d프로젝트 1

20181259

조수민

출석번호 130

프로젝트1 (sic/xe머신 어셈블러 pass2 구현)

동기/목적

지난 과제에서 pass1을 구현했고 이번 과제에선 pass2를 구현해야 했습니다.

pass2에서는 pass1에서 계산한 location counter와 opcode를 기반으로 오브젝트 코드를 생성해야 했습니다.

지난 과제에서 수행하지 않았던 literal table과 symbol table을 pass1에서 초기화 하고 파일로 만들었습니다.

제공된 input은 control section 기반 파일이었습니다.

base register는 사용하지 않았고 pc기반 계산과 4형식 + modification record를 사용해서 오브젝트 코드를 생성했습니다.

설계 구현 아이디어

object_code안에 모든 오브젝트 코드 정보를 담고있어야 한다는 명세가 있었습니다.

때문에 main 프로그램과 서브 루틴으로 나누어서 초기화를 진행했습니다.

```
typedef struct _object_code {
    obj_code_data main_program;
    int end_addr;
    obj_code_data subroutine[MAX_SUBROUTINE_NUM];
    int subroutine_length;
} object_cod
```

obj_code_data라는 구조체는 오브젝트 코드에 필요한 정보를 가지고 있습니다.

obj_code_data 형식의 main 변수가 main 프로그램에 오브젝트 코드를 만들기 위한 정보를 담고 있고

서브루틴은 n개일 수 있기 때문에 obj_code_data배열을 사용합니다.

```
typedef struct _obj_code_data {
    header header_record;
    extref extref_table[MAX_EXTREF];
    extdef extdef_table[MAX_EXTDEF];
    int extdef_length;
    text_table text[MAX_TEXT_LENGTH];
    int text_length;
    char *literals[MAX_TABLE_LENGTH];
    modification_record modification_table[MAX_MODIFICATION_RECORD];
    int modification_table_length;
} obj_code_data;
```

각 구조체의 변수에 대한 설명입니다

- header : header에 필요한 정보. 시작 주소, 프로그램 길이 등

```
typedef struct _header {
    char *header_symbol;
    int start_address;
    int program_length;
} header;
```

- extref, extdef : reference하거나 define한 심볼들의 정보

```
typedef struct _extref {
    char *symbol;
    int addr;
} extref;

typedef struct _extdef {
    char *symbol;
    int addr;
} extdef;
```

- text_table : 변환된 오브젝트 코드(text), location, operator

```
typedef struct _text_table {
    unsigned int text;
    unsigned int loc;
    char *operator;
} text_table;
```

- modification_record : modification_record가 필요한 주소와 정보

```
typedef struct _modification_record {
    int addr;
    int modification_length;
    char *symbol;
} modification_record;
```

이를 통해 오브젝트 코드에 필요한 정보를 전부 담을 수 있었습니다.

주어진 input에는 고려할 점이 많았습니다. input이 다소 까다로웠기 때문에 관련해서 변수를 많이 사용했습니다.

1. x레지스터를 사용할 때 nixbpe 초기화

   ,를 구분자로 operand를 나누었고 두번째 인자가 x라면 nixbpe에서 x비트를 올려주었습니다

2. 심볼의 연산

   가장 까다로운 부분이었습니다. operand에 연산자가 존재한다면 그를 기준으로 symbol을 나누고 나누어진 symbol의 주소값을 value로 연산을 진행했습니다. 이는 modification record에도 명시해줘야 했습니다. 연산자는 1개 즉 symbol 2개까지만 연산 가능하게 구현했습니다.

3. 리터럴 할당

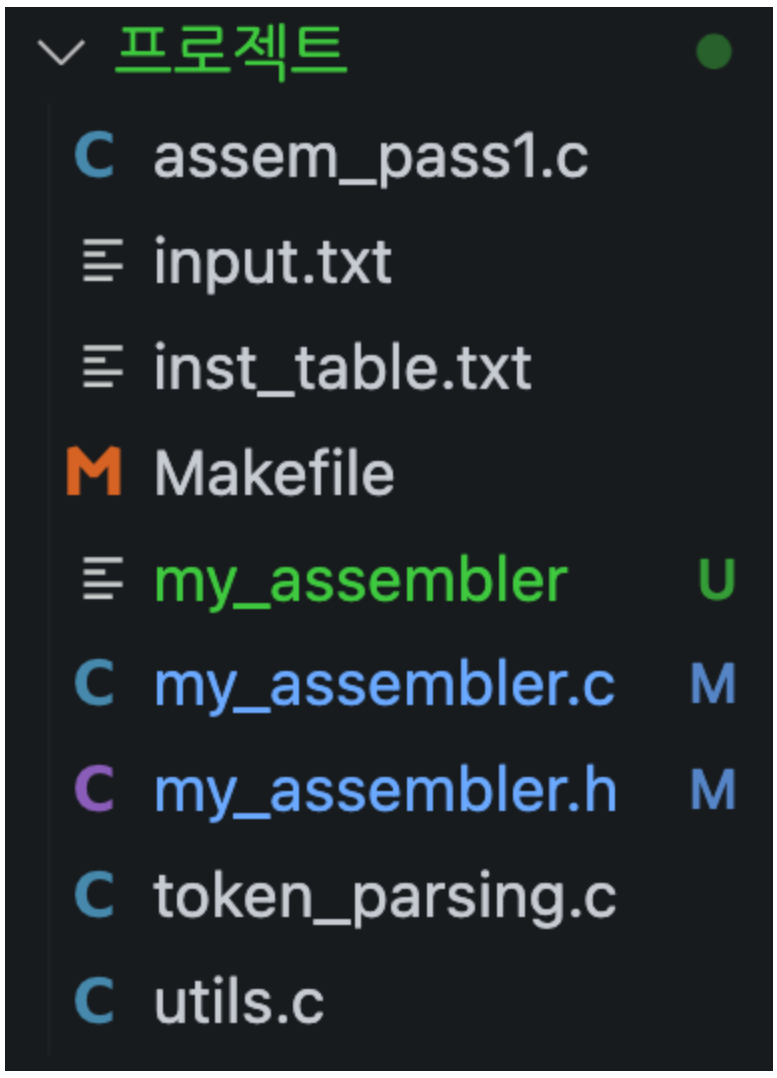
   리터럴을 만나면 리터럴 count를 올려주었고 ltorg를 만나면 리터럴 count만큼 리터럴을 할당해주었습니다. 리터럴이 존재하는데 ltorg가 없다면 end 이후에 해줘야 할당해줘야 했습니다. end를 만났을 때 리터럴 길이가 1 이상이라면 end 이후에 할당해줬습니다.

input 이 까다로워서 고려할 부분이 많았습니다. 밑에서 서술하겠지만, 사실 extref나 심볼 연산, 리터럴이 사용된 이상 고려할 사항이 정말 많을 것입니다.

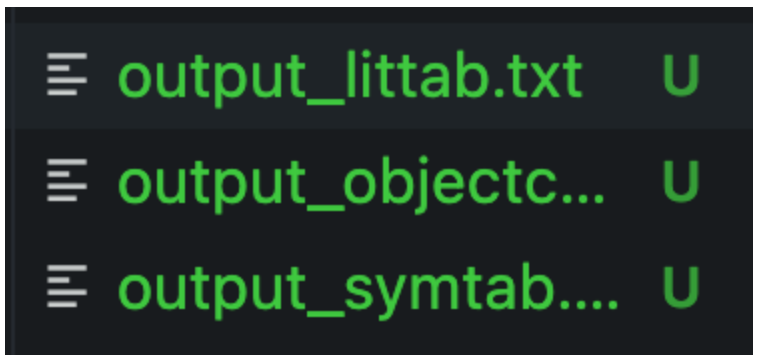
실행 결과

```
sumjo 🚀  프로젝트  main ±
 make
gcc -o my_assembler my_assembler.c token_parsing.c utils.c assem_pass1.c -I.
```

makefile 통해 빌드합니다



my_assembler라는 실행파일이 생성됩니다



실행 후 3개의 output 파일이 생성됩니다

심볼 테이블

output_symtab.txt

```
COPY    0000    +1  COPY
FIRST   0000    +1  COPY
CLOOP   0003    +1  COPY
ENDFIL  0017    +1  COPY
RETADR  002A    +1  COPY
LENGTH  002D    +1  COPY
BUFFER  0033    +1  COPY
BUFEND  1033    +1  COPY
MAXLEN  1000    +1  COPY
RDREC   0000    +1  RDREC
RLOOP   0009    +1  RDREC
EXIT    0020    +1  RDREC
INPUT   0027    +1  RDREC
MAXLEN  0028    +1  RDREC
WRREC   0000    +1  WRREC
WLOOP   0006    +1  WRREC
```

