시스템프로그래밍(나) Project 1

- SIC/XE Assembler -

컴퓨터학부 20222904 정해성(240)

목차

1. 동기 및 목적

본 보고서는 지난 파싱 과제의 연장선인, SIC/XE 문법으로 작성된 소스를 오브젝트 프로그램으로 바꾸는 어셈블러를 구현하는 과제에 대한 보고서입니다.

Pass 1에서 중간 파일을 생성하지 않고 오브젝트 프로그램 생성에 필요한 정보들을 토큰 테이블과 심볼 테이블, 리터럴 테이블에 모두 저장해두는 방식으로 진행하였습니다. 그리고 오브젝트 프로그램을 생성할 때에는 `object_code` 객체 하나로 모든 것을 구성할 수 있게 설계하였습니다.

2. 설계/구현 아이디어

- **instruction 파싱, 소스 파싱**

  파싱하는 과정은 지난 과제와 동일하나, inst_table.txt 형식이 콤마 문자로 구분되어 있던 것을 탭 문자('₩t')로 구분하는 것으로 변경하면서 파싱하는 구현 코드를 간결하게 할 수 있었습니다.

- **Pass 1**

  Pass 1 단계에서는 Pass 2 단계에서 활용하기 위한 토큰 테이블, 심볼 테이블, 리터럴 테이블을 설정합니다. Pass 2에서는 이 세 테이블만 활용하여 오브젝트 프로그램을 구성할 것이므로 각 구조체 정의에 필드를 추가하는 것이 필요했습니다.

  우선 token 구조체에는 `addr` 필드를 추가하였습니다. 이미 Pass 1에서 location counter를 계산해주고 있기 때문에 Pass 2에서 한 번 더 계산하는 것이 번거롭다고 판단했기 때문입니다. 해당 필드는 Pass 2에서 PC 상대주소를 계산할 때 활용됩니다. 그리고 symbol 구조체에는 `rflag`와 `csect_name`이라는 필드를 추가하였습니다. `rflag`는 해당 심볼의 주소가 상대적인지를 확인하는 플래그이며, `csect_name`은 심볼이 정의된 control section의 이름을 담고 있습니다. literal 구조체에는 `bytes`, `type`, `value` 필드를 추가하였습니다. 해당 리터럴이 차지하는 메모리의 바이트 크기를 `bytes` 필드에 넣어두었으며, `type`과 `value`는 이름에서도 알 수 있듯이 리터럴의 실제 타입과 값을 나타냅니다.

  Pass 1은 소스 코드의 각 라인을 파싱한 결과를 직접 활용하며, 크게 심볼 처리, 리터럴 처리, 연산자 처리 총 세 단계로 구분할 수 있습니다.

  심볼 처리 단계에서는 토큰의 label이 존재할 때만 수행됩니다. 일반적으로는

location counter(locctr)와 해당 label로 심볼을 구성하여 심볼 테이블에 삽입하고, 이 과정은 insert_label_into_symtbl() 함수에서 수행합니다. 하지만 EQU 지시어가 사용된 경우에는 값을 계산해주는 등 특수한 처리가 있어야 합니다. 이 과정은 calculate_equ() 함수에서 수행하며, 피연산자가 "*"일 경우 해당 위치의 locctr로 대입해주고 수식이 있다면 이름이 같은 심볼을 심볼 테이블에서 찾아 해당 주소값을 더하거나 빼주는 방식입니다. 덧셈 연산과 뺄셈 연산만 가능하며 괄호가 들어간 연산은 아직 구현하지 못했습니다.

다음은 리터럴 처리 단계입니다. 이 단계는 단순히 피연산자의 첫 번째 문자가 '=' 이면 수행되는 것으로, insert_literal_into_littbl() 함수에서 구현하고 있습니다. 리터럴과 관련한 함수가 하나 더 정의되어 있는데, set_literal_addr()이라는 함수가 그것입니다. 리터럴이 정의되는 메모리 주소는 바로 정해지지 않습니다. END로 프로그램이 종료되거나, CSECT로 새로운 control section이 생기거나, LTORG 지시어로 리터럴의 위치를 지정해줄 때 비로소 리터럴의 주소가 정해집니다. 이 세 지시어 중 하나를 만났을 때 set_literal_addr() 함수는 호출함으로써 주소가 정해지지 않은 채 쌓여 있던 리터럴들을 해소시키고 있습니다.

마지막으로 수정사항이 가장 많았던 연산자 처리 단계입니다. 연산자가 지시어가 아니라면 instruction table에서 대응되는 명령어를 찾고 형식에 맞춰 locctr 값을 갱신해줍니다. 만약 해당 연산자가 지시어라면 해당 지시어의 동작에 맞게 구현하였는데, EXTREF, EXTDEF는 Pass 2에서 수행되므로 제외하였습니다.

- **Pass 2**

오브젝트 프로그램을 구성하기 위해 필요한 레코드의 구조체(header_record, text_record, end_record, modify_record, define_record, reference_record)를 정의하고 control_section 구조체의 필드로 적절히 정의하였습니다. 그리고 오브젝트 프로그램은 여러 control section으로 구성될 수 있기 때문에 object_code 구조체의 필드는 control_section 배열로 정의되어 있습니다.

Pass 2에서는 Pass 1에서 파싱한 토큰이 저장되어 있는 토큰 테이블을 순회하면서 연산자에 맞게 적절히 레코드를 채워 넣습니다. 각 레코드의 내용을 채우는 과정을 간단히 설명하겠습니다.

- header_record: START나 CSECT가 등장하여 새로운 control section가 시작될 때 정의합니다.
- define_record: 먼저 symbol 필드와 심볼의 개수를 나타내는 symbol_cnt를

EXTDEF가 등장했을 때 대입해줍니다. 그리고 이후에 토큰에서 label이 나올 때마다 define_record에 등록된 label인지 확인하고, 존재한다면 대응되는 위치에 상대주소 값을 넣습니다. 해당 label의 상대주소 값은 토큰의 addr 필드를 참조하여 확인합니다.

