**과목명: 시스템프로그래밍**

**1 분반**

**<<Project #3>>**

**서강대학교 [컴퓨터공학과]**

**[20161559]**

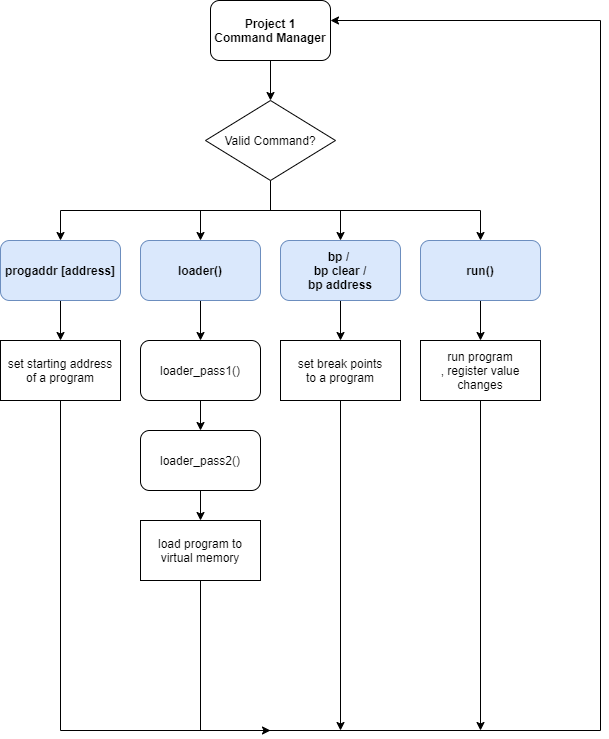
**[강민석]**

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 1, 2에서 구현한 shell에 linking과 loading 기능을 추가하는 프로그램으로, obj 파일을 가상메모리에 load하는 기능을 구현한다. 추가적으로 프로그램을 실행하는 run 기능을 구현하여 각 단계마다 레지스터의 값을 변화시킬 수 있도록 한다.

1. **프로그램 설명**
   1. 프로그램 흐름도



프로그램의 전반적인 흐름은 위와 같다. 우선 첫 번째 기능인 progaddr [address]는 프로그램의 시작주소를 지정해주는 것으로 address가 가상메모리의 index 범위 안에 있으면 정상적으로 프로그램의 시작주소를 address로 설정한다. 그렇지 않을 경우 에러메시지를 출력한다. 두 번째 기능의 loader의 경우 2-pass algorithm을 이용하여 프로그램을 메모리에 로딩한다. 만약 pass1 또는 pass2에서 에러가 발생한다면 실행 중 동적으로 할당된 모든 메모리를 해제하고 관련된 값들을 이전상태로 되돌린다. 세 번째 기능인 bp, bp clear, bp [address]의 경우 break point를 설정하는 기능이다. 마지막 기능인 run 함수의 경우 프로그램의 시작 주소로부터 하나 하나 읽어나가며 레지스터 값의 변화를 기록하는 기능이다.

* 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>

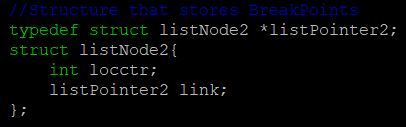
해쉬테이블 및 가상 메모리의 동적할당 및 해제 기능을 위해 사용하였다.

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

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

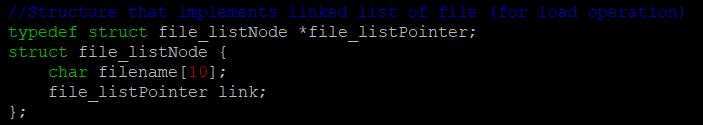
* 1. 사용자 정의 구조체

(1) listPointer2



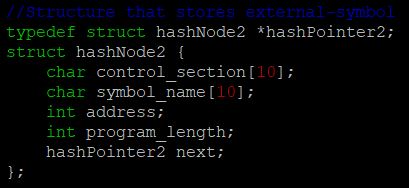
Break point를 구현하기 위한 구조체이다. 기본적으로 linked list와 동일한 구조를 갖는다.

(2) file\_listPointer



obj 파일 목록을 저장하기 위한 구조체이다. 기본적으로 linkned list와 동일한 구조를 갖는다. 예제인 loader proga.obj progb.obj progc.obj의 경우 각 파일 목록이 순서대로 file\_listPointer에 저장된다.

(3) hashPointer2



external-symbol table을 구현하기 위한 구조체이다. linked-list와 array의 조합으로 나타나며 hashPointer2[index]마다 symbol의 개수만큼 linked list가 연결되어 있는 형태를 갖는다. 예제를 기준으로 설명하면 proga.obj는 hashPointer2[0]에 external symbol들이 linked list의 형태로 연결되어 있다.

(4) Reference



modification에서 reference 부분을 reference number를 이용하여 효율적으로 modification을 진행할 수 있도록 하기 위한 구조체이다. used\_flag를 이용하여 해당 external symbol이 탐색이 되었는지, 탐색이 되었다면 해당 external symbol의 주솟값은 어떻게 되는지 파악할 수 있다. (검사를 중복으로 하는 것을 방지한다.

1. **모듈 정의**

추가로 정의한 모듈에 대해 간략한 설명

**void help(void)** : 수행가능한 명령어들을 화면에 출력한다. (progaddr, loader등의 내용 추가)

**void insert\_file(char\*)** : loader 명령어로부터 읽어들인 파일의 목록을 추가 정의한 file\_listPointer에 추가하는 함수이다.

**void insert\_bp(int)** : bp [address] 명령어로부터 읽어들인 address 값을 breakpoint에 추가하는 기능을 하는 함수이다.

**void bp\_cleaner(void)** : 설정된 break pointer를 모두 free해주는 함수이다.

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

**void loader(char\*)** : 프로그램을 메모리에 올려주는 함수로 loader\_pass1, loader\_pass2 함수를 호출 한 뒤, 성공적으로 함수가 종료되면 관련된 load map을 화면에 출력한다.

**int loader\_pass1(void)** : obj 파일의 H(헤더)와 D(symbol-definition)부분으로부터 control\_section name과 external\_symbol을 추출하여 프로그램의 주솟값과 길이 등을 파악하는 기능을 한다. 성공적으로 수행되었을 시 1을, 그렇지 않을 경우 0을 반환한다.

**int loader\_pass2(void)** : 실질적으로 obj파일의 내용을 linking 과정을 수행하며 loading 하는 기능을 한다. 성공적으로 수행되었을 시 1을, 그렇지 않을 경우 0을 반환한다.

**void run(void)**: 프로그램의 시작주소부터 하나하나 읽어나가며 레지스터 값을 업데이트 한다.

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**

**listPointer2 first\_bp :** break point 구현 구조체의 첫 번째 부분을 가리키는 변수이다.

**listPoitner2 trail\_bp :** break point 구현 구조체의 마지막 부분을 가리키는 변수이다.

**file\_listPointer first\_file :** 파일 목록의 첫 번째 부분을 가리키는 변수이다.

**file\_listPointer trail\_file** : 파일 목록의 마지막 부분을 가리키는 변수이다.

**int start\_progaddr** : 프로그램의 시작주소를 나타내는 변수이다.

**hashPointer2 ESTable[10]** : external symbol table을 저장하기 위한 변수이다.

**int bp\_status[0xFFFFF]** : 가상메모리에서 break point가 설정되어있는지 판별하기 위한 변수이다.

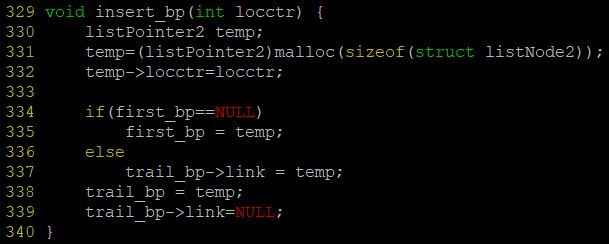
**int A, X, L, PC, B, S, T** : 레지스터에 저장되어 있는 값을 담고 있는 변수이다.

