

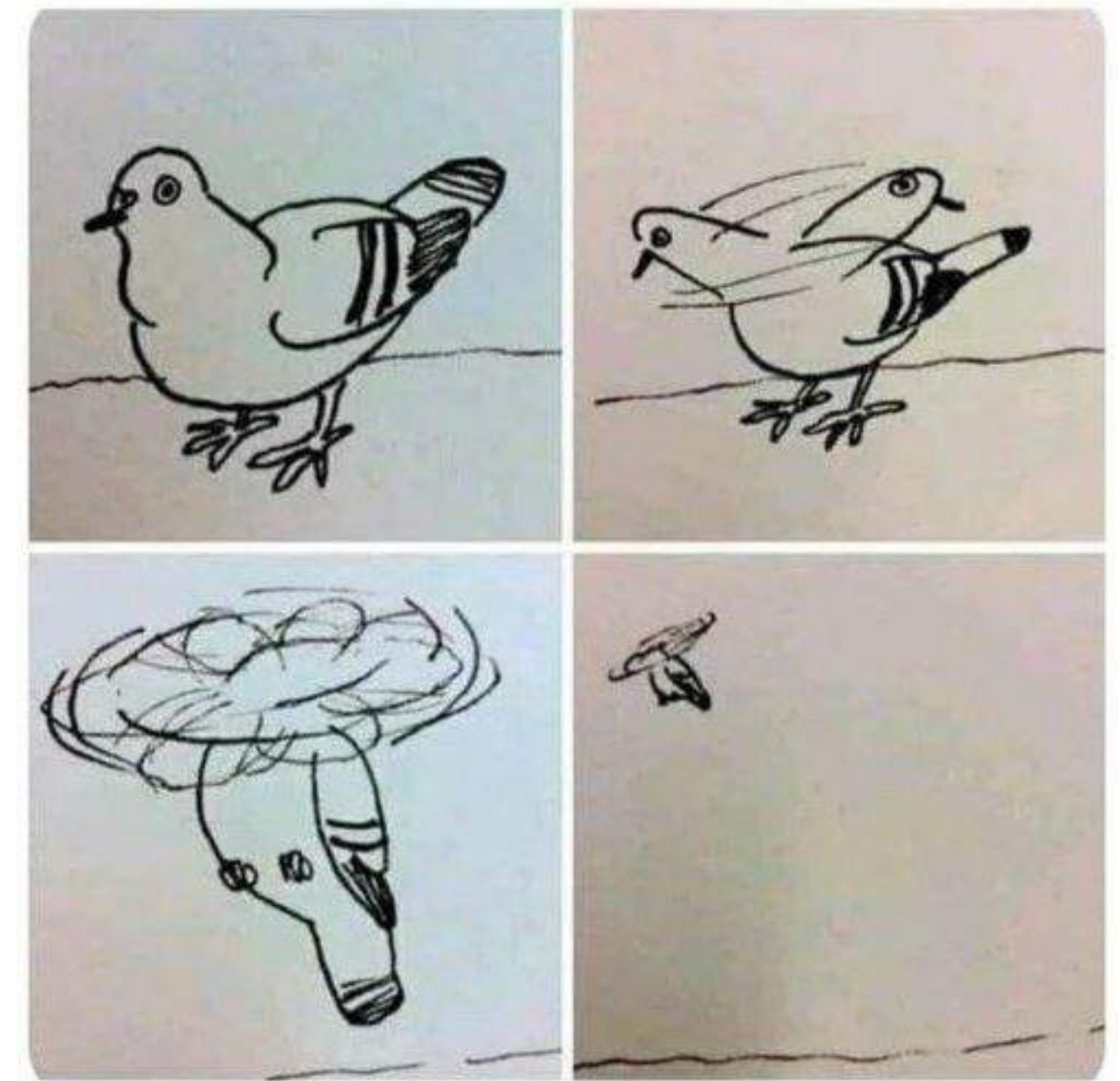
시프 파싱 과제6 해설

2024.04.11

정예녹

! 경고 !

- ‘아 이런 구조로 코드를 작성하기를 의도했구나’ 를 설명하기 위한 ppt입니다.
- ‘이런 구조로 안 짤으면 감점이구나’ 가 아닙니다!!!!
- 코드를 이상하게 짰더라도 동작만 하면 장땡입니다.



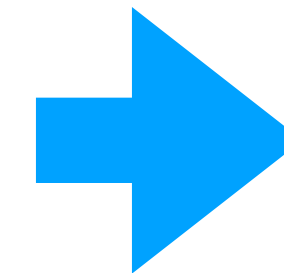
flying 비둘기
출처: 인터넷 어딘가

과제의 목표

- input 소스코드를 파싱하고, operator에 해당하는 opcode를 inst_table을 참고하여 출력하기
- input.txt 및 inst_table.txt를 입력으로 받고, output_opcode.txt를 생성하기

```
input.txt
COPY START 0 COPY FILE FROM IN TO OUTPUT
EXTDEF BUFFER, BUFEND, LENGTH
EXTREF RDREC, WRREC
FIRST STL RETADR SAVE RETURN ADDRESS
CLOOP +JSUB RDREC READ INPUT RECORD
LDA LENGTH TEST FOR EOF (LENGTH = 0)
COMP #0
JEQ ENDFIL EXIT IF EOF FOUND
+JSUB WRREC WRITE OUTPUT RECORD
J CLOOP LOOP
ENDFIL LDA =C'EOF' INSERT END OF FILE MARKER
STA BUFFER
LDA #3 SET LENGTH = 3
STA LENGTH
+JSUB WRREC WRITE EOF
J @RETADR RETURN TO CALLER
RETADR RESW 1
LENGTH RESW 1 LENGTH OF RECORD
LTORG
BUFFER RESB 4096 4096-BYTE BUFFER AREA
BUFEND EQU *
MAXLEN EQU BUFEND-BUFFER MAXIMUM RECORD LENGTH
RDREC CSECT
.
. SUBROUTINE TO READ RECORD INTO BUFFER
.
EXTREF BUFFER, LENGTH, BUFEND
CLEAR X CLEAR LOOP COUNTER
CLEAR A CLEAR A TO ZERO
CLEAR S CLEAR S TO ZERO
```

```
inst_table_tmp.txt
M 3 18
3 58
2 90
M 3 40
2 B4
3 28
3 88
2 A0
M 3 24
3 64
2 9C
M 0 0E
M 0 09
1 C0
M 3 3C
M 3 30
M 3 34
M 3 38
3 48
M 3 00
M 3 68
3 50
M 3 70
M 3 08
M 3 6C
M 3 74
M 3 04
M 2 00
```



```
output_opcode.txt
COPY START 0
EXTDEF BUFFER, BUFEND, LENGTH
EXTREF RDREC, WRREC
FIRST STL RETADR 0x14
CLOOP +JSUB RDREC 0x48
LDA LENGTH 0x00
COMP #0 0x28
JEQ ENDFIL 0x30
+JSUB WRREC 0x48
J CLOOP 0x3C
ENDFIL LDA =C'EOF' 0x00
STA BUFFER 0x0C
LDA #3 0x00
STA LENGTH 0x0C
+JSUB WRREC 0x48
J @RETADR 0x3C
RETADR RESW 1
LENGTH RESW 1
LTORG
BUFFER RESB 4096
BUFEND EQU *
MAXLEN EQU BUFEND-BUFFER
RDREC CSECT
.
. SUBROUTINE TO READ RECORD INTO BUFFER
.
EXTREF BUFFER, LENGTH, BUFEND
CLEAR X 0xB4
CLEAR A 0xB4
CLEAR S 0xB4
```

