**과목명: 시스템프로그래밍**

**1분반**

**<<Project #1>>**

**서강대학교 [컴퓨터공학과]**

**[20181617]**

**[김채연]**

목 차

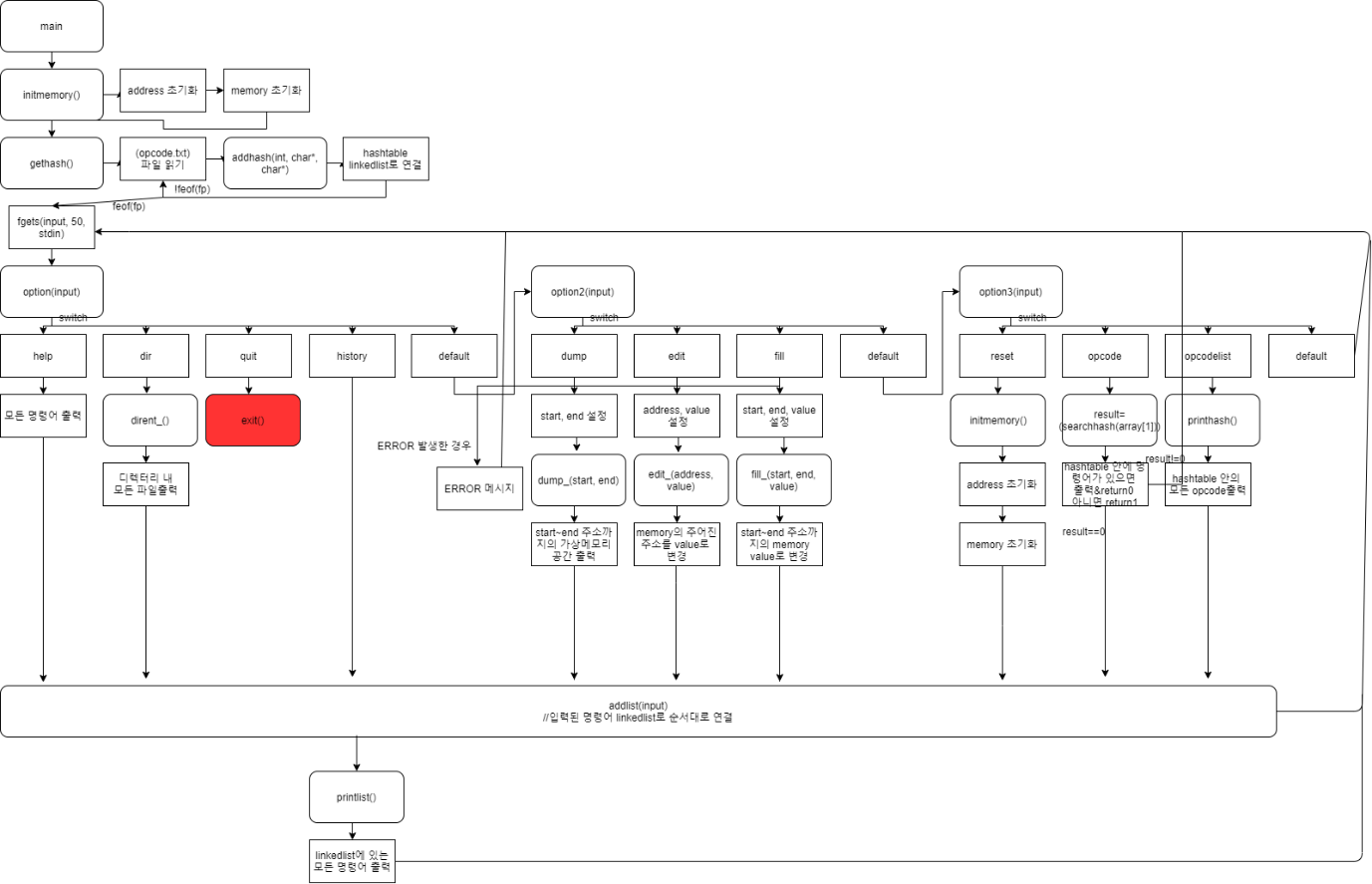
1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
   2. 프로그램 기능
3. **모듈 정의**
4. **전역 변수 정의**
5. **코드 설명**

**1. 프로그램 개요**

SIC/XE 머신을 구현하기 위한 전 단계로서 어셈블러, 링크, 로더 들을 실행하게 될 shell과 컴파일을 통해서 만들어진 object 코드가 적재되고 실행될 메모리 공간과 mnemonic 을 opcode값으로 변환하는 opcode 테이블과 관련 명령어들을 구현하는 프로그램이다.

