HW1

201421155 이안규

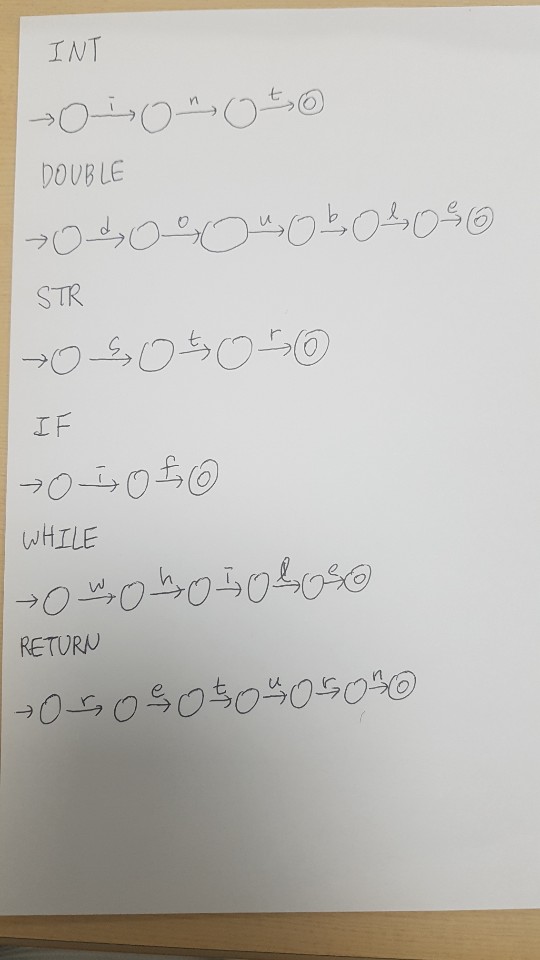
1. 이번 programming 과제에서는 주어진 Mini-C 언어의 정의된 어휘를 가지고 Lexical Analyzer를 작성해보는 과제이다. 작성한 Lexical Analyzer로 동작시킨 token들의 경우 모두 잘 작동하는 것 같지만 int \_\_\_\_\_라고 정의한 잘못된 id 정의에서 error를 전체적으로 출력하는 것이 아니라 각각의 \_ 하나씩 error를 출력하는 것까지 구현하였다. 그리고 0012의 경우 00이 에러로 출력되는 것이 아니라 12라는 DOUBLE\_NUM으로 인식된다.

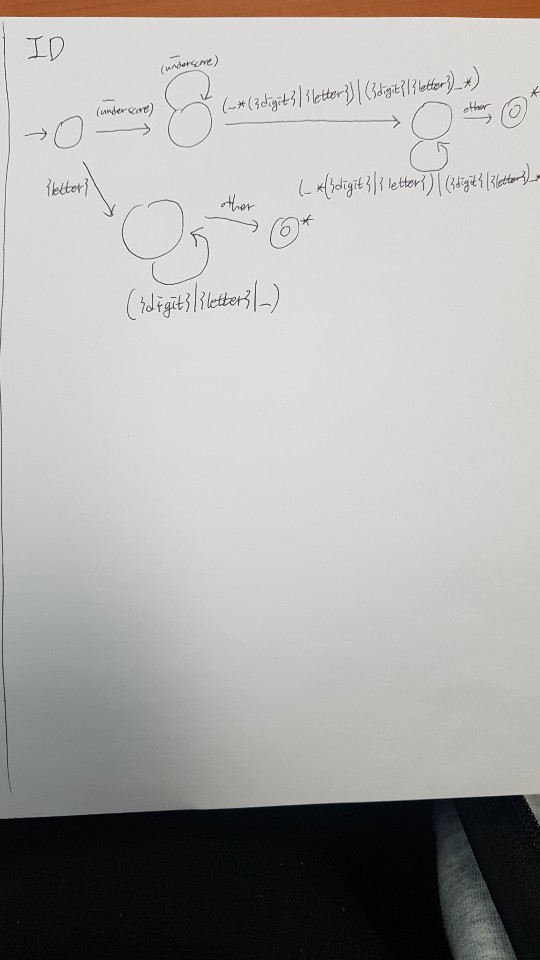
* Token의 종류는

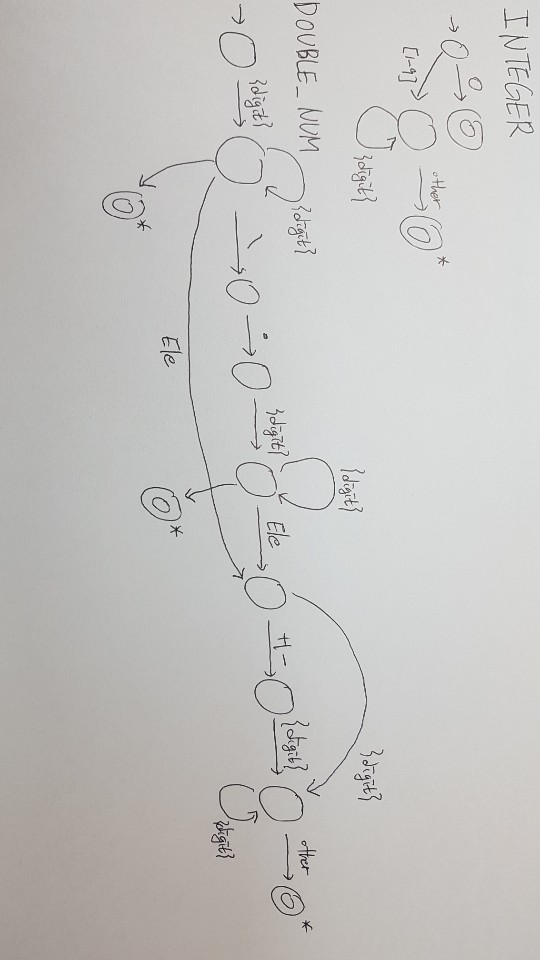
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INT | DOUBLE | STR | IF | WHILE |
| RETURN | ID | INTEGER | DOUBLE\_NUM | STRING |
| OPERATORS | QUOTE | COMMA | LPAREN | RPAREN |
| SEMI | LBRACE | RBRACE | COMMENT | ERROR |

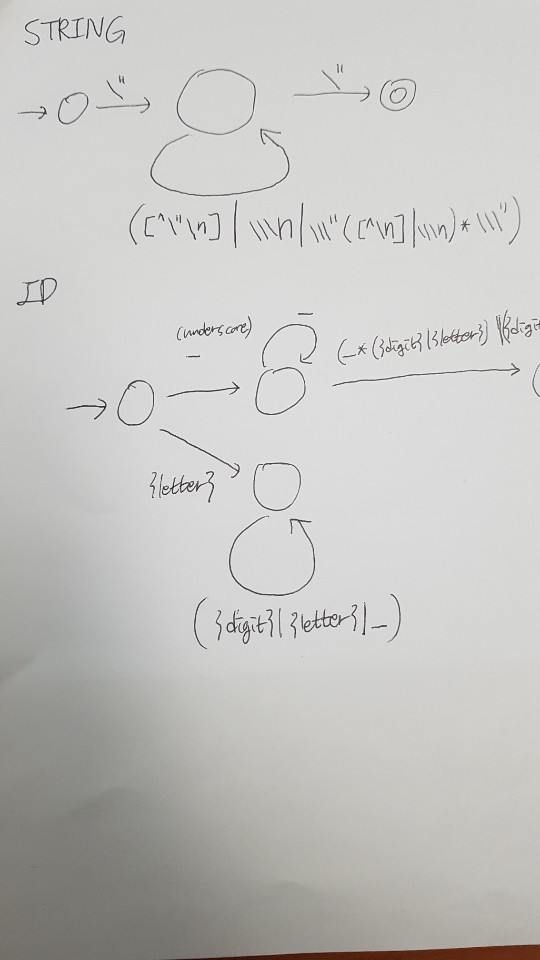
로 구성되어 있다.

