Programming Language hw3

B811056 노이진

April 2021

코드가 가로로 길어서 페이지의 가로 길이를 길게 조절하였습니다.

1 콜그래프

콜그래프란 말 그대로 그래프입니다. 다만 어떤 함수가 다른 어떤 함수를 콜 하는지, 몇번 어디에서 하는지를 나타내주는 것이 특징입니다. 콜그래프를 만들기 위해서 parsing tree의 어느 부분에서 함수가 call되는지 파악하는 것이 중요합니다.

2 구현 방법

구현에 실패하였기 때문에 시도한 내용을 우선 작성하면, 처음에는 main이 root가 되도록 m이라는 변수를 따로 만들어 name이 main이 됐을 경우 m이 1이 되어 m이 1일 때만 call과 func 배열에 name들이 저장되도록 했습니다. 하지만 그럴 경우 main내부에서 함수를 호출했을 경우 main의 위쪽에서 정의된 함수는 func 배열에 저장하지 못해 그 함수가 func이 되어 다른 것들을 call하도록 할 수 없다는 것을 알았습니다. 이때 제가 print한 값은 main −⟩ printf와 main −⟩ ms.f 뿐이었습니다.(test1.c) 그 뒤에 시도한 방법은 direct declarator에서 모든 name을 일단 func배열에 저장한 다음 main이 몇번째에 있는지 확인한 다음 그걸 이용해 해보려 많은 시도를 했지만 결국 실패해 다 지우고 다시 그 전으로 돌아왔습니다. 그 뒤로도 배열과 변수를 추가하며 여러 시도와 생각을 해봤지만 실패했고 저에게 좀 더 많은 시간이 있다면 할 수 있었을까 싶습니다. 아래는 비록 실패이지만 최종적으로 작성한 코드입니다.

