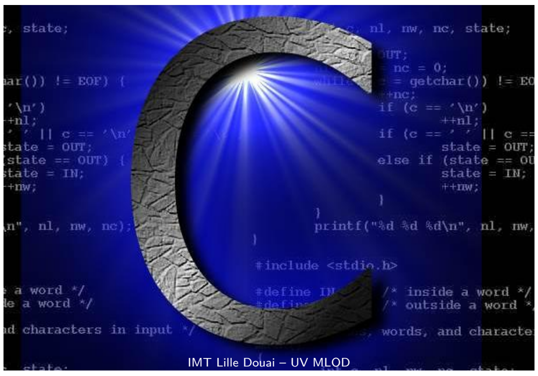
Algorithmique et langage C
— Structures de données dynamiques linéaires —



IMT Lille Douai – UV MLOD

Luc Fabresse
luc.fabresse@imt-nord-europe.fr

version 1.2

Plan

Plan

Pourquoi des structures de données dynamiques ?

Les SSD permettent de représenter des données :

- dont on ne connaît pas a priori la taille
- dont la taille est variable selon les cas ou au cours du temps

Exemple

Lecture et stockage d'une suite de nombres dont on ne connaît pas la quantité de nombres à lire

Solution 1 : définir un tableau surdimensionné

Risques :

- sous-estimation du nombre maximum (tableau trop petit)
- perte de place mémoire (tableau trop grand)

Solution 2

Utilisation d'une structure dynamique qui s'agrandit au fur et à mesure de la lecture des nombres

Structure de données dynamiques

Principe

La représentation physique (en mémoire) des données suit les évolutions de la structure :

- Attribution de la place en mémoire quand elle grandit
- Récupération de la place en mémoire quand elle diminue

Fonctionnement

En toute généralité, il existe au moins deux procédures standards d'acquisition et de libération d'espace mémoire : `reserve` et `libere`. Ces procédures d'allocation et de libération dynamique utilise une zone mémoire particulière appelée le *TAS*.

Plan

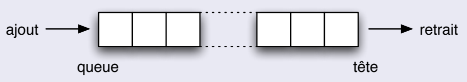
File d'attente

Définition

- Type des éléments homogène
- Les données sont ajoutées par l'intermédiaire d'un pointeur d'écriture appelé `Queue`
- Les données sont prélevées par l'intermédiaire d'un pointeur de lecture appelé `Tête`
- Accès FIFO (*first in*, *first out*), PAPS (premier arrivé, premier servi)

Exemple

- File d'attente des clients devant un guichet
- Les travaux en attente d'exécution dans un système de traitement par lots



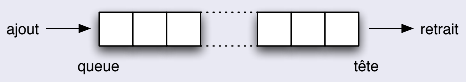
File d'attente : Spécification

Structure décomposable en deux parties :

- La tête qui corresponds au premier élément, celui qui peut être retiré
- Le corps qui comprend à tous les autres éléments

Les primitives d'accès à une file d'attente permettent de :

`tete`	Obtenir l'index (ou le pointeur) vers l'élément en tête
`queue`	Obtenir l'index (ou le pointeur) vers l'emplacement où sera inséré le prochain élément
`defiler`	Retirer l'élément de tête
`enfiler`	Ajouter un élément à la file
`vide`	Déterminer si la file est vide



File d'attente : algorithmes

Représentation avec un tableau :

la longueur maximale d'une file est $n-1$ avec une représentation par un tableau de taille n

`estPleine(f) : Booleen`

```
begin
    if queue(f) $=$ longueur(f) then
        return tete(f) $=$ 1;
    else
        return tete(f) $=$ queue(f) $+$ 1;
    end
end
```

`estVide(f) : Booleen`

```
begin
    return tete(f) $=$ queue(f) ;
end
```

File d'attente : algorithmes

Représentation avec un tableau :

la longueur maximale d'une file est $n-1$ avec une représentation par un tableau de taille n

`enfiler(f,x)`

```
begin
    f[queue(f)] $\leftarrow$ x;
    if queue(f) $=$ longueur(f) then
        queue(f) $\leftarrow$ 1;
    else
        queue(f) $\leftarrow$ queue(f) $+$ 1;
    end
end
```

