|  |  |  |
| --- | --- | --- |
| **Structure de données abstraites linéaires (SDA)** | **Les piles** | |
| En informatique, une **pile** (en anglais stack) est une structure de données fondée sur le principe «**dernier arrivé, premier sorti**» (ou LIFO pour **Last In, First Out**), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés.  Le fonctionnement est donc celui d’une pile d’assiettes : on ajoute des assiettes sur la pile (=on **empile**) , et on les récupère dans l’ordre inverse, en commençant par la dernière ajoutée (=on **dépile**). | |  | |

*Voici quelques exemples d’usage courant d’une pile :*

* *Dans un navigateur web, une pile sert à mémoriser les pages Web visitées. L’adresse de chaque nouvelle page visitée est empilée et l’utilisateur dépile l’adresse de la page précédente en cliquant le bouton «Afficher la page précédente».*
* *La fonction «Annuler la frappe» (en anglais «Undo») d’un traitement de texte mémorise les modifications apportées au texte dans une pile.*
* *La pile d’exécution d’un algorithme récursif*

**Opérations** de bases :

* **CREER\_PILE()** qui retourne un objet de type pile : *La pile existe et elle est vide*
* **EMPILER(P,e)** : L’élément e est inséré au sommet de la pile P.
* **DEPILER(P)** qui retourne l’élément situé au sommet de la pile et le supprime de la pile
* **EST\_VIDE(P)** qui retourne un objet de type Booléen : *Retourne Vrai si la pile est vide et retourne Faux sinon.*

|  |  |
| --- | --- |
| Exercice 1 : Donner l’état de la pile créée à la fin de l’exécution du programme : (utiliser une gomme !)  P=CREER\_PILE()  EMPILER(P,3) 9  EMPILER(P,2) 2  N=DEPILER(P) 5  EMPILER(P,5) 3  EMPILER(P,N)  EMPILER(P,9) P | Exercice 2 : Donner le contenu de P1 et P2 à la fin  P1=CREER\_PILE() 6  P2=CREER\_PILE() 4  Pour i allant de 1 à 5 : 5  EMPILER(P1,i) 4  EMPILER(P2,i+1) 5 2  Si i est pair alors 3 3  N=DEPILER(P1) 1 2  EMPILER(P2, N) P1 P2 |

|  |  |  |
| --- | --- | --- |
| Exercice 3 :  Écrire l’algorithme de la fonction sommet(P) qui renvoie le sommet de la pile (sans le supprimer de la pile) s’il existe.  Fonction sommet(P) :  Début  S DEPILER(P)  EMPILER(P,S)  RENVOYER S  fin. | Exercice 4 : a) Écrire l’algorithme de la fonction taille(P) qui retourne la taille de la pile P.  Fonction taille(P)  Début  C 0  Tant que non(EST\_VIDE(P)) faire  C  DEPILER(P)  Fin tant que  RENVOYER C  Fin | b) écrire une version où la pile P **n’est pas modifiée à la fin**  Fonction taille\_bis(P)  Début  P2 CRÉER\_PILE()  C 0  Tant que non(EST\_VIDE(P)) faire  C  EMPILER(P2,DEPILER(P))  Fin tant que  Tant que non(EST\_VIDE(P2)) faire  EMPILER(P,DEPILER(P2)  Fin tant que  RENVOYER C  Fin |

Début

………………………………………………….

Exercice 5 : on se donne une pile P1 contenant des entiers positifs.

|  |  |
| --- | --- |
| 1. Ecrire un algorithme pour déplacer les entiers de P1 dans une pile P2 de façon à avoir dans P2 tous les nombres pairs en dessous des nombres impairs   P2CRÉER\_PILE()  P3CRÉER\_PILE()  Tant que non(EST\_VIDE(P1)) faire  NDEPILER(P1)  Si N est pair alors  EMPILER(P2,n)  Sinon  EMPILER(P3,N)  Fin si  Tant que non(EST\_VIDE(P3) faire  EMPILER(P1,DEPILER(P3)  Fin tant que | 1. Écrire un algorithme pour copier dans P2 les nombres pairs contenus dans P1. Le contenu de P1 après exécution de l’algorithme doit être identique à celui avant exécution. Les nombres pairs dans P2 doivent apparaitre dans l’ordre où ils sont dans P1   P2 CRÉER\_PILE()  P3 CRÉER\_PILE()  Tant que non(EST\_VIDE(P1)) faire  EMPILER(P3,DEPILER(P1))  Fin Tant que  NDEPILER(P3)  EMPILER(P1,N)  Si N est pair alors  EMPILER(P2,N)  Fin Si  Fin tant que |
|  |  |

