UNIVERSITE VIRTUELLE DU SENEGAL

foo nekk foofu la

NDEYE MASSATA NDIAYE

ALGORITHME STRUCTURES DES DONNES

SEQUENCE 3 : Les pointeurs et les listes chaînées

Les variables déclarées avant l'exécution d'un programme pour ranger les valeurs nécessaires à ce programme sont des variables statiques. Toute variable manipulée dans un programme est stockée quelque part en mémoire centrale. La place réservée en mémoire à une variable statique ne change pas au cours de l'exécution du programme. Cela peut entraîner (i) un gaspillage de place si l'espace mémoire réservée est plus grande que celle qui est effectivement utilisée par le programme ou (ii) un manque place si l'espace mémoire réservé est trop petite (il faut alors que le programme prenne en charge le contrôle du débordement).

Pour retrouver une variable, il suffit donc de connaître son adresse. Pour des raisons de lisibilité, on désigne souvent les variables par des identificateurs et non par leurs adresses. C'est le compilateur qui fait alors le lien entre l'identificateur d'une variable et son adresse en mémoire. Parfois, il est très pratique de manipuler une variable par son adresse: c'est l'utilité des pointeurs.

Les variables dynamiques, les pointeurs vont permettre de (i) d'allouer de l'espace mémoire au fur et à mesure des besoins dans la limite de l'espace disponible et (ii) de libérer de l'espace non utilisé.

1. Adresse & valeur d'une variable et Pointeur

Une variable est caractérisée par :

- son adresse mémoire
- sa valeur

Il existe deux modes d'accès à une variable:

- adressage direct: accès au contenu de la variable par son nom
- adressage indirect: accès au contenu de la variable par un pointeur qui contient l'adresse de la variable

Définition: un pointeur est une variable dont la valeur est égale à l'adresse d'un autre objet.

Déclaration d'un pointeur en algorithmique:

nom_du_pointeur: *Type

nom_du_pointeur: identificateur dont la valeur est l'adresse d'un objet de type Type

Type: type de la variable pointée

Remarque: L'opérateur unaire d'indirection * permet d'accéder directement à la valeur de l'objet pointé. Ainsi si p est un pointeur vers un entier i, *p désigne la valeur de i.

Exemple:

```
Algorithme Exple_pointeur1
Var
    i : Entier
    p : *Entier
Début
    i←3
    p←&i//p contient adresse de i
    Ecrire("Le contenu de la case mémoire pointé par p est :", *p)
    p←5 //changement de i à travers p
    Ecrire("i =" , i , "*p =", *p)
fin
```

Exécution

- le contenu de la case mémoire pointé par p est 3
- les objets i et *p sont identiques. Ils ont la même valeur.
- Toute modification de *p modifie i.

2. Les pointeurs

La valeur d'un pointeur étant un entier. On peut lui appliquer un certain nombre d'opérateurs arithmétiques classiques. Les seules opérations arithmétiques valides sur les pointeurs sont:

- l'addition d'un entier à un pointeur.
- la soustraction d'un entier à un pointeur.
- la différence de deux pointeurs p et q pointant tous les deux vers des objets de même type. Le résultat est un entier dont la valeur est égale à (p-q)/Taille(type).

Attention ! ! : Notons que la somme de deux pointeurs n'est pas autorisée.

Exemple : Si k est un entier et p est un pointeur sur un objet de type type. L'expression "p+k" désigne un pointeur sur un objet de type type dont la valeur est égale à la valeur de p incrémentée de k*sizeof(type). Il en va de même pour la soustraction d'un entier à un pointeur et pour les opérateurs d'incrémentation et de décrémentation + + et - -.

Exemple 1 :

```
Algorithme Exemple1
Var
    i : Entier
    p1,p2 : *Entier
Debut
    i←3
    p1←&i
    p2←p1
    Ecrire("p1 =" , p1 , "et p2 =", p2)
Fin
```

NB: Les opérateurs de comparaison sont applicables aux pointeurs à condition de comparer des pointeurs qui pointent vers des objets de même type.