- reference_record: EXTREF가 등장했을 때 피연산자의 심볼을 바로 추가해줍니다.
- end_record: header_record가 정의될 때 함께 정의되며, header_record에도 지정되어 있는 시작주소로 설정됩니다.
- text_record, modification_record: 토큰의 연산자가 instruction_table에 존재한다면, opcode와 토큰의 nixbpe 필드를 적절히 조합하여 오브젝트 코드를 삽입합니다. 만약 EXTREF로 정의된 심볼을 피연산자로 사용하고 있다면 주소 부분을 0으로 치환하고, modification_record의 필드를 추가합니다.

- **오브젝트 프로그램 생성**

오브젝트 프로그램 생성 단계에서는 Pass 2에서 정의한 object_code의 레코드들을 오브젝트 프로그램의 형식에 맞게 출력합니다. make_objectcode_output()이라는 함수에서 수행합니다.

3. **수행 결과**

output_symtab.txt

```
COPY    0
FIRST   0       1 COPY
CLOOP   3       1 COPY
ENDFIL  17      1 COPY
RETADR  2A      1 COPY
LENGTH  2D      1 COPY
BUFFER  33      1 COPY
BUFEND  1033    1 COPY
MAXLEN  1000    1 COPY
RDREC   0
RLOOP   9       1 RDREC
EXIT    20      1 RDREC
INPUT   27      1 RDREC
MAXLEN  28      1 RDREC
WRREC   0
WLOOP   6       1 WRREC
```

1의 의미는 상대주소임을 의미하는 `rflag` 값입니다.

output_littab.txt

```
=C'EOF' 30
=X'05'  1B
```

output_objectcode.txt

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
E000000
HRDREC 00000000002B
RBUFFERLENGTHBUFEND
E000000
HWRREC 00000000001C
RLENGTHBUFFER
E000000
```

text_record, modification_record는 아직 구현하지 못했습니다.

4. 결론 및 보충할 점

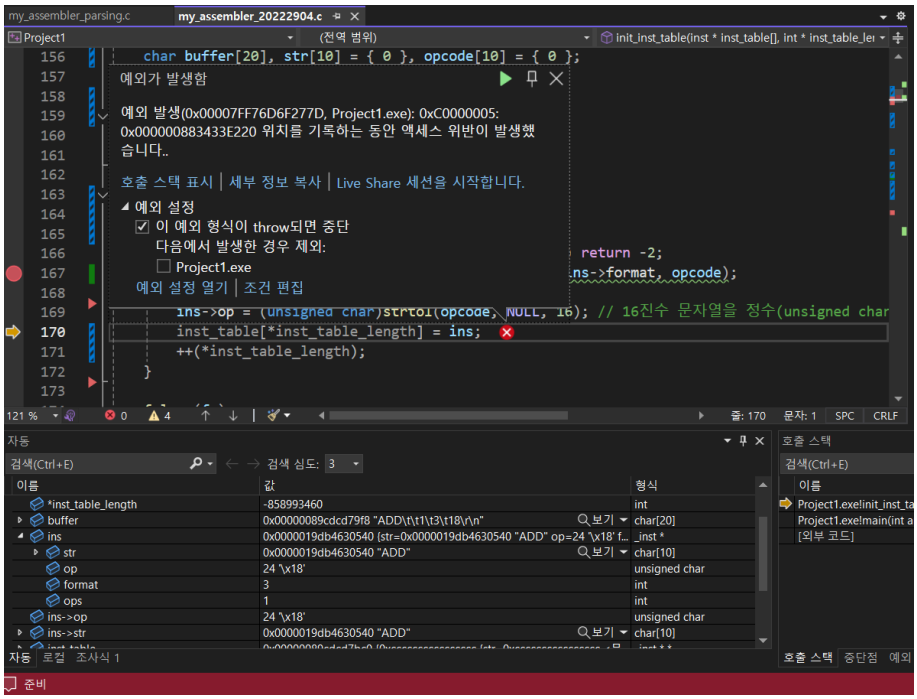
- 결론

어셈블러가 SIC/XE 소스를 파싱하고, 오브젝트 코드를 생성하기 위해 필요한 심볼 테이블, 리터럴 테이블 등을 직접 채우는 프로그램을 구현해보면서 어셈블러의 동작을 더 잘 이해할 수 있었습니다.

- 보충할 점

코드를 다듬을 시간이 부족하여 에러의 종류를 세분화하지 못한 점이 아쉽습니다. 조교님께서 올려주신 파싱 프로그램처럼 에러 코드를 좀 더 상세하게 구분한다면 디버깅, 유지보수 효율성을 올릴 수 있을 것 같습니다.

아래는 파싱 함수을 다시 작성하는 과정에서 오류가 나서 디버거를 사용하여 버그를 해결한 스크린샷입니다. 원인은 *inst_table_length를 0으로 초기화해주지 않고 배열의 인덱스를 참조해서 발생했던 것이었습니다.



5. 소스코드(+주석)

```
/**
 * @file my_assembler_20222904.c
 * @date 2024-04-09
 * @version 0.1.0
 *
 * @brief SIC/XE 소스코드를 object code로 변환하는 프로그램
 *
 * @details
 * SIC/XE 소스코드를 해당 머신에서 동작하도록 object code로 변환하는
 * 프로그램이다. 파일 내에서 사용되는 문자열 "00000000"에는 자신의 학번을
 * 기입한다.
 */
#define _CRT_SECURE_NO_WARNINGS
//#define DEBUG
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* 파일명의 "00000000"은 자신의 학번으로 변경할 것 */
#include "my_assembler_20222904.h"

#define DIR_START 1
#define DIR_CSECT 2
#define DIR_LTORG 3
#define DIR_RESW 4
#define DIR_RESB 5
#define DIR_WORD 6
#define DIR_BYTE 7
#define DIR_END 8
#define DIR_EXTDEF 9
#define DIR_EXTREF 10
#define DIR_EQU 11

/**
 * @brief 사용자로부터 SIC/XE 소스코드를 받아서 object code를 출력한다.
 *
 * @details
 * 사용자로부터 SIC/XE 소스코드를 받아서 object code를 출력한다.
 * 특별한 사유가
 * 없는 한 변경하지 말 것.
 */
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;

    /** 소스코드 내의 심볼을 저장하는 테이블 */
    symbol *symbol_table[MAX_TABLE_LENGTH];
    int symbol_table_length;

