

묵찌빠 프로그래밍 언어 제작 보고서

컴파일러 수업

이재성 교수

2019038025 소프트웨어학부 강희수

목차

1. 설계 및 구현

1.1 개요

- 영감

- 프로그래밍 언어의 목표

- 타겟 사용자

1.2 언어의 특징

- 문법 및 구문

- 변수 선언 및 자료형

- 제어 구조(조건문,반복문)

- 함수 구현

1.3 컴파일러의 설계

- 컴파일러의 구현 방법(Lexer, Parser)

2. 결과

2.1 실행예시 1,2,3

3. 토론

3.1 개발 과정 문제점 및 이슈

3.2 향후 개선 및 추가 기능

4. 결론

4.1 프로젝트를 마무리하며

- 프로젝트 주요 성과 및 배운점

- 후기

1. 설계 및 구현

1.1 개요

-영감



최근 youtube 및 sns를 강타한 오페라 <리타>의 한 장면이다.

이 내용은 오페라와 전혀 어울리지 않는 목찌빠를 소재로 장난스럽고 무겁지 않은 분위기로 많은 웃음을 자아내며 인기를 끌었다.

그래서 여기에 나온 대화들을 소재로, 프로그래밍 언어를 만들어본다면 재밌지 않을까? 라는 생각에 도전하게 되었다.

-프로그래밍 언어의 목표-

1.변수구현 및 사칙연산 구현

int float char형

+ , / , * , -

2.조건문 구현

3.출력 (print) 구현

4.함수 구현

.. 등등

-타겟사용자-

목찌빠를 좋아하는 사람들

1.2 언어의 특징

문법 및 구문

이 프로그래밍 언어는 간결하고 직관적인 문법을 목표로 설계되었다.

▶ 변수 선언 :

정수형 : “가위”

실수형 : “바위”

문자열형: “보”

할당 : "제로" ex)a 제로 3 => a=3

▶ 제어 구조(조건문,반복문)

-조건문

“난목찌빠로(IF) 조건(변수 혹은 비교문) 다녀왔단(question) 명령문1 니놈을(ELSE) 명령문2”
조건 만족 시 명령문1을 실행하며, 아닌 경우 명령문 2을 실행한다.

-반복문

“다시해(WHILE) 조건(변수 혹은 반복할 횟수) 명령문”
명령문을 n번만큼 실행한다.

▶ 함수 구현

함수 정의 : “정정[FUNC] 함수이름 게임[DEF] 명령문들 한[FEND]”

함수이름 및 해당 명령문들을 입력하여, 함수를 정의한다.

함수 호출 : “당당[CALL] 함수이름”

함수의 이름을 사용하여, 함수를 호출한다.

▶ 사칙연산

+ :묵, - :찌빠, * :찌, /(나눠셈) :빠

1.3 컴파일러 설계

Lexer file

사용할 키워드를 정의해 놓았다. (lexer는 소스코드를 읽어들이어 의미 단위인 토큰으로 분리한다)

```
"사랑" { yyval.sval = _strdup("I LOVE"); return PSTRING; }
"니를" { return ELS; }
"이거" { yyval.sval = _strdup("I LOVE LEE JAE SUNG PROFESSOR"); return PSTRING; }
"(" { return LBRACKET; }
")" { return RBRACKET; }

"가문의이름" { yyval.sval = _strdup("THANK YOU SO MUCH!"); return PSTRING; }
"제출" { return ASSIGN; }
"종이리" { return END; }

"가온" { return INT; }
"부" { return FLOAT; }
"보" { return CHAR; }
"문" { return ELUSI; }
"피" { return MULT; }
"로" { return DIV; }
"리" { return FUNC; }
"리" { return CALL; }
"리" { return FEND; }
"리" { return DEF; }
"리" { return ADD; }
"리" { return SUB; }

"리" { return PRINT; }
"리" { return PRINT_END; }
"리" { yyval.ival = 3; return NUMBER; }
"리" { return IVALUE; }
"리" { return FVALUE; }
"리" { return SVALUE; }
"리" { return BSN; }
">" { return BIG; }
">" { return BIGGER; }
"<" { return SMALL; }
"<=" { return SMALLER; }
"다시" { return WHILE; }
\[^\"]*\[^\"]* { yyval.sval = _strdup(yytext+1); yyval.sval[strlen(yyval.sval)-1] = 0; return RSTRING; }
[0-9]+ { yyval.ival = atoi(yytext); return NUMBER; }
[a-zA-Z][a-zA-Z0-9]* { yyval.sval = _strdup(yytext); return IDENTIFIER; }
; { return EOL; }
[ \t] { /* 공백 무시 */ }
. { /* 감지 안되는 문자는 무시한다. */ }

%%

int yywrap(void) {
    return 1;
}
```

주목할만한 사항

```
\[^\"]*\[^\"]* { yyval.sval = _strdup(yytext+1); yyval.sval[strlen(yyval.sval)-1] = 0; return RSTRING; }
```

▶ 해당 키워드는, 큰따옴표 두 개 사이에 오는 모든 것을 문자열로 해석한다는 뜻이다.

```
[a-zA-Z][a-zA-Z0-9]* { yyval.sval = _strdup(yytext); return IDENTIFIER; }
```

▶ 주로 변수 이름에 사용되며, 영어로 사용되는 모든 문자열을 의미한다. 처음에 숫자는 올 수 없다.

