Les structures de données

Désigne l'ensemble des objets permettant de stocker les données en mémoire et de les gérer de façon dynamique. Selon les besoins on peut stocker au sein d'une structure unique des données hétérogènes (mélange de types de données) ou non (un seul type).

Les structures livrées avec les compilateurs sont en général peu nombreuses
Tableaux (= matrices et vecteurs)
Structures (propre au C ; données hétérogènes quelconques)
Chaines de caractères (type str du Pascal ou Python)

Mais certains langages offrent :
Listes, Ensembles, **Tuples**, **Dictionnaires** ,…(Python)

On construit, à partir des structures simples, des méta-structures grâce à des variables particulières appelées **pointeurs**:
Listes qui permettent selon les règles d'insertion /délétion de constituer des **Piles** ou des **Files**, des **Arbres**, des **Graphes**, des **Tables de hachage**…

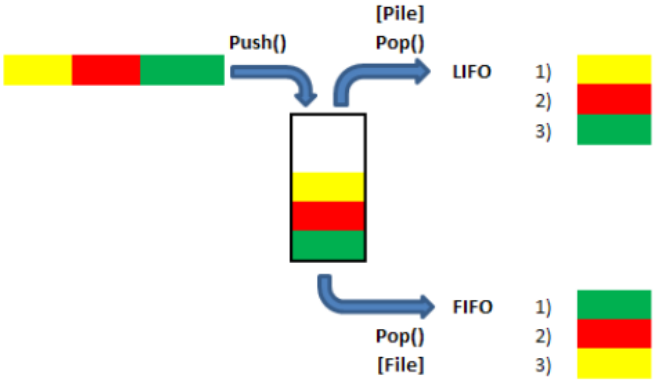
Il existe chez les langages Orientés-Objets des structures très particulières appelées **Classes** qui ressemblent aux structures du C mais qui de plus encapsulent des fonctions (C++, C#, Pascal, Java, Python, VBA …). Il existe au moins 650 langages de programmation. Voici une liste : http://www.scriptol.fr/programmation/liste-langages-programmation.php#language-a Les plus populaires : https://pypl.github.io/PYPL.html?country=FR

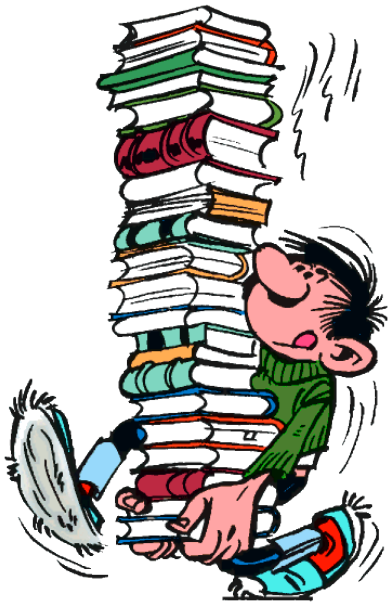
La connaissance et la manipulation des structures de données est capitale pour le programmeur car leur utilisation retenti sur :

1. La clarté, la lisibilité et la facilité de maintenance du code.
2. La sécurité des données à l'intérieur des applications
3. La vitesse d'exécution du code

Piles et Files

- Dans une pile, l'élément supprimé est le dernier inséré
  - LIFO – Last In First Out – Dernier arrivé premier parti
- Dans une file, l'élément supprimé est le plus ancien
  - FIFO – First In First Out – Premier arrivé premier parti





Piles

Principales opérations

Empiler (= rajouter un élément) *Push(S,x)*

Dépiler (= enlever un élément) *Pop(S,x)*

Autres opérations

Search(S,k) : chercher k dans S, retourne un pointeur sur k. *Minimum(S)*, *Maximum(S)*, *Successeur(S, k)*, *Prédécesseur(S, k)*

Le plus simple est d'utiliser **un tableau** (entiers, réels, chaines de caractères…), c'est-à-dire une matrice à une ligne (= vecteur) :

- Il possède $(n + 1)$ éléments car son premier élément contient l'indice du dernier élément inséré, c.à.d. le nombres d'éléments *n*.
- En `T[1+1]` on a l'élément le plus ancien (base de la pile)
- En T[[T[1] + 1] l'élément le plus récemment ajouté.