`defiler(f) : T`

```
begin
    x $\leftarrow$ f[tete(f)] ;
    if tete(f) $=$ longueur(f) then
        tete(f) $\leftarrow$ 1;
    else
        tete(f) $\leftarrow$ tete(f) $+$ 1;
    end
end
```

Plan

Pile : spécification

Définition

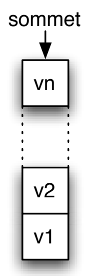
Une pile est une structure de données dynamique homogène à un seul point d'accès, les données sont ajoutées ou retranchées par l'intermédiaire d'une tête d'accès appelée `sommet` de la pile. La gestion d'une pile est dite LIFO (*Last In, First Out*).

Les opérations caractéristiques d'une pile sont :

sommet	Obtenir l'index (ou le pointeur) vers l'élément au dessus de la pile
empiler	Ajouter un élément
depiler	Retirer un élément

Exemples

- Pile d'assiettes
- Les appels en cascade de programmes. Il faut terminer d'abord le sous-programme appelé en dernier.



Pile : algorithmes

Représentation avec un tableau :

la longueur maximale d'une pile est n avec une représentation par un tableau de taille n

`estVide(p) : Booleen`

```
begin
    return sommet(p) $=$ 0;
end
```

`empiler(p,x)`

```
begin
    sommet(p) $\leftarrow$ sommet(p) $+$ 1;
    p[sommet(p)] $\leftarrow$ x;
end
```

`depiler(p) : T`

```
begin
    if estVide(p) then
        erreur;
    else
        sommet(p) $\leftarrow$ sommet(p) - 1;
        return p[sommet(p)+1];
    end
end
```

Plan

Liste chaînée

Definition

Une liste chaînée linéaire est un ensemble ordonné et extensible d'éléments de même type auxquels on accède séquentiellement, et où l'on peut ajouter ou retrancher un élément en n'importe quelle position.

Exemple

Un éditeur de texte représente un texte source comme une suite de lignes dans laquelle on peut effectuer des insertions et des retraits en n'importe quel point.

Liste chaînée : spécification

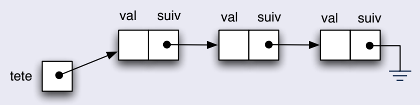
Structure

Une liste linéaire peut être décomposée en deux parties :

- le début (`tete`)
- le corps

Les primitives d'accès :

estVide	Teste la vacuité
tete	Fournit le premier élément de la liste
ajout	Ajouter un élément en tête de liste
supprimer	Supprimer un élément



Liste chaînée : exercice

Compléter le fichier `liste-chainee.c`

Liste chaînée : variations

Liste linéaire :

- Simplement chaînée
- Doublement chaînée
  - Chaque élément de la liste possède un lien de chaînage vers l'élément suivant et un lien de chaînage vers l'élément précédent
  - La liste peut être ainsi parcourue dans les deux sens

Liste circulaire :

Le premier élément est le suivant du dernier

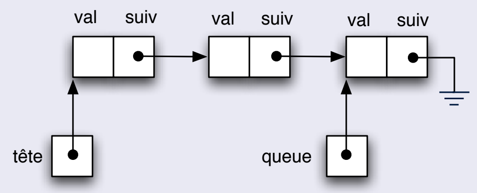
- Simplement chaînée
- Doublement chaînée

Plan

File d'attente

Représentation chaînée :

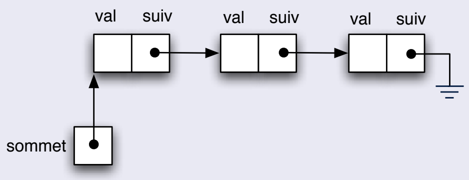
- Chaque *cellule* est un enregistrement contenant l'information utile et un pointeur sur la cellule suivante
- Il faut maintenir deux pointeurs d'accès : `tête` et `queue`



Pile

Représentation chaînée :

- Chaque *cellule* est un enregistrement contenant l'information utile et un pointeur sur la cellule suivante
- Il faut maintenir le pointeur d'accès : `sommet`



Plan

Place aux exercices !

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

[Brian Kernighan]