3. Allocation dynamique

Lorsque l'on déclare une variable caractère, entier, réel, etc. Un nombre de cases mémoire bien défini est réservé pour cette variable. Il n'en est pas de même avec les pointeurs. Avant de manipuler un pointeur, il faut l'initialiser soit par une allocation dynamique soit par une affectation d'adresse p←&i. Sinon, par défaut la valeur du pointeur est égale à une constante symbolique notée NULL.

On peut initialiser un pointeur p par une affectation sur p. Mais il faut d'abord réserver à p un espace mémoire de taille adéquate. L'adresse de cet espace m´emoire sera la valeur de p. Cette opération consistant à réserver un espace mémoire pour stocker l'objet pointé s'appelle allocation dynamique. Elle se fait par la fonction "Allouer". Sa syntaxe est la suivante :

Ptr$\leftarrow$Allouer(nbr_objets)

Cette fonction retourne un pointeur pointant vers "nbr objets" objets.

Exemples : la fonction Allouer

Exemple 1 :

```
Algorithme Exemple1
Var
    pi,pj : *Entier
```

```
    pr : Réel
    pc : Caractère
Début
    //réserver 10 octets mémoire, soit la place pour 10 caractères
    pc←Allouer(10)
    //réserver un espace pour 4 entiers
    pi←Allouer(4)
    //réserver un espace pour 6 réels
    pr←Allouer(6)
    //réserver un espace pour 1 entiers
    pj←Allouer()  // Par défaut le nombre d'objet est 1
Fin
```

Remarque 1

Dans le programme suivant:

Algorithme Remarque1

Var

```
    i : Entier
    p : *Entier
Debut
    p←&i
Fin
```

i et *p sont identiques et on n'a pas besoin d'allocation dynamique puisque l'espace memoire à l'adresse &i est deja reserve pour un entier. Enfin, lorsque l'on n'a plus besoin de l'espace memoire allouee dynamiquement c'est-à-dire quand on n'utilise plus le pointeur p, il faut libérer cette place en mémoire. Ceci se fait à l'aide de l'instruction Libérer qui a pour syntaxe :

```
    Liberer(nom_du_pointeur)
```

Liberation de la mémoire : la fonction Liberer

Algorithme Exemple1

Var

```
    pi : *Entier
    pr : Réel
Début
```

```
//réserver un espace pour 4 entiers
pi←Allouer(4)
//réserver un espace pour 6 réels
pr←Allouer(6)
...
//libèrer la place précédemment réservée pour pi
Libérer(pi)
//libèrer la place précédemment réservée pour pr
Libérer(pr)
Fin
```

4. Pointeurs et tableaux

Tout tableau est un pointeur constant. Dans la déclaration :

```
tab : Tableau[1..10]d'entier
p : *Entier
p←tab// ou p←&tab[0] adresse du premier élément
```

Les écritures suivantes sont équivalentes : tab[3] ou p[3] ou *(p+3) tab est un pointeur constant non modifiable dont la valeur est l'adresse du premier ´el´ement du tableau. Autrement dit tab a pour valeur &tab[0]. On peut donc utiliser un pointeur initialisé à tab pour parcourir les éléments du tableau.

Exemple :

```
tab : Tableau[1..5]d'entier
p : *Entier i : Entier ...
p←tab
Pour i de 1 à 5 faire
Ecrire(*p)
p←p+1
Fin Pour
```

Qui est équivalent à :

```
tab : Tableau[1..5]d'entier
p : *Entier i : Entier ...
p←tab
```

```
Pour i de 1 à 5 faire
        Ecrire(p[i])
Fin Pour
```

Qui est équivalent à :

```
tab : Tableau[1..5]d'entier
p : *Entier ...
p←tab
Pour p de tab à tab+4 faire
        Ecrire(*p)
Fin Pour
```