1. **코드 설명**

**- 20161559.h (20161559.c는 proj1, 2와 같다.)**

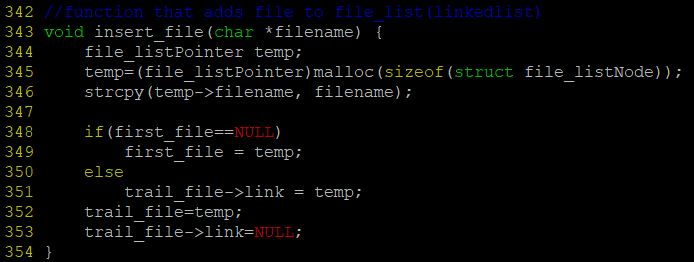
**(1) Secondary function(Primary function의 기능을 보조하는 함수 코드)**

**- void insert\_bp(int);**

****

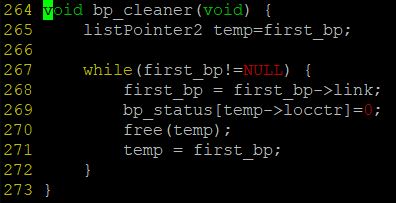
입력으로 들어온 locctr값을 갖는 listPointer2를 생성하고 기존에 있던 bp linked list에 연결하는 삽입 함수이다.

**- void insert\_file(char\*);**



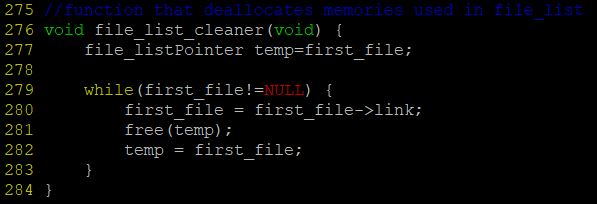
입력으로 들어온 filename을 갖는 file\_list를 생성하고 기존에 있던 file linked list에 연결하는 삽입 함수이다.

**- void bp\_cleaner(void);**

****

프로그램의 종료 및 bp\_cleaner 호출 시 동적으로 할당 된 bp linked list를 모두 free해주는 기능을 하는 함수이다.

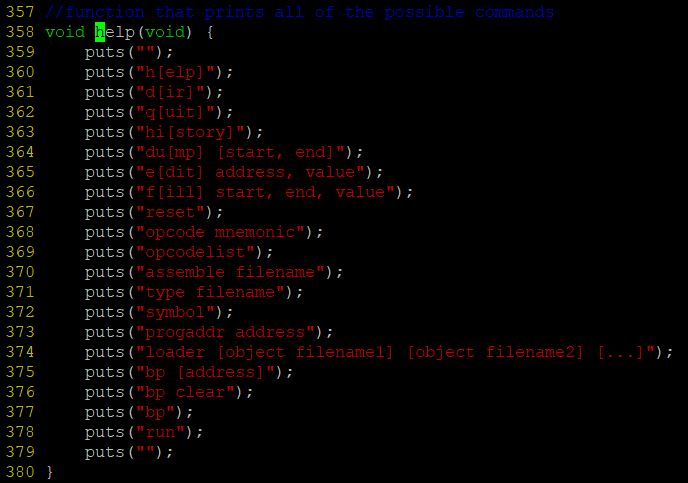
**- void file\_list\_cleaner(void);**



프로그램의 종료 및 에러 발생시 동적으로 할당 된 file linked list를 모두 free 해주는 기능을 하는 함수이다.

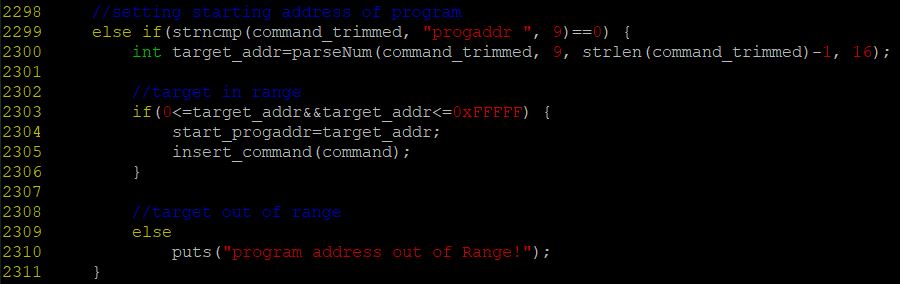
**(2) Primary function(프로그램의 주 흐름을 담당하는 함수)**

**- void help(void)**

****

실행 가능한 명령어들을 출력하는 부분으로 project2와 비교했을 때 6가지 내용이 더 추가되었다. (line 373 ~ 378)