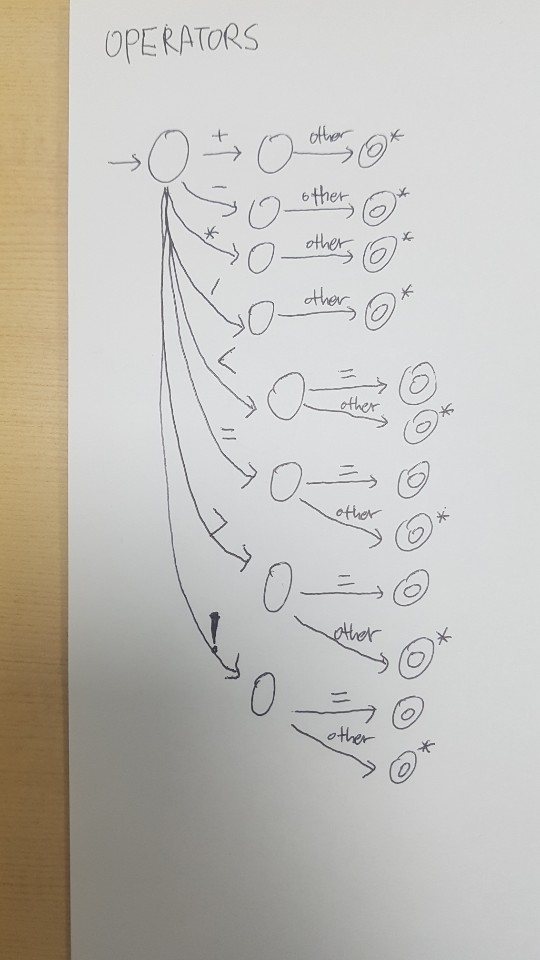
* Transition diagram

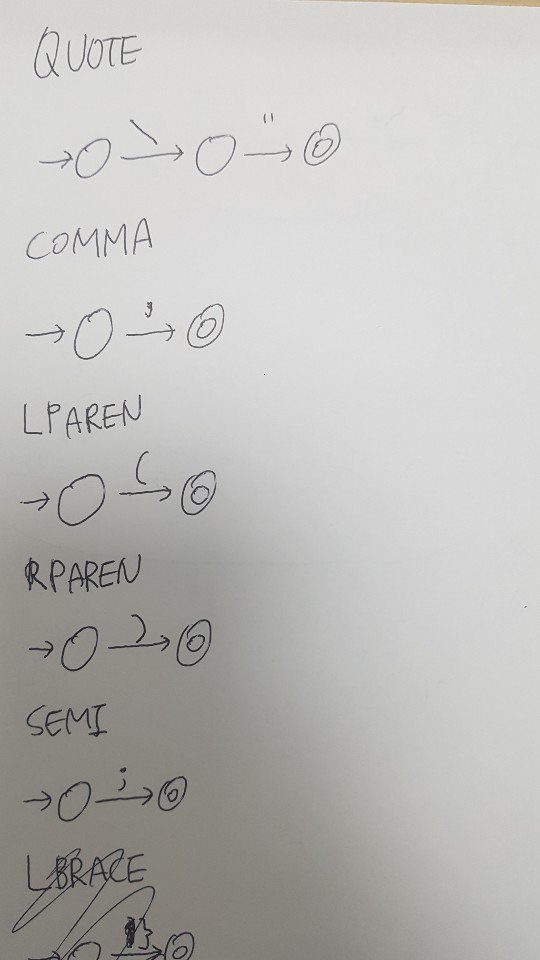


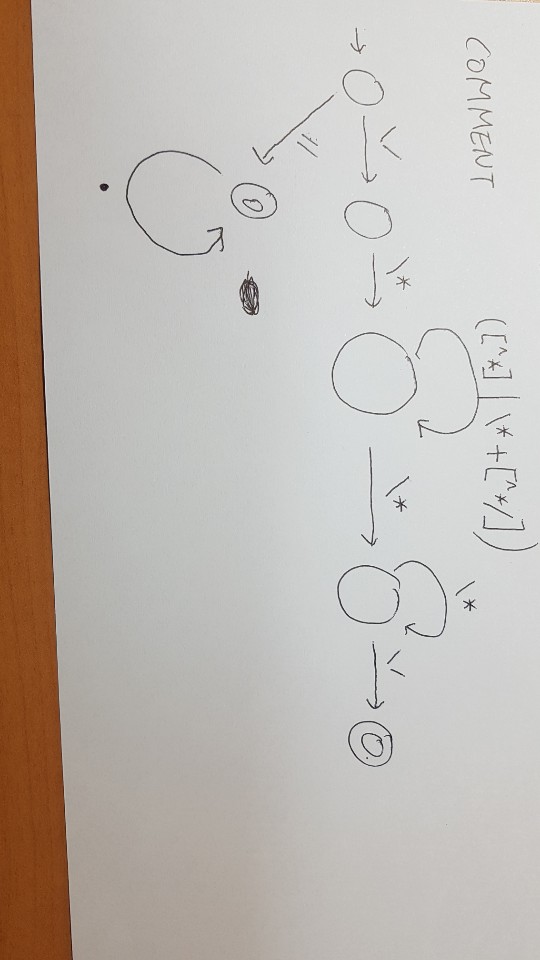
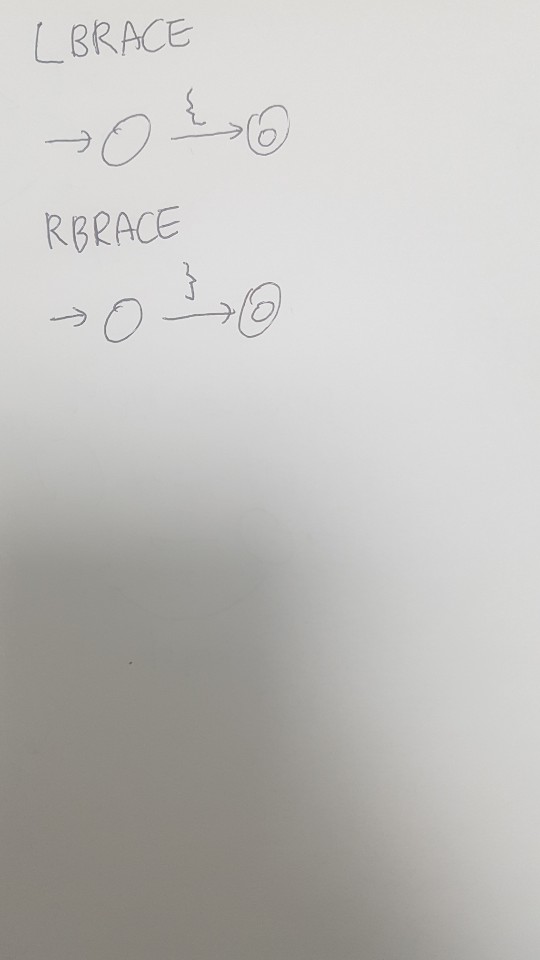


