과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 1>>

**서강대학교 컴퓨터공학과**

**20171601**

**강주형**

**목 차**

1. 프로그램 개요 4

2. 프로그램 설명 4

2.1 프로그램 흐름도 4

3. 모듈 정의 5

3.1 모듈 이름 : main() 5

3.1.1 기능 5

3.1.2 사용 변수 5

3.2 모듈 이름: create\_opcode() 5

3.2.1 기능 5

3.2.2 사용 변수 5

3.3 모듈이름: insert() 5

3.3.1 기능 5

3.3.2 사용변수 5

3.4 모듈이름: check1() 5

3.4.1 기능 5

3.4.2 사용변수 5

3.5 모듈이름: check2() 5

3.5.1 기능 5

3.5.2 사용변수 6

3.6 모듈이름: record() 6

3.6.1 기능 6

3.6.2 사용변수 6

3.7 모듈이름: help() 6

3.7.1 기능 6

3.7.2 사용변수 6

3.8 모듈이름: dir() 6

3.8.1 기능 6

3.8.2 사용변수 6

3.9 모듈이름: quit() 6

3.9.1 기능 6

3.9.2 사용변수 6

3.10 모듈이름: history() 7

3.10.1기능 7

3.10.2 사용변수 7

3.11 모듈이름: dump() 7

3.11.1 기능 7

3.11.2 사용변수 7

3.12 모듈이름: edit() 7

3.12.1 기능 7

3.12.2 사용변수 7

3.13 모듈이름: fill() 7

3.13.1 기능 7

3.13.2 사용변수 7

3.14 모듈이름: reset() 7

3.14.1 기능 7

3.14.2 사용변수 7

3.15 모듈이름: mnemonic() 8

3.15.1 기능 8

3.15.2 사용변수 8

3.16 모듈이름: opcodelist() 8

3.16.1 기능 8

3.16.2 사용변수

4. 전역 변수 정의 8

4.1 Node \*head 8

4.2 Op \*ophead[20] 8

4.3 unsigned char MEMORY[MEMORY\_SIZE] 8

4.4 unsigned int address 8

5. 코드 8

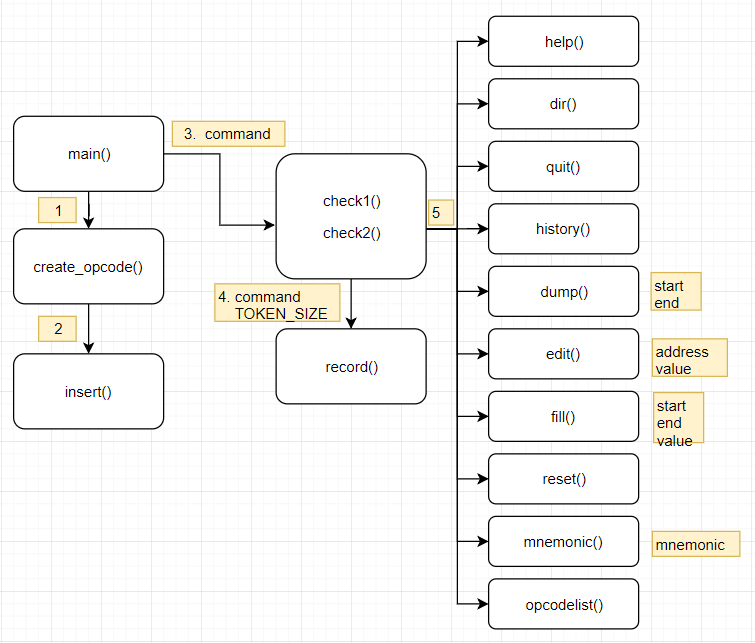
5.1 20171601.h 8

5.2 20171601.c 9

1. **프로그램 개요**

이 프로그램은 SIC/XE 기계를 실행하기 위한 예비 단계다. 이번 프로그램에서는, 유저 명령을 실행하는 쉘 환경, 오브젝트 코드가 로드되는 메모리 공간, opcode를 opcode table로 변환하는 opcode table 등을 실시할 예정이다.

프로그램이 실행되면 “sicsim> ” 형태의 쉘 프롬프트가 화면에 출력된다. 쉘은 다양한 종류의 명령어를 받아들여야 하며 이를 실행시켜야 한다. 명령어에는 크게 쉘 관련 명령어(help, dir, quit, history), 메모리 관련 명령어(dump, edit, fill, reset), opcode에 관련된 명령어(opcodelist, opcode mnemonic)가 있으며, history와 opcodelist 명령어의 관리는 linked list를 이용한다.

1. **프로그램 설명**
   1. ** 프로그램 흐름도**

**그림 1> 프로그램 흐름도**

1. **모듈 정의**
   1. **모듈 이름 : main()**
      1. *기능*

Display\_instructions를 호출해 프로그램이 하는 일을 설명한 후, 사용자의 입력을 받아서 나머지 모듈을 호출하거나 끝낸다.

* + 1. *사용 변수*

없음

* 1. **모듈 이름: create\_opcode()**
     1. *기능*

opcode.txt를 읽어와 hash table을 만들어 이를 linked list 형식으로 저장한다.

* + 1. *사용 변수*

FILE \*fp : opcode.txt.를 읽기 위한 파일포인터

int I : 반복문을 위한 변수

Op \*node : opcode의 명령어 정보를 저장하기 위한 구조체

* 1. **모듈이름: insert(Op \*node)**
     1. *기능*

create\_opcode() 함수에서 hash table 구성 시 데이터들을 linked list로 연결하기 위한 함수

* + 1. *사용변수*

int I : 반복문을 위한 변수

int key : 연산을 통해 얻어진 key값 저장

int num : 입력받은 명령어의 길이를 저장하는 변수

int sum : key값 생성을 위한 변수 & 명령어의 아스키 값의 합을 저장

Op \*new : 새롭게 읽어들인 명령어 정보를 저장하기 위한 구조체

* 1. **모듈이름: check1(char \*origin, char \*\*Command)**
     1. *기능*

TOKEN\_SIZE=1일 경우 명령어를 받아와 이가 shell에서 제공하는 명령어일 경우 이를 수행하는 함수로 이동한다.

* + 1. *사용변수*

unsigned int start : dump 명령어가 주어졌을 때 시작 주소를 저장하기 위한 변수

unsigned int end : dump 명령어가 주어졌을 때 끝 주소를 저장하기 위한 변수

* 1. **모듈이름: check2(char \*origin, char \*\*command, int TOKEN\_SIZE)**
     1. *기능*

TOKEN\_SIZE가 2개 이상일 경우 명령문을 받아와 조건문을 통해 확인한 후 이를 수행하는 함수로 이동한다.

* + 1. *사용변수*

unsigned int start : dump,fill 명령어에서 시작 주소를 저장하기 위한 변수

unsigned int end : dump,fill 명령어에서 끝 주소를 저장하기 위한 변수

unsigned int addr : edit 명령어에서 값을 수정하고자 하는 주소를 저장하기 위한 변수

unsigned int new : edit,dill 명령어에서 새로운 값을 저장하기 위한 변수

unsigned int trash : dump, edit, fill 명령어에서 필요없는 값을 저장하기 위한 변수

int I : 반복문 수행을 위한 변수

int count : 컴마의 개수를 세기 위한 변수

* 1. **모듈이름: record(char \*origin, char \*\*command, int TOKEN\_SIZE)**
     1. *기능*

입력받은 명령어를 linked list 형태로 저장한다.

