Compiladors

Pràctica 1: Calculadora

NEUS OLLER MATAS

Enginyeria Informàtica

Universitat Rovira i Virgili

1 Introducció

La calculadora ha de suportar:

- Literals: serien els enters, reals, cadenes i booleans
- Comentaris: //, /* ... */, i també amb #).
- Expressions aritmètiques, booleanes i assignacions.
- Funcions trigonomètriques i funcions de cadena (LEN, SUBSTR).
- Constants predeterminades (PI, E).
- Diversos formats numèrics (octal, binari, hexadecimal, decimal).

La sortida ha de mostrar el tipus i valor de les expressions i assignacions.
A més, s'ha generar un arxiu de log per registrar les produccions reconegudes per la gramàtica.

2 Fitxer Flex

El nom del fitxer s'anomena "*calculator.l*". Identifica els diferents *tokens* del llenguatge, com literals, identificadors, operadors, etc.

2.1 Literals i comentaris

Literals

```
 /* Enter */
int [0-9]*
```

- [0-9]*: expressió regular que fa coincidir qualsevol nombre de dígits d'entre el 0 i el 9
- * : indica que hi ha 0 o més números (no-limitat)

```
 /* Real */
float [0-9]*\.[0-9]*([eE][-+]?[0-9]+)?
```

- [0-9]*: actual igual que en el identificador enter (abans d'arribar a ' . ')
- \. : accepta el '.' a l'expressió regular
- [0-9]*: Aquesta part fa coincidir qualsevol nombre de dígits després del punt decimal
- ([eE][-+]?[0-9]+)?: per capturar notació científica:
  - [eE]: es fan servir per a representar l'exponent en notació científica
  - [-+]?: en cas que es posi si és positiu (+) o negatiu (-) per l'exponent
  - [0-9]+: per tenir més d'1 dígit a l'exponent
  - ?: la notació científica pot estar present o no

```
 /* Cadena de caràcters */
string \".*\"
```

Com que l'String es representa amb cometes dobles doncs s'inicia (\") i es finalitza (\") amb doble cometes.

- .*: escriu qualsevol cosa com a caràcters. S'acaba quan es troba una cometa doble.

```
 /* Números octals */
octal 0[oO][0-7]+
```

Són aquells números que comencen amb un 0 seguit de la lletra *o* o *O* i, després un o més dígits octals ([0 ... 7]).

- 0: comença amb número 0 per identificar nombres en sistemes de numeració com binari, octal o hexadecimal.
- [oO]: el següent caràcter ha de ser una lletra *o* o *O*. Està entre claudàtors ja que indica que pot ser qualsevol d'aquests dos caràcters.
- [0-7]+: Després de la lletra *o* o *O*, ha d'haver-hi un o més dígits entre 0 i 7. El símbol + indica que ha de haver-hi almenys un d'aquests dígits (pot ser més d'un).

```
 /* Números binaris */
binari 0[bB][01]+
```

Aquesta expressió està pensada per tenir un format com 0b1010 o 0B1010.

- 0: comença amb número 0 per identificar nombres en sistemes de numeració com binari, octal o hexadecimal.
- [bB]: Després del 0, ha d'aparèixer la lletra b o B, que indica que és un número binari. [bB] permet tant majúscules com minúscules.
- [01]+: Aquest fragment especifica que després de b o B ha d'haver-hi una seqüència d'un o més dígits binaris (0 o 1). El símbol + indica que hi ha d'haver almenys un dígit binari, però poden ser més.

```
 /* Números hexadecimals */
hexadecimal 0[xX][0-9a-fA-F]+
```

L'expressió s'utilitza per identificar nombres hexadecimal escrits amb el prefix estàndard 0x o 0X, seguit de lletres i números vàlids en hexadecimal.