**2. 프로그램 설명**

2.1 프로그램 흐름도



2.2 프로그램 기능

1) shell 관련 명령어

1. help

shell 에서 실행 가능한 모든 명령어들의 리스트를 화면에 출력해준다.

1. dir

현재 디렉터리에 있는 파일들을 출력한다. 실행파일 이름 옆에는 ‘\*’, 디렉터리는 ‘/’표시 를 한다.

1. quit

sicsim을 종료한다.

1. History

현재까지 사용한 명령어들을 순서대로 번호와 함께 보여준다. 가장 최근 사용한 명령어 가 리스트의 하단에 오도록 한다.

2) 메모리 관련 명령어

1. dump start, end

입력된 주소 범위 내의 할당되어 있는 메모리의 내용을 출력한다.

1. edit address, value

입력된 주소의 메모리 내용을 value로 변경한다.

1. fill start, end, value

입력된 주소 범위 내의 메모리 내용을 value로 모두 변경한다.

1. reset

메모리 전체를 전부 0으로 변경시킨다.

3) OPCODE TABLE 관련 명령어

1. opcode mnemonic

명령어를 입력하면 해당하는 opcode를 출력한다.

1. opcodelist

opcode Hash Table의 내용을 모두 출력한다.

**3. 모듈 정의**

1) stdio.h

표준 입출력 함수를 담고있는 헤더파일

2) string.h

C형식 문자열 관련 함수를 담고있는 헤더 파일

3) stdlib.h

난수생성, 문자열 형식을 다른 형식으로 변환, 의사 난수생성, 동적 메모리 관리 등의 함수를 담고있는 헤더 파일

4) sys/stat.h

파일의 상태를 확인하기 위한 자료형, 구조체, 상수와 관련된 함수를 담고있는 헤더 파일

5) dirent.h

파일 시스템의 디렉터리를 나타내기 위한 구조체를 정의하는 헤더 파일

6) enum

열거형이라고 부르며 정수형 상수에 이름을 붙여서 코드를 작성하고 이해하는 것을 쉽게 도와준다.

7) typedef

어떠한 자료형 이름에 새 이름을 부여하여 정의하는 것이다.

8) struct

하나 이상의 변수를 묶어서 좀 더 편리하게 사용할 수 있도록 도와준다.

3.1 함수 설명

1) int main() : 메인함수

char input[50] : 입력받는 명령어, 최대 길이는 50으로 설정

char \*p : 입력받을 때 엔터가 들어온 경우 엔터를 없애주기 위해 필요한 변수

2) void initmemory() : 메모리를 초기화하는 함수

int i, j : memory가 2차원배열, address가 1차원배열이기 때문에 배열의 순서를 지정하는데 필요한 변수

3) void gethash() : 파일에서 데이터를 읽는 함수

FILE \*fp : “opcode.txt” 파일을 여는데 필요한 포인터

char value2[5], char ins[50], char n[50] : 파일의 데이터를 읽어서 각각 저장하는 변수

int value : opcode를 char로 받기 때문에 int로 바꿔 저장하는데 필요한 변수

4) void addhash(int value, char\* ins, char\*n) : value, ins, n을 받아 hashtable을 만드는 함수