    /** 소스코드 내의 리터럴을 저장하는 테이블 */
    literal *literal_table[MAX_TABLE_LENGTH];
```

```
    int literal_table_length;

    /** 오브젝트 코드를 저장하는 변수 */
    object_code* obj_code =
(object_code*)malloc(sizeof(object_code));

    int err = 0;

    if ((err = init_inst_table(inst_table,
&inst_table_length,
                               "inst_table.txt")) < 0) {
        fprintf(stderr,
                "init_inst_table: 기계어 목록 초기화에 실패했습
니다. "
                "(error_code: %d)\n",
                err);
        return -1;
    }

    if ((err = init_input(input, &input_length,
"input.txt")) < 0) {
        fprintf(stderr, "init_input: 소스코드 입력에 실패했습니
다. (error_code: %d)\n", err);
        return -1;
    }

    if ((err = assem_pass1((const inst **)inst_table,
inst_table_length,
                           (const char **)input, input_length,
                           tokens, &tokens_length,
                           symbol_table, &symbol_table_length,
                           literal_table,
&literal_table_length)) < 0) {
        fprintf(stderr, "assem_pass1: 패스1 과정에서 실패했습니
다. (error_code: %d)\n", err);
        return -1;
    }

    /** 프로젝트1에서는 불필요함 */
    /*
    if ((err = make_opcode_output("output_opcode.txt",
(const token **)tokens,
                                  tokens_length, (const inst
**)inst_table,
                                  inst_table_length)) < 0) {
        fprintf(stderr,
                "make_opcode_output: opcode 파일 출력 과정에서
실패했습니다. "
                "(error_code: %d)\n",
                err);
        return -1;
    }
    */

    if ((err = make_symbol_table_output("output_symtab.txt",
                                        (const symbol
**)symbol_table,
                                        symbol_table_length)) <
0) {
        fprintf(stderr,
                "make_symbol_table_output: 심볼테이블 파일 출력
과정에서 "
                "실패했습니다. (error_code: %d)\n",
                err);
        return -1;
    }

    if ((err =
```

```
make_literal_table_output("output_littab.txt",
                                        (const literal
**)literal_table,
                                        literal_table_length))
< 0) {
        fprintf(stderr,
                "make_literal_table_output: 리터럴테이블 파일
출력 과정에서 "
                "실패했습니다. (error_code: %d)\n",
                err);
        return -1;
    }

    if ((err = assem_pass2((const token **)tokens,
tokens_length,
                           (const inst **)inst_table,
inst_table_length,
                           (const symbol **)symbol_table,
symbol_table_length,
                           (const literal **)literal_table,
                           literal_table_length, obj_code)) <
0) {
        fprintf(stderr,
                "assem_pass2: 패스2 과정에서 실패했습니다.
(error_code: %d)\n",
                err);
        return -1;
    }

    if ((err = make_objectcode_output(NULL, //
"output_objectcode.txt",
                                      (const object_code
*)obj_code)) < 0) {
        fprintf(stderr,
                "make_objectcode_output: 오브젝트코드 파일 출력
과정에서 "
                "실패했습니다. (error_code: %d)\n",
                err);
        return -1;
    }

    return 0;
}

/**
 * @brief 기계어 목록 파일(inst_table.txt)을 읽어 기계어 목록
 * 테이블(inst_table)을 생성한다.
 *
 * @param inst_table 기계어 목록 테이블의 시작 주소
 * @param inst_table_length 기계어 목록 테이블의 길이를 저장하는
변수 주소
 * @param inst_table_dir 기계어 목록 파일 경로
 * @return 오류 코드 (정상 종료 = 0)
 *
 * @details
 * 기계어 목록 파일(inst_table.txt)을 읽어 기계어 목록 테이블
(inst_table)을
 * 생성한다. 기계어 목록 파일 형식은 자유롭게 구현한다. 예시는 다음
과 같다.
 *
=============================================================
 *          | 이름 | 형식 | 기계어 코드 | 오퍼랜드의 갯수 | \n |
 *
=============================================================
 */
```

```
  int init_inst_table(inst *inst_table[], int
*inst_table_length,
                      const char *inst_table_dir) {
      FILE* fp;
      inst* ins;
      char buffer[20], str[10] = { 0 }, opcode[10] = { 0 };

      if ((fp = fopen(inst_table_dir, "rb")) == NULL) {
          return -1;
      }

      *inst_table_length = 0;

      while (!feof(fp)) {
          fgets(buffer, 20, fp);

          if ((ins = (inst*)malloc(sizeof(inst))) == NULL)
return -2;
          sscanf(buffer, "%s %d %d %s", str, &ins->ops, &ins-
>format, opcode);
          strcpy(ins->str, str);
          ins->op = (unsigned char)strtol(opcode, NULL, 16); //
16진수 문자열을 정수(unsigned char)로 변환
          inst_table[*inst_table_length] = ins;
          ++(*inst_table_length);
      }

      fclose(fp);
      return 0;
  }

  /**
   * @brief SIC/XE 소스코드 파일(input.txt)을 읽어 소스코드 테이블
(input)을
   * 생성한다.
   *
   * @param input 소스코드 테이블의 시작 주소
   * @param input_length 소스코드 테이블의 길이를 저장하는 변수 주
소
   * @param input_dir 소스코드 파일 경로
   * @return 오류 코드 (정상 종료 = 0)
   */
  int init_input(char *input[], int *input_length, const char
*input_dir) {
      FILE* fp;
      char buffer[250] = { 0, }, *line;

      if ((fp = fopen(input_dir, "rb")) == NULL) {
          return -1;
      }

      *input_length = 0;

      while (!feof(fp)) {
          fgets(buffer, 249, fp);
          if ((line = (char*)malloc(strlen(buffer) + 1)) ==
NULL) {
              fclose(fp);
              return -2;
          }
          sscanf(buffer, "%[^\r^\n]", line);
          input[*input_length] = line;
          ++(*input_length);
      }

      fclose(fp);

      return 0;
  }
```

```
/**
 * label이 유효한지 체크하는 함수입니다.
 * - label의 길이는 6보다 작거나 같아야 합니다.
 * - 같은 control section에 같은 label이 존재하면 안 됩니다.
 *
 * @return 오류: < 0, 정상 종료 == 0
 */
static int check_symbol_valid(const char* label, const char* csect_name, int line_number,
                              symbol* symbol_table[], int* symbol_table_length) {
    if (label == NULL) return -1;

    if (strlen(label) > 6) {
        fprintf(stderr, "line #%d : label '%s' is too long. label's length must be <= 6\n", line_number, label);
        return -5; // exceed symbol length
    }

