**과목명: 시스템프로그래밍**

**1 분반**

**<<Project #2>>**

**서강대학교 [컴퓨터공학과]**

**[20161559]**

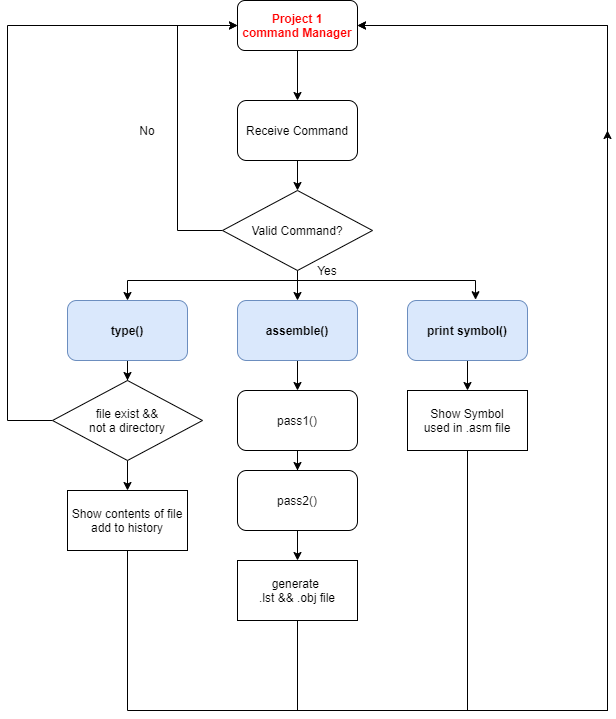
**[강민석]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
   2. 프로그램 파일구성
   3. 사용 라이브러리
   4. 사용자 정의 구조체
3. **모듈 정의**
4. **전역 변수 정의**
5. **코드 설명**
6. **프로그램 개요**

프로젝트 1에서 구현한 shell에 assemble 기능을 추가하는 프로그램으로, assembly 파일을 입력으로 받아 object 파일을 생성하고 그 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 구현한다.

1. **프로그램 설명**
   1. 프로그램 흐름도



프로그램의 흐름은 위와 같다. 프로젝트 1에서 구현한 while loop를 통해 명령어를 입력받고, 그 명령어가 type, assemble, symbol인 경우에 대해 적절한 결과를 출력한다. 우선 type이 명령어로 들어올 경우, 두 번째 인자(file 이름)이 현재 경로에 있으며 디렉토리가 아닌 경우 파일의 내용을 출력한다. assemble 명령어가 들어올 경우, pass1, pass2 알고리즘을 통해 .lst 파일과 .obj 파일을 생성한다. symbol이 명령으로 들어올 경우, assemble 명령어를 통해 생성된 symbol table에 저장된 symbol을 사전상 오름차순이 되도록 출력한다. 위의 명령어에 대해 문제 없이 함수가 종료되면 history에 해당 명령어를 삽입하고, 그렇지 않은 경우 history에 삽입하지 않고 다시 명령어를 입력받는 형태로 돌아간다.

* 1. 프로그램 파일 구성

구현한 기능은 20161559.c 파일과 20161559.h 파일로 구성되어있다. c 소스파일에서는 기본적인 프로그램의 흐름 등을 명시해두었고, 구체적인 함수 및 에러처리는 헤더파일에 작성하였다.

* 1. 사용 라이브러리

<stdio.h>

표준 입출력을 위한 라이브러리이다.

<string.h>

명령어 문자열 복사 및 조작, 문자열 길이 계산을 위해 관련 라이브러리를 사용하였다.

<stdlib.h>

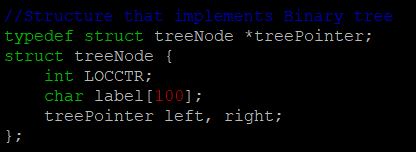
해쉬테이블 및 가상 메모리의 동적할당 및 해제 기능을 위해 사용하였다.

<dirent.h>, <sys/stat.h>

프로그램에서 디렉토리 및 파일 정보를 조회하는 기능을 하는 함수 dir()를 구현하기 위해 사용하였다.

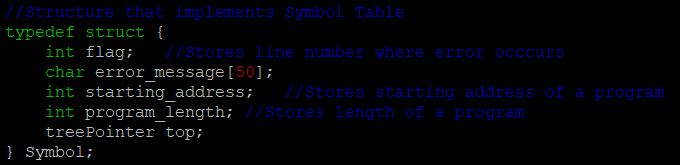
* 1. 사용자 정의 구조체

(1) treePointer



Symbol table의 symbol을 저장하기 위한 자료구조로, binary tree 형태로 구현하였다. label(symbol 문자열)과 해당 label이 나타나는 위치(location counter)를 포함하고 있다.

(2) Symbol



Symbol Table을 저장하는 자료구조로 위에서 기술한 treePointer를 기반으로 error가 발생하는 위치를 가리키는 flag, error 내용을 저장하는 error\_message, 프로그램의 시작주소 및 길이를 저장하는 구조체이다.

1. **모듈 정의**

정의한 모듈에 대해 간략적인 설명

**void help(void)** : 수행가능한 명령어들을 화면에 출력한다.

**void tokenize(char\*, char\*, char\*, char\*)** : 파일을 읽어들여 적당히 parsing 하는 함수로 하나의 input line으로부터 label, opcode, operand를 분리해내는 역할을 하는 함수이다.

**void tree\_cleaner(treePointer)** : 프로그램이 종료되거나 에러가 발생하는 상황등에 대해 만들어져있던 tree를 free 시키는 함수이다.

**treePointer search(treePointer, char\*)** : 해당 문자열이 tree에 있는지 검색 후, 있으면 해당 treePointer를, 그렇지 못하면 NULL을 반환하는 함수다.

**void insert\_node(treePointer\*, int, char\*)** : treePointer에 element를 삽입하는 함수이다.

**void type(char\*)** : 입력으로 들어온 파일의 내용을 화면에 출력한다.

**void assemble(char\*, char\*)** : .asm 파일을 assemble하여 .lst 파일과 .obj 파일을 생성한다.

**int pass1(char\*)** : assemble 함수에서 호출하는 첫 번째 알고리즘으로, symbol table을 업데이트하며 input file을 바탕으로 intermediate.txt 파일을 생성한다.

**int pass2(char\*)** : assemble 함수에서 호출하는 두 번째 알고리즘으로, pass1에서 생성된 intermediate.txt 파일을 바탕으로 lst 파일과 obj파일을 생성한다.

기본적으로 모든 모듈은 primary function과 secondary function으로 구분한다. primary function은 프로그램의 주 흐름을 담당하는 함수이며, secondary function은 primary function의 기능을 보조하는 역할로 문자열 처리, 메모리 free 등에 해당한다.

1. **전역 변수 정의**

**Symbol symbolTable :** symbol을 저장하기 위한 전역변수이다.

**treePointer temp\_symbol :** symbol을 임시저장하는 tree 전역변수로, assemble 명령 수행 도중 오류가 났을 경우 이전 symbol을 유지하게끔 하기 위한 기능을 한다.

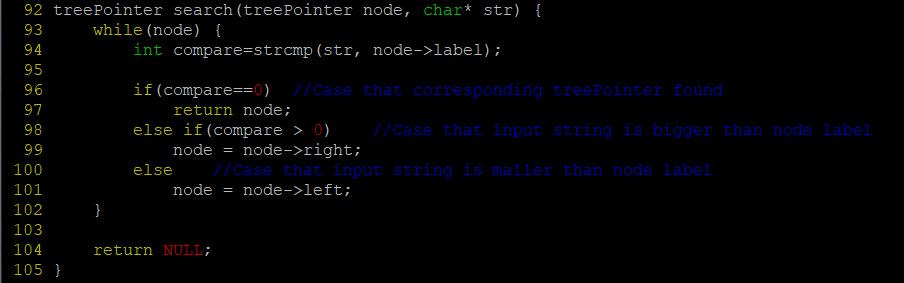
**int program lines :** 프로그램의 line 수를 저장하는 변수이다.

1. **코드 설명**

**- 20161559.h (20161559.c는 proj1과 같다.)**

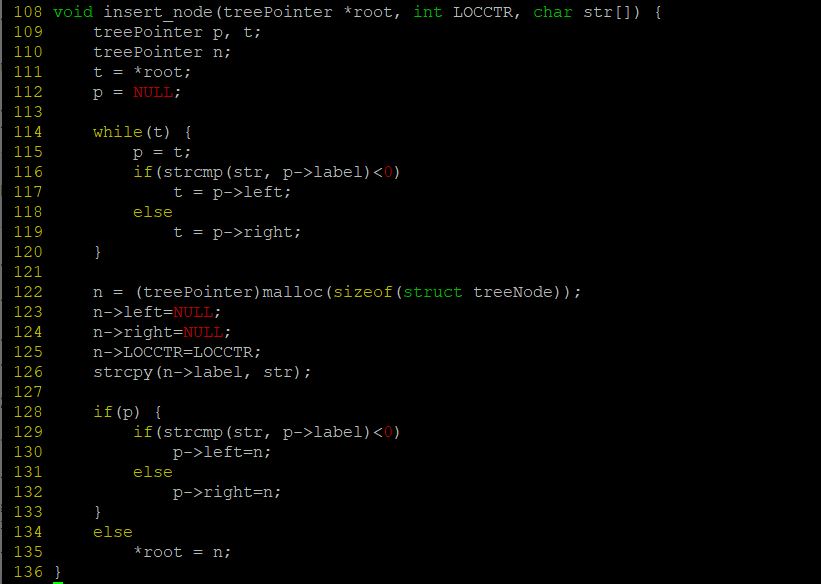
**(1) Secondary function(Primary function의 기능을 보조하는 함수 코드)**

**- treePointer search(treePointer, char\*);**

****

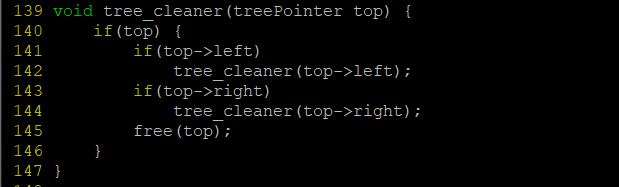
입력으로 들어온 treePointer가 NULL 값이 아니라면 반복문이 진행된다. 입력으로 들어온 두 번째 parameter인 str의 값과 node의 label값을 strcmp한 결과에 따라 그 값이 같으면 해당 treePointer를 반환하고 그렇지 않은 경우, 만약 node->label이 str보다 사전상 앞에 오게되면 오른쪽을 확인하고 그렇지 않은 경우 왼쪽을 확인한다. (Binary search tree 형태로 구현하였기 때문에 위와 같은 방법이 가능하다.)