Parser file

```
4 #include <stdlib.h>
5 #include <string.h>
6 #include <stdarg.h>
7
8 extern int yylex(void);
9 extern int yyparse(void);
10 extern FILE *yyin;
11 extern int yylineno;
12 extern char *yytext;
13
14 void yyerror(const char *s);
15 int goto_line = -1;
16
17 typedef enum { STMT_VAR_DECL, STMT_ASSIGN, STMT_ASSIGN_S, STMT_PRINT_N, STMT_PRINT_F, STMT_PRINT_S, STMT_IF, STMT_ST
18
19 typedef struct statement {
20     stmt_type type;
21     union {
22         struct {
23             char *var;
24             int ivalue;
25             float fvalue;
26             char *st;
27             int count;
28         } assignment;
29         struct {
30             char *var;
31             int Iv, Fv, Cv;
32         } var_decl;
33         struct {
34             int ivalue;
```

(※내용이 너무 많아, 일부만 올립니다.)

▷ 어휘 분석기로부터 받은 토큰들을 사용하여 구문 분석을 하는 단계

주요 구성사항

1. 프로그램의 시작

```
int main(int argc, char **argv) {
    if (argc > 1) {
        FILE *file = fopen(argv[1], "r");
        if (!file) {
            fprintf(stderr, "Could not open file: %s\n", argv[1]);
            return 1;
        }
        yyin = file;
    }
    //yydebug = 1; // Enable debug output
    yyparse();
    return 0;
}

program:
    START statements END { printf("Parsing completed.\n"); process_statements($2); }
;
```

▷ main을 통해 시작하며, 입력파일을 열어 yyparse 함수를 호출하여 시작한다. START 와 END 토큰을 사용해서 시작과 끝을 나타낸다. process_statements 함수를 실행해준다.

2. statements

```

statements:
    { $$ = NULL; }
    | statements statement { add_statement(&$$, $2); }
    | statements EOL { $$ = $1; /* 빈 줄 무시 */ }
;

statement:
    variable_declaration
    | assignment
    | print_statement
    | if_statement
    | string_statement
    | function_definition
    | function_call

```

statement는, 가장 기본적인 문장으로 statements를 통해 여러 문장들을 나열할 수 있으며, 각각의 statement는 다양한 유형의 구문으로 표현이 가능하다. add_statement 함수를 사용하여, statement를 추가한다.

3. 변수 선언

```

typedef struct {
    char *name;
    int ivalue;
    float fvalue;
    char *str_value;
    int Iv, Fv, Cv;
} variable;

variable vars[100]; // 최대 100개의 변수 저장
int var_count = 0;

variable_declaration:
    INT IDENTIFIER EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_VAR_DECL;
        stmt->data.var_decl.var = __strdup($2);
        stmt->data.var_decl.Iv = 1;
        vars[var_count].name = stmt->data.var_decl.var;
        vars[var_count].ivalue = 0;
        vars[var_count].Iv = 1;
        var_count++;
        $$ = stmt;
        free($2);
    }
    FLOAT IDENTIFIER EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_VAR_DECL;
        stmt->data.var_decl.var = __strdup($2);
        stmt->data.var_decl.Fv = 1;
        vars[var_count].name = stmt->data.var_decl.var;
        vars[var_count].fvalue = 0.0;
        vars[var_count].Fv = 1;
        var_count++;
        $$ = stmt;
        free($2);
    }
    CHAR IDENTIFIER EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_VAR_DECL;
        stmt->data.var_decl.var = __strdup($2);
        stmt->data.var_decl.Cv = 1;
        vars[var_count].name = stmt->data.var_decl.var;
        vars[var_count].str_value = NULL;
        vars[var_count].Cv = 1;
        var_count++;
        $$ = stmt;
        free($2);
    }

```

전역 variable 구조체 배열 var을 사용하여, 해당 변수의 타입과 이름을 저장한다. 구조체 정수형 변수 Iv,Fv,Cv를 이용하여, 해당 타입에 맞게 1을 설정시켜주었다

