**과목명: 시스템프로그래밍**

**2 분반**

**<<Project #2>>**

**서강대학교 [학부명] 컴퓨터공학과**

**[학번] 20171640**

**[이름] 박수진**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
   2. 알고리즘
3. **모듈 정의**

3.1 모듈 이름 : getfilename()

3.1.1 기능

3.1.2 사용 변수

3.2 모듈 이름: type(char\*)

3.2.1 기능

3.2.2 사용 변수

3.3 모듈이름: assemble(char\*)

3.3.1 기능

3.3.2 사용변수

3.4 모듈이름: pass\_1(char\*)

3.4.1 기능

3.4.2 사용변수

3.5 모듈이름: pass\_2(char\*)

3.5.1 기능

3.5.2 사용변수

3.6 모듈 이름 : ZtoAhash(char\*)

3.6.1 기능

3.6.2 사용 변수

3.7 모듈 이름: symbolcmd()

3.7.1 기능

3.7.2 사용 변수

3.8 모듈이름: add\_objlist(char\*, int, int\*)

3.8.1 기능

3.8.2 사용변수

1. **구조체 정의**

4.1 struct symbol

4.1.1 기능

4.1.2 사용 변수

4.2 struct objlist

4.2.1 기능

4.2.2 사용 변수

1. **전역 변수 정의**

5.1 symbol \*symhead[20]

5.2 objlist \*objhead

5.3 int adr\_start

5.4 int pro\_length

5.4 int symflag

1. **코드 설명**

6.1 20171640.h

6.2 20171640.c

# 프로그램 개요

1주차 assignment에서 만든 프로그램을 기본으로 하여, 어셈블러를 구현한다. 어셈블러는 \*.asm 파일의 형식이 들어올 때 (‘assemble’ 명령어) pass1과 pass2의 과정을 거쳐 어셈블한 후 철저하게 오류를 검사하여 오류없이 성공적으로 끝나면 \*.lst와 \*.obj 파일을 생성 및 저장하도록 한다. 만약 오류가 발생하면 디버깅을 위해 줄번호와 오류 내용을 출력하고 어셈블을 종료하도록 한다. 또한 assemble 과정에서 입력 받은 label들은 symbol hash table에 저장하여 ‘symbol’이라는 명령어를 통해 확인할 수 있도록 한다.‘type’ 명령어를 구현하여 파일을 불러오고 출력할 수 있도록 한다.

# 프로그램 설명

## 프로그램 흐름도

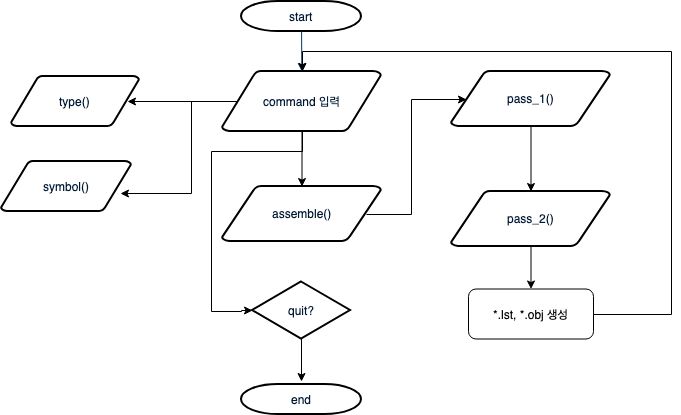


그림 1> 전체 프로그램 흐름도

## 알고리즘 - assemble assemble 명령어가 입력되면 인자로 들어온 파일이 유효한 파일인지 확인한다. 유효하지 않은 파일일 경우에는 에러메세지를 출력하고 함수를 종료한다. 파일이 유효하면 pass\_1()을 호출하여 어셈블링을 시작한다. 만약 pass1 또는 pass2에서 에러가 발생하면 \*.lst 와 \*.obj 파일을 생성하지 않고 종료한다. - pass 1 Untitled Diagram pass 1의 주요 목적은 location counter를 표기하는 것이다. Pass 1은 다음 pseudo code를 참고하여 작성하였다. 1. 우선 파일에서 첫 줄을 불러온 후 START가 있으면 location counter와 start address에 값을 저장하고 START가 없으면 파일의 첫 줄을 다시 읽어야 하므로 파일을 닫았다가 다시 열어준다. 2. 파일의 끝에 도달할 때까지 (입력이 없을 때까지) 반복문으로 파일을 한 줄 씩 읽어온다. 3. 읽어온 줄이 주석이면 줄 번호와 함께 그대로 intermediate 파일에 출력한다. 4. 주석이 아니면 읽어온 라인을 인자 네 개로 나누어 저장한다. s1에는 label, s2에는 directive, s3에는 operand, s4에는 콤마 뒤에 오는 오퍼랜드를 저장하도록 한다. 만약 입력형식이 올바르지 않으면 에러를 출력 후 함수를 종료한다. 5. Label(s1)이 존재하면 symbol table을 통해 label을 찾아보고 해당 라벨이 있으면 중복된 symbol 이므로 에러처리 해준다. Symbol table에 없으면 table에 추가해준다. 여기서 symbol table은 알파벳 내림차순으로 저장되도록 설계하였다. 6. Directive(s2)를 opcode list가 저장된 hash table에서 찾아보고 hash table에 있으면 intermediate파일에 줄번호와 location counter, 읽어 들인 내용을 출력한 후 저장된 형식만큼 location counter를 증가하도록 한다. 이제부터 intermediate 파일에 출력하는 형식은 (줄번호) \t (location counter) \t (읽어 들인 line)\n 으로 통일 하도록 한다. 7. Hash table에 없는 directive일 경우 아래와 같은 경우를 고려한다. 또한 지정된 형식과 맞지 않은 입력은 모두 에러 처리 하도록 한다. 7-1. “BASE” 이면 intermediate파일에 그대로 출력 7-2. “END” 이면 intermediate파일에 출력 후 반복문 종료 7-3. “WORD” 이면 intermediate파일에 출력 후 location counter 3 증가시켜준다. 올바르지 않은 operand 이면 에러 7-4. “BYTE” 이면 intermediate파일에 출력 후 operand(s3)가 C 또는 X 로 시작하는지 확인하고 C로 시작하면 location counter를 (s3의 길이 – 3) 만큼 증가 시킨다(C’XX’의 형태이므로). X로 시작하면 (s3의 길이 – 3) / 2 만큼 location counter를 증가시킨다. C도 X도 아니면 에러를 출력한다. 7-5. “RESW” 이면 intermediate파일에 출력 후 operand(s3) 의 3배만큼 location counter 증가한다. 7-6. “RESB”이면 intermediate파일에 출력 후 operand(s3) 만큼 location counter를 증가한다. 7-7. 위 경우 중 어떠한 경우도 해당되지 않으면 잘못된 directive 이므로 에러 처리한다. 8. 위와 같은 방식으로 계속 파일을 읽다가 반복문이 끝나면 끝날 때END가 있었는지 확인하고 END가 없으면 에러처리 한다. 9. 여기까지 성공적으로 에러없이 진행되면 pass1이 끝났으므로 모든 파일 포인터를 닫는다. 10. 프로그램 길이에 location counter – start address 값을 할당한 후 pass\_2()를 호출한다. - pass 2 Untitled Diagram pass 2가 성공적으로 끝나면 assemble이 완료되어 \*.obj 파일과 \*.lst 파일이 생성되어야 한다. Pass2는 다음과 같은 pseudo code를 참고하여 작성하였다. 1. 시작하기 전에 효율적으로 레지스터를 인식하기 위해서 레지스터 번호와 레지스터 이름을 array에 저장한다. 저장 형태는 다음을 따른다. (실제 C코드에서는 이렇게 표현할 수 없지만 가독성을 위해 document에는 아래와 같이 표기하였다.) register[register\_number] = “register\_name” 2. \*.asm파일의 형태에 맞추어 \*.lst, \*.obj 파일을 쓰기 전용으로 open하고 intermediate 파일을 읽기용으로 open한다. 3. Pass1과 같이 먼저 첫 줄을 받아와서 START가 있으면 lst와 obj 파일에 START 라인을 작성하고 START가 없으면 obj파일만 H부분을 작성하고(프로그램 이름 부분은 공백 처리) 파일을 다시 닫았다가 열어준다. 4. 이제 END 명령어에 도달할 때까지 반복문을 돌면서 계속 intermediate 파일에서 한 줄 씩 읽어와 pass1 처럼 s1, s2, s3, s4에 인자를 나누어 저장하도록 한다. 반복문에서는 \*.lst 파일에만 출력하도록 한다. 반복문이 돌때마다 n=i=1, x=b=p=e=0으로 초기화해준다. 5. 입력된 줄이 주석이면 \*.lst 파일에 그대로 출력한다. 6. 주석이 아니면 다음과 같은 mechanism으로 동작한다. 7. s2(directive)가 ‘END’이면 \*.lst파일에 형식 맞춰 출력한 후 반복문을 종료한다. 8. s2(directive)가 ‘BASE’ 이면 BASE 뒤의 operand(s3)가 symbol table에 있는지 확인하고 없으면 에러를 출력하고 있으면 해당 symbol의 location을 base register에 저장한 후 \*.lst 파일에 형식 맞춰 출력한다. 9. Operand(s3)을 확인한다. 9-1. Operand에 #이 붙어있으면 immediate addressing 이므로 n = 0, i = 1, b =0을 할당하고 # 뒤에 상수가 있으면 p = 0 처리한다. 9-2. Operand에 @가 붙어있으면 indirect addressing이므로 n = 1, i = 0을 할당한다. 9-3. 위 두가지 경우가 아니면 보통 simple addressing 이므로 n =1, i =1을 저장한다. 10. Directive(s2)가 opcode list에 있는 mnemonic인지 확인하고 opcode list에 존재하는 mnemonic일 경우 Mnemonic의 형식을 확인 후 pc address에 location address + format 값을 저장한다. 만약 4형식이면(‘+’) b = 0, p = 0, e = 1을 저장한다. 10-1. 1형식인경우 operand(s3)가 있으면 에러 처리하고 object code에 opcode 값을 저장한 후 \*.lst 파일에 형식 맞춰 저장한다. Object code를 linked list에 저장한다. 10-2. 2형식인 경우 operand(s3)가 없으면 에러 처리한다. 입력 받은 하나 또는 두개의 레지스터가 유효한 레지스터인지 확인한 후 유효한 레지스터이면 object code에 첫 두자리는 opcode, 뒤 두자리는 레지스터 번호를 저장한다. \*.lst 파일에 형식 맞춰 출력하고 object code를 linked list에 저장한다. 10-3. 3형식인 경우 10-3-1. object code에 opcode값이 그대로 들어가지 않으므로 opcode를 2진수로 나타냈을 때 제일 뒤 두자리를 각각 n값과 i값으로 바꿔준다. 10-3-2. Immediate addressing일 때 #뒤에 상수가 오면 disp 영역에 상수를 그대로 올려준다 10-3-3. Operand(s3)가 symbol table에 있는 symbol인지 확인한다. Symbol table에 없으면 에러이고 symbol table에 존재하면 disp <= (symbol의 loc) – (pc address)를 수행한다. 10-3-4. Disp가 2047보다 크거나 -2048보다 작으면 pc relative를 사용할 수 없으므로 b = 1 p = 0 으로 바꿔준 후 base relative를 사용한다(disp <= (symbmol의 loc) – (base address) 10-3-5. 만약 콤마 뒤에 X가 오면 x = 1이다. 10-3-6. Object code의 앞 두자리는 10-3-1에서 새로 설정한 opcode, 중간의 한자리는 xbpe의 값, 뒤 3자리는 disp값을 할당한다. 10-3-7. Object code를 linked list에 저장하고 \*.lst 파일에 형식 맞춰 출력한다. 10-4. 4형식인 경우 10-4-1. 3형식과 같이 opcode를 수정해준다. 10-4-2. Immediate addressing 일 때 뒤에 상수가 오면 addr 필드에 그대로 올려준다. 10-4-3. 상수가 아니면 s3가 symbol table에 존재하는지 확인하고 symbol table에 있으면 s3의 loc값을 addr 필드에 올려준다. 10-4-4. 만약 콤마 뒤에 X가 오면 x = 1이다. 10-4-5. Immediate addressing인 경우를 제외하고 4형식은 modification flag를 1로 설정해준다. 10-4-6. Object code의 앞 두자리는 10-4-1. 에서 새로 설정한 opcode, 중간의 한자리는 xbpe의 값, 뒤 5자리는 addr 필드의 값을 할당한다. 10-4-7. Object code를 linked list에 저장하고 \*.lst 파일에 형식 맞춰 출력한다. 11. Directive(s2)가 opcode list에 없는 mnemonic이면 다음의 경우를 고려한다. 11-1. “BYTE” 인 경우 C형식으로 들어오면 object code에 각각 char들의 ascii 코드를 이어 붙여서 저장한다. X인 경우 object code에 그대로 저장한다. Object code를 linked list에 저장하고 \*.lst 파일에 출력한다. 11-2. “WORD” 인 경우 입력 받은 10진수를 16진수 형태로 object code에 저장한 후 linked list에 저장하고 \*.lst에 출력한다. 11-3. “RESB” 또는 “RESW” 인 경우 object code가 없으므로 \*.lst에 그대로 출력한다. 11-4. Pass1에서 예외는 에러 처리를 해주었기 때문에 언급한 입력 이외의 다른 입력은 들어오지 않는다. 12. 위와 같은 방식으로 END까지 반복문을 수행한다. 13. 반복문을 나와서 Linked list에 저장된 object code를 차례로 출력하며 \*.obj 파일에 형식 맞추어 출력한다. 14. \*.lst \*obj 파일을 닫고 pass2를 종료한다. (assemble 완료)