리터럴 테이블

output_littab.txt

```
=C'EOF' 0030
=X'05'  001B
```

오브젝트 코드

프로젝트 > output_objectcode.txt

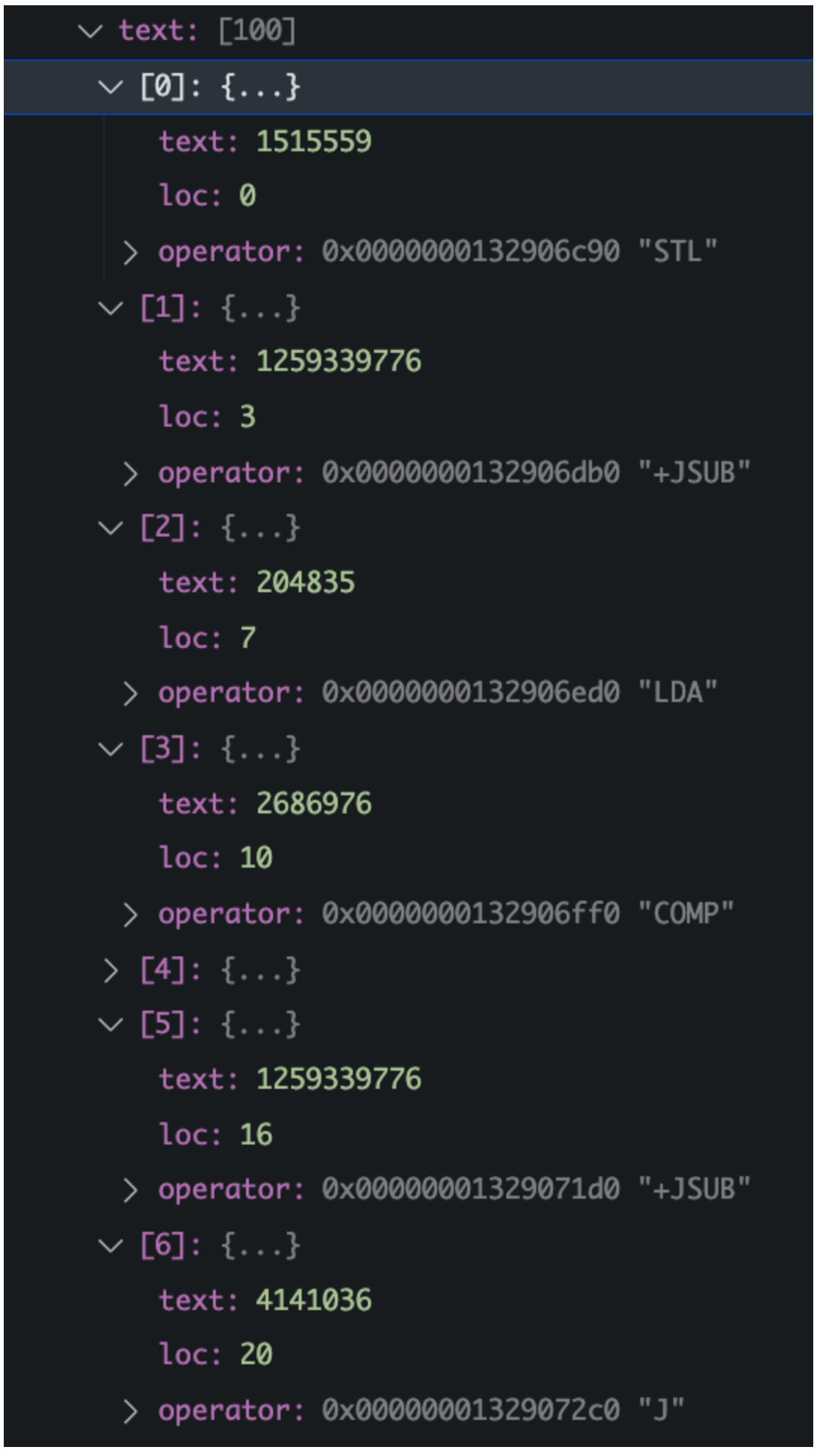
```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
T0000001D1720274B10000032023290000332007 4B1000003F2FEC32016F2016
T00001D0D10003F200A4B1000003E2000
T00003003454F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000

HRDREC 00000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T00001D0E3B2FE9131000004F000000F1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E

HWRREC 00000000001C
RLENGTHBUFFER
T00000000B41077100000E32012332FFA53900000DF2008B8503B2FEE4F00000005
M00000305+LENGTH
M00000D05+BUFFER
E


```

디버깅

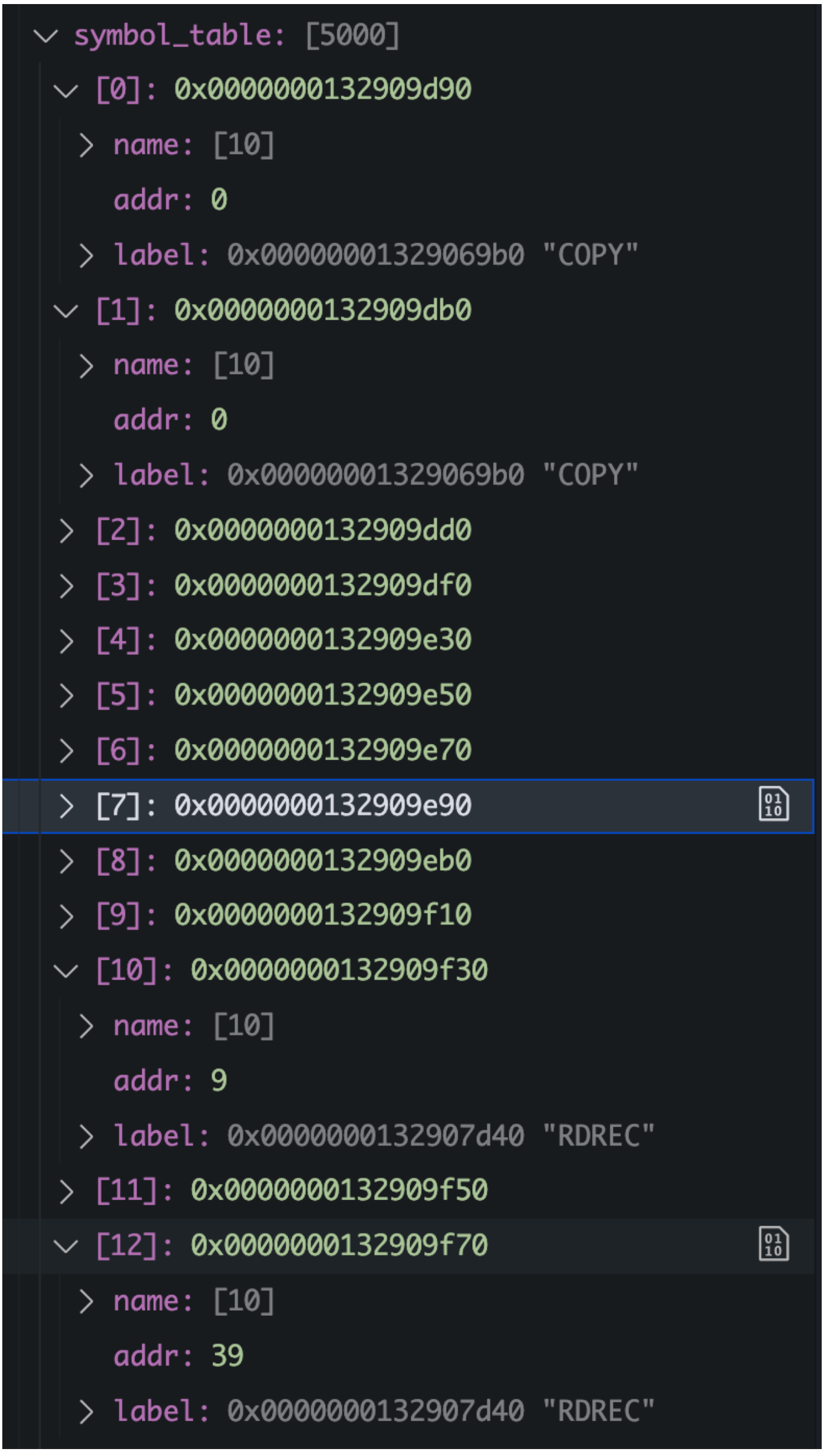
vscode 사용

오브젝트 코드의 text필드와 관련된 값들

오브젝트 코드, location, operator의 값이 초기화된 모습을 볼 수 있다.