* + 1. *사용변수*

Node \*node : 입력 받은 명령어의 정보를 저장하는 구조체

Node \* tmp : linked list의 head 역할을 하는 변수

* 1. **모듈이름: help()**
     1. *기능*

shell이 지원하는 모든 명령어를 보여준다.

* + 1. *사용변수*

int ask - 사용자의 입력을 받는 저장공간

* 1. **모듈이름: dir()**
     1. *기능*

현재 디렉토리에 있느 파일 및 폴더를 보여준다.

* + 1. *사용변수*

DIR \*dir\_info : 디렉터리 내에 파일들을 가리킬 포인터

struct dirent \*dir\_entry : 디렉터리 내에 파일의 정보를 담을 구조체

strcut stat buf : 디렉터리 내의 파일의 형식을 담을 구조체

int space : 한 줄에 3개의 이름이 출력되면 다음 줄로 출력하고자 만든 변수

* 1. **모듈이름: quit()**
     1. *기능*

프로그램 종료 전 동적할당 된 메모리들을 free시킨다.

* + 1. *사용변수*

Node \*ptr1 : record()를 통해 저장된 명령어들의 linked list를 해제하기 위한 포인터

Op \*ptr2 : insert()를 통해 저장된 linked list형태의 opcodelist를 해제하기 위한 포인터

int I : 반복문 수행을 위한 변수

* 1. **모듈이름: history()**
     1. *기능*

이전의 유효한 명령들을 순서대로 보여준다.

* + 1. *사용변수*

int count : 입력 순서를 저장하기 위한 변수

Node \*tmp : 입력받은 명령어들을 출력하기 구조체

* 1. **모듈이름: dump(unsigned int start, unsigned int end)**
     1. *기능*

입력받은 start 번지와 end 번지를 이용, 프로그램에서 관리하는 MEMORY를 start 번지부터 end 번지까지 출력하는 모듈. 출력 라인 왼쪽에는 16진수 번지수가 나타나며 라인당 16개씩 16진수값으로 출력한다. 또한 한 라인 출력이 끝나면 오른편에 MEMORY 값에 해당하는 ASCII 코드를 출력한다 출력 범위는 16진수로 20~7E까지다.

* + 1. *사용변수*

int I : 반복문 수행을 위한 변수

int j : 반복문 수행을 위한 변수

unsigned int start\_h : 시작주소를 16진수 형태로 저장한 변수

unsigned int end\_h : 끝 주소를 16진수 형태로 저장한 변수

* 1. **모듈이름: edit(unsigned int origin, unsigned int new)**
     1. *기능*

address의 값을 입력받은 value 값으로 수정한다.

* + 1. *사용변수*

없음

* 1. **모듈이름: fill(unsigned int start, unsigned int end, unsigned int new)**
     1. *기능*

start address부터 end address까지의 값을 입력받은 value값으로 수정한다.

* + 1. *사용변수*

없음

* 1. **모듈이름: reset()**
     1. *기능*

모든 메모리의 값을 0으로 수정한다.

* + 1. *사용변수*

int I : 반복문 수행을 위한 변수

* 1. **모듈이름: mnemonic(char \*origin, char \*\*command)**
     1. *기능*

입력받은 명령어에 해당하는 opcode를 출력한다.

* + 1. *사용변수*

int I :반복문 수행을 위한 변수

Op \*tmp : linked list 탐색을 위한 변수

* 1. **모듈이름: opcodelist()**
     1. *기능*

opcode의 hash table의 내용물을 출력한다.

* + 1. *사용변수*

int I : 반복문 수행을 위한 변수

int count : 좌측에 인덱스 출력을 위한 변수

Op \*tmp ; linked list 탐색을 위한 포인터

1. **전역 변수 정의**
   1. **Node \*head**

history에 저장할 linked list의 head의 위치를 가리킬 포인터

* 1. **Op \*ophead[20]**

opcode.txt 파일을 읽어 opcodelist를담을 20개의 사이즈 hash table head 포인터

* 1. **unsigned char MEMORY[MEMORY\_SIZE]**

shell 내에서 사용하는 1MB의 MEMORY의 값을 담을 변수

* 1. **unsigned int address**

dump 명령어에서 start, end 주소 모두가 주어진 경우를 제외하고 출력을 시작할 위치와 출력을 끝낸 후 다음 명령어 수행 시 출력시작 위치를 저장할 변수

1. **코드**

**5.1**  **20171601.h**

/\*정의되는 상수\*/

#define INPUT\_SIZE 500

#define MEMORY\_SIZE 0x100000

/\*history에 사용될 linked list node\*/

typedef struct \_Node {

char a[INPUT\_SIZE];

struct \_Node \*list;

}Node;

/\*opcodelist에 사용될 linked list node\*/

typedef struct \_Op {

int num;

char inst[10];

char type[5];

struct \_Op \*list;

}Op;

/\*함수 원형\*/

void opcodelist();

void mnemonic(char \*origin,char \*\*command);

void create\_opcode();

void insert(Op \*node);

void check1(char \*origin,char \*\*command);

void check2(char \*origin,char \*\*command, int TOKEN\_SIZE);

void quit();

void record(char \*origin,char \*\*command, int TOKEN\_SIZE);

void help();

void history();

void dir();

void dump(unsigned int start, unsigned int end);

void edit(unsigned int origin, unsigned int new);

void fill(unsigned int start, unsigned int end, unsigned int new);

void reset();

**5.2**  **20171601.c**

/\*포함되는 파일\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include "20171601.h"

/\*전역 변수\*/

Node \*head;

Op \*ophead[20];

unsigned char MEMORY[MEMORY\_SIZE];

unsigned int address;

/\*프로그램 시작\*/

void main()

{

create\_opcode();

char command[INPUT\_SIZE] = { "\0" };

char command\_temp[INPUT\_SIZE] = { "\0" };

char \*\*token;

char \*ptr;

int flag = 0;

int TOKEN\_SIZE;

int i;

while (!flag)

{

TOKEN\_SIZE = 1;

printf("sicsim> ");

fgets(command, sizeof(command), stdin);

command[strlen(command) - 1] = '\0';

strcpy(command\_temp, command);

ptr = strtok(command\_temp, " \t\n");

if (!ptr) continue;

while (1)

{

ptr = strtok(NULL, " \t\n");

if (!ptr) break;

TOKEN\_SIZE++;

}

token = (char \*\*)malloc(sizeof(char \*)\*(TOKEN\_SIZE + 1));

strcpy(command\_temp, command);

ptr = strtok(command\_temp, " \t\n");

if (!ptr) continue;

token[0] = (char \*)malloc(sizeof(char) \* (INPUT\_SIZE));

strcpy(token[0], ptr);

for (i = 1; i < TOKEN\_SIZE; i++)

{

ptr = strtok(NULL, " \t\n");

if (ptr == NULL) break;

token[i] = (char \*)malloc(sizeof(char) \* (INPUT\_SIZE));

strcpy(token[i], ptr);

}

if (strcmp(\*token, "quit") == 0 || strcmp(\*token, "q") == 0)

{

quit();

flag = 1;

}

else

{

if (TOKEN\_SIZE == 1)

check1(command,token);

else if (TOKEN\_SIZE >= 2)

check2(command,token, TOKEN\_SIZE);