    // symbol table에 존재하는지 확인
    // 존재한다면 에러
    for (int s = 0; s < *symbol_table_length; s++) {
        if (strcmp(symbol_table[s]->name, label) == 0 &&
            strcmp(symbol_table[s]->csect_name, csect_name) == 0) {
            fprintf(stderr, "line #%d : symbol '%s' is duplicated\n", line_number, label);
            return -3; // duplicated symbol
        }
    }

    return 0;
}
```

```
/**
 * symbol 인스턴스를 생성하여 symbol_table의 마지막에 삽입합니다.
 */
static int insert_label_into_symtbl(const char* label, int addr, const char* csect_name,
                                    symbol* symbol_table[], int* symbol_table_length) {
    if (label == NULL) return -1;

    symbol* sb;
    if ((sb = (symbol*)malloc(sizeof(symbol))) == NULL) return -2;

    sb->addr = addr;
    strcpy(sb->name, label);
    if (csect_name != NULL) {
        strcpy(sb->csect_name, csect_name);
        sb->rflag = 1;
    }
    else {
        sb->csect_name[0] = '\0';
        sb->rflag = 0;
    }
    symbol_table[*symbol_table_length] = sb;
    ++(*symbol_table_length);

    return 0;
}

/**
 * literal 인스턴스를 생성하여 literal_table의 마지막에 삽입합니다.
 */
static int insert_literal_into_littbl(const char* literal_str, literal* literal_table[], int* literal_table_length) {
```

```
    if (literal_str == NULL) return -1;

    // 중복되는 리터럴은 삽입할 필요 없음
    for (int i = 0; i < *literal_table_length; i++) {
        if (strcmp(literal_table[i]->literal, literal_str) ==
0)
            return 0;
    }

    literal* lit;
    if ((lit = (literal*)malloc(sizeof(literal))) == NULL)
return -2;
    lit->addr = 0;
    strcpy(lit->literal, literal_str);

    int lit_len = strlen(lit->literal);
    switch (lit->literal[1]) {
    case 'C':
        if (lit->literal[2] != '\'' || lit->literal[lit_len -
1] != '\'')
            return -10; // wrong literal format
        lit->bytes = lit_len - 4;
        lit->type = 'C';
        if ((lit->val_chars = (char*)malloc(lit->bytes + 1))
== NULL) return -2;
        memcpy(lit->val_chars, lit->literal + 3, lit->bytes);
        lit->val_chars[lit->bytes] = '\0';
        break;
    case 'X':
        if (lit->literal[2] != '\'' || lit->literal[lit_len -
1] != '\'')
            return -10; // wrong literal format
        lit->bytes = (lit_len - 4 + 1) / 2;
        lit->type = 'X';
        char hex[10] = { 0 };
        memcpy(hex, lit->literal + 3, lit_len - 4);
        lit->val_hex = (int)strtol(hex, NULL, 16);
        break;
    default:
        lit->bytes = 3; // WORD
        lit->type = 'N';
        lit->val_num = atoi(lit->literal + 1);
        break;
    }

    literal_table[*literal_table_length] = lit;
    ++(*literal_table_length);
    return 0;
}

/**
 * @param lit_last_idx 주소가 지정되지 않은 리터럴의 첫 번째 인덱
스
 * @param locctr 현재 위치의 location counter
 *
 * @details
 * location counter를 리터럴 바이트 수에 따라 적절히 증가시키면서
리터럴 테이블의 리터럴의 주소를 지정해줍니다.
 *
 * 그리고 토큰으로도 변환하여 토큰 테이블에 넣어줍니다.
 * 변환된 토큰은 label이 "*"이며, operator가 리터럴의 값입니다.
 */
static int set_literal_addr(int* lit_last_idx, int* locctr,
                            token* tokens[], int* tokens_length,
                            literal* literal_table[], const int*
literal_table_length) {
    // lit_last_idx == literal_table_length이면 모든 리터럴의
주소가 지정되었다는 의미
    if (*lit_last_idx == *literal_table_length) return 0;
```

```
    while (*lit_last_idx < *literal_table_length) {
        literal* lit = literal_table[*lit_last_idx];
        lit->addr = *locctr;
        *locctr += lit->bytes;
        ++(*lit_last_idx);

        // 토큰으로 변환하여 토큰 테이블에도 대입
        token* lit_tok;
        if ((lit_tok = (token*)malloc(sizeof(token))) == NULL)
return -2;
        lit_tok->comment = NULL;
        lit_tok->nixbpe = 0;
        for (int i = 0; i < MAX_OPERAND_PER_INST; i++)
            lit_tok->operand[i] = NULL;

        if ((lit_tok->label = (char*)malloc(2)) == NULL)
return -2;
        if ((lit_tok->operator = (char*)malloc(strlen(lit-
>literal) + 1)) == NULL) return -2;
        lit_tok->addr = lit->addr;
        lit_tok->label[0] = '*';
        lit_tok->label[1] = '\0';
        strcpy(lit_tok->operator, lit->literal);
        tokens[*tokens_length] = lit_tok;
        ++(*tokens_length);
    }
    return 0;
}

/**
 * str이 어떤 지시어인지 확인하고 대응되는 정수값을 반환합니다.
 * 지시어가 아니라면 -1을 반환합니다.
 */
static int directive(const char* str) {
    if (str == NULL) return -1;
    if (strcmp(str, "START") == 0) {
        return DIR_START;
    }
    else if (strcmp(str, "CSECT") == 0) {
        return DIR_CSECT;
    }
    else if (strcmp(str, "LTORG") == 0) {
        return DIR_LTORG;
    }
    else if (strcmp(str, "RESW") == 0) {
        return DIR_RESW;
    }
    else if (strcmp(str, "RESB") == 0) {
        return DIR_RESB;
    }
    else if (strcmp(str, "WORD") == 0) {
        return DIR_WORD;
    }
    else if (strcmp(str, "BYTE") == 0) {
        return DIR_BYTE;
    }
    else if (strcmp(str, "END") == 0) {
        return DIR_END;
    }
    else if (strcmp(str, "EXTDEF") == 0) {
        return DIR_EXTDEF;
    }
    else if (strcmp(str, "EXTREF") == 0) {
        return DIR_EXTREF;
    }
    else if (strcmp(str, "EQU") == 0) {
        return DIR_EQU;
    }
    return -1;
}

/**
 * 'BUFEND-BUFFER'와 같은 수식을 계산하여 그 값을 dest에 대입합니
```

다.

