**과목명: 시스템프로그래밍**

**1반**

**<<Project #1>>**

**서강대학교 경영학과**

**20140825**

**박진혁**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. helpers
   2. shell\_commands
   3. memory\_commands
   4. opcode\_commands
   5. 20140825
4. **전역 변수/매크로/데이터 타입 정의**
   1. helpers.h
   2. shell\_commands.h
   3. memory\_commands.h
   4. opcode\_commands.h
   5. 20140825.h
5. **코드 설명**
   1. helpers.h
      1. strcmp\_twice
      2. hexstr\_to\_int
   2. shell\_commands.h
      1. help
      2. dir
      3. quit
      4. history
      5. push\_log
      6. free\_log
   3. memory\_commands.h
      1. dump
      2. edit
      3. fill
      4. reset
      5. print\_chars
      6. validate\_hexstr\_argumetns functions
         1. validate\_one\_hexstr\_argument
         2. validate\_two\_hexstr\_arguments
         3. validate\_three\_hexstr\_arguments
      7. validate\_address
      8. validate\_value
      9. validate\_range
   4. opcode\_commands.h
      1. opcode
      2. opcodelist
      3. init\_hash\_table
      4. get\_hash\_index
      5. free\_hash\_table
   5. 20140825.h
      1. flush\_tokens
      2. get\_next\_token\_idx
      3. tokenize\_input
      4. get\_command
      5. execute\_instructions
      6. init
      7. exit\_program
      8. main

**1. 프로그램 개요**

이번 첫번째 프로젝트의 프로그램은 앞으로의 프로젝트들을 수행하기 위하여 SIC/XE 머신을 본격적으로 구현하기 전 단계로, 여러 명령어들을 실행할 shell과 메모리 공간, opcode 테이블 등과 관련된 사항들을 구현하는 것을 목표로 한다.

본 프로그램의 세부 요구사항은 다음과 같다.

* 이 프로그램을 실행시키면 아래와 같은 입력 프롬프트 상태가 되어야 한다.
  + **sicsim>**
* 1MB의 메모리를 할당하여 메모리를 사용해야 한다.
* hash table 형태로 opcdoe 테이블을 구현해야 한다.
* 입력 프롬프트 상태에서 아래 명령어들을 입력할 때 해당하는 기능을 수행해야 한다.
  + shell 관련 명령어 : help, dir, quit, history
  + 메모리 관련 명령어 : dump, edit, fill, reset
  + opcode 관련 명령어 : opcode, opcodelist

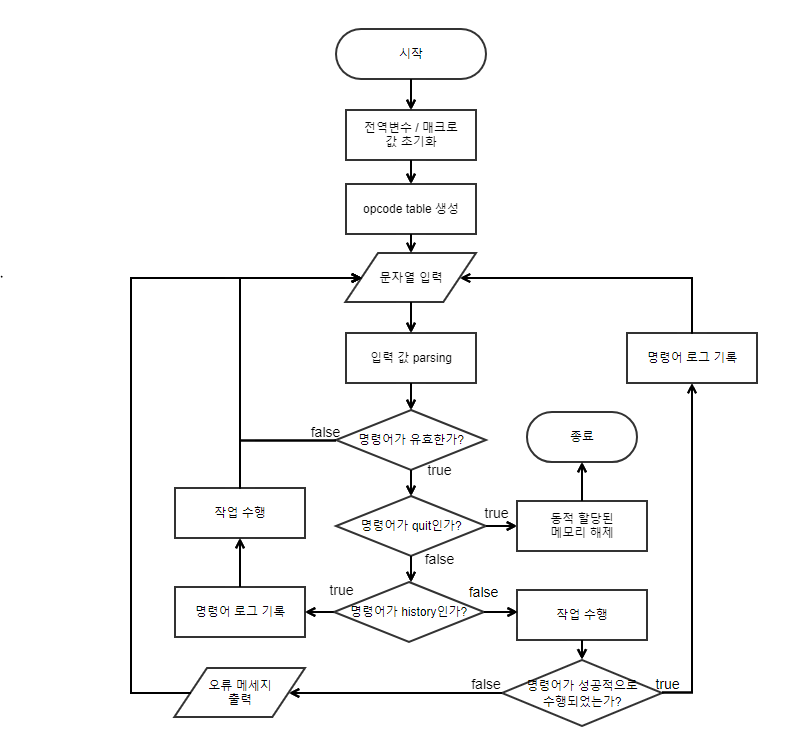
또한 본 프로그램은 위와 같은 세부 요구사항들을 만족시킴과 동시에, 개발 과정에서 코드의 가독성과 재사용성을 위해 가능한 독립적으로 각 기능들을 모듈화 하고자 하는 방향으로 개발되었다.

**2. 프로그램 설명**

이번 과제의 프로그램은 ‘20140825.c’ 파일에 있는 main 함수와, main 함수가 의존성을 가지고 있는 모든 모듈들 간의 컴파일을 통해 생성된 ‘20140825.out’ 파일의 실행을 통해 사용될 수 있다.

**2.1 프로그램 흐름도**

프로그램의 논리적 흐름은 다음과 같다.



**3. 모듈 정의**

본 프로그램의 소스 코드는 크게 5개의 모듈로 구분되어 있다. 각 모듈은 header 파일과 c 소스 파일로 이루어져 있으며, main 함수가 있는 20140825의 헤더 파일에서 모듈들을 참조해 의존성 관계를 형성한다.

* 1. **helpers**

helpers는 여러 모듈에서 필요로 하는 utility 함수들을 모아놓은 모듈이다. 두 개의 문자열을 하나의 문자열과 동시에 비교하는 strcmp\_twice와 문자열 형태의 16진수를 10진수 int 타입의 데이터로 변환해주는 hexstr\_to\_int 등의 함수가 정의되어 있다.

* 1. **shell\_commands**

shell\_commands는 help, dir, quit, history 등 shell과 관련된 명령어를 처리하기 위한 모듈이다. 또한 이 모듈에는 history와 관련된 명령어를 처리하기 위한 별도의 함수들도 포함되어 있다. 명령어 로그들을 저장하기 위한 push\_log, log들을 저장하기 위해 동적 할당된 메모리들을 해제하는 free\_log 등의 함수들이 여기에 포함된다.

