PROJET DE COMPILATION - CONSTRUCTION DE L'ARBRE ABSTRAIT

1. Objectif

L'objectif de ce TP est de construire un arbre abstrait correspondant au programme en langage source. Cet arbre abstrait sera utilisé lors des étapes suivantes de la compilation, en particulier lors de la production de code.

2. Arbre abstrait

L'arbre de dérivation produit par l'analyse syntaxique possède de nombreux nœuds superflus, qui ne véhiculent pas d'information.
De plus, la mise au point d'une grammaire (élimination de l'ambiguïté, élimination de la récursivité à gauche, factorisation) nécessite souvent l'introduction de règles dont le seul but est de simplifier l'analyse syntaxique.
Un arbre abstrait constitue une interface plus naturelle entre l'analyse syntaxique et l'analyse sémantique, il ne garde de la structure syntaxique que les parties nécessaires à l'analyse sémantique et à la production de code.
On trouvera dans la figure 1 un arbre de dérivation et l'arbre abstrait lui correspondant.

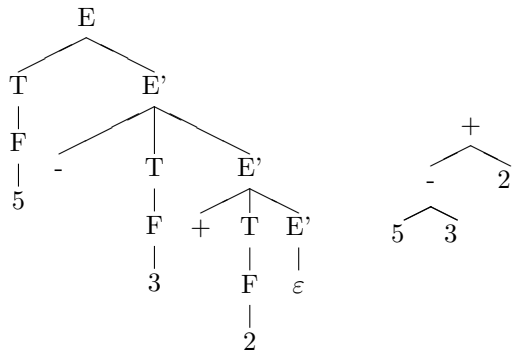

Figure 1. Un arbre de dérivation et l'arbre abstrait correspondant

3. Construction de l'arbre abstrait

L'arbre abstrait est construit lors de l'analyse syntaxique, en ajoutant, à toute fonction correspondant à un non terminal de votre grammaire, du code destiné à la construction de l'arbre abstrait.
L'idée est de construire, pour certains noœuds de l'arbre de dérivation, un nœud de l'arbre abstrait (appelé nœud courant) et de le retourner sous forme d'attribut synthétisé. Étant donné la règle $A \rightarrow BC$, les fils du nœud courant, construit par la fonction `A()`, sont les nœuds renvoyés par les appels aux fonctions `B()` et `C()`.
L'idée est illustrée ci-dessous sur la fonction correspondant à la règle de grammaire suivante :

```
n_instr *instructionSi(void) {
  n_instr *$$ = NULL;         n_exp *$2 = NULL;
  n_instr *$4 = NULL;         n_instr *$5 = NULL;
  affiche_balise_ouvrante(__FUNCTION__, trace_xml);
  if(uc == SI){
    consommer( SI );
    $2 = expression();
    consommer( ALORS );
    $4 = instructionBloc();
    $5 = optSinon();
    $$ = cree_n_instr_si($2, $4, $5);
    affiche_balise_fermante(__FUNCTION__, trace_xml);
    return $$;
  }
  erreur("");
}
```

Les variables locales `$2`, `$4` et `$5` vont accueillir les nœuds renvoyés par les appels aux fonctions `expression()`, `instructionBloc` et `optSinon()`. Ces nœuds vont constituer les fils du nœud courant, appelé `$$`. Cette opération est effectuée par l'appel `cree_n_instr_si($2, $4, $5)`. Le nœud ainsi créé (stocké dans la variable `$$`) constitue la valeur de retour de la fonction `instructionSi()`.

4. Associativité gauche des expressions

Les nœuds de l'arbre abstrait renvoyés par les fonctions correspondent à des attributs synthétisés. Ils doivent donc être construits à la fin de la fonction. Cependant, pour implémenter un arbre abstrait associatif à gauche pour les expressions, vous aurez besoin d'attributs hérités, qui prennent la forme de paramètres passés aux fonctions de type `bis`. Vous pouvez implémenter la grammaire attribuée ci-dessous, généralisée aux autres opérateurs :

	règle			action sémantique	
1	E	$\rightarrow$	T E'	$E'.h = T.s$	$E.s = E'.s$
2	E'	$\rightarrow$	+ T E'$_1$	$E'_1.h = E'.h + T.s$	$E'.s = E'_1.s$
3	E'	$\rightarrow$	- T E'$_1$	$E'_1.h = E'.h - T.s$	$E'.s = E'_1.s$
4	E'	$\rightarrow$	ε		$E'.s = E'.h$
5	T	$\rightarrow$	F T'	$T'.h = F.s$	$T.s = T'.s$
6	T'	$\rightarrow$	* F T'$_1$	$T'_1.h = T'.h \times F.s$	$T'.s = T'_1.s$
7	T'	$\rightarrow$	/ F T'$_1$	$T'_1.h = T'.h \div F.s$	$T'.s = T'_1.s$
8	T'	$\rightarrow$	ε		$T'.s = T'.h$
9	F	$\rightarrow$	(E)		$F.s = E.s$
10	F	$\rightarrow$	n		$F.s = n$

N'oubliez pas que les attributs synthétisés sont évalués à la fin de la dérivation/appel de la règle/fonction, alors que les attributs hérités sont évalués juste avant la dérivation/appel au non-terminal/fonction qui contient cet attribut. Voici un exemple pratique de fonction contenant des attributs synthétisés et hérités à la fois :

```
n_exp *termeBis( n_exp *herite ) {
  n_exp *$2;         n_exp *$$;        n_exp *herite_fils;
  if(uc == FOIS){
    consommer(FOIS);
    $2 = facteur(); // F n'a pas d'attribut hérité
    // Ligne 6 : évalue T1'.h avant de dériver T1'
    herite_fils = cree_n_exp_op(fois, herite, $2);
    // Ligne 6 : attribut synthetisé T'.s = attribut hérité T1'.s
    $$ = termeBis( herite_fils );
    return $$;
  }
  if(est_suivant(uc, _termeBis_)){
    /* Règle 8 : attribut synthetisé T'.s = attribut hérité T'.h */
    $$ = herite;
    return $$;
  }
  erreur( "Expression mal formée" );
}
```

5. Fichiers fournis

Les différents types de nœuds pouvant être renvoyés par les fonctions de votre analyseur ainsi que leurs constructeurs ont été définis dans les fichiers `syntabs.h` et `syntabs.c` disponibles sur le site du cours.
Vous pourrez les utiliser tels quels ou vous en inspirer pour écrire vos propres types et constructeurs. Si vous utilisez les structures fournies pour l'arbre abstrait, vous pourrez également utiliser la fonction `affiche_n_prog`, dans `affiche_arbre_abstrait.c`, pour afficher un arbre XML et vérifier votre résultat en le comparant aux fichiers d'exemple `.asynt` disponibles sur le site du cours. Cette fonction doit être appelée à la fin de l'analyse syntaxique, sur le nœud de type `n_prog *` renvoyé par la fonction `programme`, axiome de la grammaire.