# 모듈 정의

## 모듈 이름 : getfilename(char\* str)

### 기능 입력 받은 스트링의 가장 뒤 공백을 지운다

### 사용 변수 - int i, j : 인덱스를 표현할 임시 변수

* int flag : 제거할 공백이 있는지 나타내는 flag
* char \*str (매개변수) : 공백을 제거할 스트링

## 모듈 이름: type(char \* f\_name)

### 기능

입력 받은 파일을 열어 파일 내용을 그대로 출력한다

### 사용 변수 - FILE \*fp : 파일 포인터

* char tmp : 캐릭터 값을 저장할 임시 변수
* char \* f\_name (파라미터) : 파일 이름\_

## 모듈이름: assemble(char\* f\_name)

### 기능 파일 이름을 입력 받아 asm 파일이면 어셈블하여 obj파일과 lst 파일을 생성한다.

### 사용변수

* int i : 인덱스에 사용되는 임시 변수
* FILE \*fp : 파일 포인터
* char name\_tmp[100] : 파일 이름에서 확장자 빼고 저장
* char filetmp[100] : 파일이름 저장할 임시 변수
* symbol \*ptr, \*rmv : symbol table free할 때 사용되는 포인터
* Objlist \*obptr, \*obrmv : object code list free 할 때 사용되는 포인터
* char\* f\_name (파라미터) : 파일이름

## 모듈이름: pass\_1(char\* f\_name)

### 기능 파일 이름을 입력 받아 열고 asm 파일에 오류가 있는지 검토하면서 location counter를 계산하여 intermediate 파일에 출력한 후 pass\_2()를 호출한다.

### 사용변수

* int locctr : location counter
* FILE \*src : 소스 파일(.asm)
* FILE \*intmd : intermediate 파일
* char line[300] : 한 줄 읽어올 때 사용할 공간
* char s1[100], s2[100], s3[100], s4[100] : 읽어온 줄에서 인자별로 저장된 값
* char \*tmpstr : 스트링 저장할 임시 변수
* int lnum : 줄 번호
* int cmflag : 콤마의 위치 인덱스
* int I, k : 인덱스나 반복문에 사용되는 임시 변수
* int tmp : 임시 변수
* int dectmp : 10진수 저장하는 임시 변수
* int endflag : END가 들어왔는지 확인하는 flag
* symbol \*symptr, \*pre, \*cur : symbol table을 이동, 추가할 때 사용하는 포인터
* OPlist \*opptr : opcode list를 이동할 때 사용하는 포인터
* int idx : hash function에서 도출된 인덱스

## 모듈이름: pass\_2(char\* f\_name)

### 기능 intermediate 파일에서 asm 정보와 location counter 등을 불러와 object code를 계산하여 obj 파일과 lst 파일을 생성 및 값을 저장한다.

### 사용변수

* FILE \*intmd, \*obj, \*lst : 각각 intermediate, \*.obj, \*.lst 파일을 가리키는 포인터
* char of\_name[100], lf\_name[100] : 각각 \*.obj, \*lst 파일의 이름이 저장됨
* char line[500] : 파일에서 한 줄 씩 읽어올 때 사용
* char s1, s2, s3, s4 : 받아온 line에서 인자를 나누어 각각 저장
* int k, l : 임시 변수
* int n, i, x, b, p, e : SIC/XE 머신의 3, 4형식에서 사용되는 레지스터
* int pc\_adr : pc relative addressing할 때 사용할 pc address를 저장
* int b\_adr : base relative addressing 할 때 사용할 base address를 저장
* int tmp : 임시 변수, 라인에 있는 인자 개수
* int lnum, locctr : 각각 줄 번호, location counter 값
* int cmflag : 콤마의 인덱스 번호
* int format : 해당 directive의 형식
* int idx : hash function 이용한 인덱스 값
* OPlist \*opptr : opcode list 탐색할 때 사용할 포인터
* symbol \*symptr : symbol table 탐색할 때 사용할 포인터
* char reg[10][2] : SIC/XE의 레지스터 번호와 레지스터 이름 저장
* char objcode[10] : 해당하는 라인의 object code
* int newop : 보정해준 opcode
* int disp : displacement 값
* int dectmp : 10진수 전용 임시 변수
* int xbpe : xbpe 값을 합쳐서 저장
* char strtmp[100], tmpstr[100] : 스트링 전용 임시변수
* char bytetmp[100] : “BYTE” 일 때 사용되는 임시 변수