* 1. **memory\_commands**

memory\_commands는 dump, edit, fill, reset 등 memory와 관련된 명령어를 처리하기 위한 모듈이다. dump, edit, fill의 명령어 모두 별도의 argument를 필요로 하는 명령어들이므로, 이 모듈에는 validate\_one\_hexstr\_argument, validate\_two\_hexstr\_arguments, validate\_three\_hexstr\_arguments과 같은 인자의 개수에 따라 인자들을 validation 해주는 함수들이 포함되어 있다. 또한 memory와 관련된 모듈이므로, 메모리 주소에 대한 validation 함수인 validate\_adderess와 validate\_range가 있고, memory에 입력할 값을 검증하는 validate\_value 함수도 이 모듈에 존재한다. 이에 더하여 dump 함수 실행 시에 생산성을 높여주기 위해, 함수 내에서 반복되는 구문을 별도의 함수인 print\_chars를 정의해 구분시켜 모듈에 포함시켰다.

* 1. **opcode\_commands**

opcode\_commands는 opcode, opcodelist 등 instruction의 opcode와 관련된 명령어를 처리하기 위한 모듈이다. opcode table은 해시 테이블 형태로 만들어야 하므로, 파일을 읽어들여 해시 테이블을 만드는 init\_hash\_table, mnemonic을 바탕으로 hash 값을 만드는 hash function인 get\_hash\_index, 동적 할당으로 만들어진 해시 테이블의 메모리를 해제하는 free\_hash\_table 등의 함수도 이 모듈에 포함된다.

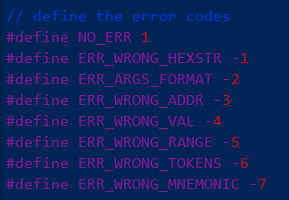
* 1. **20140825**

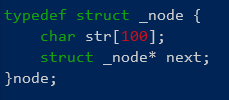
메인 함수가 포함되어 있는 20140825.h와 20140825.c 파일에는 명령어의 처리와 쉘 프로그램 자체를 처리하기 위한 별도의 함수들이 포함되어 있다. 입력으로 들어온 문자열을 파싱하여 tokenize 해주기 위해 필요한 flush\_tokens, get\_next\_token\_idx, tokenize\_input 등의 함수들과, 입력 문자열 중 명령어를 처리하기 위한 get\_command, execute\_instructions와 같은 함수들, 그리고 쉘 프로그램의 시작과 끝의 설정을 처리하기 위한 init과 exit\_program등의 함수들이 해당 파일에 정의되어 있다.

**4. 전역 변수/매크로/데이터 타입 정의**

각 모듈들의 헤더파일들에는 모듈에서 필요로 하는 전역변수나 매크로, 데이터 타입 등이 명시되어 있다.

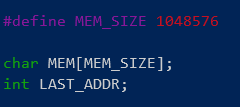
* 1. **helpers.h**

helpers.h에는 명령어 수행 시 발생할 수 있는 에러 코드들이 macro로 정의되어 있다. 이러한 에러 코드들은 명령어 관련 함수 수행 시 발생하는 에러 사항들의 종류들을 나타내어주 고, 함수가 명령어가 제대로 수행되지 않았을 때 반환되어지는 값으로서 명령어 수행에 오류가 있음을 알려주는 역할을 수행한다.

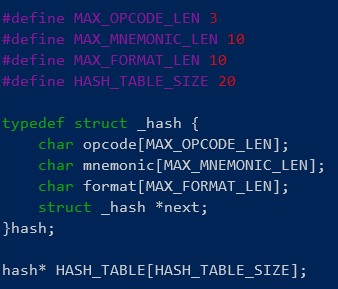
* 1. **shell\_commands.h**

shell\_commands.h에는 history 명령어 수행을 위해, 입력된 명령어들을 linked list 형태로 저장하기 위해 node라는 구조체가 새로운 데이터 타입으로 선언되어 있다.

이렇게 선언한 node 데이터 타입의 포인터 값인 HEAD\_LOG와 TAIL\_LOG를 전역 변수로 선언함으로써, history에서 사용할 linked list를 구현하였다. 각각은 history를 저장하는 linked list의 맨 앞 원소와 맨 뒤 원소를 참조할 수 있도록 해준다.

* 1. **memory\_commands.h**

memory\_commands.h에는 메모리 관련 명령어를 수행하기 위한 가상 메모리 공간을 나타내는 전역변수 MEM이 선언되어 있다. MEM의 크기는 매크로 MEM\_SIZE를 통해 1048576 byte(1MB)로 나타내어 진다. LAST\_ADDR는 dump 명령어 수행 후 저장되는 최근 참조 메모리의 위치를 저장하기 위해 전역변수로 선언되었다.

* 1. **opcode\_commands.h**

opcode\_commands.h에는 해시 테이블 형태로 만들어야 하는 opcode 테이블을 위해 hash라는 새로운 데이터 타입이 선언되어 있다. hash는 해당 instruction의 opcode, mnemonic, format을 담기 위한 문자열과 해당 개체와 동일한 해시 값을 가지는 개체들을 연결하기 위한 hash 포인터 next를 속성 값으로 갖는다. 이를 위해 각 문자열 속성들은 최대 길이값을 가지며, opcode 테이블을 만들기 위해 hash 포인터들의 배열인 HASH\_TABLE 변수를 선언하였다. 이 때 매크로를 통해 해시 테이블의 크기는 20으로 지정했다.

* 1. **20140825.h**

20140825.h에는 메인 함수에서 입력 문자열을 받아 명령어와 인자들을 parsing하는 작업을 하기 위한 데이터 타입들과 전역변수들이 선언되어 있다.

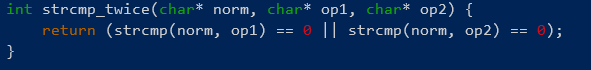
문자열 전역변수 input은 쉘에서 사용자로부터 문자열을 입력받는 버퍼 역할을 하는 변수이다. 이렇게 입력 받은 문자열을 tokenize하여 토큰 별로 저장하는 버퍼가 tokens라는 문자열 전역변수이며, tokens에 토큰이 저장되는 대로 그 개수는 TOKEN\_COUNT에 기록된다.

