Structures de données : les listes, les piles et les files:

De nombreux algorithmes "classiques" manipulent des structures de données plus complexes que des simples nombres (nous aurons l'occasion d'en voir plusieurs cette année). Nous allons ici voir quelques-unes de ces structures de données. Nous allons commencer par des types de structures relativement simples : les listes, les piles et les files. Ces trois types de structures sont qualifiés de linéaires.

1. **Les listes:**
   Une liste est une structure de données permettant de regrouper des données. Une liste L est composée de 2 parties : sa tête (souvent noté car), qui correspond au dernier élément ajouté à la liste, et sa queue (souvent noté cdr) qui correspond au reste de la liste. Le langage de programmation Lisp (inventé par John McCarthy en 1958) a été un des premiers langages de programmation à introduire cette notion de liste (Lisp signifie "list processing"). Voici les opérations qui peuvent être effectuées sur une liste:

   - créer une liste vide (L=vide() on a créé une liste L vide)
   - tester si une liste est vide (estVide(L) renvoie vrai si la liste L est vide)
   - ajouter un élément en tête de liste (ajouteEnTete (x,L) avec L une liste et x l'élément à ajouter)
   - supprimer la tête x d'une liste L et renvoyer cette tête x (supprEnTete(L))
   - Compter le nombre d'éléments présents dans une liste (compte(L) renvoie le nombre d'éléments présents dans la liste L)

   La fonction cons permet d'obtenir une nouvelle liste à partir d'une liste et d'un élément (L1 = cons(x,L)). Il est possible "d'enchaîner" les cons et d'obtenir ce genre de structure : cons(x, cons(y, cons(z,L)))

   Exemple: voici une série d'instructions (les instructions ci-dessous s'enchaînent):

   - L=vide() $\longrightarrow$ on a créé une liste vide
   - estVide(L) $\longrightarrow$ renvoie vrai
   - ajoutEnTete(3,L) $\longrightarrow$ La liste L contient maintenant l'élément 3
   - estVide(L) $\longrightarrow$ renvoie faux
   - ajoutEnTete(5,L) $\longrightarrow$ la tête de la liste L correspond à 5, la queue contient l'élément 3
   - ajoutEnTete(8,L) $\longrightarrow$ la tête de la liste L correspond à 8, la queue contient les éléments 3 et 5
   - t = supprEnTete(L) $\longrightarrow$ la variable t vaut 8, la tête de L correspond à 5 et la queue contient l'élément 3
   - L1 = vide()

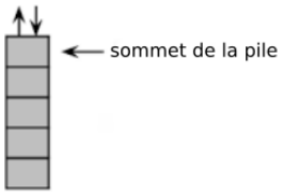
- L2 = cons(8, cons(5, cons(3, L1))) $\longrightarrow$ La tête de L2 correspond à 8 et la queue contient les éléments 3 et 5

Exercice 1: Voici une série d'instructions (les instructions ci-dessous s'enchaînent), expliquez ce qui se passe à chacune des étapes:

- L = vide()
- ajoutEnTete(10,L)
- ajoutEnTete(9,L)
- ajoutEnTete(7,L)
- L1 = vide()
- L2 = cons(5, cons(4, cons(3, cons (2, cons(1, cons(0,L1))))))

2. **Les piles:**

On retrouve dans les piles une partie des propriétés vues sur les listes. Dans les piles, il est uniquement possible de manipuler le dernier élément introduit dans la pile. On prend souvent l'analogie avec une pile d'assiettes : dans une pile d'assiettes la seule assiette directement accessible et la dernière assiette qui a été déposée sur la pile. Les piles sont basées sur le principe



LIFO (Last In First Out : le dernier rentré sera le premier à sortir). On retrouve souvent ce principe LIFO en informatique. Voici les opérations que l'on peut réaliser sur une pile:

- on peut savoir si une pile est vide (pile_vide?)
- on peut empiler un nouvel élément sur la pile (push)
- on peut récupérer l'élément au sommet de la pile tout en le supprimant. On dit que l'on dépile (pop)
- on peut accéder à l'élément situé au sommet de la pile sans le supprimer de la pile (sommet)
- on peut connaitre le nombre d'éléments présents dans la pile (taille)

Exemple: Soit une pile P composée des éléments suivants : 12, 14, 8, 7, 19 et 22 (le sommet de la pile est 22) Pour chaque exemple ci-dessous on repart de la pile d'origine:

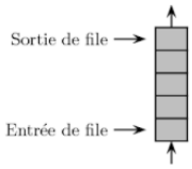
- pop(P) renvoie 22 et la pile P est maintenant composée des éléments suivants : 12, 14, 8, 7 et 19 (le sommet de la pile est 19)
- push(P,42) la pile P est maintenant composée des éléments suivants : 12, 14, 8, 7, 19, 22 et 42
- sommet(P) renvoie 22, la pile P n'est pas modifiée
- si on applique pop(P) 6 fois de suite, pile_vide?(P) renvoie vrai

- Après avoir appliqué pop(P) une fois, taille(P) renvoie 5

Exercice 2: Soit une pile P composée des éléments suivants : 15, 11, 32, 45 et 67 (le sommet de la pile est 67). Quel est l'effet de l'instruction pop(P)?