## 모듈이름: ZtoAhash(char\* label)

### 기능 symbol table을 내림차순으로 정리하기 위한 hash function

### 사용변수 - char\* label (파라미터) : 라벨이 저장된 스트링

## 모듈이름: symbolcmd()

### 기능 symbol table이 존재하는지 확인하고 symbol table이 존재하면 그 내용을 순서대로 출력한다. 그렇지 않으면 에러를 출력한다.

### 사용변수

* symbol \*ptr : symbol table을 탐색하는 포인터
* int i : symbol table의 인덱스, 반복문에 사용

## 모듈이름: add\_objlist(char\* objcode, int loc, int m)

### 기능 object code와 location, modify flag를 받아와서 새 노드에 저장한 후 object code가 저장된 linked list에 연결하여 저장한다.

### 사용변수

* objlist \*newNode : object 코드와 그 외의 값들이 저장된 새 노드
* objlist \*ptr : linked list 탐색할 때 사용할 포인터
* char \*objcode (파라미터 ) : object code
* in loc (파라미터) : location
* in m (파라미터) : modification 여부가 저장된 변수

# 구조체 정의

### **4.1 struct symbol**

### 기능 symbol table linked list 저장에 사용되는 구조체

### 사용변수

- char name[100] : label 이름

- int locctr : location counter 값

- struct symbol \*link : 다음 노드를 가리키는 포인터

### **4.2 struct objlist**

### 4.2.1 기능 object code가 저장된 linked list에 사용되는 구조체

### 4.2.2 사용변수

- char objcode[100] : object code

- int loc : location

- int flag : 줄바꿈이 필요한가에 대한 flag

- int mflag : modification이 필요한가에 대한 flag

- struct objlist \*link : 다음 노드를 가리키는 포인터

# 전역 변수 정의

## symbol \*symhead[26] symbol이 저장된 hash table의 head들의 집함

## objlist \*objhead object code가 저장된 linked list의 head 포인터

## int adr\_start 프로그램 시작 주소 (start location)

## int pro\_length 프로그램의 길이

## int symflag symbol table이 있는지 확인하는 flag

# 코드 설명

## 20171640.h

/\* 포함되는 파일 및 라이브러리\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <dirent.h>

/\* 정의되는 구조체\*/

typedef struct hiList { // history linked list 저장할 구조체

char cmd\_name[100]; // 커맨드 저장

struct hiList \*link; // 다음 노드를 가리키는 link

}hiList;

typedef struct OPlist{ // opcode linked list 저장할 구조체

int opcode; // opcode 저장

char inst\_name[10]; // mnemonic 저장

char form[5]; // format 저장

struct OPlist \*link; // 다음 노드를 가리키는 link

} OPlist;

typedef struct symbol{ // label 저장할 linkedlist용 구조체

char name[100]; // label명

int locctr; // location counter

struct symbol \*link; // 다음 노드 link

}symbol;

typedef struct objlist{ // object code를 저장한 linkedlist 용 구조체

char objcode[100]; // objcode

int loc; // location

int flag; // 줄바꿈 필요한지 flag

int mflag; // modification 필요한 경우 flag

struct objlist \*link; // 다음 노드 가리키는 link

}objlist;

/\* global 변수 \*/

hiList \*hiHead = NULL; // hiList head

OPlist \*OPhead[20]; // OPlist head 배열

symbol \*symhead[26]; // symlist head 배열

objlist \*objhead = NULL; // objlist head

unsigned char memory[1048576]; // 16 X 65536 가상메모리

int dumpend = -1; // dump에서 마지막으로 저장된 end 값

int adr\_start = 0; // 프로그램 시작주소

int pro\_length = 0; // 프로그램 길이

int symflag = 0; // symbol table flag

/\* 함수 원형 \*/

/\*---------------------proj 1----------------------------\*/

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

/\* 함수 : editcmd \*/

/\* 목적 : 입력받은 커맨드를 명령어와 \*/

/\* 나머지로 나눈다 \*/

/\* 리턴값 : 없음 \*/

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

void editcmd(char\*, char\*, char\*);

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

/\* 함수 : help \*/

/\* 목적 : 프로그램 상에서 사용할 수 있는 \*/

/\* 명령어들을 출력한다 \*/

/\* 리턴값 : 없음 \*/

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

void help();

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

/\* 함수 : dir \*/

/\* 목적 : 현재 디렉토리 내 파일들을 \*/

/\* 출력한다 \*/

/\* 리턴값 : 없음 \*/

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

void dir();

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

/\* 함수 : history \*/

/\* 목적 : history linked list에 저장된 \*/

/\* 정보를 차례로 출력 \*/

/\* 리턴값 : 없음 \*/

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

void history();

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

/\* 함수 : add\_list \*/

/\* 목적 : 입력받은 instruction을 history \*/

/\* linked list에 저장 \*/

/\* 리턴값 : 없음 \*/

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

void add\_list(char\*);

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

/\* 함수 : dump \*/

/\* 목적 : 가상 메모리에 저장된 정보를 특정 \*/

/\* 범위 만큼 출력 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

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

int dump(char\*);

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

/\* 함수 : edit \*/

/\* 목적 : 가상 메모리의 특정 주소에 저장된 \*/

/\* 값을 치환한다 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

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

int edit(char\*);

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

/\* 함수 : fill \*/

/\* 목적 : 가상 메모리에서 원하는 범위만큼의\*/

/\* 값을 특정 값으로 치환한다 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 \*/

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

int fill(char\*);

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

/\* 함수 : reset \*/

/\* 목적 : 가상 메모리의 모든 값을 0으로 \*/

/\* 초기화 \*/

/\* 리턴값 : 없음 \*/

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

void reset();

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

/\* 함수 : makeOPlist \*/

/\* 목적 : opcode 파일로부터 hash table 생성\*/

/\* 리턴값 : 없음 \*/

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

void makeOPlist();

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

/\* 함수 : hashFunction \*/

/\* 목적 : 입력받은 string을 암호화 \*/

/\* 리턴값 : 암호화된 int \*/

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

int hashFunction(char\*);

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

/\* 함수 : opcode \*/

/\* 목적 : 입력받은 mnemonic에 해당하는 \*/

/\* opcode 출력 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

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

int opcode(char\*);

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

/\* 함수 : opcodelist \*/

/\* 목적 : hashtable에 저장된 opcode list를 \*/

/\* 차례로 출력 \*/

/\* 리턴값 : 없음 \*/

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

void opcodelist();

/\*---------------------proj 2----------------------------\*/

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

/\* 함수 : getfilename \*/

/\* 목적 : 문자열의 제일 뒤에 있는 공백들을 \*/

/\* 제거 \*/

/\* 리턴값 : 없음 \*/

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

void getfilename(char\*);

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

/\* 함수 : type \*/

/\* 목적 : 파일 안의 내용을 그대로 출력 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

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

int type(char\*);

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

/\* 함수 : assemble \*/

/\* 목적 : 입력받은 .asm 파일을 어셈블 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

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

int assemble(char\*);

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

/\* 함수 : pass\_1 \*/

/\* 목적 : location counter 계산하여 \*/

/\* intermediate 파일에 저장 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

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

int pass\_1(char\*);

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

/\* 함수 : pass\_2 \*/

/\* 목적 : intermediate 파일에서 어셈블하여 \*/

/\* obj, lst 파일 생성 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

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

int pass\_2(char\*);

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

/\* 함수 : ZtoAhash \*/

/\* 목적 : 알파벳 내림차순으로 저장하기 위한\*/

/\* hash function \*/

/\* 리턴값 : index number \*/

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

int ZtoAhash(char\*);

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

/\* 함수 : symbolcmd \*/

/\* 목적 : symbol table 내용 출력 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

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

int symbolcmd();

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

/\* 함수 : add\_objlist \*/

/\* 목적 : object code를 linked list에 저장 \*/

/\* 리턴값 : 없음 \*/

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

void add\_objlist(char\*, int, int);

## 20171640.c

/\* 포함되는 파일 \*/

#include "20171640.h"

/\* 프로그램 시작\*/

int main() {

// oricmd : 처음 입력받은 커맨드

// cmd : 명령어 부분만 잘라낸 char 배열

// idx : 명령어 뒷부분만 잘라낸 char 배열

char oricmd[100], cmd[100], idx[100];

makeOPlist(); // opcode list로 hash table 생성

while (1) {

printf("sicsim> ");

// local 변수 초기화

memset(oricmd, '\0', sizeof(oricmd));

memset(cmd, '\0', sizeof(cmd));

memset(idx, '\0', sizeof(idx));

// 커맨드 입력받아 저장

fgets(oricmd, sizeof(oricmd), stdin);

// cmd와 idx로 분리

editcmd(oricmd, cmd, idx);