4. 변수 할당

```
assignment:
    IDENTIFIER ASSIGN expression EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_ASSIGN;
        stmt->data.assignment.var = _strdup($1);
        for (int i = 0; i < var_count; i++) {
            if (strcmp(vars[i].name, $1) == 0) {
                if (vars[i].Iv) {
                    stmt->data.assignment.ivalue = $3;
                    vars[i].ivalue = $3;
                } else if (vars[i].Fv) {
                    stmt->data.assignment.fvalue = (float)$3;
                    vars[i].fvalue = (float)$3;
                } else {
                    yyerror("Type error: variable type not defined.");
                }
                break;
            }
        }
        $$ = stmt;
        free($1);
    }
    IDENTIFIER ASSIGN RSTRING EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_ASSIGN_S;
        stmt->data.assignment.var = _strdup($1);
        stmt->data.assignment.st = _strdup($3);
        for (int i = 0; i < var_count; i++) {
            if (strcmp(vars[i].name, $1) == 0) {
                vars[i].str_value = _strdup($3);
                break;
            }
        }
        $$ = stmt;
        free($1);
    }
};
```

변수 할당은 IDENTIFIER=expression으로 이루어지며, 정수 실수 및 문자열 할당을 처리한다. 모든 변수들은 var_count를 통해 0부터 99까지 var[] 배열안에 정의되고, 해당 값들을 통해 변수의 타입을 파악하여, 해당 데이터에 알맞은 값을 넣어주게 된다.

RSTRING은, 큰 따옴표 안에 있는 모든 문자열을 가르키는 토큰으로, 사칙연산 및 수를 다루는 EXPRESSION과는 다르므로 따로 정의를 해 주었다.

5. expression

```
expression:
    expression ADD term { $$ = $1 + $3; }
    | expression SUB term { $$ = $1 - $3; }
    | term
;

term:
    term MULT factor { $$ = $1 * $3; }
    | term DIV factor { $$ = $1 / $3; }
    | factor
;
```



```

factor:
    variable PLUS1 { $$ = $1 + 1; }
    | variable
    | number
;

variable:
    IDENTIFIER {
        int i;
        for (i = 0; i < var_count; i++) {
            if (strcmp(vars[i].name, $1) == 0) {
                if (vars[i].Iv) { $$ = vars[i].ivalue; }
                else if (vars[i].Fv) { $$ = (int)vars[i].fvalue; }
                else if (vars[i].Cv) { $$ = (int)vars[i].str_value; }
                else { yyerror("type error: variable type not defined"); }
                break;
            }
        }
        if (i == var_count) {
            yyerror("Undefined variable");
            $$ = 0;
        }
        free($1);
    }
;

```

expression은, 변수 혹은 계산식을 담당하는 곳으로, 사칙연산 시 우선 순위 구현을 위해 term 및 factor를 추가로 구현 하였다.

변수가 나온 경우, 해당 identifier와, vars에 저장된 변수의 이름들을 대조하여, 변수를 찾고 그 값을 이용한다.

6. string_statement

```

string_statement:
    PSTRING EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_STRING;
        stmt->data.string = $1;
        $$ = stmt;
    }
;

```

pstring 값은, lexer에서 제가 따로 저장한 문자열이다.

7. print

```
print_statement:
    PRINT IVALUE expression PRINT_END EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_PRINT_N;
        stmt->data.print.ivalue = $3;
        $$ = stmt;
    }

    | PRINT FVALUE expression PRINT_END EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_PRINT_F;
        stmt->data.print.fvalue = $3;
        $$ = stmt;
    }

    | PRINT SVALUE IDENTIFIER PRINT_END EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_PRINT_S;
        for (int i = 0; i < var_count; i++) {
            if (strcmp(vars[i].name, $3) == 0) {
                stmt->data.print.s = _strdup(vars[i].str_value);
                break;
            }
        }
        $$ = stmt;
    }

    | PRINT RSTRING PRINT_END EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_PRINT_S;
        stmt->data.print.s = _strdup($2);
        $$ = stmt;
    }
```

print는 PRINT VALUE타입 PRINT_END EOL 토큰들을 사용하며 int형과 float형 char형 출력을 구분하기 위한 value 타입 토큰을 사용했다.

stmt->type또한 타입에 맞게 구분해 놓았으며, 각각의 값은 print.value 값에 저장된다.

