**DE**

**Structures de données et algorithmique**

**L’3, ING1**

**Les algorithmes devront être écrits en langage algorithmique. Les données, les données modifiées, le résultat et les variables locales devront être précisés à chaque exercice.**

**Il vous suffit de faire sept exercices au choix sur neuf qui figurent dans l’énoncé.**

**Sans documents, sans calculatrice.**

1. ***Questions de cours (5)***
2. Quelle est la différence entre une LSC (liste simplement chainée) et une LDC (liste doublement chainée)
3. Quel est le critère d'arrêt lors d'un parcours d'une liste simplement chainée circulaire?
4. Comparer les notions de file et de pile.
5. Donner le contenu d'une pile pour chaque opération de la suite A B C D E\* \* \* M N\* \*. Chaque lettre provoque un empilement et chaque astérisque un dépilement. Faites la même chose pour une file.
6. Montrez les résultats des parcours préordre, ordre et postordre sur l’exemple :

5

 9 6

 3 8 4

 3 7

 10

1. Construire un ABR en y rajoutant au tour de rôle :

10, 20, 18, 5, 8, 11, 6, 13

1. Réponse dans le cours
2. quand le maillon étudié (pointé par cour) pointe sur le premier maillon de la liste (pointé par l) : cour🡪succ = l
3. Réponse dans le cours
4. ABCDE – AB – ABMN – AB (pile), ABCDE – DE – DEMN – MN (file)
5. Préordre : 5, 9, 3, 8, 3, 10, 7, 6, 4

Ordre : 3, 9, 3, 10, 8, 7, 5, 6, 4

Postordre : 3, 10 , 3, 7, 8, 9, 4, 6, 5

f)

 10

 5 20

 8 18

 6

 11

 13

***Listes simplement chaînées***

1. ***(+) (2)***

Concevoir un algorithme qui permet de déterminer la valeur minimale d’une liste chaînée d’entiers positifs.

2.

Si la liste est vide, l’algorithme retourne 0.

Minimum (l : liste) : entier

donnée l : liste

résultat de type entier

variable locale  min : entier

début

 si l = nul

 retourner 0

 min 🡨 l🡪info

 tant que l <> nul faire

 si l🡪info < min alors

 min 🡨 l 🡪info

 l🡨 l 🡪 succ

 fin

retourner min

fin

1. ***(+) (2)***

Concevoir un algorithme qui permet de calculer le produit des valeurs contenues dans une liste chaînée d’entiers.

3.

Si la liste est vide, l’algorithme retourne 0.

Produit (l : liste) : entier

donnée l : liste

résultat de type entier

variable locale produit : entier

début

 si l = nul

 retourner 0

produit 🡨 1

 tant que l <> nul faire

 produit 🡨 produit \* l🡪info

 l 🡨 l🡪succ

 fin

 retourner produit

fin

1. ***(++) (3)***

Concevoir un algorithme qui permet d’insérer un entier dans une liste chaînée d’entiers triés dans l’ordre décroissant.

4.

Insertion (l : liste, x : entier)

donnée modifié  l : liste

donnée x : entier

variable locale cour : liste

variable locale tmp : liste

variable locale prec : liste

début

 reserver (cour)

 cour 🡪info 🡨 x

 cour 🡪 succ 🡨 nul

 SI l = nul Alors

 l 🡨 cour

retourner

 fin

sI x > l🡪info Alors

 cour 🡪 succ 🡨 l

 l 🡨 cour

 retourner

fin

tmp 🡨 l🡪succ

prec 🡨 l

tant que tmp <> nul faire

si x > tmp🡪info alors

prec 🡪 succ 🡨 cour

 cour 🡪 succ 🡨 tmp

 retourner

fin

prec 🡨 tmp

tmp 🡨 tmp🡪succ

fin

prec 🡪 succ 🡨cour

 fin

1. ***(++) (3)***

Concevoir un algorithme permettant de supprimer d’une liste chaînée d’entiers triée dans l’ordre croissant tous les entiers supérieurs à une valeur n donnée.

5.

L’algorithme retourne faux s’il n’a rien supprimé

Suppression (l : liste, n : entier) : booléen

donnée modifiée  l : liste

donnée n : entier

résultat de type booléen

variable locale ok : booléen

variable locale cour : liste

variable locale tmp : liste

début

 si l = nul alors

 retourner faux

 fin

ok 🡨 faux

cour 🡨 l

tant que ok = faux et cour <> nul faire

 si cour 🡪 info < = n alors

 cour 🡨 cour 🡪succ

 sinon

 ok 🡨 vrai

 fin

fin

si cour = nul alors

 retourner faux

tant que cour <>nul faire

 tmp 🡨 cour

 cour 🡨 cour 🡪 succ

 libérer tmp

fin

retourner vrai

fin

***Piles et files***

1. **(++) (3)**

Concevoir un algorithme permettant de transférer le contenu d'une pile dans une file.

Vous avez le droit d’utiliser les algorithmes de traitement des piles/files :

Empiler (p : pile\_adr, e : T)

Dépiler (p : pile\_adr) :T

Enfiler (f : file\_adr, e : T)

Defiler (f : file\_adr) : T

Est\_Pile\_Vide (p : pile\_adr) : booléen

Est\_File\_vide (f : file\_adr) : booléen

6.

Créer file\_de\_pile (p : pile) : f

donnée p : pile

résultat de type file

Variable locale x : T

Variable locale f : file

début

f 🡨créer\_file ()

tant que est\_Pile\_vide (p) = faux faire

x 🡨 dépiler(p)

enfiler (f, x)

fin

fin

***Arbres***

1. **(+) (2)**

Ecrire un algorithme permettant d’afficher le contenu d’un arbre ABR dans l’ordre décroissant.

Affichage (a : arbre)

Donnée  a : arbre

début

 Si a <> nul alors

 Affichage (a🡪fd)

 Afficher (a🡪info)

 Affichage (a🡪fg)

 fin

fin

1. **(++) (3)**

Concevoir un algorithme permettant de trouver la différence entre la plus grande et la plus petite valeur d’un ABR.

8.

L’algorithme retourne -1 si l’arbre est vide

Différence (a : arbre) : entier

donnée a : arbre

variable locale x : T

variable locale cour : arbre

début

 Si a = nul

 retourner (-1)

 cour 🡨 a

 Tant que cour 🡪 fg <> nul faire

 cour 🡨cour 🡪fg

 fin

 x 🡨 cour 🡪info

 cour 🡨 a

 tant que cour 🡪fd <> nul faire

 cour 🡨 cour 🡪fd

 fin

 retourner (cour🡪info – x)

fin

1. **(++) (3)**

Concevoir un algorithme pour supprimer le plus petit élément d’un ABR

Suppression (a : arbre) : booléen

donnée modifiée a : arbre

résultat de type booléen

variable locale père : arbre

variable locale fils : arbre

début

 Si a = nul alors

 retourner faux

fin

père 🡨 a

fils 🡨 père 🡪 fg

Si fils = nul alors

 a🡨 a 🡪fd

 libérer père

 retourner vrai

fin

tant que fils🡪fg <> nul faire

 père 🡨 fils

 fils 🡨fils 🡪fg

fin

père 🡪fg 🡨 fils 🡪 fd

libérer fils

retourner vrai

fin