심볼테이블이 라벨과 함께 초기화 돼있는 모습



리터럴 테이블의 값

C'EOF'와 X'05' 가 초기화 돼있다

```
literal_table: [5000]
  [0]: 0x0000000132909e10
    literal: [20]
      [0]: 61 '='
      [1]: 67 'C'
      [2]: 39 '''
      [3]: 69 'E'
      [4]: 79 'O'
      [5]: 70 'F'
      [6]: 39 '''
      [7]: 0 '\0'
      [8]: 0 '\0'
      [9]: 0 '\0'
      [10]: 0 '\0'
      [11]: 0 '\0'
      [12]: 0 '\0'
      [13]: 0 '\0'
      [14]: 0 '\0'
      [15]: 0 '\0'
      [16]: 0 '\0'
      [17]: 0 '\0'
      [18]: 0 '\0'
      [19]: 0 '\0'
    addr: 48
  [1]: 0x0000000132909ff0
    literal: [20]
      [0]: 61 '='
      [1]: 88 'X'
      [2]: 39 '''
      [3]: 48 '0'
      [4]: 53 '5'
      [5]: 39 '''
      [6]: 0 '\0'
      [7]: 0 '\0'
      [8]: 0 '\0'
      [9]: 0 '\0'
      [10]: 0 '\0'
      [11]: 0 '\0'
      [12]: 0 '\0'
      [13]: 0 '\0'
      [14]: 0 '\0'
      [15]: 0 '\0'
      [16]: 0 '\0'
      [17]: 0 '\0'
      [18]: 0 '\0'
      [19]: 0 '\0'
    addr: 27
  [2]: 0x0000000000000000
```

결론 및 보충할 점

단순히 주어진 input에 맞추어 구현하는 것이 아니라, 대부분의 (납득 가능한 레벨의) input에서는 작동되게 하고 싶었습니다. 비교할 필요는 없겠지만, 아마 꼼꼼하게 구현한 프로젝트 중 하나일 것이라고 예상합니다. 스스로 소소한 만족을 얻기도 했습니다. 하지만 제너럴하게 변환이 가능한 어셈블러를 구현하기 위해서는 고려할 사항이 너무 많았습니다. 리터럴이 할당되는 위치와, 중복된 리터럴은 어떻게 관리해야 하는 지, word혹은 byte값의 실제 value값 관리, equ와 심볼 연산 등을 건드리기 시작하면 분명히 바로 처리하지 못하는 예외가 나오게 될 것입니다.

그래서 아쉬운 점은 분명히 있지만, 사실 어셈블러의 작동 원리에 대해 깊게 공부할 수 있었던 과제였습니다. pass1에서 어떤 값들이 초기화 되며, 그 값들은 pass2에서 어떻게 활용되는 지 알 수 있었고, 한단계 더 나아가 어떤방식의 명령어를 사용하는게 더 효율적인지도 생각해볼 수 있었습니다. 이를테면 c언어에서 define은 어떻게 작동할 것인지, 외부에서 참조하여 modification을 사용하면 효율에 어떤 영향을 끼칠 지, 리터럴은 어떤 원리이며 immediate와는 어떤 관계인지 등입니다. 이런 요인들을 실제 코드를 작성할 때도 고려해볼 수 있을 것 같습니다.

다만, 어셈블러를 이해하고자 하는 과제의 목적에 비해 parsing과 관련하여 대부분의 시간을 쏟아야 했던 점은 아쉬웠습니다. 자잘하게 파싱이 많이 필요한 input보다는 어셈블러 구현의 기본적인 동작 위주로 input이 나왔으면 더 좋았을 것 같습니다.

소스 코드

이전에 pass1을 구현했던 과제는 길이 때문에 첨부하지 않겠습니다.

오브젝트 코드 작성에 필요한 유틸 함수들입니다.

```
#include "my_assembler.h"


int is_in_program_literal_table(char *literal[], char *str) {
    for (int i = 0; i < 10; i++) {
        if (literal[i] && strcmp(literal[i], str) == 0)
            return i;
    }
    return -1;
}


int make_symbol_table_output(const char *symtab_dir,
                             const symbol **symbol_table,
                             int symbol_table_length) {

    FILE *fp;
    int err;

    err = 0;

    fp = fopen(symtab_dir, "w");
    if (fp == NULL) {
        fprintf(stderr, "make_symbol_table_output: 심볼테이블 파일을 열 수 없습니다.\n");
        err = -1;
        return err;
    }

    for (int i = 0; i < symbol_table_length; i++) {
        fprintf(fp, "%s\t%04X\t+1\t%s\n", symbol_table[i]->name, symbol_table[i]->addr, s
    }

    fclose(fp);
    return 0;
}
```

```
int make_literal_table_output(const char *literal_table_dir,
                              const literal **literal_table,
                              int literal_table_length) {

    FILE *fp;
    int err;

    err = 0;

    fp = fopen(literal_table_dir, "w");
    if (fp == NULL) {
        fprintf(stderr, "make_literal_table_output: 리터럴테이블 파일을 열 수 없습니다.\n");
        err = -1;
        return err;
    }

    for (int i = 0; i < literal_table_length; i++) {
        fprintf(fp, "%s\t%04X\n", literal_table[i]->literal, literal_table[i]->addr);
    }

    fclose(fp);
    return 0;
}

unsigned int get_register_number(char *reg) {
    if (strcmp(reg, "A") == 0)
        return 0;
    else if (strcmp(reg, "X") == 0)
        return 1;
    else if (strcmp(reg, "L") == 0)
        return 2;
    else if (strcmp(reg, "B") == 0)
        return 3;
    else if (strcmp(reg, "S") == 0)
        return 4;
    else if (strcmp(reg, "T") == 0)
        return 5;
    else if (strcmp(reg, "F") == 0)
        return 6;
    else if (strcmp(reg, "PC") == 0)
        return 8;
    else if (strcmp(reg, "SW") == 0)
        return 9;
    else
        return -1;
}

int get_format_wrapper(char *operator, int table_index, const inst **inst_table) {
    if(operator[0] == '+')
        return 4;
    else
        return get_format(table_index, inst_table);
}

//총 8비트 2진수로 변환해주는 함수 뒤에 6비트를 nixbpe로 사용함
char *convert_binary(char nixbpe) {
    char *binary = (char *)malloc(sizeof(char) * 9);
    int i = 0;
    for (i = 0; i < 8; i++) {
```

```
        binary[i] = '0';
    }
    binary[i] = '\0';
    i = 7;
    while (nixbpe > 0) {
        binary[i] = (nixbpe % 2) + '0';
        nixbpe /= 2;
        i--;
    }
    return binary;
}


int find_from_token_operator(char *operator, const token *tokens[], int tokens_length) {
    for (int i = 0; i < tokens_length; i++) {
        if (tokens[i]->operator && strcmp(operator, tokens[i]->operator) == 0) {
            return i;
        }
    }
    return -1;
}

int find_from_token_label(char *label, const token *tokens[], int tokens_length) {
    for (int i = 0; i < tokens_length; i++) {
        if (tokens[i]->label && strcmp(label, tokens[i]->label) == 0) {
            return i;
        }
    }
    return -1;
}

int find_from_token_operand(char *operand, const token *tokens[], int tokens_length) {
    for (int i = 0; i < tokens_length; i++) {
        if (tokens[i]->operand[0] && strcmp(operand, tokens[i]->operand[0]) == 0) {
            return i;
        }
    }
    return -1;
}

int calculate_pc(int loc, char *operator) {
    if (strcmp(operator, "RESW") == 0)
        return loc + 3 * atoi(operator);
    else if (strcmp(operator, "RESB") == 0)
        return loc + atoi(operator);
    else if (strcmp(operator, "BYTE") == 0)
        return loc + 1;
    else if (strcmp(operator, "WORD") == 0)
        return loc + 3;
    else if (operator[0] == '+')
        return loc + 4;
    else
        return loc + 3;
}

int is_in_extref(char *symbol, extref *extref_table, int extref_table_length) {
    for (int i = 0; i < extref_table_length; i++) {
        if (strcmp(symbol, extref_table[i].symbol) == 0)
            return i;
    }
```

```
    return -1;
}

// 앞의 c 혹은 x를 제외하고 ' 사이에 있 제외한 리터럴을 찾아서 리터럴만 반환해주는 함수
char *get_literal(char *operand) {
    char *literal = (char *)malloc(sizeof(char) * 20);

    int j = 0;
    int i = 3;
    while (operand[i] != '\'') {
        literal[j] = operand[i];
        i++;
        j++;
    }
    return literal;
}
```

