3103 – Algorithmique Avancée

Récursion, Mola Kucher.

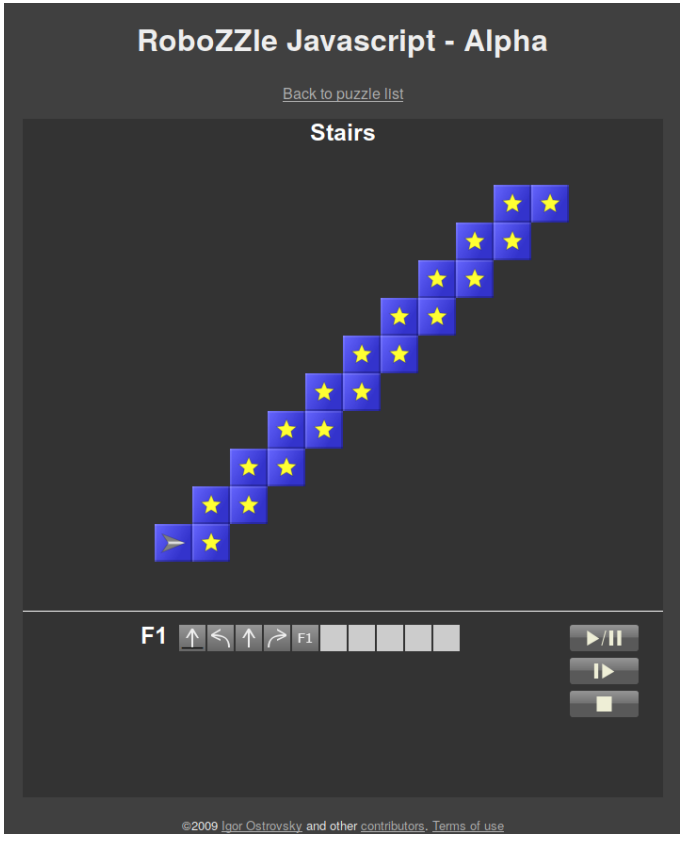
Contenu du cours

Idéalement, dans ce module, nous allons parler de :

1. La récursivité
   - Récursivité simple
   - Condition d'arrêt
   - La pile et la récursivité terminale.
2. Les structures récursives
   - Les arbres
   - XML
3. Les structures associatives
   - Table de hashage
   - dictionnaire

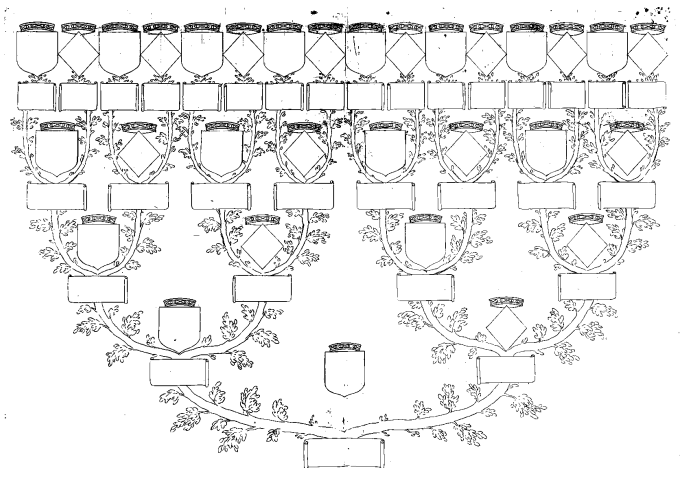
Pourquoi la récursivité ?

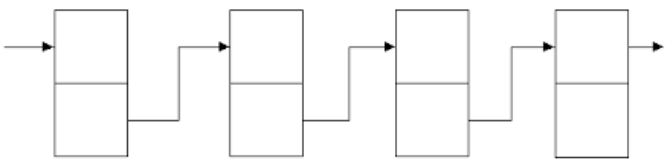
- Un autre modèle de boucle.



Pourquoi la récursivité ?

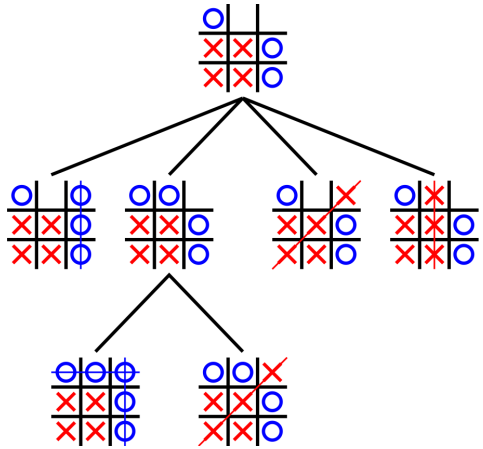
- Un autre modèle de boucle.
- Structures naturellement récursives





Pourquoi la récursivité ?