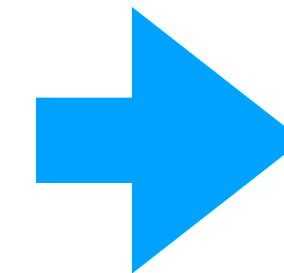

과제의 목표

- input 소스코드를 파싱하고, operator에 해당하는 opcode를 inst_table을 참고하여 출력하기
- input.txt 및 inst_table.txt를 입력으로 받고, output_opcode.txt를 생성하기

```
input.txt
COPY START 0 COPY FILE FROM IN TO OUTPUT
EXTDEF BUFFER, BUFEND, LENGTH
EXTREF RDREC, WRREC
FIRST STL RETADR SAVE RETURN ADDRESS
CLOOP +JSUB RDREC READ INPUT RECORD
LDA LENGTH TEST FOR EOF (LENGTH = 0)
COMP #0
JEQ ENDFIL EXIT IF EOF FOUND
+JSUB WRREC WRITE OUTPUT RECORD
J CLOOP LOOP
ENDFIL LDA =C'EOF' INSERT END OF FILE MARKER
STA BUFFER
LDA #3 SET LENGTH = 3
STA LENGTH
+JSUB WRREC WRITE EOF
J @RETADR RETURN TO CALLER
RETADR RESW 1
LENGTH RESW 1 LENGTH OF RECORD
LTORG
BUFFER RESB 4096 4096-BYTE BUFFER AREA
BUFEND EQU *
MAXLEN EQU BUFEND-BUFFER MAXIMUM RECORD LENGTH
RDREC CSECT
.
. SUBROUTINE TO READ RECORD INTO BUFFER
.
EXTREF BUFFER, LENGTH, BUFEND
CLEAR X CLEAR LOOP COUNTER
CLEAR A CLEAR A TO ZERO
CLEAR S CLEAR S TO ZERO
```

이건 본인 마음대로 작성하심 됩니다.

```
inst_table_tmp.txt
M 3 18
3 58
2 90
M 3 40
2 B4
3 28
3 88
2 A0
M 3 24
3 64
2 9C
M 0 0E
M 0 09
1 C0
M 3 3C
M 3 30
M 3 34
M 3 38
3 48
M 3 00
M 3 68
3 50
M 3 70
M 3 08
M 3 6C
M 3 74
M 3 04
M 2 00
```



```
output_opcode.txt
COPY START 0
EXTDEF BUFFER, BUFEND, LENGTH
EXTREF RDREC, WRREC
FIRST STL RETADR 0x14
CLOOP +JSUB RDREC 0x48
LDA LENGTH 0x00
COMP #0 0x28
JEQ ENDFIL 0x30
+JSUB WRREC 0x48
J CLOOP 0x3C
ENDFIL LDA =C'EOF' 0x00
STA BUFFER 0x0C
LDA #3 0x00
STA LENGTH 0x0C
+JSUB WRREC 0x48
J @RETADR 0x3C
RETADR RESW 1
LENGTH RESW 1
LTORG
BUFFER RESB 4096
BUFEND EQU *
MAXLEN EQU BUFEND-BUFFER
RDREC CSECT
.
. SUBROUTINE TO READ RECORD INTO BUFFER
.
EXTREF BUFFER, LENGTH, BUFEND
CLEAR X 0xB4
CLEAR A 0xB4
CLEAR S 0xB4
```

프로그램 구조

- main 함수를 보면 알 수 있습니다.

```
int main(int argc, char **argv) {
    /** SIC/XE 머신의 instruction 정보를 저장하는 테이블 */
    inst *inst_table[MAX_INST_TABLE_LENGTH];
    int inst_table_length;

    /** SIC/XE 소스코드를 저장하는 테이블 */
    char *input[MAX_INPUT_LINES];
    int input_length;

    /** 소스코드의 각 라인을 토큰 전환하여 저장하는 테이블 */
    token *tokens[MAX_INPUT_LINES];
    int tokens_length;

    init_inst_table(
        inst_table, &inst_table_length,
        "inst_table.txt"
    );
    init_input(
        input, &input_length,
        "input.txt"
    );
    assem_pass1(
        (const inst **)inst_table, inst_table_length,
        (const char **)input, input_length,
        tokens, &tokens_length,
        ...
    );
    make_opcode_output(
        "output_opcode.txt",
        (const token **)tokens, tokens_length,
        (const inst **)inst_table, inst_table_length);

    return 0;
}
```

프로그램 구조

- main 함수를 보면 알 수 있습니다.
- inst_table 초기화하고,

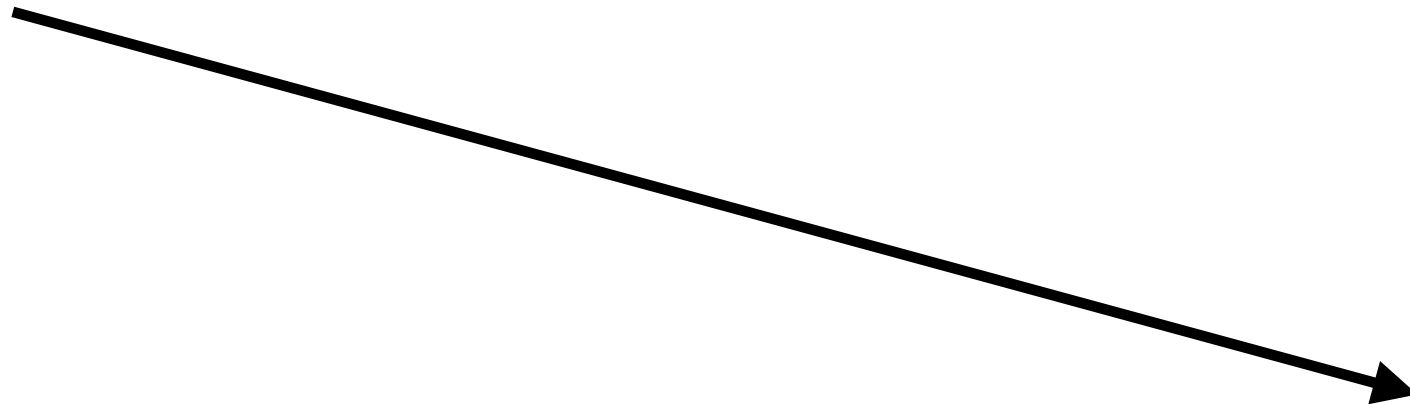
```
int main(int argc, char **argv) {
    /** SIC/XE 머신의 instruction 정보를 저장하는 테이블 */
    inst *inst_table[MAX_INST_TABLE_LENGTH];
    int inst_table_length;

    /** SIC/XE 소스코드를 저장하는 테이블 */
    char *input[MAX_INPUT_LINES];
    int input_length;

    /** 소스코드의 각 라인을 토큰 전환하여 저장하는 테이블 */
    token *tokens[MAX_INPUT_LINES];
    int tokens_length;

    init_inst_table(
        inst_table, &inst_table_length,
        "inst_table.txt"
    );
    init_input(
        input, &input_length,
        "input.txt"
    );
    assem_pass1(
        (const inst **)inst_table, inst_table_length,
        (const char **)input, input_length,
        tokens, &tokens_length,
        ...
    );
    make_opcode_output(
        "output_opcode.txt",
        (const token **)tokens, tokens_length,
        (const inst **)inst_table, inst_table_length);

    return 0;
}
```



프로그램 구조

- main 함수를 보면 알 수 있습니다.

