

## Assignment 12: AST

Date: 2024. 12. 02

Student ID: 201924451

Name: 김태훈

### 1. AST 설계 및 생성자 함수 설명[1]

```
typedef enum{
    INT_NODE,
    SYM_NODE,
    BINOP_NODE,
    UNOP_NODE
} NodeType;
typedef struct Node{
    NodeType type;
    union{
        int ival;
        char *sval;
        struct{
            char *op;
            struct Node *left;
            struct Node *right;

        } opNode;
    } data;
} Node;
```

Node 구조체는 Node의 타입과 union으로 구성된 ival, sval, struct opNode가 있다. Node에는 정수, 심볼, 연산자 중 하나만 들어가므로 해당 부분은 union으로 구성하였다. struct opNode는 연산자와 하위 노드(피연산자) 2개로 구성되어있다.

```

Node *mkIntNode(int n) {
    Node *node =(Node *)malloc(sizeof(Node));
    node->type =INT_NODE;
    node->data.ival =n;
    return node;
}

Node *mkSymNode(char *s) {
    Node *node =(Node *)malloc(sizeof(Node));
    node->type =SYM_NODE;
    node->data.sval =strdup(s);
    return node;
}

Node *mkBopNode(char *op, Node *left, Node *right) {
    Node *node =(Node *)malloc(sizeof(Node));
    node->type =BINOP_NODE;
    node->data.opNode.op =strdup(op);
    node->data.opNode.left =left;
    node->data.opNode.right =right;
    return node;
}

Node *mkUopNode(char *op, Node *son) {
    Node *node =(Node *)malloc(sizeof(Node));
    node->type =UNOP_NODE;
    node->data.opNode.op =strdup(op);
    node->data.opNode.left =son;
    return node;
}

```

(1) mkIntNode

새 노드를 생성하고, INT\_NODE로 타입을 설정한 다음 정수값을 저장하는 ival에 정수를 저장한다.

(2) mkSymNode

새 노드를 생성하고, SYM\_NODE로 타입을 설정한 다음, 심볼을 저장하는 sval에 심볼을 저장한다.

(3) mkBopNode

새 노드를 생성하고, BINOP\_NODE로 타입을 설정한 다음, struct opNode에 연산자와 하위 노드 2개를 저장한다.

(4) mkUopNode

새 노드를 생성하고, UNOP\_NODE로 타입을 설정한 다음, struct opNode에 연산자와 하위 노드 1개를 저장

한다.

## 2. Bison 입력 코드 및 설명

```
%{ #include <stdio.h>
#include <stdio.h>
#include "ast.h"
int yyerror(const char *msg), yylex();
Node *Root;
%}
%union {
char *sval;
int ival;
Node *pval;
}
%token <ival> NUM
%token <sval> ID '+' '-' '*' '/'
%type <pval> Exp Term Factor
%%
Prg :Exp { Root = $1; }
    ;
Exp :Exp '+' Term { $$=mkBopNode("+", $1, $3); }
    |Exp '-' Term { $$=mkBopNode("-", $1, $3); }
    |Term          { $$=$1; }
    ;
Term :Term '*' Factor { $$=mkBopNode("*", $1, $3); }
    |Term '/' Factor { $$=mkBopNode("/", $1, $3); }
    |Factor          { $$=$1; }
    ;
Factor : '(' Exp ')' { $$=$2; }
    | '-' Factor { $$=mkUopNode("-", $2); }
    | '+' Factor { $$=mkUopNode("+", $2); }
    | NUM      { $$=mkIntNode($1); }
    | ID       { $$=mkSymNode($1); }
    ;
%%

int main() { yyparse(); printTree(Root, 0); }
int yyerror(const char *msg) { fputs(msg, stderr); return -1; }
```

곱셈과 나눗셈이 덧셈과 뺄셈보다 먼저 수행하도록 하였고, 괄호 안의 식이 가장 먼저 수행되도록 하였다. 이항 연산자일 경우 mkBopNode를 사용하여 노드를 구성하고, 단일 연산자의 경우 mkUopNode를 사용하여 노드를 구성하였다. 숫자일 경우 mkIntNode를, ID일 경우 mkSymNode를 사용하여 노드를 구성한다.

## 3. Flex 입력 코드 및 설명

```
%{
    #include <stdlib.h>
    #include "ast.h"
    #include "y.tab.h"
}%
%%
[0-9]+    { yylval.ival =atoi(yytext); returnNUM; }
[a-zA-Z]+ { yylval.sval =strdup(yytext); returnID; }
[ \t]    ;
\+ return ('+');
\- return ('-');
\* return ('*');
\/ return ('/');
\( return ('(');
\) return (')');
\n return (0);
. {printf("'%'c': illegal character\n"),yytext[0]; exit(-1);}

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

0에서 9로 이루어진 문자열은 숫자로 바꾼 후 `yylval.ival`에 저장하고 숫자 토큰을 반환한다.

문자로 이루어진 문자열은 `yylval.sval`에 저장하고 ID 토큰을 반환한다. 이 때 `-1`, `+1` 등 기호가 붙어있는 숫자는 bison에서 단일 연산자로 처리하도록 하였다. Flex에서 처리하면 5-7은 옳은 문자열임에도 불구하고, 5와 -7로 인식하여 연산자가 없어 오류를 발생시킨다.

덧셈, 뺄셈, 나눗셈, 곱셈, 괄호 등의 토큰을 만나면 해당 토큰 문자를 반환한다.

#### 4. LISP 소스 코드일 때 AST 설계 방법

```

%{ #include <stdio.h>
#include <stdio.h>
#include "ast.h"
int yyerror(const char *msg), yylex();
Node *Root;
%}
%union {
char *sval;
int ival;
Node *pval;
}
%token <ival> NUM
%token <sval> ID '+' '-' '*' '/'
%type <pval> Exp
%%
Prg :Exp { Root = $1; }
;
Exp :NUM { $$=mkIntNode($1); }
|ID { $$=mkSymNode($1); }
| '-' Exp { $$=mkUopNode("-", $2); }
| '+' Exp { $$=mkUopNode("+", $2); }
| '(' '+' Exp Exp ')' { $$=mkBopNode("+", $3, $4); }
| '(' '-' Exp Exp ')' { $$=mkBopNode("-", $3, $4); }
| '(' '*' Exp Exp ')' { $$=mkBopNode("*", $3, $4); }
| '(' '/' Exp Exp ')' { $$=mkBopNode("/", $3, $4); }
;
%%
int main() { yyparse(); printTree(Root, 0); }
int yyerror(const char *msg) { fputs(msg, stderr); return -1; }

```

Bison 코드만 바꾸면 된다. LISP수식은 모든 수식에 괄호가 있으므로, 각 연산자의 우선순위는 고려하지 않고, (OP Exp Exp) 의 형태인 경우 mkBopNode를 통해 2개의 child node가 있는 노드(다항 연산자)를 만들고, -Exp, +Exp인 경우 mkUopNode를 통해 1개의 child node가 있는 노드를 만든다(단항 연산자)

```

$ ./main
(+ (- -2 (* 3 2)) 5)
Op(+)
  op(-)
    Uop(-)
      Int(2)
    op(*)
      Int(3)
      Int(2)
  Int(5)

```

## References

- [1] Compiler Lab 11: Making the AST Using Bison. available at: <https://youtu.be/FaGeBd3kDXk>