```
   * 수식이 '*'이라면 locctr 값을 대입해줍니다.
   */
  static int calculate_equ(const char* expr, int* dest, int
locctr,
                           const symbol* symbol_table[], int
symbol_table_length) {
      if (expr == NULL) return -1;

      // operand가 * 이면 현재 주소
      if (expr[0] == '*') {
          *dest = locctr;
          return 0;
      }

      int add_flag = 1;
      char tok_buf[10];
      symbol* sb = NULL;

      *dest = 0;
      for (int st = 0; expr[st] != '\0' && st < 10; st++) {
          int end = st;
          while (expr[end] != '\0' && expr[end] != '-' &&
expr[end] != '+') ++end;
          memcpy(tok_buf, expr + st, end - st);
          tok_buf[end - st] = '\0';

          // 토큰과 같은 symbol 찾기
          sb = NULL;
          for (int i = 0; i < symbol_table_length; i++) {
              if (strcmp(symbol_table[i]->name, tok_buf) == 0) {
                  sb = symbol_table[i];
                  break;
              }
          }
          if (sb == NULL) return -2;

          if (add_flag) *dest += sb->addr;
          else *dest -= sb->addr;

          add_flag = (expr[end] == '+');
          st = end;
      }

      return 0;
  }


  static int insert_dummy_token(token* tokens[], int*
tokens_length, int addr) {
      token* tok;
      if ((tok = (token*)malloc(sizeof(token))) == NULL) return
-1;
      tok->addr = addr;
      tok->comment = NULL;
      tok->operand[0] = NULL;
      tok->operator = NULL;
      tok->label = NULL;
      tokens[*tokens_length] = tok;
      ++(*tokens_length);
      return 0;
  }

  /**
   * @brief 어셈블리 코드을 위한 패스 1 과정을 수행한다.
   *
   * @param inst_table 기계어 목록 테이블의 주소
   * @param inst_table_length 기계어 목록 테이블의 길이
   * @param input 소스코드 테이블의 주소
   * @param input_length 소스코드 테이블의 길이
```

```
   * @param tokens 토큰 테이블의 시작 주소
   * @param tokens_length 토큰 테이블의 길이를 저장하는 변수 주소
   * @param symbol_table 심볼 테이블의 시작 주소
   * @param symbol_table_length 심볼 테이블의 길이를 저장하는 변수 주소
   * @param literal_table 리터럴 테이블의 시작 주소
   * @param literal_table_length 리터럴 테이블의 길이를 저장하는 변수 주소
   * @return 오류 코드 (정상 종료 = 0)
   *
   * @details
   * 어셈블리 코드를 위한 패스1 과정을 수행하는 함수이다. 패스 1에서는 프로그램
   * 소스를 스캔하여 해당하는 토큰 단위로 분리하여 프로그램 라인별 토큰 테이블을
   * 생성한다. 토큰 테이블은 token_parsing 함수를 호출하여 설정하여야 한다. 또한,
   * assem_pass2 과정에서 사용하기 위한 심볼 테이블 및 리터럴 테이블을 생성한다.
   */
  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) {
      int locctr = 0; // location counter
      int err;
      char* csect_name = NULL; // current control section name
      token* tok;

      // 주소가 지정되지 않은 리터럴의 첫 번째 인덱스
      // (lit_last_idx == *literal_table_length이면 테이블의 모든 리터럴이 주소가 지정되었다는 의미)
      int lit_last_idx = 0;

      *tokens_length = 0;
      *symbol_table_length = 0;
      *literal_table_length = 0;

      for (int i = 0; i < input_length; i++) {
          if (input[i][0] == '.') continue; // '.'으로 시작하는 라인은 주석으로 판단
          if ((tok = (token*)malloc(sizeof(token))) == NULL) return -2;

          // Parsing
          if ((err = token_parsing(input[i], tok, inst_table, inst_table_length)) < 0) return err;
          tok->addr = locctr;
          tokens[*tokens_length] = tok;
          ++(*tokens_length);

          int dir = directive(tok->operator);
          if (dir == DIR_END) break;

          /// [symbol 처리]
          if (tok->label != NULL) {
              if ((err = check_symbol_valid(tok->label, csect_name, i + 1, symbol_table, symbol_table_length)) < 0) return err;
```

```
            if (dir == DIR_EQU && tok->operand[0] != NULL) {
                int addr;
                // 수식 계산 후 addr에 넣어준다
                if (calculate_equ(tok->operand[0], &addr,
locctr, symbol_table, *symbol_table_length) < 0) return -5;
                if ((err = insert_label_into_symtbl(tok->label,
addr, csect_name, symbol_table, symbol_table_length)) < 0)
return err;
                // EQU이면 더 이상 할 일 없음. 다음 라인으로 넘어가
도 된다
                continue;
            }
            else if (dir != DIR_START && dir != DIR_CSECT) {
                // START이거나 CSECT는 아래에서 따로 처리해주기 때
문에
                // 여기서는 넣어주지 않는다
                if ((err = insert_label_into_symtbl(tok->label,
locctr, csect_name, symbol_table, symbol_table_length)) < 0)
return err;
            }
        }

        /// [리터럴 삽입]
        if (tok->operand[0] != NULL && tok->operand[0][0] ==
'=') {
            if ((err = insert_literal_into_littbl(tok-
>operand[0], literal_table, literal_table_length)) < 0) return
err;
        }

