2021-01

프로그래밍언어론(001분반)

Lab01

보고서

**학과:소프트웨어융합전공**

**학번: 1912339**

**이름: 신지우**

**제출일: 2021/3/16**

1. 구현한 프로그램에 대한 설명

사용자에게 입력받은 수식은 먼저 parse()를 향한다. parse()에서 getToken()을 통해 첫번째 토큰을 받아온 후, command()에서 expr() 값을 받아와 'The result is: ~~~~' 형태로 값을 출력하고, hasRelop 변수를 0으로 세팅하는 것이 한 사이클이다. hasRelop 변수는 비교연산이 있거나 expr()값이 true, false일 때(논리연산을 하는 경우는 비교연산이나 true,false가 반드시 선행되므로 생략했다.) 함수 expr() 리턴값을 boolean으로 설정해주기 위해 존재한다. hasRelop이 0이면 int, hasRelop이 1이면 boolean으로 반환한다.

expr()에서는 우선 !(not 연산)이 맨앞에 존재할 경우 재귀적으로 expr()을 불러와 모든 계산을 끝마친 후 !(not 연산)이 적용되도록 가장 바깥 if-else문을 구성하였다.(<expr>-> !<expr>) 첫번째 토큰이 !(not 연산)이 아닌 모든 경우는 else로 들어가 bexp() 함수에서 값을 받아오고, 첫번째 안쪽 if-else if문에서 (<expr>->true | false) true나 false인지 확인한다. 이때 true나 false에 해당하면 hasRelop을 1로 설정한다. 두번째 안쪽 if-else if문에서 (<expr>-><bexp>{'&'<bexp> | '|'<bexp>}) 논리연산을 수행하는지 확인한다. 논리연산을 여러번 수행할 수 있도록 논리연산할 또다른 피연산자를 bexp()가 아니라 expr()로 받아줬다. 마지막 안쪽 if-else문에서 hasRelop의 값에 따라 0이면 사칙연산만 존재하므로 int값으로 반환하고, 1이면 boolean값으로 반환한다.

bexp()에서는 일단 aexp()에서 int값을 받아온다. 그 후 token이 비교연산자의 서두('<', '>', '!', '=')에 해당하면 switch문에서 relop()을 통해 어떤 비교연산자인지 확인 후, 비교연산을 수행하여 boolean값을 expr로 반환한다.(<bexp>-><aexp>[<relop><aexp>]) 하지만 그 외에 해당하면(숫자나 true의 t, false의 f) int값으로 반환한다.(<bexp>-><aexp>)

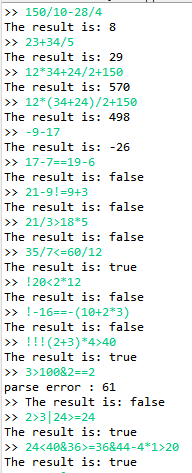
relop()에서는 이 비교연산이 어떤 비교연산자인지를 확인하여 bexp()에 넘겨주는 역할을 한다. 우선 비교연산의 서두로 확실한 '<', '>', '!', '='을 받아 '!'는 바로 '!='로, '='는 바로 '=='로 처리해줬고, '<'과 '>'는 그 다음 토큰에 따라 '<', '>','<=', '>='을 구분했다.(<relop>->== | != | < | > | <= | >=)

aexp()에서는 일단 term()에서 int값을 받아온다. 그 후 token이 '+', '-'이면 재귀적으로 피연산자를 불러와 여러 번 연산할 수 있도록 구현했다. 만약 token이 '+', '-'에 해당하지 않으면 바로 int값을 반환했다.(<aexp>->term{+<term>|-<term>)

term()에서는 일단 factor()에서 int값을 받아온다. 그 후 token이 '\*', '/'이면 재귀적으로 피연산자를 불러와 여러 번 연산할 수 있도록 구현했다. 만약 token이 '\*', '/'에 해당하지 않으면 바로 int값을 반환했다.(<term>-> <factor>{\*<factor>|/<factor>})

factor()에서는 현재 token이 괄호'()'면 괄호 안을 먼저 계산하도록 if문이 구성되어있다. 이후 token이 숫자면 token이 다음 순서를 가르키게 하고 int값을 반환했다.(<factor> -> [-](number | (<aexp>))