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

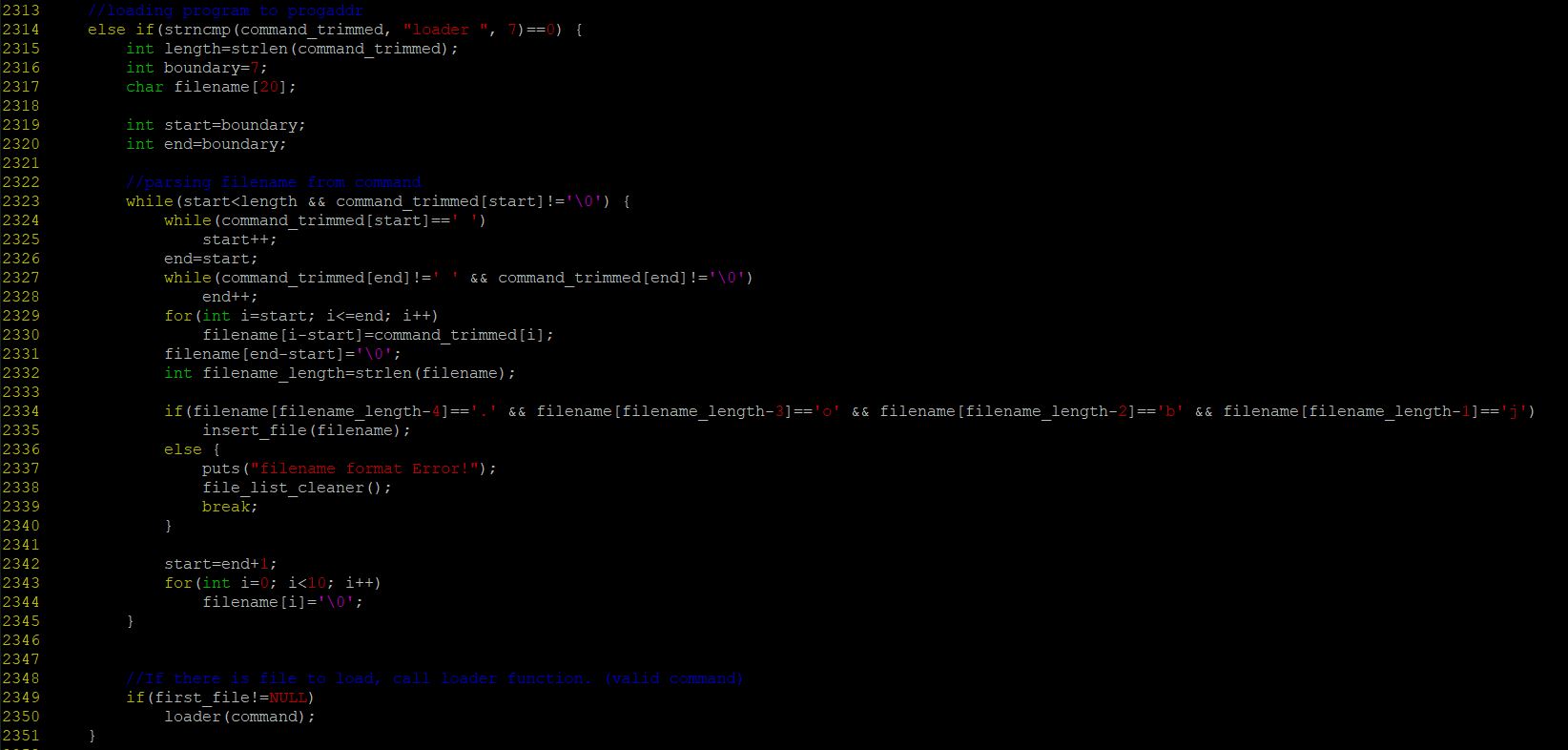


**line 2299** : command를 입력 받았을 때 앞 부분이 “progaddr “와 매치되는 조건이다.

**line 2303 ~ 2306** : progaddr의 address 부분이 가상메모리의 범위(0 ~ 0xFFFFF) 안에 들어가면 start\_progaddr을 address로 설정하고 history에 입력으로 들어온 명령어를 추가한다.

**line 2309 ~ 2310** : 범위를 벗어나는 경우 에러 메시지를 출력한다.

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

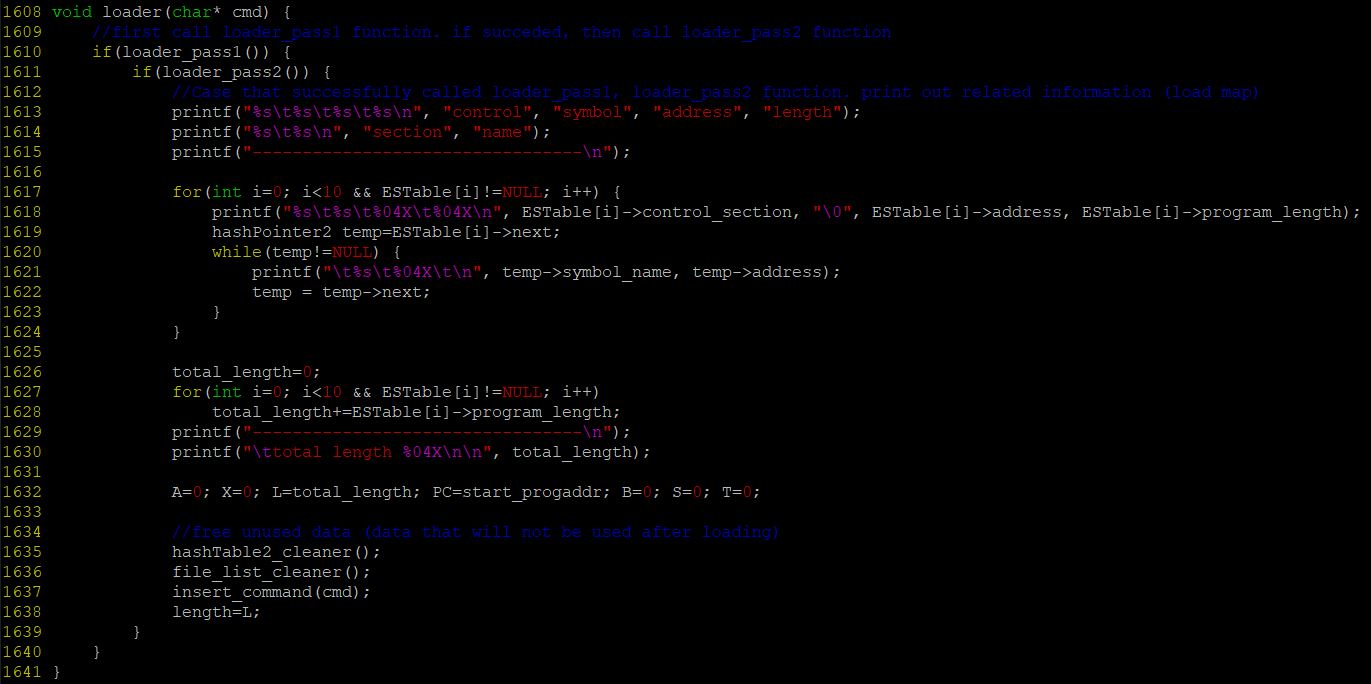
****

**line 2315 ~ 2320 :** 명령어로부터 인자(file 이름)등을 분리하기 위한 작업을 수행하기 위해 필요한 변수들을 선언한다.

**line 2323 ~ 2345** : 명령어로부터 file name을 분리해내고, 관련하여 file\_list에 파일이름들을 삽입한다. 만약 파일의 확장자가 .obj가 아닌경우 에러미시지를 출력하며 이전에 생성되었던 관련 메모리들을 free해준다.

