

Travaux Dirigés Compilation: TP2

Informatique 2ème année. ENSEIRB-MATMECA 2014/2015

► **Exercice 1.** *Construction d'un arbre syntaxique abstrait* On souhaite construire un arbre syntaxique abstrait pour les expressions. On ignorera les autres instructions ainsi que les déclarations. Le principe est de construire cet arbre dans les actions sémantiques des règles pour les expressions et affectations. On pourra prendre un arbre binaire, dont la racine définit l'opération effectuée. Par exemple :

```
typedef struct node_s {
    char kind; // kind of operation, or leaf
    struct node_s *left, *right; // NULL if leaf
    union {
        int i;
        float f;
        char *id;
    } leaf;
} node_t;
```

Les feuilles de cet arbre seront des constantes ou des identificateurs.

1. Déclarer ce type dans l'analyseur syntaxique et l'utiliser comme type pour les attributs de `primary_expression`, `argument_expression_list`, `unary_expression`, `multiplicative_expression`, `additive_expression`, `comparison_expression` et `expression`.
2. Faire une fonction C qui étant données 2 valeurs de type `node`, crée un noeud père dont le type (`kind`) est passé en paramètre. Faire une fonction qui crée une feuille d'un certain type (entier, flottant ou identifiant).
3. Faire une fonction qui affiche l'arbre d'une expression (en C par exemple).
4. Utiliser ces fonctions pour créer l'arbre syntaxique des expressions et l'afficher à chaque noeud `expression`.

► **Exercice 2.** *Threads* Le but de cet exercice est la manipulation des fonctions de création et d'attente des threads. On pourra faire `man pthread(3)` pour avoir une liste exhaustive des fonctions utilisables pour les threads. La création de threads se fait avec la fonction `pthread_create(3)` qui prend notamment en argument un pointeur sur la fonction qu'exécutera le thread.

1. Ecrire un programme qui lance 8 nouveaux threads (en plus de celui du programme principal) sur une fonction `void *start(void *)` qu'on écrira. Cette fonction affichera un numéro (entre 0 et 7) passé à la fonction `start` par le dernier argument de `pthread_create`. On compilera avec l'argument `-lpthread`.
2. La fonction `pthread_join(3)` permet d'attendre un thread identifié par son `THREAD ID` (défini lors de l'appel de `pthread_create`). Modifiez le programme précédent pour que le thread principal attende tous les threads qu'il a créés puis affiche un message sur la sortie standard.