3. **Les files:**

Comme les piles, les files ont des points communs avec les listes. Différences majeures : dans une file on ajoute des éléments à une extrémité de la file et on supprime des éléments à l'autre extrémité. On prend souvent l'analogie de la file d'attente devant un magasin pour décrire une file de données. Les files sont basées sur le principe FIFO (First In First Out : le premier qui est



rentré sera le premier à sortir. Ici aussi, on retrouve souvent ce principe FIFO en informatique. Voici les opérations que l'on peut réaliser sur une file:

- on peut savoir si une file est vide (file_vide?)
- on peut ajouter un nouvel élément à la file (ajout)
- on peut récupérer l'élément situé en bout de file tout en le supprimant (retire)
- on peut accéder à l'élément situé en bout de file sans le supprimer de la file (premier)
- on peut connaitre le nombre d'éléments présents dans la file (taille)

Exemples: Soit une file F composée des éléments suivants : 12, 14, 8, 7, 19 et 22 (le premier élément rentré dans la file est 22 ; le dernier élément rentré dans la file est 12). Pour chaque exemple ci-dessous on repart de la file d'origine:

- ajout(F,42) la file F est maintenant composée des éléments suivants : 42, 12, 14, 8, 7, 19 et 22 (le premier élément rentré dans la file est 22 ; le dernier élément rentré dans la file est 42)
- retire(F) la file F est maintenant composée des éléments suivants : 12, 14, 8, 7, et 19 (le premier élément rentré dans la file est 19 ; le dernier élément rentré dans la file est 12)
- premier(F) renvoie 22, la file F n'est pas modifiée
- si on applique retire(F) 6 fois de suite, pile_vide?(F) renvoie vrai
- Après avoir appliqué retire(F) une fois, taille(F) renvoie 5

Exercice 3: Soit une file F composée des éléments suivants : 1, 12, 24, 17, 21 et 72 (le premier élément rentré dans la file est 72 ; le dernier élément rentré dans la file est 1). Quel est l'effet de l'instruction ajout(F,25)

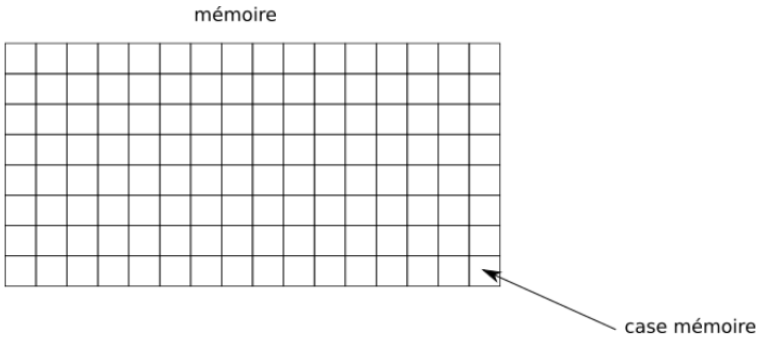
4. **Types abstraits et représentation concrète des données:**

Nous avons évoqué ci-dessus la manipulation des types de données (liste, pile et file) par des algorithmes, mais, au-delà de la beauté intellectuelle de réfléchir sur ces algorithmes, le but de l'opération est souvent, à un moment ou un autre, de "traduire" ces algorithmes dans un langage compréhensible pour un ordinateur (Python, Java, C,...). On dit alors que l'on implémente un algorithme. Il est donc aussi nécessaire d'implémenter les types de données comme les listes, les piles ou les files afin qu'ils soient utilisables par les ordinateurs. Les listes, les piles ou les files sont des "vues de l'esprit" présent uniquement dans la tête des informaticiens, on dit que ce sont des types abstraits de données (ou plus simplement des types abstraits). L'implémentation de ces types abstrait, afin qu'ils soient utilisables par une machine, est loin d'être une chose triviale. L'implémentation d'un type de données dépend du langage de programmation. Il faut, quel que soit le langage utilisé, que le programmeur retrouve les fonctions qui ont été définies pour le type abstrait (pour les listes, les piles et les files cela correspond aux fonctions définies ci-dessus). Certains types abstraits ne sont pas forcément implémentés dans un langage donné, si le programmeur veut utiliser ce type abstrait, il faudra qu'il le programme par lui-même en utilisant les "outils" fournis par son langage de programmation.