8. if

```
if_statement:
    IF expression QUESTION statement ELS statement EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_IF;
        stmt->data.if_stmt.condition = $2;
        stmt->data.if_stmt.if_stmt = (statement_list *)malloc(sizeof(statement_list));
        stmt->data.if_stmt.if_stmt->stmt = $4;
        stmt->data.if_stmt.if_stmt->next = NULL;
        stmt->data.if_stmt.else_stmt = (statement_list *)malloc(sizeof(statement_list));
        stmt->data.if_stmt.else_stmt->stmt = $6;
        stmt->data.if_stmt.else_stmt->next = NULL;
        $$ = stmt;
    }

    | IF expression bigger_or_smaller expression statement ELS statement EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_IF_COMPARE;
        stmt->data.if_stmt.condition = $2;
        stmt->data.if_stmt.big_small = $3;
        stmt->data.if_stmt.opcondition = $4;
        stmt->data.if_stmt.if_stmt = (statement_list *)malloc(sizeof(statement_list));
        stmt->data.if_stmt.if_stmt->stmt = $5;
        stmt->data.if_stmt.if_stmt->next = NULL;
        stmt->data.if_stmt.else_stmt = (statement_list *)malloc(sizeof(statement_list));
        stmt->data.if_stmt.else_stmt->stmt = $7;
        stmt->data.if_stmt.else_stmt->next = NULL;
        $$ = stmt;
    }
;
```

조건문의 시작은 IF 토큰으로 시작한다. ELS를 통해 else문을 표현한다.

첫 번째 식은 expression이 1일 경우, 앞에 먼저 나온 statement를 실행한다, 이를 위해 statement_list의 포인터를 할당을 새로 해 준다.

두 번째 식은 expr 과 expr을 비교하여 해당 식이 옳으면 앞선 식, 아니면 두 번째 식을 실행한다. 앞선 방식과 마찬가지로 포인터 할당을 해주었으며, 이후 식은 process_statements 함수에서 처리된다.

9. function

```
function_call:
    CALL IDENTIFIER EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_FUNC_CALL;
        stmt->data.func_call.name = $2;
        $$ = stmt;
    }
;

function_definition:
    FUNC IDENTIFIER DEF statements FEND {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_FUNC_DEF;
        stmt->data.func_def.name = $2;
        stmt->data.func_def.body = $4;
        $$ = stmt;
    }
;
```

FUNC 토큰을 통해 함수 이름을 정의해준다.

CALL 토큰을 통해, 함수를 불러들인다.

이후 식은 process_statement에서 처리된다.

10.STMT

처음 PROGRAM에서 process_statements를 실행하는 것을 보았을 것이다. 이는 모든 타입의 statement들을 관리하며 실행해주는 함수이다.

```
void process_statements(statement_list *list) {
    statement_list *current = list;
    while (current != NULL) {
        statement *stmt = current->stmt;
        switch (stmt->type) {
            case STMT_VAR_DECL:
                printf("Variable declaration: %s\n", stmt->data.var_decl.var);
                break;
            case STMT_ASSIGN:
                for (int i = 0; i < var_count; i++) {
                    if (strcmp(vars[i].name, stmt->data.assignment.var) == 0) {
                        if (vars[i].Iv == 1) {
                            vars[i].Ivalue = stmt->data.assignment.Ivalue;
                            printf("Assignment: %s = %d\n", stmt->data.assignment.var, stmt->data.assignment.Ivalue);
                        } else if (vars[i].Fv == 1) {
                            vars[i].Fvalue = stmt->data.assignment.Fvalue;
                            printf("Assignment: %s = %f\n", stmt->data.assignment.var, stmt->data.assignment.Fvalue);
                        }
                        break;
                    }
                }
                break;
            case STMT_ASSIGN_S:
                printf("Assignment: %s = %s\n", stmt->data.assignment.var, stmt->data.assignment.st);
                for (int i = 0; i < var_count; i++) {
                    if (strcmp(vars[i].name, stmt->data.assignment.var) == 0) {
                        vars[i].str_value = stmt->data.assignment.st;
                        break;
                    }
                }
                break;
            case STMT_PRINT_N:
                break;
        }
        current = current->next;
    }
}
```

```

void process_statements(statement_list *list) {
    statement_list *current = list;
    while (current != NULL) {
        statement *stmt = current->stmt;
        switch (stmt->type) {
            case STMT_VAR_DECL:
                break;
            case STMT_ASSIGN:
                for (int i = 0; i < var_count; i++) {
                    if (strcmp(vars[i].name, stmt->data.assignment.var) == 0) {
                        if (vars[i].Iv == 1) {
                            vars[i].ivalue = stmt->data.assignment.ivalue;
                        } else if (vars[i].Fv == 1) {
                            vars[i].fvalue = stmt->data.assignment.fvalue;
                        }
                    }
                    break;
                }
                break;
            case STMT_ASSIGN_S:
                for (int i = 0; i < var_count; i++) {
                    if (strcmp(vars[i].name, stmt->data.assignment.var) == 0) {
                        vars[i].str_value = stmt->data.assignment.st;
                        break;
                    }
                }
                break;
            case STMT_PRINT_N:
                printf("%d", stmt->data.print.ivalue);
                break;
            case STMT_PRINT_F:
                printf("%f", stmt->data.print.fvalue);
                break;
            case STMT_PRINT_S:
                printf("%s", stmt->data.print.s);
                break;
            case STMT_PRINT_BS:
                printf("\n");
                break;
            case STMT_IF:
                if (stmt->data.if_stmt.condition) {
                    process_statements(stmt->data.if_stmt.if_stmt);
                } else {
                    process_statements(stmt->data.if_stmt.else_stmt);
                }
                break;
            case STMT_IF_COMPARE:
                if (stmt->data.if_stmt.big_small==1) {
                    if (stmt->data.if_stmt.condition > stmt->data.if_stmt.opcondition){
                        process_statements(stmt->data.if_stmt.if_stmt);
                    } else {
                        process_statements(stmt->data.if_stmt.else_stmt);
                    }
                } else {
                    if (stmt->data.if_stmt.condition < stmt->data.if_stmt.opcondition){
                        process_statements(stmt->data.if_stmt.if_stmt);
                    } else {
                        process_statements(stmt->data.if_stmt.else_stmt);
                    }
                }
                break;
        }
        current = current->next;
    }
}