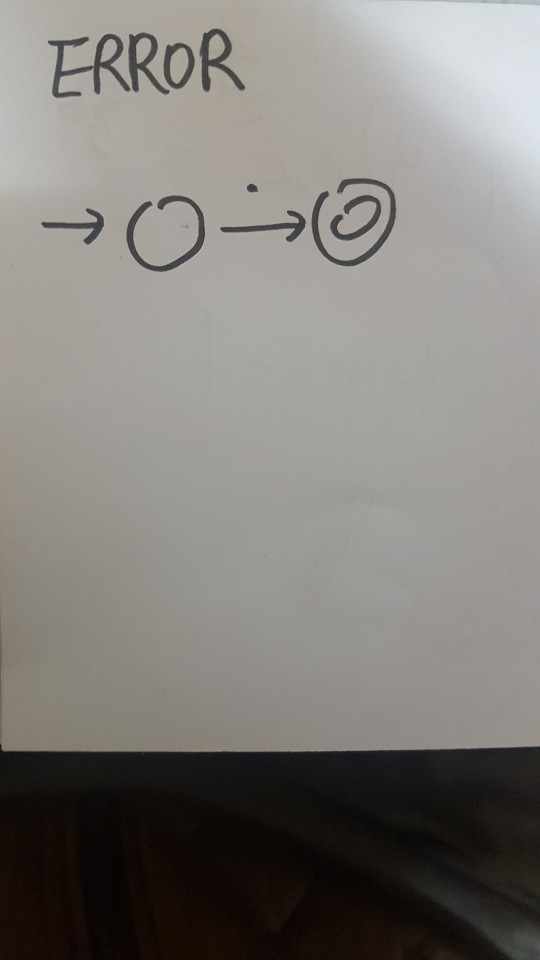












* Regular expression
* %{

blank [ \t\n]+

digit [0-9]

letter [A-Za-z]

%}

{blank} ;

INT -> int

DOUBLE -> double

STR -> str

IF -> if

WHILE -> while

RETURN -> return

ID -> (\_+)(\_\*({digit}|{letter})|({digit}|{letter})\_\*)+|{letter}({digit}|{letter}|\_)\*

INTEGER -> 0|[1-9]{digit}\*

DOUBLE\_NUM -> {digit}+(\.{digit}+)?((“E”|”e”)(“+”|”-“)?{digit}+)?

STRING -> \”([^\”\n]|\\\n|\\\”([^\\n]|\\\n)\*\\\”)\*\”

OPERATORS -> “+”|”-“|”\*”|”/”|”=”|”>”|”>=”|”<”|”<=”|”==”|”!=”

QOUTE -> “\””

COMMA -> “,”

LPAREN -> “(“

RPAREN -> “)”

SEMI -> “;”

LBRACE -> “{“

RBRACE -> “}”

COMMENT -> \/\\*([^\*]|\\*+[^\*/])\*\\*+\/|”//”.\*

ERROR -> .

1. 이번 과제를 수행하면서 string table 과 symbol table을 구성하기 위해서 구조체 string\_table과 symbol\_table을 정의하였다. linked list를 이용하여 table을 구성하였고 각각의 테이블에 이미 존재하는 symbol이나 string인지를 확인한 후 없는경우 linked list에 추가해주는 add\_symbol, add\_string 함수를 구성하였고 이미 존재하는 symbol이나 string의 경우 index를 출력해주기 위한 string\_index, symbol\_index 함수를 구성하였다. 그리고 symbol table과 string table을 출력한 뒤에 동적할당 된 메모리를 모두 해제했다. Integer의 길이를 확인하고 integer의 길이가 10자리보다 크면 뒤에서 부터 10자리를 잘라서 출력하기 위한 integer\_convert함수를 구성하였고 id길이도 16자리보다 크면 앞에서부터 16자리만을 찾아서 구분하기 위한 id\_convet함수를 구성하였다. 또한 id와 string의 패턴이 확인되는 횟수를 확인하기 위해 id\_counter와 string\_counter를 extern 선언으로 정의하였다.
2. 입력 데이터(파일명 : ex1.txt, ex2.txt)

* ex1.txt

int main(int a, double b)

{

char str = “asdasd\

sdasdsa”;

if(a==0)

{

return 2;

}

while(b!=0)

{

return 1;

}

int x = 1234567890123;

double y = 1.21315e+5;

double z = 1.3123e2;

1>2;

2<1;

3>=1;

1<=3; /\* judge ex1.txt \*/

}

* ex2.txt

int f(double a, int v)

{

int \_a\_2\_3 = 20; //\_a\_2\_var //

int a\_\_\_\_ = 30; /\* a\_\_\_\_\_

var \*/

int \_2\_\_\_ = 40; //\_2\_\_\_ var \*/

int \_\_\_\_\_3 = 50; ///\* \_\_\_\_3 var

int \_\_\_2\_\_ = 60;

int \_ = 10;

int \_\_\_ = 15;

“abc””;

“abcd”””;

double a = 32.23E+2;

double we = -0.23e+2;

int abcdefghijklmnopqrstuvwxyz = 12312312312;

if(a==2)

{

a=3;