int order : 입력된 명령어가 몇번째 table에 들어갈지 지정하는 변수

node\_\*new : table의 node 구조체

5) void option (char\* input) : help, dir, quit, history 명령어를 처리하는 함수

char input2[50] : input을 복사해놓은 변수

int I : for문을 돌리는데 필요한 변수

char \* array : 명령어를 공백을 없애고 저장하는 변수

int i2 : i를 2로 나눈 몫을 저장하는 변수

6) void option2(char\* input) : dump, edit, fill 명령어를 처리하는 함수

char array[4][50] : 인자가 , 로 구분되어 있기 때문을 명령어와 각각의 인자를 따로 저장하는 변수

int len : input의 길이

int I, j, k : for문을 돌리는데 필요한 변수

int c : , 의 개수를 세는 변수

int start, end, address, value : 각 명령어에 필요한 변수

int i2 : i를 2로 나눈 몫을 저장하는 변수

int len1 : array[1]의 길이

int len2 : array[2]의 길이

int len3 : array[3]의 길이

7) void option3 (char\* input) : reset, opcode, opcodelist 명령어를 처리하는 함수

char array[2][50] : 인자가 공백으로 구분되기 때문에 각각의 인자를 따로 저장하는 변수

int len : input의 길이

int I, j, k : for문을 돌리는데 필요한 변수

int result : searchhash의 결과를 저장하는 변수

8) void printhash() : hashtable에 있는 모든 데이터를 출력하는 함수

int order : table의 순서를 지정하는 변수

node\_\* temp : table의 node 구조체

9) int searchhash(char \* array) : hashtable에 해당 데이터가 있는지 검색하는 함수

int order : 검색할 table의 순서를 지정하는 변수

node\_\* temp : table의 node 구조체

int len1 : array의 길이

int len2 : temp의 ins 길이

10) int check (char c) : c가 16진수인지 검사하는 함수

11) void dump\_(int start, int end) : 메모리 공간을 출력하는 함수

int start\_a : start 주소의 줄 memory[start\_a][]

int start\_b : start 주소의 칸 memory[][start\_b]

int end\_a : end 주소의 줄 memory[end\_a][]

int end\_b : end 주소의 칸 memory[][end\_b]

int I, j : for문을 돌리는데 필요한 변수

12) void edit\_(int address, int value) : 메모리 공간을 변경하는 함수

int start\_a : start주소의 줄

int start\_b : start주소의 칸

13) void fill\_(int start, int end, int value) : 메모리 공간을 변경하는 함수

int start\_a : start 주소의 줄 memory[start\_a][]

int start\_b : start 주소의 칸 memory[][start\_b]

int end\_a : end 주소의 줄 memory[end\_a][]

int end\_b : end 주소의 칸 memory[][end\_b]

int I, j : for문을 돌리는데 필요한 변수

14) void dirent\_() : 현재 디렉터리 내 파일을 출력하는 함수

DIR\* dir : 디렉터리 포인터

struct dirent \*ent : 디렉터리 구조체

struct stat st : 파일상태 구조체

15) void addlist (char\* input) : 입력된 올바른 명령어를 저장하는 함수

int I : 순서를 지정하는 변수

node\* new : linkedlist의 node 구조체

node\* temp : linkedlist의 node구조체 head로 지정

16) void printlist() : linkedlist에 저장되어 있는 모든 명령어를 출력하는 함수

node\* temp : linkedlist의 node 구조체 head로 지정

**4. 전역 변수 정의**

1) enum instruction

switch문으로 명령어를 구분할 때 조금 더 편하게 코드를 작성하기 위해 명령어들을 enum을 이용하여 선언해 주었다.

2) char \*inst\_[]

명령어가 예를 들어 help와 h는 같은 명령어로 취급되어야 되므로 strcmp를 쓸 때 조금 더 편하게 코드를 작성하기 위해서 입력될 수 있는 모든 명령어를 배열에 넣어두었다.

3) typedef struct NODE

History가 입력되었을 때 이때까지 입력된 명령어들을 모두 출력해야 되는데 그 명령어들을 linkedlist로 저장하기 위해 필요한 구조체이다. num은 linkedlist로 연결된 순서이고 data는 node에 입력된 명령어이고 next는 linkedlist로 연결하기 위한 링크이다. typedef로 선언하여 조금 더 사용하기 편하게 하였다.

4) node\* head

node 구조체의 head를 NULL로 설정하였다.

5) typedef struct NODE\_

hashtable을 구성할 때 필요한 구조체이다. value는 opcode값이고 ins는 명령어, n은 함께 입력되는 숫자이고 next는 linkedlist로 연결하기 위한 링크이다. typedef로 설정하여 조금 더 사용하기 편하게 하였다.

6) node\_\* table[20]

hashtable의 사이즈가 20이어야 되므로 table의 변수 이름으로 node\_를 20개의 배열로 하였고 모두 NULL로 설정하였다.

