1 Actividade 0x06 - Compilador de *CaLC*

Voltando ao programa da calculadora, vamos pensar agora em termos de compilação ao invés de interpretação: vamos construir um compilador para a linguagem "*CaLC*", que consiste em expressões aritméticas que podem conter variáveis e variáveis. Como não faz muito sentido gerar código para fazer operações e afectações, faremos um compilador que gera um diagrama da *APT* a partir de uma sequência de operações.

2 Compilador de diagramas de APT para a linguagem *CaLC*

Para representar a APT, são necessárias algumas estruturas:

2.1 calctypes.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "calctypes.h"

struct calc_t_exp_ {
  enum {EXP_NUM, EXP_ID, EXP_BINOP, EXP_UNOP, EXP_ASSIGN} kind;

  union {
    int num;
    char *id;
    /* ..................... */
  } u;
};

struct calc_t_seq_ {
  enum {SEQ_EMPTY, SEQ_EXP} kind;

  union {
    struct {
      calc_t_exp e;
      calc_t_seq s;
    } exp;
  } u;
};
```

2.2 calctypes.h

```
#ifndef CALC_TYPES_H_
#define CALC_TYPES_H_

typedef struct calc_t_exp_ *calc_t_exp;
typedef struct calc_t_seq_ *calc_t_seq;

calc_t_exp calc_exp_new_num(int num);
calc_t_exp calc_exp_new_id(char *id);
calc_t_exp calc_exp_new_binop(char op[], calc_t_exp arg1,
    calc_t_exp arg2);
calc_t_exp calc_exp_new_unop(char op[], calc_t_exp arg);
calc_t_exp calc_exp_new_assign(char *id, calc_t_exp rvalue);

calc_t_seq calc_seq_new_empty();
calc_t_seq calc_seq_new_exp(calc_t_exp e, calc_t_seq s);

#endif
```

Temos também de mudar as acções semânticas no parser, de forma a criar os nós da APT ao invés de interpretar as operações e calcular os resultados. Agora não queremos fazer as contas, mas sim representá-las na APT.

2.3 calc.y

```
%{
#include <stdio.h>
#include "calctypes.h"
#include "latex.h"  /* print_prologue(), etc. */

int yylex (void);
void yyerror (char const *);
%}

%union {
    double val;
    char *name;
    calc_t_exp exp;
    calc_t_seq seq;
}

%token   <val>      NUM
```

```
%token   <name>     ID

%right              ASSIGN

%left               SUB ADD
%left               MUL DIV
%left               NEG     /* negation--unary minus */
%token              LPAR RPAR
%type    <exp>      exp
%type    <seq>      seq

%%
input:   seq        { print_prologue();
                      calc_seq_print($1);
                      print_epilogue(); }
;

seq:       /* empty */  { $$ = calc_seq_new_empty(); }
        |  exp NL seq   { $$ = calc_seq_new_exp($1, $3); }
;

exp:         NUM                { $$ = calc_exp_new_num($1); }
        |    ID                 { $$ = calc_exp_new_id($1); }
        |    exp ADD exp        { $$ =  }
        |    exp SUB exp        { $$ =  }
        |    exp MUL exp        { $$ =  }
        |    exp DIV exp        { $$ =  }
        |    SUB exp  %prec NEG { $$ =  }
        |    LPAR exp RPAR      { $$ = $2; }
        |    ID ASSIGN exp      { $$ =  }
;
%%
void yyerror (char const *s)
{
  fprintf (stderr, "%s\n", s);
}

int main (void)
{
    return yyparse();
}

```

E as funções que geram e "imprimem" os nós da APT:

2.4 `calctypes.c` (continuação)

```
1  calc_t_exp calc_exp_new_num(int num)
2  {
3    calc_t_exp ret = (calc_t_exp) malloc (sizeof (*ret));
4
5    ret->kind = EXP_NUM;
6    ret->u.num = num;
7
8    return ret;
9  }
10
11 calc_t_exp calc_exp_new_id(char *id)
12 {
13   calc_t_exp ret = (calc_t_exp) malloc (sizeof (*ret));
14
15   ret->kind = EXP_ID;
16   ret->u.id = id;
17
18   return ret;
19 }
20
21 void calc_exp_print(calc_t_exp exp)
22 {
23   switch (exp->kind) {
24   case EXP_NUM:
25     printf("[.exp [.num $%d$ ] ]\n", exp->u.num);
26     break;
27   case EXP_ID:
28     printf("[.exp [.id $%s$ ] ]\n", exp->u.id);
29     break;
30
31     /* .......... */
32
33   }
34 }
35
36 calc_t_seq calc_seq_new_empty()
37 {
38   calc_t_seq ret = (calc_t_seq) malloc (sizeof (*ret));
39
40   ret->kind = SEQ_EMPTY;
```

```
41
42   return ret;
43 }
44
45 calc_t_seq calc_seq_new_exp(calc_t_exp e, calc_t_seq s)
46 {
47   calc_t_seq ret = (calc_t_seq) malloc (sizeof (*ret));
48
49   ret->kind = SEQ_EXP;
50   ret->u.exp.e = e;
51   ret->u.exp.s = s;
52
53   return ret;
54 }
55
56 void calc_seq_print(calc_t_seq s)
57 {
58   switch (s->kind) {
59   case SEQ_EMPTY:
60     printf("[.\\emph{empty} ]\n");
61     break;
62   default:
63     printf("[.seq \n");
64     calc_exp_print(s->u.exp.e);
65     calc_seq_print(s->u.exp.s);
66     printf("] \n");
67   }
68 }
```

Para gerar as árvores, usamos a biblioteca `tikz` do LaTeX.

2.5 `exemplo.tex`

```
\documentclass{standalone}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}

\usepackage[margin=1in]{geometry}
\usepackage{tikz-qtree}
\usetikzlibrary{shadows,trees}
\begin{document}
\tikzset{font=\small,
level distance=.8cm,
every node/.style=
    {color=white,
    rectangle,rounded corners,
    align=center,
    text = black
    },
edge from parent/.style=
    {draw=blue,
    thick
    }}

\centering
\begin{tikzpicture}
\Tree [.exp [.assign [.id $a$ ] [.exp [.num $1$ ] ] ] ]
\end{tikzpicture}
\end{document}
```

3 Exercício

1. Analise o código apresentado e tente explicar cada uma das secções.
2. Complete as partes em falta, de forma a gerar diagramas de APT para a linguagem *CaLC*.