- inst_table 초기화하고,

- input 입력받고,

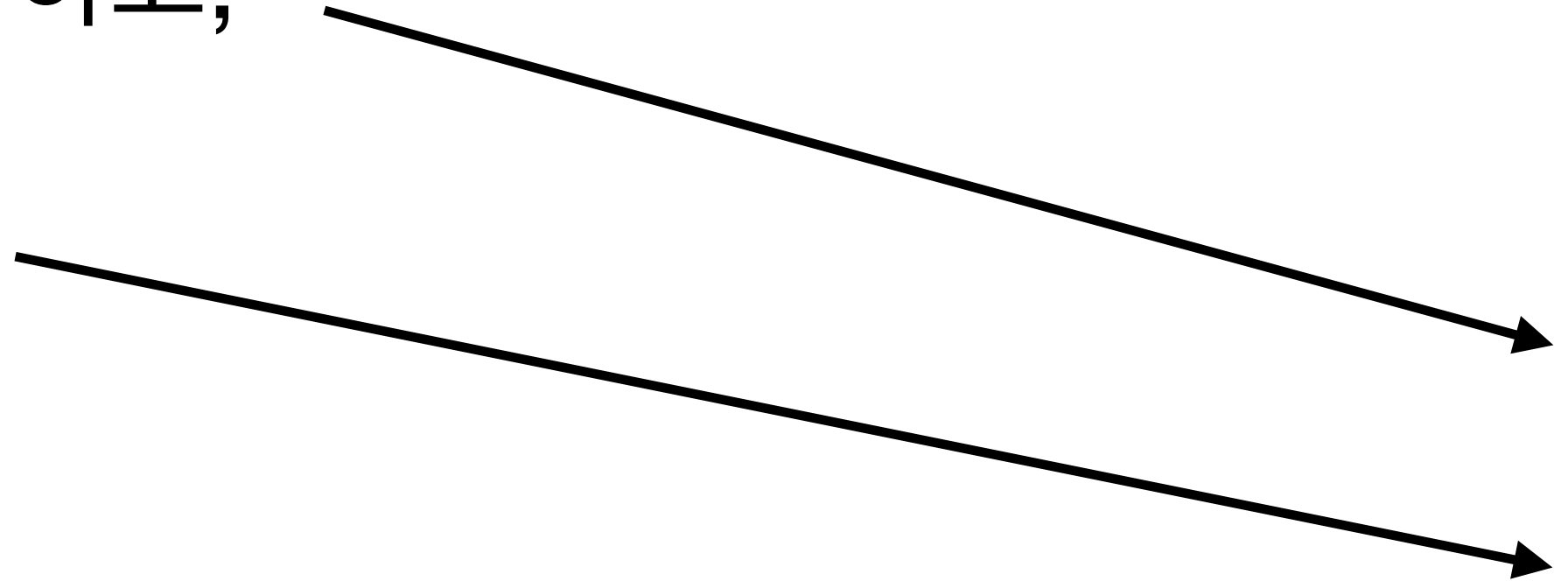
```
int main(int argc, char **argv) {
    /** SIC/XE 머신의 instruction 정보를 저장하는 테이블 */
    inst *inst_table[MAX_INST_TABLE_LENGTH];
    int inst_table_length;

    /** SIC/XE 소스코드를 저장하는 테이블 */
    char *input[MAX_INPUT_LINES];
    int input_length;

    /** 소스코드의 각 라인을 토큰 전환하여 저장하는 테이블 */
    token *tokens[MAX_INPUT_LINES];
    int tokens_length;

    init_inst_table(
        inst_table, &inst_table_length,
        "inst_table.txt"
    );
    init_input(
        input, &input_length,
        "input.txt"
    );
    assem_pass1(
        (const inst **)inst_table, inst_table_length,
        (const char **)input, input_length,
        tokens, &tokens_length,
        ...
    );
    make_opcode_output(
        "output_opcode.txt",
        (const token **)tokens, tokens_length,
        (const inst **)inst_table, inst_table_length);

    return 0;
}
```



프로그램 구조

- main 함수를 보면 알 수 있습니다.

- inst_table 초기화하고,

- input 입력받고,

- input 내용을 tokens로 파싱하고,

```
int main(int argc, char **argv) {
    /** SIC/XE 머신의 instruction 정보를 저장하는 테이블 */
    inst *inst_table[MAX_INST_TABLE_LENGTH];
    int inst_table_length;

    /** SIC/XE 소스코드를 저장하는 테이블 */
    char *input[MAX_INPUT_LINES];
    int input_length;

    /** 소스코드의 각 라인을 토큰 전환하여 저장하는 테이블 */
    token *tokens[MAX_INPUT_LINES];
    int tokens_length;

    init_inst_table(
        inst_table, &inst_table_length,
        "inst_table.txt"
    );
    init_input(
        input, &input_length,
        "input.txt"
    );
    assem_pass1(
        (const inst **)inst_table, inst_table_length,
        (const char **)input, input_length,
        tokens, &tokens_length,
        ...
    );
    make_opcode_output(
        "output_opcode.txt",
        (const token **)tokens, tokens_length,
        (const inst **)inst_table, inst_table_length);

    return 0;
}
```


프로그램 구조

- main 함수를 보면 알 수 있습니다.

- inst_table 초기화하고,
- input 입력받고,
- input 내용을 tokens로 파싱하고,
- tokens와 inst_table을 참고해서 opcode를 출력한다.

```
int main(int argc, char **argv) {
    /** SIC/XE 머신의 instruction 정보를 저장하는 테이블 */
    inst *inst_table[MAX_INST_TABLE_LENGTH];
    int inst_table_length;

    /** SIC/XE 소스코드를 저장하는 테이블 */
    char *input[MAX_INPUT_LINES];
    int input_length;

    /** 소스코드의 각 라인을 토큰 전환하여 저장하는 테이블 */
    token *tokens[MAX_INPUT_LINES];
    int tokens_length;

    init_inst_table(
        inst_table, &inst_table_length,
        "inst_table.txt"
    );
    init_input(
        input, &input_length,
        "input.txt"
    );
    assem_pass1(
        (const inst **)inst_table, inst_table_length,
        (const char **)input, input_length,
        tokens, &tokens_length,
        ...
    );
    make_opcode_output(
        "output_opcode.txt",
        (const token **)tokens, tokens_length,
        (const inst **)inst_table, inst_table_length);

    return 0;
}
```

테이블 구조 !important

- 이것도 main 함수를 보고 알 수 있습니다.

```
inst *inst_table[MAX_INST_TABLE_LENGTH];  
int inst_table_length;
```

```
/** SIC/XE 소스코드를 저장하는 테이블 */  
char *input[MAX_INPUT_LINES];  
int input_length;
```

```
/** 소스코드의 각 라인을 토큰 전환하여 저장하는 테이블 */  
token *tokens[MAX_INPUT_LINES];  
int tokens_length;
```

테이블 구조 !important

- 이것도 main 함수를 보고 알 수 있습니다.
- `inst *inst_table[MAX_INST_TABLE_LENGTH];`
- 포인터 배열? 배열 포인터?

```
inst *inst_table[MAX_INST_TABLE_LENGTH];  
int inst_table_length;
```

```
/** SIC/XE 소스코드를 저장하는 테이블 */  
char *input[MAX_INPUT_LINES];  
int input_length;
```

```
/** 소스코드의 각 라인을 토큰 전환하여 저장하는 테이블 */  
token *tokens[MAX_INPUT_LINES];  
int tokens_length;
```

테이블 구조 !important

- 이것도 main 함수를 보고 알 수 있습니다.
- `inst *inst_table[MAX_INST_TABLE_LENGTH];`
- 포인터 배열? 배열 포인터?
- [inst를 가리키는 포인터]를 저장하는 배열!

```
inst *inst_table[MAX_INST_TABLE_LENGTH];  
int inst_table_length;
```

```
/** SIC/XE 소스코드를 저장하는 테이블 */  
char *input[MAX_INPUT_LINES];  
int input_length;
```

```
/** 소스코드의 각 라인을 토큰 전환하여 저장하는 테이블 */  
token *tokens[MAX_INPUT_LINES];  
int tokens_length;
```

테이블 구조 !important

- 이것도 main 함수를 보고 알 수 있습니다.