assem pass2 밑에 첨부할 두 함수를 통해 오브젝트 코드로 변환합니다

```
int assem_pass2(const token *tokens[], int tokens_length,
                const inst *inst_table[], int inst_table_length,
                const symbol *symbol_table[], int symbol_table_length,
                const literal *literal_table[], int literal_table_length,
                object_code *obj_code)
{

    int assem_index = 0;
    int subroutine_count = 0;

    while(assem_index < tokens_length) {
        while(assem_index < tokens_length &&
            (strcmp(tokens[assem_index]->operator, "START") != 0 &&
            strcmp(tokens[assem_index]->operator, "CSECT") != 0))
            assem_index++;

        if(assem_index >= tokens_length)
            break;

        if (strcmp(tokens[assem_index]->operator, "START") == 0){
            init_main_program(&assem_index, obj_code, tokens, tokens_length, inst_table,
        }
        else if (strcmp(tokens[assem_index]->operator, "CSECT") == 0) {
            init_subroutine(&assem_index, obj_code, tokens, tokens_length, inst_table, in
            subroutine_count++;
        }
    }

    obj_code->subroutine_length = subroutine_count;

    return 0;
}
```

output 파일 생성과 관련된 코드들입니다

```
int calculate_text_length(const obj_code_data *data, int cur_len) {
    int len = cur_len;
    int start_addr = data->text[cur_len].loc;
    int temp_addr = 0;

    if(strcmp(data->text[len].operator, "LTORG") == 0){
        while( len < data->text_length && strcmp(data->text[len].operator, "LTORG") == 0)
            len++;
        return len;
    }

    while (data->text[len].text == 0)
            {
                len++;
                if(len == data->text_length)
                    break;
            }
    if(len == data->text_length)
        return len;

    while(len < data->text_length) {
        if (
            strcmp(data->text[len].operator, "CSECT") == 0 ||
            strcmp(data->text[len].operator, "LTORG") == 0 ||
            (strcmp(data->text[len].operator, "WORD") != 0 && data->text[len].text == 0)
            break;
        temp_addr = data->text[len].loc;
        if(temp_addr - start_addr >= 29){
            break;
        }
        len++;
    }
    return len;
}

int calculate_print_section(const obj_code_data *data, int index) {
    while( data->text[index].operator != NULL &&
            (strcmp(data->text->operator, "END") == 0 ||
                strcmp(data->text->operator, "RESB") == 0 ||
            strcmp(data->text->operator, "RESW") == 0 ||
            strcmp(data->text->operator, "EQU") == 0)) {
        if(index == data->text_length)
            break;
        index++;
    }
    return index;
}

int make_objectcode_output(const char *filename, const object_code *obj_code) {
    FILE *fp = fopen(filename, "w");
    if (fp == NULL) {
        fprintf(stderr, "make_object_code_output: 파일을 열 수 없습니다.\n");
        return -1;
    }

    fprintf(fp, "H%-6s%06X%06X\n", obj_code->main_program.header_record.header_symbol, ob
    //extdef 출력 맨 처음에만 알파벳 D 출력하고 심볼 주소값 출력 줄바꿈 없이 출력

    if(obj_code->main_program.extdef_length != 0){
```

```
        fprintf(fp, "D");
        for(int i = 0; i < obj_code->main_program.extdef_length; i++) {
            fprintf(fp, "%-6s%06X", obj_code->main_program.extdef_table[i].symbol, obj_co
        }
        fprintf(fp, "\n");
    }

    //extref 출력 맨 처음에만 알파벳 R 출력하고 심볼 주소값 출력 줄바꿈 없이 출력
    if(obj_code->main_program.extref_table[0].symbol != NULL){
        fprintf(fp, "R");
        for(int i = 0; i < 3; i++) {
            if(obj_code->main_program.extref_table[i].symbol != NULL)
                fprintf(fp, "%-6s", obj_code->main_program.extref_table[i].symbol);
        }
        fprintf(fp, "\n");
    }


    //맨 앞에 T를 쓰고 해당 텍스트의 시작 주소와 줄바꿈 전까지의 길이 출력함 길이는 최대 16진수 1D까지 넘어
    // 그러므로 미리 길이를 체크해야함
    int prev_text_index = 0;
    int cur_text_index = 0;

    int test = 0;
    while(prev_text_index < obj_code->main_program.text_length) {
        cur_text_index = calculate_text_length(&(obj_code->main_program), prev_text_index

        if(cur_text_index == prev_text_index)
            break;


        if(strcmp(obj_code->main_program.text[prev_text_index].operator, "RESW") == 0
            || strcmp(obj_code->main_program.text[prev_text_index].operator, "RESB") == 0
            || strcmp(obj_code->main_program.text[prev_text_index].operator, "EQU") == 0)
            prev_text_index = calculate_print_section(&(obj_code->main_program), cur_text
            continue;

        }


        unsigned int text_start_addr = obj_code->main_program.text[prev_text_index].loc;
        unsigned int text_len = obj_code->main_program.text[cur_text_index].loc - text_st

        //T와 함계 시작주소와 텍스트 길이 16진수로 출력
        fprintf(fp, "T%06X%02X", text_start_addr, text_len);


        for(int i = prev_text_index; i < cur_text_index; i++) {
            //if ltorg 면서 16진수 FF보다 작다면 2자리로 출력
            if(strcmp(obj_code->main_program.text[i].operator, "LITERAL") == 0 && obj_cod
                fprintf(fp, "00%02X", obj_code->main_program.text[i].text);
            else if (strcmp(obj_code->main_program.text[i].operator, "WORD") == 0)
                fprintf(fp, "%06X", obj_code->main_program.text[i].text);
            else
                fprintf(fp, "%04X", obj_code->main_program.text[i].text);
        }
        fprintf(fp, "\n");

        prev_text_index = calculate_print_section(&(obj_code->main_program), cur_text_ind
        test++;
```

```
        if(test > 4)
            break;

    }

    //M 출력
    for(int i = 0; i < obj_code->main_program.modification_table_length; i++) {
        fprintf(fp, "M%06X%02X%s\n", obj_code->main_program.modification_table[i].addr, c
    }

    //end 출력

    //token 에서 end를 찾아서 그 인덱스를 찾아서 그 인덱스의 텍스트 주소를 출력
    fprintf(fp, "E%06X\n", obj_code->end_addr);

    //가독성 위한 개행
    fprintf(fp, "\n");


    //같은 방식으로 서브루틴 출력

    for(int i = 0; i < obj_code->subroutine_length; i++) {
        fprintf(fp, "H%-6s%06X%06X\n", obj_code->subroutine[i].header_record.header_symbo
        //extdef 출력 맨 처음에만 알파벳 D 출력하고 심볼 주소값 출력 줄바꿈 없이 출력
        if(obj_code->subroutine[i].extdef_length != 0){
            fprintf(fp, "D");
            for(int j = 0; j < obj_code->subroutine[i].extdef_length; j++) {
                fprintf(fp, "%-6s%06X", obj_code->subroutine[i].extdef_table[j].symbol, o
            }
            fprintf(fp, "\n");
        }

        //extref 출력 맨 처음에만 알파벳 R 출력하고 심볼 주소값 출력 줄바꿈 없이 출력
        if(obj_code->subroutine[i].extref_table[0].symbol != NULL){
            fprintf(fp, "R");
            for(int j = 0; j < 3; j++) {
                if(obj_code->subroutine[i].extref_table[j].symbol != NULL)
                    fprintf(fp, "%-6s", obj_code->subroutine[i].extref_table[j].symbol);
            }
            fprintf(fp, "\n");
        }

        //맨 앞에 T를 쓰고 해당 텍스트의 시작 주소와 줄바꿈 전까지의 길이 출력함 길이는 최대 16진수 1D까지
        // 그러므로 미리 길이를 체크해야함
        int prev_text_index = 0;
        int cur_text_index = 0;

        while(prev_text_index < obj_code->subroutine[i].text_length) {
            cur_text_index = calculate_text_length(&(obj_code->subroutine[i]), prev_text_
            if(cur_text_index == prev_text_index)
                break;


            unsigned int text_start_addr = obj_code->subroutine[i].text[prev_text_index]
            unsigned int text_len = obj_code->subroutine[i].text[cur_text_index].loc - te

            //T와 함계 시작주소와 텍스트 길이 16진수로 출력
            fprintf(fp, "T%06X%02X", text_start_addr, text_len);

            for(int j = prev_text_index; j < cur_text_index; j++) {
```

```
            //if ltorg 면서 16진수 FF보다 작다면 2자리로 출력
            if(strcmp(obj_code->subroutine[i].text[j].operator, "LITERAL") == 0 && ob
                fprintf(fp, "00%02X", obj_code->subroutine[i].text[j].text);
            else if (strcmp(obj_code->subroutine[i].text[j].operator, "WORD") == 0)
                fprintf(fp, "%06X", obj_code->subroutine[i].text[j].text);
            else
                fprintf(fp, "%04X", obj_code->subroutine[i].text[j].text);
        }

        fprintf(fp, "\n");

        prev_text_index = calculate_print_section(&(obj_code->subroutine[i]), cur_tex
    }

    //M 출력
    for(int j = 0; j < obj_code->subroutine[i].modification_table_length; j++) {
        fprintf(fp, "M%06X%02X%s\n", obj_code->subroutine[i].modification_table[j].ad
    }

        //E 출력과 가독성 위한 개행
        fprintf(fp, "E\n\n");
    }

return 0;
}
```