- 0: comença amb número 0 per identificar nombres en sistemes de numeració com binari, octal o hexadecimal.
- [xX]: Després del 0, hi ha la lletra x o X, que indica que és un número hexadecimal.
- [0-9a-fA-F]+:
  - [0-9]: Números del 0 al 9
  - a-f: Lletres de la a a la f (representen valors de 10 a 15 en hexadecimal).
  - A-F: Lletres majúscules equivalents a la a fins a la f (també valors de 10 a 15).
  - El símbol + especifica que hi ha d'haver almenys un dígit hexadecimal, però poden ser més.

Finalment, respecte els números decimals, es tracten sense prefix, per tant serà qualsevol altre número que es tracti al programa realitzat. Està definit al patró de int.

Comentaris

S'ha implementat:

```
 /* COMENTARI SIMPLE */

"//".*         { yylval.var.type = STRING;

                 yylval.var.value.sval = strdup(yytext);

                 return SCOMMENT; }

"#".*          { yylval.var.type = STRING;

                 yylval.var.value.sval = strdup(yytext);

                 return SCOMMENT; }
```

```
 /* COMENTARI MÚLTIPLE */

"/*"([^#=].|\n)*"*/"    { yylval.var.type = STRING;

                        yylval.var.value.sval = strdup(yytext); /* copio tot el comentari */

                        return MCOMMENT; }
```

2.2 Identificadors i Expressions

```
 /* ENTER */

{int}           { yylval.var.type = INTEGER; yylval.var.value.ival = atoi(yytext); return INT; }
```

- {int}: identificador que es refereix a un tipus de dada integer
- yylval.var.type = INTEGER: Assigna el tipus *INTEGER* a la variable *yylval.var.type*. La variable l'utilitzaré per passar informació entre el *lexer* i el *parser*
- yylval.var.value.ival = atoi(yytext): *atoi(yytext)* converteix la cadena de text *yytext* a un enter (int) i ho assigna a la variable *yylval.var.value.ival*
- return INT: retorna amb el valor INT, que és utilitzat pel *parser*

```
 /* REAL */

{float}         { yylval.var.type = FLOAT; yylval.var.value.fval = atof(yytext); return FL; }
```

- {float}: identificador que es refereix a un tipus de dada float
- yylval.var.type = FLOAT: Assigna el tipus *FLOAT* a la variable *yylval.var.type*
- yylval.var.value.fval = atof(yytext): *atof(yytext)* converteix la cadena de text *yytext* a un flotant (float) i ho assigna a *yylval.var.value.fval*
- return FL: Es retorna el *token* corresponent amb el valor FL, que és utilitzat pel *parser*

```
 /* STRING */

{string}        { yylval.var.type = STRING;
                  int len = yyleng-2;
                  yylval.var.value.sval = (char*)malloc(sizeof(char)*len+1);
                   strncpy(yylval.var.value.sval, yytext+1, len);
                  return STR; }
```

- {string}: identificador que es refereix a una entrada que conté una cadena de caràcters String
- yylval.var.type = STRING: Assigna el tipus *STRING* a la variable *yylval.var.type*
- int len = yyleng - 2: *yyleng* és la longitud de la cadena que es troba a *yytext*, però com que les cadenes en C estan envoltades per cometes (") en l'entrada, doncs es resta 2 per obtenir la longitud real de la cadena
- yylval.var.value.sval = (char*)malloc(sizeof(char)*len+1): Assigna memòria amb una longitud de len + 1 per incloure el caràcter de final de cadena '\0'.
- strncpy(yylval.var.value.sval, yytext+1, len): *strncpy* copia la subcadena de *yytext* que comença a partir de l'índex 1 (per ometre la primera cometa) fins a la longitud len
- return STR: Es retorna el *token* STR, que representa una cadena de caràcters

```
 /* OCTAL */

{octal}       { yylval.var.type = INTEGER;
                yylval.var.value.ival = strtol(yytext + 2, NULL, 8); /* Base 8 */
                 return INT; }
```