- `inst *inst_table[MAX_INST_TABLE_LENGTH];`

- 포인터 배열? 배열 포인터?

- [inst를 가리키는 포인터]를 저장하는 배열!

```
inst *inst_table[MAX_INST_TABLE_LENGTH];  
int inst_table_length;
```

- `int inst_table_length;`

```
/** SIC/XE 소스코드를 저장하는 테이블 */  
char *input[MAX_INPUT_LINES];  
int input_length;
```

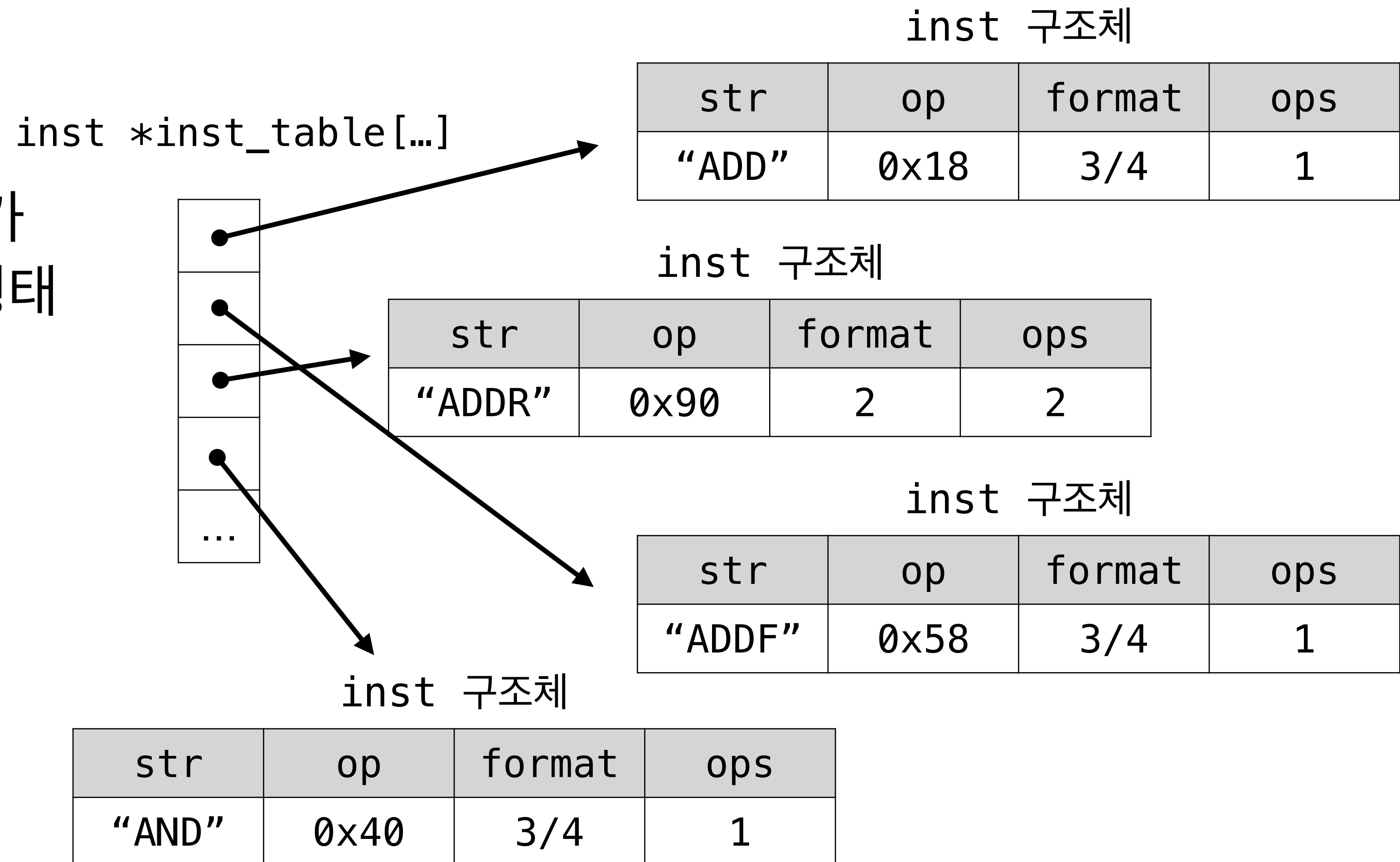
- 배열의 길이!

```
/** 소스코드의 각 라인을 토큰 전환하여 저장하는 테이블 */  
token *tokens[MAX_INPUT_LINES];  
int tokens_length;
```


테이블 구조 !important

- 다르게 말하자면...

- 그림과 같이, 배열의 요소가 구조체 하나를 가리키는 형태입니다.

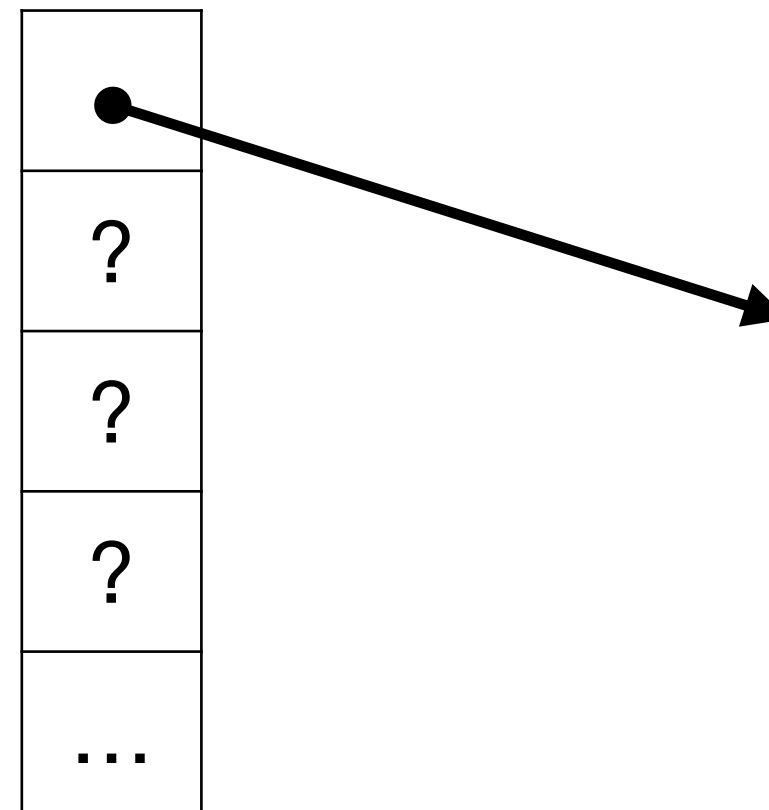


테이블 구조 !important

- 다르게 말하자면...

- 이런 형태가 아니에요!

