- TP 3 -
Les super agents

On souhaite écrire un programme permettant d'enregistrer des agents secrets et de faire des recherches dans le fichier d'une agence de renseignements.

On peut encore une fois utiliser l'unité qui implémente les primitives des arbres binaires et qui a été développée précédemment mais il faudra la compléter avec les opérations nécessaires pour les nouvelles fonctionnalités. Plus précisément, il faudra que les arbres construits soient des arbres binaires ordonnés permettant une recherche potentiellement plus efficace que pour les arbre binaires classiques.

1.Ajouter un agent

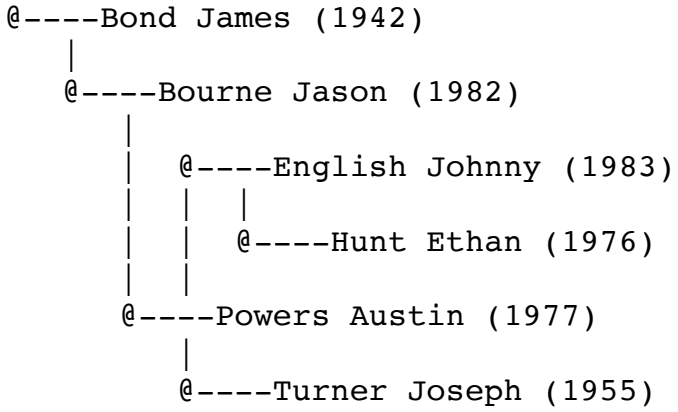
Proposer un type permettant d'identifier les agents qui sont caractérisés par un nom, un prénom et une date de naissance. Implanter la procédure permettant d'insérer un agent dans l'arbre de telle manière que l'arbre de recherche reste ordonné par rapport à l'ordre lexicographique sur les noms des agents.

Par exemple, l'exécution du code

```
jb = {"James","Bond",1942};
bo = {"Jason","Bourne",1982};
ap = {"Austin","Powers",1977};
je = {"Johnny","English",1983};
eh = {"Ethan","Hunt",1976};
jt = {"Joseph","Turner",1955};
insert(&Ajb,jb);
insert(&Ajb,bo);
insert(&Ajb,ap);
insert(&Ajb,je);
insert(&Ajb,eh);
insert(&Ajb,jt);
infix(Ajb);
graphical_print(Ajb,0);
```

où `infix` est la fonction qui réalise un affichage infixé de l'arbre et `graphical_print` est la procédure qui permet la simulation d'une présentation graphique, doit produire l'affichage

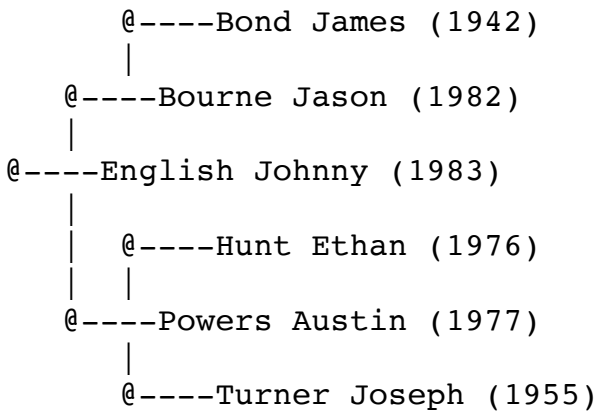
```
Bond James (1942)
Bourne Jason (1982)
English Johnny (1983)
Hunt Ethan (1976)
Powers Austin (1977)
Turner Joseph (1955)
```



2.Chercher un agent
Implanter la fonction permettant de chercher efficacement un agent dans l'agence.

3.Supprimer un agent
Implanter la procédure permettant de supprimer un agent de l'agence. L'arbre binaire correspondant doit bien évidement rester ordonné à la fin de l'opération.

4.Améliorer la recherche
L'agence emploie maintenant un nombre important d'agents et les recherches se révèlent inefficaces. Il faudra compléter l'implantation actuelle pour permettre l'utilisation des arbres binaires de recherche équilibrés. Si la procédure d'insertion est modifiée pour garantir des arbres équilibrés alors l'exécution du code pour le premier point doit produire l'affichage:



5.Dernières recrues
On a remarqué que généralement les derniers agents arrivés dans l'agence sont ceux sollicités pour les missions courantes. Il faudra donc effectuer l'insertion des agents à la racine de l'arbre pour améliorer l'efficacité des recherches les concernant.

Il faudra implémenter la méthode présentée en cours qui consiste à découper l'arbre en initial en deux arbres avant l'insertion.

Optionnel: Vous pouvez également implémenter la méthode basée sur les rotations gauche/droite. Dans ce dernier cas, les axiomes pour les procédures correspondantes sont les suivantes:

```
rotationGauche(Cons(v,l,r)) = Cons(racine(r), Cons(v,l,FG(r)), FD(r))
rotationDroite(Cons(v,l,r)) = Cons(racine(l), FG(l), Cons(v,FD(l),r))
```

```
insertionRacine(v,  ConsVide())  =  Cons(v,  ConsVide(),  ConsVide())
insertionRacine(v2, Cons(v1, l, r)) =
      rotationDroite(Cons(v1, insertionRacine(v2,l), r)) si v2 < v1
 insertionRacine(v2, Cons(v1, l, r)) =
      rotationGauche(Cons(v1, l, insertionRacine(v2,r))) si v1 < v2
```

Voici un exemple de rotation gauche est parfaitement symétrique. rotation droite; la rotation droite