}

for (i = 0; i < TOKEN\_SIZE; i++)

{

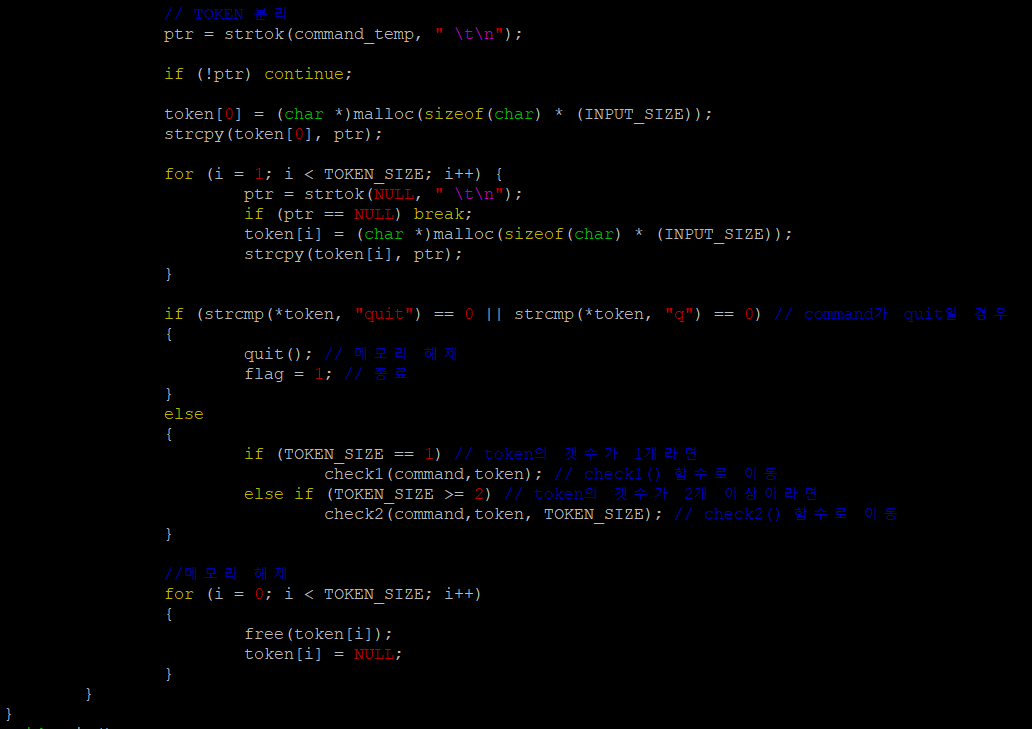
free(token[i]);

token[i] = NULL;

}

}





/\*------------------------------------------------------------------------------------\*/

/\*함수 : create\_opcode()

/\*목적 : opcode.txt를 읽어서 hash table을 만든다.

/\*리턴값 : 없음

/\*------------------------------------------------------------------------------------\*/

void create\_opcode()

{

FILE \*fp = fopen("opcode.txt", "r");

int i;

Op \*node;

if (!fp) printf("There's no Opcode!\n");

node = (Op\*)malloc(sizeof(Op));

for (i = 0; i<20; i++)

ophead[i] = NULL;

while (!feof(fp))

{

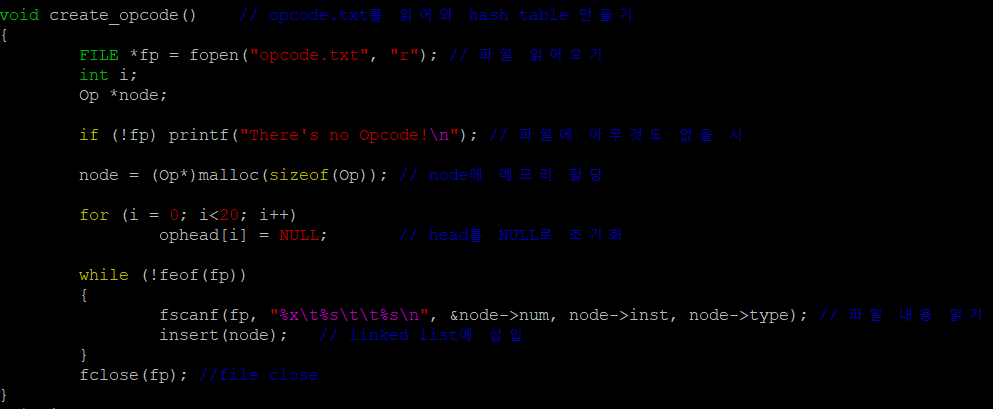
fscanf(fp, "%x\t%s\t\t%s\n", &node->num, node->inst, node->type);

insert(node);

}

fclose(fp);

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : insert(Op \*node)

/\*목적 : 받아온 node의 정보를 linked list 형식으로 저장한다. 만약 head[i]가 NULL이라면 head와 연결하고 NULL이 아니라면 기존에 존재는 노드에 연결하여서 저장한다.

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void insert(Op \*node)

{

int i, key, num, sum = 0;

Op \*new;

num = strlen(node->inst);

for (i = 0; i < num; i++)

sum += node->inst[i];

key = sum % 20;

if (ophead[key] == NULL)

{

ophead[key] = (Op \*)malloc(sizeof(Op));

ophead[key]->num = node->num;

strcpy(ophead[key]->inst, node->inst);

strcpy(ophead[key]->type, node->type);

ophead[key]->list = NULL;

}

else

{

new = (Op \*)malloc(sizeof(Op));

new->num = node->num;

strcpy(new->inst, node->inst);

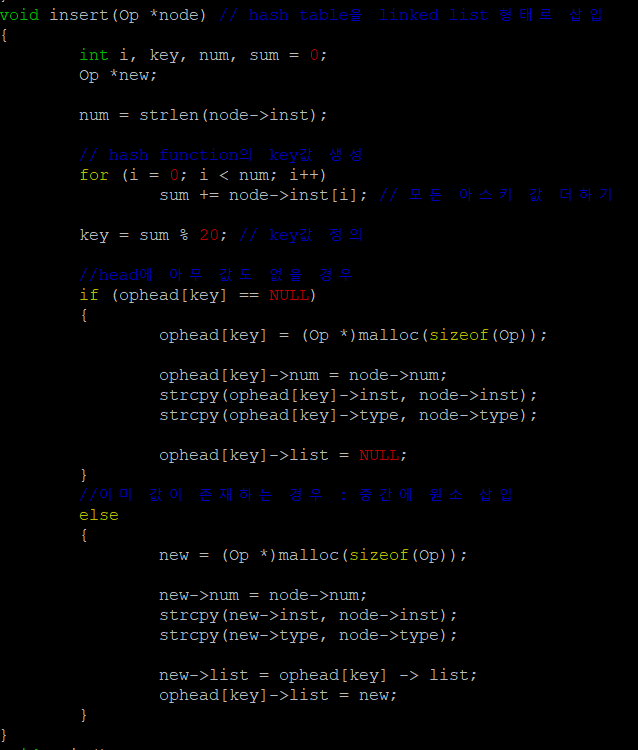
strcpy(new->type, node->type);

new->list = ophead[key] -> list;

ophead[key]->list = new;

}

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : quit()\*/

/\*목적 : 프로그램 종료 전에 동적 할당된 메모리를 free 시킨다.\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void quit()