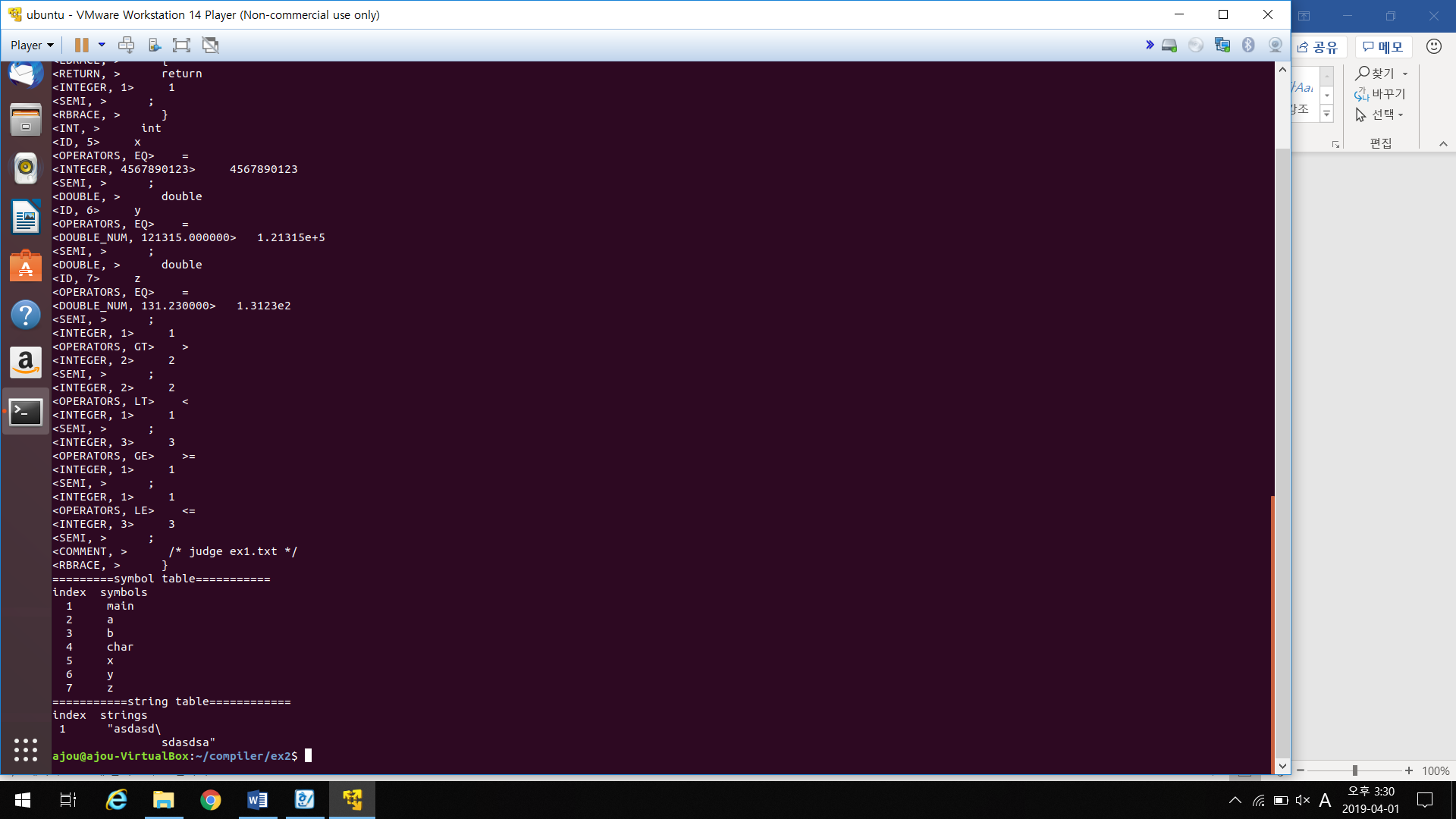
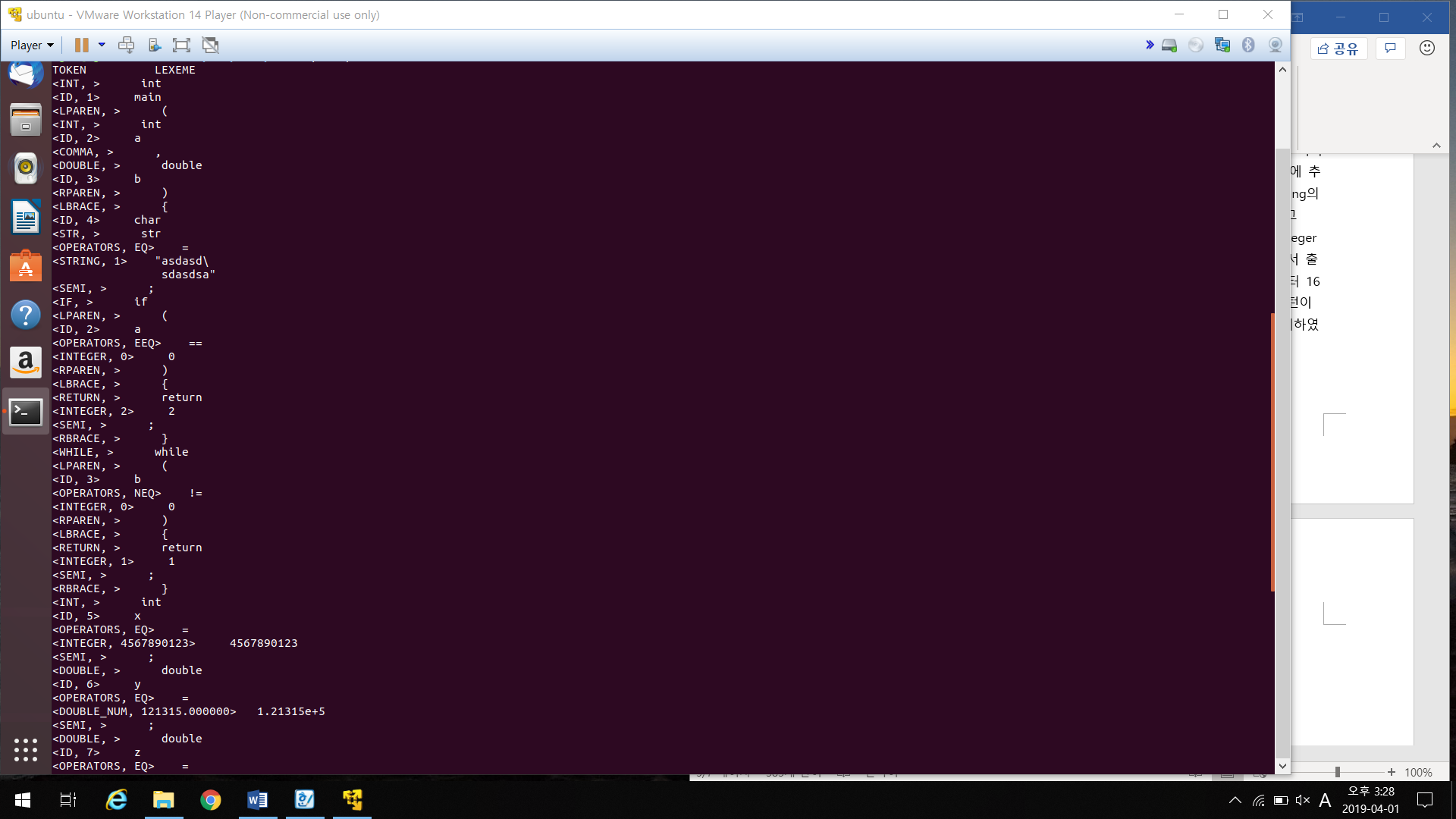
}