**line 2349 ~ 2350** : 만약 file이 NULL이 아니면, loader 함수를 호출한다.

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

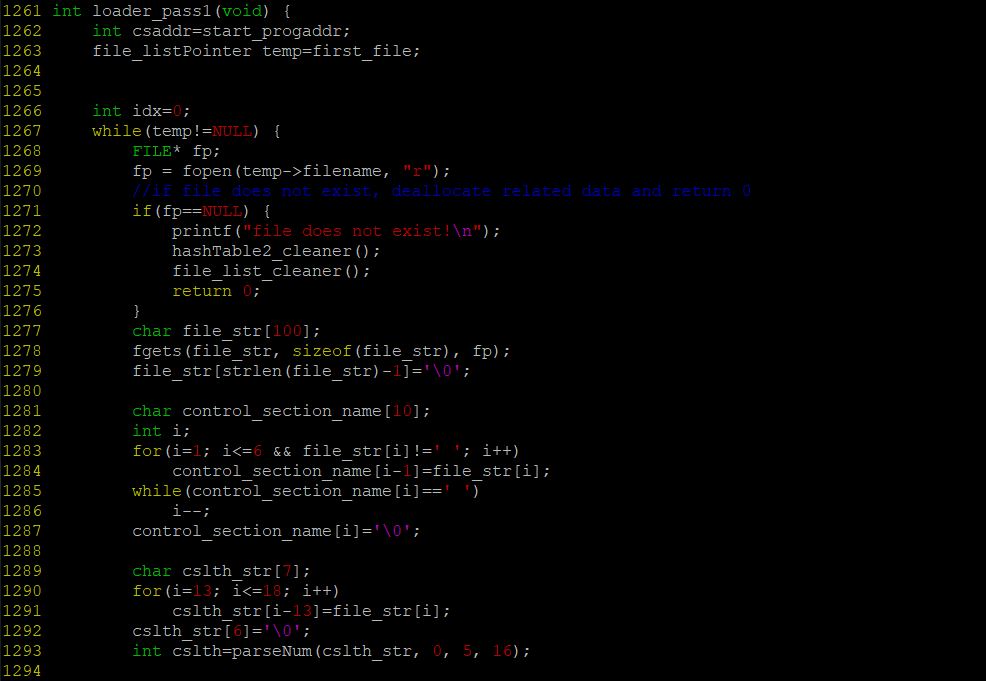


**line 1610** : 우선 loader\_pass1 함수를 호출한다. 성공적으로 pass1 함수가 종료되면 이후 문장을 실행한다.

**line 1611** : loader\_pass2 함수를 호출한다. 성공적으로 pass2 함수가 종료되면 이후 문장을 실행한다.

**line 1613 ~ 1638** : 성공적으로 pass1, pass2 함수가 호출된 경우 load map을 출력하고 레지스터 값들을 0으로 초기화한다. (단, PC 레지스터는 프로그램의 시작주소로, L 레지스터는 프로그램의 길이로 초기화한다. 그리고 사용되었던 동적으로 할당된 데이터들을 free해준다.

**- int loader\_pass1(void);**



**line 1262 ~ 1263 :** file\_list에 들어있는 첫 번째 파일을 temp에 저장한 후 while loop를 돌아간다.

**line 1267 ~ 1276** : file list에 있는 파일명에 대응되는 파일을 fopen한다. 만약 존재하지않는다면 에러메시지를 출력한 후 비정상적인 종료를 알리는 0을 return 한다.

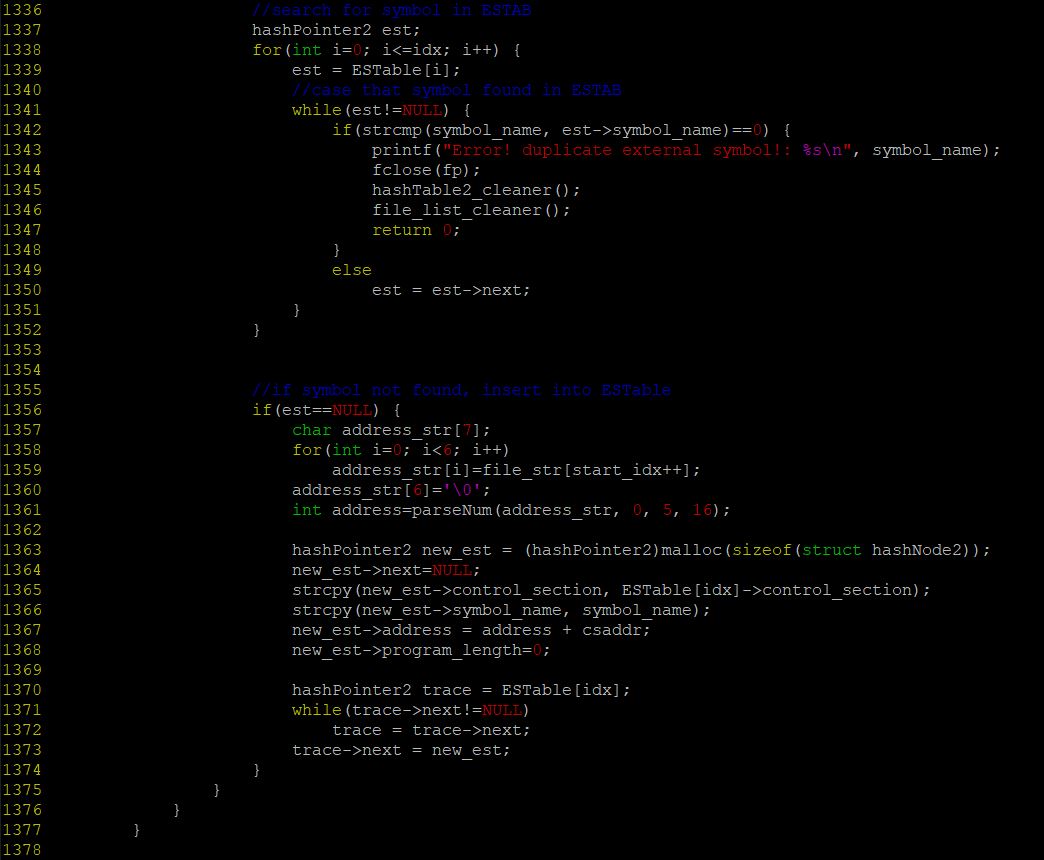
**line 1277 ~ 1292** : .obj 파일로부터 한 line을 읽어들여 control\_section\_name (프로그램 이름)을 추출해내고 파일의 길이 cslth를 추출해낸다.



**line 1296 ~ 1304** : 올라가는 프로그램들 중 중복되는 control\_section이름이 있는지 검사한다. 만약 중복되는 부분이 있다면 에러메세지를 출력한 후, 관련되어 생성된 데이터들을 free한 후 0을 return 한다.

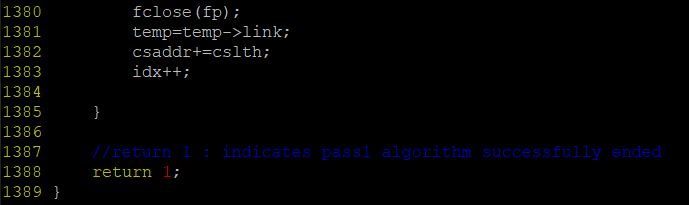
**line 1307 ~ 1313** : 중복된 control section이 없어 정상적으로 내용을 external –symbol table에 삽입하는 부분이다.

**line 1316 ~ 1334** : .obj 파일에서 파일이 ‘D’(심볼 정의)으로 시작하는 문장의 경우, 관련 내용을 추출하고 중복검사를 진행한다.



**line 1337 ~ 1352** : 심볼 중복검사를 진행한다. 만약 심볼테이블에 있는 심볼과 비교 대상 심볼의 match 가 일어나면 (line1342) 에러메세지를 출력하고 관련된 데이터들을 free해준후 0을 return 한다.