- {octal}: Identificador que conté una entrada en format octal
- yylval.var.type = INTEGER: Assigna el tipus INTEGER a la variable yylval.var.type
- yylval.var.value.ival = strtol(yytext + 2, NULL, 8): Conté el text que el lexer ha identificat com un nombre octal. El "strtol" converteix la cadena a un valor numèric de tipus long, especificant 8 com a base.
- return INT: Retorna el *token* INT, que representa un nombre enter.

```
 /* BINARI */

{binari}      { yylval.var.type = INTEGER;
                yylval.var.value.ival = strtol(yytext + 2, NULL, 2); /* Base 2 */
                return INT; }
```

- {binari}: Identificador que conté una entrada en format binari
- yylval.var.type = INTEGER: Assigna el tipus INTEGER a la variable yylval.var.type
- yylval.var.value.ival = strtol(yytext + 2, NULL, 2): Conté el text que el lexer ha identificat com un nombre octal. El "strtol" converteix la cadena a un valor numèric de tipus long, especificant 2 com a base.
- return INT: Retorna el *token* INT, que representa un nombre enter.

```
/* HEXADECIMAL */

{hexadecimal} { yylval.var.type = INTEGER;
                yylval.var.value.ival = strtol(yytext + 2, NULL, 16); /* Base 16 */
                 return INT; }
```

- {hexadecimal}: Identificador que conté una entrada en format hexadecimal
- yylval.var.type = INTEGER: Assigna el tipus INTEGER a la variable yylval.var.type
- yylval.var.value.ival = strtol(yytext + 2, NULL, 16): Conté el text que el lexer ha identificat com un nombre octal. El "strtol" converteix la cadena a un valor numèric de tipus long, especificant 16 com a base.
- return INT: Retorna el *token* INT, que representa un nombre enter.

```
 /* BOOLEÀ */

"true"          { yylval.var.type = BOOLEAN; yylval.var.value.bval = 1; return BOOL; }

"false"         { yylval.var.type = BOOLEAN; yylval.var.value.bval = 0; return BOOL; }
```

- "true" i "false": Aquestes dues regles fan coincidir les paraules true i false
- yylval.var.type = BOOLEAN: Assigna el tipus BOOLEAN a la variable *yylval.var.type*
- yylval.var.value.bval = 1: Per a *true*, es guarda el valor 1 com a valor booleà
- yylval.var.value.bval = 0: Per a *false*, es guarda el valor 0 com a valor booleà
- return BOOL: Es retorna el *token* BOOL, que indica que es tracta d'un valor booleà

2.3 Operadors Aritmètiques, Relacionals i Booleans

```
 /* OPERADORS ARITMÈTICS */
"**"            { yylval.var.type = STRING; yylval.var.value.sval = "**"; return POW; }

"*"             { yylval.var.type = STRING; yylval.var.value.sval = "*"; return MUL; }

"/"             { yylval.var.type = STRING; yylval.var.value.sval = "/"; return DIV; }

"%"             { yylval.var.type = STRING; yylval.var.value.sval = "%"; return MOD; }

"+"             { yylval.var.type = STRING; yylval.var.value.sval = "+"; return ADD; }

"-"             { yylval.var.type = STRING; yylval.var.value.sval = "-"; return SUB; }

"("             { return LPAREN; }

")"             { return RPAREN; }
```

```
/* OPERADORS RELACIONALS */
">"             { yylval.var.type = STRING; yylval.var.value.sval = ">"; return BOOLOP;  }

">="            { yylval.var.type = STRING; yylval.var.value.sval = ">="; return BOOLOP; }

"<"             { yylval.var.type = STRING; yylval.var.value.sval = "<"; return BOOLOP;  }

"<="            { yylval.var.type = STRING; yylval.var.value.sval = "<="; return BOOLOP; }

"="             { yylval.var.type = STRING; yylval.var.value.sval = "=="; return BOOLOP; }

"<>"            { yylval.var.type = STRING; yylval.var.value.sval = "<>"; return BOOLOP; }
```

```
/* OPERADORS BOOLEANS */

"not"           { return NOT; }

"and"           { return AND; }

"or"            { return OR;  }
```

```
/* ASSIGNACIONS */

":="            { return ASSIGN; }
```