또한 명령어들을 보다 효율적으로 처리하기 위해 enum type의 command를 새로운 데이터 타입으로 정의하여 사용했다.

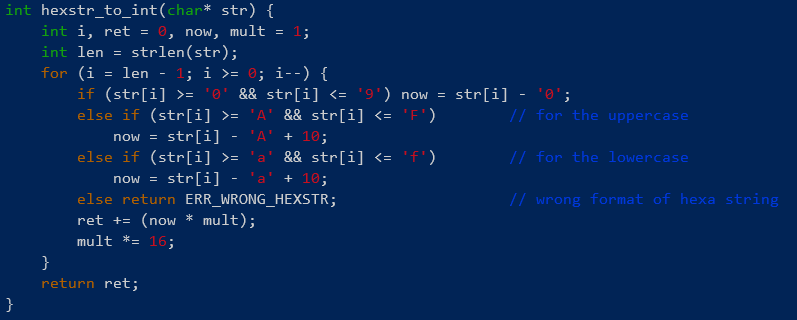
**5. 코드 설명**

각각의 모듈들은 여러 개의 함수를 가지며, 각 함수들의 소스코드와 수행 작업 내용은 다음과 같다.

**5.1 helpers**

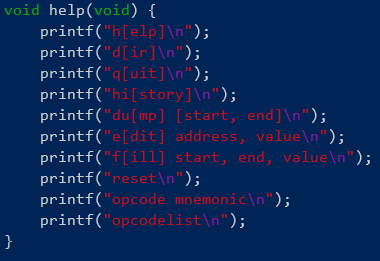
**5.1.2 strcmp\_twice**

비교 기준이 될 하나의 문자열과 비교 대상이 두 개의 문자열, 총 세 개의 문자열을 인자로 받아, 비교 대상이 되는 두 문자열 중 하나의 문자열이라도 비교 기준 문자열과 동일하다면 true(1)를 반환하고, 그렇지 않다면 false(0)를 반환한다.

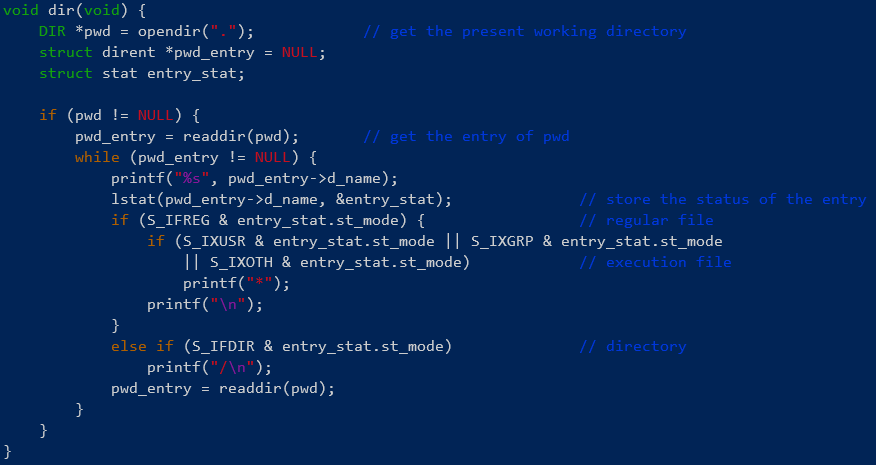
**5.1.2 hexstr\_to\_int**

16진수 형태의 문자열을 인자로 받아 이를 10진수 형태의 실제 값을 가지는 int형 데이터로 변환한 후 그 값을 반환한다. 16진수 형태에 적합하지 않은 문자열일 경우, 매크로로 정의된 ERR\_WRONG\_HEXSTR을 반환한다.

**5.2 shell\_commands**

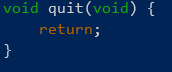
**5.2.1 help**

shell 관련 명령어 중, help 명령어를 처리하기 위한 함수이다. 별다른 argument가 필요 없으며, 함수를 호출하면, shell에서 실행 가능한 모든 명령어들의 리스트를 화면에 출력해준다.

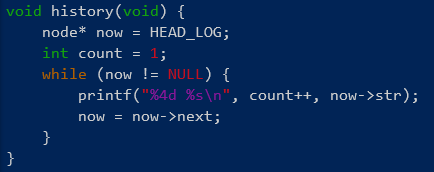
**5.2.2 dir**

shell 관련 명령어 중 dir 명령어를 처리하기 위한 함수이다. help 와 마찬가지로 별다른 인자를 필요로 하지 않으며, 현재 디렉터리에 있는 파일들을화면 상에 출력한다.

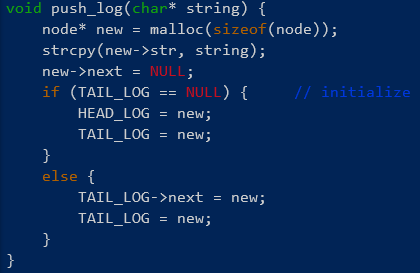
이를 위해서, 우선 dirent.h에 정의되어 있는 opendir 함수를 사용하여 현재 위치의 디렉토리에 대한 DIR 값을 받고, 이와 관련된 dirent 변수를 readdir 함수를 통해 순서대로 하나씩 읽어낸다. 우선 읽어낸 엔트리의 이름을 출력하고, 만약 읽어낸 디렉토리 엔트리(dirent)가 일반 파일이면서 (S\_IFREG) 동시에 실행 권한을 가졌다면(S\_IXUSR, S\_IXGRP, S\_IXOTH) 추가로 ‘\*’을 출력하고, 읽어낸 디렉토리 엔트리가 일반 파일이 아닌 디렉토리(S\_IFDIR)라면 ‘/’를 추가로 출력한다. 최종적으로 파일 이름을 모두 출력했다면 개행문자를 출력한다. 이 과정을 현재 디렉토리 내에서 모든 dirent를 읽어낼 때 까지 계속 반복한다.