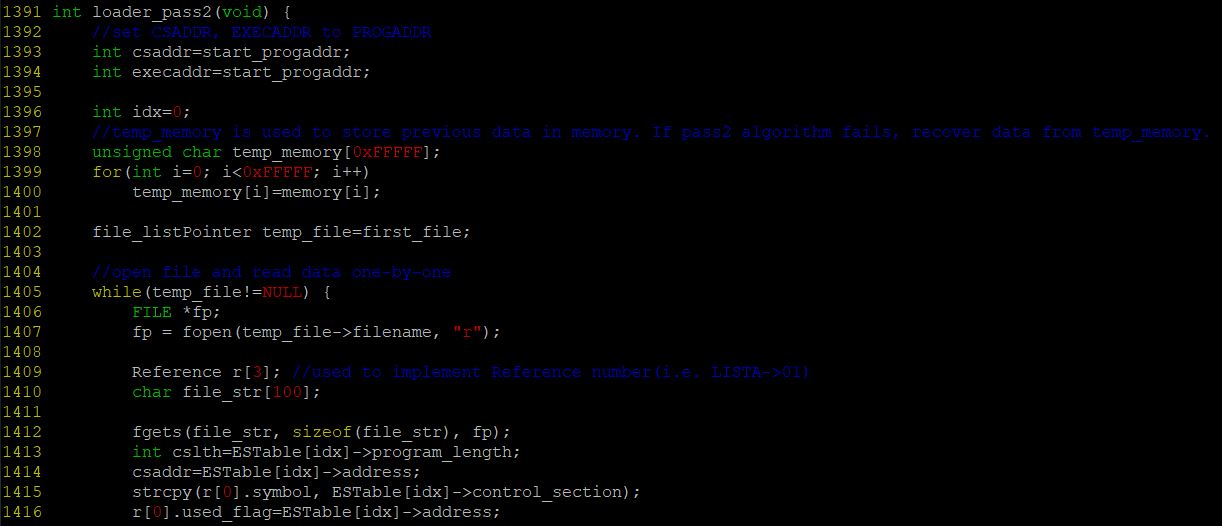
**line 1356 ~ 1373** : 심볼 중복검사를 통과한 이후 부분으로, external symbol table에 새로운 symbol을 추가한다.



**line 1380 ~ 1383** : 한 파일에 대한 정보 입력을 마친 후, 다음 파일에 대한 데이터를 받아 들이기 위한 변수 수정부분이다.

**line 1388** : 성공적으로 loader\_pass1알고리즘이 끝났음을 의미한다.

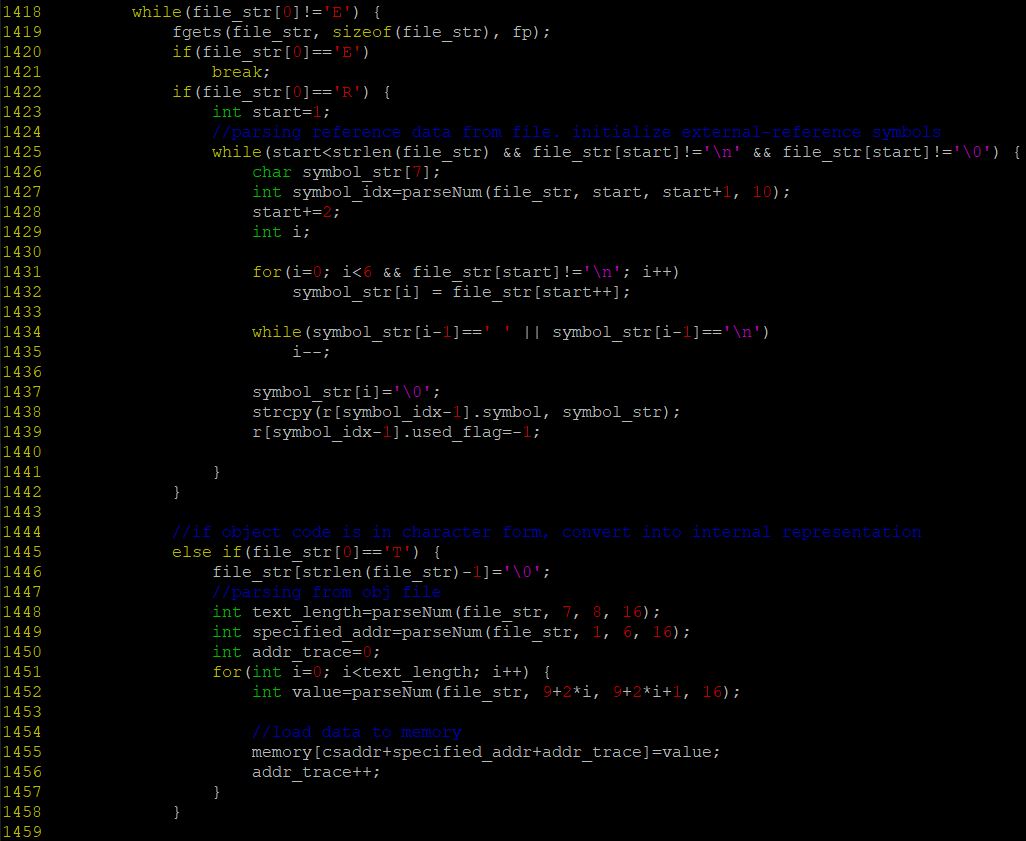
**- int loader\_pass2(void);**



**line 1393 ~ 1394** : 컨트롤섹션의 주소와 프로그램 실행주소를 start\_progaddr로 초기화한다.

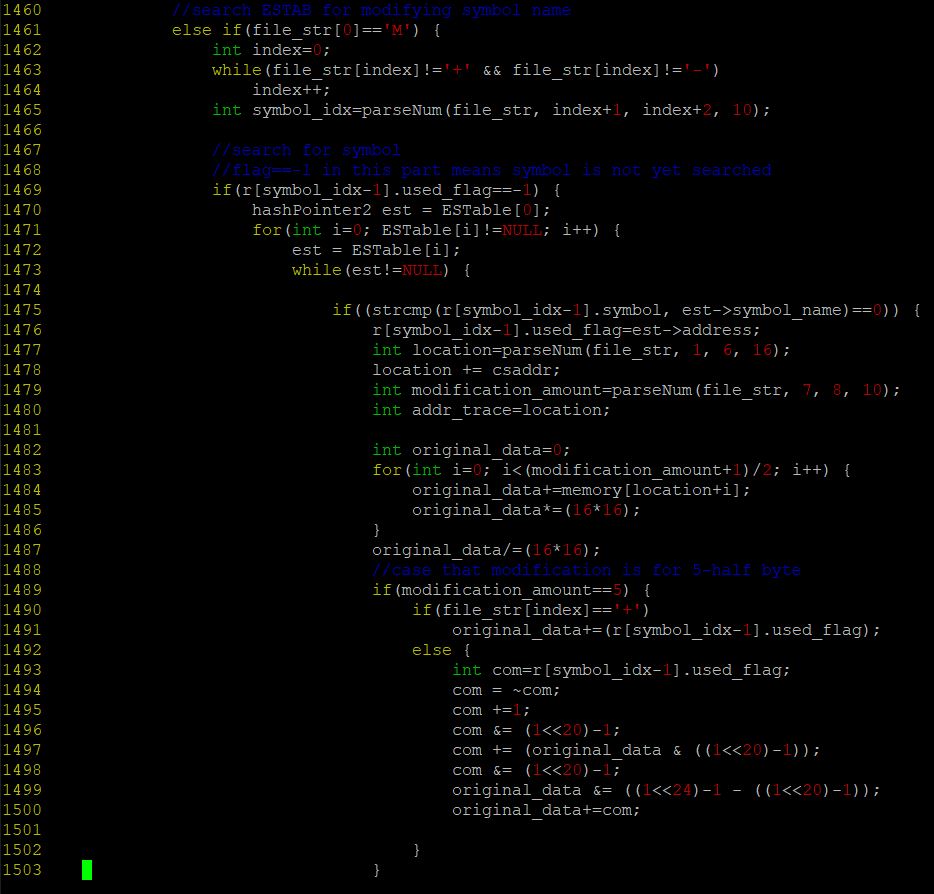
**line 1398 ~ 1400** : pass2 함수 실행 중 오류가 발생했을 시 이전 메모리 상태를 유지하기 위한 임시 메모리이다. 0을 반환할 경우 이전에 저장해두었던 temp\_memory 값을 다시 memory에 돌려놓는다.