2.4 Funcions trigonomètriques

```
/* FUNCIONS TRIGONOMÈTRIQUES */
```

```
"sin"           { yylval.var.type = STRING; yylval.var.value.sval = "sin"; return SIN; }

"cos"           { yylval.var.type = STRING; yylval.var.value.sval = "cos"; return COS; }

"tan"           { yylval.var.type = STRING; yylval.var.value.sval = "tan"; return TAN; }
```

2.5 Longitud de cadena

```
 /* LONGITUD DE CADENA */

"LEN"           { yylval.var.type = STRING; yylval.var.value.sval = "LEN"; return LEN; }
```

2.6 Extracció de subcadena (SUBSTR)

```
/* EXTRACCIÓ DE SUBCADENA -> SUBSTRING */

"SUBSTR"      { yylval.var.type = STRING; yylval.var.value.sval = "SUBSTR"; return SUBSTR; }
";"           { return ";"; }
```

2.7 Constants predeterminades (pi, e)

```
/* CONSTANTS PREDETERMINADES -> PI, E */

"pi"            { yylval.var.type = FLOAT; yylval.var.value.fval = 3.14159; return FL; }

"e"             { yylval.var.type = FLOAT; yylval.var.value.fval = 2.71828; return FL; }
```

2.8 Altres

```
\n { return EOL; } // salt de línia
"\t"            {} // tabulador
"\r"            {} // Ignora retorns de carro
" "             {} // espais

[a-zA-Z0-9]*   { yylval.var.name = (char*)malloc(sizeof(char) * yyleng + 1);
                 strncpy(yylval.var.name, yytext, yyleng);
                 variable aux;

                 // tenim en consideració si hi ha nous IDs
                 if (sym_lookup(yytext, &aux) == SYMTAB_NOT_FOUND) {
                     sym_enter(yytext, &yylval.var);
                     return ID;
                 }
                 // tenim en compte també els IDs antics
                 return (aux.type == BOOLEAN) ? B_ID : A_ID;
               }

. {return LERR;}

<<EOF>>                 { return END; }
```

3 Fitxer Bison

Les expressions i assignacions ocupen una línia, ja que cada sentència acaba amb un final de línia (*EOL*). Això significa que el *parser* esperarà un final de línia al final de cada expressió aritmètica, expressió booleana o assignació.

3.1 Definicions de tokens, tipus i estructura de la gramàtica

```
%union { variable var; };

%token <var> FL INT BOOL STR B_ID ID A_ID ADD SUB MUL DIV MOD POW BOOLOP SIN COS TAN LEN SUBSTR
SCOMMENT MCOMMENT

%token ASSIGN LPAREN RPAREN AND OR NOT EOL END LERR

%type <var> statement statement_list arithmetic_op1 arithmetic_op2 arithmetic_op3 boolean_op1
boolean_op2 boolean_op3 boolean_arithmetic exp arithmetic boolean trigonometric
```

3.2 Gramàtica principal

```
program : statement_list END;
statement_list : statement_list statement | statement;
```

"program" defineix el punt d'entrada del programa, on serà una llista de sentències (statement_list) que acaba amb el token END.

La llista de sentències pot estar seguida per més sentències o una sola.

3.3 Definició de sentències (statement)

Amb el següent tros de codi, asseguro que l'assignació sigui de la forma "$id \coloneqq expressió$", on "$\coloneqq$" és l'operador d'assignació (*ASSIGN*). "*exp*" permet que sigui una expressió aritmètica o booleana.