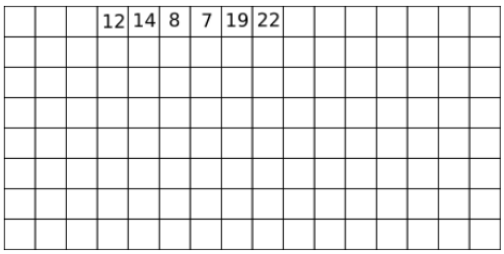
Pour implémenter les listes (ou les piles et les files), beaucoup de langages de programmation utilisent 2 structures: les tableaux et les listes chaînées.

4.1 **Les tableaux:**

Un tableau est une suite contiguë de cases mémoires (les adresses des cases mémoire se suivent):



Le système réserve une plage d'adresse mémoire afin de stocker des éléments.



La taille d'un tableau est fixe : une fois que l'on a défini le nombre d'éléments que le tableau peut accueillir, il n'est pas possible modifier sa taille. Si l'on veut insérer une donnée, on

doit créer un nouveau tableau plus grand et déplacer les éléments du premier tableau vers le second tout en ajoutant la donnée au bon endroit!

Dans certains langages de programmation, on trouve une version "évoluée" des tableaux : les tableaux dynamiques. Les tableaux dynamiques ont une taille qui peut varier. Il est donc relativement simple d'insérer des éléments dans le tableau. Ce type de tableaux permet d'implémenter facilement le type abstrait liste (de même pour les piles et les files)

À noter que les "listes Python" `https://docs.python.org/fr/3/tutorial/datastructures.html` sont des tableaux dynamiques. Attention de ne pas confondre avec le type abstrait liste défini ci-dessus, ce sont de "faux amis".

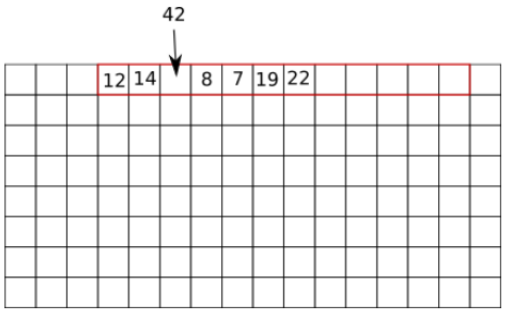
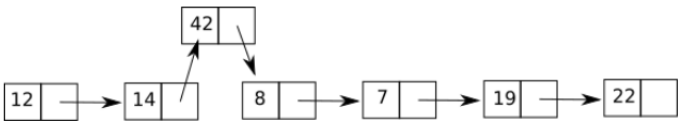


tableau dynamique

4.2 **Les listes chaînées:** Autre type de structure que l'on rencontre souvent et qui permet d'implémenter les listes, les piles et les files: les listes chaînées. Dans une liste chaînée, à chaque élément de la liste on associe 2 cases mémoire: la première case contient l'élément et la deuxième contient l'adresse mémoire de l'élément suivant. Il est relativement facile

d'insérer un élément dans une liste chaînée: Il est aussi possible d'implémenter les types



abstraits en utilisant des structures plus complexes que les tableaux et les listes chaînées. Par exemple, en Python, il est possible d'utiliser les tuples pour implémenter le type abstrait liste:

Exercice 4: Étudiez attentivement les fonctions suivantes et dire ce que chacune fait.

```
def vide():
    return None
def cons(x,L):
    return(x,L)
def ajouteEnTete(L,x):
    return cons(x,L)
def supprEnTete(L):
    return (L[0],L[1])
def estVide(L):
    return L is None
def compte(L):
    if estVide(L):
        return 0
    return 1 + compte(L[1])
```

Exercice 5: Après avoir saisi et exécuté le programme étudié dans l'exercice 4, tapez successivement les commandes suivantes dans une console Python:

- L = vide()
- estVide(L)
- L = cons(5, cons(4, cons(3, cons(2, cons(1, cons(0,L))))))
- estVide(L)
- compte(L)
- L = ajouteEnTete(L,6)
- compte(L)
- x, L=supprEnTete(L)
- x
- compte(L)
- x, L=supprEnTete(L)
- x
- compte(L)

Exercice 6: Étudiez l'implémentation des piles et des files en Python en vous aidant de la documentation officielle `https://docs.python.org/fr/3/tutorial/datastructures.html` (5.1.1 et 5.1.2). Pour ceci, vous testerez **toutes les opérations possibles sur les piles et les files de ce document** avec python.