{

Node \*ptr1;

Op \*ptr2;

int i;

ptr1 = head;

while(head)

{

ptr1 = head;

head = head->list;

free(ptr1);

}

for(i = 0; i < 20; i++)

{

ptr2 = ophead[i];

while(ophead[i])

{

ptr2 = ophead[i];

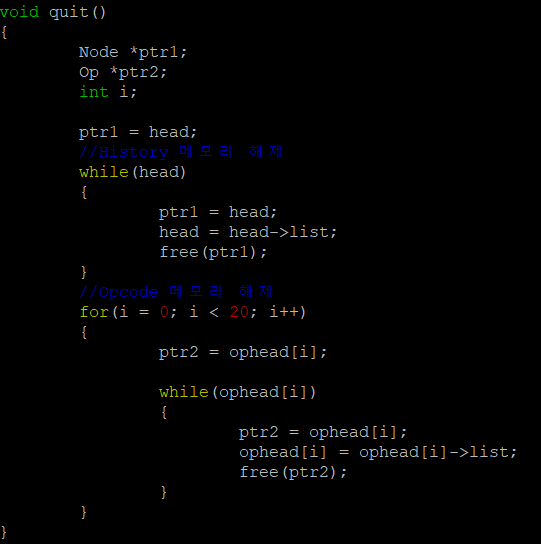
ophead[i] = ophead[i]->list;

free(ptr2);

}

}

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : check1(char \*origin, char \*\*command)\*/

/\*목적 : TOKEN\_SIZE가 1인 경우 명렁어를 받아와 조건문을 통해 판별한 후, shell에서 제공하는 명령어이면 해당 명령어를 수행하는 함수로 이동한다.

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void check1(char \*origin,char \*\*command)

{

unsigned int start, end = 0;

if (strcmp(\*command, "help") == 0 || strcmp(\*command, "h") == 0)

{

record(origin,command, 1);

help();

}

else if (strcmp(\*command, "dir") == 0 || strcmp(\*command, "d") == 0)

{

record(origin,command, 1);

dir();

}

else if (strcmp(\*command, "history") == 0 || strcmp(\*command, "hi") == 0)

{

record(origin,command, 1);

history();

}

else if (strcmp(\*command, "dump") == 0 || strcmp(\*command, "du") == 0)

{

record(origin,command, 1);

if ((address + 160) > 0xFFFFF)

{

start = address;

end = 0xFFFFF;

address = 0;

}

else

{

start = address;

end = address + 159;

address = end + 1;

}

dump(start, end);

}

else if (strcmp(\*command, "reset") == 0)

{

record(origin,command, 1);

reset();

}

else if (strcmp(\*command, "opcodelist") == 0)

{

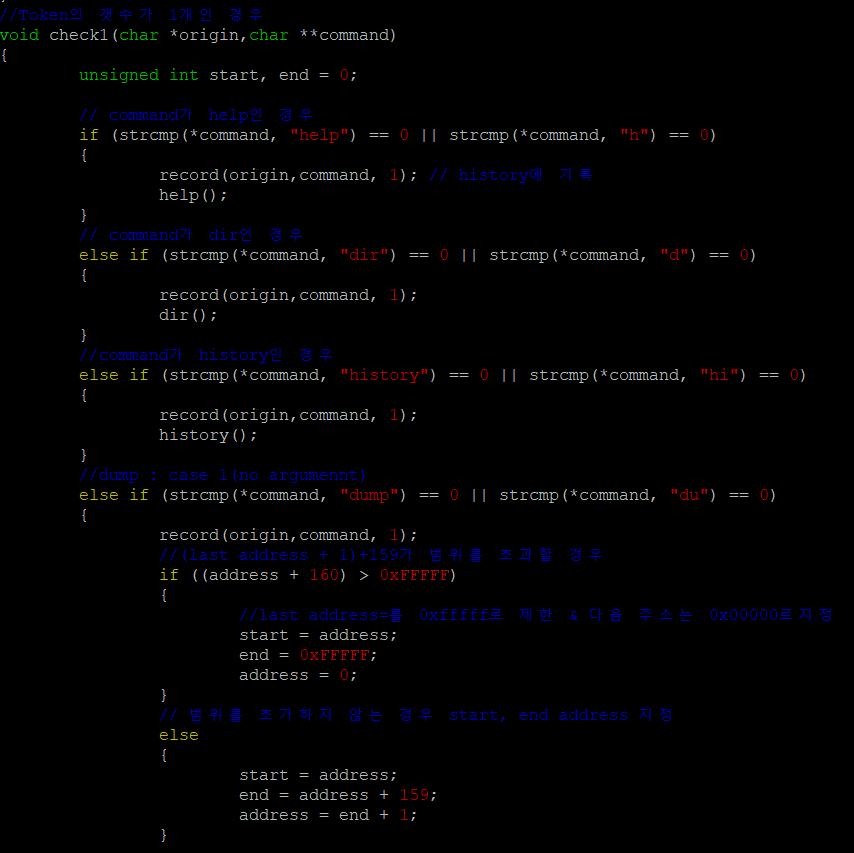
record(origin,command, 1);

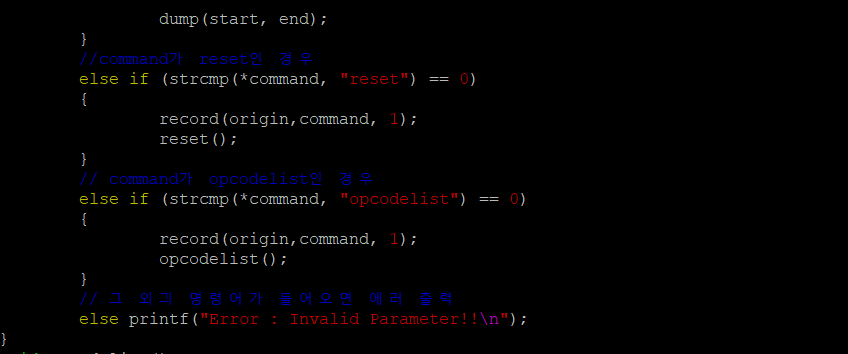
opcodelist();

}

else printf("Error : Invalid Parameter!!\n");

}





/\*------------------------------------------------------------------------------------\*/

/\*함수 : opcodelist()\*/

/\*목적 : linked list 형식으로 저장된 hash table의 내용물을 출력한다.\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void opcodelist()

{

int i, count = 0;

Op \*tmp;

for (i = 0; i < 20; i++)

{

tmp = ophead[i];

printf("%d : ", count);

if (tmp == NULL)

printf(" Empty!");

while (tmp != NULL)

{

printf("[%s, %02X]",tmp->inst, tmp->num);

if (tmp->list != NULL)

printf(" -> ");

tmp = tmp->list;

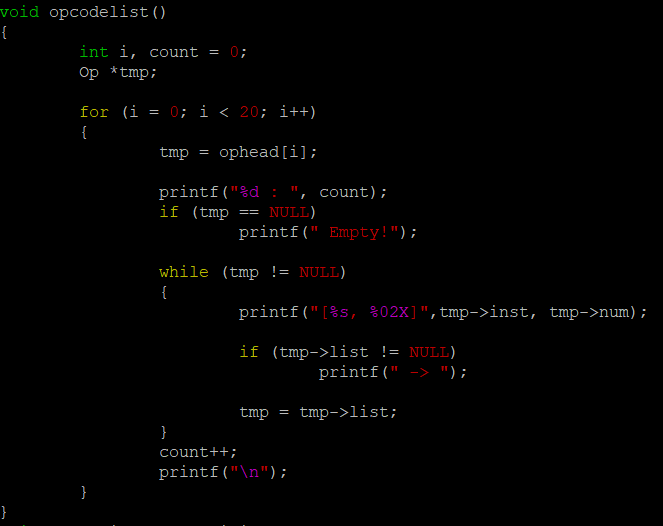
}

count++;

printf("\n");