**5.2.3 quit**

shell 관련 명령어 중 quit 명령어를 처리하기 위한 함수로, 함수가 호출되면 별다른 작업 없이 함수를 바로 종료한다.

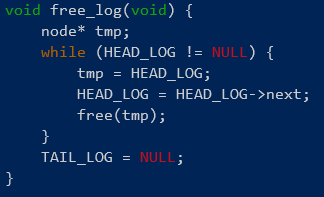
**5.2.4 history**

shell 관련 명령어 중 history 명령어를 처리하기 위한 함수로, 현재까지 사용한 명령어를 사용한 순서대로 번호와 함께 화면에 출력한다. 함수가 호출되면 HEAD\_LOG 변수를 통해 명령어 log가 기록된 linked list에 접근하며, 해당 linked list가 끝날 때까지 명령어 log들을 출력한다.

**5.2.5 push\_log**

history 함수 수행 시 나타나는 명령어의 log들을 기록하기 위해, 수행한 명령어를 linked list 형태의 log에 추가하는 함수이다.

만약 현재 로그가 비어 있다면, 동적할당을 통해 새롭게 만든 log(node\*)가 HEAD\_LOG이면서 동시에 TAIL\_LOG가되도록 한다. 만약 그렇지 않다면 새 log를 새로운 TAIL\_LOG로 지정함으로써, 최근에 발생한 log일수록 history 함수에서 더 나중에 출력될 수 있도록 한다.

**5.2.6 free\_log**

push\_log 함수를 통해 동적할당 되었던 메모리를 다시 해제해주는 역할을 수행하는 함수이다. HEAD\_LOG가 가리키는 log(node\*)가 존재하지 않을 때까지 반복문을 수행해 linked list에 연결되어 있던 모든 동적할당 메모리들을 해제한다.

**5.3 memory\_commands**

**5.3.1 dump**

memory 관련 명령어 중 dump 명령어를 처리하기 위한 함수로, 할당되어 있는 메모리의 내용을 왼쪽의 그림과 같이 일정한 형식으로 출력시켜 주는 함수이다.

이 형식에서 왼쪽의 칼럼은 출력하는 메모리의 offset을 5자리 16진수로 나타내어주며, 가운데 칼럼은 메모리의 내용을 16진수 형태로, 오른쪽 칼럼은 메모리의 내용을 byte 별로 대응하는 아스키 코드 형태로 보여준다.

함수의 내용은 크게 전반부와 후반부 두 부분으로 나누어 볼 수 있다. 함수의 전반부에는 왼쪽 그림과 같이 입력된 토큰의 개수에 따라, 각 argument들의 형식 상 유효성을 검증하는 로직이 작성되어 있다.

dump의 argument가 하나도 들어오지 않은 경우, 최근에 참조했던 주소가 저장된 전역 변수 LAST\_ADDR의 다음 바이트 주소를 시작 주소로 하고, 이로부터 10라인을 출력할 수 있는 끝 주소를 계산해 저장한다. 만약 각 주소 값이 유효한 메모리 주소의 범위를 넘을 경우, 각각의 주소 값에 0과 MEM\_SIZE – 1 이라는 기본 값을 할당한다.

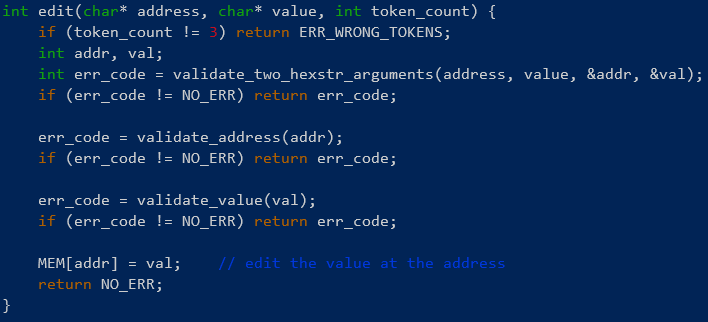
argument가 하나만 들어올 경우, 들어온 start 변수가 알맞은 16진수 형태로 들어왔는지 우선 확인한 뒤, 변환된 시작 주소 값을 기준으로 끝 주소 값을 지정해준다. Argument가 두개가 들어올 경우, 두 변수가 올바른 형태로 들어왔는지 확인한 후, 오류가 없다면 10진수로 변환된 값을 시작과 끝 주소값으로 사용한다.

시작 주소와 끝 주소 값의 변환이 끝났다면, 두 주소를 사용하는 것이 전체 메모리의 유효한 범위를 넘어가는지 아닌지를 검토한다. 앞선 모든 변수의 유효성 검사에서 오류가 하나라도 발생한다면, 더 이상의 작업을 중단하고, 오류코드를 반환한다.

함수의 후반부에는 검증된 시작 주소와 끝 주소 값을 바탕으로 정해진 형식에 맞추어 올바른 메모리 값들을 출력하는 로직이 작성되어 있다.

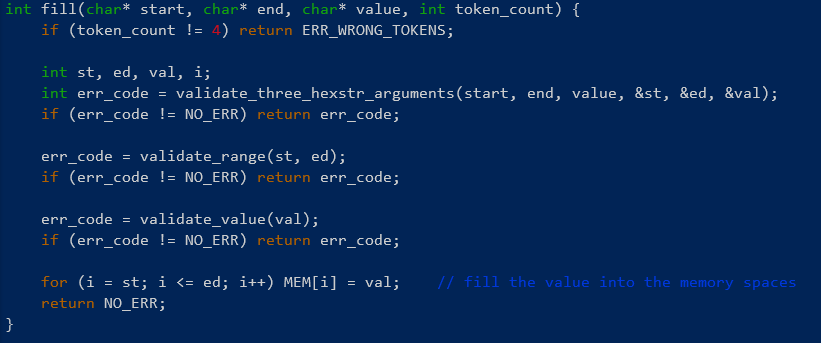
우선 검증된 10진수 시작 주소값 st,와 끝 주소값 ed를 기준으로 dump를 시작하고 끝낼 행(st\_row, ed\_row)을 계산한다. 이를 토대로 for 문을 돌려 각 행이 시작될 때마다 해당 row의 오프셋을 5자리 16진수 형태로 출력한다. 행 별로 오프셋을 출력한 뒤에는 출력하는 메모리가 시작과 끝 주소값을 벗어날 경우에는 공백 문자를 출력하고, 그렇지 않다면 2자리 16진수 형태로 해당 주소에 있는 값들을 출력한다.

