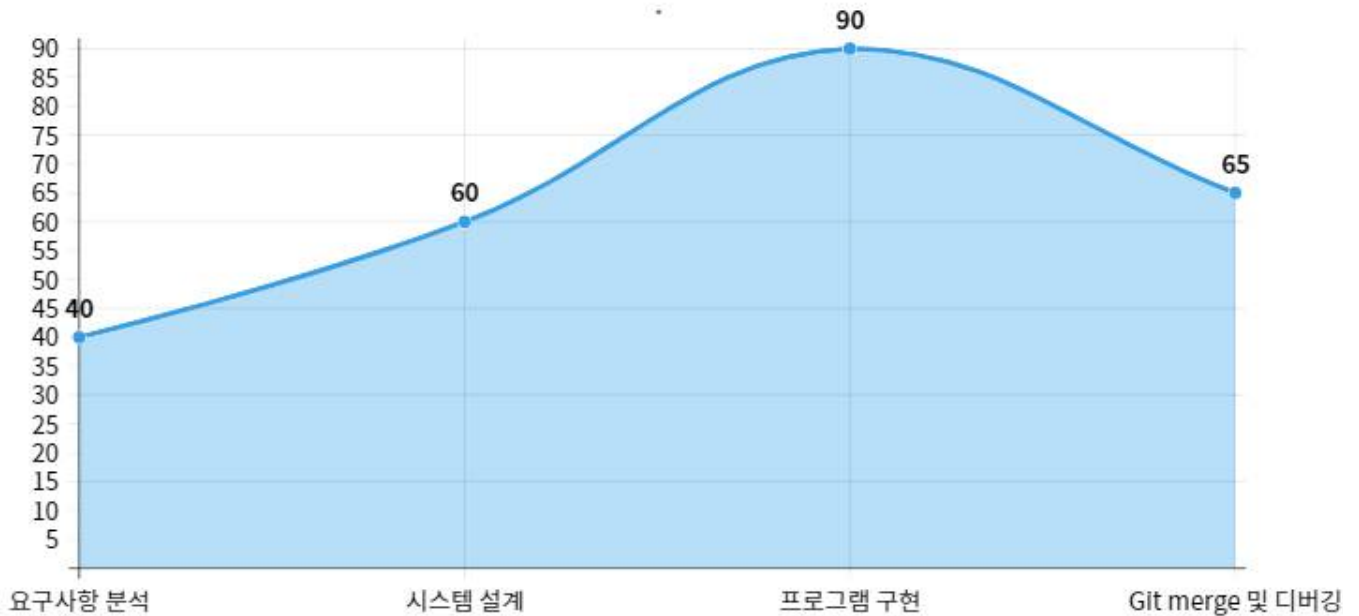
[통합프로세스(UP)]

- 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. 사용자에게 의한 기능 보강 요청에 의해 발생하는 유지보수로도 불린다.
- 유지보수 프로세스에는 UP(Unified Process)모델을 사용하여 진행하게 된다. 특히 우리 조가 선택한 개발방법론도 UP를 사용하는데, 이러한 UP 모델을 사용하여 유지보수를 진행하면 반복/점진적으로 유지보수를 하여 효율적인 유지보수가 가능하였다.

7. Ending

7-1. 프로젝트를 마치며

최종 프로그램 제작 소요시간



이번 프로젝트에선 구현과 디버깅에 가장 많은 시간을 투자하였다. 처음 Git을 사용한 것인데 사용법이 익숙치 않아 이 과정에서도 시간을 사용하였다.

7-2. 진행일정

(수행 기간 : 2022년 11월 23일 ~ 2022년 12월 20일)					
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

7-3. 역할분담

프로그램 구현 및 최종 보고서의 역할분담은 아래와 같이 담당하였다.

안현진	김현진	송제용	정지수
<p>최종보고서 작성 관련</p> <ul style="list-style-type: none"> - 문제 개요 및 정의와 목표 - 프로그램 전체 아키텍처 및 설계(클래스 다이어그램) - Result (행, 열 출력 및 파일저장, 불러오기 확인) - Conclusion : Undo/Redo 논의 <p>프로그램 구현 관련</p> <p>FileController구현 MainView 구현 Github 전체 관리</p>	<p>최종보고서 작성 관련</p> <ul style="list-style-type: none"> -Abstract 작성 -Results (EditController 부분 및 설계구성요소 및 제한요소 작성) -Discussion (EditController 부분 작성) -Conclusion(향후 발전 및 보안 계획 파일위치 탐색기 부분 작성) -최종 보고서 추합 및 작성 <p>프로그램 구현 관련</p> <p>EditController 구현 MainView와 내부 리스너들과 연동</p>	<p>최종보고서 작성 관련</p> <ul style="list-style-type: none"> - Related works 작성 - Methodology and Implementation 작성 (모듈의 알고리즘, 구현에 활용된 핵심 아이디어) -Results (InputController 부분 작성) -Discussion (InputController 부분 작성) -Conclusion (향후 발전 및 보안 계획 InputController 부분 작성) -일정 기획 - 보고서 최종 점검 <p>프로그램 구현 관련</p> <p>-Datatype, Header, Keyword 모델 구현 및 InputController 구현</p> <p>-프로그램 코드 최종 마무리</p>	<p>최종보고서 작성 관련</p> <ul style="list-style-type: none"> - Methodology and Implementation (Program Interface FlowChart 관련 작성) -Conclusion (기술/경제/사회 파급효과 및 기대효과 작성, 결과물 활용가능성 작성, 테스트 및 유지보수 부분 작성, 향후 발전 및 보안 계획 중 다양한 문법체크부분 작성) -PPT 검수 <p>프로그램 구현 관련</p> <p>Edit View 구현 및 메인과 연결 구현</p>