```

process_statement의 식은 이렇게 되어있으며, 각각의 타입을 동작함에 따라 개발자가 직접 정의한 stmt_type을 지정해주어, 알맞은 작업을 수행하도록 한다. 이는, 작업 수행시 각각의 사용할 데이터 공간을 직접 만들고 저장함을 위함이고, case를 분류해놓음에 따라 유지보수가 편하여 선택했다.

```

typedef struct statement {
    stmt_type type;
    union {
        struct {
            char *var;
            int ivalue;
            float fvalue;
            char *st;
            int count;
        } assignment;
        struct {
            char *var;
            int Iv, Fv, Cv;
        } var_decl;
        struct {
            int ivalue;
            float fvalue;
            char *s;
        } print;
        struct {
            int condition;
            struct statement *if_stmt;
            struct statement *else_stmt;
        } if_stmt;
        char *string;
        struct {
            char *name;
            struct statement_list *body;
        } func_def;
        struct {
            char *name;
        } func_call;
    } data;
} statement;

```

위 그림은, 각각의 타입에 맞는 데이터를 저장하기 위한, union , struct 공간을 만든 것이다.

이제부터 STATEMENT들을 설명해 보겠다.

먼저 가장 기본적인 statement 구조체에 대해 알아보자.

statement들 또한 연결리스트로 연결되어 있게 구현 하였다.

```

typedef struct statement_list {
    statement *stmt;
    struct statement_list *next;
} statement_list;

void add_statement(statement_list **list, statement *stmt) {
    statement_list *new_node = (statement_list *)malloc(sizeof(statement_list));
    new_node->stmt = stmt;
    new_node->next = NULL;

    if (*list == NULL) {
        *list = new_node;
    } else {
        statement_list *current = *list;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = new_node;
    }
}

```


add_statement는 statements:에서 사용되는 함수로, statement들이 증가될 때마다 하나씩 추가를 하게 된다. 각각을 연결리스트로 구현하여 current를 통해, 연결리스트에 마지막 부분에 추가한다.

0. ASSIGN

```
case STMT_ASSIGN:
    for (int i = 0; i < var_count; i++) {
        if (strcmp(vars[i].name, stmt->data.assignment.var) == 0) {
            if (vars[i].Iv == 1) {
                vars[i].ivalue = stmt->data.assignment.ivalue;
            } else if (vars[i].Fv == 1) {
                vars[i].fvalue = stmt->data.assignment.fvalue;
            }
            break;
        }
    }
    break;
case STMT_ASSIGN_S:
    for (int i = 0; i < var_count; i++) {
        if (strcmp(vars[i].name, stmt->data.assignment.var) == 0) {
            vars[i].str_value = stmt->data.assignment.st;
            break;
        }
    }
    break;
```

들어온 변수의 값을 Iv ,Fv로 나누어 해당 값에 맞는 자료형에 따라 값을 저장한다. string의 경우 Iv Fv와는 다르게 들어온 식이 string값이므로, 같은 case에 넣을 필요가 없었다.

1. if문

```
case STMT_IF:
    if (stmt->data.if_stmt.condition) {
        process_statements(stmt->data.if_stmt.if_stmt);
    } else {
        process_statements(stmt->data.if_stmt.else_stmt);
    }
    break;
case STMT_IF_COMPARE:
    if (stmt->data.if_stmt.big_small==1) {
        if(stmt->data.if_stmt.condition > stmt->data.if_stmt.opcondition){
            process_statements(stmt->data.if_stmt.if_stmt);
        }else{
            process_statements(stmt->data.if_stmt.else_stmt);
        }
    } else if(stmt->data.if_stmt.big_small==2){
        if(stmt->data.if_stmt.condition >= stmt->data.if_stmt.opcondition){
            process_statements(stmt->data.if_stmt.if_stmt);
        }else{
            process_statements(stmt->data.if_stmt.else_stmt);
        }
    } else if(stmt->data.if_stmt.big_small==3){
        if(stmt->data.if_stmt.condition < stmt->data.if_stmt.opcondition){
            process_statements(stmt->data.if_stmt.if_stmt);
        }else{
            process_statements(stmt->data.if_stmt.else_stmt);
        }
    }
    else if(stmt->data.if_stmt.big_small==4){
        if(stmt->data.if_stmt.condition <= stmt->data.if_stmt.opcondition){
            process_statements(stmt->data.if_stmt.if_stmt);
        }else{
            process_statements(stmt->data.if_stmt.else_stmt);
        }
    }
    break;
```