// 명령어가 "h" 또는 "help" 일 때

if (!strcmp(cmd, "help") || !strcmp(cmd, "h")) {

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

help(); // help 실행

}

else // idx가 비어있지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "d" 또는 "dir" 일 때

else if (!strcmp(cmd, "d") || !strcmp(cmd, "dir")) {

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

dir(); // dir 실행

}

else // idx가 비어있지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "q" 또는 "quit" 일 때

else if (!strcmp(cmd, "q") || !strcmp(cmd, "quit")) {

if (idx[0] == '\0') // idx가 비었으면

break; // 루프문 종료

else // 비어있지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "hi" 또는 "history" 일 때

else if (!strcmp(cmd, "hi") || !strcmp(cmd, "history")) {

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

history(); // history 실행

}

else // 그렇지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "du" 또는 "dump" 일 때

else if (!strcmp(cmd, "du") || !strcmp(cmd, "dump")){

if (dump(idx) == 1) // dump() return 값이 1이면

add\_list(oricmd); // history 추가

}

// 명령어가 "e" 또는 "edit" 일 때

else if (!strcmp(cmd, "e") || !strcmp(cmd, "edit")){

if (edit(idx) == 1) // edit() return 값이 1이면

add\_list(oricmd); // history 추가

}

// 명령어가 "f" 또는 "fill" 일 때

else if (!strcmp(cmd, "f") || !strcmp(cmd, "fill")){

if (fill(idx) == 1) // fill() return 값이 1이면

add\_list(oricmd); // history에 추가

}

// 명령어가 "reset" 일 때

else if (!strcmp(cmd, "reset")){

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

reset(); // reset 실행

}

else // 그렇지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "opcode" 일 때

else if (!strcmp(cmd, "opcode")){

if (opcode(idx) == 1) // opcode() return 값이 1이면

add\_list(oricmd); // history에 추가

}

// 명령어가 "opcodelist" 일 때

else if (!strcmp(cmd, "opcodelist")){

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

opcodelist(); // opcodelist 실행

}

else // 그렇지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "type" 일 때

else if (!strcmp(cmd, "type")){

if (type(idx) == 1) // type() return이 1이면

add\_list(oricmd); // history에 추가

}

// 명령어가 "assemble" 일 때

else if (!strcmp(cmd, "assemble")){

if (assemble(idx) ==1) // assemble() return이 1이면

add\_list(oricmd); // history에 추가

}

// 명령어가 "symbol" 일 때

else if (!strcmp(cmd, "symbol")) {

if (idx[0] == '\0'){ // idx가 비었으면

if (symbolcmd() == 1)

add\_list(oricmd);

}

else // 비어있지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 그 이외의 입력 에러 처리

else

printf("Invalid Command\n");

}

// linked list dellocate를 위한 local 변수

hiList \*hifree, \*tmp1;

OPlist \*opfree, \*tmp2;

int i;

// history linked list dellocate

hifree = hiHead;

while (hifree != NULL){

tmp1 = hifree;

hifree = hifree -> link;

free(tmp1);

}

// Opcodelist linked list dellocate

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

opfree = OPhead[i];

while (opfree != NULL){

tmp2 = opfree;

opfree = opfree->link;

free(tmp2);

}

}

// free symbol table

symbol \*symptr, \*symrmv;

for (i = 0; i < 26; i++){

symptr = symhead[i];

while (symptr){

symrmv = symptr;

symptr = symptr -> link;

free(symrmv);

}

symhead[i] = NULL;

}

// free objcode list

objlist \*obptr, \*obrmv;

obptr = objhead;

while (obptr){

obrmv = obptr;

obptr = obptr -> link;

free(obrmv);

}

return 0;

}

/\* 사용되는 function \*/

/\* start of proj 1 \*/

void editcmd(char\* cmd, char\* new, char\* idx) {

int i = 0, j = 0; // 반복분과 인덱스에 사용하는 임시 변수

cmd[(unsigned)strlen(cmd) - 1] = '\0'; // 커맨드의 '\n'을 '\0'으로 치환

while (cmd[i] == ' ' || cmd[i] == '\t')

// i에 처음으로 공백이 아닌 값의 인덱스 저장

i++;

while (cmd[i] != ' ' && cmd[i] != '\0' && cmd[i] != '\t'){

// 공백 또는 NULL 값이 나오기 전까지의 값을 new에 저장(명령어 부분만 저장)

new[j] = cmd[i];

i++;

j++;

}

if (cmd[i] != '\0'){ // 명령어 뒤에 파라미터가 더 있을 때

j = 0;

while (cmd[i] == ' ' || cmd[i] == '\t')

// i에 처음으로 공백이 아닌 값의 인덱스를 저장

i++;

while (cmd[i] != '\0'){ // idx에 NULL이 나오기 전까지 저장

idx[j] = cmd[i];

i++;

j++;

}

}

}

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");

printf("assemble filename\n");

printf("type filename\n");

printf("symbol\n");

}

void dir() {

DIR \*ptrDir;

struct dirent \*ptrFile;

struct stat fileStat;

int num = 0;

ptrDir = opendir("."); // 현재 디렉토리 포인터 불러오기

while ((ptrFile = readdir(ptrDir)) != NULL){ // 디렉토리 내의 파일

stat(ptrFile->d\_name, &fileStat);

printf("\t%s", ptrFile->d\_name); // 파일 읽어서 출력

if (S\_ISDIR(fileStat.st\_mode))

printf("/"); // 파일이 디렉토리이면 '/' 붙이기

else if (S\_IXUSR & fileStat.st\_mode)

printf("\*"); // 파일이 실행파일이면 '\*' 붙이기

num++;

if (num % 4 == 0)

printf("\n"); // 4개 파일 출력되면 줄바꿈

}

if (num % 4 != 0)

printf("\n");

closedir(ptrDir); // 디렉토리 포인터 닫기

}

void history() {

hiList \*ptr; // linked list를 탐색하기 위한 임시변수

int num = 1; // 줄 번호 나타낼 정수

ptr = hiHead;

while (ptr != NULL) {

// ptr이 NULL 하기 전까지 linked list 탐색하며 출력

printf(" %-5d%s\n", num, ptr->cmd\_name);

num++;

ptr = ptr->link;

}

}

void add\_list(char\* cmd) {

hiList \*newNode = (hiList\*)malloc(sizeof(hiList)); // 새로운 값을 저장할 노드

hiList \*ptr = hiHead; // linked list 탐색에 사용되는 임시변수

strcpy(newNode->cmd\_name, cmd);

newNode->link = NULL;

// 새 노드에 입력받은 커맨드 저장, link 값은 아직 없으니 NULL

if (hiHead == NULL) { // head가 NULL 이면

hiHead = (hiList\*)malloc(sizeof(hiList));

hiHead = newNode;

// head 할당한 후 새 노드 저장

}

else { // head가 NULL하지 않으면

while (ptr->link != NULL)

ptr = ptr->link; // 새 노드를 연결할 노드까지 이동

ptr-> link = newNode; // 새 노드 연결

}

}

int dump(char\* idx){

int cmflag = -1; // 콤마의 위치 인덱스

int i = 0, j; // 인덱스와 반복문에 사용할 변수

int start, end; // 메모리 시작 값과 마지막 값을 저장할 변수

char tmp[100];

// 입력값이 올바르지 않음을 판단할 때 사용할 임시 변수

int adflag = 0; // dump 뒤에 인자가 없을 때 1로 set(마지막 address 저장)

while (idx[i] != '\0'){

if (idx[i] == ','){

cmflag = i;

break;

}

i++;

} // cmflag : 콤마의 위치

if (cmflag >= 0){ // 콤마가 있을 경우

if (sscanf(idx, "%x \t , \t %x %s", &start, &end, tmp) != 2){

// 파라미터가 두개 존재하지 않는 경우

// 두 파라미터가 16진수가 아닌 경우 에러

printf("Invalid parameter\n");

return 0;

}

}

else { // 콤마가 없을 경우

if (idx[0] == '\0'){ // dump 뒤로 들어온 파라미터가 없으면

start = dumpend + 1;

end = start + 159;

}

else { // dump 뒤로 들어온 파라미터가 있을 때

if (sscanf(idx, "%x %s", &start, tmp) != 1){

// 파라미터가 하나 존재하지 않는 경우

// 파라미터가 16진수가 아닌 경우 에러

printf("Invalid parameter\n");

return 0;

}

end = start + 159;

}

}

if (end > 0xFFFFF) // end 범위가 최대값을 벗어나면 최대값으로 저장

end = 0XFFFFF;

if (start < 0 || end < 0 || start > 0xFFFFF || end > 0xFFFFF || start > end){

// start나 end가 허용된 범위안에 있지 않으면 에러

printf("Address is out of range\n");

return 0;

}

for (i = start / 16 \* 16; i <= end / 16 \* 16; i+= 16){

printf("%05X ", i); // address 출력 (왼쪽 칼럼)

for (j = i; j < i + 16; j++){

// 메모리 값 16진수로 표현 (가운데 칼럼)

if (j >= start && j <= end)

printf("%02X ", memory[j]);

else

printf(" ");

}

printf("; ");

for (j = i; j < i + 16; j++){

// 메모리 값 ASCII 값으로 표현 (오른쪽 칼럼)

if(j >= start && j <= end && memory[j] >= 0X20 && memory[j] <= 0x7E)

// 20에서 7E 까지의 값은 ASCII 값으로 출력한다

putchar(memory[j]);

else

// 그 이외의 값은 '.' 으로 출력한다

putchar('.');

}

printf("\n");