저번 과제에서 진행하지 않은 리터럴, 심볼, nixbpe 초기화 코드입니다

```
void init_nixbpe(token *tokens, const inst **inst_table, int inst_table_length) {

if(tokens->operator && search_opcode(tokens->operator, inst_table, inst_table_length) >=

    //RSUB
    if(strcmp(tokens->operator, "RSUB") == 0) {
        tokens->nixbpe |= 1 <<5;
        tokens->nixbpe |= 1 <<4;
        return;
    }

    if(tokens->operand[0]) {
        if(tokens->operand[0][0] == '@') {
            tokens->nixbpe |= 1 << 5;
        } else if(tokens->operand[0][0] == '#') {
            tokens->nixbpe |= 1 << 4;
        } else {
            tokens->nixbpe |= 1 << 5;
            tokens->nixbpe |= 1 << 4;
        }
    }

    if(tokens->operator && tokens->operator[0] == '+') {
        tokens->nixbpe |= 1 << 0;
    }
    else if (tokens->operand[0][0] != '#' && get_format(search_opcode(tokens->operato
        tokens->nixbpe |= 1 << 1;
    }
```

```
        if(tokens->operand[1] && tokens->operand[1][0] == 'X') {
            tokens->nixbpe |= 1 << 3;
        }
    }
}

int assem_pass1(const inst **inst_table, int inst_table_length,
                const char **input, int input_length, token **tokens,
                int *tokens_length, symbol **symbol_table,
                int *symbol_table_length, literal **literal_table,
                int *literal_table_length) {


    *tokens_length = 0;
    int err = 0;
    // 한줄씩 토큰을 만들어 tokens에 저장
    for(int i = 0; i < input_length; i++) {
        tokens[i] = (token *)malloc(sizeof(token));
        err = token_parsing(input[i], tokens[i], inst_table, inst_table_length);
        if(err < 0) {
            fprintf(stderr, "assem_pass1: 토큰 파싱에 실패했습니다.\n");
            return err;
        }
        init_nixbpe(tokens[i], inst_table, inst_table_length);
        (*tokens_length)++;
    }

    int loc = 0;

    //token을 기반으로 symbol_table과 literal_table을 만듦
    int len = *tokens_length;
    int symbol_len = 0;

    int literal_len = 0;
    int literal_cur = 0;
    char *label = 0;
    for(int i = 0; i < len; i++) {

         if(tokens[i]->operator) {
                if(strcmp(tokens[i]->operator, "START") == 0) {
                    loc = strtol(tokens[i]->operand[0], NULL, 16);
                    label = tokens[i]->label;
                }
                else if (strcmp(tokens[i]->operator, "CSECT") == 0) {
                     for(int j = literal_cur; j < literal_len; j++) {

                    literal_table[j]->addr = loc;

                    if(literal_table[j]->literal[1] == 'C') {
                        loc += strlen(literal_table[j]->literal) - 4;
                    } else {
                        loc += (strlen(literal_table[j]->literal) - 4) / 2;
                    }

                }
                    label = tokens[i]->label;
                    loc = 0;
                }
```

```
            else if (strcmp(tokens[i]->operator, "END") == 0) {
                for(int j = literal_cur; j < literal_len; j++) {

                literal_table[j]->addr = loc;

                if(literal_table[j]->literal[1] == 'C') {
                    loc += strlen(literal_table[j]->literal) - 4;
                } else {
                    loc += (strlen(literal_table[j]->literal) - 4) / 2;
                }
            }
                break;
            }
        }


        if(tokens[i]->label) {
            symbol_table[symbol_len] = (symbol *)malloc(sizeof(symbol));
            if (symbol_table[symbol_len] == NULL) {
                fprintf(stderr, "assem_pass1: 메모리 할당에 실패했습니다.\n");
                exit(1);
            }
            strcpy(symbol_table[symbol_len]->name, tokens[i]->label);

            if (tokens[i]->operator && strcmp(tokens[i]->operator, "EQU") == 0) {
                if(strcmp(tokens[i]->operand[0], "*") == 0) {
                    symbol_table[symbol_len]->addr = loc;
                }
                else {
                    if(return_operator_equ(tokens[i]->operand[0]) == -1) {
                        int symbol_index = is_in_symbol_table(tokens[i]->operand[0], symb
                        if(symbol_index == -1) {
                            fprintf(stderr, "assem_pass1: %d번째 토큰의 EQU 연산자의 피연산자가
                            return -1;
                        }
                        symbol_table[symbol_len]->addr = symbol_table[symbol_index]->addr
                    } else{
                        char **temp = equ_split(tokens[i]->operand[0]);

                        if(strings_len(temp) > 2) {
                            fprintf(stderr, "assem_pass1: %d번째 토큰의 EQU 연산자의 피연산자가
                            return -1;
                        }


                        int operator_num = return_operator_equ(tokens[i]->operand[0]);
                        int symbol_index1 = is_in_symbol_table(temp[0], symbol_table, sym
                        int symbol_index2 = is_in_symbol_table(temp[1], symbol_table, sym

                        // printf("%d %d\n", symbol_index1, symbol_index2);
                        // printf("%X %X\n", symbol_table[symbol_index1]->addr, symbol_ta

                        if(symbol_index1 == -1 || symbol_index2 == -1) {
                            fprintf(stderr, "assem_pass1: %d번째 토큰의 EQU 연산자의 피연산자가
                            return -1;
                        }

                        if(operator_num == 1) {
                            symbol_table[symbol_len]->addr = symbol_table[symbol_index1]-
                        } else if(operator_num == 2) {
```

```
                        symbol_table[symbol_len]->addr = symbol_table[symbol_index1]
                    } else if(operator_num == 3) {
                        symbol_table[symbol_len]->addr = symbol_table[symbol_index1]
                    } else if(operator_num == 4) {
                        symbol_table[symbol_len]->addr = symbol_table[symbol_index1]
                    }
                }
            }
        }
        else {
            symbol_table[symbol_len]->addr = loc;
        }
        symbol_table[symbol_len]->label = label;
        symbol_len++;
    }


    if (tokens[i]->operator) {
        if(tokens[i]->operand[0] && starts_with(tokens[i]->operand[0], '=')) {
            if(is_in_literal_table(tokens[i]->operand[0], literal_table, literal_len)
                literal_table[literal_len] = (literal *)malloc(sizeof(literal));
                if (literal_table[literal_len] == NULL) {
                    fprintf(stderr, "assem_pass1: 메모리 할당에 실패했습니다.\n");
                    exit(1);
                }
                strcpy(literal_table[literal_len]->literal, tokens[i]->operand[0]);
                literal_len++;
            }
        }

        if(search_opcode(tokens[i]->operator, inst_table, inst_table_length) >= 0) {
            if(tokens[i]->operator[0] == '+') {
                loc += 4;
            } else {
                loc += get_format(search_opcode(tokens[i]->operator, inst_table, inst
            }
        }
        else if (strcmp(tokens[i]->operator, "WORD") == 0) {
            loc += 3;
        }
        else if (strcmp(tokens[i]->operator, "RESW") == 0) {
            loc += 3 * atoi(tokens[i]->operand[0]);
        }
        else if (strcmp(tokens[i]->operator, "RESB") == 0) {
            loc += atoi(tokens[i]->operand[0]);
        }
        else if (strcmp(tokens[i]->operator, "LTORG") == 0) {
            for(int j = literal_cur; j < literal_len; j++) {

                literal_table[j]->addr = loc;

                if(literal_table[j]->literal[1] == 'C') {
                    loc += strlen(literal_table[j]->literal) - 4;
                } else {
                    loc += (strlen(literal_table[j]->literal) - 4) / 2;
                }
            }
            literal_cur = literal_len;
        }
        else if (strcmp(tokens[i]->operator, "BYTE") == 0) {
```

```
                loc += 1;
            }
        }

    }
    *symbol_table_length = symbol_len;
    *literal_table_length = literal_len;

    return 0;
}
```