**Piles : Exercices d’application**

|  |  |
| --- | --- |
| **Exercice 1 – le bon parenthésage :** On souhaite écrire une fonction permettant d’abord de vérifier si une expression est bien ‘’parenthésée’’, et ensuite de renvoyer la liste (ou la pile) des couples correspondant aux indices de chaque parenthèse ouvrante et fermante qui lui correspond.  Algo1 : avec juste le retour d’un booléen  Algo2 : avec le retour (si bien parenthésée) d’une liste contenant les indices des couples de parenthèses comme les résultats à droite  Algo3 : on rajoute des crochets [ ] et des accolades {} à l’expression |  |

Principe de l’algorithme de l’Algo1 :

On crée une ………….,

On parcourt l’expression de gauche à droite.

Si on rencontre une parenthèse ouvrante "( " alors on ……………………….,

Si on rencontre une parenthèse fermante " ) " alors

Si la pile n’est pas vide on …………….

Sinon on retourne ……

A la fin la pile doit être ……..

1. Ecrire son code en Python sur spyder ou jupyter , en utilisant l’une des implémentation de la structure pile que vous avez déjà réalisées, et tester-le sur des expressions.
2. Réaliser les programmes Algo2 et Algo3

**Exercice 2 : la notation polonaise inversée  (fini les parenthèses ! ):**

Définition : La notation polonaise inverse (NPI) (en anglais RPN pour Reverse Polish Notation), également connue sous le nom de notation post-fixée, permet d’écrire de façon non ambiguë les formules arithmétiques sans utiliser de parenthèses.

Historique : Elle est dérivée de la notation polonaise utilisée pour la première fois en 1924 par le mathématicien polonais Jan Lukasiewicz, la NPI a été inventée par le philosophe et informaticien australien Charles Leonard Hamblin dans le milieu des années 1950, pour permettre les calculs sans faire référence à une quelconque adresse mémoire. À la fin des années 1960, elle a été diffusée dans le public comme interface utilisateur avec les calculatrices de bureau de Hewlett-Packard.

Exemple: Voici une expression algébrique

En notation in-fixée (celle que nous utilisons la plupart du temps) : ((3 + 4) - 2)^3

En notation post-fixée ( NPI) cela s’écrit : 3 4 + 2 - 3 ^

Evaluer (=calculer) le résultat à la main les expressions :

3 4 \* 2 - 3 + donne …….. ; 3 4 2 - + 3 \* donne ……… ; 3 4 2 3 - + \* donne …………

L’objectif est d’écrire un programme pour évaluer des expressions écrites en notation polonaise inverse.

Les expressions en NPI seront représentées par des tableaux contenant des entiers et des caractères.

l’expression précédente sera représentée par le tableau : tab=[3,4,'+',2,'\*',3,'^']

Principe du programme: Une pile vide est créée, le tableau est parcouru de gauche à droite

* Chaque nombre rencontré est empilé.
* Si l’élément rencontré est un opérateur, on dépile le sommet et le sous-sommet puis on empile le résultat du calcul (sous-sommet "opération" sommet )
* Si l’élément rencontré est une fonction (comme : p ), on dépile le sommet et on calcule la valeur de la fonction pour le sommet, puis on empile le résultat

Coder ce programme en Python, et évaluer différentes expressions.

On entrera une chaine de caractères avec des nombres et les opérateurs séparés des espaces et cette chaine sera convertie en tableau avec la commande : tab=chaine.split(sep=" ")

**Exercice 3 : Résoudre un labyrinthe**

|  |  |
| --- | --- |
|  | On représentera le labyrinthe à l’aide d’un tableau ( liste de listes ).  Le labyrinthe ( 0 pour les murs, 1 pour les cases blanches )  laby=[[0,1,0,0,0,0],  [0,1,1,1,1,0],  [0,1,0,1,0,0],  [0,1,0,1,1,0],  [0,1,1,0,1,0],  [0,0,0,0,1,0]] |

On repère une case par ses coordonnées (i ; j), on y accède dans le tableau par laby[i][j]

L’entrée dans cet exemple se fait par la case (0,1) et la sortie par la case (5,4)

Compléter le code pour obtenir le nombre de lignes et le nombre de colonnes de ce tableau:

lignes=len(……….. ) ; colonnes=len( ………………)

|  |  |
| --- | --- |
| Question 1:  Nous avons besoin de créer une copie de ce tableau pour pouvoir travailler dessus sans modifier le tableau initial laby  Nous utiliserons la bibliothèque copy pour cela en utilisant la fonction **deepcopy**  Tester ce code | from copy import deepcopy  laby=[[0,1,0,0,0,0], [0,1,1,1,1,0],[0,1,0,1,0,0],  [0,1,0,1,1,0],[0,1,1,0,1,0],[0,0,0,0,1,0]]  T=deepcopy(laby)  T[3][2]='hello'  for ligne in laby:  print(ligne)  print("----------------")  for ligne in T:  print(ligne) |