}

if (adflag == 1) // dump 뒤에 인자가 없으면

dumpend = end; // 마지막으로 저장된 end값 global var에 저장

if (dumpend >= 0xFFFFF) // 저장된 global var이 최댓값을 넘어가면 초기화

dumpend = -1;

return 1;

}

int edit(char\* idx){

int cmflag = -1; // 콤마 인덱스 값

int i = 0; // 인덱스나 반복문에 사용할 변수

int adr, val; // 주소값과 메모리값을 저장할 변수

char tmp[100]; // 임시 변수

while (idx[i] != '\0'){

if (idx[i] == ','){

cmflag = i;

break;

}

i++;

} // cmflag : 콤마의 위치

if (cmflag == -1){ // 콤마가 없으면 에러

printf("Invalid parameter\n");

return 0;

}

else{ // 콤마가 있으면

if (sscanf(idx, "%x \t , \t %x %s", &adr, &val, tmp) != 2){

// 파라미터 개수가 2개가 아니거나

// 들어온 파라미터가 16진수가 아니면 에러

printf("Invalid parameter\n");

return 0;

}

}

if (adr < 0x00000 || adr > 0xFFFFF){

// 주소값 범위가 허용된 범위를 벗어나면 에러

printf("Address is out of range\n");

return 0;

}

if (val > 0xFF || val < 0){

// value 값이 허용된 범위를 벗어나면 에러

printf("Value is out of range\n");

return 0;

}

memory[adr] = val; // 메모리에 입력된 16진수 값 저장

return 1;

}

int fill(char\* idx){

int cm1 = -1, cm2 = -1; // 콤마 인덱스 값

int i = 0; // 변수, 반복문에 쓰이는 변수

int start, end, value; // 파라미터로 들어온 start, end, value 값

char tmp[100]; // 임시 변수

while (idx[i] != '\0'){ // 첫번째 콤마 탐색

if (idx[i] == ','){

cm1 = i;

break;

}

i++;

}

i++;

while (idx[i] != '\0'){ // 두번째 콤마 탐색

if (idx[i] == ','){

cm2 = i;

break;

}

i++;

}

if (cm1 == -1 || cm2 == -1){ // 콤마가 2개가 아니면 에러

printf("Invalid parameter\n");

return 0;

}

if (sscanf(idx, "%x \t , \t %x \t , \t %x %s", &start, &end, &value, tmp) != 3){

// 16진수인 파라미터 값이 3개가 아닐경우 에러

printf("Invalid parameter\n");

return 0;

}

if (start < 0 || end < 0 || start > 0xFFFFF || end > 0xFFFFF || start > end){

// start와 end 값이 범위 내에 없을 경우 에러

printf("Address is out of range\n");

return 0;

}

if (value > 0xFF || value < 0){

// value 값이 허용된 범위를 벗어나면 에러

printf("Value is out of range\n");

return 0;

}

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

memory[i] = value; // 메모리에 입력받은 값 저장

return 1;

}

void reset(){

memset(memory, 0, sizeof(memory)); // 메모리 값 0으로 초기화

}

void makeOPlist(){

FILE \*fp = fopen("opcode.txt", "r"); // 파일 포인터에 읽기전용으로 로드

char inst[10], form[5];

int op, tmp;

// 파일포인터가 비었으면 에러 출력하고 종료

if (fp == NULL){

printf("Opcode list file is NULL\n");

return;

}

while (1){

if (fscanf(fp, "%x %s %s", &op, inst, form) == EOF) // 파일 정보 저장

break; // 더이상 읽을 줄이 없으면 탐색 중단

// 읽어들인 정보를 새로운 노드에 저장

OPlist \*newNode = (OPlist\*)malloc(sizeof(OPlist));

newNode->opcode = op;

strcpy(newNode->inst\_name, inst);

strcpy(newNode->form, form);

newNode->link = NULL;

// 명령어를 hash function을 이용하여 암호화된 값으로 변환한다.

tmp = hashFunction(inst);

// 새 노드를 linked list에 연결

if (OPhead[tmp] == NULL){ // 해당 인덱스의 헤드가 NULL이면

OPhead[tmp] = newNode; // 제일 첫 노드에 저장

}

else { // 그렇지 않으면

OPlist \*ptr = OPhead[tmp];

while (ptr->link != NULL)

ptr = ptr->link;

ptr->link = newNode; // 가장 뒤에 있는 노드 뒤에 연결

}

}

fclose(fp); // 파일 포인터 닫기

}

int hashFunction(char\* inst){

int i;

int len = strlen(inst);

// 명령어를 읽어서 정수형으로 암호화 한다

i = inst[0] + inst[len - 1] + len;

return i % 20;

}

int opcode(char\* idx){

char tmp[100]; // 임시 변수

char inst[10]; // 오피코드 검색할 명령어

int i = -1; // 임시 변수

OPlist \*ptr; // linked list 탐색할 때 사용하는 변수

// idx에서 명령어 읽어온다

// 파라미터가 1개가 아니면 에러를 출력한다

if (sscanf(idx, "%s %s", inst, tmp) != 1){

printf("Invalid mnemonic\n");

return 0;

}

i = hashFunction(inst); // hash function으로 암호화

ptr = OPhead[i]; // 암호화된 인덱스로 linked list 탐색

// linked list에서 입력된 instruction과 같은 mnemonic을 찾는다

while (ptr != NULL){

if (!strcmp(ptr->inst\_name, inst))

break;

ptr = ptr->link;

}

// 해당하는 mnemonic이 없으면 에러 처리

if (ptr == NULL){

printf("Invalid mnemonic\n");

return 0;

}

// 찾으면 결과 출력

printf("opcode is %02X\n", ptr->opcode);

return 1;

}

void opcodelist(){

int i; // 반복문에 사용할 변수

OPlist \*ptr; // linked list 탐색할 포인터

for (i = 0; i < 20; i++){ // 모든 index 탐색

printf("%d : ", i);

ptr = OPhead[i];

// 한 인덱스에서 연결된 모든 linked list 출력

if (ptr){

while (ptr -> link != NULL){

printf("[%s,%02X] -> ", ptr->inst\_name, ptr->opcode);

ptr = ptr->link;

}

printf("[%s,%02X]", ptr->inst\_name, ptr->opcode);

}

printf("\n");

}

}

/\* end of proj 1 \*/

/\* start of proj 2 \*/

void getfilename(char \*str){

int i = 0, j; // i, j : 배열의 인덱스를 표현

int flag = 0; // 공백을 제거할지 안할지 나타내는 플래그

// 들어온 string의 가장 끝에 의미없는 공백이 있는지 확인

while (str[i] != '\0'){

if (flag == 0) {

if (str[i] == ' ' || str[i] == '\t'){

flag = 1;

j = i;

}

}

else if (flag == 1){

if (str[i] != ' ' && str[i] != '\t')

flag = 0;

}

i++;

}

// 공백을 제거한다

if (flag == 1){

while (str[j] != '\0'){

str[j] = '\0';

j++;

}

}

}

int type(char \*f\_name){

FILE \*fp;

char tmp;

// 파일 열기

getfilename(f\_name);

fp = fopen(f\_name, "r");

// 파일 없으면 종료

if (!fp) {

printf("No such file in this directory\n");

return 0;

}

// 파일 내용 출력

while (1) {

tmp = fgetc(fp);

if (tmp == EOF)

break;

printf("%c", tmp);

}

//파일 닫기

fclose(fp);

return 1;

}

int assemble(char \*f\_name){

FILE \*fp;

int i = 0;

char name\_tmp[100];

char filetmp[100];

symbol \*ptr, \*rmv;

objlist \*obptr, \*obrmv;

memset(filetmp,'\0',sizeof(filetmp));

// free symbol table

for (i = 0; i < 26; i++){

ptr = symhead[i];

while (ptr){

rmv = ptr;

ptr = ptr -> link;

free(rmv);

}

symhead[i] = NULL;

}

symflag = 0;

// free objcode list

obptr = objhead;

while (obptr){

obrmv = obptr;

obptr = obptr -> link;

free(obrmv);

}

objhead = NULL;

// 파일 열기

getfilename(f\_name);

fp = fopen(f\_name, "r");

// 파일이 없으면 에러 출력 후 종료

if (!fp) {

printf("No such file in this directory\n");

return 0;

}

// .asm 파일인지 확인

i = 0;

while (1){

if (f\_name[i] == '.') // i는 '.'이 찍힌 인덱스

break;

name\_tmp[i] = f\_name[i]; // '.' 이 나오기 전까지 파일 이름 저장

i++;

}

i++;

if (strcmp(f\_name+i, "asm")) { // . 다음에 asm이 아니면

printf("Invalid file extension\n"); // 에러 메세지 출력 후

return 0; // 종료

}

// 패스1, 패스2 에서 오류 발생했으면 파일 삭제하고 종료

if (pass\_1(f\_name) == 0){

strcpy(filetmp, name\_tmp);

strcat(filetmp, ".lst");

remove(filetmp);

memset(filetmp, '\0', sizeof(filetmp));

strcpy(filetmp, name\_tmp);

strcat(filetmp, ".obj");

remove(filetmp);

remove("intermediate");

return 0;

}

remove("intermediate");

// 어셈블 정상작동 되면 생성된 파일명 출력

printf("Successfully assemble %s.\n", f\_name);

symflag = 1;