**line 1405 ~ 1416** : 파일 리스트에 있는 파일을 하나하나 읽어들이는 과정이다.



**line 1418 ~ 1442** : End record를 만날 때 까지 .obj 파일로부터 문장을 읽어들인다. ‘R’ record를 만날 경우(line 1422) symbol name을 추출해낸다.

**line 1445 ~ 1459** : text record를 만난 경우로 해당 데이터를 문자형에서 16진수 값으로 변환 후 memory에 올리는 작업을 진행한다.



**line 1461 ~ 1465** : Modification record를 만난경우이다.

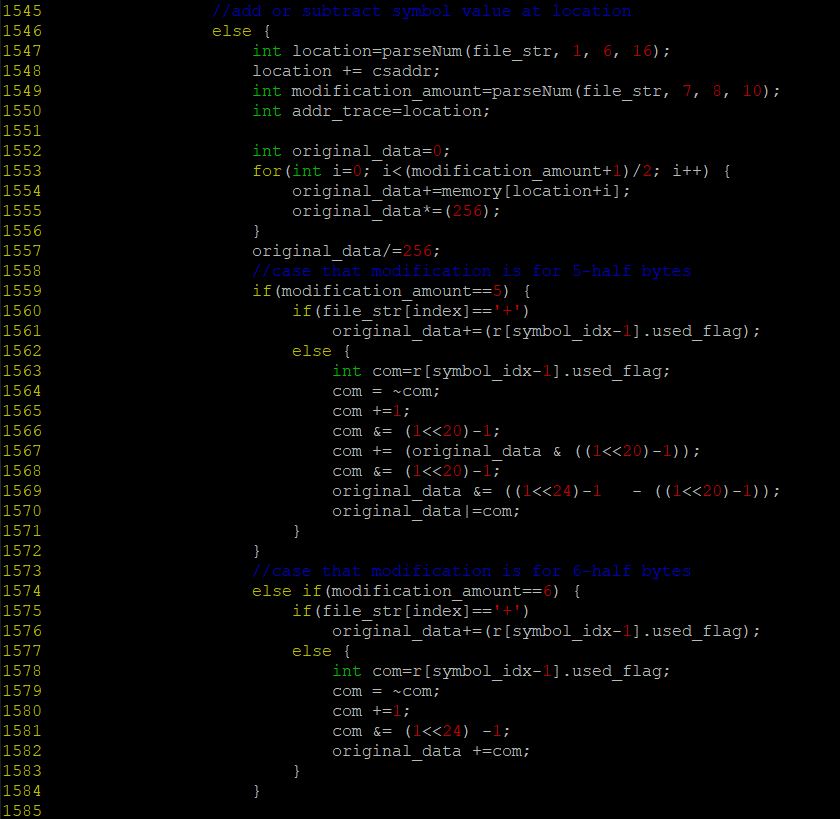
**line 1469** : used\_flag==-1인 경우로, 아직 symbol 탐색이 이뤄지지 않은 경우다.

**line 1470 ~ 1503** : symbol 탐색을 진행한다. 만약 발견되면(line 1475) used\_flag를 해당 symbol의 주소로 설정하고 관련된 정보들을 추출한다.

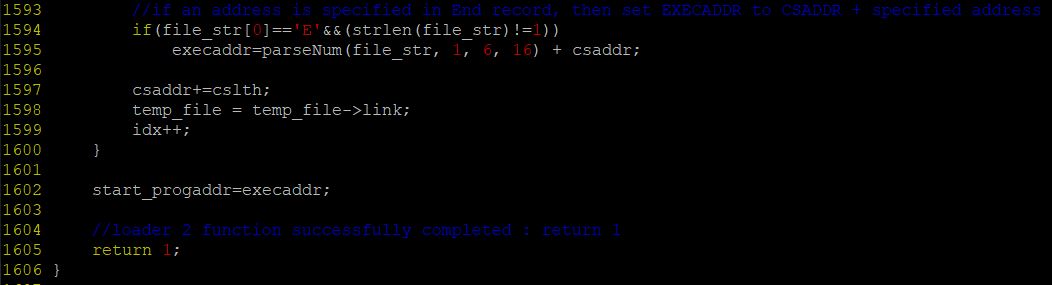


**line 1518 ~ 1521** : memory에 modification 부분을 추가하는 내용이다.

**line 1533 ~ 1544** : symbol 탐색을 진행했음에도 여전히 used\_flag가 -1로 세팅되어있는 경우로 external symbol이 없다는 에러메세지를 출력한다. 관련되어 할당 되었던 데이터들을 free해주고 임시메모리로부터 데이터를 복사한 후 비정상적인 종료를 알리는 0을 return 한다.

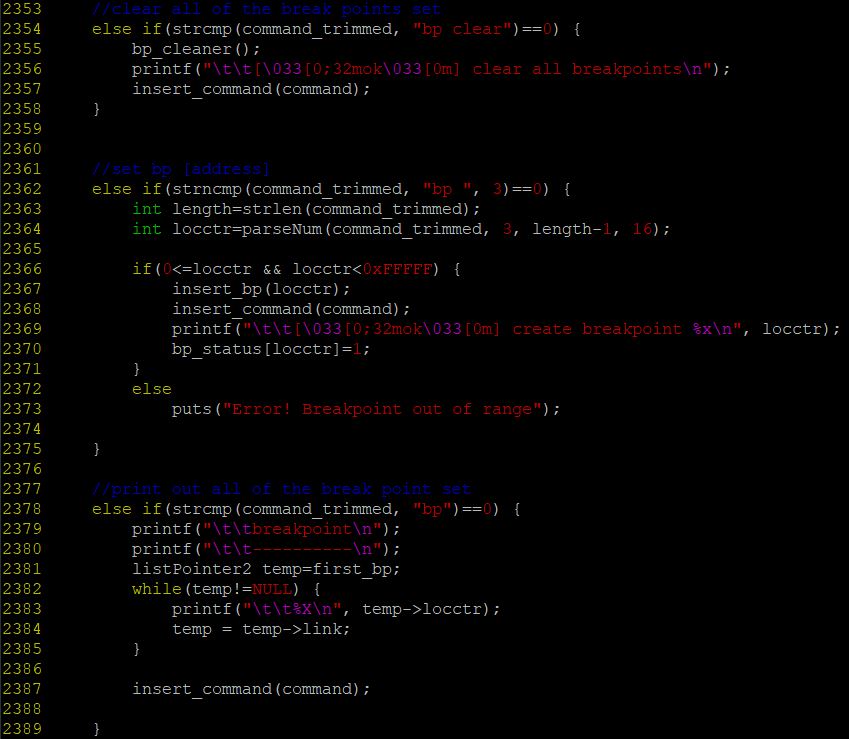


**line 1545 ~ 1585** : 해당 부분은 이미 이전에 탐색이 이루어져 used\_flag가 -1이 아닌경우에 대한 부분이다. 탐색을 중복으로 할 필요가 없으므로 바로 modification을 진행한다.



**line 1594 ~ 1606** : 만약 end record 뒤에 프로그램의 시작 주소를 명시하는 부분이 있으면 해당 부분을 execution address로 세팅한다. loop가 끝나면 정상적으로 함수종료를 뜻하는 1을 return 한다.