}

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : mnemonic(char \*origin, char \*\*command)\*/

/\*목적 : 입력받은 명령어가 opcodelist에 존재하면 이에 해당하는 opcode를 출력한다.\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void mnemonic(char \*origin,char \*\*command)

{

int i;

Op \*tmp;

for (i = 0; i < 20; i++)

{

tmp = ophead[i];

while (tmp != NULL)

{

if (strcmp(tmp->inst, command[1]) == 0)

{

printf("opcode is %02X.\n", tmp->num);

record(origin,command, 2);

return;

}

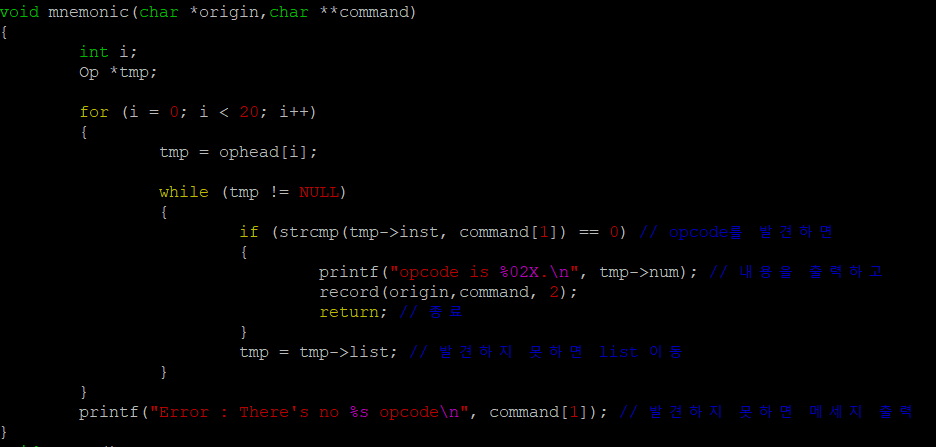
tmp = tmp->list;

}

}

printf("Error : There's no %s opcode\n", command[1]);

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : reset()\*/

/\*목적 : 모든 메모리 주소의 값을 0으로 수정한다.\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void reset()

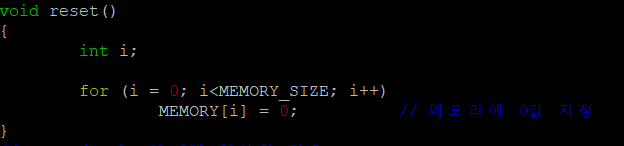
{

int i;

for (i = 0; i<MEMORY\_SIZE; i++)

MEMORY[i] = 0;

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : check2()\*/

/\*목적 : TOKEN\_SIZE가 2개 이상인 경우 명렁어를 받아와 조건문을 통해 판별한 후, shell에서 제공하는 명령어이면 해당 명령어를 수행하는 함수로 이동한다./

/\*리턴값 : 없음/

/\*------------------------------------------------------------------------------------\*/

void check2(char \*origin,char \*\*command, int TOKEN\_SIZE)

{

unsigned int start, end, addr, new = 0,trash = 0;

int i, count = 0;

if ((strcmp(\*command, "dump") == 0 || strcmp(\*command, "du") == 0) && (TOKEN\_SIZE <=4))

{

if ((TOKEN\_SIZE == 2) && (strchr(command[1],',') == NULL))

{

for(i= 0; i < strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') ||(command[1][i] >= 'a' && command[1][i] <= 'f') || (command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ','|| (command[1][0] == '0' && strtol(command[1],NULL,16) >= 0x00000 && strtol(command[1],NULL,16) <= 0xfffff))

continue;

printf("Error : Invalid Parameter!!\n");

return;

}

start = strtol(command[1],NULL,16);

if(start <= 0xfffff && start >= 0x00000)

{

if((start + 160) > 0xfffff)

{

end = 0xfffff;

address = 0;

}

else

{

end = start + 159;

address = end +1;

}

dump(start, end);

record(origin,command, 2);

}

else printf("Error : Invalid Parameter!!\n");

}

else

{

for(i = 1; i < TOKEN\_SIZE - 1; i++)

{

strcat(command[1], " ");

strcat(command[1], command[i + 1]);

}

for(i = 0; i < strlen(command[1]); i++)

if(command[1][i] == ',') count++;

if(count >= 2) printf("Error : Invalid Parameter!!\n");

else

{

i = sscanf(command[1], "%x , %x %x", &start, &end,&trash);

if(i == 2)

{

for(i = 0; i < strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') || (command[1][i] >= 'a' && command[1][i] <= 'f') || (command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ',' || command[1][i] == ' '||(start>=0x00000 &&start<=0xfffff && end<=0xfffff && end >= 0x00000)

)

continue;

printf("Error : Invalid Parameter!!\n");

return;

}

if(start <= end)

{

if((start >= 0 &&start <= 0xfffff) && (end >= 0 && end <= 0xfffff))

{

dump(start, end);

record(origin,command, 2);

if(end == 0xfffff) address = 0;

else address = end + 1;

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

}

}

else if (strcmp(\*command, "edit") == 0 || strcmp(\*command, "e") == 0)

{

if (TOKEN\_SIZE == 2 && strchr(command[1],',') == NULL)

{

printf("Error : Invalid Parameter!!\n");

return;

}

else

{

for(i = 1; i < TOKEN\_SIZE - 1; i++)

{

strcat(command[1], " ");

strcat(command[1], command[i + 1]);

}

for(i = 0; i < strlen(command[1]); i++)

if(command[1][i] == ',') count++;

if(count >= 2) printf("Error : Invalid Parameter!!\n");

else

{

i = sscanf(command[1], "%x , %x %x", &addr, &new,&trash);

if(i == 2)

{

for(i = 0; i < strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') || (command[1][i] >= 'a' && command[1][i] <= 'f') || (command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ',' || command[1][i] == ' ' || (addr >= 0x00000 && addr<=0xfffff))

continue;

printf("Error : Invalid Parameter!!\n");

return;

}

if(addr >= 0x00000 && addr <= 0xFFFFF)

{

if(new >= 0x00 && new <= 0xFF)

{

edit(addr, new);

record(origin,command, TOKEN\_SIZE);

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

}

else if (strcmp(\*command, "fill") == 0 || strcmp(\*command, "f") == 0)

{

for(i = 1; i < TOKEN\_SIZE - 1; i++)

{

strcat(command[1], " ");

strcat(command[1], command[i + 1]);

}

for(i = 0; i < strlen(command[1]); i++)

if(command[1][i] == ',') count++;

if(count != 2) printf("Error : Invalid Parameter!!\n");

else

{

i = sscanf(command[1], "%x , %x , %x %x", &start, &end,&new,&trash);

if(i == 3)

{

for(i = 0; i < strlen(command[1]); i++)