`inst *inst_table[...]`



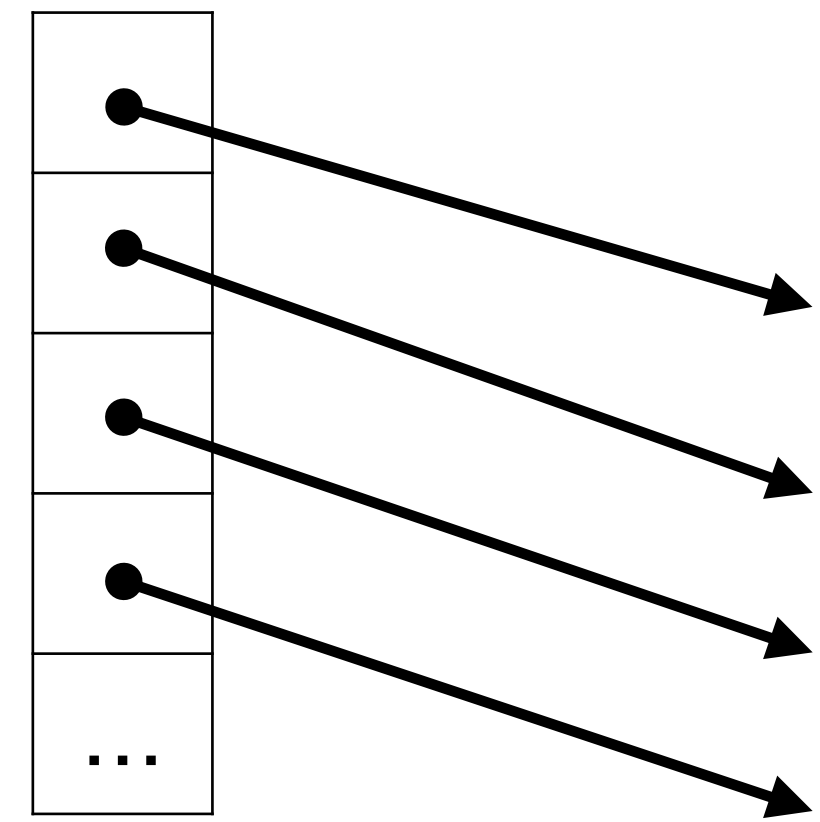
inst 구조체 배열?

str	op	format	ops
"ADD"	0x18	3/4	1
"ADDF"	0x58	3/4	1
"ADDR"	0x90	2	2
"AND"	0x40	3/4	1
...

테이블 구조 !important

- 다르게 말하자면...
- 이걸 의도한 것도 아니고요! `inst *inst_table[...]`

inst 구조체 배열



str	op	format	ops
"ADD"	0x18	3/4	1
"ADDF"	0x58	3/4	1
"ADDR"	0x90	2	2
"AND"	0x40	3/4	1
...

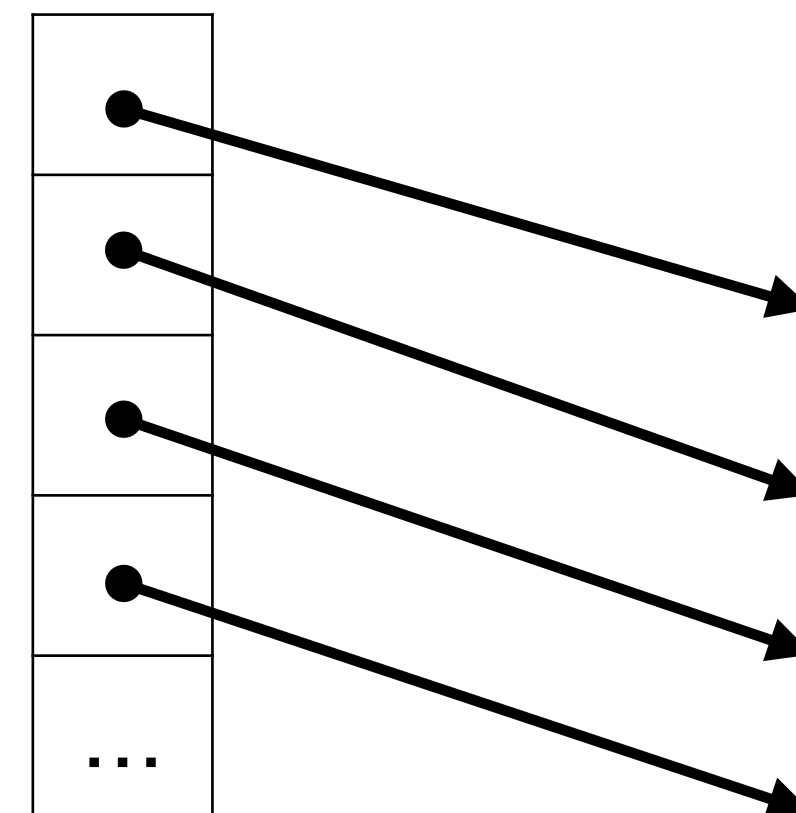
테이블 구조 !important

- 다르게 말하자면...

- 이걸 의도한 것도 아니고요! `inst *inst_table[...]`

- 다시 말하지만, 구현이 어떻게
간에 동작만 하면 장땡입니다.

- 다만 버그는 버그를 낳는 법
이기에, 본인의 비둘기가 이
상하게 날고 있다면 수정을
권장합니다.



inst 구조체 배열

str	op	format	ops
"ADD"	0x18	3/4	1
"ADDF"	0x58	3/4	1
"ADDR"	0x90	2	2
"AND"	0x40	3/4	1
...

테이블 구조 !important

- Q. 근데 테이블 자료형이 이중 포인터 아니었나요?

```
int init_inst_table(inst **inst_table, ...)
```


테이블 구조 !important

- Q. 근데 테이블 자료형이 이중 포인터 아니었나요?
- C언어 특징인데, 배열이 함수 인자로 주어질 때는 포인터 형태로 전달됩니다. 그렇기에, 함수 매개변수에서는 배열과 포인터를 구분하지 않습니다.
- 1학년 때 배우는 내용이라 기억이 가물가물하실 것 같아서 복습 권장 차원에서 넣었습니다. 헛갈리라고 일부러 함수 매개변수를 이중포인터로 작성했는데, 이것 때문에 헤매시는 모습을 보니 뿌듯하네요.

```
int init_inst_table(inst **inst_table, ...)  
=  
int init_inst_table(inst *inst_table[], ...)
```

각 함수 구현

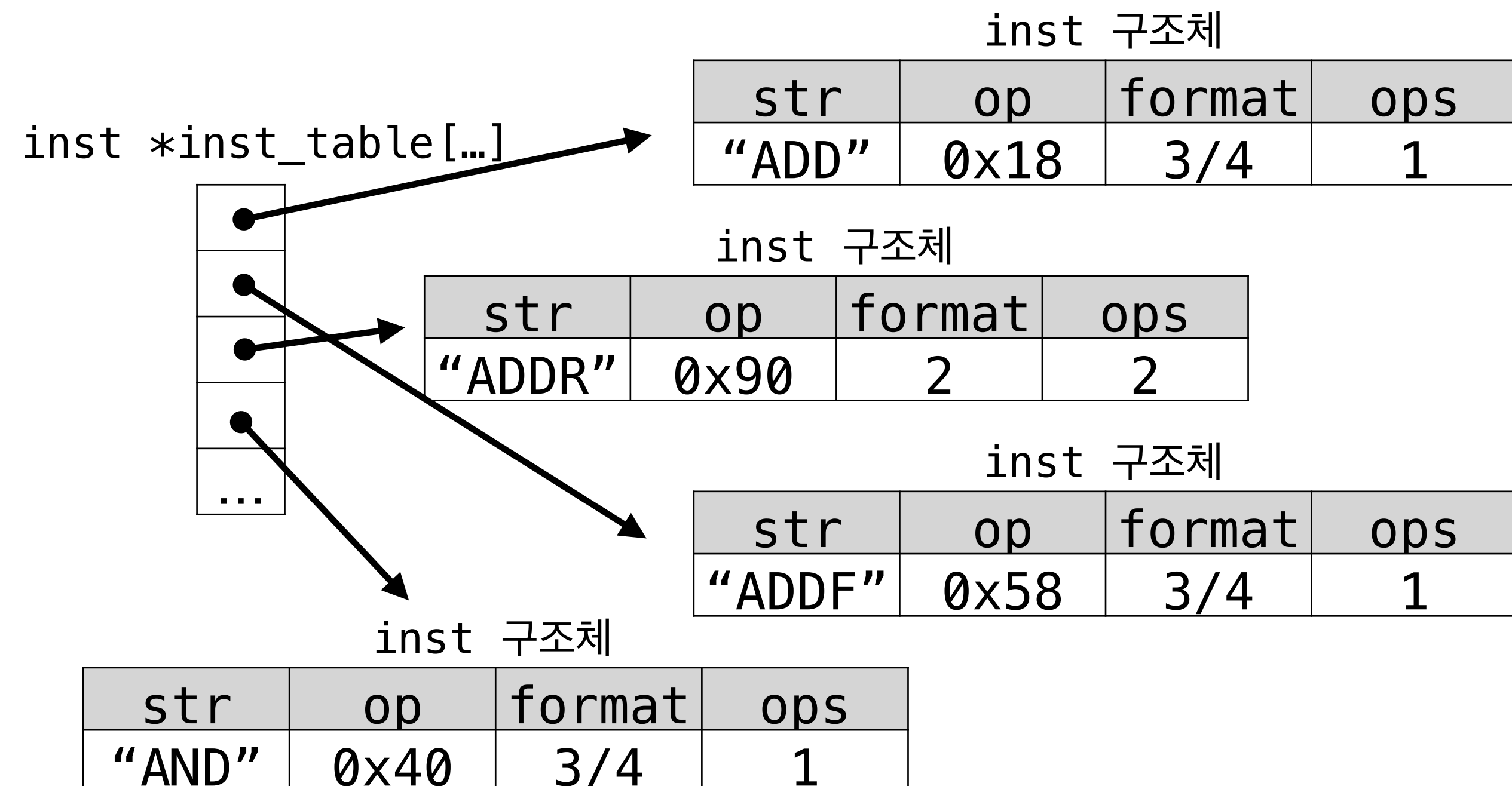
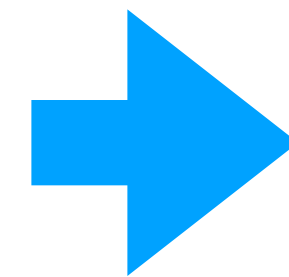
- 과제 6에서 구현해야 했던 함수는 다음과 같습니다 :
 - init_inst_table
 - init_input
 - assem_pass1
 - token_parsing
 - search_opcode
 - make_opcode_output
- 주석으로도 대략적으로 설명했지만, 각 함수가 어떤 작업을 수행해야 하는지 보여드리자면...

```
int init_inst_table(inst *inst_table[], int *inst_table_length,
                  const char *inst_table_dir);
```