**- void insert\_node(treePointer\*, int, char\*);**



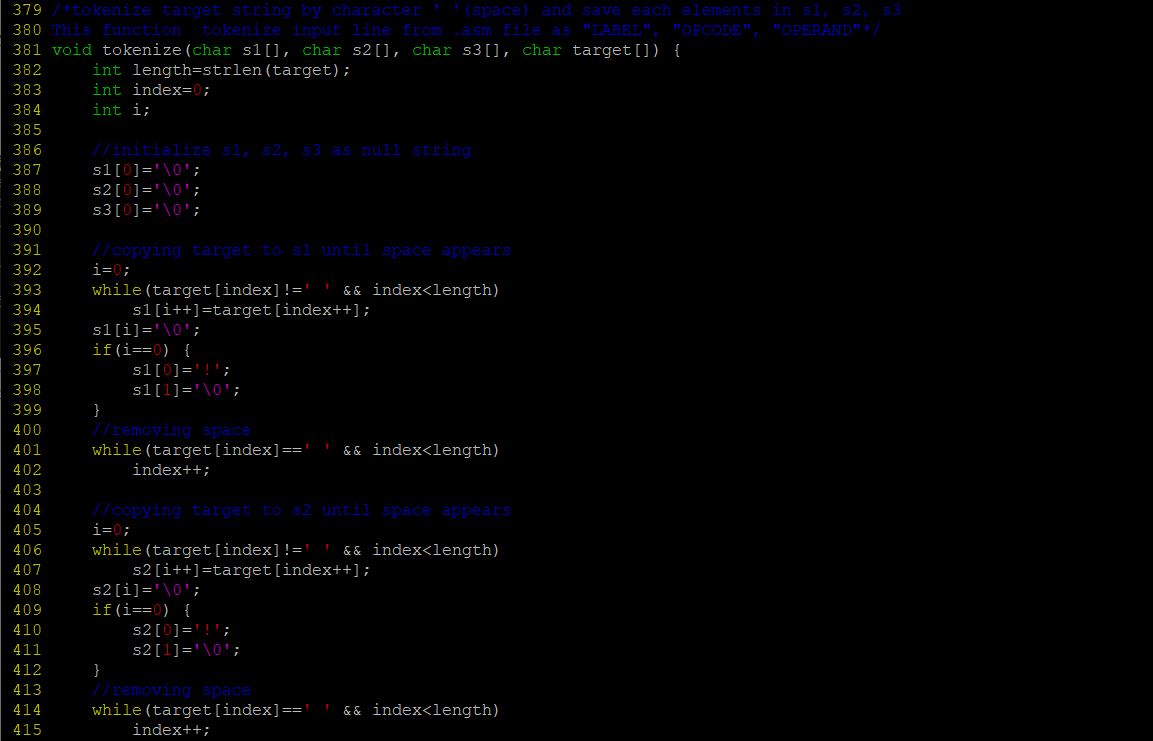
Binary Search Tree에 element를 삽입하는 함수라고 할 수 있다. 3번째 parameter인 str과 node의 label 값을 비교하여 지속적으로 p와 t를 바꾼 후, 적합한 위치에 포인터로 연결하는 구조이다. 만약 비어있는 tree였다면 (\*root) = n식을 통해 연결한다.

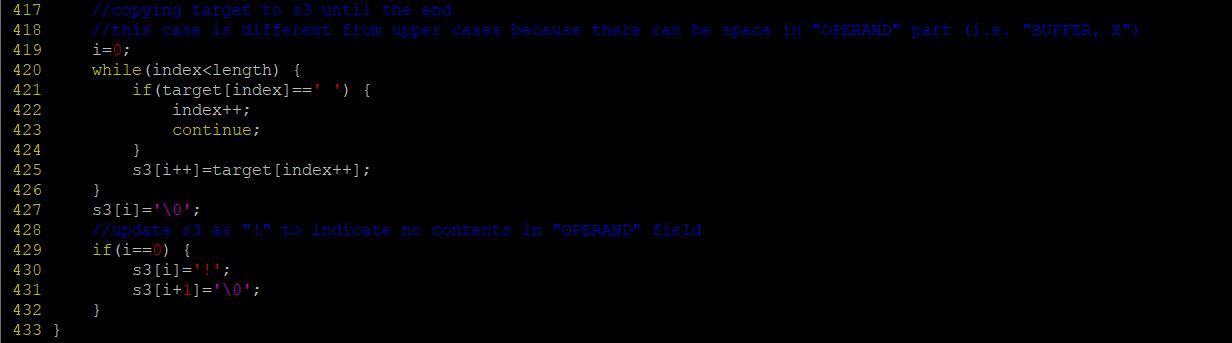
**- void tree\_cleaner(treePointer);**

****

프로그램의 종료, 각종 에러 발생시 호출되는 함수로 생성되었던 tree를 free 해준다. 재귀적으로 구성되어있으며 해당 treePointer에서 left가 NULL이 아닌 경우 tree\_cleaner(left), right가 NULL이 아닌 경우 tree\_cleaner(right)를 수행한 후 free(treePointer)를 해준다.

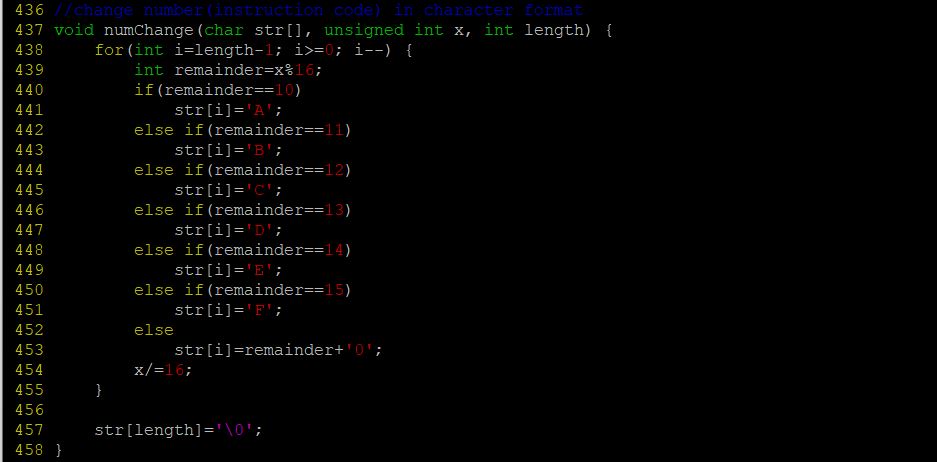
**-void tokenize(char\*, char\*, char\*, char\*);**





tokenize 함수를 통해 입력으로 들어온 문장으로부터 LABEL, OPCODE, OPERAND를 분리해낸다. s1, s2, s3는 각각 label, opcode, operand를 나타내며, target은 input으로 들어온 문장 한 줄을 의미한다. 우선 기본적으로 공백을 기준으로 구분되므로 공백이 없어질 때 까지 index를 반복문을 통해 증가시키고, 공백이 아닌 문자가 나오면 하나씩 s1, s2, s3에 저장하는 역할을 한다. 이 때, operand의 경우 예외적으로 BUFFER, X 와 같이 operand 파트 자체에 공백이 있는 경우도 있으므로 문장의 끝까지 복사하는 쪽으로 구현하였다. 또한 s1, s2, s3에 아무것도 저장되지 않는 경우, 이후에 편의를 위해 ‘!’ 문자를 통해 비어있음을 표현하였다.

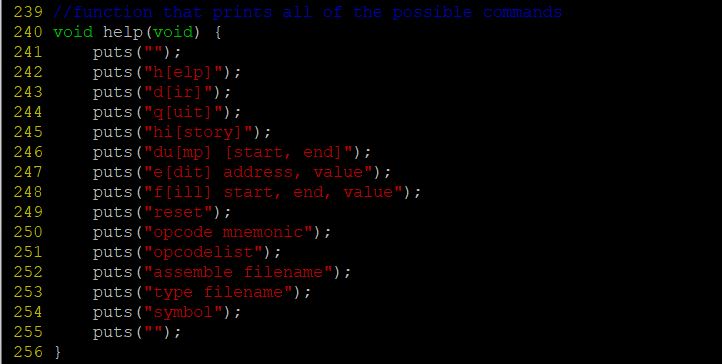
**-void numChange(char\*, unsigned int, int);**



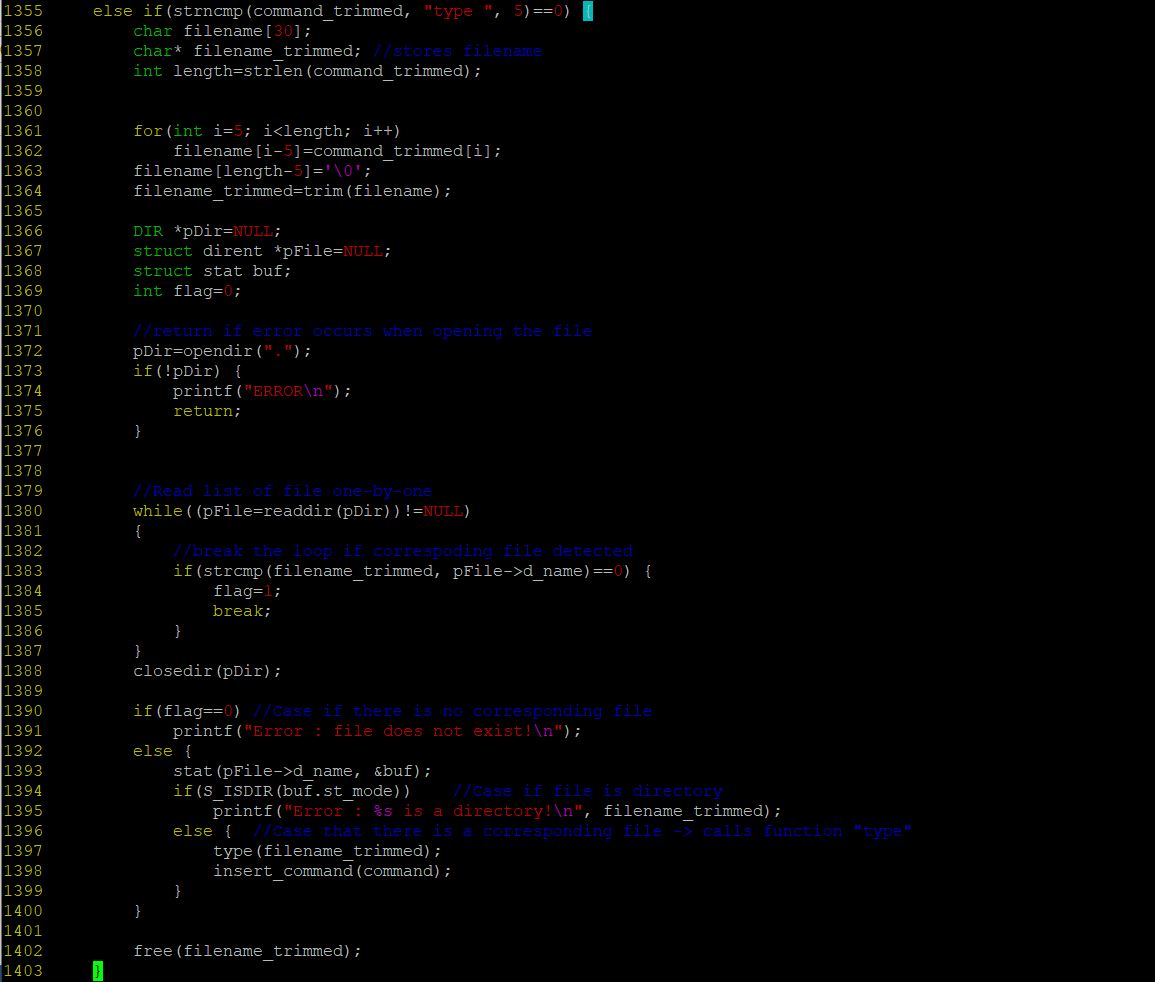
위 함수는 이후 pass2에서 obj파일을 작성할 때 instruction code가 정수인 것보다 문자열인 경우 훨씬 간단히 작동하므로 정수로 표현된 instruction code를 문자열로 바꿔주는 기능을 한다. 3번째 parameter length는 문자로 표현될 경우의 길이를 나타낸다. 예를 들어 FF라는 instruction code를 length=4인 문자열로 바꾸기 위해 위 함수를 호출하면 “00FF”가 되어 str에 저장된다. 이는 format에 따라 instruction code의 길이가 다르기 때문에 구현하였다. x를 16으로 나눈 나머지를 끝에서부터 차례대로 저장하는 방식으로 구현하였다.