{

if(command[1][i] >= '0' && command[1][i] <= '9') || (command[1][i] >= 'a' && command[1][i] <= 'f') || (command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ',' || command[1][i] == ' ' || (start >= 0x00000 && start <= 0xfffff && end >=0x00000 && end <= 0xfffff))

continue;

printf("Error : Invalid Parameter!!\n");

return;

}

if((start <= end)&&(start >= 0x00000 && start <= 0xFFFFF)&&((end >= 0x00000 && end <= 0xFFFFF)))

{

if(new >= 0x00 && new <= 0xFF)

{

fill(start, end, new);

record(origin,command, TOKEN\_SIZE);

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

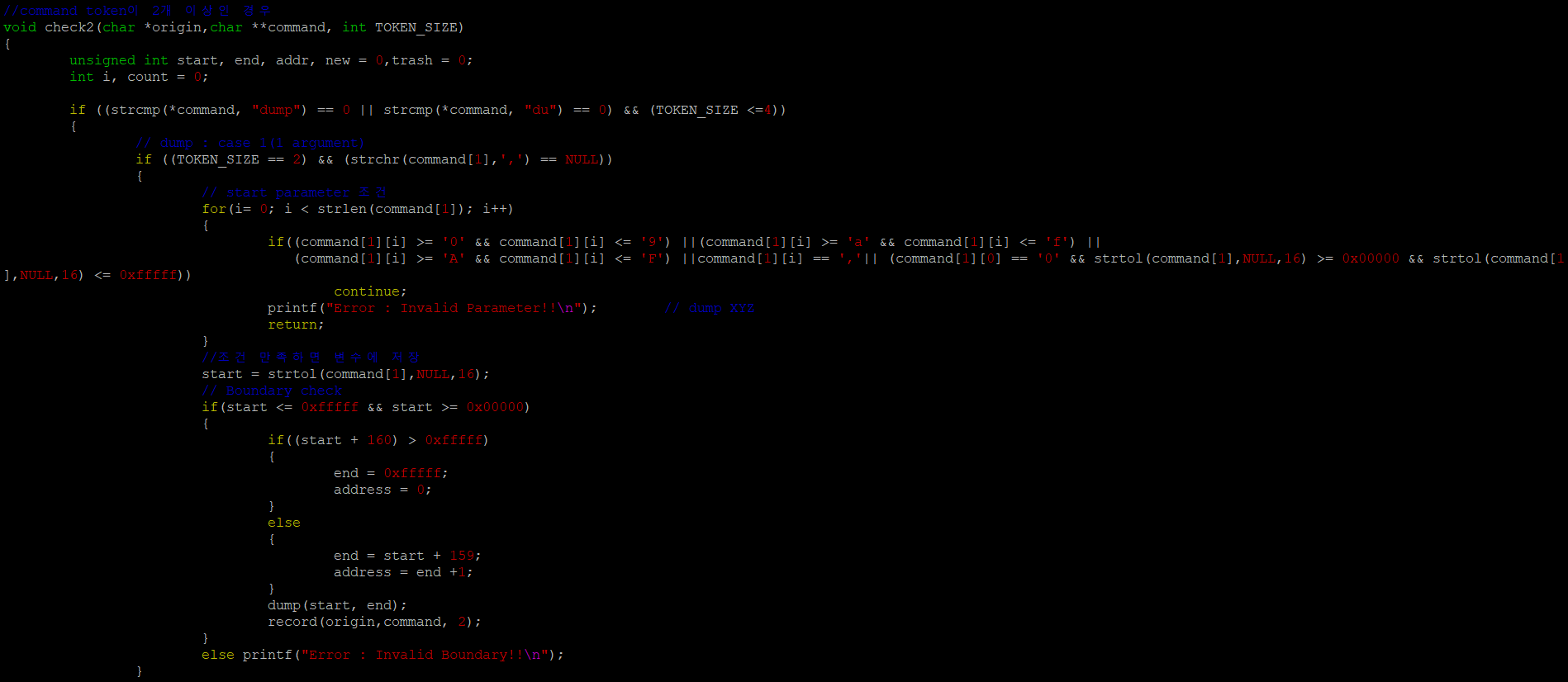
}

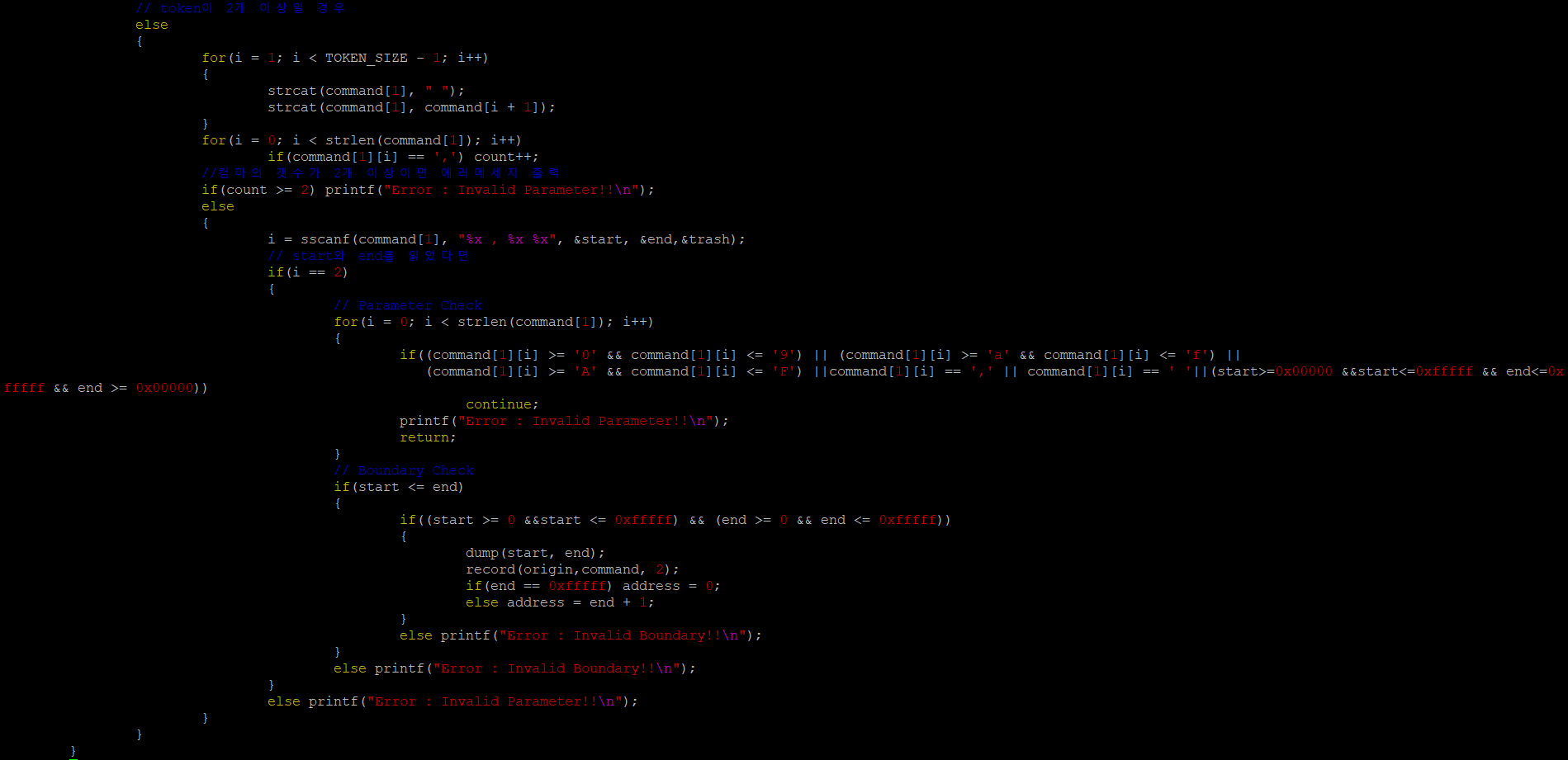
else if (strcmp(\*command, "opcode") == 0)

mnemonic(origin,command);