해당 IF 조건문을 나타내며, condition의 값에 따라 구조체에 저장된 수행해야 될 일을 수행한다. 아래 compare는 중간에 비교문이 들어왔을 경우, 해당 기호에 값에 따라 알맞은 if문을 실행해주는 것으로

```
bigger_or_smaller:
    BIG {$$ = 1;}
    BIGGER{$$ = 2;}
    SMALL{$$ = 3;}
    SMALLER{$$ = 4;}
```

해당 값에 알맞게 > >= < <=을 구현하였다.

2. function

```
case STMT_FUNC_DEF:
    add_function(stmt->data.func_def.name, stmt->data.func_def.body);
    printf("Function definition: %s\n", stmt->data.func_def.name);
    break;
case STMT_FUNC_CALL:
    printf("Function call: %s\n", stmt->data.func_call.name);
    statement_list *func_body = find_function(stmt->data.func_call.name);
    if (func_body != NULL) {
        process_statements(func_body);
    } else {
        printf("Error: Function %s not defined\n", stmt->data.func_call.name);
    }
    break;
```

func은 add_function 및 find_function이라는 함수를 사용한다.

func은 자체의 구조체를 갖고 있으며,

```
typedef struct function {
    char *name;
    statement_list *body;
    struct function *next;
} function;
```

연결리스트로 연결된 모습을 보여준다.

함수의 이름인 name과 수행해야하는 일 *body를 갖고 있다.

```
void add_function(char *name, statement_list *body) {
    function *new_func = (function *)malloc(sizeof(function));
    new_func->name = _strdup(name);
    new_func->body = body;
    new_func->next = functions;
    functions = new_func;
}
```

함수 추가하는 방식으로, 이름 및 body를 넣어준다.

이제 함수를 찾는 방식에 대해 보겠다.

```
statement_list *find_function(char *name) {
    function *current = functions;
    while (current != NULL) {
        if (strcmp(current->name, name) == 0) {
            return current->body;
        }
        current = current->next;
    }
    return NULL;
}
```

해당 함수를 찾는 방법으로, 이름을 통해 연결리스트를 순회하며 찾는다.

위 함수를 찾았다면, CALL함수에서는 해당 함수의 Body 즉 사용자가 작성하여 저장된 statement를 process_statment 함수를 통해 다시 재 실행시킨다.

3.반복문

```
while_statement:
    WHILE expression statement EOL {
        statement *stmt = (statement *)malloc(sizeof(statement));
        stmt->type = STMT_WHILE;
        stmt->data.while_stmt.times = $2;
        stmt->data.while_stmt.while_stmt = (statement_list *)malloc(sizeof(statement_list));
        stmt->data.while_stmt.while_stmt->stmt = $3;
        stmt->data.while_stmt.while_stmt->next = NULL;
        $$ = stmt;
    }
;
```

해당 while 토큰과 expression(실행할 횟수) statement(실행할 명령문) 을통해 사용자에게 값을 받는다.

받은값은 구조체를 통해 데이터가 저장되며, stmt는 새로 실행시켜줄 명령문이니, stmt의 포인터를 할당하여 준다.

```
case STMT_WHILE:
    for(int k=0;k<stmt->data.while_stmt.times;k++){
        process_statements(stmt->data.while_stmt.while_stmt);
    }
    break;
```

그이후 process_statment 함수를 통해, 실행할 횟수동안 반복하며 해당 명령문을 일정 횟수 실행한다.

2. 결과

2.1 변수 할당 및 사칙연산 시행

```
C:\Users\강희수\Downloads\homework>
난대학시절 //start
가위 a; // int a;
a 제로 3; //a=3;
가위 d;
d 제로 4;
남자는역시 눈물 a 목 d 주먹; //PRINT IVALUE a + d PRINT_END;

인정; //Wn

남자는역시 눈물 a 찌빠 d 주먹; // a-d 출력

인정;

바위 b; //float
b 제로 4;
바위 c;
c 제로 3;
남자는역시 콧물 b 찌 c 주먹; //b*c

인정;

보 f; //char
f 제로 "hellow my name is Hee Su"; //str 할당
남자는역시 빼주마 f 주먹; // str출력 (빼주마 -> SVALUE)

인정;

높이리

Parsing completed.
7
-1
12.000000
hellow my name is Hee Su
C:\Users\강희수\Downloads\homework\homework>
```

