TP Analyse syntaxique

Yohann D'ANELLO

Analyse lexicale

Le fichier `example.l` peut se compiler en un analyseur lexical `example` en appelant simplement la commande `make example`. Il permet à partir d'une entrée textuelle (un fichier, le TTY ou l'entrée standard suivie d'un Ctrl+D) de filter les nombres et de les afficher, sous forme décimale ou hexadécimale. De plus, les mots clés `if`, `then` et `else` sont réécrits en majsucule. Tout le reste est ignoré.

Un entier `n` affichera sur la sortie `int(n)`, un entier `0xh` écrit en hexadécimal affichera dans la sortie `hex(n)` où `n` est la représentation décimale de `0xh`.

Par exemple sur une entrée :

```
Papa maman ! Faites-moi 1 câlin !

if content then
    Oh oui ! Faisons 1 gros câlin ! Tu l'as bien mérité après tes 20/20 que
    tu viens de ramener ! On t'aime à 100% <3
else
    - Mon enfant, il va falloir que tu vois la vie avec ses 0xFFFFFF couleurs, si tu veux
    espérer avoir des 10aines de câlins ... Va nous acheter une demi-12aine d'œufs.
    - Bouuh ! Je m'en vais pleurer pendant 42 ans.
    * La justice condamna la famille à 14 années de prison et 7832 € d'amende. *
```

Le programme va renvoyer la sortie suivante :

```
int(1)IFTHENint(1)int(20)int(20)int(100)int(3)ELSEhex(16777215)int(10)int(12)int(42)int(14)int(7832)
```

Analyse syntaxique

Le programme `lang`, après compilation en appelant successivement `make langlex.c` et `make lang`, se comporte comme un interpréteur d'un langage de programmation fait à la main. L'extension attribuée est le `*.my`, même si cela n'est pas obligatoire.

Un programme est composé de trois parties :

- La déclaration des variables booléennes (faculatif). Cela se fait en écrivant `bool` suivi de la liste des variables booléennes à déclarer séparées par des virgules. La déclaration doit se finir par un point-virgule. Par exemple, on notera `bool x, y, z;` pour déclarer les variables booléenes `x`, `y` et `z`.
- De la même manière, on notera `int a, b, c;` pour déclarer les variables entières `a`, `b` et `c` (facultatif). La déclaration des variables entières doit se faire après la déclaration des variables booléennes s'il y en a.
- Enfin, le code. Le code est une succession d'instructions séparées par des point-virgules. Il existe 4 types d'instructions :
  - L'affectation, de la forme `x := expr` où `x` est une variable déclarée et `expr` une expression du même type que `x`.
  - Un test conditionnel, de la forme `if cond then stmt1 else stmt2` où `cond` est une expression booléenne et `stmt1` et `stmt2` deux blocs d'instructions. L'interprétation est naturelle : si la condition `cond` vaut `true`, on exécute le bloc `stmt1`, sinon on évalue le bloc `stmt2`. La partie `else stmt2` est facultative. Le problème du `dangling else` est résolu de la même manière qu'en C : le premier `else` porte sur le premier `if`.

- Une boucle de la forme `while cond do stmt od` où `cond` est une expression booléenne et `stmt` une suite d'instructions. L'expression `cond` sera évaluée à chaque passage de boucle, et on reste dans la boucle tant que que la condition vaut `true`.
- Un affichage des variables. Le code doit être de la forme `print list` où `list` est la liste des variables à afficher, séparées par des virgules. La fonction fonctionne à la fois avec des variables booléennes et des variables entières.

Une expression peut être de différentes formes. Chaque type d'expression est associée à une "classe" : booléen ou entier, ou les deux.

- La valeur d'une variable (du type de la variable)
- Un ou exclusif de deux expressions booléennes (renvoie un booléen)
- La disjonction de deux expressions booléennes (renvoie un booléen)
- La conjection de deux expressions booléennes (renvoie un booléen)
- L'équivalence deux expressions booléennes (renvoie un booléen)
- Une négation d'une expression booléenne (renvoie un booléen)
- `true` (booléen pour dire vrai)
- `false` (booléen pour dire faux)
- Une expression entre parenthèse (du type de ce qui est entre parenthèses)
- La somme de deux expressions entières (renvoie un entier)
- Le produit de deux expressions entières (renvoie un entier)
- La comparaison par inégalité stricte entre deux expression entières (renvoie un booléen)
- L'égalité entre deux expressions entières (renvoie un booléen)

Les priorités opérations sont les mêmes qu'en C :

1. La négation booléenne
2. La mutliplication entière
3. L'addition entière
4. L'inégalité stricte entre entiers
5. L'égalité entière
6. L'équivalence booléenne
7. La conjonction
8. La disjonction (éventuellement exclusive)

Lors de l'évaluation des expressions, les types sont correctement vérifiés.