**(2) Primary function(프로그램의 주 흐름을 담당하는 함수)**

**- void help(void)**

****

실행 가능한 명령어들을 출력하는 부분으로 project1과 비교했을 때 3가지 내용이 더 추가되었다. (line 252 ~ 254)

**- type 관련 command Manager 부분 기능 구현**

**line 1355** : command를 입력 받았을 때 앞 부분이 “type “와 매치되는 조건이다.

**line 1356 ~ 1369** : 명령의 입력으로 들어온 부분 중 type의 뒷부분에 해당하는 파일이름을 filename\_trimmed라는 char 배열에 저장하고, 해당이름을 갖는 파일이 존재하는 지 확인하기 이전에 각종 변수 등을 선언하고 준비하는 작업이다.

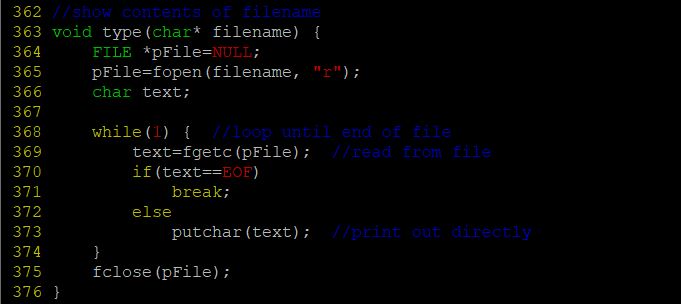
**line 1372 ~ 1376** : 현재 디렉토리를 open하는 과정 중 에러가 발생하는 지 확인한 후 에러가 발생하면 간단한 에러 메시지를 출력하고 return 한다.

**line 1380 ~ 1387** : 차례대로 디렉토리에 있는 파일들을 순회하면서 filename\_trimmed와 같은 이름을 갖는 파일이 있는지 확인한다. 발견되면 flag변수를 1로 설정한다.

**line 1390 ~ 1391** : 파일이 존재하지 않으면 에러메시지를 출력한다.

**line 1392 ~ 1403** : 파일이 존재하면 추가적으로 해당 파일이 디렉토리인지 확인한 후, 디렉토리 이면 파일이 아니라는 에러메시지를 출력하고 디렉토리가 아니라면 type 함수를 호출한다.

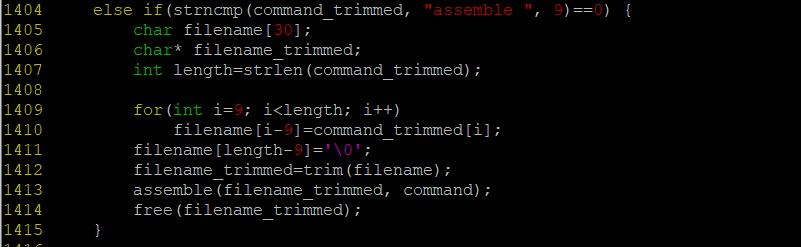
**-void type(char\*);**



**line 364 ~ 366** : FILE 포인터를 이용해 filename을 갖는 파일을 읽기모드로 open한다. text 변수는 파일로부터 한 글자 읽은 후, 한 글자를 출력하기 위한 변수이다.

**line 368 ~ 375** : pFile이 파일의 끝에 도달할 때 까지 한 문자를 읽어 text에 저장하고, text가 EOF가 아닌 경우 화면에 출력하는 기능을 한다. 반복문이 끝나면 열어두었던 파일 포인터를 fclose를 통해 닫아준다.

**- assemble 관련 command Manager 부분 기능 구현**

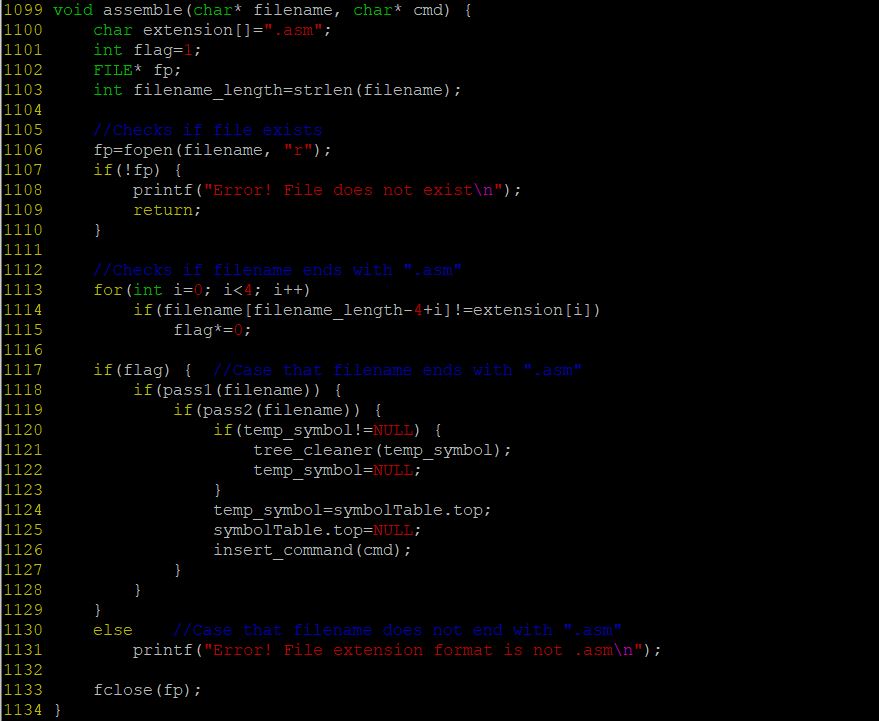


line 1404 : 명령어의 앞부분이 “assemble “와 일치하는 조건이다.

line 1405 ~ 1407 : 명령어의 뒷부분에서 파일이름을 추출하기 위한 변수선언 부분이다.

line 1409 ~ 1414 : 명령어로부터 filename을 extract하고, 해당 filename에 대해 assemble 함수를 호출한다.

**- void assemble(char\*, char\*);**



**line 1100** : 확장자명(filename의 끝부분)이 “.asm”으로 끝나는지 확인하기 위한 변수이다.

**line 1101** : 파일의 끝부분 4자리가 “.asm”과 하나라도 일치하지 않으면 0, 그렇지 않으면 1을 flag변수에 저장한다.

**line 1106 ~ 1110** : filename을 갖지 않는 경우 에러메시지를 출력하고 함수를 종료한다.

**line 1113 ~ 1115** : filename이 “.asm”으로 끝나는지 비교한다.

**line 1117 ~ 1129** : flag==1인 경우(filename이 “.asm”으로 끝나는 경우) pass1함수를 수행하고, 그 결과가 1인 경우 pass2를 수행한다. pass2도 1을 반환하게 되면 pass1, pass2가 성공적으로 이루어졌다는 뜻이며 line 1120 ~ 1126은 symbol table을 업데이트 하는 과정이다. pass1, pass2를 통해 symbolTable.top이 업데이트 된 후, temp\_symbol이 symbolTable.top을 가리키도록 한다. 이후 assemble 명령어를 수행 도중 pass1 또는 pass2에서 오류가 발생했을 경우, symbol 명령어가 입력으로 들어왔을 때 이전 symbol을 가리키도록 하기 위해 temp\_symbol을 이용한다. 모든 과정이 끝나면 insert\_command를 통해 history에 추가한다.

**- int pass1(char\* filename);**



pass1 함수는 filename을 input으로 받아 intermediate.txt파일을 생성하고, symbol table을 생성하는 것이 최종목표이다.

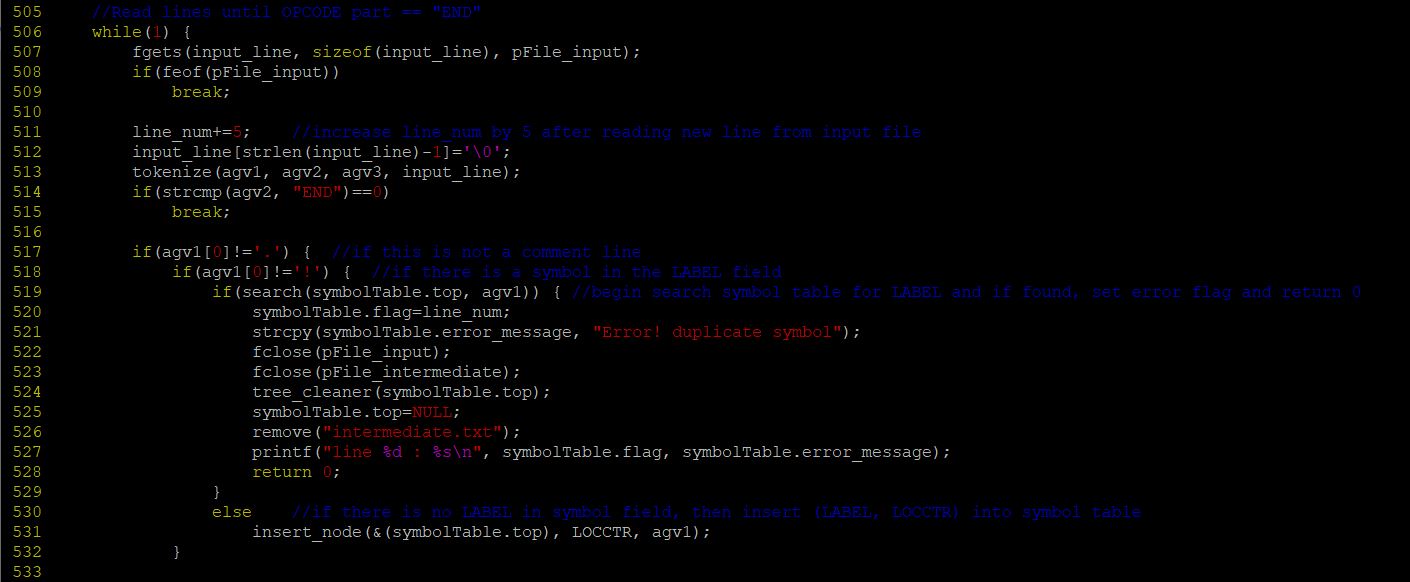
**line 462 ~ 471** : pass1에서 사용될 변수들의 목록이다. **LOCCTR**은 Location Counter로, 명령어가 수행되는 위치를 나타낸다. **PC**는 Program counter로 명령이 수행된 직후의 Location counter 값을 나타낸다. **line\_num**은 명령이 수행될 때의 행 위치로 5의 배수 단위로 증가한다. **pFile\_input**은 filename의 이름을 갖는 파일을 읽기모드로 open하기 위한 FILE 포인터이다. **pFile\_intermediate**는 intermediate.txt라는 이름을 갖는 파일을 생성하기 위한 파일포인터이다. **input\_line**은 filename으로부터 한 줄 씩 읽어 저장하는 char형 배열이다. **agv1**, **agv2**, **agv3**는 각각 input\_line으로부터 tokenize 함수를 이용해 label, opcode, operand를 저장하는 char형 배열이다.