간단한 사칙연산 및 변수 출력을 보여줬다

여기서 나온 인정은 개행문자로

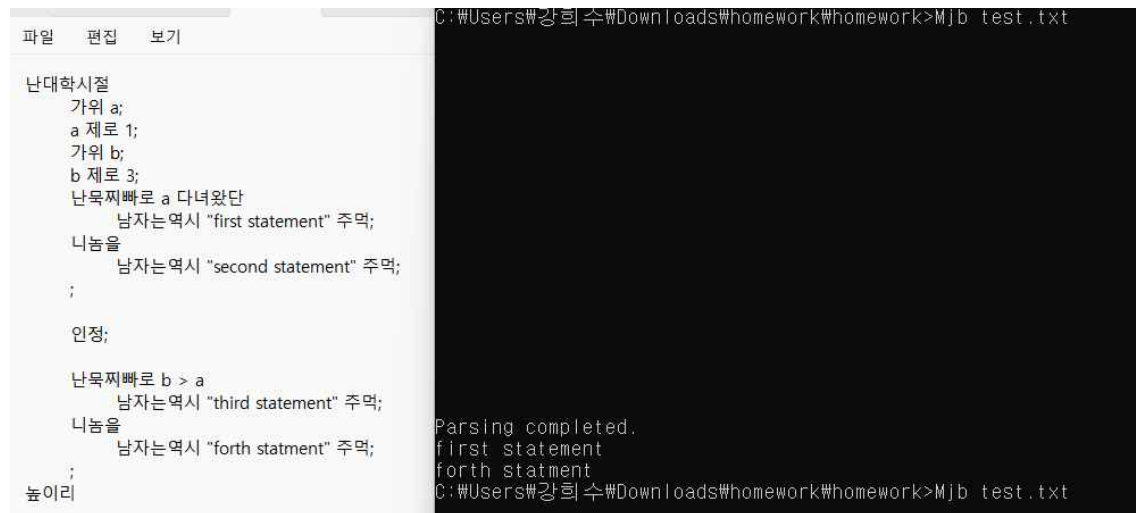
BackSlash:

```
BSN EOL{
    statement *stmt = (statement *)malloc(sizeof(statement));
    stmt->type = STMT_PRINT_BS;
    $$ = stmt;
}

;
case STMT_PRINT_BS:
    printf("\n");
    break;
```

를 통해, 간단하게 한칸 내림을 표현하였다.

2.2 조건문 및 반복문 사용하기



```
파일 편집 보기 C:\Users\강희수\Downloads\homework\homework>Mjb test.txt

난대학시절
가위 a;
a 제로 1;
가위 b;
b 제로 3;
난목찌빠로 a 다녀왔단
    남자는역시 "first statement" 주먹;
니놈을
    남자는역시 "second statement" 주먹;
;

인정;

난목찌빠로 b > a
    남자는역시 "third statement" 주먹;
니놈을
    남자는역시 "forth statment" 주먹;
;

높이리

Parsing completed.
first statement
forth statement
C:\Users\강희수\Downloads\homework\homework>Mjb test.txt
```

조건문 사용시,

IF(난목찌빠로) EXPRESSION(a) QUESTION(다녀왔단)

STATEMENT:(print)

ELSE

STATEMENT;

를했다.

비교문의 경우 QUESTION을 사용하지 않는다.

처음 값은 1 즉 TRUE 이므로 앞선값을 실행한다.

뒤에 값 또한 $b > a$ 이므로 앞선 값을 실행했다.



```
난대학시절
가위 a;
a 제로 0;
가위 b;
다시해 10
    남자는역시 "hellow " 주먹;
;

높이리

줄 6, 열 22 74자 100% Windows (CRLF) UTF-8

Parsing completed.
hellow hellow hellow hellow hellow hellow hellow hellow hellow hellow
C:\Users\강희수\Downloads\homework\homework>
```

10번동안 hellow문을 반복하는 반복문을 만들었다.

다시해 while 횟수 10

3. 함수 사용

```
파일  편집  보기

난대학시절
    가위 a;
    a 제로 3;
    가위 b;
    b 제로 4;

    정정 function 게임
        남자는역시 "hellow" 주먹;
        a 제로 a 목 b;
        인정;
        남자는역시 눈물 a목b 주먹;
    한

    당당 function;
    인정;
    남자는역시 눈물 a 주먹;
    높이리
```

Parsing completed.
hellow
11
7

내용을 보면, a,b를 선언 후, function 안에서 $a=a+b$ 이후 $a+b$ 의 값을 출력하는 함수 function을 만들었다. 이후 당당(call)을 통해 function을 부르니, 해당 값이 알맞게 나오는 것을 볼 수 있고, a의 값을 출력하였음을 볼 수 있다.