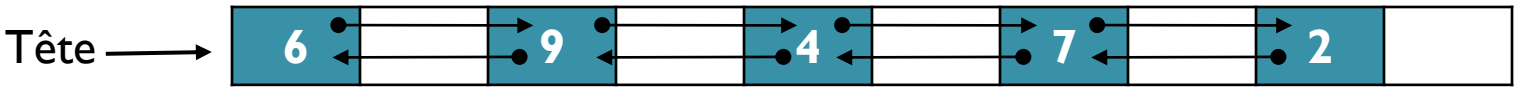
1	2	3	4	5	6	7	8
4	32	45	12	23			

T[1] = 4 est le sommet, donc n= 4
T[4+1]=23 est le dernier élément

1	2	3	4	5	6	7	8
6	32	45	12	23	12	5	

T[1] = 6 Le sommet
T[6+1] = 5 n = 6

Alternativement on peut implémenter une pile en utilisant **une liste**.
Ici l'exemple d'une liste doublement chainée d'entiers.



La liste est composée d'éléments dont chacun possède un attribut (ici un entier) et deux pointeurs appelés *prédécesseur* et *successeur*, chacun **pointant respectivement sur l'adresse** de l'élément précédent et du suivant.

Ce qui fait que la liste n'est pas forcément stockée de façon contigüe en mémoire. Autre avantage, il est possible de supprimer un élément quelconque de la liste en raboutant les 2 parties dissociées en faisant correctement pointer les pointeurs sur les bonnes adresses… Mais pour les piles, on aura pas besoin de telles opérations.

Cependant la création et la gestion est simplifiée par rapport au tableaux, **car Scilab a prévu des procédures automatiques**, transparentes pour le programmeur.

Files

La file fait penser à un tuyau (ou un couloir) dans lequel le premier élément inséré, sera le premier à sortir (par l'autre bout). D'où : FIFO.

1	2	3	4	5	6
11	15	8	9	12	14

Indice = ordre chronologique d'insertion

Délétion (11 est le plus ancien)

Insertion (12 est le plus récent)

	1	2	3	4	5
t_1	A				
t_2	B	A			
t_3	C	B	A		
t_4	D	C	B	A	
t_5	D	C	B		

L'insertion se fait en tête de tableau avec décalage vers le bas.
La délétion se fait en fin de tableau.

Si la file est importante et contient beaucoup de données dans chaque élément, l'insertion peut être couteuse en temps qui augmente linéairement avec *n*.

Pour la construction d'une file et l'implémentation des fonctions qui la gèrent, il est plus aisé d'utiliser une liste !

Notion de pointeur

Un pointeur est une variable d'un type particulier en ce qu'elle contient non pas une valeur (entier, réel, caractère…) mais une adresse mémoire.

Par exemple dans la figure ci-contre le pointeur a (adresse 8) contient l'adresse de b à laquelle on trouve la valeur qui nous intéresse : la valeur 17 en hexadécimal est contenue à l'adresse 1008 (en hexadécimal)) : a pointe sur b.

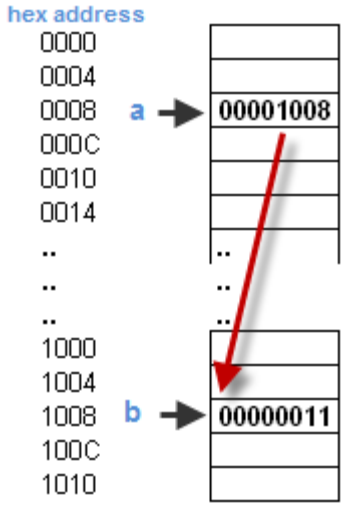


Fig. d'après wikipedia

Lorsqu'un pointeur pointe sur une variable complexe (tableau, liste, chaîne, fonction…), le fait de l'incrémenter de 1 ($p = p+1$) ne le déplace pas d'une unité de mémoire (32 ou 64 bits) *mais de la taille de l'élément pointé*. On peut ainsi naviguer par sauts, directement en mémoire.

Autre avantage, le fait de passer un pointeur à une fonction est beaucoup plus efficace que de passer à la fonction le contenu de la variable pointée, qui n'est alors pas recopiée dans la pile mémoire de la fonction, d'où un gain énorme de temps d'exécution et d'occupation de la mémoire.

Tous les langages utilisent les pointeurs. Mais seuls certains permettent leur utilisation explicite (Assembleur, C, C++ , Pascal, Cobol, Ada, Fortran, PHP, …).
Les autres (Java, Python,…) effectuent des passages par valeur (copient la valeur de la variable et n'autorisent pas la modification de l'original). Quelques dispositions autorisent cependant la modification des champs d'une variable structurée.

Retour