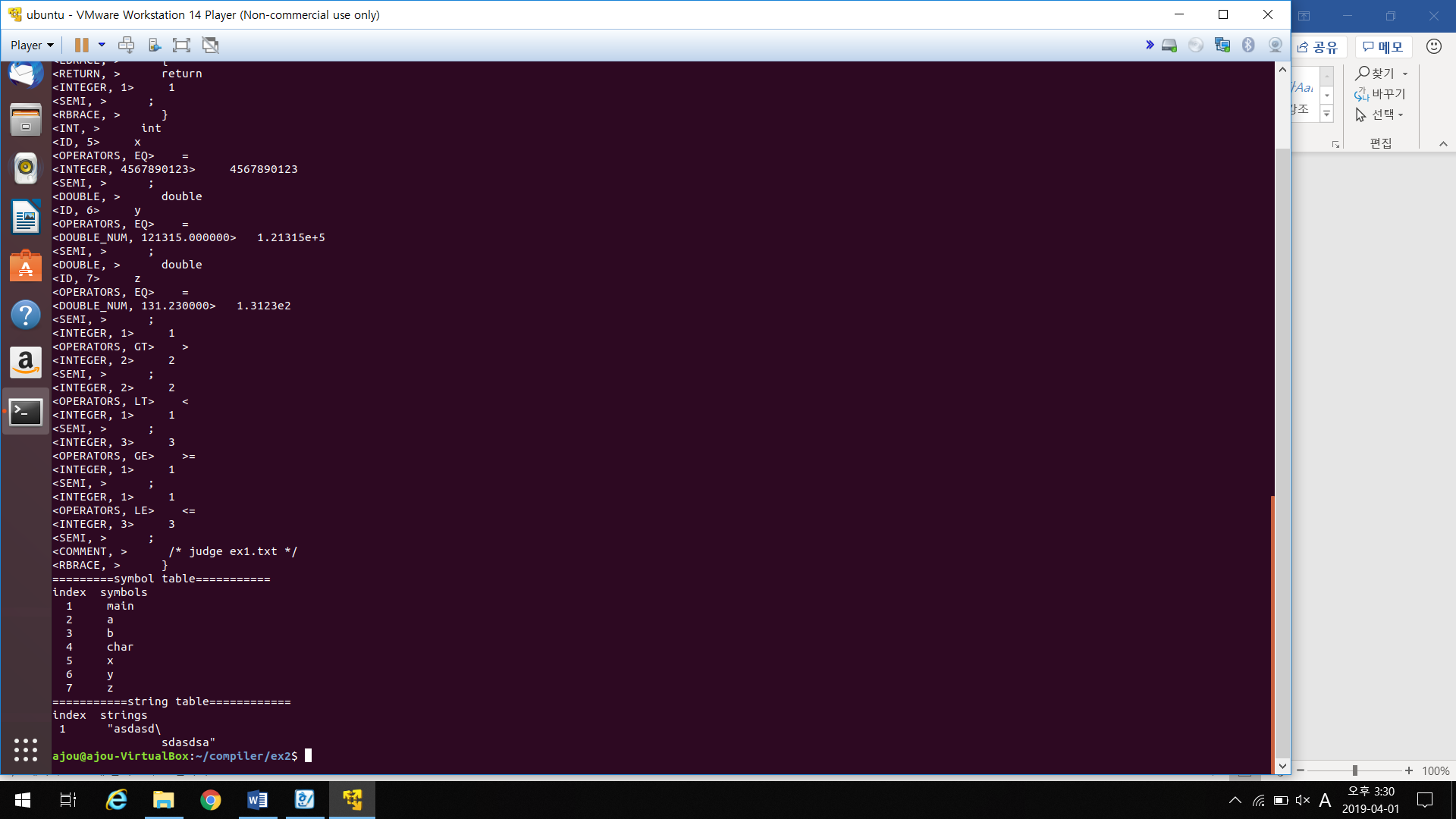
int a&^# = 23;

