Projet Informatique S3 : Interpréteur *SCHEME*

Rapport de premier livrable
26 Septembre 2016

I – Contexte & Objectifs

Il s'agit d'écrire en langage *C* un interpréteur *SCHEME*, qui est un langage fonctionnel dont nous ne connaissions pas le fonctionnement.

La première étape de ce projet a donc été d'écrire des fonctions qui vont récupérer une S-expression saisie au clavier par l'utilisateur, analyser chaque atome puis créer un arbre syntaxique qui sera exploité dans la suite du projet.

II – Démarche Suivie

Nous avons tout d'abord commencé à définir l'environnement de codage : nous travaillons sur nos machines personnelles équipées d'une distribution Ubuntu et nous échangeons les fichiers du projet via un GitHub créé pour l'occasion.

Une fois le sujet lu et assimilé, nous avons pu observer les parties du code déjà fourni afin de nous lancer dans l'écriture des fonctions de base pour la création des types d'objets.
En effet la structure *object* accepte les types *integer, symbol, string, character, boolean, pair & special* qui seront utilisés pour stocker les atomes lus dans la S-expression.

```
object make_integer( int val ){
        object t = make_object( SFS_NUMBER );
        t->this.number = val;
        return t;
}
```

Fig II.1 Code de la fonction make_integer

Outre les variables globales *nil* (liste vide en scheme), T*RUE & FALSE* (booléens créés à partir des fonctions make_boolean) nous avons également dû nous occuper des fonctions de lecture des S-expressions.
Il faut d'abord lire les atomes pour déterminer leur type et les stocker accordément dans la structure *object.* Pour cela nous avons choisi de découper la chaine de caractères entrée au clavier entre les parenthèses en une chaîne de caractère qui est séparée par des espaces : on a ainsi une chaine qui correspond directement à un atome. En faisant des test sur cette nouvelle chaine isolée on peut alors déterminer le type correspondant de l'atome avec la fonction *get_atom* dont le détail est donné Fig II.2

```
void get_atom(char* input, uint *here, char *str){
  uint indice =0;
  while (isspace(input[*here]) && input[*here] != '\0') (*here)++;
  if(input[*here]=='"'){
    do{
      str[indice] = input[*here];
      (*here)++;
      indice ++;
    }while (input[*here] != '"' );
    str[indice]=input[*here];
  }
  else{
    while (isspace(input[*here]) == 0 && input[*here]!='\0'){
      str[indice] = input[*here];
      (*here)++;
      indice ++;
    }
  }
  if(strlen(str)<BIGSTRING) str[indice +1] = '\0';
}
```

Fig II.2 Code de la fonction get_atom

Une fois cette chaine récupérée et traitée selon son type, l'atome lu peut être affiché par la fonction *sfs_print_atom*

III – Résultats Obtenus & Evaluation

Pour le moment nous avons supposé que l'utilisateur ne faisait pas de faute de syntaxe lors de sa saisie.
Les espaces superflus sont tous gérés, par exemple ' 5 ' sera lu comme '5'.
Nous avons d'abord mené des tests basiques pour vérifier le stockage correct des types d'atomes : cela fonctionne pour les int, les string, les boolean, les caractères spéciaux *#\space & #\newline* ainsi que les symboles.
Cependant les fonctions de lecture des paires n'étant pas encore terminées, la construction de l'abre syntaxique n'est pas possible

IV – Suite du Projet

Maintenant que les atomes sont correctement reconnus, il faut les stocker dans l'arbre et pour cela il nous faut impérativement terminer les fonctions read_pair et print afin de pouvoir analyser lexicalement les S-expressions entrées au clavier.