**line 473 ~ 481** : 각종 파일을 open 하고 변수를 적절히 초기화한다.

**line 483 ~ 485** : filename파일로부터 한 줄 읽어들인 후, tokenize를 통해 label, opcode, operand저장한다.

**line 488 ~ 494** : opcode 부분이 “START”인 경우 symbolTable의 시작 주소를 operand로 초기화하고, 그렇지 않은 경우 location counter를 0으로 초기화한다.

**line 497 ~ 502** : intermediate.txt 파일에 읽어들인 정보를 저장한다. 총 6가지 정보를 한 줄에 하나씩 저장하는데 line 위치, label, opcode, operand, PC, LOCCTR가 해당된다.



**line 506** : while loop를 통해 지속적으로 filename으로부터 문장을 읽어 들인다.

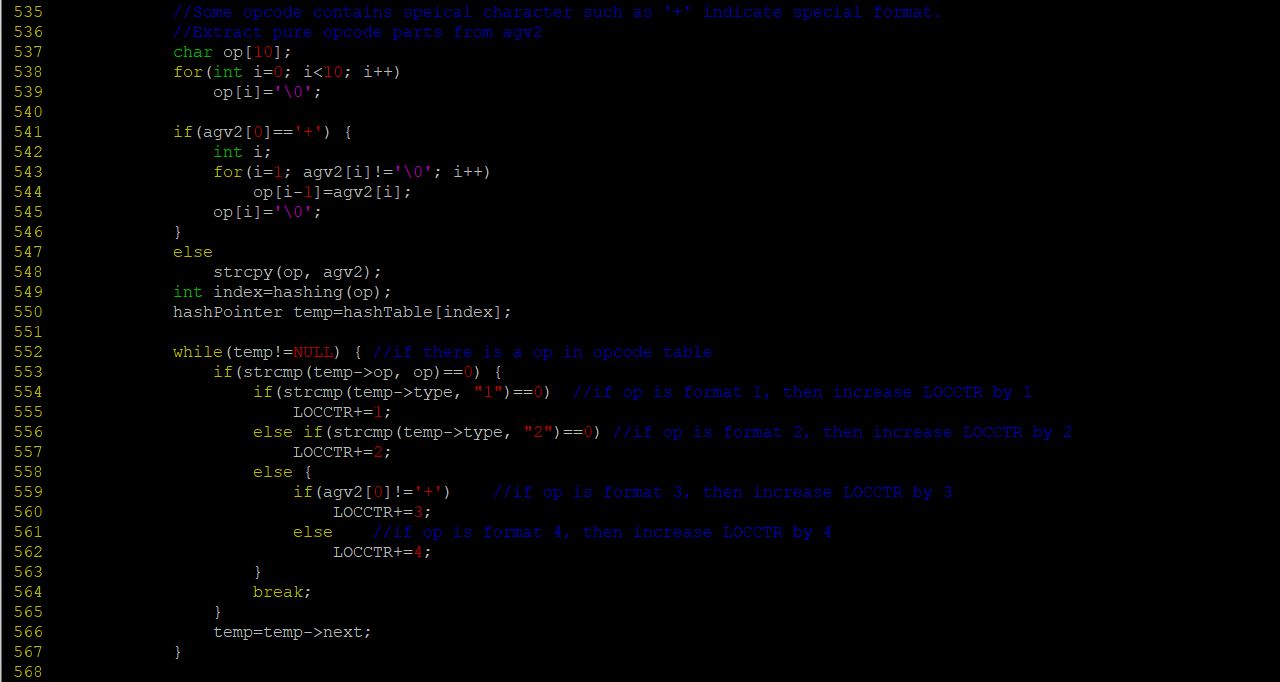
**line 514** : opcode가 “END”인 경우 반복문을 break한다.

**line 517** : label 부분이 comment가 아닌 경우

**line 518** : label 부분이 비어있는 상태가 아닌경우 (‘!’는 비어있음을 의미한다.)

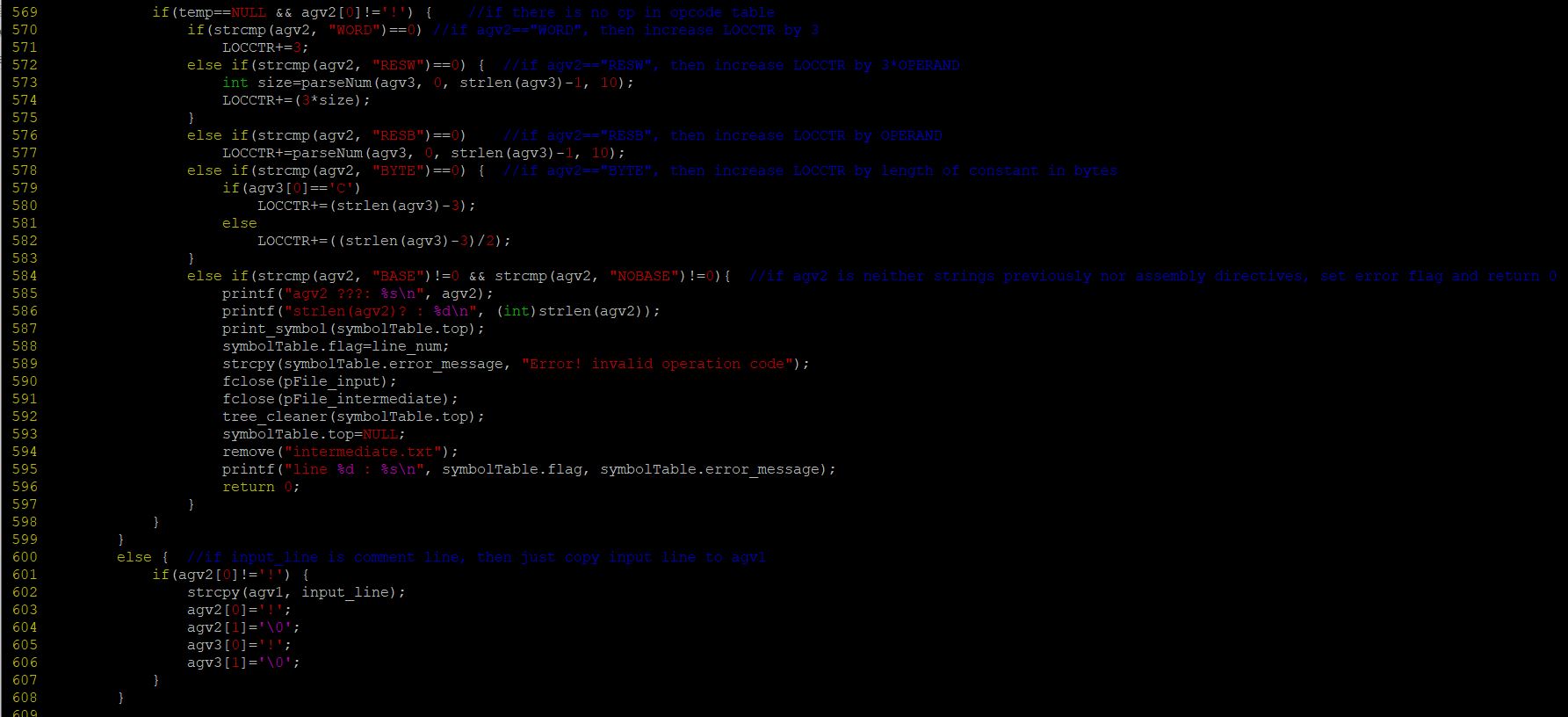
**line 519 ~ 530** : Symbol Table에 해당 label이 있는지 확인 후, 있으면 각종 에러 처리 후, pass1 함수를 종료한다. 이는 .asm에 중복된 label이 있음을 의미하여 assemble 할 수 없음을 의미한다.

**line 530 ~ 531** : Symbol Table에 해당 label이 없어 정상적으로 label을 symbol table에 집어 넣을 수 있음을 의미한다.



**line 537 ~ 550** : opcode 부분 관련하여 +JSUB와 같이 특수한 기호가 앞에 붙는 경우가 있다. 이와 관련하여 특수 문자가 붙는 경우 op 배열을 통해 JSUB만 추출해내는 작업을 해주고 이후에 hashing을 이용하여 해당 opcode가 있는지 hashTable 탐색 준비를 한다.

**line 552 ~ 567** : hashTable 탐색을 진행한다. 만약 op와 일치하는 operation이 hashtable에 있을 경우, opcode의 format에 따라 LOCCTR을 format1인 경우 1, format2인 경우2, format3인 경우 3, format 4인경우 4를 증가 시킨 후 반복문을 빠져 나온다.