        /// [명령어 처리]
        if (dir == -1) {
            int inst_idx = search_opcode(tok->operator,
inst_table, inst_table_length);
            if (inst_idx == -1) return -4; // unknown operator
            inst* ins = inst_table[inst_idx];

            locctr += inst_table[inst_idx]->format;
            if (tok->operator[0] == '+') locctr++;
        }
        else {
            switch (dir) {
            case DIR_START:
                if (tok->operand[0] != NULL) {
                    locctr = atoi(tok->operand[0]); // init
                    csect_name = tok->label;
                    if ((err = insert_label_into_symtbl(tok-
>label, locctr, NULL, symbol_table, symbol_table_length)) < 0)
return err;
                }
                break;
            case DIR_CSECT:
                // Control Section이 시작하기 전에 앞에서 쌓인 리
터럴 넣어줌
                --(*tokens_length);
                if ((err = set_literal_addr(&lit_last_idx,
&locctr,
                    tokens, tokens_length, literal_table,
literal_table_length)) < 0) return err;
                // 길이 계산 편의를 위해 넣어줍니다.
                if (insert_dummy_token(tokens, tokens_length,
locctr) < 0) return -1;

                tokens[*tokens_length] = tok;
                ++(*tokens_length);
                tok->addr = 0;
                locctr = 0;
                csect_name = tok->label;
```

```
                if ((err = insert_label_into_symtbl(tok->label,
locctr, NULL, symbol_table, symbol_table_length)) < 0) return
err;
                break;
            case DIR_LTORG:
                // LTORG 토큰은 삭제하고 리터럴로 채우기
                --(*tokens_length);
                free(tokens[*tokens_length]);
                if ((err = set_literal_addr(&lit_last_idx,
&locctr,
                    tokens, tokens_length, literal_table,
literal_table_length)) < 0) return err;
                break;
            case DIR_RESW:
                if (tok->operand[0] != NULL)
                    locctr += atoi(tok->operand[0]) * 3;
                break;
            case DIR_RESB:
                if (tok->operand[0] != NULL)
                    locctr += atoi(tok->operand[0]);
                break;
            case DIR_WORD:
                locctr += 3;
                break;
            case DIR_BYTE:
                if (tok->operand[0] != NULL) {
                    switch (tok->operand[0][0]) {
                    case 'C':
                        locctr += strlen(tok->operand[0]) - 3;
// C, 따옴표 2개 총 3개 제외
                        break;
                    case 'X':
                        locctr += (strlen(tok->operand[0]) - 3
+ 1) / 2; // X, 따옴표 2개 총 3개 제외 (두 문자 당 1byte)
                        break;
                    }
                }
                break;
            }
        }
    } // end for

    if ((err = set_literal_addr(&lit_last_idx, &locctr,
        tokens, tokens_length, literal_table,
literal_table_length)) < 0) return err;
    // 길이 계산 편의를 위해 넣어줍니다.
    if (insert_dummy_token(tokens, tokens_length, locctr) <
0) return -1;

 #ifdef DEBUG
    // 토큰 테이블 출력
    for (int i = 0; i < *tokens_length; i++) {
        token* tok = tokens[i];
        printf("%X\t%s\t%s\t", tok->addr, tok->label, tok-
>operator);
        for (int j = 0; j < 3 && tok->operand[j] != NULL; j++)
            printf("%s ", tok->operand[j]);
        printf("\n");
    }
 #endif
    return 0;
 }

 /**
  * @brief 한 줄의 소스코드를 파싱하여 토큰에 저장한다.
  *
  * @param input 파싱할 소스코드 문자열
  * @param tok 결과를 저장할 토큰 구조체 주소
  * @param inst_table 기계어 목록 테이블의 주소
  * @param inst_table_length 기계어 목록 테이블의 길이
```

```
  * @return 오류 코드 (정상 종료 = 0)
  */
 int token_parsing(const char *input, token *tok,
                   const inst *inst_table[], int
inst_table_length) {
     tok->label = NULL;
     tok->operator = NULL;
     tok->comment = NULL;
     for (int i = 0; i < MAX_OPERAND_PER_INST; i++)
         tok->operand[i] = NULL;
     tok->nixbpe = 0;

     char label[10] = { 0 }, opr[10] = { 0 }, opnd[100] =
{ 0 }, comment[100] = { 0 };

     if (input[0] == '\t' || input[0] == ' ') {
         // 공백 문자로 시작하면 레이블이 존재하지 않는다고 판단
         sscanf(input, "\t%[^\t]\t%[^\t]\t%[^\0]", opr, opnd,
comment);
     }
     else {
         sscanf(input, "%[^\t]\t%[^\t]\t%[^\t]\t%[^\0]", label,
opr, opnd, comment);
         if ((tok->label = (char*)malloc(strlen(label) + 1)) ==
NULL) return -2;
         strcpy(tok->label, label);
     }

     // [명령어 복사 단계]
     if (opr[0] != '\0') {
         if ((tok->operator = (char*)malloc(strlen(opr) + 1))
== NULL) return -2;
         strcpy(tok->operator, opr);
     }

     // [comment 복사 단계]
     if (comment[0] != '\0') {
         if ((tok->comment = (char*)malloc(strlen(comment) +
1)) == NULL) return -2;
         strcpy(tok->comment, comment);
     }

     // [operand 복사 단계]
     int opnd_cnt = 0;
     for (int st = 0; opnd[st] != '\0' && opnd_cnt <
MAX_OPERAND_PER_INST; st++) {
         int end = st;
         while (opnd[end] != ',' && opnd[end] != '\0' &&
opnd[end] != '\t') ++end;
         int len = end - st;
         if ((tok->operand[opnd_cnt] = (char*)malloc(len + 1))
== NULL) return -2;
         memcpy(tok->operand[opnd_cnt], opnd + st, len);
         tok->operand[opnd_cnt][len] = '\0';
         ++opnd_cnt;
         st = end;
     }

     // [nixbpe]
     int inst_idx = search_opcode(opr, inst_table,
inst_table_length);
     if (inst_idx >= 0 && inst_table[inst_idx]->format != 0 &&
tok->operand[0] != NULL) {
         char c = tok->operand[0][0];
         if (c == '@') tok->nixbpe += (1 << 5);      // 10 0000
         else if (c == '#') tok->nixbpe += (1 << 4); // 01 0000
         else tok->nixbpe += (3 << 4);              // 11 0000

         if (opnd_cnt > 0 && strcmp(tok->operand[opnd_cnt - 1],
"X") == 0)
             tok->nixbpe += (1 << 3); // 00 1000
```