7) int memory[65536][16]

가상메모리 공간의 memory를 2차원 배열로 구현하였다. 그래서 int 배열로 하였고 크기는 줄과 칸으로 생각하여 65536, 16으로 했다.

8) int address[65536]

가상메모리 공간의 address를 65536개의 줄로 표현하기 위해 그만큼의 크기로 int 배열로 설정하였다.

9) int dumpstart

‘dump’ 가 입력되었을 때 그 전에 출력된 주소의 다음 번지부터 160개를 출력해야 하므로 그전에 출력된 주소+1을 저장하는 변수이다.

**5. 코드 설명**

1) main

메인함수이다. 처음에 initmemory를 불러 가상 메모리 공간을 초기화한다. 그리고 gethash를 통해 opcode.txt파일의 내용을 읽어 Hash Table을 만든다. 그런 다음 “sicsim>” 을 출력하여 명령어 input을 입력받는다. 명령어의 길이는 50으로 설정하였다. 명령어를 fgets로 받기 때문에 엔터인 ‘\n’도 입력되므로 이것을 ‘\0’으로 바꿔주었다. 그리고 다음 함수인 option에 input을 넘겨준다.

2) initmemory

가상 메모리 공간을 초기화하는 함수이다. Address와 memory를 따로 배열을 설정하였기 때문에 각각 따로 초기화한다. Address의 경우 한줄 씩 늘어나기 때문에 16씩 더해 주었고 memory는 모두 0으로 초기화하였다.

3) gethash

Hash table을 만드는 함수이다. Opcode.txt파일을 열어서 fscanf를 통해 각각 value2, ins, n으로 string형식으로 파일이 끝날 때까지 값을 받는다. 그리고 value2는 16진수 char로 표현되어있기 때문에 strtol을 이용하여 10진수 int 형식으로 바꾸어 value에 저장한다. 그다음 addhash함수에 값을 넘겨서 hash table에 저장되도록 한다.

4) addhash

opcode 값들을 받아서 hash table에 저장하는 함수이다. 명령어인 ins의 첫 알파벳을 아스키코드로 변환한 값을 20으로 나누어 그 나머지 값 순서로 table을 구성하였다. 나머지 값이 0~19까지 나오기 때문에 table의 사이즈는 20이다. Hash table은 linkedlist로 구현하였다. 전역변수로 table의 첫번째 칸들은 20개로 배열로 선언하였다. 만약 같은 나머지 값을 가지게 된다면 그 순서의 칸에 linkedlist로 연결해주었다. 그래서 만약 그 순서번째 table 칸이 비어있다면 그 칸에 값들을 바로 넣어주었고 만약 비어있지 않다면 그 칸의 마지막으로 linkedlist로 연결해주었다.

5) option

help, dir, quit, history 의 명령어들을 구분하는 함수이다. 명령어 뒤에 공백이 들어오는 경우가 있을 수 있기 때문에 strtok을 이용하여 공백 전까지의 명령어로 구분하였다. 그다음 strcmp를 통해 명령어가 4개의 명령어 중 무엇인지 비교하였다. h와 help는 같은 명령어로 취급해야 된다. 그래서 inst\_에 help, h, dir, d, quit, q, history, hi 순서대로 저장되어 있어서 각각 비교해서 i를 통해 몇번째 순서인지 센 다음 2로 나누어 그 몫으로 명령어를 구분하였다. 몫은 switch문을 이용하여 0은 help, 1은 dir, 2는 quit, 3은 history로 구분하였다.

Help는 shell에서 실행 가능한 모든 명령어들의 리스트를 화면에 print 해주고 history에 더해주기 위해 addlist 함수를 호출했다. 그다음 dir는 dirent\_함수를 부른뒤 addlist를 했다. quit은 exit(0)을 통해 프로그램을 종료했다. History는 먼저 addlist를 한뒤 printlist를 불렀다.

이외의 명령어가 들어오면 그 다음 명령어를 구분하는 함수인 option2를 불렀다.