Utilitzant sym_enter() es pot aconseguir que no es pugui assignar un valor d'un tipus a una variable d'un altre tipus.

```
statement:
       ID ASSIGN exp {      if($3.type == UNDEFINED){
                                    yyerror($3.value.sval);
                             } else {
                                    sym_enter($1.name, &$3);
                                    fprintf(flog, "\nLínia %d: -- ASSIGNACIÓ -- (%s := %s)\n",
                                    yylineno, $1.name, valueToString($3));

                                    fprintf(flog, "\t-> NOM: %s\n\t-> TIPUS: %s\n\t-> VALOR:
                                    %s\n", $1.name, typeToString($3), valueToString($3));
                             }

                             yylineno++;

       }
```

Per poder processar expressions de diferents tipus. Si l'expressió és indefinida, es genera un error amb yyerror. Si és vàlida, es registra al fitxer de sortida amb el seu valor, tipus i identificador.

```
| exp          {      if($1.type == UNDEFINED){
                             yyerror($1.value.sval);
                      } else {
                      if ($1.name == NULL)
                              fprintf(flog, "\nLínia %d: L'expressió és nul·la. Valor:
                              '%s'\n", yylineno, valueToString($1));
                      else {
                               fprintf(flog, "\nLínia %d: -- EXPRESSION -- %s serà %s\n",
                               yylineno, $1.name, valueToString($1));

                               fprintf(flog, "\t-> TIPUS: %s\n\t-> VALOR: %s\n",
                               typeToString($1), valueToString($1));
                      }
                      }
                      yylineno++;
              }
```

Cada vegada que es trobi un final de línia (EOL), incrementarem el comptador de línies (yylineno) per mantenir el seguiment.

```
| EOL          {                    yylineno++;    }
```

Per gestionar els comentaris del codi, simples i múltiples, registro al fitxer de sortida la línia en què es troba, mostro el contingut del comentari i netejo la memòria utilitzada pel comentari. Quan s'acaba, incremento el comptador de línies esmentat.

```
| SCOMMENT     {      fprintf(flog, "Línia %d: Comentari simple '%s'\n", yylineno, $1.value.sval);
                      free($1.value.sval);
                      yylineno++;
              }

| MCOMMENT     {      fprintf(flog, "Línia %d: Comentari múltiple '%s'\n", yylineno,
                     $1.value.sval);
                      free($1.value.sval);
                      yylineno++;
              }
```

El tractament d'error durant l'anàlisi lèxic i sintàctic son; error de final de fitxer, errors lèxics (caràcters invàlids) i errors sintàctics (quan no troba cap regla que coincideixi). A tots aquests es genera un missatges concret i el registra a la línia corresponent.

```
| END          {      yyerror("Final de fitxer. Execució completada !!\n");
                      YYABORT;
              }
| LERR EOL     {      yyerror("Error lèxic: caràcter invàlid.\n");
                      yylineno++;
              }
| LERR         {      yyerror("LEXICAL ERROR: invalid character.\n");
              }
| errorEOL     {      if (errflag == 1) errflag = 0;
                      else fprintf(flog,"\nLínia %d: SYNTAX ERROR: No hi ha cap regla que
                     coicideixi.\n", yylineno);

                      yylineno++;

              }
              ;
```

3.4 Tipus d'expressions

Tal i com se'ns demana, podrem tenir dos tipus d'expressions; aritmètiques o booleanes:

```
exp: arithmetic | boolean;
```

ORDRE DE PRECEDÈNCIA

Com que no podem utilitzar *%left* o *%right* per assignar associativitat i precedència, m'he decantat per utilitzar regles de gramàtica jerarquitzades per definir l'ordre en què s'apliquen els operadors.

D'aquesta manera cada nivell representa una precedència diferent. Els operadors amb més precedència s'hauran d'executar abans dels que en tenen menys.

Per dur-ho a terme, utilitzo les regles de la gramàtica, on els operadors de més alta precedència es troben a les regles de nivell més baix, i els operadors de més alta precedències es troben a les regles de nivell més alt.

Operadors aritmètics

1. Operador de potència (**)
2. Operadors de producte (*), divisió (/) i mòdul (%)
3. Operadors suma (+) i resta (−) i Operadors Unaris (+, −)

Operadors aritmètics

1. Not (major)
2. And (mitjà)
3. Or (menor)

EXPRESSIONS ARITMÈTIQUES

Aquí defineixo els operadors unaris (+, -) aplicat sobre una expressió aritmètica o per si mateix. L'operador (+) manté el signe, mentre que l'operador (-) canvia el signe de l'operand, si és de tipus enter o real.

