REPORT

(컴파일러 2차 과제)



|  |  |
| --- | --- |
| 제목 | Ucode |
| 제출일자 | 21.11.25 |
| 소속학과 | 컴퓨터공학과 |
| 학번 | 2017305039 |
| 성명 | 신동민 |

**문제) 부록 A에 있는 MiniC 프로그램 중 2개를 Ucode로 작성하고 Ucodei로**

**실행하여 제출하시오.**

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

Perfect.mc

**파일을 입력**

**stack**

shift

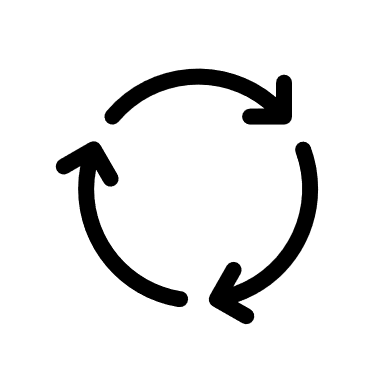
reduce

accept

**Scanner()**

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

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

 **토큰 추출**

**parsingTable**

**출력**

**종료**

**/\* scanner \*/**

**+**

**<parser>**

#include "miniC.tbl" // 파일을 참조

#define PS\_SIZE 200 // 스택의 사이즈

// 에러 카운트, 스택에 대한 포인터, 상태 스택, 심벌 스택

int errcnt = 0, sp, stateStack[PS\_SIZE], symbolStack[PS\_SIZE];

// 토큰의 이름들

const char\* tokenName[] = {

/\* 0 1 2 3 4 5 \*/

"!", "!=", "%", "%=", "%ident", "%number",

/\* 6 7 8 9 10 11 \*/

"&&", "(", ")", "\*", "\*=", "+",

/\* 12 13 14 15 16 17 \*/

"++", "+=", ",", "-", "--", "-=",

/\* 18 19 20 21 22 23 \*/

"/", "/=", ";", "<", "<=", "=",

/\* 24 25 26 27 28 29 \*/

"==", ">", ">=", "[", "]", "eof",

// .................... word symbol ...................... //

/\* 30 31 32 33 34 35 36 \*/

"const", "else", "if", "int", "return", "void", "while",

/\* 37 38 39 \*/

"{", "||", "}"

};

// reduce했을 때 rule넘버를 출력하는 함수

void semantic(int n) {

printf("reduce rule number = %d\n", n);

}

// token을 받아 알맞은 형태로 출력하는 함수

void printToken(struct tokenType token) {

if(token.number == tident)

printf("%s\n", token.value.id);

else if (token.number == tnumber)

printf("%d\n", token.value.num);

else

printf("%s\n", tokenName[token.number]);

}

// 스택을 채웠는데 에러가 있을 시 비워주는 함수

void dumpStack() {

int i, start;

if (sp > 10) start = sp - 10;

else start = 0;

printf("\n \*\*\* dump state stack :");

for(i=start; i <= sp; ++i)

printf(" %d", stateStack[i]);

printf("\n \*\*\* dump symbol stack :");

for (i = start; i <= sp; ++i)

printf(" %d", symbolStack[i]);

printf("\n");

}

// 에러에 대한 토큰과 상태 스택을 조정하는 함수

void errorRecovery(FILE\* source\_file) {

struct tokenType tok;

int parenthesisCount, braceCount;

int i;

// step1: skip to the semicolon

parenthesisCount = braceCount = 0;

while (true)

{

tok = scanner(source\_file);

if (tok.number == teof) exit(1);

if (tok.number == tlparen) parenthesisCount++;

else if (tok.number == trparen) parenthesisCount--;

if (tok.number == tlbrace) braceCount++;

else if (tok.number == trbrace) braceCount--;

if ((tok.number == tsemicolon) && (parenthesisCount <= 0) && (braceCount <= 0))

break;