6) option2

dump, edit, fill의 명령어들을 구분하는 함수이다. 입력된 input을 명령어와 start, end, address, value로 나누기 위해 for문을 사용하여 input을 char 하나씩 비교하여 ‘,’를 기준으로 array[]에 순서대로 넣어주었다. 인자가 더 들어오는 경우를 방지하기 위해 명령어 3개의 최대 인자가 3개 까지만 들어가기 때문에 일단 ‘,’가 3개가 들어오는 순간 에러로 인식하도록 했다. 그리고 입력이 숫자와 알파벳만 들어오도록 하여 만약 이외의 것이 들어오면 에러로 인식하였다. 그다음 명령어 구분은 option함수에서 한 것과 같이 똑같이 하였다. switch문을 통해 dump, edit, fill로 구분하였다.

Dump는 최대 주소값 2개를 받아야 되기 때문에 일단 array[1]과 array[2]의 값들을 주소값으로 변경해주었다. 모두 16진수여야 되므로 16진수의 값으로 입력되지 않으면 error처리를 했다. 만약 array[1]이 입력되지 않았다면 “dump”의 경우이기 때문에 start는 dumpstart로 end는 start+159로 하였다. array[2]만 입력되지 않았다면 “dump start”의 경우이기 때문에 start는 strtol을 통해 array[1]의 16진수를 10진수 int로 바꾸었고 end는 start+159를 해주었다. 둘다 입력이 있다면 “dump start, end”의 경우이기 때문에 start와 end둘다 strtol을 사용하여 값을 설정해주었다. 만약 start가 end보다 큰 경우나 주소 값이 0~1048575의 사이가 아닐 경우 error처리를 해주었다. 그리고 dump는 인자를 2개까지만 필요로 하기 때문에 만약 3개가 들어왔을 경우도 error처리를 해주었다. dumpstart가 끝주소 번지인 경우는 다시 처음부터 시작하도록 0으로 설정하였고 end가 1048575보다 큰 경우는 주소의 끝까지만 출력하기 위해 1048575로 설정해주었다. 그다음 dump\_함수를 불러 화면에 출력하도록 하였고 dumpstart는 end+1을 해주어 다음 dump를 할 경우 다음 주소 번지부터 출력되도록 하였다. 그리고 addlist를 했다.

Edit은 address와 value 2개를 받아야 한다. 이것도 모두 16진수이어야 되므로 strtol을 이용 하였고 만약 이외의 값이 입력되었다면 error처리를 해주었다. Array[1]은 address이므로 dump와 똑같이 정해진 주소값을 벗어났다면 error처리를 했다. Value는 0부터 ff까지의 값이어야 되므로 이외의 값들이 입력되었다면 이것도 error처리를 했다. 그리고 인자가 2개가 아닌 경우도 모두 error처리를 해주었다. 그다음 edit\_함수를 호출하고 addlist를 했다.

Fill은 start, end, value 3개를 받아야 한다. 인자가 3개가 아닌 경우 모두 error처리를 해주었고 각각 위에서 설명한 것과 같이 정해진 범위와 형식이 아니라면 모두 Error처리를 했다. 그다음 fill\_함수를 불렀고 addlist를 해주었다.

이 이외의 명령어가 들어오면 그 다음 명령어를 구분하는 함수인 option3를 불렀다.

7) option3

Reset, opcode mnemonic, opcodelist의 명령어들을 구분하는 함수이다. opcode명령어는 인자 하나가 필요하기 때문에 공백을 기준으로 array[0]과 array[1]로 나누었다. 앞선 option들과 똑같이 strcmp를 통해 array[0]과 명령어를 비교하여 switch문으로 reset, opcode, opcodelist로 구분해주었다.

Reset은 메모리를 초기화 하기 위해서 initmemory함수를 불러 초기화해주었고 addlist 했다.

Opcode 는 searchhash함수를 통해 인자가 hash table안에 있는지 판단하고 그 결과를 result로 받았다. Result가 0이라면 테이블안에 있는 것이므로 addlist를 하였고 만약 table에 없는 명령어였다면 addlist를 하지 않았다.

Opcodelist는 printhash함수를 통해 hash table의 모든 값을 print하고 addlist하였다.

이외의 명령어가 들어온다면 무시한다.

8) printhash

Hash table에 있는 모든 값들을 print해주는 함수이다. Table[0]부터 시작해서 만약 linkedlist로 연결되어 있다면 끝까지 각각 명령어와 opcode를 출력해주고 그다음 table로 넘어간다. 연결되어 있지 않다면 table값만 출력해주고 만약 table에 값이 없다면 다음으로 넘어간다. table의 0~19까지 모든 값을 print하면 끝난다.