3 LEX CODE

기본적인 문법은 모두 pdf를 참고했으며 주석은 모두 //을 이용해 달았습니다.
Lex 문법은 hw3과 거의 동일하게 작성했습니다.

```
%{
#include <stdio.h>
#include "y.tab.h"
extern char name[10];

extern void yyerror(const char *);
int check_type(void);
%}
D [0-9]
L [a-zA-Z_]
H [a-fA-F0-9]
E [Ee][+-]?{D}+
FS (f|F|l|L)
IS (u|U|l|L)*
%%
"//".*\n          { ; }
"/*" { comment(); }
"#include".*\n    { ; }
#define.*\n      { ; }
"auto" { return(AUTO); }
```

```
"break" { return(BREAK); }
"case" { return(CASE); }
"char" { return(CHAR); }
"const" { return(CONST); }
"continue" { return(CONTINUE); }
"default" { return(DEFAULT); }
"do" { return(DO); }
"double" {return(DOUBLE); }
"else" { return(ELSE); }
"enum" { return(ENUM); }
"extern" { return(EXTERN); }
"float" { return(FLOAT); }
"for" { return(FOR); }
"goto" { return(GOTO); }
"if" { return(IF); }
"int" { return(INT); }
"long" { return(LONG); }
"register" { return(REGISTER); }
"return" { return(RETURN); }
"short" { return(SHORT); }
"signed" { return(SIGNED); }
"sizeof" { return(SIZEOF); }
"static" { return(STATIC); }
"struct" { return(STRUCT); }
"switch" { return(SWITCH); }
"typedef" { return(TYPEDEF); }
"union" { return(UNION); }
"unsigned" { return(UNSIGNED); }
"void" { return(VOID); }
"volatile" { return(VOLATILE); }
"while" { return(WHILE); }
{L}({L}|{D})* { strcpy(name,yytext); return(IDENTIFIER); }
{L}({L}|{D})*"."{L}({L}|{D})* { strcpy(name,yytext); return(IDENTIFIER); }
// ms.f를 읽기 위해 임의로 만든 규칙
0[xX]{H}+{IS}? { return(CONSTANT); }
0{D}+{IS}? { return(CONSTANT); }
{D}+{IS}? { return(CONSTANT); }
L?'(\\.|[^\\'])+' { return(CONSTANT); }
{D}+{E}{FS}? { return(CONSTANT); }
{D}*"."{D}+({E})?{FS}? { return(CONSTANT); }
{D}+"."{D}*({E})?{FS}? { return(CONSTANT); }
L?\"(\\.|[^\\"])*\" { return(STRING_LITERAL); }
"..." { return(ELLIPSIS);}
">>=" { return(RIGHT_ASSIGN);}
"<<=" { return(LEFT_ASSIGN);}
"+=" { return(ADD_ASSIGN); }
"-=" { return(SUB_ASSIGN); }
"*=" { return(MUL_ASSIGN); }
"/=" { return(DIV_ASSIGN); }
"%=" { return(MOD_ASSIGN); }
"&=" { return(AND_ASSIGN); }
"^=" { return(XOR_ASSIGN); }
"|=" { return(OR_ASSIGN); }
">>" { return(RIGHT_OP); }
"<<" { return(LEFT_OP); }
"++" { return(INC_OP); }
"--" { return(DEC_OP); }
```

```
"->"  { return(PTR_OP); }
"&&"  { return(AND_OP); }
"||"  { return(OR_OP); }
"<="  { return(LE_OP); }
">="  { return(GE_OP); }
"=="  { return(EQ_OP);}
"!="  { return(NE_OP); }
";"   { return(';'); }
("{"|"<%") { return('{'); }
("}"|"%>") { return('}'); }
","   { return(','); }
":"   { return(':'); }
"="   { return('='); }
"("   { return('('); }
")"   { return(')'); }
("["|"<:") { return('['); }
("]"|":>") { return(']'); }
"."   { return('.'); }
"&"   { return('&'); }
"!"   { return('!'); }
"~"   { return('~'); }
"-"   { return('-'); }
"+"   { return('+'); }
"*"   { return('*'); }
"/"   { return('/'); }
"%"   { return('%'); }
"<"   { return('<'); }
">"   { return('>'); }
"^"   { return('^'); }
"|"   { return('|'); }
"?"   { return('?'); }
[\t\v\f] { ;}
\n        { ; }          //몇 번째 줄의 함수인지 세기 위해 따로 분리해 놓은 상태
.  { ; }
%%

int yywrap(void){
        return 1;
}

//필요없는 함수는 정리
comment()
{
        char c, c1;
        loop:
        while ((c = input()) != '*' && c != 0) ;
        if ((c1 = input()) != '/' && c != 0){
                unput(c1);
                goto loop;
        }
}
```

4 YACC CODE

기본적인 문법은 모두 pdf를 참고했으며 주석은 모두 //을 이용해 달았습니다.

```
%{
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char name[10];
char func[10];
char call[10][10];
char a[10][10]; //배열을 활용해보려고 추가
char b[10][10]; //배열을 활용해보려고 추가
int line;    //줄을 세기 위해 추가
int num;     //함수 사용 횟수를 세기 위해 추가
int checkfunc = 0;
int i = 0;
int k = 0;
int m = 0;
int f = 0;
int j = 0;
%}
%token INCLUDE DEFINE
%token IDENTIFIER CONSTANT STRING_LITERAL SIZEOF
%token PTR_OP INC_OP DEC_OP LEFT_OP RIGHT_OP LE_OP GE_OP EQ_OP NE_OP
%token AND_OP OR_OP MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN
%token SUB_ASSIGN LEFT_ASSIGN RIGHT_ASSIGN AND_ASSIGN
%token XOR_ASSIGN OR_ASSIGN TYPE_NAME
%token TYPEDEF EXTERN STATIC AUTO REGISTER
%token CHAR SHORT INT LONG SIGNED UNSIGNED FLOAT DOUBLE CONST VOLATILE VOID
%token STRUCT UNION ENUM ELLIPSIS
%token CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK RETURN
%start translation_unit
%%

primary_expression
        : IDENTIFIER
        | CONSTANT
        | STRING_LITERAL
        | '(' expression ')'
        ;
postfix_expression
        : primary_expression
        | postfix_expression '[' expression ']'
        | postfix_expression '(' ')'
        {
                int x;
                if(m == 1){
                        strcpy(call[i], name);
                        for(x = 0; x < k; x++){
                                if(strcmp(a[x], name) == 0){
                                        strcpy(func, name);
                                }
                        }
                        i++;
                        checkfunc=0;
                }
```