2. 실행 결과 캡처 사진



3. 프로그램 소스 코드 (복사-붙여넣기)

import java.io.IOException;

import java.io.PushbackInputStream;

class Calc {

int token; int value; int ch; int hasRelop=0;

private PushbackInputStream input;

final int NUMBER=256;

Calc(PushbackInputStream is) {

input = is;

}

int getToken( ) { /\* tokens are characters \*/

while(true) {

try {

ch = input.read();

if (ch == ' ' || ch == '\t' || ch == '\r') ;

else if (Character.isDigit(ch)) {

value = number( );

input.unread(ch);

return NUMBER;

}

else return ch;

} catch (IOException e) {

System.err.println(e);

}

}

}

private int number( ) {

/\* number -> digit { digit } \*/

int result = ch - '0';

try {

ch = input.read();

while (Character.isDigit(ch)) {

result = 10 \* result + ch -'0';

ch = input.read();

}

} catch (IOException e) {

System.err.println(e);

}

return result;

}

void error( ) {

System.out.printf("parse error : %d\n", ch);

//System.exit(1);

}

void match(int c) {

if (token == c)

token = getToken();

else error();

}

void command( ) {

/\* command -> expr '\n' \*/

Object result = expr();

if (token == '\n') /\* end the parse and print the result \*/

System.out.println("The result is: "+result);

else error();

}

Object expr() {

///expr -> bexp { '&' bexp | '|' bexp} | '!' expr | true | false

if(token=='!') {

match('!');

return !(boolean) expr();

}

else {

Object result = bexp();

if (token == 't') { //true

match('t');

match('r');

match('u');

match('e');

result = true;

hasRelop = 1;//Relop을 갖지는 않지만 boolean 처리를 해주기 위해

}

else if (token == 'f') { //false

match('f');

match('a');

match('l');

match('s');

match('e');

result = false;

hasRelop = 1;

}

if(token=='&') {

match('&');

result = (boolean) result && (boolean) expr();

}

else if(token=='|') {

match('|');

result = (boolean) result || (boolean) expr();

}

if(hasRelop==1)

return (boolean) result;

else

return (int) result;

}

}

Object bexp() {

int result = aexp();

if(token=='<'||token=='>'||token=='!'||token=='=') {

hasRelop=1;

switch(relop()) {

case "<":

return (boolean)(result < aexp());

case ">":

return (boolean)(result > aexp());

case "<=":

return (boolean)(result <= aexp());

case ">=":

return (boolean)(result >= aexp());

case "!=":

return (boolean)(result != aexp());

default:

return (boolean)(result == aexp());

}

}

return result;

}

String relop() {

if(token=='<') {

match('<');

if(token=='=') {

match('=');

return "<=";

}

else

return "<";

}

else if(token=='>') {

match('>');

if(token=='=') {

match('=');

return ">=";

}

else

return ">";

}

else if(token=='!') {

match('!');

match('=');

return "!=";

}

else {

match('=');

match('=');

return "==";

}

}

int aexp( ) {

/\* aexp -> term { '+' term | '-' term} \*/

int result = term();

while (token == '+' || token == '-') {

if(token == '+'){

match('+');

result += term();

}

else {

match('-');

result -= term();

}

}

return result;

}

int term( ) {

/\* term -> factor { '\*' factor | '/' factor } \*/

int result = (int) factor();

while (token == '\*' || token == '/') {

if(token == '\*'){

match('\*');

result \*= term();

}

else {

match('/');

result /= term();

}

}

return result;

}

Object factor() {

/\* factor -> '(' expr ')' | number \*/

Object result = 0;

if (token == '(') {

match('(');

result = aexp();

match(')');

}

else if (token == NUMBER) {

result = value;

match(NUMBER); //token = getToken();

}

return result;

}

void parse( ) {

token = getToken(); // get the first token

command(); // call the parsing command

hasRelop=0;

}

public static void main(String args[]) {

Calc calc = new Calc(new PushbackInputStream(System.in));

while(true) {

System.out.print(">> ");

calc.parse();

}

}

}