}

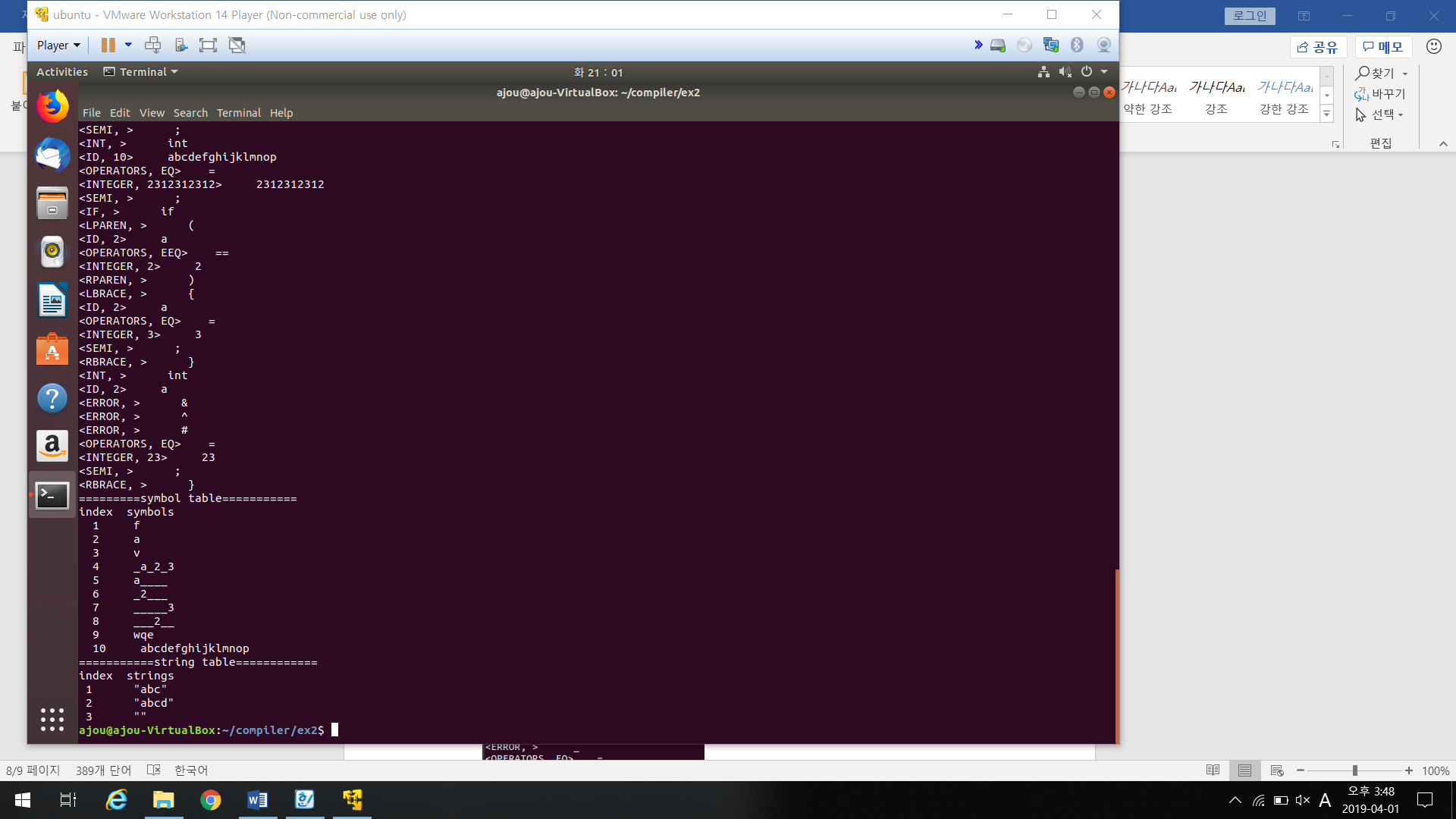
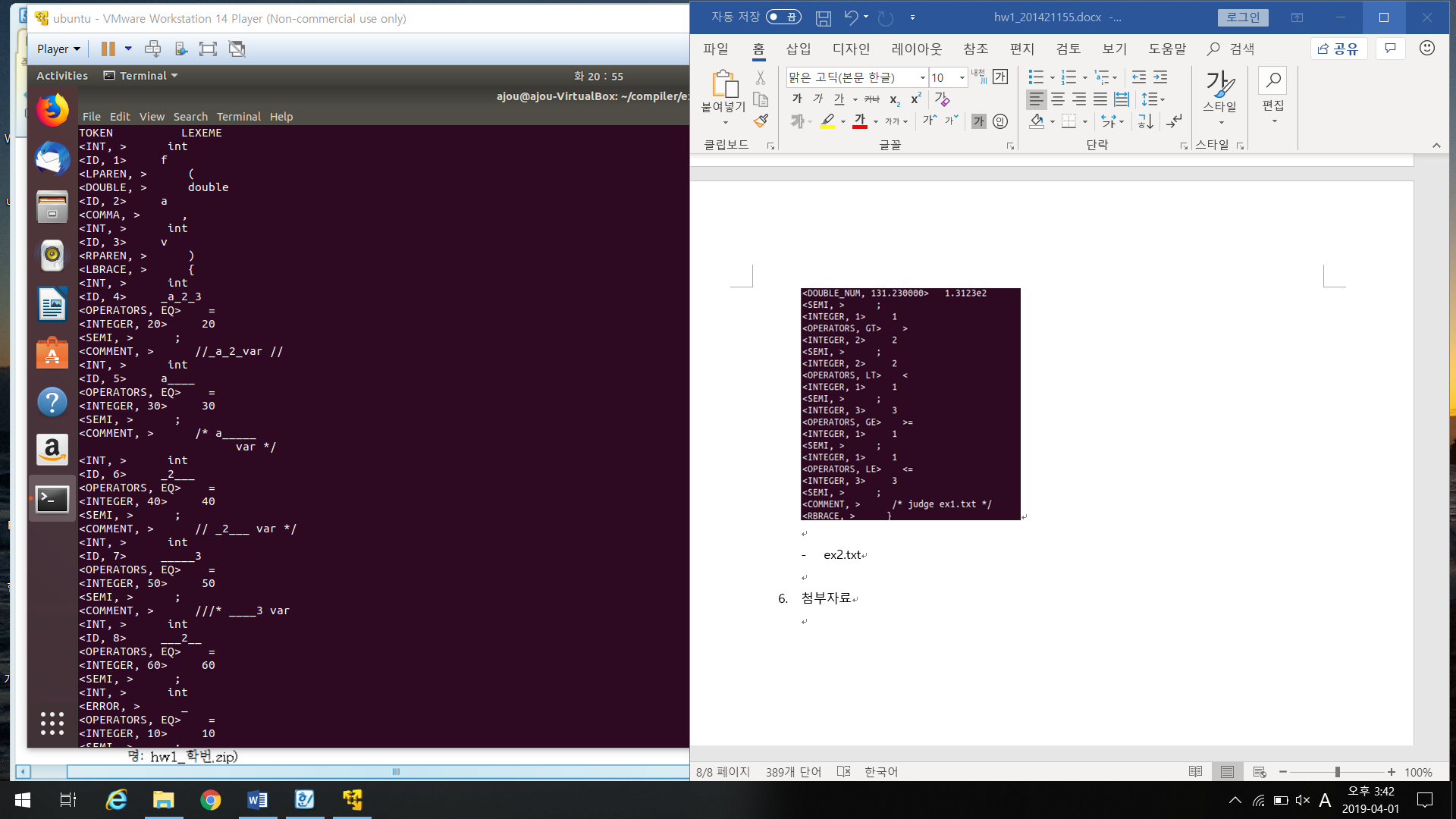
1. 결과 데이터(토큰리스트, 심볼테이블, 스트링테이블)

