REPORT

(컴파일러 3차 과제)



|  |  |
| --- | --- |
| 제목 | AST |
| 제출일자 | 21.11.30 |
| 소속학과 | 컴퓨터공학과 |
| 학번 | 2017305039 |
| 성명 | 신동민 |

**문제) Mini C 프로그램에 대한 AST을 생성하는 문법-지시적 변환기를 구현하시오.**

**프로그램 동작:** 1. 파일 입력 -> 2. 스캐너 -> 3. 파서 -> 4. SDT -> 트리 출력

Perfect.mc

**파일을 입력**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Scanner()**

**Parser()**

**valueStack**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |

**shift: 명칭, 상수이면 노드 생성**

**SDT**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**reducec: 서브 트리 생성**

**트리 출력**

**종료**

**/\* scanner \*/**

**+**

**/\* parser 변경 \*/**

NODE\* parser(FILE\* source\_file) {

extern int parsingTable[NO\_STATES][NO\_SYMBOLS + 1];

extern int leftSymbol[NO\_RULES + 1], rightLength[NO\_RULES + 1];

int entry, rulenumber, lhs;

int currentState;

struct tokenType token;

NODE\* ptr;

sp = 0; stateStack[sp] = 0;

token = scanner(source\_file);

while (true)

{

currentState = stateStack[sp];

entry = parsingTable[currentState][token.number];

if (entry > 0) {

sp++;

if (sp > PS\_SIZE) {

printf("critical compiler error: parsing stack overflew");

exit(1);

}

symbolStack[sp] = token.number;

stateStack[sp] = entry;

valueStack[sp] = meaningfulToken(token) ? buildNode(token) : NULL;

token = scanner(source\_file);

}

else if (entry < 0) {

rulenumber = -entry;

if (rulenumber == GOAL\_RULE) {

if (errcnt == 0) return valueStack[sp - 1];

else printf(" \*\*\* error in source: %d\n", errcnt);

}

ptr = buildTree(ruleName[rulenumber], rightLength[rulenumber]);

sp = sp - rightLength[rulenumber];

lhs = leftSymbol[rulenumber];

currentState = parsingTable[stateStack[sp]][lhs];

sp++;

symbolStack[sp] = lhs;

stateStack[sp] = currentState;

valueStack[sp] = ptr;

}

else {

printf(" === error in source ===\n");

errcnt++;

printf("Current Token : ");

printToken(token);

dumpStack();

errorRecovery(source\_file);

token = scanner(source\_file);

}

}

}

**+**

**<AST>**

// 노드에 대한 정보를 가진 구조체

enum noderep { terminal, nonterm };

typedef struct nodetype {

struct tokenType token;

enum noderep noderep;

struct nodetype\* son;

struct nodetype\* brother;

}NODE;

// 노드 스택

NODE\* valueStack[PS\_SIZE];

// 노드들의 이름들

const char\* nodeName[] = {

"ACTUAL\_PARAM", "ADD", "ADD\_ASSIGN", "ARRAY\_VAR", "ASSIGN\_OP",

"CALL", "COMPOUND\_ST", "CONST\_NODE", "DCL", "DCL\_ITEM",

"DCL\_LIST", "DCL\_SPEC", "DIV", "DIV\_ASSIGN", "EQ",

"ERROR\_NODE", "EXP\_ST", "FORMAL\_PARA", "FUNC\_DEF", "FUNC\_HEAD",

"GE", "GT", "IDENT", "IF\_ELSE\_ST", "IF\_ST",

"INDEX", "INT\_NODE", "LE", "LOGICAL\_AND", "LOGICAL\_NOT",

"LOGICAL\_OR", "LT", "MOD", "MOD\_ASSIGN", "MUL",

"MUL\_ASSIGN", "NE", "NUMBER", "PARAM\_DCL", "POST\_DEC",

"POST\_INC", "PRE\_DEC", "PRE\_INC", "PROGRAM", "RETURN\_ST",

"SIMPLE\_VAR", "STAT\_LIST", "SUB", "SUB\_ASSIGN", "UNARY\_MINUS",

"VOID\_NODE", "WHILE\_ST"

};

// 노드들의 이름마다 열거 값을 지정

enum nodeNumber

{

ACTUAL\_PARAM, ADD, ADD\_ASSIGN, ARRAY\_VAR, ASSIGN\_OP,

CALL, COMPOUND\_ST, CONST\_NODE, DCL, DCL\_ITEM,

DCL\_LIST, DCL\_SPEC, DIV, DIV\_ASSIGN, EQ,

ERROR\_NODE, EXP\_ST, FORMAL\_PARA, FUNC\_DEF, FUNC\_HEAD,

GE, GT, IDENT, IF\_ELSE\_ST, IF\_ST,

INDEX, INT\_NODE, LE, LOGICAL\_AND, LOGICAL\_NOT,

LOGICAL\_OR, LT, MOD, MOD\_ASSIGN, MUL,

MUL\_ASSIGN, NE, NUMBER, PARAM\_DCL, POST\_DEC,

POST\_INC, PRE\_DEC, PRE\_INC, PROGRAM, RETURN\_ST,

SIMPLE\_VAR, STAT\_LIST, SUB, SUB\_ASSIGN, UNARY\_MINUS,

VOID\_NODE, WHILE\_ST

};