9) searchhash

입력된 명령어가 hashtable에 있는지 검색하는 함수이다. 명령어의 첫번째 알파벳을 20으로 나눈 나머지 값을 order로 설정하여 table의 몇번째에 있을지 먼저 찾는다. 그다음 먼저 명령어의 길이가 같은지 판단하고 같다면 table[order]에서 strcmp를 통해 명령어와 일치하는 값이 있는지 찾고 linkedlist로 넘어가면서 끝까지 검색한다. 만약 일치하는 것이 있다면 그 명령어의 opcode를 print해주고 return 0을 하였고 없다면 NONE을 출력하고 return 1을 했다.

10) check

입력된 char이 16진수의 범위인지 검사하는 함수이다. 범위를 벗어난다면 return -1이고 범위 안이라면 return 0을 했다.

11) dump\_

가상의 메모리 공간을 입력된 주소번지만큼 출력하는 함수이다. 각각 start는 start\_a와 start\_b로 end는 end\_a와 end\_b로 16으로 나눈 몫과 나머지로 설정하여 memory배열의 몇번째 줄, 칸인지 구했다. 그리고 if문을 통해 각각의 케이스에 따라 print를 해주었다. 먼저 start\_a와 end\_a가 같은 경우일 경우 같은 줄에 있기 때문에 먼저 그 줄의 address를 16진수로 print한뒤 start\_b부터 end\_b까지의 memory 값만 16진수로 출력하고 나머지 칸은 공백으로 놔두었다. ASCII 코드도 start\_b와 end\_b사이의 값만 print하고 나머지는 .으로 했고 그 사이더라도 20~7E까지의 값만 print하고 이외의 값은 .으로 출력하였다. 다음 start\_a와 end\_a가 다른 경우는 start\_a번째줄은 start\_a전까지는 공백, 이후는 16진수로 출력하였고 end\_a번째줄은 그전까지를 16진수로 출력하고 이후는 공백으로 출력하였다. 그사이의 줄들은 모두 16진수로 출력해주었다. ASCII코드도 위와 동일하게 각각 똑같은 범위에서 출력하였다.

12) edit\_

주어진 주소번지의 메모리를 value값으로 설정하는 함수이다. Start\_a와 start\_b를 16으로 나눈 몫과 나머지로 설정하여 memory[start\_a][start\_b]번째를 value값으로 바꾸어 주었다.

13) fill\_

start와 end주소번지 사이의 모든 memory를 value값으로 설정하는 함수이다. 똑같이 start\_a, start\_b, end\_a, end\_b를 설정한 다음 그 사이의 memory값들을 위의 함수들과 비슷하게 for, if문을 사용하여 value값들로 모두 설정해주었다.

14) dirent\_

현재 디렉터리에 있는 파일을 출력하는 함수이다. dirent.h와 sys/stat.h를 참조하였다. st.st\_mode & S\_IFMT 값을 switch문을 통해 구분하였고 S\_IFDIR이면 디렉터리이므로 ‘/’표시를 해주었다. 그리고 S\_IXUSR, S\_IXGRP, S\_IXOTH의 값이 모두 있다면 실행파일이므로 ‘\*’표시를 해주었다. 나머지 파일들은 파일명만 출력해주었다.

15) addlist

History 명령어가 입력되었을 때 이때까지의 입력된 올바른 명령어들을 순서대로 출력하기 위해 명령어가 입력될때마다 linkedlist에 추가해주는 함수이다. node를 new로 새로 설정한다음 new의 data로 input값을 넣고 next는 NULL로 설정한다. head가 비어있다면 head를 new로 설정하고 만약 비어있지 않다면 linkedlist의 끝까지 간 다음 그 끝의 next를 new로 연결해주었다. 그리고 순서를 세기 위해 linkedlist의 다음 node로 넘어갈 때마다 i를 하나씩 더해주어 node의 num에 입력하여 node의 순서를 설정해주었다.

16) printlist

history가 입력되었을 때 linkedlist에 있는 모든 명령어들을 출력해주는 함수이다. temp를 head로 설정하여 linkedlist의 node를 하나씩 연결로 넘어가면서 node의 num과 data를 출력해주고 다음 next가 NULL이라면 출력을 끝냈다.