main 코드를 오브젝트 코드로 변환해주는 함수입니다

```
int init_main_program(int *assem_index,
                      object_code *obj_code,
                      const token *tokens[], int tokens_length,
                      const inst *inst_table[], int inst_table_length,
                      const symbol *symbol_table[], int symbol_table_length,
                      const literal *literal_table[], int literal_table_length) {

    int start_index = find_from_token_operator("START", tokens, tokens_length);
    if (start_index == -1) {
        fprintf(stderr, "init_main_program: START가 없습니다.\n");
        return -1;
    }
    char *start_symbol = tokens[start_index]->label;
    int start_addr = symbol_table[is_in_symbol_table(start_symbol, (symbol **)symbol_tabl

    //헤더 레코드에 시작 심볼과 시작 주소를 저장
    obj_code->main_program.header_record.header_symbol = start_symbol;
    obj_code->main_program.header_record.start_address = start_addr;

    int end_index = find_from_token_operator("END", tokens, tokens_length);
    if (end_index == -1) {
        fprintf(stderr, "init_main_program: END가 없습니다.\n");
        return -1;
    }

    obj_code_data *main = &(obj_code->main_program);

    // 텍스트 테이블 초기화를 위한 변수들
    int text_index = 0;
    int text_length = 0;

    int text_table_index = 0;

    int text_loc = start_addr;
    //프로그램 카운터
    int pc = start_addr;

    int modification_index = 0;
    int literal_length = 0;

    while (text_index < tokens_length) {

        main->text[text_table_index].loc = text_loc;
```

```
 if(tokens[text_index]->operator == NULL) {
    text_index++;
    continue;
}

unsigned int assembled_code = 0;

if (strcmp(tokens[text_index]->operator, "END") == 0){
    //end 레코드 만났을 때 남은 리터럴이 있다면 여기에 선언
    for (int i = 0; i < literal_length; i++) {
        int leteral_len = strlen(main->literals[i]);
        int temp_assem_code = 0;

        //리터럴을 한글자씩 16진수로 변환해서 assembled_code에 저장
        // 예를들어 EOF는 454F46이 된다.
        for (int j = 0; j < leteral_len; j++) {
            temp_assem_code = (temp_assem_code << 8) + main->literals[i][j];
        }
        main->text[text_table_index].text = temp_assem_code;
        main->text[text_table_index].loc = text_loc;
        main->text[text_table_index].operator = "LITERAL";
        text_table_index++;
        text_loc += leteral_len;
    }
    break;


}
//CSECT를 만나면 서브루틴으로 이동
if (strcmp(tokens[text_index]->operator, "CSECT") == 0)
    break;

if (strcmp(tokens[text_index]->operator, "START") == 0 || strcmp(tokens[text_inde
    text_index++;
    continue;
}

//extref와 extdef를 구조체에 저장
if (strcmp(tokens[text_index]->operator, "EXTREF") == 0) {
    int extref_index = 0;
    while(extref_index < 3 && tokens[text_index]->operand[extref_index]) {
        main->extref_table[extref_index].symbol = tokens[text_index]->operand[ext
        main->extref_table[extref_index].addr = 0;
        extref_index++;
    }
    text_index++;
    continue;
}
if (strcmp(tokens[text_index]->operator, "EXTDEF") == 0) {
    int extdef_index = 0;
    int operand_index = 0;

    while (operand_index < 3 && tokens[text_index]->operand[operand_index]) {
        main->extdef_table[extdef_index].symbol = tokens[text_index]->operand[ope
        int index = is_in_symbol_table_wrapper(tokens[text_index]->operand[extdef
        main->extdef_table[extdef_index].addr = symbol_table[index]->addr;
        extdef_index++;
        operand_index++;
    }
```

```
            text_index++;
            main->extdef_length = extdef_index;
            continue ;
        }

        //리터럴 테이블 초기화
         if (tokens[text_index]->operand[0] && tokens[text_index]->operand[0][0] == '='){
            if(is_in_program_literal_table(main->literals, tokens[text_index]->operand[0]
                main->literals[literal_length] = get_literal(tokens[text_index]->operand
                literal_length++;
            }
        }
        //ltorg 만나면 리터럴 초기화
          if (strcmp(tokens[text_index]->operator, "LTORG") == 0) {
            for (int i = 0; i < literal_length; i++) {
                int leteral_len = strlen(main->literals[i]);
                int temp_assem_code = 0;

                //리터럴을 한글자씩 16진수로 변환해서 assembled_code에 저장
                // 예를들어 EOF는 454F46이 된다.
                for (int j = 0; j < leteral_len; j++) {
                    temp_assem_code = (temp_assem_code << 8) + main->literals[i][j];
                }
                main->text[text_table_index].text = temp_assem_code;
                main->text[text_table_index].loc = text_loc;
                main->text[text_table_index].operator = tokens[text_index]->operator;
                text_table_index++;
                text_loc += strlen(main->literals[i]);
            }
            literal_length = 0;
            text_index++;
            continue;
        }

        //레이블 예외 처리를 지나고 명령어를 만났을 때는 프로그램 카운터 계산
        pc = calculate_pc(text_loc, tokens[text_index]->operator);

        //Reserve 관련 명령어 처리
        if (strcmp(tokens[text_index]->operator, "RESW") == 0){
            text_loc += 3 * atoi(tokens[text_index]->operand[0]);
        }
        else if (strcmp(tokens[text_index]->operator, "RESB") == 0){
            text_loc += atoi(tokens[text_index]->operand[0]);
        }
        else if (strcmp(tokens[text_index]->operator, "BYTE") == 0){
            if (tokens[text_index]->operand[0][0] == 'C') {
                int i = 2;
                while(tokens[text_index]->operand[0][i] != '\'') {
                    assembled_code = (assembled_code << 8) + tokens[text_index]->operand
                    i++;
                }
            }
            else if (tokens[text_index]->operand[0][0] == 'X') {
                int i = 2;
                while(tokens[text_index]->operand[0][i] != '\'') {
                    int temp = 0;
                    if(tokens[text_index]->operand[0][i] >= '0' && tokens[text_index]->op
                        temp = tokens[text_index]->operand[0][i] - '0';
                    else if(tokens[text_index]->operand[0][i] >= 'A' && tokens[text_index
                        temp = tokens[text_index]->operand[0][i] - 'A' + 10;
```

```
                assembled_code = (assembled_code << 4) + temp;
                i++;
            }
        }
        else {
            fprintf(stderr, "init_main_program: BYTE 타입이 잘못되었습니다.\n");
            return -1;
        }
        text_loc += 1;
    }
    else if (strcmp(tokens[text_index]->operator, "WORD") == 0){

        char **temp_operand = equ_split(tokens[text_index]->operand[0]);
        if(is_in_extref(temp_operand[0], main->extref_table, 3) != -1) {


            if(strings_len(temp_operand) == 2) {
                int operator_equ = return_operator_equ(tokens[text_index]->operand[0]
                for(int i = 0; i < 2; i++){
                 char *temp = temp_operand[i];
                 if(i == 2){
                     if(operator_equ == 1){
                         char *temp2 = (char *)malloc(sizeof(char) * 10);
                         temp2[0] = '+';
                         strcat(temp2, temp_operand[i]);
                         temp = temp2;
                     }
                     else if(operator_equ == 2){
                         char *temp2 = (char *)malloc(sizeof(char) * 10);
                         temp2[0] = '-';
                         strcat(temp2, temp_operand[i]);
                         temp = temp2;
                     }
                     else if(operator_equ == 3){
                         char *temp2 = (char *)malloc(sizeof(char) * 10);
                         temp2[0] = '*';
                         strcat(temp2, temp_operand[i]);
                         temp = temp2;
                     }
                     else if(operator_equ == 4){
                         char *temp2 = (char *)malloc(sizeof(char) * 10);
                         temp2[0] = '/';
                         strcat(temp2, temp_operand[i]);
                         temp = temp2;
                     }
                 }
                 else{
                       char *temp2 = (char *)malloc(sizeof(char) * 10);
                         temp2[0] = '+';
                         strcat(temp2, temp_operand[i]);
                         temp = temp2;

                 }
                     main->modification_table[modification_index].addr = text_loc;
                     main->modification_table[modification_index].modification_length
                     main->modification_table[modification_index].symbol = temp;
                     modification_index++;
                }
            }
```

```
                else {

                    //+기호 붙이기
                    char *temp = tokens[text_index]->operand[0];
                    char *temp2 = (char *)malloc(sizeof(char) * 10);
                    temp2[0] = '+';
                    strcat(temp2, temp);
                    temp = temp2;

                    //modification record
                    main->modification_table[modification_index].addr = text_loc;
                    main->modification_table[modification_index].modification_length = 6;
                    main->modification_table[modification_index].symbol = temp;
                    modification_index++;
                }

                assembled_code = 0;
            }
            else {
                int word_symbol = is_in_symbol_table_wrapper(tokens[text_index]->operand
                if (word_symbol != -1)
                    assembled_code = symbol_table[word_symbol]->addr;
                else
                    assembled_code = atoi(tokens[text_index]->operand[0]);
            }
            text_loc += 3;
        }
        else {
            int inst_index = search_opcode(tokens[text_index]->operator, inst_table, inst
            if (inst_index == -1) {
                fprintf(stderr, "init_main_program: 명령어가 잘못되었습니다.\n");
                return -1;
            }
            //format에 따라서 assembled_code를 만들어줌
            assembled_code = inst_table[inst_index]->op;
            if (get_format_wrapper(tokens[text_index]->operator, inst_index, inst_table)
                assembled_code = assembled_code << 8;
                if (tokens[text_index]->operand[0]) {
                    int reg1 = get_register_number(tokens[text_index]->operand[0]);
                    assembled_code += reg1 << 4;
                }
                if (tokens[text_index]->operand[1]) {
                    int reg2 = get_register_number(tokens[text_index]->operand[1]);
                    assembled_code += reg2;
                }
                text_loc += 2;
            }
            else if (get_format_wrapper(tokens[text_index]->operator, inst_index, inst_ta
                assembled_code = assembled_code << 16;
                assembled_code |= tokens[text_index]->nixbpe << 12;
                if (tokens[text_index]->operand[0]) {

                    char *operand  = tokens[text_index]->operand[0];
                    if (tokens[text_index]->operand[0][0] == '#')
                        assembled_code += atoi(operand+1);
                    else if (is_in_symbol_table_wrapper(operand, main->header_record.head
                        int operand_symbol = is_in_symbol_table_wrapper(tokens[text_index
                        assembled_code += (symbol_table[operand_symbol]->addr - pc) & 0xF
                    }
                    else if (is_in_literal_table(operand, (literal **)literal_table, lite
```

```
                    int operand_literal = is_in_literal_table(tokens[text_index]->ope
                    assembled_code += (literal_table[operand_literal]->addr - pc);
                }
                else {
                    assembled_code += atoi(tokens[text_index]->operand[0]);
                }
            }
            text_loc += 3;
        }
        //4형식일때는 modification record 같이 초기화
        else if (get_format_wrapper(tokens[text_index]->operator, inst_index, inst_ta
            assembled_code = assembled_code << 24;
            assembled_code |= tokens[text_index]->nixbpe << 20;
            if (tokens[text_index]->operand[0]) {

                if (is_in_extref(tokens[text_index]->operand[0], main->extref_table,

                    char *temp = tokens[text_index]->operand[0];
                    char *temp2 = (char *)malloc(sizeof(char) * 10);
                    temp2[0] = '+';
                    strcat(temp2, temp);
                    temp = temp2;

                    //modification record
                    main->modification_table[modification_index].addr = text_loc + 1;
                    main->modification_table[modification_index].modification_length
                    main->modification_table[modification_index].symbol = temp;
                    modification_index++;

                    assembled_code += 0;
                }
                else{
                    int operand_symbol = is_in_symbol_table_wrapper(tokens[text_index
                    if (operand_symbol != -1) {
                        assembled_code |= symbol_table[operand_symbol]->addr & 0xFFFF
                    }
                    else {
                        assembled_code += atoi(tokens[text_index]->operand[0]);
                    }
                }
            }
            else {
                text_loc += 4;
            }
        }
        main->text[text_table_index].text = assembled_code;
        main->text[text_table_index].operator = tokens[text_index]->operator;
        text_table_index++;
        text_index++;
    }

    *assem_index = text_index;
    main->text_length = text_table_index;
    main->modification_table_length = modification_index;
    main->header_record.program_length = text_loc - start_addr;


    return 0;

}
```