**- bp 관련 command Manager 부분 기능 구현**



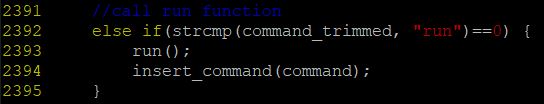
**line 2354 ~ 2358** : bp clear 명령을 구현한 부분이다. 해당 명령어가 입력으로 들어오면 break point를 구성하는 linked list를 전부 free하는 bp\_cleaner()함수를 호출한다.

**line 2362 ~ 2375** : bp [address] 명령을 구현한 부분이다. address부분이 가상메모리 index 범위 안에 들어가면 해당 부분의 locctr을 갖는 break point를 생성해 insert\_bp를 수행하고, bp\_status를 1로 세팅한다. 그렇지 않은 경우 범위를 벗어났다는 에러메세지를 출력한다.

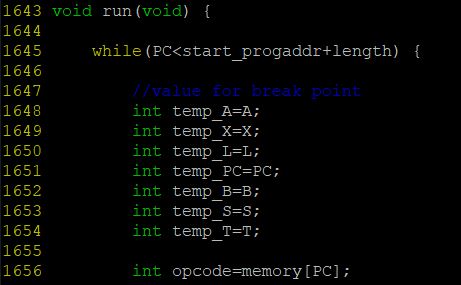
**line 2378 ~ 2389** : bp 명령을 구현한 부분이다. break point가 설정된 주소를 전부 출력하고 history에 해당 명령을 추가한다.

\*bp 부분은 구현 내용이 복잡하지 않아 command Manager 부분에 간단히 작성하였고 별도의 함수를 생성하지 않았다.

**- run 관련 command Manager 부분 기능 구현**



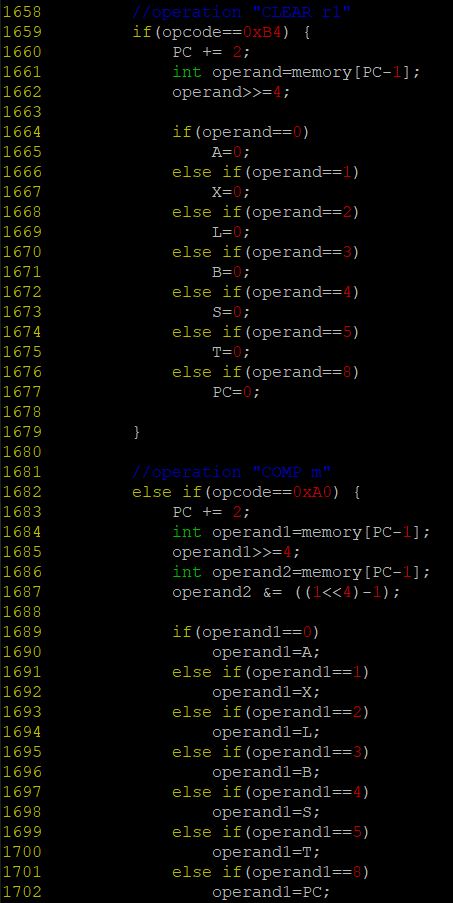
**- void run(void);**



**line 1645** : program counter가 프로그램의 길이를 넘어서지 않는 동안 반복문이 실행된다.

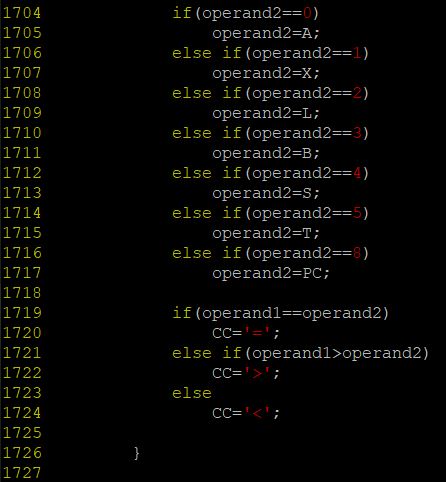
**line 1648 ~ 1654** : 이후 break point에 걸릴 경우에 출력해야 될 register 값들을 temp 변수에 저장해둔다.