**line 569** : opcode가 operation table에 없는 경우

**line 570** : opcode 부분이 WORD인 경우 LOCCTR을 3 증가시킨다.

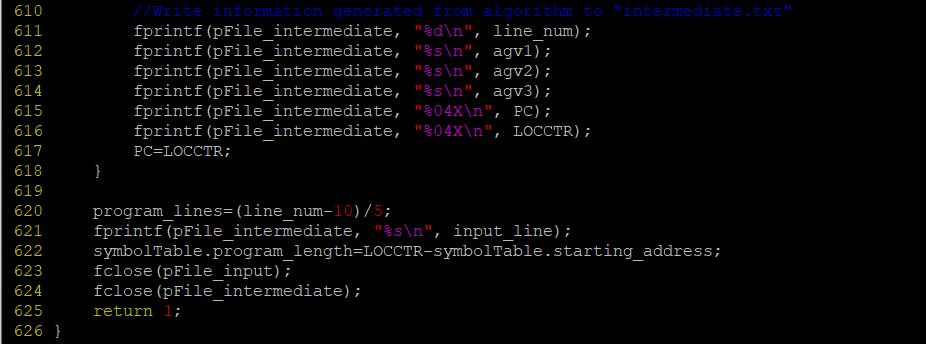
**line 752** : opcode 부분이 RESW인 경우 LOCCTR을 3\*(크기) 만큼 증가시킨다.

**line 576** : opcode 부분이 RESB인 경우 LOCCTR을 (크기) 만큼 증가시킨다.

**line 578** : opcode 부분이 BYTE인 경우 C’ ‘와 같은 char형 상수를 저장하는 경우 배열의 길이만큼 증가시키며, X’ ‘와 같이 16진수를 저장하는 경우 그 길이를 2로 나눈 만큼(16진수 하나는 half byte이므로) 저장한다.

**line 584 ~ 596** : opcode 부분이 assembler directive도 아닐 경우, 각종 에러를 세팅하고 pass1 함수를 종료한다.

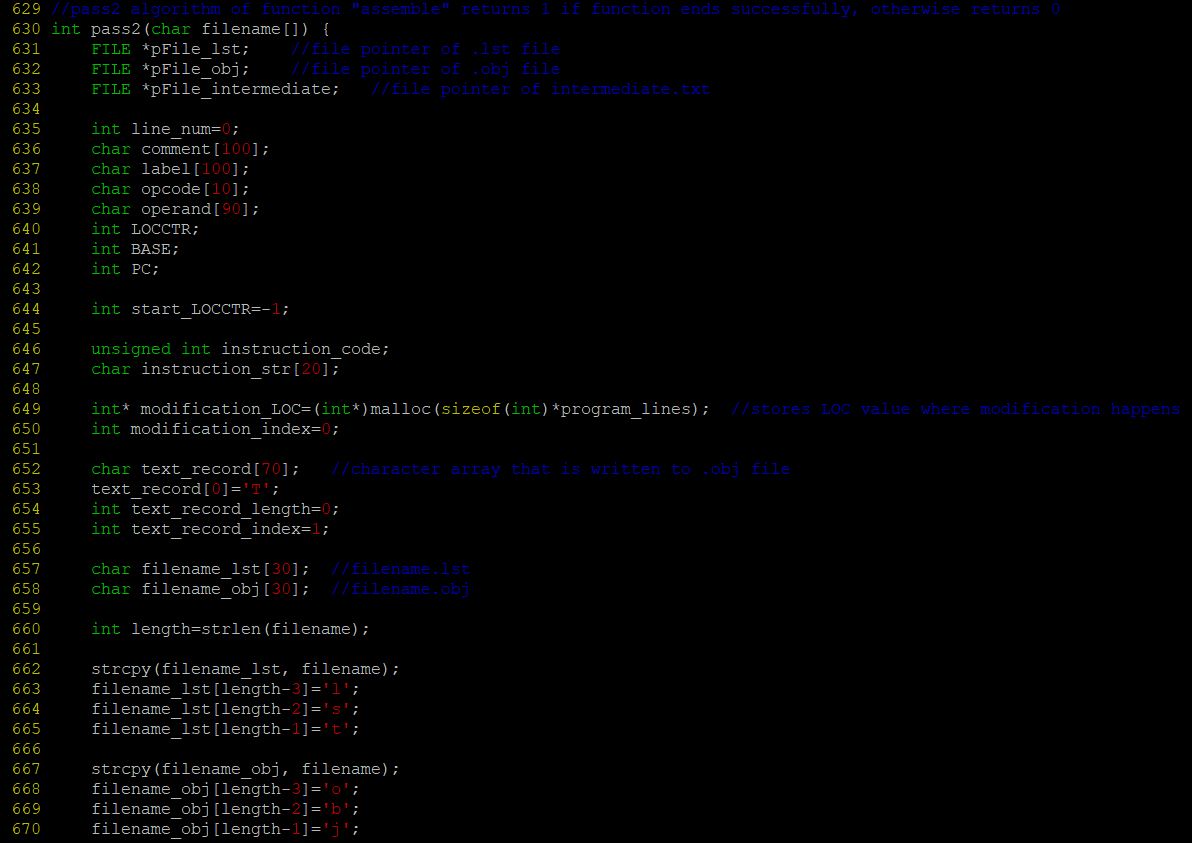
**line 600 ~ 606** : comment인 경우, agv1에 그 내용을 복사한다.



**line 611 ~ 616** : 저장된 내용들을 intermediate.txt에 write하는 과정이다.

**line 620 ~ 626** : while loop를 빠져 나온 후, 정상적으로 pass1이 종료되었음을 의미하므로 1을 반환하고 함수는 종료된다.

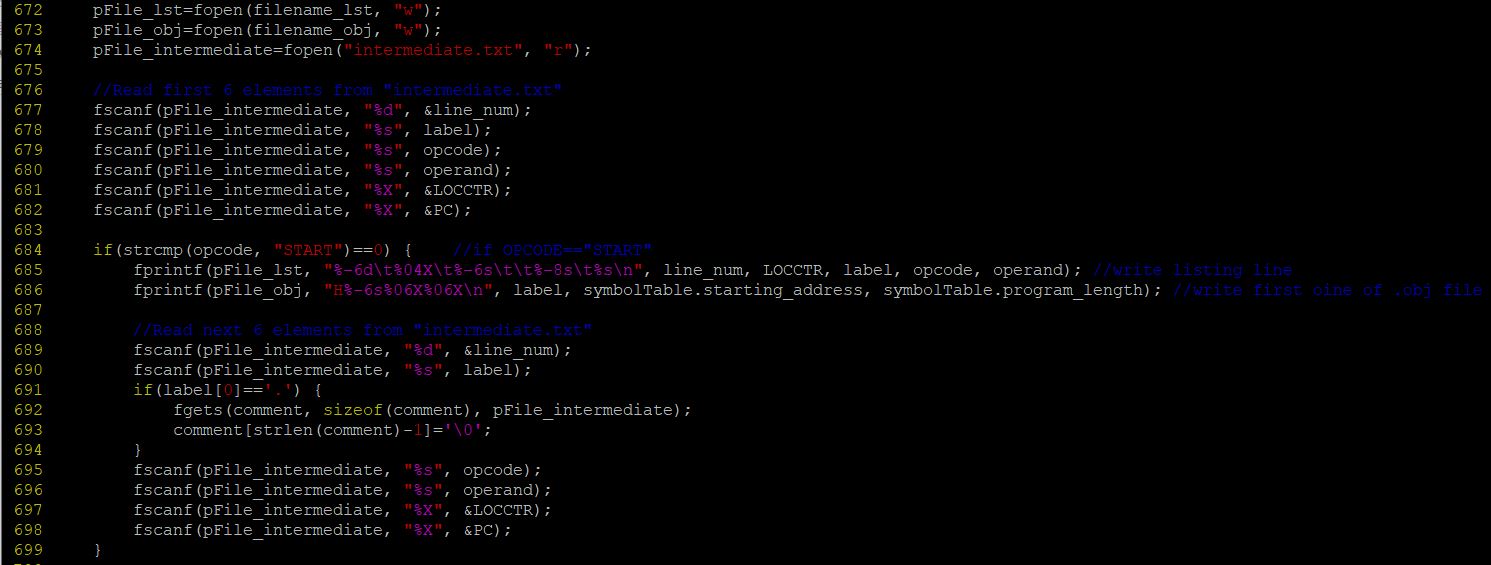
**- int pass2(char\* filename);**

****

**line 631 ~ 633** : **pFile\_lst**는 lst파일을 작성하기 위한 파일 포인터이다**. pFile\_obj**는 obj파일을 작성하기 위한 파일 포인터입니다. **pFile\_intermediate**는 intermediate.txt로부터 내용을 읽어들이기 위한 파일포인터이다.

**line 635 ~ 642** : pFile\_intermediate로부터 읽어들인 내용을 저장하는 변수들의 집합이다. **line\_num**은 명령어의 줄 위치를, **comment**는 주석을 읽어들였을 경우 저장할 char형 배열, **label**의 경우 label을, **opcode**의 경우 opcode를, **operand**의 경우 operand를 저장하며 **LOCCTR**은 해당 명령어의 location counter, **BASE**는 BASE 지시자가 기리키는 값을 저장하고, **PC**는 LOCCTR에 대응되는 명령이 수행된 이후의 LOCCTR을 가리킨다.

**line 644 ~ 658** : **start\_LOCCTR**은 프로그램이 시작하는 location counter, **instruction\_code**는 명령이 수행되는 최종 opcode를 가리키고, **instruction\_str**은 명령이 수행되는 opcode의 char형 변환형태이다. **modification\_LOC**은 SIC/XE에서 지원하는 modification이 필요한 명령어의 위치를 저장하는 기능을 한다. **text\_record**는 obj파일에 써질 한 줄을 의미한다. **filename\_lst**와 **filename\_obj**는 filename의 확장자를 lst와 obj형태로 바꾼 것이다.