- Un autre modèle de boucle.
- Structures naturellement récursives
- Stratégies récursives



Exemple

$log_2(x)$: combien de fois il faut diviser x par 2 pour atteindre 1.

```
static int log2(int x){
  if(x <= 1)
    return 0;
  else
    return 1 + log2(x/2);
}
```

- Déroulé au tableau

Autre exemple

```
static int mystere(int n){
  return n * mystere(n-1);
}
```

- Que fait cette fonction ?

Autre exemple

```
static int mystere(int n){
  return n * mystere(n-1);
}
```

- Que fait cette fonction ?
- prendre les mêmes précautions que pour un `while`

Règle du jeu :

```
<T> ma_fonction_recursive(<U> param){
  if(<condition d arret>){
    arret;
  }
  else{
    calcul recursif;
  }
}
```

Ne suffit pas toujours

```
static void syracuse(int n){
  if(n <= 1){
    System.out.println("You␣win!");
  }
  else{
    if(n \% 2 == 0)
      return syracuse(n / 2);
    else
      return syracuse(3 * n + 1);
  }
}
```

Fin de la fonction

Attention

Un appel récursif n'est pas un `goto()`

- La fin de la séquence en cours s'exécute à la sortie de l'appel.

La pile

```
void affiche(int n){
  if(n>0){
    System.out.println("A␣" + n);
    affiche(n-1);
    System.out.println("B␣" + n);
  }
}
```

- Rôle important du placement de l'appel récursif
- Pile d'exécution

Conversion itératif/récursif

```
static int log2(int x){
  if(x <= 1)
    return 0;
  else
    return 1 + log2(x/2);
}
```

```
static int log2(int x){
  int res = 0;
  while(x>1){
    x = x/2;
    res = res + 1;
    }
  return res;
}
```

- Facile quand l'appel est en fin de fonction.
- Plus délicat quand l'appel est ailleurs.

Récursivité terminale

Principe :

Le résultat de l'appel récursif est le résultat de la fonction.

- `return f(n-1);` **Ok**
- `return 3 + f(n-1);` not Ok !

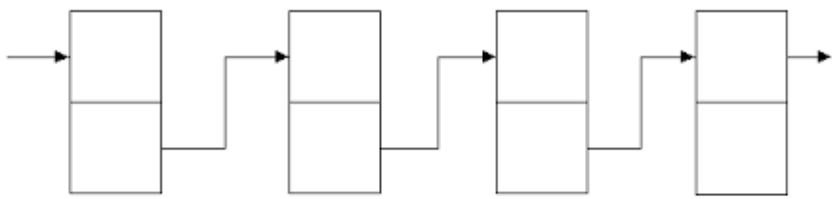
Passage à la récursivité terminale

```
static int log2(int x, int res){
  if(x <= 1)
    return res;
  else
    return log2(x/2, res + 1);
}
```

- Cela permet d'éviter la pile.
- Strictement équivalent à un algorithme itératif.
- Parfois plus lisible

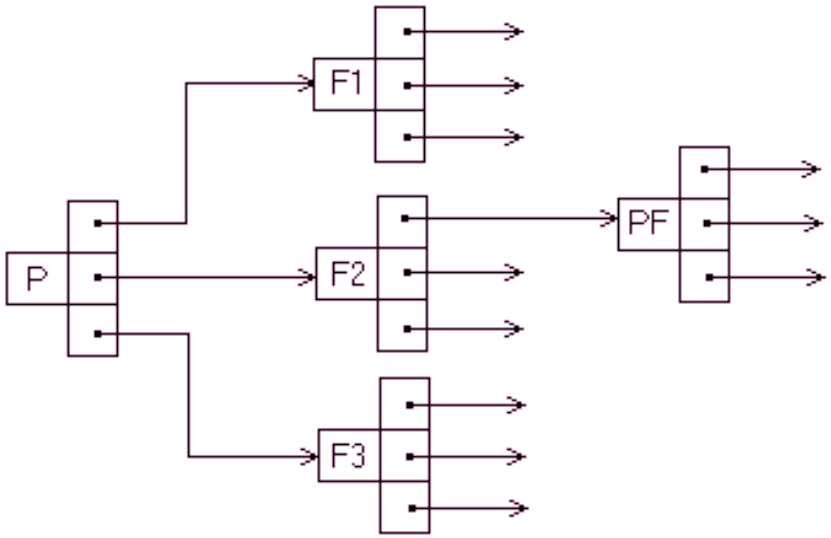
Liste chainées

```
static void affiche( ListIterator liste){
  if(liste.hasNext()){
    System.out.println(liste.next());
    affiche(liste);
  }
}
```



- Récursivité terminale naturelle
- Plus *élégant* qu'un `while`

Listes multi-chainées



- Un objet plus délicat à explorer.
- Une structure de donnée compacte.
- Donc plus rapide à fouiller ?

Synthèse du premier cours

À retenir

- Schéma d'une fonction récursive (arrêt + appel)
- Pile d'appel
- Notion de récursivité terminale

Synthèse du premier cours

À retenir

- Schéma d'une fonction récursive (arrêt + appel)
- Pile d'appel
- Notion de récursivité terminale

Pendant les vacances : Robozzle, lightbot2...