**Objectif : Ecrire un programme qui détermine s’il existe un chemin de l’entrée vers la sortie en se déplaçant vers le haut, le bas, la gauche ou la droite (mais pas en diagonale).**

|  |  |
| --- | --- |
| Etape 1 :  Soit la fonction **voisins(T,case)** qui prend en paramètre un tableau T et un tuple case : | Question 2 : Que retourne cette fonction ?  Question 3 :  Que va retourner l’instruction : print(voisins(laby,(0,1)) |

|  |  |  |
| --- | --- | --- |
| Étape 2 : parcours du labyrinthe :  L’idée est de parcourir le labyrinthe depuis l’entrée, en utilisant une pile pour stocker le chemin, pour pouvoir dépiler lorsque le chemin n’aboutit pas et redémarrer sur une autre voie.  On dispose d’une copie de notre tableau et d’une pile vide | |  |
| Le processus |  | |
| Et ainsi de suite jusqu’à ce que l’on tombe sur une impasse.. |  | |
| On examine la dernière case dépilée... |  | |
| On poursuit l’exploration... |  | |
| Et ainsi de suite jusqu’à la sortie : |  | |

Question 4 : Compléter l’algorithme :

fonction parcours( laby , entrée, sortie)

T <-- une copie de …..

p <-- Pile()

v <-- entrée

on met à ……….. la valeur de la case……... dans ……..

recherche <--True

tant que recherche est vrai

vois <-- voisins (…....,…...)

si la liste ……. est vide

si la pile vide --> renvoyer False

sinon v <--.........................................................................................

sinon

on ………...... p avec ....

v <-- le 1er ...................

on met à …... la valeur de la case….... dans ...

si v = ......

on ……........ p avec......

recherche <--.......

return ....

Question 5 : Si au cours de l’exécution la pile se trouve vide, que cela signifie-t-il?

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Question 6 : Réaliser le programme en python (mettre en évidence le chemin) et tester le avec d’autres labyrinthes

Correction exercice 3

fonction parcours( laby , entrée, sortie)

T <-- une copie de .laby.......

p <-- Pile()

v <-- entrée

on met à -1. la valeur de la case…v…... dans …T…..

recherche <--True

tant que recherche est vrai

vois <-- voisins (laby,v[0],v[1])

si la liste …vois…. est vide

si la pile vide --> renvoyer False

sinon v <-- sommet de la pile p obtenu en dépilant p

sinon

on …empile p avec .v.

v <-- le 1er .voisins.............

on met à -1 la valeur de la case v. dans T..

si v = sortie

on …empile..... p avec v

recherche <-- False

return p

**Correction exercice 1 :** Écrire en pseudo code l’algorithme du a)

fonction vérification ( expression : une chaine de caractères) ->booléen

debut

créer une pile vide p

pour caractère dans expression faire

si caractère =’(‘ alors

on emplile ‘(‘ dans p

fin si

si caractère =’)’ alors

si la pile est vide alors

on retourne faux

sinon

on depile

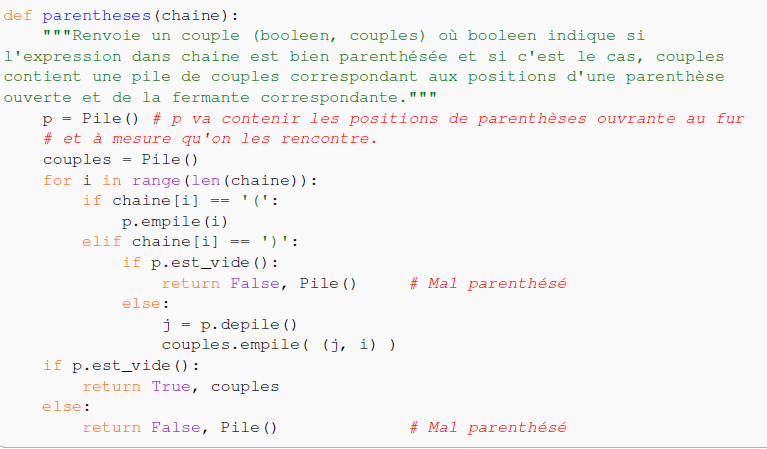
fin si

fin pour

on retourne le booléen pilevide

fin

**b)**



**c)**