}

// step2: adjust state stack

for (i = sp; i >= 0; i--) {

// statement\_list -> statement\_list . statement

if (stateStack[i] == 36) break; //second statement part

// statement\_list -> . statement

// statement\_list -> . statement\_list statement

if (stateStack[i] == 24) break; //first statement part

// declaration\_list -> declaration\_list . declaration

if (stateStack[i] == 25) break; //second internal dcl

// declaration\_list -> . declaration

// declaration\_list -> . declaration\_list declaration

if (stateStack[i] == 17) break; //internal declaration

// external declaration

// external\_dcl -> . declaration

if (stateStack[i] == 2) break; //after first external dcl

if (stateStack[i] == 0) break; //first external declaration

}

sp = i;

}

// 파일을 받아 현재 상태에 대한 토큰마다 테이블에서 값을 찾아 action을 정해서 파싱하는 함수

void parser(FILE\* source\_file) {

// miniC.tbl파일에 있는 외부 변수

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;

sp = 0; stateStack[sp] = 0; // 초기 상태

token = scanner(source\_file);

while (true)

{

currentState = stateStack[sp]; // 현재 상태

// 상태에 대한 어떤 토큰이 왔는지를 테이블에서 다음 규칙 값으로 찾아 저장

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

if (entry > 0) { // 1.shift action

sp++;

if (sp > PS\_SIZE) { // 스택의 범위를 벗어나면 에러

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

exit(1);

}

symbolStack[sp] = token.number;

stateStack[sp] = entry;

token = scanner(source\_file);

}

else if (entry < 0) { // 2.reduce action

rulenumber = -entry;

if (rulenumber == GOAL\_RULE) { // 3.accept action

// GOAL\_RULE까지 에러가 없으면 파싱 종료

if (errcnt == 0) printf(" \*\*\* valid source \*\*\*\n");

// 카운트가 되었으면 에러

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

return;

}

semantic(rulenumber); // reduce를 알림

sp = sp - rightLength[rulenumber]; // reduce되었으니 스택에서 그만큼 길이를 제거

lhs = leftSymbol[rulenumber]; // non-terminer인 왼쪽 심벌로 대체

currentState = parsingTable[stateStack[sp]][lhs]; // 대체된 것에 대한 새로운 상태

sp++;

symbolStack[sp] = lhs;

stateStack[sp] = currentState;

}

else { // 4.error action

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

errcnt++;

printf("Current Token : ");

printToken(token); // 에러가 난 현재 토큰 출력

dumpStack(); // 에러에 대한 스택 비움

errorRecovery(source\_file); // 에러에 대한 회복

token = scanner(source\_file);

}

} // while (true)

} // parser

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

{

FILE\* source\_file;

// 파일을 읽는데 없다면 종료

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

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

exit(0);

}

printf("start of parser\n");

// 해당 파일을 파서가 파싱

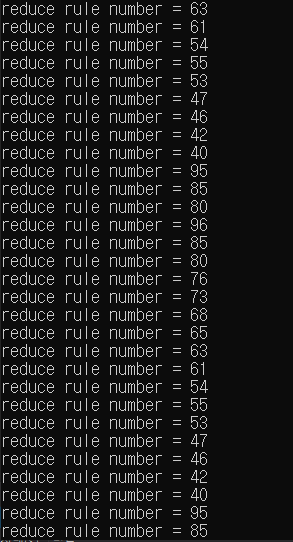
parser(source\_file);

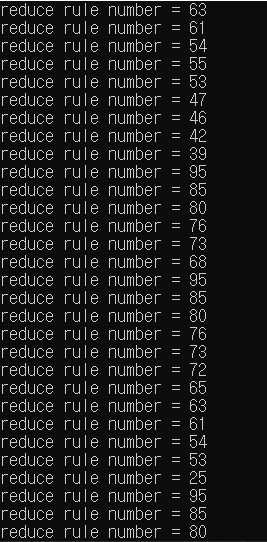
printf("end of parser\n");

fclose(source\_file);