- inst_table_dir 경로에 있는 파일을 읽고, 이걸로 inst_table 및 inst_table_length를 초기화.
- inst_table_length의 값은 본인의 inst_table.txt 구현에 따라 다름 (보통은 59)
- 여기서 inst 구조체를 할당하면 됩니다.

inst_table_tmp.txt

ADD		M	3	18
ADDF	M	3	58	
ADDR	RR	2	90	
AND		M	3	40
CLEAR	R	2	B4	
COMP	M	3	28	
COMPF	M	3	88	
COMPR	RR	2	A0	
DIV		M	3	24
DIVF	M	3	64	
DIVR	RR	2	9C	
END		M	0	0E
EQU		M	0	09
FLOAT	-	1	C0	
J		M	3	3C
JEQ		M	3	30
JGT		M	3	34
JLT		M	3	38
JSUB	M	3	48	
LDA		M	3	00
LDB		M	3	68
LDCH	M	3	50	
LDF		M	3	70
LDL		M	3	08
LDS		M	3	6C
LDT		M	3	74
LDX		M	3	04
LPS		M	3	00



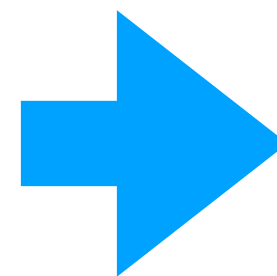
int init_input(char *input[], int *input_length, const char *input_dir)

- input_dir 경로에 있는 파일을 읽고, 이걸로 input 및 input_length를 초기화.
- input_length = 58
- 소스코드 한 줄을 위한 메모리 할당은 여기서 하시면 됩니다.

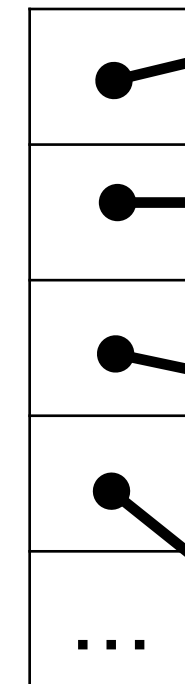
```
COPY    START    0      COPY FILE FROM IN TO OUTPUT
EXTDEF  BUFFER, BUFEND, LENGTH
EXTREF  RDREC, WRREC

FIRST   STL      RETADR  SAVE RETURN ADDRESS
CLOOP  +JSUB     RDREC   READ INPUT RECORD
        LDA      LENGTH  TEST FOR EOF (LENGTH = 0)
        COMP     #0
        JEQ      ENDFIL  EXIT IF EOF FOUND
        +JSUB     WRREC   WRITE OUTPUT RECORD
        J        CLOOP   LOOP
ENDFIL  LDA      =C'EOF'  INSERT END OF FILE MARKER
        STA      BUFFER
        LDA      #3      SET LENGTH = 3
        STA      LENGTH
        +JSUB     WRREC   WRITE EOF
        J        @RETADR  RETURN TO CALLER
RETADR  RESW      1
LENGTH  RESW      1      LENGTH OF RECORD
        LTORG
BUFFER  RESB      4096    4096-BYTE BUFFER AREA
BUFEND  EQU       *
MAXLEN  EQU      BUFEND-BUFFER  MAXIMUM RECORD LENGTH
RDREC   CSECT

.
.      SUBROUTINE TO READ RECORD INTO BUFFER
.
EXTREF  BUFFER, LENGTH, BUFEND
CLEAR  X      CLEAR LOOP COUNTER
CLEAR  A      CLEAR A TO ZERO
CLEAR  S      CLEAR S TO ZERO
```



char *input[...]



COPY\tSTART\t0\tCOPY FILE FROM IN TO OUTPUT\0

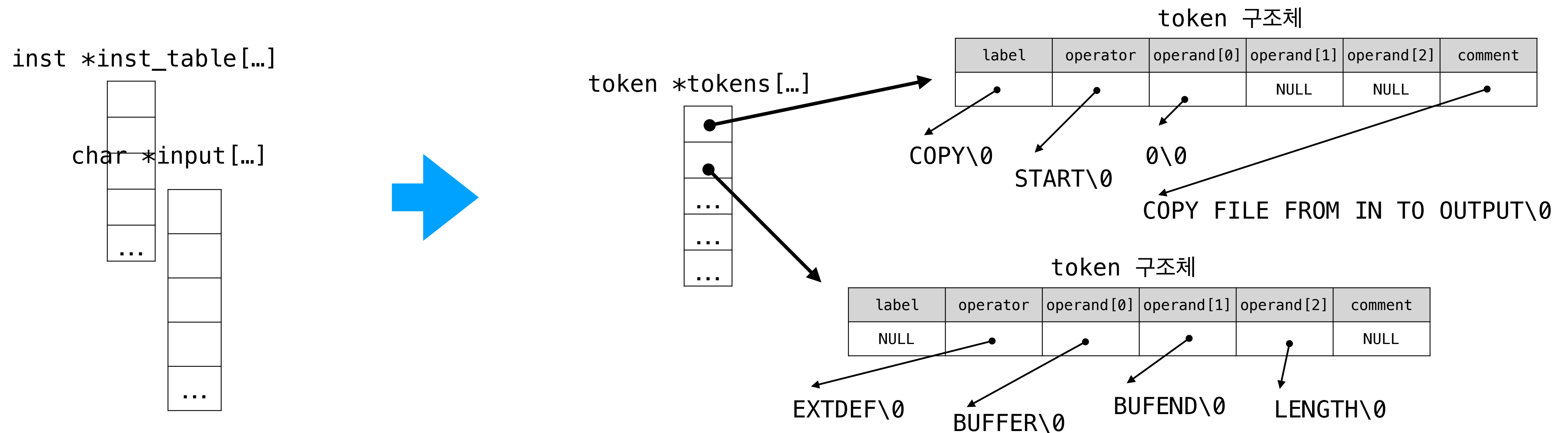
\tEXTDEF\tBUFFER, BUFEND, LENGTH\0

\tEXTREF\tRDREC, WRREC\0

FIRST\tSTL\tRETADR\tSAVE RETURN ADDRESS\0

```
int assem_pass1(const inst *inst_table[], int inst_table_length,
               const char *input[], int input_length,
               token *tokens[], int *tokens_length,
               ...);
```

- inst_table 및 input(소스 코드 여러 줄)을 참고하여, tokens 및 tokens_length를 초기화 (symbol table 및 literal table 작업은 과제6에서 수행하지 않음)
- tokens_length의 값은 본인 구현에 따라 다름 (보통은 52 혹은 58)
- token 구조체 할당은 여기서 하세요! token_parsing에서 하는 것은 크게 의미가 없습니다.