해당 행의 메모리 값 출력이 끝나면 print\_chars 함수를 사용해 해당 값을 아스키 코드라고 했을 때 출력될 수 있는 문자들을 출력한다. 모든 출력이 끝나면 전역변수 LAST\_ADDR에 이번에 참조했던 마지막 메모리 주소 값을 저장하고, 문제 없이 작업이 끝났음을 매크로 NO\_ERR를 반환함으로써 나타내어 준다.

**5.3.2 edit**

memory 관련 명령어 중 edit 명령어를 처리하기 위한 함수로, 특정 주소의 메모리에 특정 값을 저장하기 위한 함수이다.

우선 edit 명령어는 두 개의 argument를 필수적으로 받아야 하므로, 이 조건을 충족하지 못할 경우, 오류 코드 매크로를 반환하며 작업을 중단한다. 이후, 각 argument들이 올바른 형식으로 입력되었는지를 확인 후 이를 16진수에서 10진수 형태의 int 값으로 변환하고, 각 값이 주소와 저장할 바이트 값으로 적절한지 검사한다. 모든 유효성 검사에서 문제가 발생할 경우 작업을 중단하고 오류 코드 매크로 값을 반환한다. 유효성 검사를 모두 통과했다면, 해당 메모리 주소에 해당 값을 저장한 뒤, 작업이 문제없이 끝났음을 NO\_ERR 매크로를 반환함으로써 나타내어 준다.

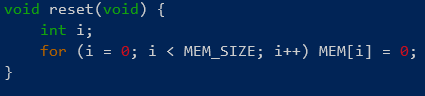
**5.3.3 fill**

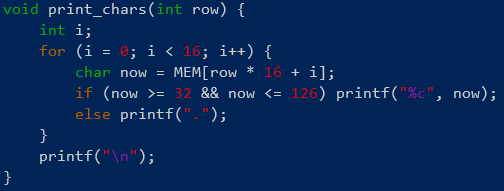
memory 관련 명령어 중 fill 명령어를 처리하기 위한 함수로, 특정 범위의 메모리들의 값을 일괄적으로 특정 값으로 바꿔주는 함수이다.

argument를 받는 다른 함수들과 마찬가지로, fill 역시 필수적으로 받아야 하는 세 개의 argument에 대한 유효성 검사를 진행하고, 오류 발생 시 작업을 중단하고 에러 코드를 반환한다. 각각의 argument는 주소의 범위를 나타내는 start, end, 값을 나타내는 value이므로, 주소와 값에 대한 유효성 검사 역시 같은 방식으로 진행한다.

유효성 검사를 모두 통과하면 해당 범위의 주소에 해당 값을 저장하고, 작업이 문제없이 끝났음을 NO\_ERR 매크로를 반환함으로써 나타내어 준다.

**5.3.4 reset**

memory 관련 명령어 중 reset 명령어를 처리하기 위한 함수로, 가상 메모리 전체의 값들을 ‘\0’으로 초기화 하는 함수이다.

**5.3.5 print\_chars**

dump 함수에서 메모리 값들을 char 형태로 출력하기 위해 사용하는 함수이다.

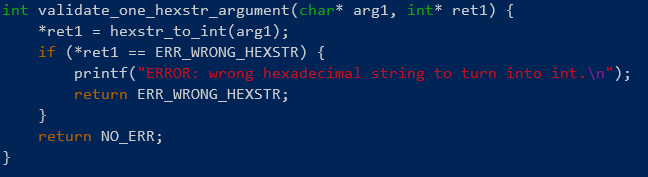
입력된 행의 값에 따라 주어진 행에 해당하는 16개 메모리의 값들을 char 형태로 출력한다. 이 때, 그 값이 0x20(32), 0x7E(126) 사이에 해당하는 경우에만 해당 문자 형태로 출력하고, 아닌 경우에는 ‘.’을 출력한다.

**5.3.6 validate\_hexstr\_argumetns functions**

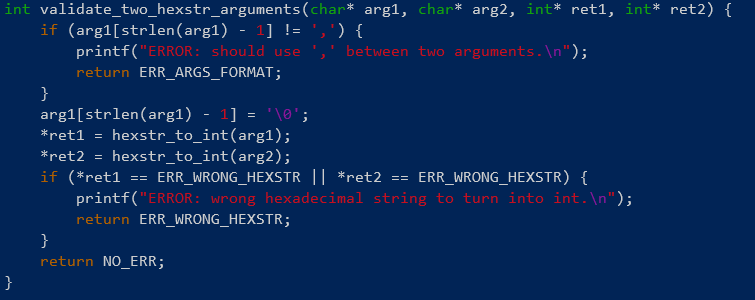
아래 세 함수는 16진수 형태의 문자열 argument들에 대한 유효성 검사를 그 개수에 따라 유효성 검사 방식을 여러 개의 함수로 다변화한 것이다.

변수들 간 관계를 나타내어 주는 ‘,’의 존재 여부를 통해 argument 입력 형식 상 유효성을 검사하고, 변수들의 값을 10진수 형태로 바꿔주면서 두 자리 16진수의 형태에 대한 유효성을 검사한다.

유효하지 않은 결과가 발생한 경우, 각 경우에 맞게 에러 메시지를 출력하고 에러 코드를 반환한다. 유효성 검사를 모두 통과했다면 NO\_ERR 매크로 값을 반환하고, 또 포인터로 참조된 리턴 변수들에 10진수로 변환한 argument들의 값을 저장해 해당 함수 바깥에서 변환된 값을 사용할 수 있도록 한다.