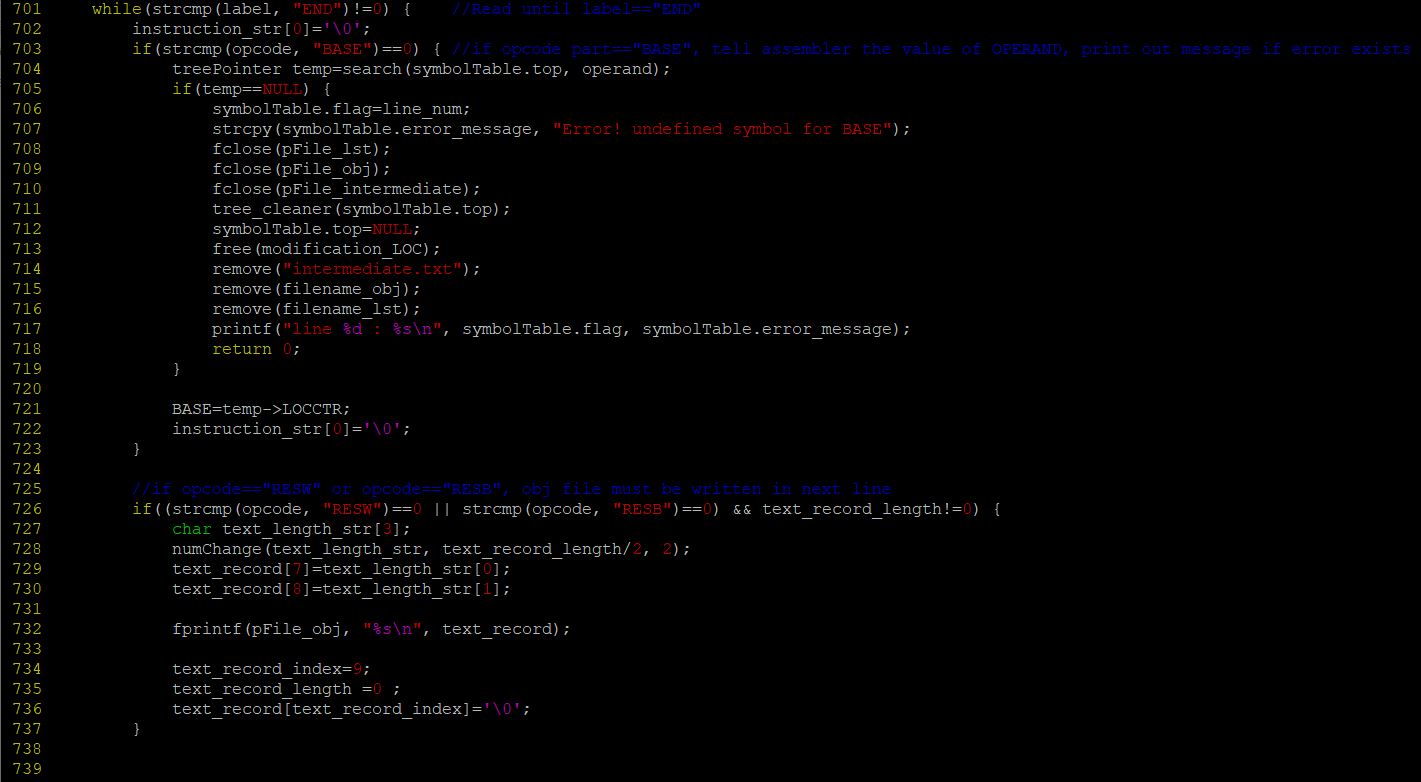
**line 672 ~ 674** : pass2에 선언된 파일포인터들을 이용해 intermediate.txt파일을 읽기모드로 open하고, filename.lst와 filename.obj를 쓰기모드로 open한다.

**line 677 ~ 682** : intermediate.txt로부터 asm파일의 첫 번째 줄에 해당되는 정보들을 읽어들인다.

**line 684** : opcode가 START인 경우

**line 685 ~ 686** : lst와 obj파일에 각각 관련 정보들을 write한다.

**line 689 ~ 699** : 다음 줄에 해당되는 정보들을 읽어들인다. 692번째 줄에 해당되는 부분은 읽어들인 내용이 주석일 경우에 대한 처리이다.



**line 701** : 읽어 들인 내용이 END이기 이전까지의 while loop이다.

**line 703 ~ 719** : 읽어들인 내용에 BASE assembler directive가 있는 경우 해당 부분의 operand를 검사하여 operand가 잘못된 operand면 관련하여 에러처리를 하고 0(비정상적인 함수의 종료)을 return 한다.

**line 721 ~ 723** : 그렇지 않은 경우 BASE에 operand의 location counter를 저장한다.

**line 726 ~ 737** : opcode 부분이 RESW 또는 RESB인 경우 obj파일은 다음 line에 작성되어야 하므로, text\_record에 저장되었던 내용을 obj파일에 write하고 text\_record관련된 변수들을 초기화한다.



**line 740 ~ 743** : opcode를 생성하기에 앞서 필요한 변수들을 선언한다. 해당 opcode 가 어떤 형식인지를 저장하는 **format** 변수, X register를 이용하는지 여부를 판단하는 **flag\_X**변수(사용할 경우 1로 세팅하고, 그렇지 않으면 0으로 세팅한다.), addressing mode가 simple인지(0), indirect인지(@), immediate(#)인지를 판단할 수 있는 **mode** 변수가 있다.

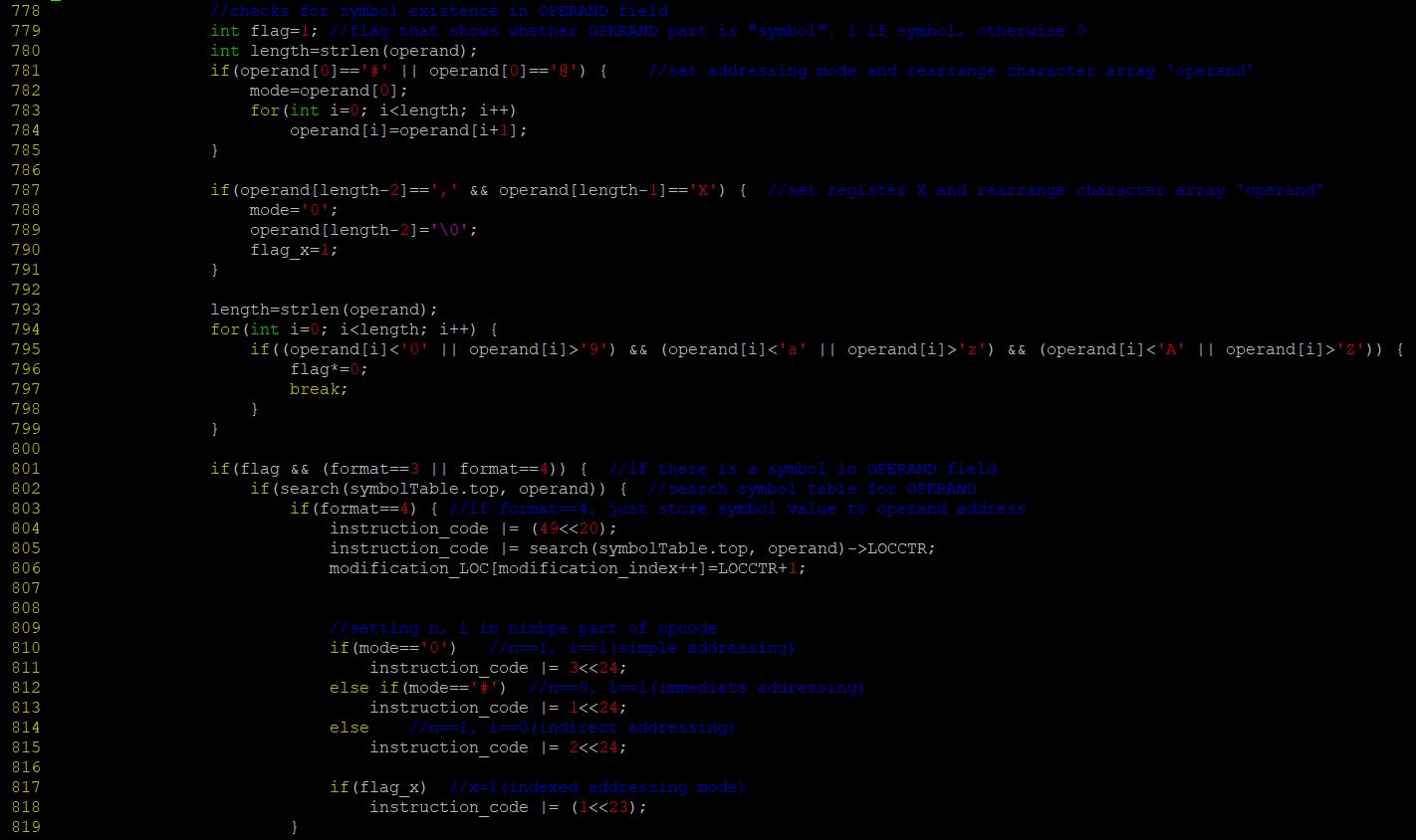
**line 746** : 읽어들인 내용이 주석이 아닌경우

**line 747 ~ 752** : opcode가 +로 시작하여 opcode 추출이 필요한 내용을 구현한 내용이다. format을 4로 지정해주고 한 칸 씩 앞으로 이동하여 opcode를 정리한다.

**line 753 ~ 759** : opcode가 OPTABLE에 있는지 탐색을 한다.

**line 760** : opcode가 OPTABLE에 있는 경우

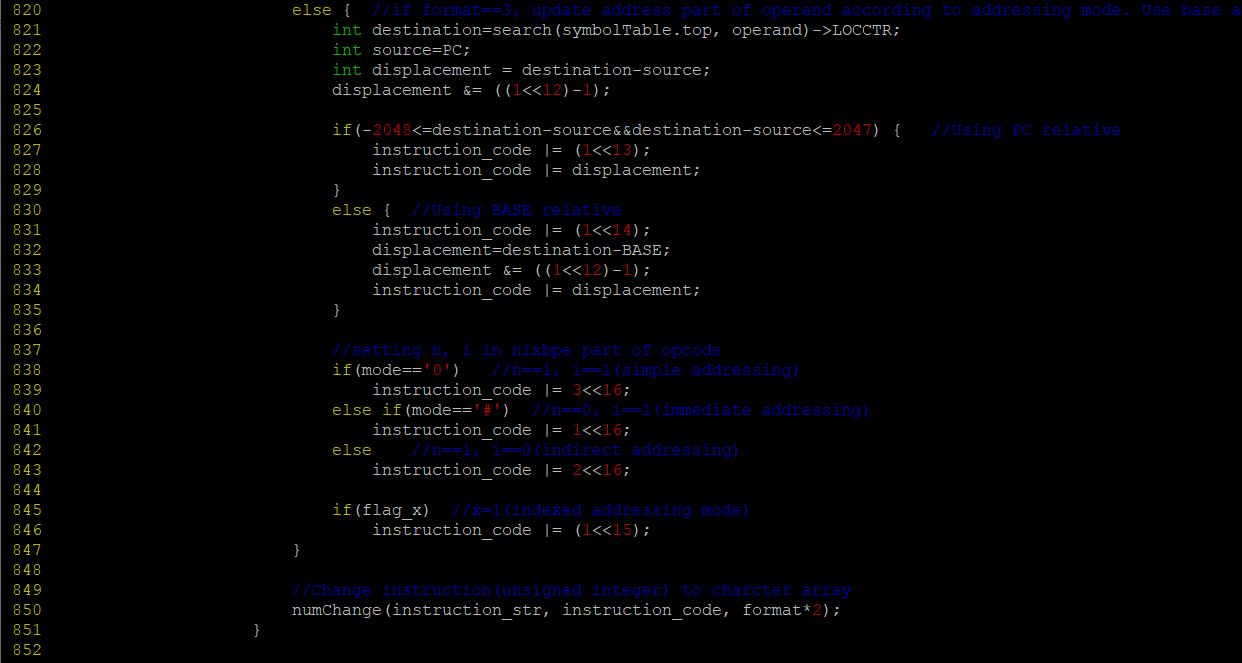
**line 761 ~ 776** : 각각 테이블에 저장된 type을 확인하여 format을 지정한다.