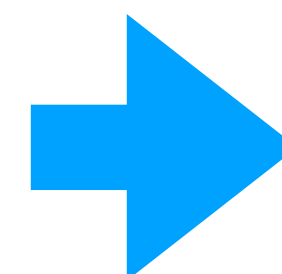

```
int token_parsing(const char *input, token *tok);
```

- input(소스 코드 한 줄)을 파싱하여 tok에 저장
- 이 input은 assem_pass1의 input과는 다름!!
 - 이 input = char* = 소스코드 한 줄을 가리키는 포인터
 - assem_pass1의 input = char*[] = 소스코드 한 줄을 가리키는 포인터 배열 = 소스코드 여러 줄
- 변수명이 똑같아서 오용하다 버그 일으키기 좋게 생겼네요. 이것 땀에 디버깅에만 하루 날리신 분이 계십니다. 의도는 아니었어요 ππ

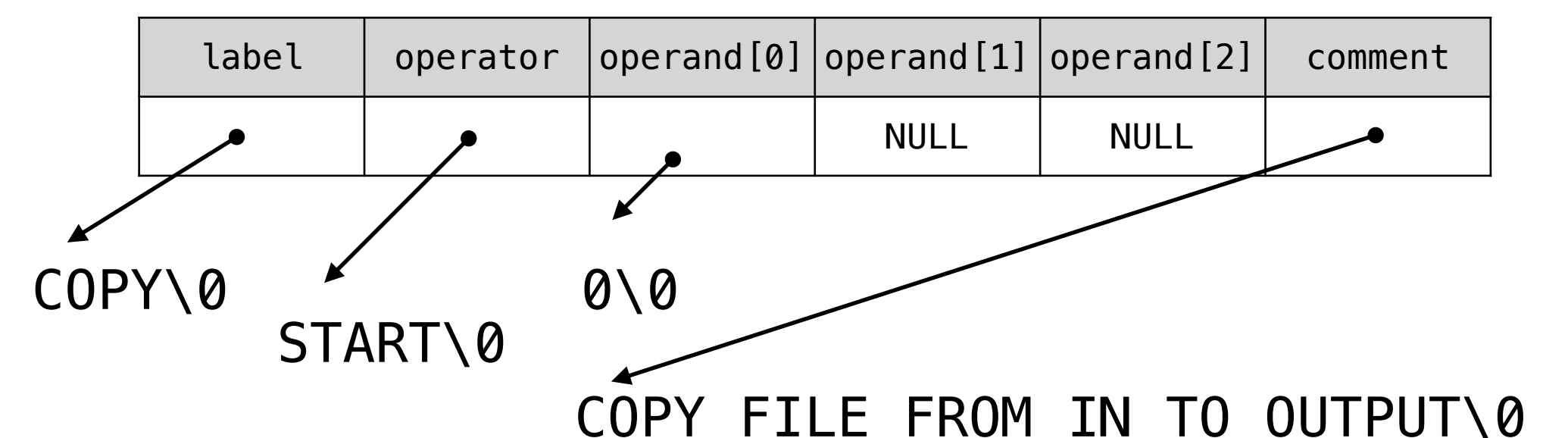
COPY\tSTART\t0\tCOPY FILE FROM IN TO OUTPUT\0

token 구조체

label	operator	operand[0]	operand[1]	operand[2]	comment
?	?	?	?	?	?



token 구조체



```
int search_opcode(const char *str,  
                  const inst *inst_table[], int inst_table_length);
```

- inst_table에서 str과 같은 이름을 가진 instruction을 찾고, 해당 instruction이 위치한 인덱스를 반환. 이름이 str인 instruction이 없으면 -1을 반환.
- str로 “+JSUB”같은 문자열이 주어질 경우에 대한 처리는 본인 자유 (implementation-defined behavior)
- assem_pass2에서도 자주 사용하시게 될 겁니다.

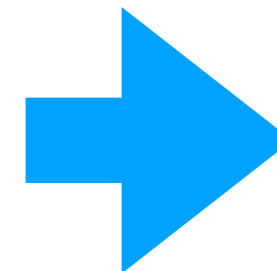
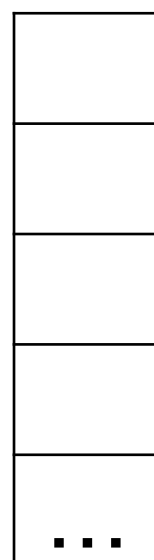
```
int make_opcode_output(const char *output_dir,
                      const token *tokens[], int tokens_length,
                      const inst *inst_table[], int inst_table_length);
```

- tokens에 저장된 정보를 output_dir에 출력하되, inst_table을 참고하여 opcode도 같이 출력.
- 크게 설명할 게 없네요.. tokens 파싱을 잘 수행했었다면 make_opcode_output 함수 구현에서는 크게 어려움이 없었을 겁니다.

inst *inst_table[...]



token *tokens[...]