//func()와 같은 함수가 나올 때를 위해 만든 조건. m == 1이라는 건 main함수가 시작된 이후임을 나타내기 위해서인데, 이는 main함수 앞에서 정의된 함수에 접근하기 어렵게 하므로 수정되어야 하나, 여러 시도를 해도 실패하여 결국 이런 조건을 적는 것이 결과물이 그나마 가장 깔끔했기 때문에 유지했습니다. a[x]배열은 main함수 이전의 함수를 저장하기 위해 임시로 만든 배열인데 사용하지는 않습니다.

```
        }
```

```
        | postfix_expression '(' argument_expression_list ')'
        | postfix_expression PTR_OP IDENTIFIER
        | postfix_expression INC_OP
        | postfix_expression DEC_OP
        ;
argument_expression_list
        : assignment_expression
        {
                int x;
                if(m == 1){
                        strcpy(call[i], name);

                        for(x = 0; x < k; x++){
                                if(strcmp(a[x], name) == 0){
                                        strcpy(func, name);
                                }
                        }
                        i++;
                        checkfunc=0;
                        //위와 동일하나 이쪽은 함수가 호출될 때 argument expression list가
괄호의 안으로 들어가므로 func(...)라고 할 수 있습니다. 코드는 동일하지만 assignment expression이 argu-
ment exprssion list됐을 때는 이미 assingment expression을 거쳐간 뒤라고 생각해서 위쪽 규칙에만 조건을
적어줬습니다.
                }
        }
        | argument_expression_list ',' assignment_expression
        ;
unary_expression
        : postfix_expression
        | INC_OP unary_expression
        | DEC_OP unary_expression
        | unary_operator cast_expression
        | SIZEOF unary_expression
        | SIZEOF '(' type_name ')'
        ;
unary_operator
        : '&'
        | '*'
        | '+'
        | '-'
        | '~'
        | '!'
        ;
cast_expression
        : unary_expression
        | '(' type_name ')' cast_expression
        ;
multiplicative_expression
        : cast_expression
        | multiplicative_expression '*' cast_expression
        | multiplicative_expression '/' cast_expression
        | multiplicative_expression '%' cast_expression
        ;
additive_expression
        : multiplicative_expression
        | additive_expression '+' multiplicative_expression
        | additive_expression '-' multiplicative_expression
```

```
        ;
shift_expression
        : additive_expression
        | shift_expression LEFT_OP additive_expression
        | shift_expression RIGHT_OP additive_expression
        ;
relational_expression
        : shift_expression
        | relational_expression '<' shift_expression
        | relational_expression '>' shift_expression
        | relational_expression LE_OP shift_expression
        | relational_expression GE_OP shift_expression
        ;
equality_expression
        : relational_expression
        | equality_expression EQ_OP relational_expression
        | equality_expression NE_OP relational_expression
        ;
and_expression
        : equality_expression
        | and_expression '&' equality_expression
        ;
exclusive_or_expression
        : and_expression
        | exclusive_or_expression '^' and_expression
        ;
inclusive_or_expression
        : exclusive_or_expression
        | inclusive_or_expression '|' exclusive_or_expression
        ;
logical_and_expression
        : inclusive_or_expression
        | logical_and_expression AND_OP inclusive_or_expression
        ;
logical_or_expression
        : logical_and_expression
        | logical_or_expression OR_OP logical_and_expression
        ;
conditional_expression
        : logical_or_expression
        | logical_or_expression '?' expression ':' conditional_expression
        ;
assignment_expression
        : conditional_expression
        | unary_expression assignment_operator assignment_expression
        ;
assignment_operator
        : '='
        | MUL_ASSIGN
        | DIV_ASSIGN
        | MOD_ASSIGN
        | ADD_ASSIGN
        | SUB_ASSIGN