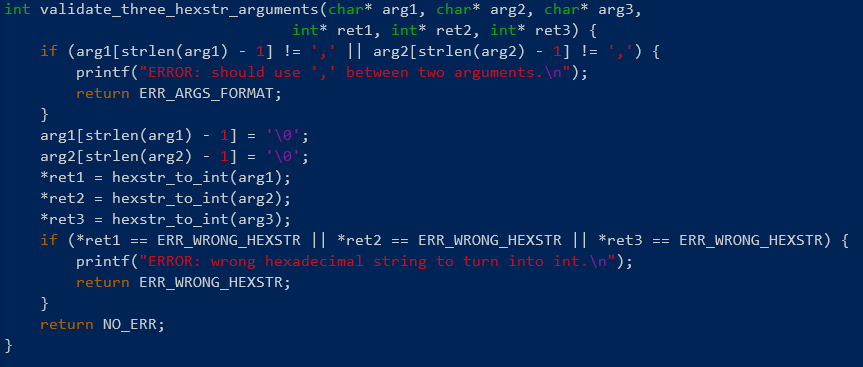
 **5.3.6.1 validate\_one\_hexstr\_argument**

하나의 argument가 들어온 경우의 인자에 대한 유효성 검사를 수행한다. 16진수 형태의 유효성 검사 및 값 변환을 수행한다.

 **5.3.6.2 validate\_two\_hexstr\_arguments**

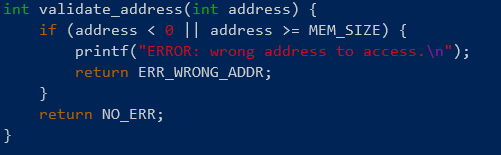
두 argument가 들어온 경우의 인자들에 대한 유효성 검사를 수행한다.

두 변수 간에 ‘,’가 존재해야 하는 형식 상의 유효성과 두 인자들의 16진수 형태의 유효성 검사 및 값 변환을 수행한다.

 **5.3.6.3 validate\_three\_hexstr\_arguments**

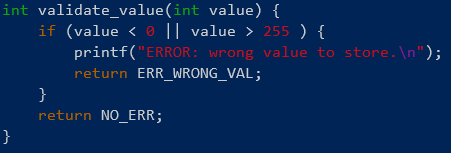
세 개의 argument가 들어온 경우의 인자들에 대한 유효성 검사를 수행한다.

세 변수들 중 연속된 두 변수들 간에 ‘,’가 존재해야 하는 형식 상의 유효성과 세 인자들에 대한 16진수 형태의 유효성 검사 및 값 변환을 수행한다.

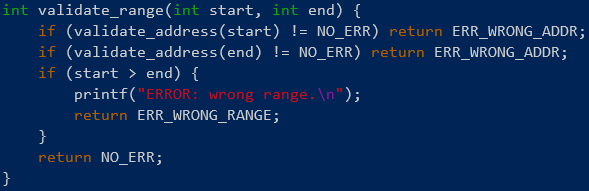
**5.3.7 validate\_address**

주어진 가상 메모리 주소값이 유효한 범위 내에 존재하는지를 검사하고, 유효하지 않다면 에러 메시지를 출력한 뒤 해당 오류 코드를 반환한다. 이는 segmentation fault 오류가 발생하는 것을 사전에 막기 위함이다.

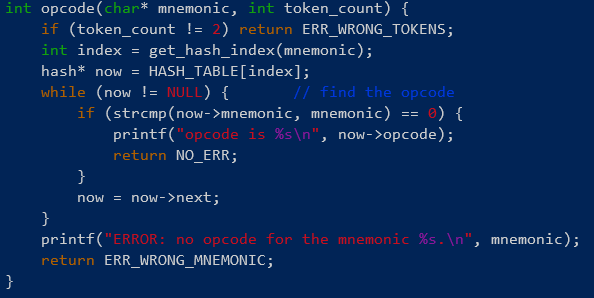
**5.3.8 validate\_value**

주어진 값이 메모리에 저장하기에 적절한 범위 내에 존재하는 지를 검사하고, 유효하지 않다면 에러 메시지를 출력하고 해당 오류 코드를 반환한다. 각 가상 메모리는 1 바이트(0 ~ 256)에 해당하므로 이 범위를 넘어갈 수는 없다.

**5.3.9 validate\_range**

주어진 두 주소 값으로 이루어지는 범위가 가상 메모리 상의 주소 범위를 넘어가는지 여부에 대하여 유효성 검사를 수행한다. 주소 범위가 유효하지 않다면, 에러 메시지를 출력하고 해당 오류 코드를 반환한다.

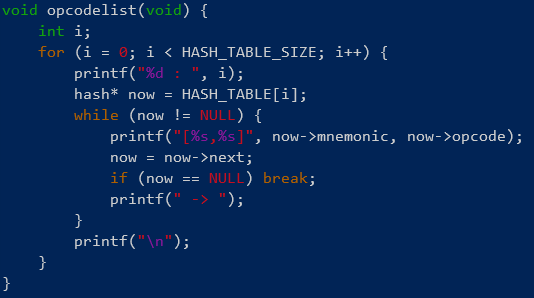
**5.4 opcode\_commands**

**5.4.1 opcode**

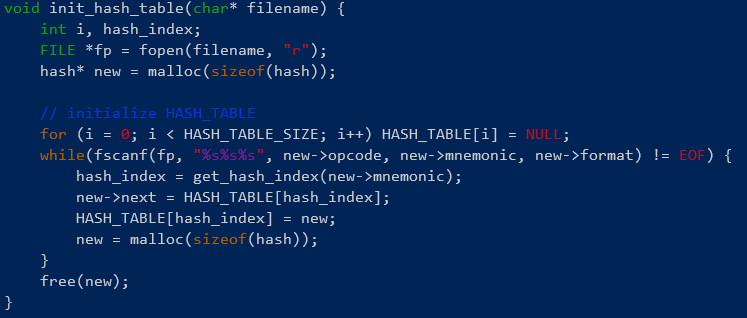
opcode 테이블과 관련된 명령어 중 명령어 opcode의 수행을 처리하기 위한 함수로, mnemoic을 인자로 받아 해당하는 instruction의 opcode를 반환한다.

우선, 필수적으로 하나의 argument를 받아야 하므로, 이를 만족하지 못할 경우 에러 코드를 반환하고 작업을 중단한다. 인자 개수에 대한 유효성 검사가 끝나면, 해당 mnemonic이 갖는 hash index 값을 받아 hash table에서 해당 인덱스에 해당하는 hash\* 변수에 접근한다. 이 후, 해당 mneomnic과 동일한 값을 자신의 mnemonic으로 가지는 hash 변수를 찾고, 해당 변수의 opcode 값을 출력한다.