정정은 FUNCTION 토큰을 반환하며, 게임은 DEF를 반환한다, 이후 statements들을 포함하며, 한은 function이 끝남을 의미한다.

.

3. 토론

3.1 문제점

정말 문제점이 셀수도 없이 많았다..

대부분의 문제점을 해결하기 위해 statement의 structure를 만든 것이다.

1 statement

초기버전의 경우, 각각의 타입에 맞는 해당 statement에서 즉시 해당 작업을 실행하였지만, 변수의 원하는 값들이 제대로 안나오게 되거나, 해당 변수의 값을 찾기 너무 힘든 경우가 많았다.

또한 statement들을 늘리게 되면서, 한눈에 보기 힘들고, 어디서 에러가 났는지 쉽게 파악하지 못해, 결국 statement 구조체 및 함수를 사용하였다.

그리고 해당 작업을 불러오는 for 혹은 function의 경우, goto 혹은 jump구현이 너무 어려웠기에, process_statements를 사용하여, 알맞은 statements들을 불러올 수 있게 되었다.

초기엔 yyparse()를 추가로 넣는 괴상한 짓을 했었지만.. 이는 아예 처음부터 실행되어 굉장히 난감했다.

2 lexer의 \n

그리고 시간을 가장 많이 잡아먹었던 문제점은 \n 이였다

기존 lexer파일에서 문장의 끝을 나타내기 위해 EOL을 \n으로 했었다. 하지만 build 까지는 무사히 되어도, 자꾸 syntax 오류가 나길래, 대략 순수 4~5시간만 이 오류를 해결하기 위해 용을 썼다.. 하지만 정확한 답변이 나오지 않았고, 혹시나 하여, 마지막 EOL을 \n에서 ;로 바꾸니 오류가 사라졌다. 줄이 바뀌면서 \n이 많은 오류를 냈던 것 같다.

3.2 추후 개선사항

반복문을 실행 시, 해당 stmt를 불러올 때, 데이터의 값 또한 다시 기존으로 유지되어 불러오는 것 같았다. 이는 $a=a+1$ 을 10번 하더라도 $a=1$ 임을 나타낸다. 추후 방법을 알아내어 고쳐보겠다.

print 시 문자열과 숫자는 따로따로 호출하여야 하는 불편함이 존재한다 추후에 시간이 된다면 둘을 한번에 출력 가능하도록 하겠다.

사실 문법이 굉장히 괴랄하다.. 재미를 위해 만들었지만 솔직히 가독성이 정말 떨어지는 것 같다. 나중에 lexer 파일을 고쳐, 이를 변경하겠다.

너무 간단한 문법만 구현해 놓은 것 같다. 구조체, 객체지향 등등 초기 구상은 정말 다양한 걸 해보고 싶었으나, 시간 및 나의 역량 부족 문제로 다가가지 못했다..

4. 프로젝트를 끝마치며

4.1 주요 성과 및 배운점

프로젝트의 주요 성과는 사칙연산 구현, 함수 구현, 반복문 구현, 조건문 구현을 한 것이다.

프로젝트에서 배운점 :

컴파일러 속에서, 다양한 연산들이 어떤 메커니즘과 순서로 움직이는지에 대한 대강적인 판단이 되었고, 어떻게 해야 최대한 메모리를 안 쓰고 코드를 짤 수 있을지에 대한 감이 잡힌 것 같다.

4.2 후기

사실 대학교에 처음 들어와 프로그래밍 언어를 처음 배울 때, 마음 한 구석에, 이 단어를써서 해당 작업이 나오게 하려면 가장 초기가 된 언어는 무엇일까? 어떻게 구현을 했을까? 라는 의문이 계속 남았었다. 이 프로젝트를 통해, 물론 C언어의 도움을 많이 받긴 했지만, 조금의 의문은 풀린 것 같았다.

너무너무 재밌었다. 솔직히 누가 묵찌빠 언어 같은 것을 만들려고 하겠는가? 내가 상상하고 구상한 언어들이 실제로 프로그래밍 언어로 바뀌어 구현된다는 것 자체가 남자의 꿈과 희망 로망을 채워주는 것 같아, 묵찌빠 언어를 생각해 냈을땐 정말 설렘다. 물론 프로젝트 진행하며 갖은 오류 및 시간을 그렇게 쓰기 전까진 ^^..

우리가 배우는 근간의 언어를 하나 직접 만들어 냈다는 것이 너무 신기했다. 이런 방식으로 java , python 같은 파생 언어들이 나온것인 것 같음을 알게 되었고, 너무 너무너무 좋은 경험이었던 것 같다!

이상 2019038025 소프트웨어학부 강희수였습니다 감사합니다!