```
    }
    if (tok->operator != NULL && tok->operator[0] == '+')
        tok->nixbpe += 1;
    return 0;
}

/**
 * @brief 기계어 목록 테이블에서 특정 기계어를 검색하여, 해당 기계에가 위치한
 * 인덱스를 반환한다.
 *
 * @param str 검색할 기계어 문자열
 * @param inst_table 기계어 목록 테이블 주소
 * @param inst_table_length 기계어 목록 테이블의 길이
 * @return 기계어의 인덱스 (해당 기계어가 없는 경우 -1)
 *
 * @details
 * 기계어 목록 테이블에서 특정 기계어를 검색하여, 해당 기계에가 위치한 인덱스를
 * 반환한다. '+JSUB'와 같은 문자열에 대한 처리는 자유롭게 처리한다.
 */
int search_opcode(const char *str, const inst *inst_table[],
                  int inst_table_length) {
    if (str[0] == '+') return search_opcode(str + 1, inst_table, inst_table_length);

    for (int i = 0; i < inst_table_length; i++) {
        if (strcmp(str, inst_table[i]->str) == 0)
            return i;
    }
    return -1;
}

/**
 * @brief 소스코드 명령어 앞에 OPCODE가 기록된 코드를 파일에 출력한다.
 * `output_dir`이 NULL인 경우 결과를 stdout으로 출력한다. 프로젝트 1에서는
 * 불필요하다.
 *
 * @param output_dir 코드를 저장할 파일 경로, 혹은 NULL
 * @param tokens 토큰 테이블 주소
 * @param tokens_length 토큰 테이블의 길이
 * @param inst_table 기계어 목록 테이블 주소
 * @param inst_table_length 기계어 목록 테이블의 길이
 * @return 오류 코드 (정상 종료 = 0)
 *
 * @details
 * 소스코드 명령어 앞에 OPCODE가 기록된 코드를 파일에 출력한다. `output_dir`이
 * NULL인 경우 결과를 stdout으로 출력한다. 명세서에 주어진 출력 예시와 완전히
 * 동일할 필요는 없다. 프로젝트 1에서는 불필요하다.
 */
int make_opcode_output(const char *output_dir, const token *tokens[],
                       int tokens_length, const inst *inst_table[],
                       int inst_table_length) {
    /* add your code */
```

```
    return 0;
}

/**
 * control section 객체를 초기화해주는 함수입니다.
 */
static int init_control_section(control_section* cs, int
start_addr, const char* csect_name) {
    header_record* h;
    end_record* e;

    if (cs == NULL) return -1;
    if ((h = (header_record*)malloc(sizeof(header_record)))
== NULL) return -2;
    if ((e = (end_record*)malloc(sizeof(end_record))) ==
NULL) return -2;
    if (strlen(csect_name) > 6) return -3;

    strcpy(h->program_name, csect_name);
    h->start_addr = start_addr;
    h->program_length = 0;

    e->program_start_addr = start_addr;

    cs->header = h;
    cs->end = e;
    cs->text_lines = 0;
    cs->modification_lines = 0;
    cs->define_lines = 0;
    cs->reference_lines = 0;
    return 0;
}

/**
 * @brief 어셈블리 코드을 위한 패스 2 과정을 수행한다.
 *
 * @param tokens 토큰 테이블 주소
 * @param tokens_length 토큰 테이블 길이
 * @param inst_table 기계어 목록 테이블 주소
 * @param inst_table_length 기계어 목록 테이블 길이
 * @param symbol_table 심볼 테이블 주소
 * @param symbol_table_length 심볼 테이블 길이
 * @param literal_table 리터럴 테이블 주소
 * @param literal_table_length 리터럴 테이블 길이
 * @param obj_code 오브젝트 코드에 대한 정보를 저장하는 구조체 주
소
 * @return 오류 코드 (정상 종료 = 0)
 *
 * @details
 * 어셈블리 코드를 기계어 코드로 바꾸기 위한 패스2 과정을 수행한다.
패스 2의
 * 프로그램을 기계어로 바꾸는 작업은 라인 단위로 수행된다.
 */
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) {
    obj_code->csect_cnt = 0;

    control_section* cs = NULL;
    define_record* def = NULL;
    reference_record* ref = NULL;
```

```
    for (int i = 0; i < tokens_length; i++) {
        token* tok = tokens[i];
        printf("%X\t%s\t%s\n", tok->addr, tok->label, tok->operator);
        int dir = directive(tok->operator);

        // Define 레코드 수정
        if (def != NULL && tok->label != NULL) {
            for (int k = 0; k < def->symbol_cnt; k++)
                if (strcmp(def->symbol[k], tok->label) == 0)
                    def->addr[k] = tok->addr;
        }

        if (dir == -1) {
            // 리터럴인 경우
            /*if (tok->label != NULL && tok->label[0] == '*') {

            }
            else {

            }*/

            // 피연산자가 Reference 레코드에 있다면 Modification 레코드 추가
            if (ref != NULL) {
                for (int j = 0; j < MAX_OPERAND_PER_INST && tok->operand[j] != NULL; j++) {
                }
            }
        }
        else {
            switch (dir) {
            case DIR_START:
            case DIR_CSECT:
                if (cs != NULL && i > 0) {
                    header_record* h = cs->header;
                    h->program_length = tokens[i - 1]->addr; // 새로운 control section이 시작되기 전 이전 토큰의 주소가 해당 csect의 길이가 된다
                }
                if ((cs = (control_section*)malloc(sizeof(control_section))) == NULL)
                    return -2;
                int start_addr = 0;
                if (tok->operand[0] != NULL)
                    start_addr = atoi(tok->operand[0]);
                if (init_control_section(cs, start_addr, tok->label) < 0) return -2;

                obj_code->csects[obj_code->csect_cnt] = cs;
                ++obj_code->csect_cnt;
                break;
            case DIR_EXTREF: {
                if ((ref = (reference_record*)malloc(sizeof(reference_record))) == NULL) return -2;
                ref->symbol_cnt = 0;
                for (int j = 0; j < MAX_OPERAND_PER_INST && tok->operand[j] != NULL; j++) {
                    strcpy(ref->symbol[j], tok->operand[j]);
                    ++ref->symbol_cnt;
                }
                if (cs != NULL) {
                    cs->ref[cs->reference_lines] = ref;
                    ++cs->reference_lines;
                }
                break;
            }
            case DIR_EXTDEF:
```