거의 유사한 로직으로 subroutine을 초기화하는 함수입니다

```
//서브루틴도 main에서 했던 것 그대로 진행
int init_subroutine(int *assem_index,
                        object_code *obj_code,
                        const token *tokens[], int tokens_length,
                        const inst *inst_table[], int inst_table_length,
                        const symbol *symbol_table[], int symbol_table_length,
                        const literal *literal_table[], int literal_table_length,
                        int subroutine_count) {

    char *csect_symbol = tokens[*assem_index]->label;
    int csect_addr = symbol_table[is_in_symbol_table(csect_symbol,(symbol **)symbol_table
```

```
    obj_code->subroutine[subroutine_count].header_record.header_symbol = csect_symbol;
    obj_code->subroutine[subroutine_count].header_record.start_address = csect_addr;

    int end_index = find_from_token_operator("END", tokens, tokens_length);
    if (end_index == -1) {
        fprintf(stderr, "init_subroutine: END가 없습니다.\n");
        return -1;
    }
    char *end_symbol = tokens[end_index]->operand[0];

    obj_code_data *subroutine = &(obj_code->subroutine[subroutine_count]);


    *assem_index = *assem_index + 1;
    int text_index = *assem_index;
    int text_length = 0;

    int text_table_index = 0;

    int text_loc = csect_addr;
    int pc = csect_addr;

    int modification_index = 0;
    int literal_length = 0;
```

```
    while (text_index < tokens_length) {

        subroutine->text[text_table_index].loc = text_loc;

        if(tokens[text_index]->operator == NULL) {
            text_index++;
            continue;
        }

        unsigned int assembled_code = 0;

        if(strcmp(tokens[text_index]->operator, "END") == 0)
        {
            //남은 리터럴 할당
            for (int i = 0; i < literal_length; i++) {
                //만약 앞 글자가 C라면 리터럴을 한글자씩 16진수로 변환해서 assembled_code에 저장
                char *temp_literal = get_literal(subroutine->literals[i]);
```

```
                // 예를들어 EOF는 454F46이 된다.
                if(subroutine->literals[i][1] == 'C') {
                    int literal_len = strlen(temp_literal);
                    int temp_assem_code = 0;
                    for (int j = 0; j < literal_len; j++) {
                        temp_assem_code = (temp_assem_code << 8) + temp_literal[j];
                    }
                    subroutine->text[text_table_index].text = temp_assem_code;
                    subroutine->text[text_table_index].loc = text_loc;
                    subroutine->text[text_table_index].operator = "LITERAL";
                    text_table_index++;
                    text_loc += literal_len;
                }
                else if (subroutine->literals[i][1] == 'X') {
                    int literal_len = strlen(temp_literal);
                    int temp_assem_code = 0;
                    for (int j = 0; j < literal_len; j++) {
                        int temp = 0;
                        if(temp_literal[j] >= '0' && temp_literal[j] <= '9')
                            temp = temp_literal[j] - '0';
                        else if(temp_literal[j] >= 'A' && temp_literal[j] <= 'F')
                            temp = temp_literal[j] - 'A' + 10;
                        temp_assem_code = (temp_assem_code << 4) + temp;
                    }
                    subroutine->text[text_table_index].text = temp_assem_code;
                    subroutine->text[text_table_index].loc = text_loc;
                    subroutine->text[text_table_index].operator = "LITERAL";
                    text_table_index++;
                    text_loc += literal_len / 2;
                }
            }
            break;
        }
        if (strcmp(tokens[text_index]->operator, "CSECT") == 0)
            break;

        if (strcmp(tokens[text_index]->operator, "START") == 0 || strcmp(tokens[text_inde
            text_index++;
            continue;
        }

        if (strcmp(tokens[text_index]->operator, "EXTREF") == 0) {
            int extref_index = 0;
            while(extref_index < 3 && tokens[text_index]->operand[extref_index]) {
                subroutine->extref_table[extref_index].symbol = tokens[text_index]->opera
                subroutine->extref_table[extref_index].addr = 0;
                extref_index++;
            }
            text_index++;
            continue;
        }
        if (strcmp(tokens[text_index]->operator, "EXTDEF") == 0) {
            int extdef_index = 0;
            int operand_index = 0;
            while(operand_index < 3 && tokens[text_index]->operand[operand_index]) {
                subroutine->extdef_table[extdef_index].symbol = tokens[text_index]->opera
                int index = is_in_symbol_table_wrapper(tokens[text_index]->operand[extdef
                subroutine->extdef_table[extdef_index].addr = symbol_table[index]->addr;
                extdef_index++;
                operand_index++;
```

```
            }
            text_index++;
            subroutine->extdef_length = extdef_index;
            continue ;
        }

        if (tokens[text_index]->operand[0] && tokens[text_index]->operand[0][0] == '='){
            if(is_in_program_literal_table(subroutine->literals, tokens[text_index]->oper
                subroutine->literals[literal_length] = tokens[text_index]->operand[0];
                literal_length++;
            }
        }



        if (strcmp(tokens[text_index]->operator, "LTORG") == 0) {
             for (int i = 0; i < literal_length; i++) {
                //만약 앞 글자가 C라면 리터럴을 한글자씩 16진수로 변환해서 assembled_code에 저장
                char *temp_literal = get_literal(subroutine->literals[i]);
                // 예를들어 EOF는 454F46이 된다.
                if(subroutine->literals[i][1] == 'C') {
                    int literal_len = strlen(temp_literal);
                    int temp_assem_code = 0;
                    for (int j = 0; j < literal_len; j++) {
                        temp_assem_code = (temp_assem_code << 8) + temp_literal[j];
                    }
                    subroutine->text[text_table_index].text = temp_assem_code;
                    subroutine->text[text_table_index].loc = text_loc;
                    subroutine->text[text_table_index].operator = "LITERAL";
                    text_table_index++;
                    text_loc += literal_len;
                }
                else if (subroutine->literals[i][1] == 'X') {
                    int literal_len = strlen(temp_literal);
                    int temp_assem_code = 0;
                    for (int j = 0; j < literal_len; j++) {
                        int temp = 0;
                        if(temp_literal[j] >= '0' && temp_literal[j] <= '9')
                            temp = temp_literal[j] - '0';
                        else if(temp_literal[j] >= 'A' && temp_literal[j] <= 'F')
                            temp = temp_literal[j] - 'A' + 10;
                        temp_assem_code = (temp_assem_code << 4) + temp;
                    }
                    subroutine->text[text_table_index].text = temp_assem_code;
                    subroutine->text[text_table_index].loc = text_loc;
                    subroutine->text[text_table_index].operator = "LITERAL";
                    text_table_index++;
                    text_loc += literal_len / 2;
                }
            }
            continue;
        }

        pc = calculate_pc(text_loc, tokens[text_index]->operator);

        if (strcmp(tokens[text_index]->operator, "RESW") == 0){
            text_loc += 3 * atoi(tokens[text_index]->operand[0]);
        }
        else if (strcmp(tokens[text_index]->operator, "RESB") == 0){
            text_loc += atoi(tokens[text_index]->operand[0]);
```

```
        }
        else if (strcmp(tokens[text_index]->operator, "BYTE") == 0){
            if (tokens[text_index]->operand[0][0] == 'C') {
                int i = 2;
                while(tokens[text_index]->operand[0][i] != '\'') {
                    assembled_code = (assembled_code << 8) + tokens[text_index]->operand
                    i++;
                }
            }
            //x 일떄는 그대로 저장 예를들어 x'F1'이면 16진수로 출력했을 때 그대로 F1로 나와야함
            else if (tokens[text_index]->operand[0][0] == 'X') {
                int i = 2;