}

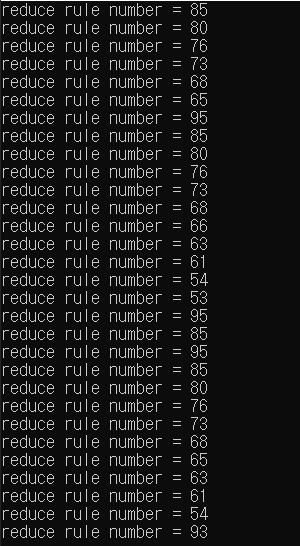
텍스트이(가) 표시된 사진

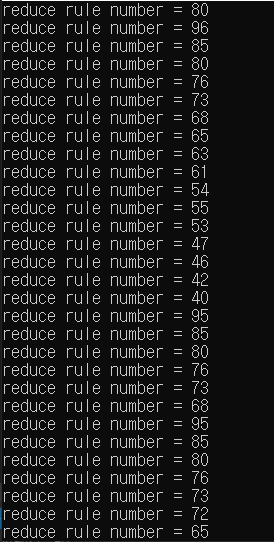
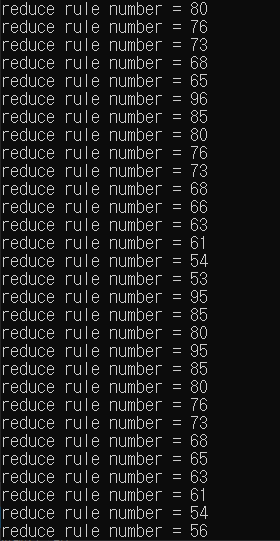
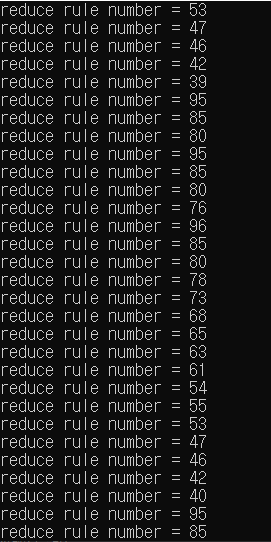
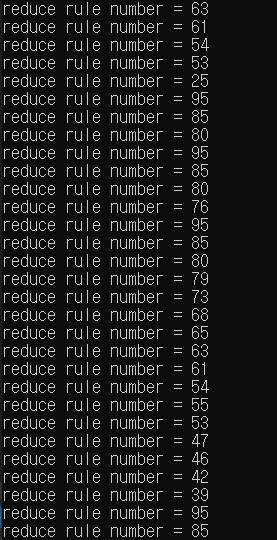
자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

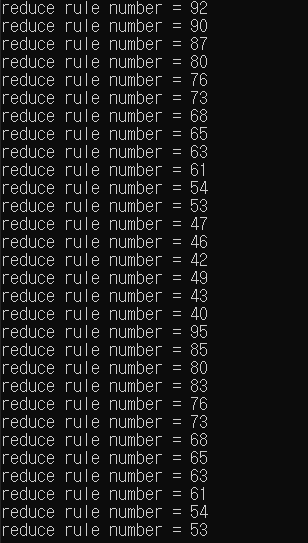
자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**결과 출력)**

텍스트, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

설명: start of parser라는 문구를 시작으로 파일안에 있는 내용을 parser()함수가 파싱을 하게 됩니다. 먼저 스캐너를 통해서 토큰을 받아와서 테이블과 비교하여 다음 규칙 값을 찾아서 알맞은 action을 실행하는데 1. Shift는 스택에 상태와 심벌을 푸쉬하며 2. Reduce는 출력을 통해 알리고 왼쪽 심벌로 대체하여 오른쪽의 길이만큼 스택들을 조정하며 3. accept는 에러 없이 GOAL\_RULE까지 왔다면 성공적으로 종료하며 마지막으로는 4. Error는 값이 0이 나온다면 에러임으로 에러 처리를 합니다. 이런 방식들이 있으며 LR 구문 분석기는 buttom-up으로 스트링에서 Shift와 reduce를 하여 시작 심벌까지 반복합니다.