```
                if ((def =
(define_record*)malloc(sizeof(define_record))) == NULL) return
-2;
                def->symbol_cnt = 0;
                for (int j = 0; j < MAX_OPERAND_PER_INST &&
tok->operand[j] != NULL; j++) {
                    strcpy(def->symbol[j], tok->operand[j]);
                    def->addr[j] = 0; // 0으로 초기화
                    ++def->symbol_cnt;
                }
                if (cs != NULL) {
                    cs->define[cs->define_lines] = def;
                    ++cs->define_lines;
                }
                break;
            case DIR_END:
                if (cs != NULL) {
                    header_record* h = cs->header;
                    h->program_length = tokens[tokens_length -
1]->addr; // 프로그램이 완전히 끝나기 전 전 이전 토큰의 주소가 해당
csect의 길이가 된다
                }
                break;
            }
        }
    }


    return 0;
}

/**
 * @brief 심볼 테이블을 파일로 출력한다. `symbol_table_dir`이
NULL인 경우 결과를
 * stdout으로 출력한다.
 *
 * @param symbol_table_dir 심볼 테이블을 저장할 파일 경로, 혹은
NULL
 * @param symbol_table 심볼 테이블 주소
 * @param symbol_table_length 심볼 테이블 길이
 * @return 오류 코드 (정상 종료 = 0)
 *
 * @details
 * 심볼 테이블을 파일로 출력한다. `symbol_table_dir`이 NULL인 경
우 결과를
 * stdout으로 출력한다. 명세서에 주어진 출력 예시와 완전히 동일할
필요는 없다.
 */
int make_symbol_table_output(const char *symbol_table_dir,
                             const symbol *symbol_table[],
                             int symbol_table_length) {
    FILE* fp;
    if (symbol_table_dir == NULL) {
        fp = stdout;
    }
    else if ((fp = fopen(symbol_table_dir, "wb")) == NULL) {
        return -1;
    }

    for (int i = 0; i < symbol_table_length; i++) {
        symbol* sb = symbol_table[i];
        fprintf(fp, "%s\t%X", sb->name, sb->addr);
        if (sb->rflag != 0) fprintf(fp, "\t%d %s", sb->rflag,
sb->csect_name);
        fprintf(fp, "\n");
    }
```

```
    if (fp != stdout) fclose(fp);

    return 0;
}

/**
 * @brief 리터럴 테이블을 파일로 출력한다. `literal_table_dir`이 NULL인 경우
 * 결과를 stdout으로 출력한다.
 *
 * @param literal_table_dir 리터럴 테이블을 저장할 파일 경로, 혹은 NULL
 * @param literal_table 리터럴 테이블 주소
 * @param literal_table_length 리터럴 테이블 길이
 * @return 오류 코드 (정상 종료 = 0)
 *
 * @details
 * 리터럴 테이블을 파일로 출력한다. `literal_table_dir`이 NULL인 경우 결과를
 * stdout으로 출력한다. 명세서에 주어진 출력 예시와 완전히 동일할 필요는 없다.
 */
int make_literal_table_output(const char *literal_table_dir,
                              const literal *literal_table[],
                              int literal_table_length) {
    FILE* fp;
    if (literal_table_dir == NULL) {
        fp = stdout;
    }
    else if ((fp = fopen(literal_table_dir, "wb")) == NULL) {
        return -1;
    }

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

    if (fp != stdout) fclose(fp);

    return 0;
}

/**
 * @brief 오브젝트 코드를 파일로 출력한다. `objectcode_dir`이 NULL인 경우 결과를
 * stdout으로 출력한다.
 *
 * @param objectcode_dir 오브젝트 코드를 저장할 파일 경로, 혹은 NULL
 * @param obj_code 오브젝트 코드에 대한 정보를 담고 있는 구조체 주소
 * @return 오류 코드 (정상 종료 = 0)
 *
 * @details
 * 오브젝트 코드를 파일로 출력한다. `objectcode_dir`이 NULL인 경우 결과를
 * stdout으로 출력한다. 명세서의 주어진 출력 결과와 완전히 동일해야 한다.
 * 예외적으로 각 라인 뒤쪽의 공백 문자 혹은 개행 문자의 차이는 허용한다.
 */
```

```
 int make_objectcode_output(const char *objectcode_dir,
                            const object_code *obj_code) {
    FILE* fp;
    if (objectcode_dir == NULL) {
        fp = stdout;
    }
    else if ((fp = fopen(objectcode_dir, "wb")) == NULL) {
        return -1;
    }

    for (int i = 0; i < obj_code->csect_cnt; i++) {
        control_section* cs = obj_code->csects[i];
        header_record* h = cs->header;
        fprintf(fp, "H%-6s%06X%06X\n", h->program_name, h-
>start_addr, h->program_length);

        for (int j = 0; j < cs->define_lines; j++) {
            define_record* d = cs->define[j];
            fprintf(fp, "D");
            for (int j = 0; j < d->symbol_cnt; j++)
                fprintf(fp, "%-6s%06X", d->symbol[j], d-
>addr[j]);
            fprintf(fp, "\n");
        }

        for (int j = 0; j < cs->reference_lines; j++) {
            reference_record* r = cs->ref[j];
            fprintf(fp, "R");
            for (int j = 0; j < r->symbol_cnt; j++)
                fprintf(fp, "%-6s", r->symbol[j]);
            fprintf(fp, "\n");
        }

        for (int j = 0; j < cs->text_lines; j++) {
            text_record* t = cs->text[j];
            fprintf(fp, "T%06X%02X%s\n", t->start_addr, t-
>bytes_length, t->obj);
        }

        for (int j = 0; j < cs->modification_lines; j++) {
            modification_record* m = cs->modi[j];
            fprintf(fp, "M%06X%02X%c%s\n", m->start_addr, m-
>length, m->m_flag, m->symbol);
        }

        end_record* e = cs->end;
        fprintf(fp, "E");
        if (e->program_start_addr != -1)
            fprintf(fp, "%06X", e->program_start_addr);
        fprintf(fp, "\n");
    }

    if (fp != stdout) fclose(fp);

    return 0;
 }
```