EXEMPLES UTILISANT DES TABLEAUX

- Toutes les listes multiplicatives avec terminaison n
- Recherche dichotomique
- recherche d'un parcours de l'échiquier par un cavalier

Eviter les effets de bord (mais aussi les copies coûteuses)!

STRUCTURES RECURSIVES

- Définition de structures qui contiennent des pointeurs vers d'autres structures du même type

- listes

- arbres

- avec sous-arbres en commun

- et boucles

- souvent traitées par des fonctions/procédures récursives

STRUCTURES RECURSIVES en Exalgo

- Nouveaux types introduits par définitions `type id : ^ type_id;` pour un pointeur ou `type id : enreg id,id,id:type;... finenreg;` pour enregistrements

- Pour le type d'un enregistrement qui contient un pointeur au même type il faut avoir deux définitions mutuellement récursives (le pointeur et l´enregistrement)

- Recommandé de définir d'abord le type pointeur et (immédiatement) ensuite l'enregistrement

- exemple d'une liste:

```
type p:^S;

type S: enreg
valeur:entier;
reste: p
finenreg;
```

Les listes

- longueur
- somme
- concaténer
- fusionner deux listes triées
- insérer: choisir le bon mode de passage de paramètres pour pouvoir insérer au début

`listes.exo`

Les arbres (binaires)

- Les trois ordres de parcours

- préfixe, symétrique, postfixe

- tous programmés de façon très simple est claire par récursion

`arb-bin-rec.exo`

Parcours en largeur d'un arbre

- Pas de solution simple récursive ou autre
- liste (FIFO) de noeuds en attente d'être traité
- Extraire un noeud, traiter le, ajouter ses successeurs
- terminer quand la liste est vide

`largeur.exo`

Parcours en largeur de structures plus complexes

- Eviter les boucles
- Prévoir un champs qui pourra être utilisé comme marqueur des noeuds déjà visités
- ne rien faire si un noeud est déjà marqué
- marquer un noeud AVANT de visiter ses successeurs
- (evite aussi de parcourir plusieurs fois la même sous-structure)
- remettre les marqueurs à 0 (faux) à la fin??

FONCTIONS PURES

- Théorie selon laquelle les fonctions écrites sans affectations et effets de bord sont plus claires et plus facilement écrites sans bugs
- plus proche de l'idée mathématique d'une fonction; et les définitions ressemblent à des équations
- Une question de goût
- mais récursion nécessaire pour ce style de programmation

QUAND DOIT ON UTILISER LA RÉCURSION?

- Quels sont les fonctions pour lesquelles récursion est essentielle?

- Aucunes: ce qu'on peut faire avec récursion, on peut toujours faire avec une pile explicite!

- Jeux de parcourir un arbre sans récursion et sans pile!!

QUAND DEVRAIT ON L'UTILISER?

- Quand le code est plus simple à écrire et comprendre avec récursion

- Dont presque tous les cas où l'alternative serait une pile explicite

- Mais si je veux parcourir deux arbres en même temps (par exemple je veux les fusionner), je ne peux pas forcément le faire récursivement
(mais "coroutines" récursives existent en certains langages)

LE COUT

- Espace: negligeable:
  nécessité de stocker les paramètres et variables locales de chaque appel (activation); utilisation invisible d'une pile; sauf très gros tableaux, cet espace n'est pas important

- Temps: negligeable:
  les instructions supplémentaires exécutées à chaque appel d'une fonction prennent un temps petit par rapport aux calculs de la fonction sauf si ces calculs sont simples

- La qualité du programme est plus importante
  et un bon compilateur peut quelquefois générer le même code pour les deux versions (cas de récursion à la fin)