université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY
Langages dynamiques
Master 1 ISD

# TP nº 2

## Calculatrice

On souhaite implémenter une calculatrice simple composées de boutons et d'un écran. Les boutons sont les chiffres de 0 à 9, le séparateur décimal « . », les cinq opérations, les parenthèses, une touche d'effacement de l'écran et une pour effacer le dernier caractère.

On procède de la manière suivante :

— Création d'une hiérarchie de classes représentant des expressions arithmétiques.
— Utilisation d'expressions régulières pour découper une chaîne de caractères en suite de *token*
— *Parsing* et création d'une expression arithmétique à partir de la liste de *tokens* en utilisant l'algorithme de « l'aiguillage » de Dijkstra.

Une fois tout ces éléments en place, on pourra concevoir l'interface graphique.

### Formules

1. Ouvrir le fichier `index.html` et rajouter dans la section `head` une référence au fichiers `lexer.js` et `formula.js` (dans cet ordre).
2. Ouvrir le fichier `formula.js`. Compléter en rajoutant une sous-classe `Const` à la classe `Formula`. Le constructeur `Const` attends une valeur numérique en argument et le stocke dans une propriété `.value` de l'objet. La propriété `.priority` doit valoir 10 et la propriété `.arity` doit valoir 0. Les objets `Const` contiennent une méthode `.eval()` qui renvoie la valeur de la constante.
3. Rajouter une sous-classe `Binop` à `Formula`. Cette sous-classe a un constructeur prenant en argument deux objets `left` et `right`, initialise les champs `left` et `right` correspondant et initialise le champ `arity` à 2.
4. Rajouter ensuite une sous-classe `Add` à `Binop` dont le constructeur prend deux arguments et les passe aux arguments de la classe parente. La propriété `.priority` est initialisée à 3. La classe `Add` doit posséder une méthode `.eval()` qui évalue les deux sous-arbres `this.left` et `this.right` et renvoie leur somme. Tester dans la console le code de la manière suivante :

   ```
   let c1 = new Const(1);
   let c2 = new Const(2);
   let a = new Add(c1, c2);
   a.eval ();
   ```

   Cette dernière expression doit bien évidemment renvoyer 3.
5. Créer des classes `Sub`, `Mul`, `Div` et `Mod`. Toutes on une propriété `.arity` vallant `2` et des propriétés `priority` valant respectivement 3, 5, 5, 5. Tester les divers constructeurs.

### Lexing/parsing

La classe `Lexer` est fournie et commentée. Il peut être judicieux de la lire et comprendre son fonctionnement. Informellement, le constructeur `Lexer` attends en argument un tableau d'objets. Chaque objet est de la forme `{ re:` $r$`, action:` $f$ `}` où $r$ est une expression régulière et $f$ une fonction. Une fois construit un `Lexer` $l$, on peut l'utiliser de la manière suivante : $l$`.scan("chaîne de caractères")`. Le lexeur analyse la chaîne en position 0. Il essaye tour à tour toutes les expressions régulières dans l'ordre du tableau. S'il y a une correspondance, le lecteur appelle l'action $f$ correspondante en lui passant en argument la chaîne reconnue, la position du début dans le texte initial et la position finale. Puis l'analyse reprends à partir de la fin de la chaîne trouvée, jusqu'à la fin du texte d'entrée. Par exemple :

```
var rules = [
      { re : /[^ ]+/, action: function (s, i, j) { return s.toUpperCase(); } },
      { re : /[ ]+/, action: function (s, i, j) { return "-"; }}
];
let lex = new Lexer (rules);
let tokens = lex.scan("chaîne   de caractères");
//tokens vaut : [ "CHAÎNE", "-", "DE", "-", "CARACTÈRES" ]
```

1. Compléter les expressions régulières dans la fonction `Formula.parse` aux endroit indiqués.
2. Lire le pseudo-code de l'algorithme du *shunting yard* [1] et le compléter, à la fin de la fonction `Formula.parse`.
3. Charger le fichier `index.html` et vérifier votre implémentation par des tests. Par exemple :

```
let f = Formula.parse ("1 + 2 * 3.5");
let r = f.eval();
// r vaut 8
```

### Interface graphique

On souhaite maintenant compléter l'interface graphique de la calculatrice.

1. Ouvrir le fichier `index.html` et en comprendre la structure. Il y a trois éléments :
   — Un `div "screen"` contenant la formule en cours d'écriture
   — Un `div "subscreen"` contenant la valeur de la formule en cours d'écriture, ou un message d'erreur
   — `div "button"` contenant les boutons. Chaque bouton à un attribut `value` contenant une chaîne de caractères.
2. Compléter le fichier `calculette.js` aux endroits demandés. On utilisera la délégation d'évènements pour gérer tous les boutons avec un unique gestionnaire accroché à l'élément `div` d'id `"buttons"`.

## Évènements (optionnel)

Proposer un petit fichier HTML et code javascript qui illustre la différence de propagation des évènements entre le mode « standard » et le mode « microsoft ».

**Réponse:** Il suffit de créer des `div` imbriqués avec des `id` différents. Pour chacun, rajouter un gestionnaire d'évènement `click` avec le troisième paramètre à `false` ou absent (comportement standard) ou `true` (comportement `microsoft`). Chaque évènement peut ensuite faire un `console.log(ev.currentTarget.id)` par exemple et on peut constater que l'ordre d'affichage change.

Le corrigé rajoute un effet visuel pour que les éléments changent de couleur au fur et à mesure qu'ils sont traversés (fichier `event.html`)

1. https://en.wikipedia.org/wiki/Shunting-yard_algorithm