**line 779 ~ 785** : operand 부분이 symbol인지 판단하기에 앞서 operand에 있는 addressing mode를 char 변수 mode에 저장하고 operand를 정리하는 부분이다. ‘#’, ‘@’와 같은 경우 mode에 그대로 ‘#’, ‘@’를 저장하고 operand를 하나 씩 앞으로 저장한다.

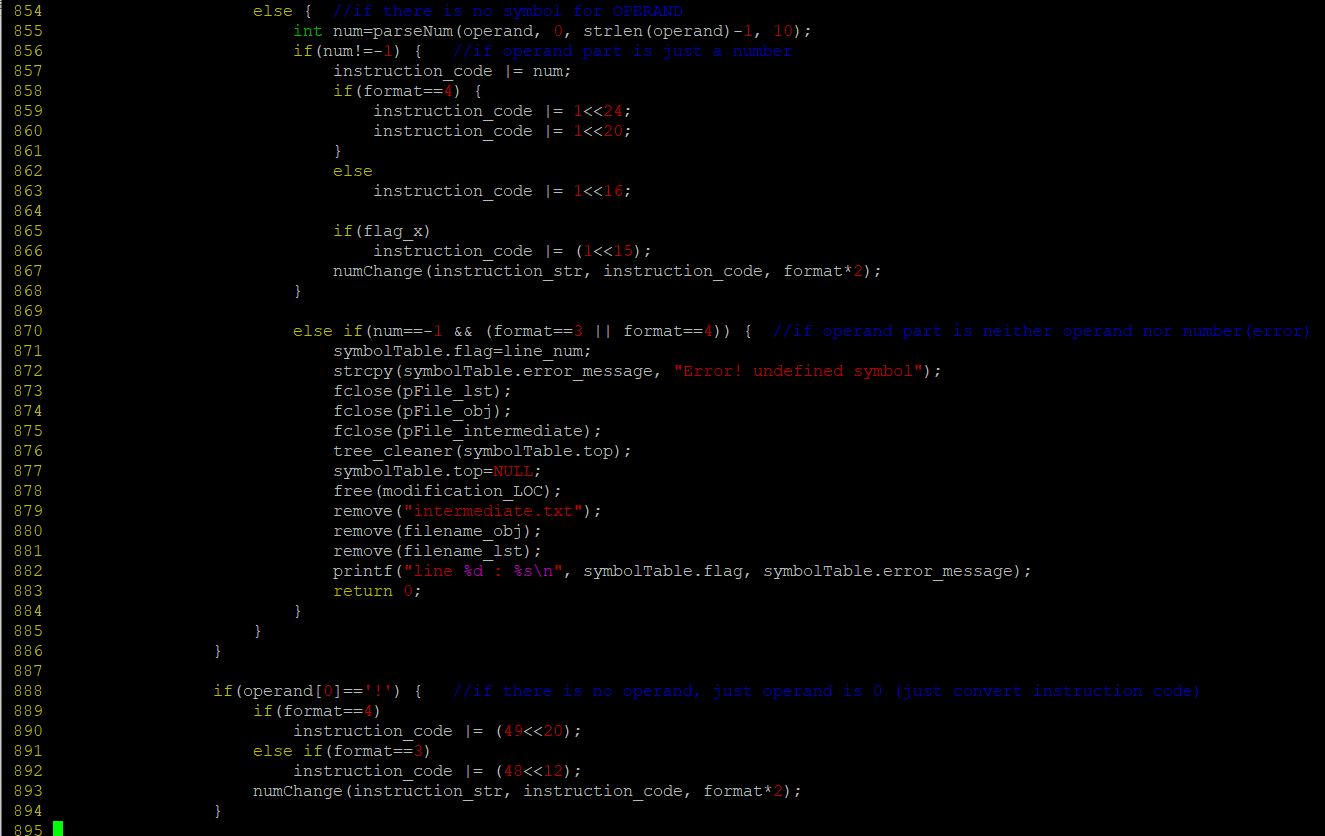
**line 794 ~ 799** : operand부분이 symbol인지 확인한다. (영문자 및 숫자로만 이루어져있는지 확인)

**line 801 ~ 819** : operand 부분이 symbol에 있고 format이 3, 4인 경우에 대한 처리다. 우선 format 4인 경우 instruction\_code를 적절히 bit연산을 이용해 정의하고 마지막으로 X register 이용여부를 판단해 추가해준다.



**line 820 ~ 847** : format 3에 대한 처리이다. 우선 변위(displacement)를 계산하여 그 값이 PC relative로 표현이 가능한 범위인 경우, (-2048 이상 +2047이하) PC relative 를 이용하고 이를 instruction\_code에 반영하고 그렇지 못한 경우 BASE relative를 이용하고 반영한다. mode 변수에 저장된 내용을 확인하여 addressing mode를 instruction\_code에 반영한다.

**line 850** : numChange함수를 이용하여 instruction\_code를 문자열 형태로 변환한 형태를 instruction\_str에 저장한다.



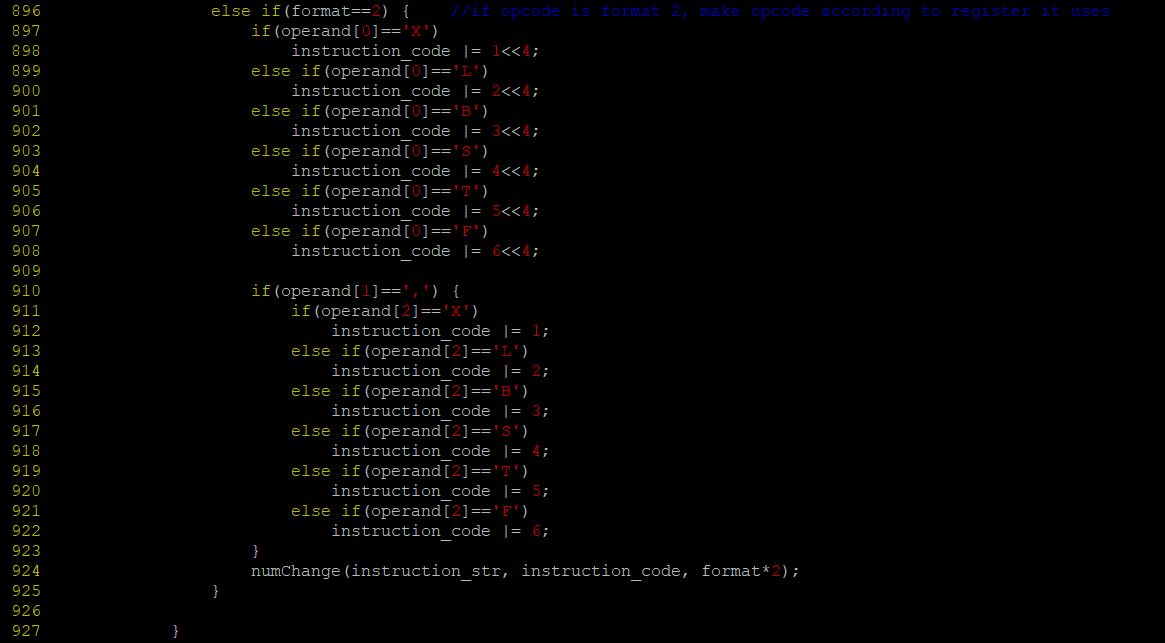
**line 854** : operand에 symbol이 없는 경우이다.

**line 855** : operand 부분에 대하여 parseNum 함수를 호출하여 숫자로 변환이 가능한지 확인한다.

**line 856 ~ 868** : operand 부분이 숫자인 경우, 숫자를 그대로 instruction\_code에 적용한다.

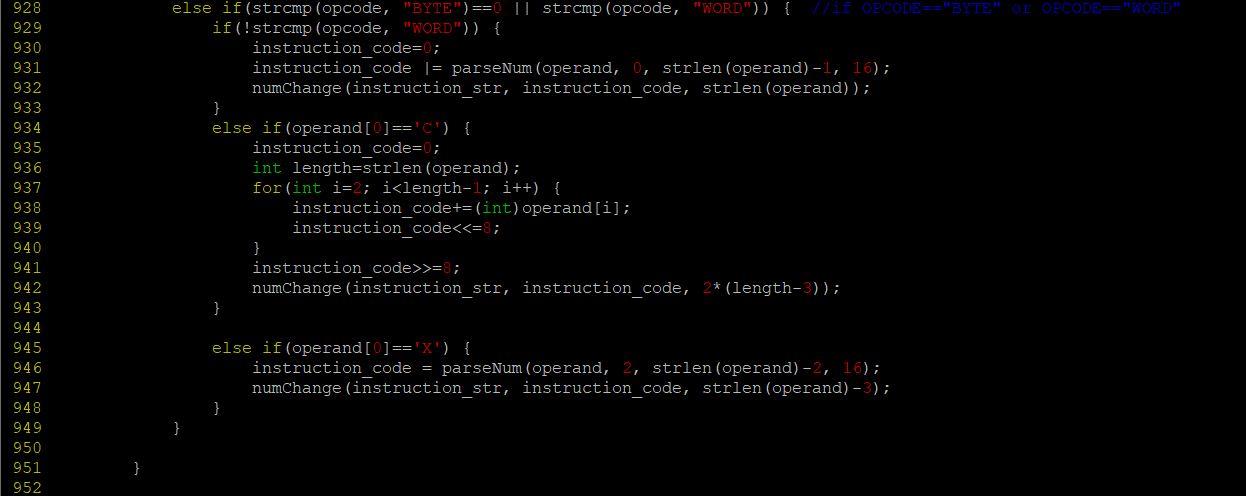
**line 870 ~ 886** : operand 부분이 숫자가 아닌경우 error를 처리하고 함수를 종료한다.