// 트리를 만들 때 넣는 노드 넘버들

int ruleName[] = {

/\* 0 1 2 3 4 \*/

0, PROGRAM, 0, 0, 0,

/\* 5 6 7 8 9 \*/

0, FUNC\_DEF, FUNC\_HEAD, DCL\_SPEC, 0,

/\* 10 11 12 13 14 \*/

0, 0, 0, CONST\_NODE, INT\_NODE,

/\* 15 16 17 18 19 \*/

VOID\_NODE, 0, FORMAL\_PARA, 0, 0,

/\* 20 21 22 23 24 \*/

0, 0, PARAM\_DCL, COMPOUND\_ST, DCL\_LIST,

/\* 25 26 27 28 29 \*/

DCL\_LIST, 0, 0, DCL, 0,

/\* 30 31 32 33 34 \*/

0, DCL\_ITEM, DCL\_ITEM, SIMPLE\_VAR, ARRAY\_VAR,

/\* 35 36 37 38 39 \*/

0, 0, STAT\_LIST, 0, 0,

/\* 40 41 42 43 44 \*/

0, 0, 0, 0, 0,

/\* 45 46 47 48 49 \*/

0, EXP\_ST, 0, 0, IF\_ST,

/\* 50 51 52 53 54 \*/

IF\_ELSE\_ST, WHILE\_ST, RETURN\_ST, 0, 0,

/\* 55 56 57 58 59 \*/

ASSIGN\_OP, ADD\_ASSIGN, SUB\_ASSIGN, MUL\_ASSIGN, DIV\_ASSIGN,

/\* 60 61 62 63 64 \*/

MOD\_ASSIGN, 0, LOGICAL\_OR, 0, LOGICAL\_AND,

/\* 65 66 67 68 69 \*/

0, EQ, NE, 0, GT,

/\* 70 71 72 73 74 \*/

LT, GE, LE, 0, ADD,

/\* 75 76 77 78 79 \*/

SUB, 0, MUL, DIV, MOD,

/\* 80 81 82 83 84 \*/

0, UNARY\_MINUS, LOGICAL\_NOT, PRE\_INC, PRE\_DEC,

/\* 85 86 87 88 89 \*/

0, INDEX, CALL, POST\_INC, POST\_DEC,

/\* 90 91 92 93 94 \*/

0, 0, ACTUAL\_PARAM, 0, 0,

/\* 95 96 97 \*/

0, 0, 0

};

// shift action일때 명칭이나 상수이면 노드를 생성하는 함수

NODE\* buildNode(struct tokenType token) {

NODE\* ptr;

ptr = (NODE\*)malloc(sizeof(NODE));

if (!ptr) {

printf("bulid node malloc error\n");

exit(1);

}

ptr->token = token;

ptr->noderep = terminal;

ptr->son = NULL;

ptr->brother = NULL;

return ptr;

}

// reduce action일때 서브 트리를 생성하는 함수

NODE\* buildTree(int nodeNumber, int rhsLength) {

int i, j, start;

NODE\* first, \* ptr;

i = sp - rhsLength + 1;

// step 1: find a first index with node in value stack

while (i <= sp && valueStack[i] == NULL) i++;

if (!nodeNumber && i > sp) return NULL;

start = i;

// step 2: linking brothers

while (i <= sp - 1) {

j = i + 1;

while (j <= sp && valueStack[j] == NULL) j++;

if (j <= sp) {

ptr = valueStack[i];

while (ptr->brother) ptr = ptr->brother;

ptr->brother = valueStack[j];

}

i = j;

}

first = (start > sp) ? NULL : valueStack[start];

// step 3: making subtree root and linking son

if (nodeNumber) {

ptr = (NODE\*)malloc(sizeof(NODE));

if (!ptr) {

printf("malloc error in buildTree()\n");

exit(1);

}

ptr->token.number = nodeNumber;

ptr->noderep = nonterm;

ptr->son = first;

ptr->brother = NULL;

return ptr;

}

else return first;

}

// 노드를 출력하는 함수

void printNode(NODE\* pt, int indent) {

int i;

for (i = 1; i <= indent; i++) printf(" ");

if (pt->noderep == terminal) { // terminal node

if (pt->token.number == tident)

printf(" Terminal: %s", pt->token.value.id);

else if (pt->token.number == tnumber)

printf(" Terminal: %d", pt->token.value.num);

}

else { // nonterminal node

int i;

i = (int)(pt->token.number);

printf(" Nonterminal: %s", nodeName[i]);

}

printf("\n");

}

// 트리형태로 출력하는 함수

void printTree(NODE\* pt, int indent) {

NODE\* p = pt;

while (p != NULL)

{

printNode(p, indent);

if (p->noderep == nonterm) printTree(p->son, indent + 5);

p = p->brother;

}

}

// 명칭이나 상수인지 확인하는 함수

int meaningfulToken(struct tokenType token) {

if ((token.number == tident) || (token.number == tnumber))

return 1;

else return 0;

}

void main(int argc, char\* argv[])

{

FILE\* source\_file;

NODE\* root;

// 파일을 읽는데 없다면 종료

if ((source\_file = fopen("perfect.mc", "r")) == NULL) {

fprintf(stderr, "file not found \n");

exit(0);

}

// 해당 파일을 파서가 파싱후 트리형태인 노드 반환

root = parser(source\_file);

// 트리형태를 출력

printTree(root, 1);

fclose(source\_file);

}

**결과 출력)**

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

설명: 앞에서 했던 파서가 각 생성 규칙에 따라 수행해야 할 4가지 action들로 구문 분석을 하는 것에서 추가적으로 트리를 만들어 내는 함수를 추가해줌으로써 파서에서 shift할 때마다 의미 있는 terminal 심벌인 경우인 명칭과 상수에만 단말 노드를 만들고 reduce할 때마다 의미 있는 생성 규칙의 여부에 따라서 지금까지 만든 노드들을 묶어 서브 트리를 만들어 나갑니다. 그리고 입력 데이터에 문제가 없으면 accept가 되면서 최종적인 추상 구문 트리를 반환하여 main문에서 받아 트리를 출력하는 함수로 결과와 같이 출력합니다.