// 파일 닫고 종료

fclose(fp);

return 1;

}

int pass\_1(char \*f\_name){

int locctr = 0;

FILE \*src = fopen(f\_name, "r");

FILE \*intmd = fopen("intermediate", "w");

char line[200];

char s1[30], s2[30], s3[30], s4[30], tmpstr[30];

int lnum = 5, cmflag = -1;

int i, k;

adr\_start = 0;

pro\_length = 0;

int tmp = 0, dectmp;

int endflag = 0;

symbol \*symptr, \*pre, \*cur;

OPlist \*opptr;

int idx;

// 한 줄 읽어오기

fgets(line, sizeof(line), src);

// 받아온 줄에서 인자 받기

tmp = sscanf(line, "%s %s %x %s", s1, s2, &adr\_start, tmpstr);

if (tmp){

// START 있으면

if (!strcmp(s2, "START")){

// 정상 형식일 때 START 라인 출력

if (tmp == 3){

locctr = adr\_start;

fprintf(intmd, "%d\t%d\t", lnum, locctr);

lnum += 5;

fprintf(intmd, "%s", line);

}

else {

printf("%d: error: start line error\n", lnum);

return 0;

}

}

// START 없으면 파일 닫고 다시 열기

else {

fclose(src);

src = fopen(f\_name, "r");

}

}

memset(line, '\0', sizeof(line));

// 파읽 끝에 도달할 때 까지 읽기

while (fgets(line, sizeof(line), src) != NULL){

// 초기화

memset(s1, '\0', sizeof(s1));

memset(s2, '\0', sizeof(s2));

memset(s3, '\0', sizeof(s3));

memset(s4, '\0', sizeof(s4));

memset(tmpstr, '\0', sizeof(tmpstr));

tmp = 0;

cmflag = -1;

// 주석이면 그대로 출력

if (line[0] == '.'){

fprintf(intmd, "%d\t%s", lnum, line);

lnum += 5;

}

// 주석이 아니면

else {

// 콤마 있는지 확인

i = 0;

while (line[i] != '\0'){

if (line[i] == ','){

cmflag = i;

break;

}

i++;

} // cmflag : 콤마의 위치

// 입력 받은 라인에서 인자로 값 할당

// label이 안들어온 경우

if (line[0] == ' ' || line[0] == '\t'){

// 콤마가 있으면

if (cmflag != -1){

tmp = sscanf(line, "%s \t %s \t %s \t %s", s2, s3, s4, tmpstr);

if (line[cmflag+1] != ' ' && line[cmflag+1] != '\t'){

if (line[cmflag -1] != ' ' && line[cmflag -1] != '\t'){

k = 0;

while (s3[k] != ',')

k++;

k++;

strcpy(s4, s3+k);

tmp++;

}

}

else if (s4[0] == ','){

if (strlen(s4) == 1){

strcpy(s4, tmpstr);

tmp--;

}

else{

strcpy(s4, s4+1);

s4[strlen(s4) - 1] = '\0';

}

}

if (tmp != 3){

printf("%d: error: wrong input line\n", lnum);

return 0;

}

tmp--;

}

// 콤마가 없으면

else {

tmp = sscanf(line, "%s \t %s \t %s", s2, s3, tmpstr);

if (tmp != 2) {

if (tmp == 1 && !strcmp(s2, "RSUB"))

tmp = 1;

else {

printf("%d: error: wrong input line\n", lnum);

return 0;

}

}

}

}

// 에러처리

else if (line[0] <= '9' && line[0] >= '0') {

printf("%d: error: first char cannot be number\n", lnum);

return 0;

}

// label이 들어온 경우

else {

// 콤마가 있으

if (cmflag != -1){

tmp = sscanf(line, "%s \t %s \t %s \t %s \t %s", s1, s2, s3, s4, tmpstr);

if (line[cmflag+1] != ' ' && line[cmflag+1] != '\t'){

if(line[cmflag-1] != ' ' && line[cmflag-1] != '\t'){

k = 0;

while (s3[k] != ',')

k++;

k++;

strcpy(s4, s3+k);

tmp++;

}

}

else if (s4[0] == ','){

if (strlen(s4) == 1){

strcpy(s4, tmpstr);

tmp--;

}

else{

strcpy(s4, s4+1);

s4[strlen(s4) - 1] = '\0';

}

}

if (tmp != 4){

printf("%d: error: wrong input line\n", lnum);

return 0;

}

tmp--;

}

// 콤마가 없으면

else {

tmp = sscanf(line, "%s \t %s \t %s \t %s", s1, s2, s3, tmpstr);

if (tmp != 3){

printf("%d: error: wrong input line\n", lnum);

return 0;

}

}

}

/\* s1 : label

s2 : directive

s3 : operand

s4 : second operand after comma \*/

// label 이 있을 때

if (s1[0] != '\0'){

// sybol table에서 label 찾기

idx = ZtoAhash(s1);

symptr = symhead[idx];

while (symptr){

if (!strcmp(symptr->name, s1))

break;

symptr = symptr -> link;

}

// label 있으면 중복 에러

if (symptr){

printf("%d: error: duplicate symbol - %s\n", lnum, s1);

return 0;

}

// 없으면 symbol table에 추가

else {

// 새 노드에 값 할당

symbol \*newNode = (symbol\*)malloc(sizeof(symbol));

strcpy(newNode->name, s1);

newNode->link = NULL;

newNode->locctr = locctr;

// head가 비었으면 head가 새 노드

if (symhead[idx] == NULL)

symhead[idx] = newNode;

// head가 NULL이 아니면

else {

pre = symhead[idx];

cur = pre;

// Z to A 순으로 linked list 저장

while (cur){

k = 0;

while (1){

if (cur->name[k] != s1[k])

break;

k++;

}

if (cur->name[k] < s1[k])

break;

pre = cur;

cur = cur -> link;

}

// 올바른 위치에 new node 삽입

if (cur == pre){

symhead[idx] = newNode;

symhead[idx]->link = cur;

}

else{

pre->link = newNode;

newNode->link = cur;

}

}

}

}

// directive가 opcode list에 있는지 확인

if (s2[0] == '+')

idx = hashFunction(s2+1);

else

idx = hashFunction(s2);

opptr = OPhead[idx];

while (opptr){

if (s2[0] != '+'){

if (!strcmp(opptr->inst\_name, s2))

break;

}

else{

if (!strcmp(opptr->inst\_name, s2+1))

break;

}

opptr = opptr -> link;

}

// opcode list에 directive가 있으면

if (opptr){

// 출력형식에 맞춰 출력

if (s1[0] != '\0')

fprintf(intmd, "%d\t%d\t%s", lnum, locctr, line);

else

fprintf(intmd, "%d\t%d\t-%s", lnum, locctr, line);

// format만큼 location counter 증가

if (!strcmp(opptr->form, "1"))

locctr += 1;

else if (!strcmp(opptr->form, "2"))

locctr += 2;

else if (s2[0] == '+')

locctr += 4;

else

locctr += 3;

// 줄번호 증가

lnum += 5;

}

// directive가 "BASE"일 때

else if (!strcmp(s2, "BASE")){

fprintf(intmd, "%d\t-1\t-%s", lnum, line);

lnum += 5;

}

// directive가 "END"일 때

else if (!strcmp(s2, "END") && tmp <= 2){

fprintf(intmd, "%d\t-1\t-%s", lnum, line);

lnum += 5;

endflag = 1;

break;

}

// directive가 "WORD" 일 때

else if (!strcmp(s2, "WORD") && tmp == 3) {

fprintf(intmd, "%d\t%d\t%s", lnum, locctr, line);

locctr += 3;

memset(tmpstr, '\0', sizeof(tmpstr));

tmp = sscanf(s3, "%d %s", &dectmp, tmpstr);

if (tmp != 1){

printf("%d: error: 'WORD' operand error - %s\n", lnum, s3);

return 0;

}

}

// directive가 "BYTE" 일때

else if (!strcmp(s2, "BYTE") && tmp == 3){

fprintf(intmd, "%d\t%d\t", lnum, locctr);

// 뒤에 C이면

if (s3[0] == 'C')

locctr += strlen(s3) -3;

// 뒤에 X이면

else if (s3[0] == 'X')

locctr += (strlen(s3) - 2) / 2;

// 둘다 아니면 에러

else {

printf("%d: error: 'BYTE' input error\n", lnum);

return 0;

}

lnum += 5;

fprintf(intmd, "%s", line);

}

// directive가 "RESW"일 때

else if (!strcmp(s2, "RESW") && tmp == 3){

fprintf(intmd, "%d\t%d\t%s", lnum, locctr, line);

lnum += 5;

locctr += 3 \* atoi(s3);

}

// directive가 "RESB"일 때

else if (!strcmp(s2, "RESB") && tmp == 3){

fprintf(intmd, "%d\t%d\t%s", lnum, locctr, line);

lnum += 5;

locctr += atoi(s3);

}

// 위에 언급된 것 중에 없고 opcodelist에도 없으면 에러

else {

printf("%d: error: Invlid directive - %s\n", lnum, s2);

return 0;

}

}

memset(line, '\0', sizeof(line));