만약 찾지 못했다면, 존재하지 않는 mnemonic을 입력한 것이므로, 관련한 에러 메시지를 출력하고 에러 코드를 반환한다.

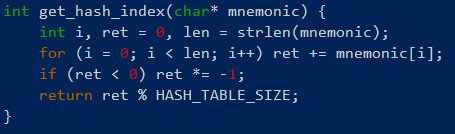
**5.4.2 opcodelist**

opcode 테이블과 관련된 명령어 중 명령어 opcodelist의 수행을 처리하기 위한 함수로, hash table 내에 존재하는 모든 명령어들의 mnemoic과 opcode를 그 hash 값에 따라 순차적으로 출력한다.

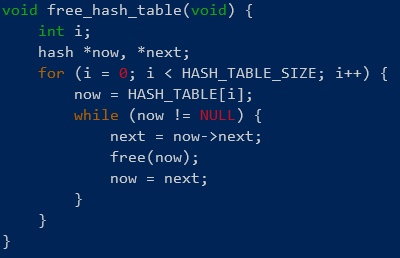
**5.4.3 init\_hash\_table**

hash table 형태의 opcode 테이블을 만드는 함수이다. 우선 opcode 테이블에 대한 정보가 담긴 파일의 이름을 인자로 받아, 해당 파일을 열어 opcode 테이블에 대한 정보를 읽어들인다.

새로운 hash 변수를 동적할당하여 읽어들인 정보들을 새 변수에 저장하고, 저장된 정보 중 mnemonic을 이용하여 hash 값을 만들어낸다. hash table에서 해당 hash 값을 인덱스로 가지는 hash 포인터 변수를 불러오고, linked list 형태로 연결된 해당 hash 포인터의 맨 앞에 새로 할당된 변수를 추가한다. 이러한 작업을 읽어들인 파일이 끝날 때까지 반복함으로써 opcode table을 구현한다.

**5.4.4 get\_hash\_index**

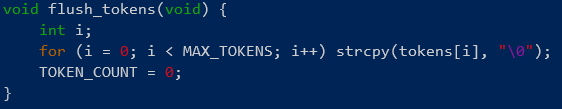
hash table의 인덱스를 만들어 내기 위한 hash function으로, 입력받은 mnemonic을 바탕으로 hash 값을 반환한다.

**5.4.5 free\_hash\_table**

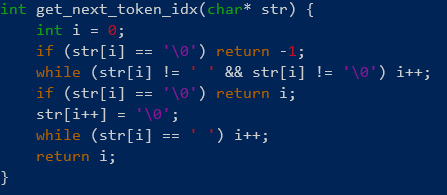
동적할당 했던 hash table의 hash 변수들에 대해 메모리를 해제해주는 함수이다. 프로그램 종료 시 memory leak을 방지하기 위해 호출된다.

**5.5 20140825**

**5.5.1 flush\_tokens**

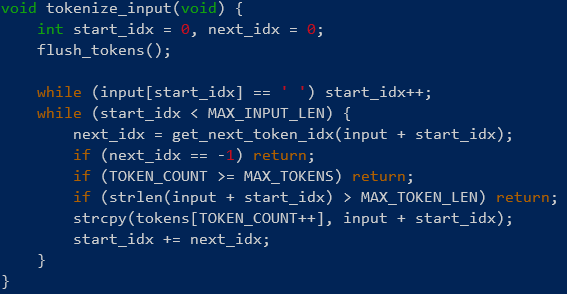
기존에 사용했던 전역변수 tokens에 담겨있던 토큰 문자열들을 모두 비워내주고, 전역변수 TOKEN\_COUNT 값을 0으로 다시 초기화 해주는 역할을 한다.

**5.5.2 get\_next\_token\_idx**

argument로 받은 str 내에 존재하는 하나의 토큰을 찾아 이를 기존 str에서 분리시켜주고, str 내에서 다음 토큰이 될 문자열의 str 상 첫번째 인덱스 값을 반환해주는 함수이다.

우선, 문자열의 첫 문자가 null 문자인 경우, 더 이상 해당 문자열에 토큰이 존재하지 않는다는 의미로 -1을 반환한다. 이후, 공백문자 또는 null 문자를 만날 때까지 str의 인덱스를 증가시켜주고, 반복문이 종료되면 해당 토큰을 분리시켜주기 위해 null 문자를 해당 토큰의 다음 문자에 할당해 준다. 이후 뒤의 문자열에서 공백이 발견된다면 공백문자가 발견되지 않을 때까지 인덱스를 증가시켜 준 뒤, 해당 인덱스를 반환해준다.

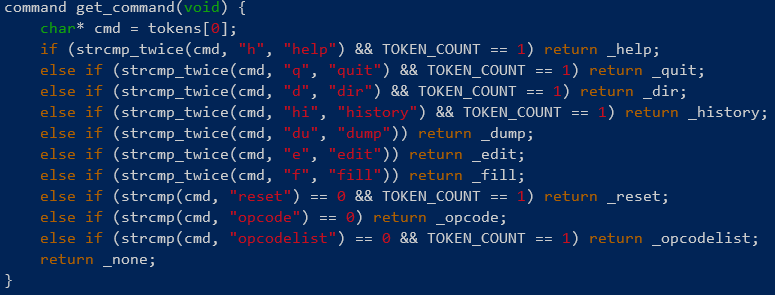
**5.5.3 tokenize\_input**

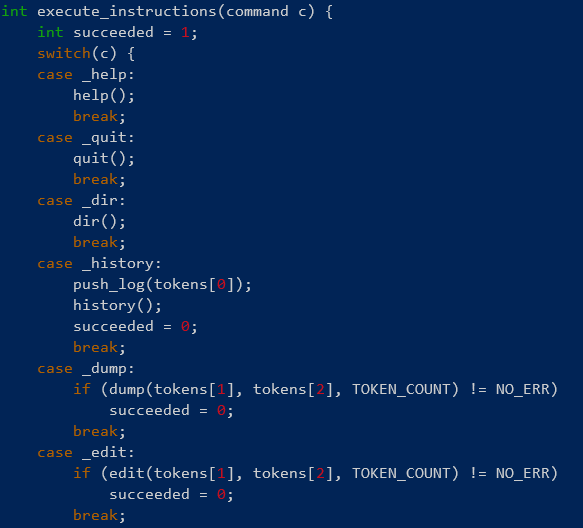
전역변수 input을 여러 개의 토큰으로 분리시켜 각각의 토큰을 전역변수 tokens에 저장해주는 작업을 수행한다.