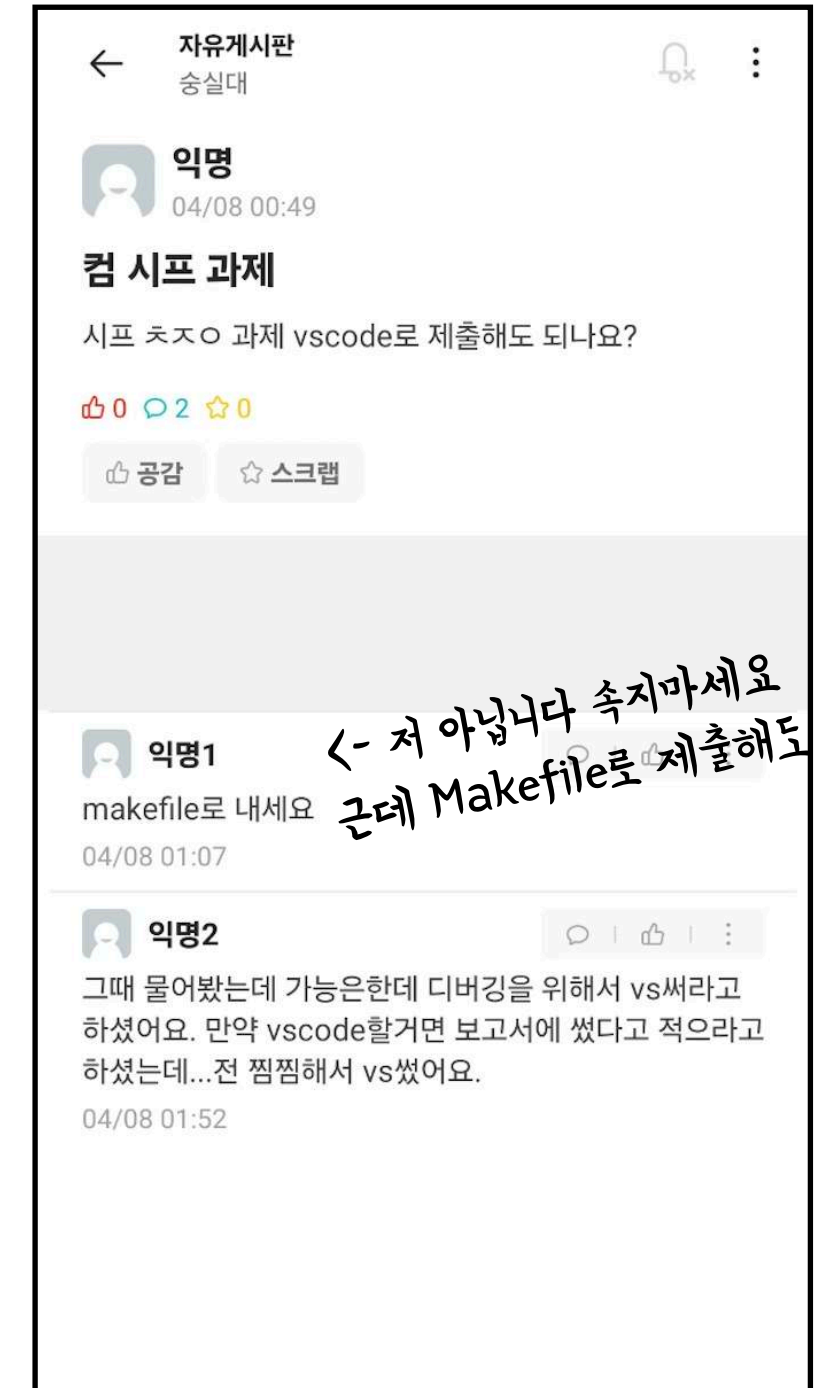
```
output_opcode.txt
COPY      START      0
          EXTDEF     BUFFER, BUFEND, LENGTH
          EXTREF     RDREC, WRREC
FIRST     STL        RETADR          0x14
CLOOP     +JSUB      RDREC           0x48
          LDA        LENGTH          0x00
          COMP       #0              0x28
          JEQ        ENDFIL          0x30
          +JSUB      WRREC           0x48
          J          CLOOP           0x3C
ENDFIL    LDA        =C'EOF'         0x00
          STA        BUFFER          0x0C
          LDA        #3              0x00
          STA        LENGTH          0x0C
          +JSUB      WRREC           0x48
          J          @RETADR         0x3C
RETADR    RESW       1
LENGTH    RESW       1
          LTORG
BUFFER    RESB       4096
BUFEND    EQU        *
MAXLEN    EQU        BUFEND-BUFFER
RDREC     CSECT
.
.         SUBROUTINE TO READ RECORD INTO BUFFER
.
          EXTREF     BUFFER, LENGTH, BUFEND
          CLEAR      X              0xB4
          CLEAR      A              0xB4
          CLEAR      S              0xB4
```

마치며...

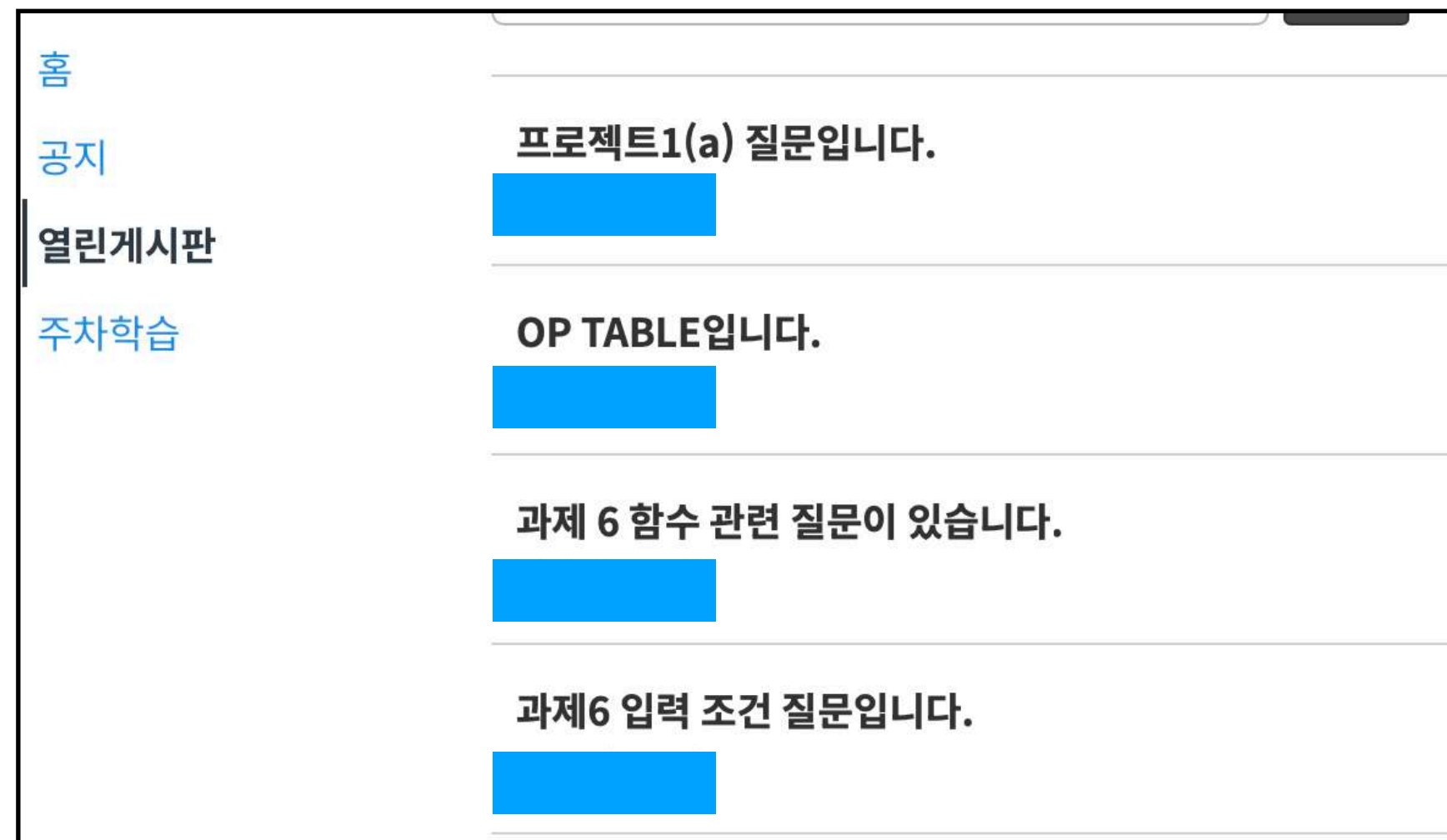
- strdup랑 strtok를 멀리해주세요
 - 문자열 관련 함수 주제에 문자열 값을 마음대로 변경하거나, 메모리 할당을 지멋대로 하는 놈들입니다 => 버그를 일으키거나 메모리 누수를 일으키기 좋은 함수입니다.
 - ANSI C standard에도 없는 근본없는 함수이니, 부담없이 혐오하셔도 됩니다.

마치며...

- 질문해주세요 ㅠ ㅠ
- 일반적인 질문(과제 명세에 관한 질문 등)은 열린게시판에 올려주시고, 개인적인 질문(제 코드가 이상해요 등)은 407호로 오시거나 메일 (enochjung@soongsil.ac.kr) 보내주세요
- 407호 오실 때는 미리 메일로 귀뜸해주세요.



잘못된 예시



좋은 예시



슬픈 예시

끝