}

// END가 없으면 에러 처리 후 종료

if (endflag == 0){

printf("error: No END sign\n");

return 0;

}

// 파일 포인터 종료

fclose(src);

fclose(intmd);

// 프로그램 길이

pro\_length = locctr - adr\_start;

// 패스2에서 오류 발생시 종료

if (pass\_2(f\_name) == 0)

return 0;

return 1;

}

int pass\_2(char \*f\_name){

FILE \*intmd, \*obj, \*lst;

char of\_name[100], lf\_name[100];

char line[200];

char s1[30], s2[30], s3[30], s4[30];

int k = 0, l = 0;

int n, i, x, b, p, e;

int pc\_adr = adr\_start;

int b\_adr = 0;

int tmp = 0;

int lnum, locctr = 0;

int cmflag = -1;

int format;

int idx;

OPlist \*opptr;

symbol \*symptr;

char reg[10][2];

char objcode[10];

int newop, disp;

int dectmp, xbpe;

char strtmp[100], tmpstr[100];

char bytetmp[100];

// 인덱스가 레지스터 값

strcpy(reg[0], "A");

strcpy(reg[1], "X");

strcpy(reg[2], "L");

strcpy(reg[3], "B");

strcpy(reg[4], "S");

strcpy(reg[5], "T");

strcpy(reg[6], "F");

strcpy(reg[7], " ");

strcpy(reg[8], "PC");

strcpy(reg[9], "SW");

memset(of\_name, '\0', sizeof(of\_name));

memset(lf\_name, '\0', sizeof(lf\_name));

// lst, obj 파일명 생성하여 쓰기로 파일 열기

while (f\_name[k] != '.'){

of\_name[k] = f\_name[k];

lf\_name[k] = f\_name[k];

k++;

}

strcat(of\_name, ".obj");

strcat(lf\_name, ".lst");

// 파일 열기

intmd = fopen("intermediate", "r");

obj = fopen(of\_name, "w");

lst = fopen(lf\_name, "w");

// 첫 줄 받아온다

fgets(line, sizeof(line), intmd);

tmp = sscanf(line, "%d \t %d \t %s \t %s \t %s", &lnum, &locctr, s1, s2, s3);

if (tmp){

// START가 있으면 lst, obj 파일 작성

if (!strcmp(s2, "START")){

fprintf(lst, "%d\t%04X\t%s\t%s\t%X\n", lnum, locctr, s1, s2, adr\_start);

fprintf(obj, "H%-6s%06X%06X\n", s1, adr\_start, pro\_length);

}

// 없으면 obj 파일 H 부분 작성

else {

fclose(intmd);

intmd = fopen("intermediate", "r");

fprintf(obj, "H %06X%06X\n", adr\_start, pro\_length);

}

}

// 계속해서 파일 읽기

while (1){

// 초기화

n = 1;

i = 1;

x = 0;

b = 0;

p = 0;

e = 0;

memset(line, '\0', sizeof(line));

memset(s1, '\0', sizeof(s1));

memset(s2, '\0', sizeof(s2));

memset(s3, '\0', sizeof(s3));

memset(s4, '\0', sizeof(s4));

memset(objcode, '\0', sizeof(objcode));

memset(strtmp, '\0', sizeof(strtmp));

memset(bytetmp, '\0', sizeof(bytetmp));

memset(tmpstr, '\0', sizeof(tmpstr));

cmflag = -1;

tmp = 0;

xbpe = 0;

disp = 0;

// 초기화

fgets(line, sizeof(line), intmd);

if (line[0] == '\0')

break;

// 콤마 있는지 확인

k = 0;

while (line[k] != '\0'){

if (line[k] == ','){

cmflag = k;

break;

}

k++;

} // cmflag : 콤마의 위치

// 인자 입력받기

// 콤마 없으면

if (cmflag == -1){

tmp = sscanf(line, "%d \t %d \t %s \t %s \t %s", &lnum, &locctr, s1, s2, s3);

}

// 콤마 있으면

else {

tmp = sscanf(line, "%d %d \t %s \t %s \t %s \t %s \t %s", &lnum, &locctr, s1, s2, s3, s4, tmpstr);

if (line[cmflag+1] != ' ' && line[cmflag+1] != '\t'){

if(line[cmflag-1] != ' ' && line[cmflag-1] != '\t'){

k = 0;

while (s3[k] != ',')

k++;

k++;

strcpy(s4, s3+k);

tmp++;

s3[strlen(s3) - strlen(s4) - 1] = '\0';

}

}

else if (s4[0] == ','){

if (strlen(s4) == 1){

strcpy(s4, tmpstr);

tmp--;

}

else{

strcpy(s4, s4+1);

s4[strlen(s4) - 1] = '\0';

}

}

else{

s3[strlen(s3) - 1] = '\0';

}

}

/\* s1 : label

s2 : directive

s3 : operand

s4 : 2nd operand after comma \*/

// 주석이면 그대로 출력

if (s1[0] == '\0'){

fprintf(lst, "%s", line);

}

else {

// 'END' 면 종료

if (!strcmp(s2, "END")){

fprintf(lst, "%d\t\t\t%s\t%s\n", lnum, s2, s3);

break;

}

// 'BASE'면 b 레지스터에 값 넣

if (!strcmp(s2, "BASE")){

idx = ZtoAhash(s3);

symptr = symhead[idx];

while (symptr){

if (!strcmp(symptr->name, s3))

break;

symptr = symptr -> link;

}

if (!symptr){

printf("%d: error: 'BASE' symbol error - %s\n", lnum, s3);

return 0;

}

b\_adr = symptr->locctr;

fprintf(lst, "%d\t\t\t%s\t%s\n", lnum, s2, s3);

}

// operand 확인

switch (s3[0]){

// '#' 있으면 immediate addresing

case '#':

n = 0;

i = 1;

b = 0;

if (s3[1] >= '0' && s3[1] <= '9')

p = 0;

break;

// '@' 있으면 indirect addressing

case '@':

n = 1;

i = 0;

break;

// 보통의 경우는 simple addressing

default:

n = 1;

i = 1;

break;

}

// opcode list에 있는 명령어인지 확인

if (s2[0] == '+')

idx = hashFunction(s2+1);

else

idx = hashFunction(s2);

opptr = OPhead[idx];

while (opptr){

if (s2[0] != '+'){

if (!strcmp(opptr->inst\_name, s2))

break;

}

else{

if (!strcmp(opptr->inst\_name, s2+1))

break;

}

opptr = opptr -> link;

}

// opcode list에 입력된 mnemonic이 있으면

if (opptr){

// 몇 형식인지 확인

if (!strcmp(opptr->form, "1"))

format = 1;

else if (!strcmp(opptr->form, "2"))

format = 2;

else if (s2[0] == '+'){

format = 4;

b = 0;

p = 0;

e = 1;

}

else

format = 3;

// pc register에 값 저장

pc\_adr = locctr + format;

// 1형식인 경우

// OPCODE가 obj code

if (format == 1){

if (s3[0] != '\0'){

printf("%d: error: format1 cannot have operand\n", lnum);

return 0;

}

if (s1[0] == '-')

fprintf(lst, "%d\t%04X\t\t%s\t\t%02X\n", lnum, locctr, s2, opptr->opcode);

else

fprintf(lst, "%d\t%04X\t%s\t%s\t\t%02X\n", lnum, locctr, s1, s2, opptr->opcode);

// 오브젝트 코드 문자열로 저장

sprintf(objcode, "%02X", opptr->opcode);

add\_objlist(objcode, locctr, 0);

}

// 2형식인 경우

// OPcode 뒤에 register 번호

else if (format == 2){

// 인자 없으면 에러

if (s3[0] == '\0'){

printf("%d: error: format2 need at least one operand\n", lnum);

return 0;

}

// 유효한 레지스터인지 확인

for (k = 0 ; k < 10 ; k++){

if (!strcmp(s3, reg[k]))

break;

}

// 아니면 에러

if (k == 10){

printf("%d: error: wrong register name - %s\n", lnum, s3);

return 0;

}

// 파일에 출력

if (s1[0] == '-')

fprintf(lst, "%d\t%04X\t\t%s\t%s", lnum, locctr, s2, s3);

else

fprintf(lst, "%d\t%04X\t%s\t%s\t%s", lnum, locctr, s1, s2, s3);

l = 0;

if (cmflag != -1){ // 콤마 있으면

for (l = 0; l < 10 ; l++){

if (!strcmp(s4, reg[l]))

break;

}

if (l == 10){

printf("%d: error: wrong register name - %s\n", lnum, s4);

return 0;

}

fprintf(lst, ", %s", s4);

}

fprintf(lst, "\t\t%02X%d%d\n", opptr->opcode, k, l);

// 오브젝트 코드 문자열로 저장

sprintf(objcode, "%02X%d%d", opptr->opcode, k, l);

add\_objlist(objcode, locctr, 0);

}

// 3형식인 경우

else if (format == 3){

// opcode 뒤에 2 bits 값을 n, i로 대체

newop = opptr->opcode / 2 / 2 \* 2 \*2 + 2 \* n + i;

// xbpe 값

xbpe = 2\*2\*2\*x+2\*2\*b+2\*p+e;