                while(tokens[text_index]->operand[0][i] != '\'') {
                    int temp = 0;
                    if(tokens[text_index]->operand[0][i] >= '0' && tokens[text_index]->op
                        temp = tokens[text_index]->operand[0][i] - '0';
                    else if(tokens[text_index]->operand[0][i] >= 'A' && tokens[text_index
                        temp = tokens[text_index]->operand[0][i] - 'A' + 10;
                    assembled_code = (assembled_code << 4) + temp;
                    i++;
                }
            }
            else {
                fprintf(stderr, "init_subroutine: BYTE 타입이 잘못되었습니다.\n");
                return -1;
            }
            text_loc += 1;
        }
        else if (strcmp(tokens[text_index]->operator, "WORD") == 0){
            char **temp_operand = equ_split(tokens[text_index]->operand[0]);
            if(is_in_extref(temp_operand[0], subroutine->extref_table, 3) != -1) {

                if(strings_len(temp_operand) == 2) {
                    int operator_equ = return_operator_equ(tokens[text_index]->operand[0]
                   for(int i = 0; i < 2; i++){
                        char *temp = temp_operand[i];
                        if(i == 1) {
                            if (operator_equ == 1) {
                                //연산자를 심볼 앞에 더해줘야함 예를 들어 +buffer
                                char *temp2 = (char *)malloc(sizeof(char) * 10);
                                temp2[0] = '+';
                                strcat(temp2, temp);
                                temp = temp2;
                            }
                            else if (operator_equ == 2) {
                                //연산자를 심볼 뒤에 더해줘야함 예를 들어 -buffer
                                char *temp2 = (char *)malloc(sizeof(char) * 10);
                                temp2[0] = '-';
                                strcat(temp2, temp);
                                temp = temp2;
                            }
                            else if (operator_equ == 3) {
                                //연산자를 심볼 앞에 더해줘야함 예를 들어 *buffer
                                char *temp2 = (char *)malloc(sizeof(char) * 10);
                                temp2[0] = '*';
                                strcat(temp2, temp);
                                temp = temp2;
                            }
                            else if (operator_equ == 4) {
                                //연산자를 심볼 앞에 더해줘야함 예를 들어 /buffer
```

```
                        char *temp2 = (char *)malloc(sizeof(char) * 10);
                        temp2[0] = '/';
                        strcat(temp2, temp);
                        temp = temp2;
                    }
                }
                else{
                      char *temp2 = (char *)malloc(sizeof(char) * 10);
                        temp2[0] = '+';
                        strcat(temp2, temp);
                        temp = temp2;
                }
                subroutine->modification_table[modification_index].addr = text_lo
                subroutine->modification_table[modification_index].modification_l
                subroutine->modification_table[modification_index].symbol = temp;
                modification_index++;
            }
        }
        else {
            //+기호 붙이기
            char *temp = tokens[text_index]->operand[0];
            char *temp2 = (char *)malloc(sizeof(char) * 10);
            temp2[0] = '+';
            strcat(temp2, temp);
            temp = temp2;

        //modification record
            subroutine->modification_table[modification_index].addr = text_loc;
            subroutine->modification_table[modification_index].modification_lengt
            subroutine->modification_table[modification_index].symbol = temp;
            modification_index++;
        }

        assembled_code = 0;

    }
    else {
        int word_symbol = is_in_symbol_table_wrapper(tokens[text_index]->operand
        if (word_symbol != -1)
            assembled_code = symbol_table[word_symbol]->addr;
        else
            assembled_code = atoi(tokens[text_index]->operand[0]);
        }

        text_loc += 3;

    }
    else {
        int inst_index = search_opcode(tokens[text_index]->operator, inst_table, inst
        if (inst_index == -1) {
            fprintf(stderr, "init_subroutine: 명령어가 잘못되었습니다.\n");
            return -1;
        }
        assembled_code = inst_table[inst_index]->op;
        if (get_format_wrapper(tokens[text_index]->operator, inst_index, inst_table)
            assembled_code = assembled_code << 8;
            if (tokens[text_index]->operand[0]) {
                int reg1 = get_register_number(tokens[text_index]->operand[0]);
                assembled_code += reg1 << 4;
            }
```

```
                if (tokens[text_index]->operand[1]) {
                    int reg2 = get_register_number(tokens[text_index]->operand[1]);
                    assembled_code += reg2;
                }
                text_loc += 2;
            }
            else if (get_format_wrapper(tokens[text_index]->operator, inst_index, inst_ta
                assembled_code = assembled_code << 16;
                assembled_code |= tokens[text_index]->nixbpe << 12;
                if (tokens[text_index]->operand[0]) {

                    char *operand  = tokens[text_index]->operand[0];
                    if (tokens[text_index]->operand[0][0] == '#')
                        assembled_code += atoi(operand+1);
                    else if (is_in_symbol_table_wrapper(operand, subroutine->header_recor
                        int operand_symbol = is_in_symbol_table_wrapper(tokens[text_index
                        assembled_code += (symbol_table[operand_symbol]->addr - pc) & 0xF
                    }
                    else if (is_in_literal_table(operand, (literal **)literal_table, lite
                        int operand_literal = is_in_literal_table(tokens[text_index]->ope
                        assembled_code += (literal_table[operand_literal]->addr - pc);
                    }
                    else {
                        assembled_code += atoi(tokens[text_index]->operand[0]);
                    }
                }
                text_loc += 3;
            }
            else if (get_format_wrapper(tokens[text_index]->operator, inst_index, inst_ta
                assembled_code = assembled_code << 24;

                assembled_code |= tokens[text_index]->nixbpe << 20;

                if (tokens[text_index]->operand[0]) {

                    if (is_in_extref(tokens[text_index]->operand[0], subroutine->extref_t

                        char *temp = tokens[text_index]->operand[0];
                        char *temp2 = (char *)malloc(sizeof(char) * 10);
                        temp2[0] = '+';
                        strcat(temp2, temp);
                        temp = temp2;

                        //modification record
                        subroutine->modification_table[modification_index].addr = text_lo
                        subroutine->modification_table[modification_index].modification_l
                        subroutine->modification_table[modification_index].symbol = temp;
                        modification_index++;

                        assembled_code += 0;
                    }
                    else{
                        int operand_symbol = is_in_symbol_table_wrapper(tokens[text_index
                        if (operand_symbol != -1) {
                            assembled_code |= symbol_table[operand_symbol]->addr & 0xFFFF
                        }
                        else {
                            assembled_code += atoi(tokens[text_index]->operand[0]);
                        }
                    }
```

```
                }

                text_loc += 4;

            }
        }
        subroutine->text[text_table_index].text = assembled_code;
        subroutine->text[text_table_index].operator = tokens[text_index]->operator;
        text_table_index++;
        text_index++;
    }

    *assem_index = text_index;
    subroutine->text_length = text_table_index;
    subroutine->modification_table_length = modification_index;
    subroutine->header_record.program_length = text_loc - csect_addr;

    return 0;
}
```