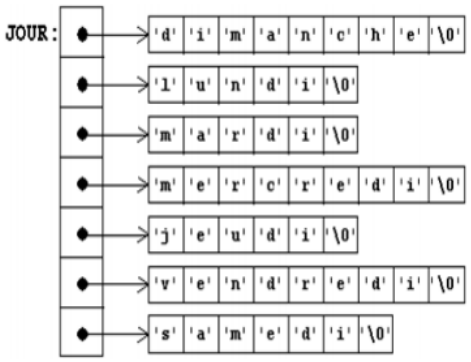
5. Tableau de pointeurs

La déclaration d'un tableau de pointeur se fait comme suit :

```
tab : Tableau[1..N]de *Type
```

C'est la déclaration d'un tableau **NomTableau** de **N** pointeurs sur des données du type **Type.**

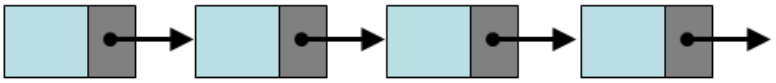
Exemple:



6. Les listes chaînées

1 Définition

Une liste chaînée est une succession de maillons, dont le dernier pointe vers une adresse invalide (NULL); voici une représentation possible :



Syntaxe:

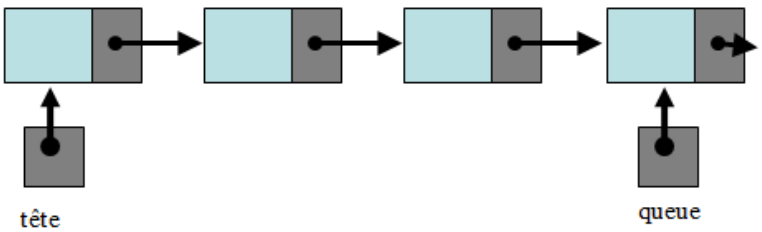
Un maillon aura la structure suivante:

```
STRUCTURE MAILLON
{
    Element : type
    Suivant :^MAILLON /*Pointeur vers le maillon suivant */
}
```

Soit m de type maillon, pour accéder au maillon suivant : m→suivant

Une liste chaînée est caractérisée par un pointeur **tete** (ou *premier*) vers le premier élément et un pointeur **queue** (ou *dernier*) vers le dernier élément de la liste.

```
STRUCTURE LISTE
{
    Tete : ^ MAILLON
    Queue: ^ MAILLON
}
```



2 .Algorithmes de manipulation de listes chaînées

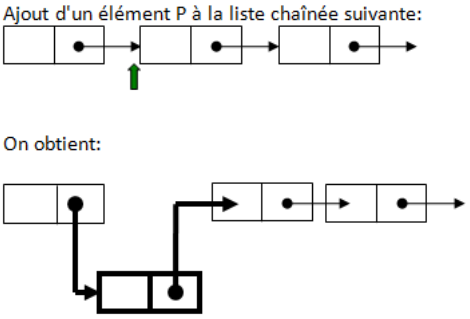
➔ Création d'une liste vide

```
Procedure viderListe(var  L: Liste);
Debut
    L:=nil
Fin
```

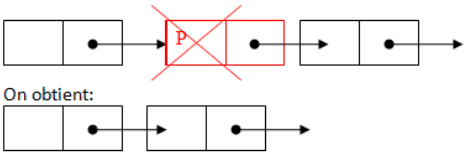
➔ Tête de la liste vide

```
Fonction estVide( L: Liste):booleen;
Debut
    estVide:=(L=nil);
Fin
```

➔ Ajout en tête de liste



➔ Suppression d'un élément



Références

1. http://www.fsg.rnu.tn/imgsite/cours/CH2.pdf
2. http://www.est-usmba.ac.ma/ALGORITHME/co/module_ALGORITHME_28.html
3. https://pages.lip6.fr/Souheib.Baarir/Cours-C/cours/Pointeurs/Pointeurs.pdf
4. http://miage.univ-nantes.fr/miage/DVD-MIAGEv2/Algo_files/DVDMIAGE_Algo_Chapitre_09_Pointeurs.pdf