// immediate addressing (#)

if (n == 0 && i == 1){

tmp = sscanf(s3+1, "%d %s", &dectmp, strtmp);

// # 뒤에 상수 이면

if (tmp == 1){

// object 코드 저장

sprintf(objcode, "%02X%01X%03X", newop, xbpe, dectmp);

}

else{

tmp = strlen(s3) - 1;

strcpy(s3, s3+1);

s3[tmp] = '\0';

}

}

// 상수가 아니어서

//오브젝코드가 생성되지 않았을때

if (objcode[0] == '\0'){

// indirect addressing (@)

if (n == 1 && i == 0){

tmp = strlen(s3) - 1;

strcpy(s3, s3+1);

s3[tmp] = '\0';

}

// symbol table에 있는 operand 인지 확인

if (s3[0] != '\0'){

idx = ZtoAhash(s3);

symptr = symhead[idx];

while (symptr){

if (!strcmp(symptr->name, s3))

break;

symptr = symptr -> link;

}

// 없으면 에러 출력하고 종료

if (!symptr){

printf("%d: error: symbol error - %s\n", lnum, s3);

return 0;

}

// disp 값 저장

p = 1; b = 0;

disp = symptr->locctr - pc\_adr;

// pc relative 사용할 수 없는 경우

if (disp > 2047 || disp < (-2048)){

// base relative 사용

b = 1;

p = 0;

disp = symptr->locctr - b\_adr;

}

if (disp < 0)

disp = disp & 4095;

if (cmflag != -1){ // 콤마 있을 때

// X 레지스터 들어오면 x 조정

if (!strcmp(s4, "X")){

x = 1;

}

// X 레지스터 아니면 에러

else {

printf("%d: error: wrong register name - %s\n", lnum, s4);

return 0;

}

}

}

// obj code 저장

xbpe = 2\*2\*2\*x + 2\*2\*b + 2\*p + e;

sprintf(objcode, "%02X%01X%03X", newop, xbpe, disp);

}

// lst 파일에 출력

if (s1[0] == '-')

fprintf(lst, "%d\t%04X\t\t%s\t%s", lnum, locctr, s2, s3);

else

fprintf(lst, "%d\t%04X\t%s\t%s\t%s", lnum, locctr, s1, s2, s3);

if (cmflag != -1)

fprintf(lst, ", %s\t%s\n", s4, objcode);

else

fprintf(lst, "\t\t%s\n", objcode);

add\_objlist(objcode, locctr, 0);

}

// 4형식인 경우

else if (format == 4){

// opcode 뒤에 2 bits 값을 n, i로 대체

newop = opptr->opcode / 2 / 2 \* 2 \*2 + 2 \* n + i;

// xbpe 값

xbpe = 2\*2\*2\*x+2\*2\*b+2\*p+e;

// immediate addressing (#)

if (n == 0 && i == 1){

tmp = sscanf(s3+1, "%d %s", &dectmp, strtmp);

// # 뒤에 상수 이면

if (tmp == 1){

// object 코드 저장

sprintf(objcode, "%02X%01X%05X", newop, xbpe, dectmp);

}

else{

tmp = strlen(s3) - 1;

strcpy(s3, s3+1);

s3[tmp] = '\0';

}

add\_objlist(objcode, locctr, 0);

}

// 상수가 아니어서

//오브젝코드가 생성되지 않았을때

if (objcode[0] == '\0'){

// indirect addressing (@)

if (n == 1 && i == 0){

tmp = strlen(s3) - 1;

strcpy(s3, s3+1);

s3[tmp] = '\0';

}

// symbol table에 있는 operand 인지 확인

if (s3[0] != '\0'){

idx = ZtoAhash(s3);

symptr = symhead[idx];

while (symptr){

if (!strcmp(symptr->name, s3))

break;

symptr = symptr -> link;

}

// 없으면 에러 출력하고 종료

if (!symptr){

printf("%d: error: symbol error - %s\n", lnum, s3);

return 0;

}

if (cmflag != -1){ // 콤마 있을 때

// X 레지스터 들어오면 x 조정

if (!strcmp(s4, "X")){

x = 1;

}

// X 레지스터 아니면 에러

else {

printf("%d: error: wrong register name - %s\n", lnum, s4);

return 0;

}

}

}

// obj code 저장

xbpe = 2\*2\*2\*x + 2\*2\*b + 2\*p + e;

sprintf(objcode, "%02X%01X%05X", newop, xbpe, symptr->locctr);

add\_objlist(objcode, locctr, 1);

}

// lst 파일에 출력

if (s1[0] == '-')

fprintf(lst, "%d\t%04X\t\t%s\t%s", lnum, locctr, s2, s3);

else

fprintf(lst, "%d\t%04X\t%s\t%s\t%s", lnum, locctr, s1, s2, s3);

if (cmflag != -1)

fprintf(lst, ", %s\t%s\n", s4, objcode);

else

fprintf(lst, "\t\t%s\n", objcode);

}

}

// 입력이 opcode list에 저장된 mnemonic이 아니면

else {

// 'BYTE'인 경우

if (!strcmp(s2, "BYTE")){

// 'C' 형식으로 들어올 때

if (s3[0] == 'C'){

strcpy(bytetmp, s3+2);

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

sprintf(objcode, "%02X", bytetmp[0]);

k = 1;

while (bytetmp[k] != '\0'){

sprintf(strtmp, "%02X", bytetmp[k]);

strcat(objcode, strtmp);

k++;

}

}

// 'X' 형식으로 들어올 때

else if (s3[0] == 'X'){

strcpy(objcode, s3+2);

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

}

fprintf(lst, "%d\t%04X\t%s\t%s\t%s\t\t%s\n", lnum, locctr, s1, s2, s3, objcode);

add\_objlist(objcode, locctr, 0);

}

// 'WORD'인 경우

else if (!strcmp(s2, "WORD")){

sscanf(s3, "%d", &k);

sprintf(objcode, "%06X", k);

fprintf(lst, "%d\t%04X\t%s\t%s\t%s\t\t%s\n", lnum, locctr, s1, s2, s3, objcode);

add\_objlist(objcode, locctr, 0);

}

// 'RESB' 나 'RESW' obj code 없음

else if (!strcmp(s2, "RESB") || !strcmp(s2, "RESW")){

fprintf(lst,"%d\t%04X\t%s\t%s\t%s\n", lnum, locctr, s1, s2, s3);

}

}

}

}

// .lst 파일 출력 완료

// .obj 파일 출력 시작

// T 부분

objlist \*objptr = objhead;

char ttmp[62];

memset(ttmp, '\0', sizeof(ttmp));

do{

fprintf(obj, "T");

fprintf(obj, "%06X", objptr->loc);

while (objptr && strlen(ttmp) + strlen(objptr->objcode) <= 62){

strcat(ttmp, objptr->objcode);

objptr = objptr -> link;

if (!objptr)

break;

if (objptr->flag == 1)

break;

}

fprintf(obj, "%02X", (unsigned int)strlen(ttmp) / 2);

fprintf(obj, "%s\n", ttmp);

memset(ttmp, '\0', sizeof(ttmp));

} while (objptr);

// M 부분

objptr = objhead;

while (objptr){

if (objptr->mflag ==1){

fprintf(obj, "M");

fprintf(obj, "%06X", objptr->loc+ 1);

fprintf(obj, "05\n");

}

objptr = objptr->link;

}

// E 부분

fprintf(obj, "E");

fprintf(obj, "%06X\n", adr\_start);

fclose(intmd);

fclose(obj);

fclose(lst);

return 1;

}

int ZtoAhash(char\* label){

// Z to A 순서로

return 25 - (label[0] - 'A');

}

int symbolcmd(){

symbol \*ptr;

int i = 0;

// symbol table 만들어져있는지 확인

if (symflag == 1){

// symbol table 내용 차례로 출력

while (i < 26){

ptr = symhead[i];

while (ptr) {

printf("\t%s\t%04X\n", ptr->name, ptr->locctr);

ptr = ptr -> link;

}

i++;

}

}

// table 없으면 에러

else {

printf("No symbol table\n");

return 0;

}

return 1;

}

void add\_objlist(char\* objcode, int loc, int m){

objlist \*newNode = (objlist\*)malloc(sizeof(objlist)); // 새로운 값을 저장할 노드

objlist \*ptr = objhead; // linked list 탐색에 사용되는 임시변수

strcpy(newNode->objcode, objcode);

newNode->link = NULL;

newNode->loc = loc;

newNode->flag = 0;

newNode->mflag = 0;

// 새 노드에 값 저장

if (objhead == NULL) { // head가 NULL 이면

objhead = (objlist\*)malloc(sizeof(objlist));

objhead = newNode;

// head 할당한 후 새 노드 저장

if (m == 1)

newNode->mflag = 1;

}

else { // head가 NULL하지 않으면

while (ptr->link != NULL)

ptr = ptr->link; // 새 노드를 연결할 노드까지 이동

ptr-> link = newNode; // 새 노드 연결

if ((newNode->loc) - (ptr->loc) >= 4096)

newNode->flag = 1;

if (m == 1)

newNode->mflag = 1;

}

}

/\* end of proj 2 \*/