Podstawy Kompilatorów

Laboratorium 12

Analiza zależności kontekstowych.

Zadanie 1:
Proszę napisać analizator zgodności typów dla podzbioru standardowych wyrażeń języka Pascal. Dla uproszczenia należy założyć, że w podzbiorze występują zmienne trzech typów: *integer*, *real* i *boolean*. W języku dostępny jest zestaw operatorów, jednak nie każdego operatora można użyć do każdych argumentów:

operator(y)	typy
+ (dodawanie), - (odejmowanie), * (mnożenie)	integer × integer → integer real × integer → real integer × real → real real × real → real
/ (dzielenie)	integer × integer → real real × integer → real integer × real → real real × real → real
div (dzielenie całkowite), *mod* (reszta z dzielenia całkowitego)	integer × integer → integer
and (iloczyn logiczny), *or* (suma logiczna)	boolean × boolean → boolean
>, >=, =, <>, <, <= (porównania)	integer × integer → boolean real × integer → boolean integer × real → boolean real × real → boolean boolean × boolean → boolean

w pozostałych przypadkach występuje błąd typu.

Operatory mają następujące priorytety:

operator(y)	priotytet
- (unarny minus)	1 (najwyższy)
*, /, *div*, *mod*, *and*	2
+, -, *or*	3
>, >=, =, <>, <, <=	4 (najniższy)

Wszystkie operatory mają lewostronną łączność. Oprócz operatorów w pliku wejściowym mogą pojawić się:

- stałe typu *real* (*[0-9]+"."[0-9]**)
- stałe typu *integer* (*[0-9]+*)
- stałe typu *boolean* (*true*, *false*)

Analizator, po zbadaniu wyrażenia, ma za zadanie wyświetlić końcowy typ wyrażenia.

Konstruując analizator składniowy nie wolno posługiwać się zmiennymi globalnymi

Przykłady:

Dla pliku o postaci: **`1+1.0`**
powinniśmy otrzymać wynik: **`real`**

Dla pliku o postaci: **`1+2`**
powinniśmy otrzymać wynik: **`integer`**

Dla pliku o postaci: **`(1<3)and(4.1>2)or false`**
powinniśmy otrzymać wynik: **`boolean`**

Dla pliku o postaci: **`(true and false) + 1`**
powinniśmy otrzymać wynik: **`type error`**

Analizator leksykalny ma następującą postać:

```
%{
  int yywrap(void);
  int yylex(void);
  #include <stdio.h>
  #include "y.tab.h"
%}
%%
[tT][rR][uU][eE]        { return TYPE_BOOLEAN; }
[fF][aA][lL][sS][eE]    { return TYPE_BOOLEAN; }
[dD][iI][vV]            { return OP_DIV; }
[mM][oO][dD]            { return OP_MOD; }
[aA][nN][dD]            { return OP_AND; }
[oO][rR]                { return OP_OR; }
[0-9]+                  { return TYPE_INTEGER; }
[0-9]+"."[0-9]*         { return TYPE_REAL;    }
\+                      { return('+'); }
\*                      { return('*'); }
\-                      { return('-'); }
\/                      { return('/'); }
\(                      { return('('); }
\)                      { return(')'); }
">"                     { return OP_GT; }
">="                    { return OP_GE; }
"="                     { return OP_EQ; }
"<"                     { return OP_LT; }
"<="                    { return OP_LE; }
"<>"                    { return OP_NE; }
" "|\t|\n               { ; }
%%
int yywrap(void) { return 1;}
```