        | LEFT_ASSIGN
        | RIGHT_ASSIGN
        | AND_ASSIGN
        | XOR_ASSIGN
```

```
        | OR_ASSIGN
        ;
expression
        : assignment_expression
        | expression ',' assignment_expression
        ;
constant_expression
        : conditional_expression
        ;
declaration
        : declaration_specifiers ';'
        | declaration_specifiers init_declarator_list ';'
        ;
declaration_specifiers
        : storage_class_specifier
        | storage_class_specifier declaration_specifiers
        | type_specifier
        | type_specifier declaration_specifiers
        | type_qualifier
        | type_qualifier declaration_specifiers
        ;
init_declarator_list
        : init_declarator
        | init_declarator_list ',' init_declarator
        ;
init_declarator
        : declarator
        | declarator '=' initializer
        ;
storage_class_specifier
        : TYPEDEF
        | EXTERN
        | STATIC
        | AUTO
        | REGISTER
        ;
type_specifier
        : VOID
        | CHAR
        | SHORT
        | INT
        | LONG
        | FLOAT
        | DOUBLE
        | SIGNED
        | UNSIGNED
        | struct_or_union_specifier
        | enum_specifier
        | TYPE_NAME
        ;
struct_or_union_specifier
        : struct_or_union IDENTIFIER '{' struct_declaration_list '}'
        | struct_or_union '{' struct_declaration_list '}'
        | struct_or_union IDENTIFIER
        ;
struct_or_union
        : STRUCT
```

```
        | UNION
        ;
struct_declaration_list
        : struct_declaration
        | struct_declaration_list struct_declaration
        ;
struct_declaration
        : specifier_qualifier_list struct_declarator_list ';'
        ;
specifier_qualifier_list
        : type_specifier specifier_qualifier_list
        | type_specifier
        | type_qualifier specifier_qualifier_list
        | type_qualifier
        ;
struct_declarator_list
        : struct_declarator
        | struct_declarator_list ',' struct_declarator
        ;
struct_declarator
        : declarator
        | ':' constant_expression
        | declarator ':' constant_expression
        ;
enum_specifier
        : ENUM '{' enumerator_list '}'
        | ENUM IDENTIFIER '{' enumerator_list '}'
        | ENUM IDENTIFIER
        ;
enumerator_list
        : enumerator
        | enumerator_list ',' enumerator
        ;
enumerator
        : IDENTIFIER
        | IDENTIFIER '=' constant_expression
        ;
type_qualifier
        : CONST
        | VOLATILE
        ;
declarator
        : pointer direct_declarator
        | direct_declarator
        ;
direct_declarator
        : IDENTIFIER
        {
                strcpy(a[k],name);
                k++;
                if(strcmp(name,"main") == 0)
                        m = 1;
        }
        | '(' declarator ')'
        | direct_declarator '[' constant_expression ']'
        | direct_declarator '[' ']'
        | direct_declarator '(' parameter_type_list ')'
```

```
        {
                if(m == 1){
                        strcpy(func, name);
                        k++;
                }
                checkfunc = 1;
        }
        | direct_declarator '(' identifier_list ')'
        {
                if(m == 1){
                        strcpy(func, name);
                        k++;
                }
                checkfunc = 1;
        }
        | direct_declarator '(' ')'
        {
                if(m == 1){
                        strcpy(func, name);
                        k++;
                }
                checkfunc = 1;
        }    //함수가 정의될 때 func에 함수의 이름이 들어갈 수 있도록 규칙을 만들어줬습니다.
        ;
pointer
        : '*'
        | '*' type_qualifier_list
        | '*' pointer
        | '*' type_qualifier_list pointer