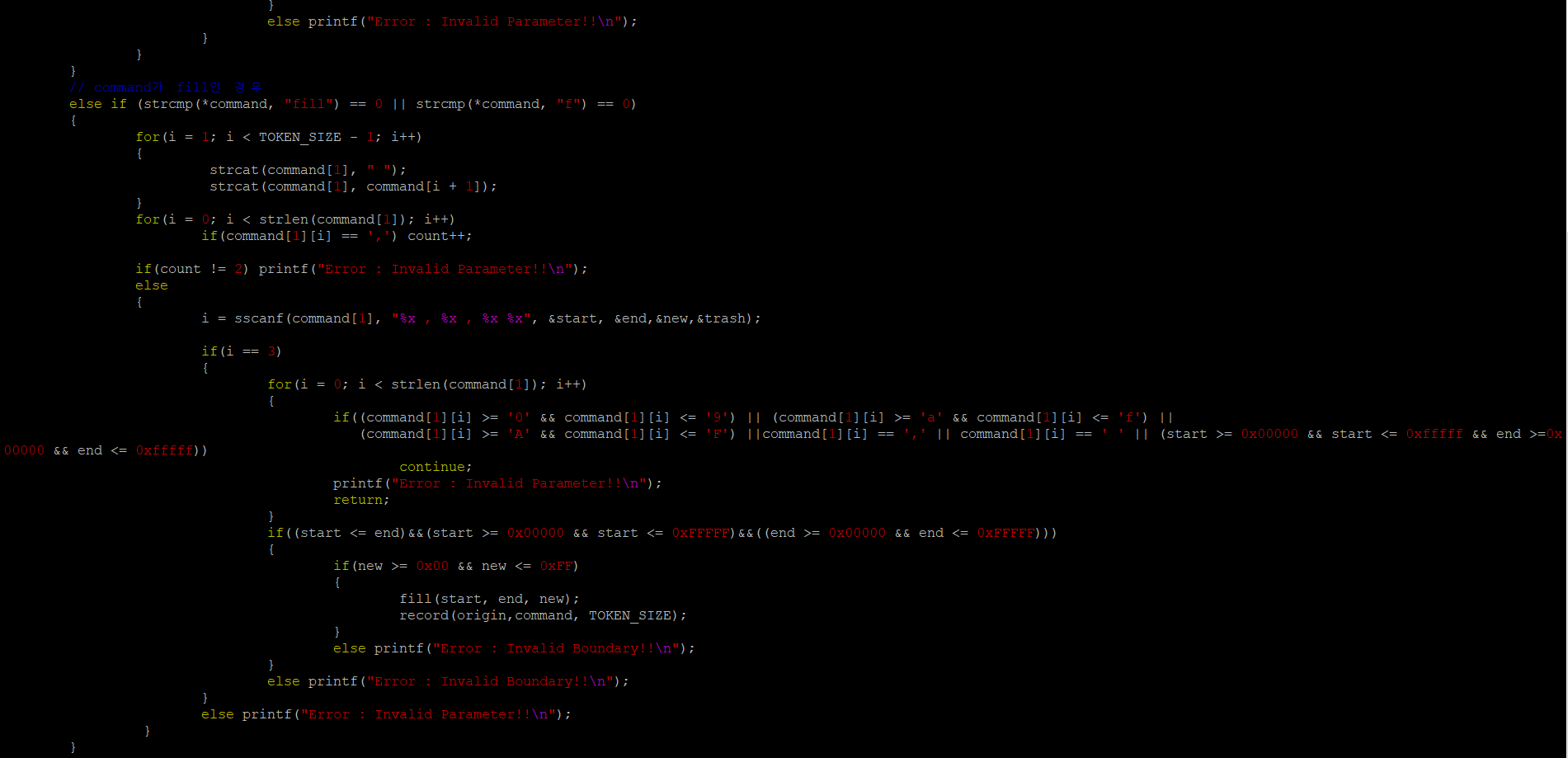
else printf("Error : Invalid Parameter!!\n");

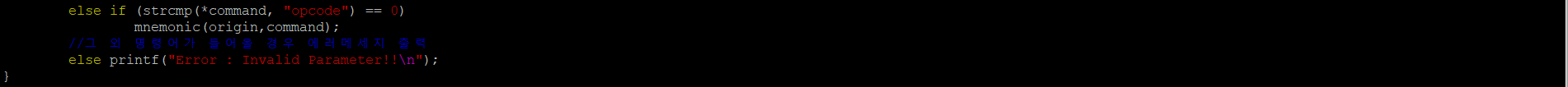
}











/\*------------------------------------------------------------------------------------\*/

/\*함수 : fill(unsigned int start, unsigned int end, unsigned int new)\*/

/\*목적 : start 주소부터 end 주소까지의 값을 new로 수정한다\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

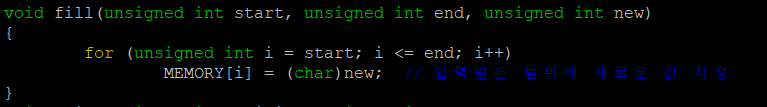
void fill(unsigned int start, unsigned int end, unsigned int new)

{

for (unsigned int i = start; i <= end; i++)

MEMORY[i] = (char)new;

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : edit(unsigned int origin, unsigned int new)\*/

/\*목적 : origin 주소의 값을 new로 수정한다.\*/

/\*리턴값 : 없음\*/

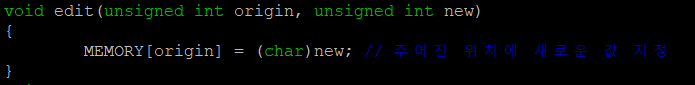
/\*------------------------------------------------------------------------------------\*/

void edit(unsigned int origin, unsigned int new)

{

MEMORY[origin] = (char)new; // 주어진 위치에 새로운 값 지정

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : help()\*/

/\*목적 : shell에서 제공하는 명령어를 보여준다.\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void help()

{

printf("h[elp]\n");

printf("d[ir]\n");

printf("q[uit]\n");

printf("hi[story]\n");

printf("du[mp] [start, end]\n");

printf("e[dit] address, value\n");

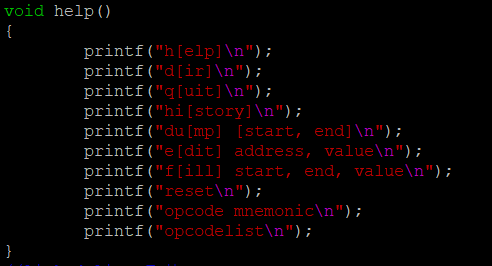
printf("f[ill] start, end, value\n");

printf("reset\n");

printf("opcode mnemonic\n");

printf("opcodelist\n");

