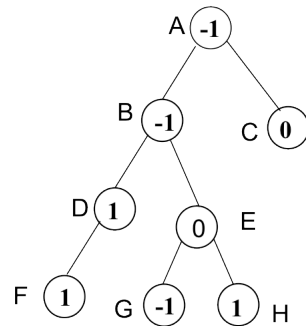


Question 3. Soit un arbre binaire dont les sommets contiennent un élément de $A=\{-1,0,1\}$. Dans la suite on appellera ce type d'arbre A-arbre. La figure ci-dessous donne un exemple de A-arbre, les lettres représentent les adresses des sommets.



1 – Ecrire la fonction *verifier* qui vérifie qu'un arbre binaire est un A-arbre c'est à dire que les sommets de l'arbre sont correctement étiquetés (i.e.éléments de A).

2 – En utilisant un parcours hiérarchique et les primitives du type abstrait listeSC, écrire la fonction *arbreListes* qui prend pour paramètre un A-arbre et fournit en sortie un tableau T tel que T[i] donne accès à la liste des adresses des sommets ayant pour étiquette i. sur l'exemple $T[-1]=(A,B,G)$, $T[0]=(E)$, $T[1]=(D,F,H)$.

3 – On dit qu'un sommet s est sympathique s'il a deux fils et si $\text{valeur}(\text{filsGauche}(s)) + \text{valeur}(\text{filsDroit}(s)) = \text{valeur}(s)$. Ecrivez la fonction *nbSympa* qui prend pour argument un A-arbre et fournit comme résultat le nombre de nœud sympathique de l'arbre. Dans l'exemple, le résultat est 2.

Question 4. On souhaite gérer un ensemble de polygones. Chaque polygone est défini par une suite de points. Chaque point est représenté par une paire (abscisse, ordonnée). Il est associé à une chaîne de caractère qui est son nom ainsi que deux attributs : la couleur de remplissage du polygone et la couleur de son contour. La couleur est représentée en système RVB (Rouge,Vert,Bleu), soit un triplet d'entiers compris entre 0 et 255 (par exemple (0,0,0) représente le noir). Chaque polygone est de plus muni d'un paramètre « profondeur » qui permet l'affichage. Le polygone de plus faible profondeur est affiché en dernier. Dans la suite, on décrira précisément chaque structure de donnée. On précisera s'il s'agit ou non d'un conteneur. On ne demande pas d'écrire les primitives mais uniquement de donner l'en-tête de la primitive et sa fonction.

- 1 – Décrivez la structure de données *couleur* permettant de stocker les couleurs. En donner les primitives.
- 2 – Décrivez la structure de données *polygone* permettant de décrire les informations liées à un polygone. En donner les primitives.
- 3 – Quelles sont les structures de données que l'on peut-on utiliser afin qu'un utilisateur puisse accéder aux paramètres d'un polygone par son nom ? Justifiez. Donnez les avantages et les inconvénients de chacune.
- 4 – Un utilisateur peut accéder à un polygone pour le faire passer au premier plan et au dernier plan en agissant sur le paramètre profondeur. Proposer une structure de donnée permettant d'accéder facilement au polygone situé en premier plan. Justifiez.
- 5 – Écrire la fonction *ajouterPolygone* qui prend comme paramètre un polygone et permet d'insérer le polygone dans toutes les structures de données que vous venez de décrire. On ne demande pas de réimplémenter les primitives vues en cours mais de dire comment on agit sur les clés.

Listes simplement chaînées (listeSC)

```
fonction valeur(val L:liste d'objet):objet;
fonction debutListe(val L:liste d'objet) :vide ;
fonction suivant(val L:liste d'objet) :vide ;
fonction listeVide(val L:liste d'objet): boolean;
fonction créerListe(ref L:liste d'objet):vide;
fonction insérerAprès(ref L:liste d'objet; val x:objet;):vide;
fonction insérerEnTete(ref L:liste d'objet val x:objet):vide;
fonction supprimerAprès(ref L:liste d'objet):vide;
fonction supprimerEnTete(ref L:liste d'objet):vide;
fonction détruireListe(ref L:liste d'objet):vide;
```

Listes doublement chaînées (listeDC)

```
fonction finListe(val L:liste d'objet):vide;
fonction précédent(val L::liste d'objet): vide;
```

Piles

```
fonction valeur(ref P:pile de objet):objet;
fonction pileVide(ref P:pile de objet):booléen;
fonction créerPile(P:pile de objet) :vide
fonction empiler(ref P:pile de objet;val x:objet):vide;
fonction dépiler(ref P:pile de objet):vide;
fonction détruirePile(ref P:pile de objet):vide;
```

Files

```
fonction valeur(ref F:file de objet):objet;
fonction fileVide(ref F:file de objet):booléen;
fonction créerFile(F:file de objet);vide;
fonction enfiler(ref F:file de objet;val x:objet):vide;
fonction défiler (ref F:file de objet):vide;
fonction détruireFile(ref F:file de objet):vide;
```

Arbres binaires

```
fonction getValeur(val S:sommet):objet;
fonction filsGauche(val S:sommet):sommet;
fonction filsDroit(val S:sommet):sommet;
fonction pere(val S:sommet):sommet;
fonction setValeur(ref S:sommet;val x:objet):vide;
fonction ajouterFilsGauche(ref S:sommet, val x:objet):vide;
fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;
fonction supprimerFilsGauche(ref S:sommet):vide;
fonction supprimerFilsDroit(ref S:sommet):vide;
fonction détruireSommet(ref S:sommet):vide;
fonction créerArbreBinaire(val Racine:objet):sommet;
fonction détruireArbreBinaire(val Racine:objet):sommet;
```

Arbres planaires

```
fonction valeur(val S:sommetArbrePlanaire):objet;
fonction premierFils(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction frere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction pere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction créerArborescence(val racine:objet):sommetArbrePlanaire;
fonction ajouterFils(ref S:sommetArbrePlanaire, val x:objet):vide;
fonction supprimerSommet(ref S:sommetArbrePlanaire):vide;
fonction détruireArborescence(val racine:objet):sommetArbrePlanaire;
```

Arbres binaire de recherche

```
fonction ajouter(ref A:arbreBinaire d'objets, val v:objet):vide;
fonction supprimer(val A: arbreBinaire d'objets, val v:objet):vide;
```

Tas

```
fonction valeur(ref T:tas d'objet): objet;
fonction ajouter(ref T:tas de objet, val v:objet):vide;
fonction supprimer(val T:tas de objet):vide;
fonction creerTas(ref T:tas, val v:objet):vide;
fonction détruireTas(ref T:tas):vide;
```

File de priorité

```
fonction changeValeur(ref T:tas d'objet, val s:sommet, val v:objet):vide;
```

Dictionnaire

```
fonction appartient(ref d:dictionnaire, val M::mot):booléen;
fonction creerDictionnaire(ref d: dictionnaire):vide ;
fonction ajouter(ref d:dictionnaire, val M::mot):vide;
fonction supprimer(ref d:dictionnaire, val M:mot):vide;
fonction détruireDictionnaire(ref d:dictionnaire):vide;
```

Table de Hachage

```
fonction chercher(ref T:tableHash de clés, val v:clé):curseur;
fonction créerTableHachage(ref T: tableHash de clé, ref h:fonction):vide;
fonction ajouter(ref T:tableHash de clé, val x:clé):booléen;
fonction supprimer((ref T:tableHash de clé, val x:clé):vide;
fonction détruireTableHachage(ref T: tableHash de clé, ref h:fonction):vide;
```

FIN