**line 888 ~ 895** : operand 부분이 아무 것도 없는 경우, 0을 operand 부분에 적용한다.

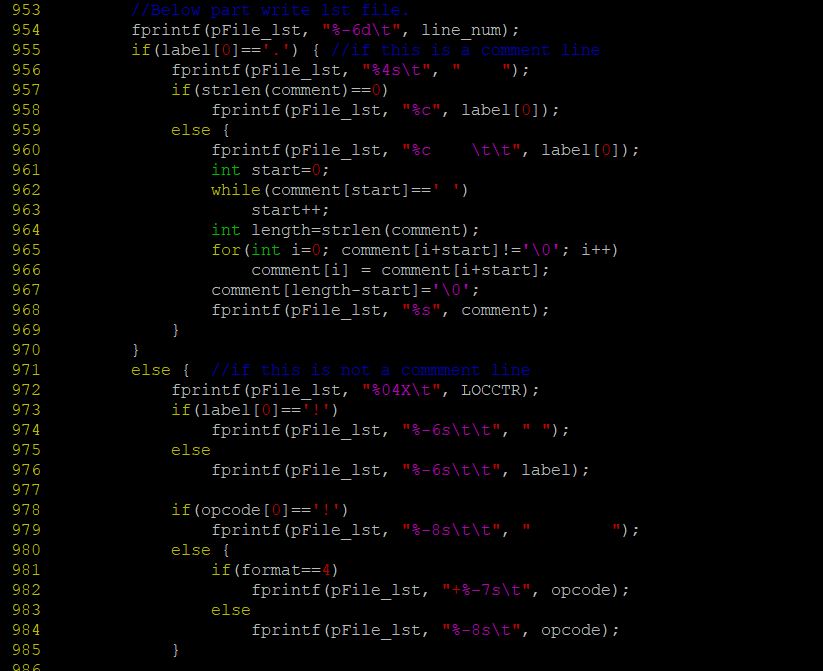


**line 896 ~ 908** : format 2이 면서 operand값이 하나인 경우, 각 레지스터의 값에 해당하는 값을 opcode에 적용한다.

**line 910 ~ 923** : format 2이면서 operand 값이 2개인 경우, 각 레지스터의 값에 해당하는 값을 opcode에 적용한다.

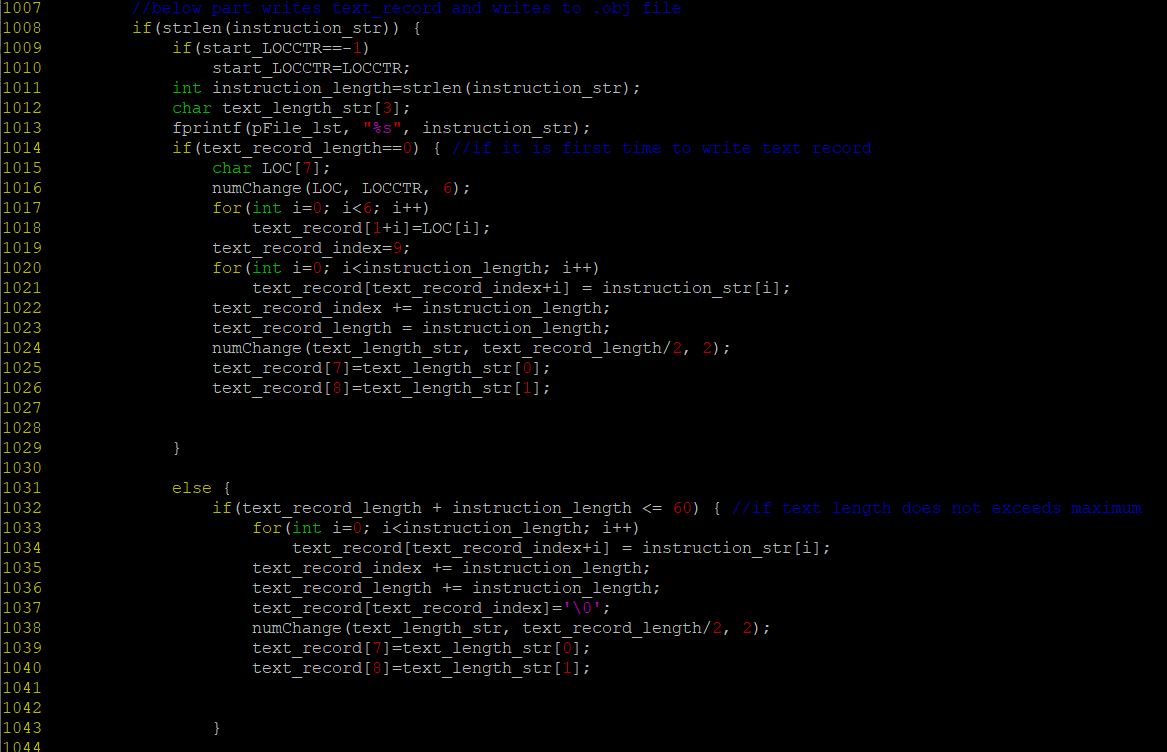


**line 928 ~ 950** : opcode가 BYTE 또는 WORD인 경우에 대한 처리이다.



**line 954 ~ 970** : lst파일에 관련 내용을 write하는 부분 중 주석일 경우에 대한 처리다.

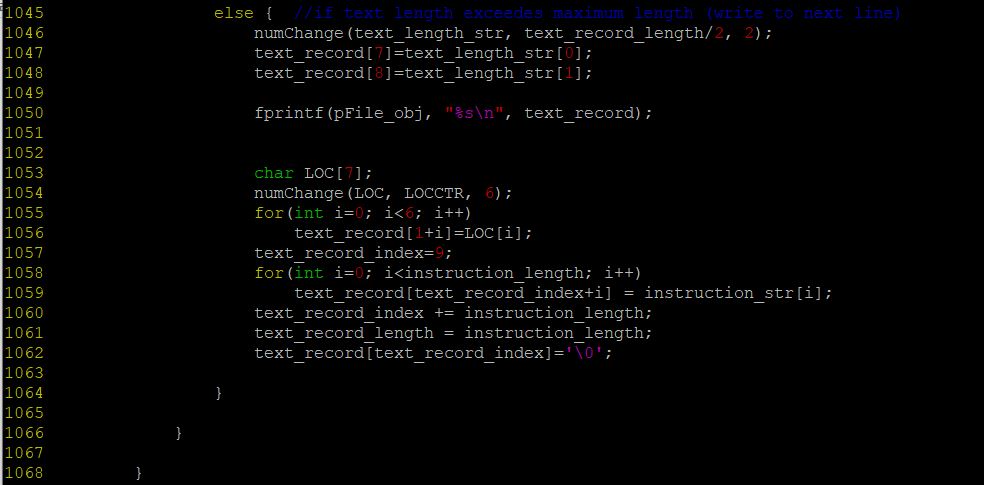
**line 971 ~ 985** : 주석이 아닐 경우 관련 내용을 양식에 맞춰 filename.lst에 write한다.



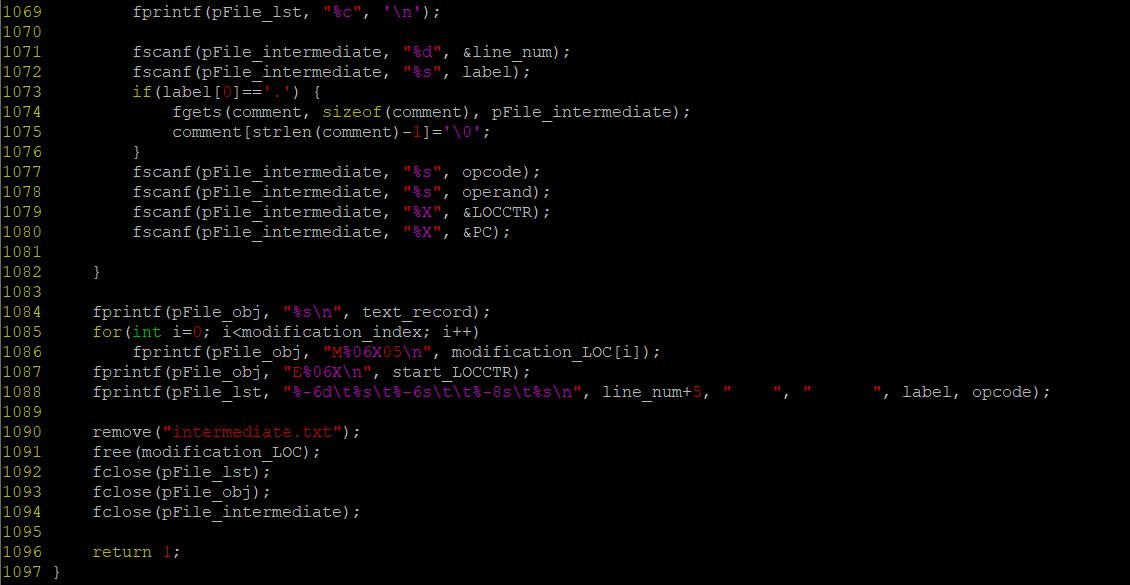
**line 1008** : instruction\_str의 길이가 0이 아닌경우로, opcode가 생성되어 obj파일에 write 될 text\_record를 작성하는 과정이다.

**line 1014 ~ 1029** : text\_record의 길이가 0인 경우로 obj파일을 시작하는 첫 text\_record인 경우다.

**line 1031 ~ 1041** : text record의 길이가 0이 아니면서 제한 길이를 넘어가지 않는 경우로 obj파일에 write하지는 않고 text\_record에 덧붙이는 작업이다.



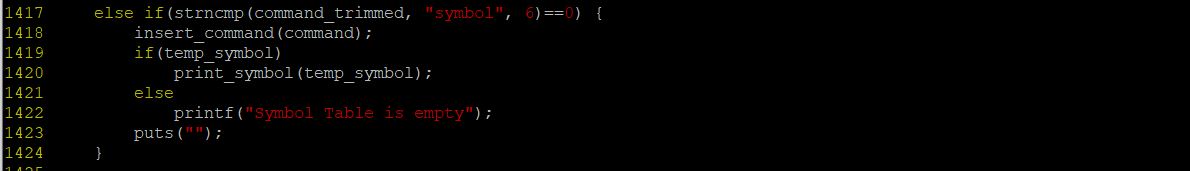
**line 1045 ~ 1068** : text\_record가 길이 제한을 넘어가 해당 내용을 obj 파일에 write하고, opcode를 새롭게 초기화된 text\_record에 저장하고 관련변수들을 최신화하는 작업이다.



**line 1069 ~ 1081** : while 반복문의 끝부분으로, intermediate.txt로부터 새롭게 내용들을 읽어들이는 부분이다.

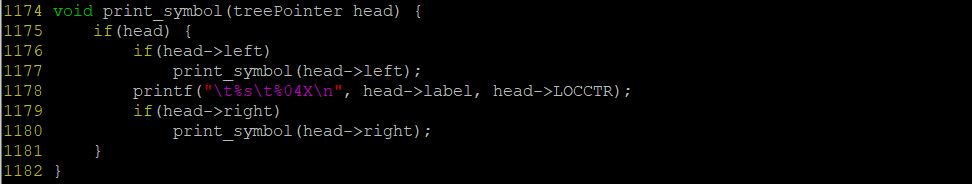
**line 1084 ~ 1097** : while loop를 빠져나온 후, 나머지 내용들을 lst파일 및 obj 파일에 작성한 후, modification에 해당되는 부분을 추가적으로 obj파일에 작성한 후, End recoord를 적고 관련된 내용들을 free해준다. 이 후 성공적으로 프로그램의 종료를 알리기 위해 1을 return 한다.

**- symbol 관련 command Manager 부분 기능 구현**



symbol명령이 입력으로 들어왔을 경우 symbol table이 비어있는 상황이면 비어있다는 메시지를 출력하고, 그렇지 않은 경우 print\_symbol함수를 호출한다.

**- void print\_symbol(treePointer head)**



symbol table의 내용을 in-order 방식으로 출력함으로써 값이 오름차순으로 정렬되게끔 출력할 수 있다. Binary Search Tree의 기능과 동일하며 재귀적으로 함수를 구성하였다.