1 Actividade 0x0a - Análise Semântica

O próximo passo na construção do nosso compilador de *Ya*! é implementar as funções que executam a análise de nomes (IDs) e tipos. Para isso, temos de ter o nosso *parser* a produzir adequadamente a APT.

A análise semântica é feita através de um percurso em ordem da APT, realizando inserções na Symbol Table (nas declarações) e *lookups* (nos usos), verificando ainda se os tipos dos nós são "compatíveis" com o local onde se situam (e.g., se uma variável está a ser afectada com um valor do tipo correcto, se uma função está a ser chamada com o número e tipos de argumentos correcto, etc.).

2 Revisão e Contextualização

Para podermos executar a análise semântica, precisamos de ter:

- Todas as "classes" para os nós da APT, com os respectivos "construtores";
- Acções semânticas para que as produções da gramática possam gerar os nós da APT;
- A Symbol Table;
- Funções para a análise de nomes e tipos;

2.1 A "union" do bison

Para que as regras da gramática do **bison** possam gerar nós da APT, temos de permitir que o **bison** tenha conhecimento dos nossos tipos, de forma a que estes possam ser usados como tipo de retorno das regras (no símbolo `$$`).

```
%union {
    int         ival;
    double      dval;
    char*       str;
    t_Exp       TExp;
    t_Lit       TLit;
    t_Args      TArgs;
    t_Decl      TDecl;
    t_Decls     TDecls;
    t_IDs       TIDs;
    t_Type      TType;
    t_Argdefs   TArgdefs;
    t_Argdef    TArgdef;
    t_Stms      TStms;
    t_Stm       TStm;
}
```

E declarar os tipos das regras:

```
%type   <TExp>          exp
%type   <TLit>          lit
%type   <TArgs>         args
%type   <TDecls>        decls
%type   <TDecl>         decl
%type   <TIDs>          ids
%type   <TStm>          stm
%type   <TStms>         stms
%type   <TArgdefs>      argdefs
%type   <TType>         type
%type   <TArgdef>       argdef
```

2.2 Gerar a APT

Chamar os construtores correspondentes em cada produção:

```
program:  /* empty */   { $$ = NULL; }
    |     decls         { $$ = $1; }
    ;

decls:    decl          { $$ = t_decls_new($1, NULL); }
    |     decl decls    { $$ = t_decls_new($1, $2); }
    ;

decl:   ids COLON type SEMI   { $$ = t_decl_new_decl($1, $3, NULL
    ); }
    |   ids COLON type ASSIGN exp SEMI { $$ = t_decl_new_decl($1,
    $3, $5); }
    . . .

stm:. . .
    |   RETURN exp SEMI  { $$ = t_stm_new_return($2); }
    . . .

exp:. . .
    |   ID          { $$ = t_exp_new_id($1); }
    . . .
    |   exp ADD exp { $$ = t_exp_new_binop('+', $1, $3); }
    . . .
```

Terminado este passo, quando corremos o nosso "compilador" sobre algum código-fonte em Ya!, obtemos a nossa APT no `$$` da regra `program`. Agora podemos realizar as nossas análises e restantes passos do compilador sobre esta "variável":

```
program:  /* empty */   { $$ = NULL; }
    |     decls         { $$ = $1;
                          semantic_analysis($$);
                          other_analisys($$);
                          ...($$);
    }
```

3 Funções para Análise Semântica

Para efectuar a Análise Semântica, precisamos de uma função para cada tipo de nó[1]:

```
void t_decls_ant(t_decls ds)
/* aka semantic_analysis(ds) */
{
  switch(ds->kind) {
  case DECLS_SINGLE:
    t_decl_ant(ds->decl);
    break:
  case DECLS_LIST:
    t_decl_ant(ds->decl);
    t_decls_ant(ds->decls);
    break;
  default:
    ERROR();
    break;
  }
}

void t_decl_ant(t_decl d)
{
  switch(d->kind) {
  case D_VAR:
    __for_each__(d->u.ids) {
      INSERT(d->u.ids[x], d->u.type);
    }
    break;
  case D_VAR_INIT:
```

[1] À semelhança das funções `*_print` do exercício das APTs em LaTeX.

```
27      /* do the same as above AND */
28      check_types(d->u.typ, d->u.exp);
29      break;
30    }
31  }
32
33  _Type t_exp_ant(t_exp e)
34  {
35    switch(e->kind) {
36    case E_ASSIGN:
37      lval_type = LOOKUP(e->u.assign.id);
38      check_types(lval_type, e->u.assign.exp);
39      break;
40      /* . . . */
41    case E_BINOP:
42      t1 = t_exp_ant(e->u.binop.e1);
43      t2 = t_exp_ant(e->u.binop.e2);
44      check_types(t1,t2);
45      break;
46      /* etc */
47    }
48  }
```

4 Exercício

1. Implementar as funções para a análise de nomes e tipos de cada tipo de nó da APT. As funções devem gerar uma lista de erros e apenas fazer o *output* no final da análise semântica.

2. Chamar a função na regra inicial, executando a análise semântica de todo o programa.