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : void record(char \*origin,char \*\*command, int TOKEN\_SIZE)\*/

/\*목적 : 입력한 명령어들을 linked list 형태로 저장한다.\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void record(char \*origin,char \*\*command, int TOKEN\_SIZE)

{

Node \*node;

Node \*tmp = head;

node = (Node \*)malloc(sizeof(Node));

node->list = NULL;

strcpy(node->a,origin);

if(head == NULL)

head = node;

else

{

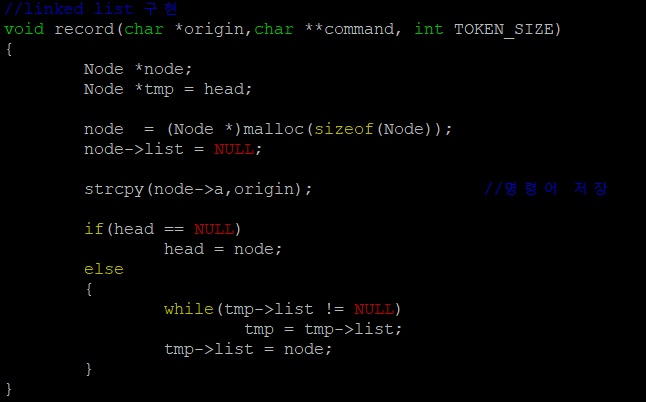
while(tmp->list != NULL)

tmp = tmp->list;

tmp->list = node;

}

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : history()\*/

/\*목적 : linked list 형태로 저장된 명령어들을 출력한다.\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void history()

{

int count = 1;

Node \*tmp = head;

while (1)

{

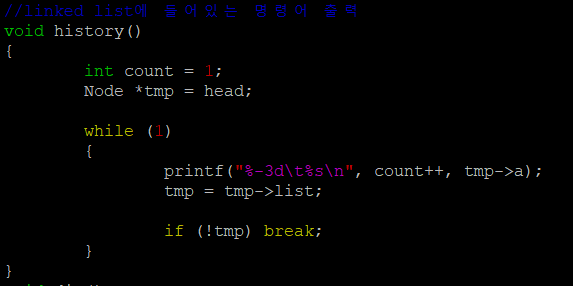
printf("%-3d\t%s\n", count++, tmp->a);

tmp = tmp->list;

if (!tmp) break;

}

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : dir()\*/

/\*목적 : 현재 디렉토리에 존재하는 파일과 폴더의 이름를 출력한다.\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void dir()

{

DIR \*dir\_info = opendir(".");

struct dirent \*dir\_entry;

struct stat buf;

int space = 0;

while ((dir\_entry = readdir(dir\_info)) != NULL)

{

stat(dir\_entry->d\_name, &buf);

if (strcmp(dir\_entry->d\_name, "..") == 0 || strcmp(dir\_entry->d\_name, ".") == 0)

continue;

if (S\_ISDIR(buf.st\_mode)) // 디렉토리 출력

{

printf("%s/\t", dir\_entry->d\_name);

space++;

}

else if (S\_ISREG(buf.st\_mode)) // 파일 출력

{

printf("%s\*\t", dir\_entry->d\_name);

space++;

}

if (space == 3)

{

printf("\n");

space = 0;

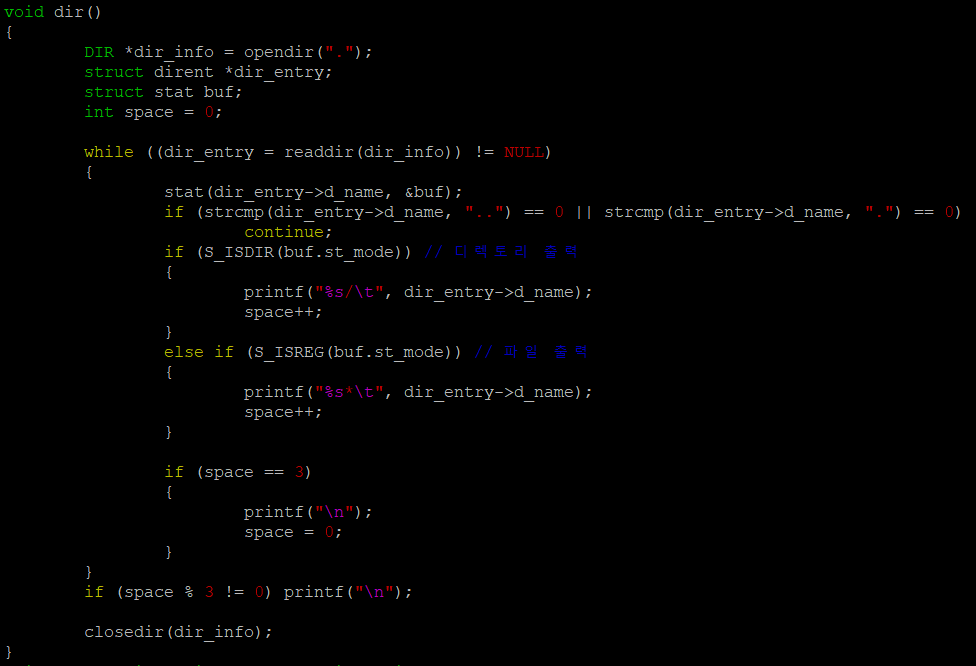
}

}

if (space % 3 != 0) printf("\n");

closedir(dir\_info);

}



/\*------------------------------------------------------------------------------------\*/

/\*함수 : dump(unsigned int start, unsigned int end)\*/

/\*목적 : 입력받은 start 번지와 end 번지를 이용, 프로그램에서 관리하는 MEMORY를 start 번지부터 end 번지까지 출력하는 함수. 출력 라인 왼쪽에는 16진수 번지수가 나타나며 라인당 16개씩 16진수값으로 출력한다. 또한 한 라인 출력이 끝나면 오른편에 MEMORY 값에 해당하는 ASCII 코드를 출력한다.\*/

/\*리턴값 : 없음\*/

/\*------------------------------------------------------------------------------------\*/

void dump(unsigned int start, unsigned int end)

{

int i, j;

unsigned int start\_h = start / 16;

unsigned int end\_h = end / 16;

for (i = 0; i <= (end\_h - start\_h); i++)

{

printf("%05X ", (start\_h + i) \* 16);

for (j = 0; j<16; j++)

{

if ((start\_h + i) \* 16 + j < start || (start\_h + i) \* 16 + j > end)

printf(" ");

else if ((start\_h + i) \* 16 + j >= start && (start\_h + i) \* 16 + j <= end)

printf("%02X ", MEMORY[(start\_h + i) \* 16 + j]);

}

printf(";");

for (j = 0; j<16; j++)

{

if ((start\_h + i) \* 16 + j < start || (start\_h + i) \* 16 + j > end)

printf(".");

else if ((start\_h + i) \* 16 + j >= start && (start\_h + i) \* 16 + j <= end)

{

if (MEMORY[(start\_h + i) \* 16 + j] >= 20 && MEMORY[(start\_h + i) \* 16 + j] <= 127)

printf("%c", MEMORY[(start\_h + i) \* 16 + j]);

else

printf(".");

}

}

printf("\n");

}

}