        ;
type_qualifier_list
        : type_qualifier
        | type_qualifier_list type_qualifier
        ;
parameter_type_list
        : parameter_list
        | parameter_list ',' ELLIPSIS
        ;
parameter_list
        : parameter_declaration
        | parameter_list ',' parameter_declaration
        ;
parameter_declaration
        : declaration_specifiers declarator
        | declaration_specifiers abstract_declarator
        | declaration_specifiers
        ;
identifier_list
        : IDENTIFIER
        | identifier_list ',' IDENTIFIER
        ;
type_name
        : specifier_qualifier_list
        | specifier_qualifier_list abstract_declarator
        ;
abstract_declarator
        : pointer
```

```
        | direct_abstract_declarator
        | pointer direct_abstract_declarator
        ;
direct_abstract_declarator
        : '(' abstract_declarator ')'
        | '[' ']'
        | '[' constant_expression ']'
        | direct_abstract_declarator '[' ']'
        | direct_abstract_declarator '[' constant_expression ']'
        | '(' ')'
        | '(' parameter_type_list ')'
        | direct_abstract_declarator '(' ')'
        | direct_abstract_declarator '(' parameter_type_list ')'
        ;
initializer
        : assignment_expression
        | '{' initializer_list '}'
        | '{' initializer_list ',' '}'
        ;
initializer_list
        : initializer
        | initializer_list ',' initializer
        ;
statement
        : labeled_statement
        | compound_statement
        | expression_statement
        | selection_statement
        | iteration_statement
        | jump_statement
        ;
labeled_statement
        : IDENTIFIER ':' statement
        | CASE constant_expression ':' statement
        | DEFAULT ':' statement
        ;
compound_statement
        : '{' '}'
        | '{' statement_list '}'
        | '{' declaration_list '}'
        | '{' declaration_list statement_list '}'
        ;
declaration_list
        : declaration
        | declaration_list declaration
        ;
statement_list
        : statement
        | statement_list statement
        | statement_list declaration_list statement
        ;
expression_statement
        : ';'
        | expression ';'
        ;
selection_statement
        : IF '(' expression ')' statement
```

```
        | SWITCH '(' expression ')' statement
        ;
iteration_statement
        : WHILE '(' expression ')' statement
        | DO statement WHILE '(' expression ')' ';'
        | FOR '(' expression_statement expression_statement ')' statement
        | FOR '(' expression_statement expression_statement expression ')' statement
        ;
jump_statement
        : GOTO IDENTIFIER ';'
        | CONTINUE ';'
        | BREAK ';'
        | RETURN ';'
        | RETURN expression ';'
        ;
translation_unit
        : external_declaration
        | translation_unit external_declaration
        ;
external_declaration
        : function_definition
        | declaration
        ;
function_definition
        : declaration_specifiers declarator declaration_list compound_statement
        | declaration_specifiers declarator compound_statement
        | declarator declaration_list compound_statement
        | declarator compound_statement
        ;
%%

int main(){

        int j=0;
        yyparse();
    FILE * graph = fopen("test1Graph.gv","w");

        for(j;j<i;j++)
        {
                printf("\"%s\" -> \"%s\"\n", func, call[j]);
        }    //이곳도 수정이 매우 필요한 부분인 것을 알지만 못하였습니다..

    fclose(graph);   //fprintf를 하려고 약간 시도한 흔적
    system("dot -Tjpg test1Graph.gv -o test1Graph.jpg");

        return 0;

}

void yyerror(const char *str)
{
        fprintf(stderr, "error: %s\n", str);
}
```

5 함수 호출 관계에 대한 내용

배열을 이용해 구현하려고 했으나 실패했습니다...