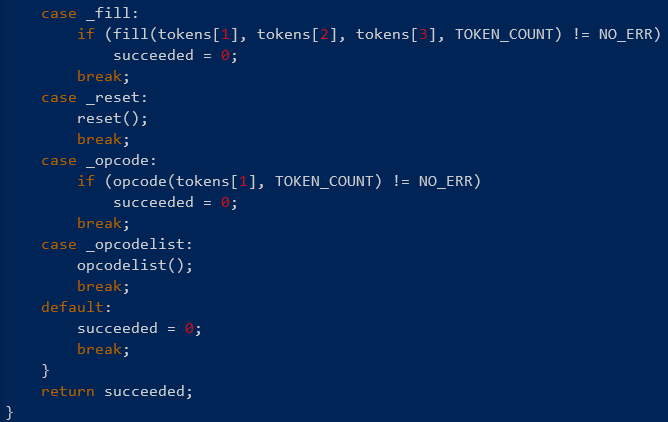
우선 flush\_tokens를 호출해 tokens의 값들을 초기화해주고, 들어온 input의 앞에 존재하는 모든 공백문자들을 trim 해준다.

이후, 반복문에서 get\_next\_token\_idx를 반복 수행함으로써, 토큰들을 input 문자열에서 분리해 tokens에 저장해주고, 동시에 token들의 개수를 count 하여 전역변수 TOKEN\_COUNT에 저장해준다. 반복문 수행 도중, 전역변수 tokens나 input의 메모리 범위를 넘어가는 경우로 segmentation fault가 발생해 프로그램이 종료 되는 일이 발생하지 않도록, 예외 상황을 지정하여 함수를 종료시키는 로직도 추가되어 있다.

**5.5.4 get\_command**

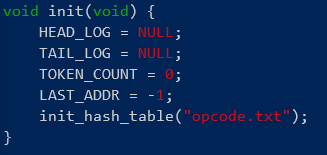
strcmp\_twice와 strcmp 함수를 사용하여, 입력으로 들어온 명령어가 올바른 형태로 들어왔는지를 판단하여 해당 커맨드에 해당하는 enum 타입의 command 변수 값을 반환한다.

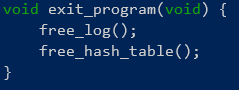
**5.5.5 execute\_instructions**



command 타입의 변수를 인자로 받아 해당 command에 해당하는 함수를 실행한다. 단, history의 경우, 함수 수행 이전에 push\_log 함수를 수행해 history 명령어를 먼저 기록한다. 또한 dump, edit, fill, reset, opcode 등 인자를 추가로 필요로 하는 명령어들의 경우, 인자들과 관련하여 오류 상황이 발생한다면, 해당 작업이 성공적으로 수행되었다고 볼 수 없으므로, 그런 상황이 발생할 경우에 지역변수 succeeded의 값을 0으로 지정해준다. 아무런 명령어에 해당하지 않는 경우도 마찬가지로 succeeded 값을 0으로 지정해주며, 마지막으로 succeeded 값을 반환하며 함수의 수행을 종료한다.

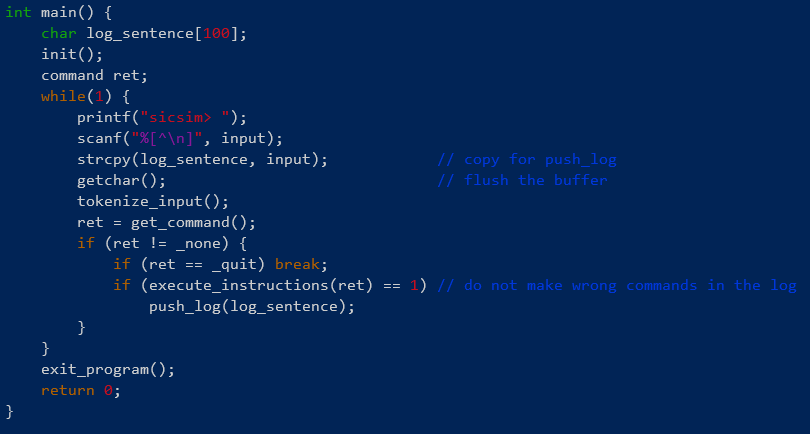
**5.5.6 init**

프로그램이 실행되고 유저로부터 입력값을 받기 이전, 준비를 위해 호출되는 함수이다. 각 모듈들에서 선언된 전역변수들의 값에 대하여 초기화를 진행한다. 또한 opcode table을 만들기 위해 init\_hash\_table을 호출한다.

**5.5.7 exit\_program**

프로그램의 실행이 마무리되고, 동적으로 할당된 메모리들을 해제하기 위해 free\_log와 free\_hash\_table을 호출해주는 역할을 하는 함수다.

**5.5.8 main**

프로그램의 실제 수행을 담당하는 메인함수이다. 우선 init을 호출하여 프로그램 수행을 위한 초기 세팅을 진행하고, while문을 사용해 quit 명령어가 입력될 때까지 명령어를 입력 받는다. 이후, 로그를 기록하기 위해 log\_sentence에 입력 값을 복사해 저장하며, 해당 명령어가 성공적으로 수행된 경우에만, push\_log를 호출하여 해당 입력 문자열을 log에 저장한다.

quit 명령어의 입력으로 반복문이 종료되면, 프로그램 실행 도중 동적 할당된 메모리를 해제하는 등의 마무리 작업을 수행해주기 위하여 exit\_program을 호출한 뒤, 메인 함수를 반환한다.