**line 1656** : opcode를 memory로부터 불러오는 작업이다.

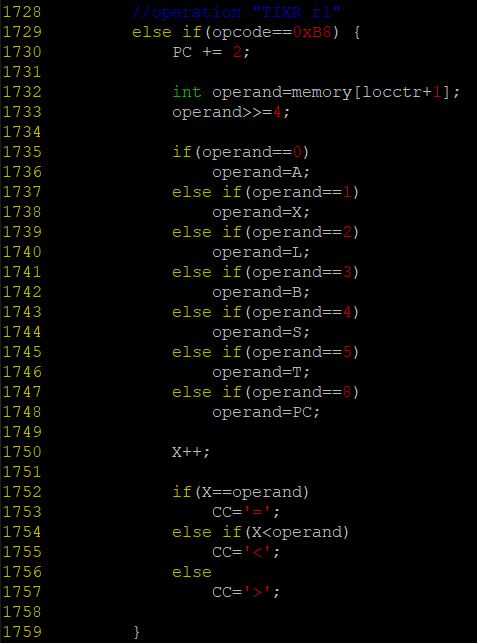


**line 1658 ~ 1679** : CLEAR operation에 대한 부분이다. 대응되는 레지스터 값을 0으로 초기화한다.

**line 1682 ~ 1702** : COMP operation에 대한 부분이다. operand1과 operand2에 각각 대응되는 레지스터의 value를 저장한다.



**line 1704 ~ 1727** : COMP 연산에서 operand 1과 operand 2값을 비교하여 경우에 따라 CC(Condition Code)를 설정한다.

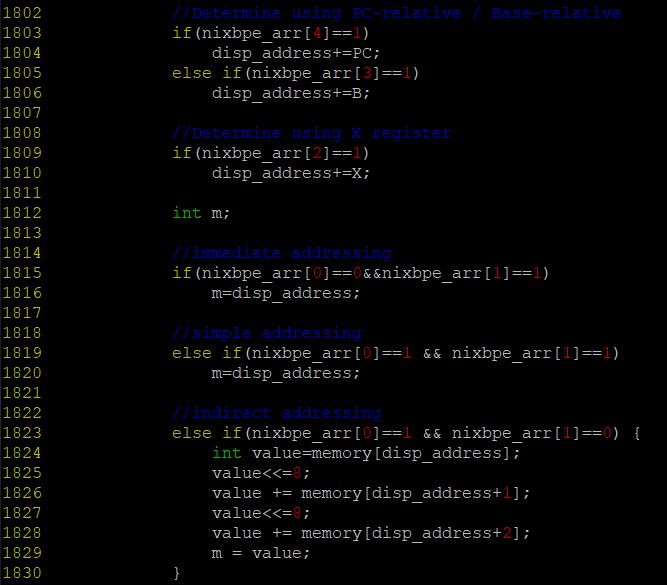


**line 1728 ~ 1759** : TIXR operation에 대한 부분이다. X+1값과 operand의 부분을 비교하여 condition code를 적절히 세팅한다.



**line 1761 ~ 1776** : format 3또는 format 4에 대한 operation을 구현한다. 두 가지 형식에 대해서는 nixbpe를 통해 정보를 추출할 수 있으므로 nixbpe부분을 추출한다.

**line 1778 ~ 1800** : nixbpe 중 e 값을 통해 format 3인지 format 4인지 판단한다. 이에 따라 displacement address를 설정한다.



**line 1803 ~ 1806** : 해당 instruction이 PC relative인지 base relative인지에 따라 displacement를 수정한다.

**line 1809** : 해당 instruction이 x register를 이용하는지 여부에 따라 displacement를 수정한다.

**line 1815 ~ 1830**  :해당 instruction의 addressing mode (simple, immediate, indirect)에 따라 displacement를 변경한다.



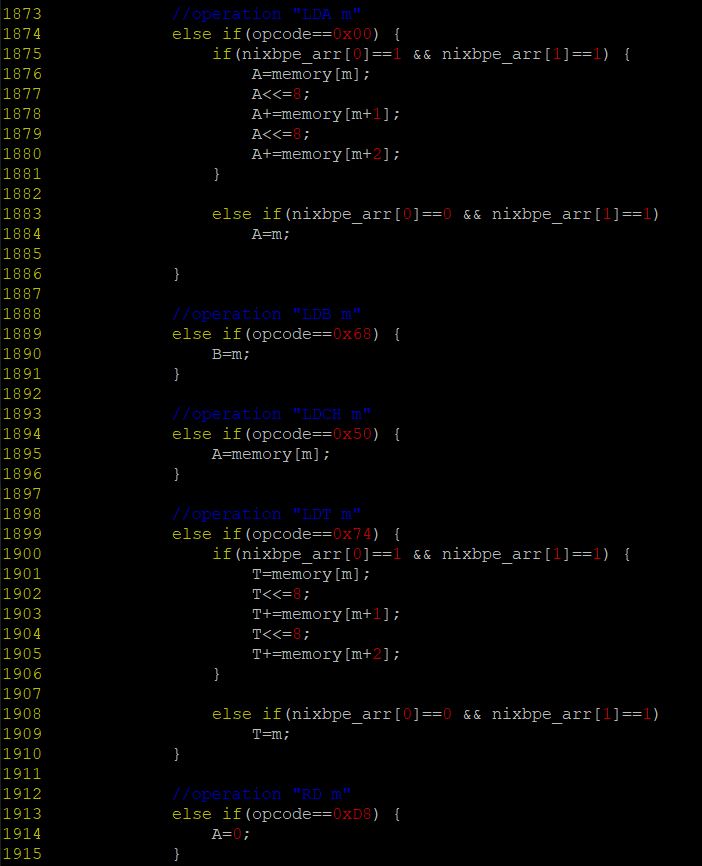
**line 1836** : COMP operation 구현 부분이다.

**line 1845** : J (jump) operation 구현 부분이다.

**line 1852** : JEQ operation 구현 부분이다.

**line 1860** : JLT operation 구현 부분이다.

**line 1868** : JSUB operation 구현 부분이다.



**line 1874** : LDA operation 구현부분이다.

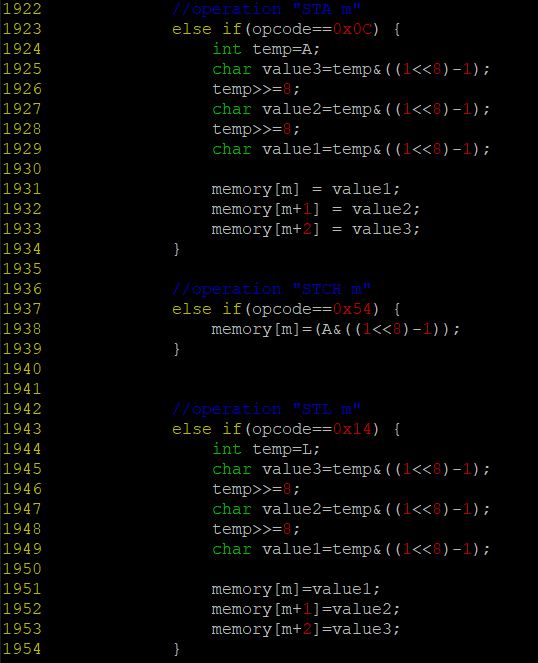
**line 1889** : LDB operation 구현부분이다.

**line 1894** : LDCH operation 구현부분이다.

**line 1899** : LDT operation 구현부분이다.

**line 1913** : RD operation 구현부분이다.

**line 1918** : RSUB operation 구현부분이다.

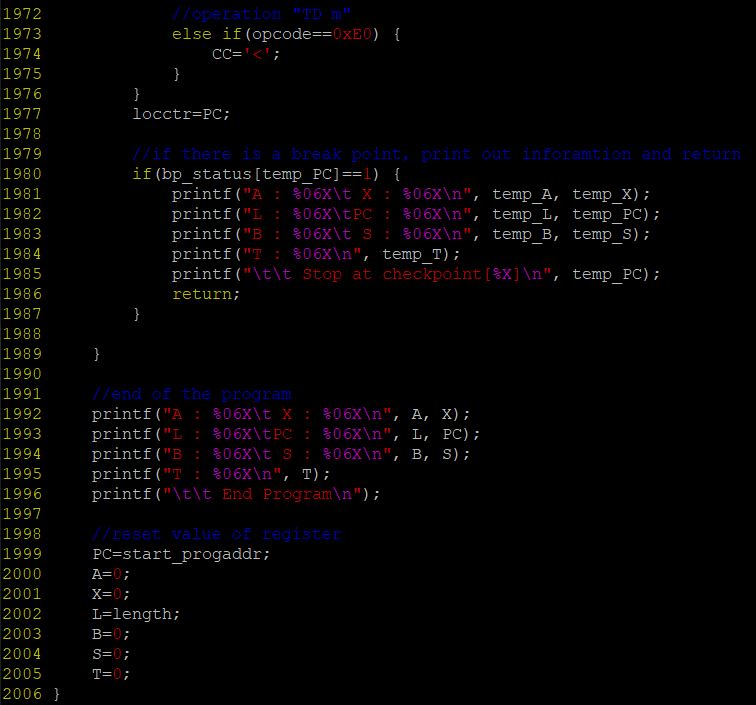


**line 1923** : STA operation 구현부분이다.

**line 1937** : STCH operation 구현부분이다.

**line 1943** : STL operation 구현부분이다.

**line 1957** : STX operation 구현부분이다.



**line 1973** : TD operation 부분 구현이다.

**line 1980 ~ 1989** : break point가 걸려 있는 경우에 대한 처리로, 레지스터 값들을 출력하고 반복loop를 더 이상 돌지 않고 return 한다.

**line 1991 ~ 2006** : 프로그램의 끝에 도달 했을 경우 end program을 출력한 후 레지스터 값들을 출력한 후 프로그램을 종료한다.

**line 1999 ~ 2006** : 프로그램이 종료되면 관련 레지스터 값들을 프로그램 시작 이전의 값으로 초기화해준다.