Odpowiedzi do zadań

Zadanie 1:

```
%{
  int yylex(void);
  void yyerror(const char *,...);
  int yyparse(void);
  #include <stdio.h>
  extern int yylineno;

  char *type2str(int);
  int compute_type(int,int,int);
%}

%token TYPE_INTEGER TYPE_REAL TYPE_BOOLEAN TYPE_ERROR

%token OP_GT OP_GE OP_EQ OP_LT OP_LE OP_NE
%left OP_GT OP_GE OP_EQ OP_LT OP_LE OP_NE

%token OP_OR
%left '+' '-' OP_OR

%token OP_AND
%left '*' '/' OP_AND OP_DIV OP_MOD

%left UMINUS

%%
P : E     { printf("%s",type2str($1)); }
  ;
E : E '+' E                { $$ = compute_type('+',$1,$3); }
  | E '-' E                { $$ = compute_type('+',$1,$3);  }
  | E OP_OR E              { $$ = compute_type(OP_OR,$1,$3);  }
  |   '-' E  %prec UMINUS  { $$ = compute_type('-',$2,$2); }
  | E '*' E                { $$ = compute_type('*',$1,$3);  }
  | E '/' E                { $$ = compute_type('/',$1,$3);  }
  | E OP_AND E             { $$ = compute_type(OP_AND,$1,$3);  }
  | '(' E ')'              { $$ = $2; }
  | E OP_GT E              { $$ = compute_type(OP_GT,$1,$3);  }
  | E OP_GE E              { $$ = compute_type(OP_GE,$1,$3);  }
  | E OP_EQ E              { $$ = compute_type(OP_EQ,$1,$3);  }
  | E OP_LT E              { $$ = compute_type(OP_LT,$1,$3);  }
  | E OP_LE E              { $$ = compute_type(OP_LE,$1,$3);  }
  | E OP_NE E              { $$ = compute_type(OP_NE,$1,$3);  }
  | TYPE_INTEGER           { $$ = TYPE_INTEGER; }
  | TYPE_REAL              { $$ = TYPE_REAL; }
  | TYPE_BOOLEAN           { $$ = TYPE_BOOLEAN; }
  ;
%%
void yyerror(const char *fmt,...) {
   printf("%s in line %d\n", fmt, yylineno); }
int main() { return yyparse(); }

char *type2str(int type)
```

```
{
  static char *typenames[] = { "integer", "real", "boolean",
"type error" };
  switch(type)
  {
    case TYPE_INTEGER : return typenames[0];
    case TYPE_REAL    : return typenames[1];
    case TYPE_BOOLEAN : return typenames[2];
    default           : return typenames[3];
  }
}
int compute_type(int op, int left, int right)
{
  switch(op)
  {
    case '+'   :
    case '-'   :
    case '*'   : if((left == TYPE_INTEGER) &&
                    (right == TYPE_INTEGER))return TYPE_INTEGER;
                 if((left == TYPE_INTEGER) &&
                    (right == TYPE_REAL))return TYPE_REAL;
                 if((left == TYPE_REAL) &&
                    (right == TYPE_INTEGER))return TYPE_REAL;
                 if((left == TYPE_REAL) &&
                    (right == TYPE_REAL))return TYPE_REAL;
                 return TYPE_ERROR;
    case '/'   : if((left == TYPE_INTEGER) &&
                    (right == TYPE_INTEGER))return TYPE_REAL;
                 if((left == TYPE_INTEGER) &&
                    (right == TYPE_REAL))return TYPE_REAL;
                 if((left == TYPE_REAL) &&
                    (right == TYPE_INTEGER))return TYPE_REAL;
                 if((left == TYPE_REAL) &&
                    (right == TYPE_REAL))return TYPE_REAL;
                 return TYPE_ERROR;
    case OP_GT :
    case OP_GE :
    case OP_EQ :
    case OP_LT :
    case OP_LE :
    case OP_NE : if((left == TYPE_INTEGER) &&
                    (right == TYPE_INTEGER))return TYPE_BOOLEAN;
                 if((left == TYPE_INTEGER) &&
                    (right == TYPE_REAL))return TYPE_BOOLEAN;
                 if((left == TYPE_REAL) &&
                    (right == TYPE_INTEGER))return TYPE_BOOLEAN;
                 if((left == TYPE_REAL) &&
                    (right == TYPE_REAL))return TYPE_BOOLEAN;
                 if((left == TYPE_BOOLEAN) &&
                    (right == TYPE_BOOLEAN))return TYPE_BOOLEAN;
                 return TYPE_ERROR;
    case OP_OR :
    case OP_AND: if((left == TYPE_BOOLEAN) &&
                    (right == TYPE_BOOLEAN))return TYPE_BOOLEAN;
                 return TYPE_ERROR;
    case OP_DIV:
```

```
    case OP_MOD: if((left == TYPE_INTEGER) &&
                    (right == TYPE_INTEGER))return TYPE_INTEGER;
                 return TYPE_ERROR;
  }
  return TYPE_ERROR; /* ? */
}
```