1 Actividade 0x09 - APT para *Ya*!

Finalizado o *parser* da actividade 0x07, queremos agora gerar Árvores Abstractas de Pesquisa (APTs) para programas em *Ya*!. Para isso, precisamos de 3 coisas:

1. "classes" para os vários tipos de nós a produzir;
2. "construtores" para as referidas classes;
3. acções semânticas para cada produção da nossa gramática.

2 "Classes"

Para cada regra da nossa gramática, precisamos de uma "classe" (ou `struct` em C) que represente de forma eficaz os nós da APT gerados por essa mesma regra.

```
struct t_exp_ {
  enum {EXP_INTLIT, EXP_FLOATLIT, EXP_ID, EXP_BINOP, /* . . . */}
    kind;

  union {
    int intlit;
    float floatlit;
    char *id;
    /* ..................... */
  } u;
};

/* ................ */

struct t_decls_ {
  enum {DECLS_SINGLE, DECLS_LIST} kind;

  struct {
    t_decl decl;
    t_decls decls;
  } u;
};

/* ................ */

struct t_stm_ {
  enum {STM_DECL, STM_EXP, STM_RETURN, /* . . . */} kind;

  union {
    t_decl decl;
```

```
30      t_exp  exp;
31      t_exp  return;
32      /* ................ */
33    } u;
34 };
35
36 /* ................ */
```

3 "Construtores"

Para cada "classe" definida anteriormente, temos de definir "construtores" apropriados (mais ou menos um por produção[1], que vão ser usados nas acções semânticas para produzir os nós da APT.

```
1  t_exp t_exp_new_id(char *id)
2  {
3    t_exp ret = (t_exp) malloc (sizeof (*ret));
4
5    ret->kind = EXP_ID;
6    ret->u.id = id;
7
8    return ret;
9  }
10
11 /* ................ */
12
13 t_decls t_decls_new(t_decl decls, t_decls decls)
14 {
15   t_decls ret  = (t_decls) malloc (sizeof (*ret));
16
17   if (decls) {
18     ret->kind  = DECLS_LIST;
19   }
20   else {
21     ret->kind  = DECLS_SINGLE;
22   }
23   ret->u.decl  = decl;
24   ret->u.decls = decls;
25
26   return ret;
27 }
28
```

[1] embora seja útil aproveitar, em alguns casos, o mesmo "construtor" para várias produções (e.g., operações binárias, etc.)

```
/* ............... */

t_stm t_stm_new_return(t_exp exp)
{
  t_stm ret = (t_stm) malloc (sizeof (*ret));

  ret->kind = STM_RETURN;
  ret->u.return = exp;

  return ret;
}

/* ............... */
```

4 Acções semânticas para produções

Para cada produção, temos agora de chamar o construtor correspondente, de forma a gerar os nós da APT. Na listagem seguinte podemos ver exemplos de algumas acções semânticas:

```
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); }
    . . .

```

5 Exercício

1. Implementar as estruturas necessárias para os nós da APT para programas em *Ya*!.
2. Completar o *parser* da actividade 0x07 com as acções semânticas, de forma a obter uma APT no final da análise sintáctica.
3. Pensar de que forma poderá ser integrada a estrutura da actividade 0x08 no nosso programa.