En cas d'una operació aritmètica s'aplica directament la funció arithmeticCalc() la qual s'encarrega d'operar l'expressió. En cas que sigui un operador únic, no caldrà fer cap operació, simplement tractar el signe.

```
/* ---- nivell més baix ---- */
arithmetic:
    arithmetic_op1
    | arithmetic ADD arithmetic_op1 { $$ = arithmeticCalc($1, $2, $3); }
    | arithmetic SUB arithmetic_op1 { $$ = arithmeticCalc($1, $2, $3); }
    | ADD arithmetic_op1 { $$ = $1; }
    | SUB arithmetic_op2 {
        $$.type = $2.type;
        if ($2.type == INTEGER)
            $$.value.ival = -$2.value.ival;
        else if ($2.type == FLOAT)
            $$.value.fval = -$2.value.fval;
    }
    ;
```

En el nivell intermedi s'inclou: multiplicació, divisió i mòdul. Bàsicament es crida la funció arithmeticCalc() per realitzar el càlcul.

```
/* ---- nivell intermedi: operadors de mul, div i mod ---- */

arithmetic_op1:

               arithmetic_op2

               | arithmetic_op1 MUL arithmetic_op2        { $$ = arithmeticCalc($1, $2, $3); }

               | arithmetic_op1 DIV arithmetic_op2        { $$ = arithmeticCalc($1, $2, $3); }

               | arithmetic_op1 MOD arithmetic_op2 { $$ = arithmeticCalc($1, $2, $3); }

               ;
```

Al nivell més alt, es tracta l'operador de potència. El procediment és igual que el nivell anterior.

```
/* ---- nivell més alt: operador pow ---- */

arithmetic_op2:

               arithmetic_op3

               | arithmetic_op2 POW arithmetic_op3 { $$ = arithmeticCalc($1, $2, $3); }

               ;
```

Finalment, gestiono el nivell base, el que té més prioritat: gestió de parèntesis, constant literals, LEN, SUBSTR, variables i funcions trigonomètriques.

```
trigonometric: SIN | COS | TAN;
```

```
/*  ---- nivell base  ----*/

arithmetic_op3:
               LPAREN arithmetic RPAREN            { $$ = $2; }
               | INT                               { if($1.type == UNDEFINED)
                                                         yyerror($1.value.sval);
                                                   else $$ = $1;
                                                   }
               | FL                                { if($1.type == UNDEFINED)
                                                         yyerror($1.value.sval);
                                                   else $$ = $1;
                                                   }
               | STR                               { if($1.type == UNDEFINED)
                                                         yyerror($1.value.sval);
                                                   else $$ = $1;
                                                   }
               | LEN LPAREN arithmetic RPAREN      { $$ = lenCalc($3); }

               | SUBSTR LPAREN arithmetic ';' arithmetic ';' arithmetic RPAREN {
                                                   $$ = substrCalc($3, $5, $7); }

               | A_ID        { if(sym_lookup($1.name, &$1) == SYMTAB_NOT_FOUND) {
                                       yyerror("SEMANTIC ERROR: VARIABLE NOT FOUND.\n");
                                       errflag = 1;
                                       YYERROR;
                             } else {
                                       $$.type = $1.type;
                                       $$.value=$1.value;
                             }
                             }



               |ID           {      if(sym_lookup($1.name, &$1) == SYMTAB_NOT_FOUND) {
                                          yyerror("SEMANTIC ERROR: VARIABLE NOT FOUND.\n");
                                          errflag = 1;
```

```
                                        YYERROR;
                                } else {
                                        $$.type = $1.type;
                                        $$.value=$1.value;
                                }
                        }
                | trigonometric LPAREN arithmetic RPAREN          {$$ = trigonometricCalc($1, $3);}
                ;
```

Els INT i FL són *tokens* que representen literals enters i literals reals. Quan el Lexer troba un literal d'aquest tipus i no han estat guardats prèviament, doncs els reconeix, i els guarda de tal manera que els pugui incloure a futures expressions aritmètiques.