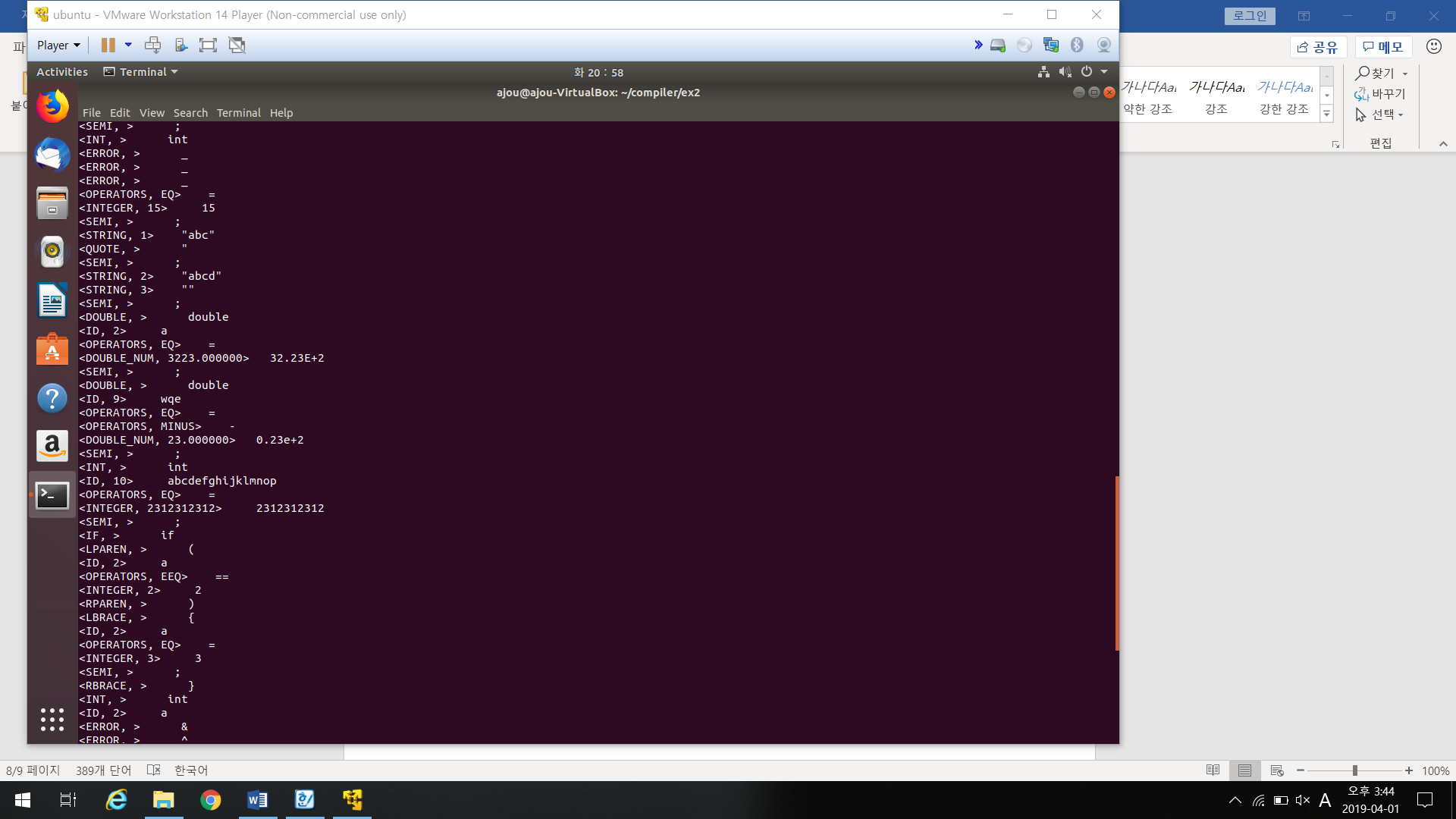
* ex1.txt

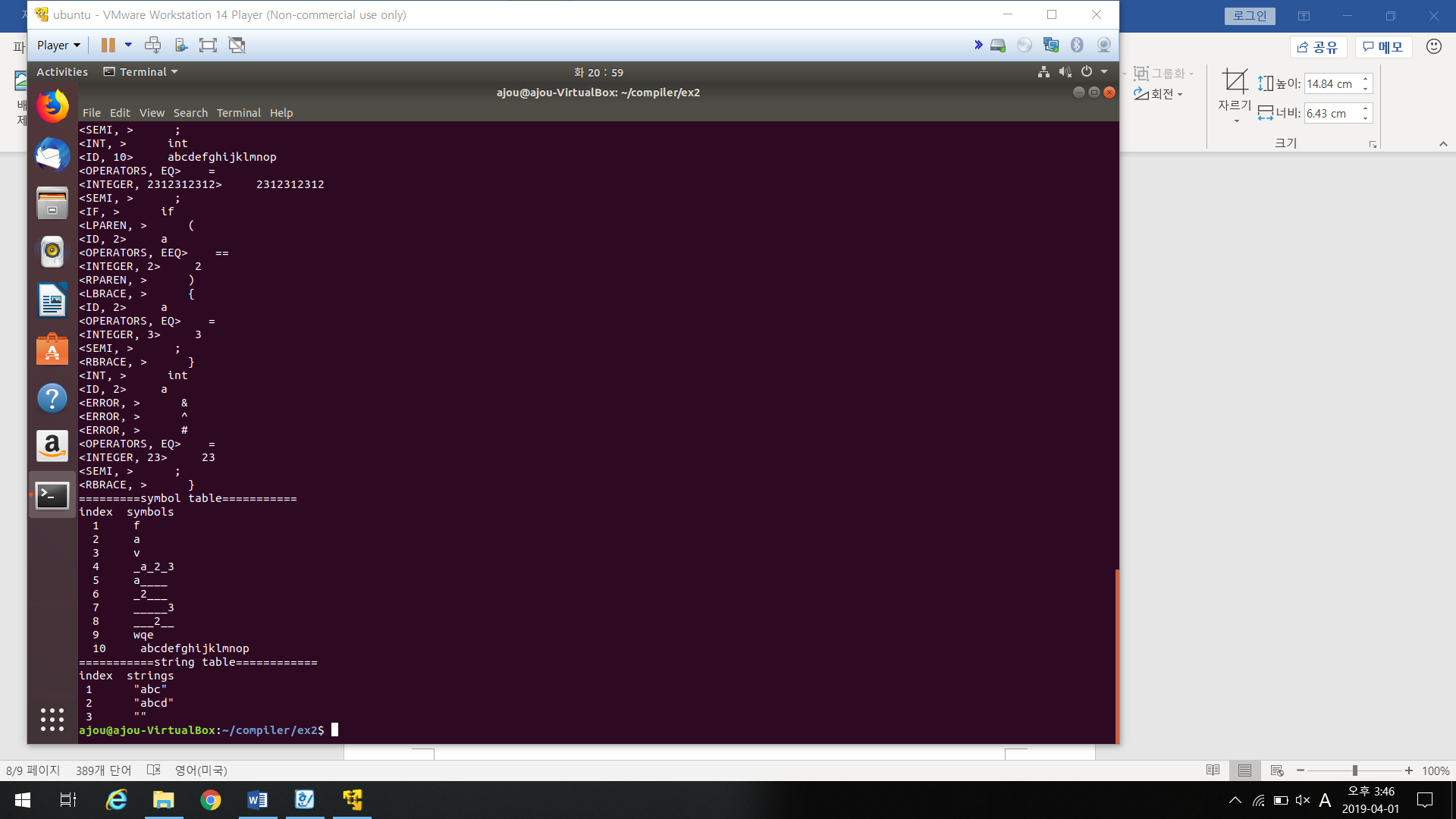




* ex2.txt







1. 첨부자료 : 소스 프로그램

* hw1.l
* %{
* extern int id\_counter;
* extern int string\_counter;
* extern char \* yytext;
* extern int yyleng;
* #define INT 1
* #define DOUBLE 2
* #define STR 3
* #define IF 4
* #define WHILE 5
* #define RETURN 6
* #define ID 7
* #define INTEGER 8
* #define DOUBLE\_NUM 9
* #define STRING 10
* #define OPERATORS 11
* #define QUOTE 12
* #define COMMA 13
* #define LPAREN 14
* #define RPAREN 15
* #define SEMI 16
* #define LBRACE 17
* #define RBRACE 18
* #define COMMENT 19
* #define ERROR 20
* %}
* blank [ \t\n]+
* digit [0-9]
* letter [A-Za-z]
* %%
* {blank} ;
* "if" {return IF;}
* "int" {return INT;}
* "double" {return DOUBLE;}
* "str" {return STR;}
* "while" {return WHILE;}
* "return" {return RETURN;}
* (\_+)(\_\*({digit}|{letter})|({digit}|{letter})\_\*)+|{letter}({digit}|{letter}|\_)\* {++id\_counter; return ID;}
* 0|[1-9]{digit}\* {return INTEGER;}
* {digit}+(\.{digit}+)?(("E"|"e")("+"|"-")?{digit}+)? {return DOUBLE\_NUM;}
* \"([^\"\n]|\\\n|\\\"([^\n]|\\\n)\*\\\")\*\" {++string\_counter; return STRING;}
* "+"|"-"|"\*"|"/"|"="|">"|">="|"<"|"<="|"=="|"!=" {return OPERATORS;}
* "\"" {return QUOTE;}
* "," {return COMMA;}
* "(" {return LPAREN;}
* ")" {return RPAREN;}
* ";" {return SEMI;}
* "{" {return LBRACE;}
* "}" {return RBRACE;}
* \/\\*([^\*]|\\*+[^\*/])\*\\*+\/|"//".\* {return COMMENT;}
* . {return ERROR;}
* %%
* #include<stdio.h>
* #include<stdlib.h>
* #include<string.h>
* struct symbol\_table
* {
* int symbol\_index;
* char id[17];
* struct symbol\_table \* symbol\_next;
* };
* struct string\_table
* {
* int string\_index;
* char \* string;
* struct string\_table \* string\_next;
* };
* int id\_counter=0;
* int string\_counter=0;
* extern FILE \*yyin;
* extern char \*yytext;
* extern int yyleng;
* void add\_symbol(struct symbol\_table \*\* first\_address,char \*symbol,int \*\* id\_count)
* {
* struct symbol\_table \* new = (struct symbol\_table\*)malloc(sizeof(struct symbol\_table));
* strcpy(new->id,symbol);
* new->symbol\_index = \*\*id\_count;
* \*first\_address = new;
* }
* int symbol\_index(struct symbol\_table \* first,char \*symbol,int \* id\_count)
* {
* struct symbol\_table \* s\_next;
* s\_next = first->symbol\_next;
* while(s\_next!=NULL)
* {
* if(strcmp(symbol,s\_next->id)==0)
* {
* \*id\_count = \*id\_count-1;
* return s\_next->symbol\_index;
* }
* s\_next = s\_next->symbol\_next;
* }
* add\_symbol(&s\_next,symbol,&id\_count);
* while(first->symbol\_next!=NULL)
* {
* first = first->symbol\_next;
* }
* first->symbol\_next = s\_next;
* return \*id\_count;
* }
* void add\_string(struct string\_table \*\* h1\_address,char\*string1,int \*\*string\_count)
* {
* struct string\_table \* new1 = (struct string\_table\*)malloc(sizeof(struct string\_table));
* new1 -> string =(char\*)malloc(strlen(string1));
* strcpy(new1->string,string1);
* new1->string\_index = \*\*string\_count;
* \*h1\_address = new1;
* }
* int string\_index(struct string\_table \* h1,char\*string1,int \*string\_count)
* {
* struct string\_table \* st\_next;
* st\_next = h1 -> string\_next;
* while(st\_next!=NULL)
* {
* if(strcmp(string1,st\_next->string)==0)
* {
* \*string\_count = \*string\_count-1;
* return st\_next->string\_index;
* }
* st\_next = st\_next->string\_next;
* }
* add\_string(&st\_next,string1,&string\_count);
* while(h1->string\_next!=NULL)
* {
* h1=h1->string\_next;
* }
* h1->string\_next = st\_next;
* return \*string\_count;
* }
* int OP\_DECIDE(char \* a)
* {
* if(strcmp(a,"+")==0)
* return 1;
* else if(strcmp(a,"-")==0)
* return 2;
* else if(strcmp(a,"\*")==0)
* return 3;
* else if(strcmp(a,"/")==0)
* return 4;
* else if(strcmp(a,"=")==0)
* return 5;
* else if(strcmp(a,">")==0)
* return 6;
* else if(strcmp(a,">=")==0)
* return 7;
* else if(strcmp(a,"<")==0)
* return 8;
* else if(strcmp(a,"<=")==0)
* return 9;
* else if(strcmp(a,"==")==0)
* return 10;
* else if(strcmp(a,"!=")==0)
* return 11;
* }
* char\* integer\_convert(int i\_len,char \*b,char \* it\_temp)
* {
* if(strncmp(b,"-",1)==0)
* {
* if(i\_len<=11)
* return b;
* else
* {
* strncpy(it\_temp,"-",1);
* strcpy(it\_temp+1,b+(i\_len-10));
* return it\_temp;
* }
* }
* else
* {
* if(i\_len<=10)
* return b;
* else
* {
* strcpy(it\_temp,b+(i\_len-10));
* return it\_temp;
* }
* }
* }
* char\* id\_convert(int id\_len,char \* c,char \*id\_temp)
* {
* if(id\_len<=16)
* return c;
* else
* {
* strncpy(id\_temp,c,16);
* return id\_temp;
* }
* }
* int main(int argc, char\*argv[])
* {
* char \* s1[20]={"INT","DOUBLE","STR","IF","WHILE","RETURN","ID","INTEGER","DOUBLE\_NUM","STRING","OPERATORS","QUOTE","COMMA","LPAREN","RPAREN","SEMI","LBRACE","RBRACE","COMMENT","ERROR"};
* char \* s2[11]={"PLUS","MINUS","MULTI","DIVIDE","EQ","GT","GE","LT","LE","EEQ","NEQ"};
* struct symbol\_table \* symbol\_head=(struct symbol\_table\*)malloc(sizeof(struct symbol\_table));
* struct string\_table \* string\_head=(struct string\_table\*)malloc(sizeof(struct string\_table));
* symbol\_head->symbol\_index = 0;
* symbol\_head->symbol\_next = NULL;
* if(argc>1)
* {
* FILE \* file;
* file = fopen(argv[1],"r");
* if(!file)
* {
* fprintf(stderr,"could not open %s\n",argv[1]);
* exit(1);
* }
* yyin=file;
* printf("TOKEN LEXEME\n");
* int tok;
* char \* it\_temp;
* char \* id\_temp;
* while((tok=yylex())!=0)
* {
* it\_temp =(char\*)malloc(12);
* id\_temp =(char\*)malloc(20);
* if(tok==7)
* {
* printf("<%s, %d> %s\n",s1[tok-1],symbol\_index(symbol\_head,id\_convert(yyleng,yytext,id\_temp),&id\_counter),id\_convert(yyleng,yytext,id\_temp));
* }
* else if(tok==8)
* {
* printf("<%s, %s> %s\n",s1[tok-1],integer\_convert(yyleng,yytext,it\_temp),integer\_convert(yyleng,yytext,it\_temp));
* }
* else if(tok==9)
* {
* printf("<%s, %f> %s\n",s1[tok-1],atof(yytext),yytext);
* }
* else if(tok==10)
* {
* printf("<%s, %d> %s\n",s1[tok-1],string\_index(string\_head,yytext,&string\_counter),yytext);
* }
* else if(tok==11)
* {
* printf("<%s, %s> %s\n",s1[tok-1],s2[OP\_DECIDE(yytext)-1],yytext);
* }
* else
* {
* printf("<%s, > %s\n",s1[tok-1],yytext);
* }
* free(it\_temp);
* free(id\_temp);
* }
* fclose(yyin);
* }
* struct symbol\_table \*x;
* x = symbol\_head->symbol\_next;
* printf("=========symbol table===========\n");
* printf("index symbols\n");
* while(x!=NULL)
* {
* struct symbol\_table \* temp;
* printf(" %d %s\n",x->symbol\_index,x->id);
* temp = x;
* x = x->symbol\_next;
* free(temp);
* }
* struct string\_table \*y;
* y = string\_head->string\_next;
* printf("===========string table============\n");
* printf("index strings\n");
* while(y!=NULL)
* {
* struct string\_table \*temp1;
* printf(" %d %s\n",y->string\_index,y->string);
* temp1 = y;
* y=y->string\_next;
* free(temp1->string);
* free(temp1);
* }
* free(symbol\_head);
* free(string\_head);
* }