Les operacions entre expressions del mateix tipus, el resultat és del mateix tipus que els operands.

Les operacions entre enters i reals, el codi tracta l'operació aritmètica de manera que el tipus resultant sigui real (*FLOAT*). Això es fa amb la funció arithmeticCacl(), que defineix que el resultat d'una operació entre enter i real, es converteixi en real.

El token A_ID, verifica que tots els identificadors han estat prèviament inicialitzats abans de ser utilitzats en expressions. Això ho faig amb la crida de la funció sym_lookup() en la qual es comprova que l'identificador existeix a la taula de símbols, generant un error semàntic en cas que l'identificador no estigui inicialitzat. Això assegura que el tipus de cada identificador es conegui abans de ser utilitzar en una expressió.

En cas que es trobi, assigno el tipus i valor de la variable a $$, de manera que després es pugui utilitzar en expressions aritmètiques.

D'aquesta manera permet utilitzar les funcions trigonomètriques (sin, cos i tan) amb una expressió aritmètica entre parèntesis.
Utilitzo la funció trigonomtricCalc() per realitzar el càlcul.

EXPRESSIONS BOOLEANES

```
 /* OPERADOR OR */
boolean: boolean_op1

                | boolean OR boolean_op1     {
                                        $$.name = NULL;
                                        $$.type = BOOLEAN;
                                        $$.value.bval = $1.value.bval || $3.value.bval;
                                        };
```

```
 /* OPERADORS AND */
boolean_op1: boolean_op2

                | boolean_op1 AND boolean_op2 {
                                        $$.name = NULL;
                                        $$.type = BOOLEAN;
                                        $$.value.bval = $1.value.bval && $3.value.bval;
                                        };
```

```
 /* OPERADORS NOT */
boolean_op2: boolean_op3

                | NOT boolean_op2           {
                                        $$.name = NULL;
                                        $$.type = BOOLEAN;
```

```
                                        $$.value.bval = !($2.value.bval);
                                        };
```

```
 /* PARÈNTESIS, BOOLEÀ, IDENTIFICADOR */
boolean_op3: boolean_arithmetic

              | LPAREN boolean RPAREN     {      $$ = $2;       }

              | BOOL                     {      $$ = $1;       }

              | B_ID                     {

                                        if(sym_lookup($1.name, &$1) == SYMTAB_NOT_FOUND) {

                                               yyerror("SEMANTIC ERROR: VARIABLE NOT FOUND\n");
                                               errflag = 1;
                                               YYERROR;

                                        } else {
                                               $$.type = $1.type;
                                               $$.value=$1.value;
                                               }
                                        };

boolean_arithmetic: arithmetic BOOLOP arithmetic  {booleanCalc($1, $2, $3);}
```

Avaluació d'expressions booleanes sense curtcircuit

El codi s'implementa sense curtcircuit. Per tant, significa que dins d'una expressió booleana, tots els operants s'avaluen completament independentment de si el resultat es pot deduir abans de completar l'expressió.

S'ha d'assegurar que tots els operands d'una expressió booleana es processen abans de determinar el resultat final. En el codi, els operadors AND i OR es fan sempre entre dues expressions booleanes, sense cap condicional per interrompre en cas que es conegui el resultat.

4 Autoavaluació

Tinc tres errors que no he pogut resoldre per falta de temps.

El primer fa referència al tractament amb booleans. Les operacions amb booleans funcionen correctament, però el problema és que, en intentar concatenar una cadena amb altres tipus de dades, per exemple, $enter + booleà + real$, no aconsegueixo que es realitzi la concatenació.

El segon error està relacionat amb el tractament de reals. Funciona correctament, com la resta del programa, però no he aconseguit que, en utilitzar-lo dins d'una expressió per obtenir un resultat booleà, funcioni com esperava.

Per últim, el tercer error es tracta de la funció SUBSTR. No em funciona.