Vers plus de modularité en C

-Mieux structurer les programmes en rassemblant toutes les interfaces des opérations sur les listes dans le même fichier: listeint.h.

Cela permet de s'approcher des principes de la programmation orientée objet: ''rassembler les opérations manipulant la même structure de données dans le même paquet (classe en POO)''

```
typedef struct elem {int val;
                     struct elem * suivant;
                                  } element;
typedef element *listeint;

listeint ajoutert(listeint l, int v);
listeint ajouterq(listeint l, int v);
listeint supprimert (listeint l);
listeint supprimerq (listeint l);
….
```

Vers plus de modularité en C

- Compléter le fichier listeint.h en construisant un autre fichier listeint.c contenant le corps ou la réalisation des opérations

---

```
#include <listeint.h>
listeint ajoutert (listeint l,  int v)
{ liste temp ;
  temp =( element *) malloc(sizeof(element)) ;
  temp->valeur=p ;  temp->suivant=LP ;
  return temp ;
}
listeint ajouterq(listeint l, int v)
{…
…
}
listeint supprimert (listeint l)
{…
…
}
listeint supprimerq (listeint l)
{…
…
}
```

Structures de données

Les structures de données : Définition et objectif

- **Définition:** informations organisées qui peuvent être créées et manipulées par les programmes.

  programme = structures de données + algorithme

- **Objectif de l'étude des SD:** se donner les moyens permettant de concevoir les algorithmes en manipulant les données (les SD) à travers les opérations autorisées sur ces données et non à travers leurs représentations en mémoire (en terme de type de données concrets : pointeur, tableau,…)

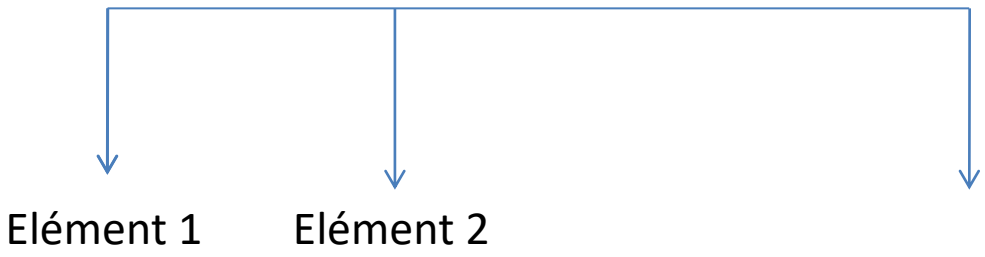
- **Comment:** définir les SD comme des types abstraits de données (des classes d'objets)

- **Type abstrait de données:** ensemble d'objets définis par les opérations permettant de les manipuler.

Type abstrait ou structure de données LISTE

- **Définition:** Une liste est une suite finie d'éléments de même type. L'accès au n^{ieme} élément d'une liste se fait en parcourant les n-1 éléments qui le précèdent. On dit que l'accès aux éléments de la liste se fait de manière séquentielle.

- **Représentation graphique :**



Type abstrait ou structure de données LISTE

D'un point de vue formel, une liste dont les éléments sont de type ELT peut se définir de la manière suivante :

- un ensemble de places P : à chaque place p de P est associé un élément de la liste.
- un ensemble de valeurs de type ELT
- une fonction Val qui associe à chaque place, la valeur qu'elle contient :

$$Val : P \rightarrow ELT$$

- une fonction Suc qui associe à chaque place de la liste, la place de l'élément suivant.

$$suc : P - \{dernier\} \rightarrow P - \{tête\}$$

Type abstrait LISTE: Opérations

On distingue 3 catégories d'opérations :

- **les constructeurs** : les opérations qui permettent de construire ou de simplifier la structure des objets du type.

  Exemple: ajoutert: LISTE x ELT → LISTE

- **les observateurs** : les opérations qui permettent de consulter l'état des objets

  Exemple: Vide : LISTE → Booléen

- **les fonctions d'accès** : se sont les opérations qui permettent d'extraire des informations en accédant aux objets du type.

  Exemple: Tete: LISTE → P

Type abstrait LISTE: Opérations

Type liste (ELT)

Opérations :
Constructeurs :
créer : → liste , *Construit un liste vide*
ajoutert : liste x ELT → liste , *ajoute un élément en tête de liste*
ajouterq : liste x ELT → liste , *ajoute un élément en queue de liste*
ajouter: liste x P X ELT → liste,
supprimert : liste → liste, supprime le premier élément
supprimerq : liste → liste, supprime le dernier élément
supprimer: liste x p → liste,
Observateurs et fonctions d'accès
vide : liste → booléen
tete : liste → P
dernier : liste → p

Fin Type

Algorithmes fondamentaux sur les listes

- **Algorithme de parcours**

  Il s'agit de parcourir une liste l pour appliquer un traitement à chacun de ses éléments. On note Trait (x); ce traitement.

x = tete(l)

Tantque *x /= indéfini* **faire**

Trait(x),

x=suc(x),

Fait

Algorithmes fondamentaux sur les listes

- **Algorithme de recherche**

  Il s'agit de trouver l'élément x d'une liste l qui vérifie une propriété donnée P(x).

trouve = faux
x = tete(l)
*T***antque** *trouve = faux* **et** *x /= indefini* **faire**
si *p(x) = vrai* ***alors*** *trouve=vrai*
sinon *x=suc(x)*
fsi
Fait

Représentation des listes: contiguë-Chaînée

- **Une fois l'algorithme écrit:** Comment définir (représenter) les types abstraits utilisés dans l'algorithme en termes de types de données (concrets) proposés par le langage de programmation cible (ici le langage C).

Objets abstrait + opérations associées ----------> Objets concrets + opérations associées

l**iste** **tableau, structure, pointeur**, etc…

Représentation des listes: contiguë

- **Représentation contiguë:**

  Elle consiste à représenter les listes par des tableaux. Par conséquent, les opérations sur les listes doivent être réalisées par des opérations sur les tableaux.

  De manière schématique, on peut représenter de manière contiguë une liste L dont le type des éléments est ELT de la manière suivante:

Niveaux abstrait	**Niveau concret**
L	Tableau [a … b] de type ELT
P	[a … b]
x	un élément de [a … b]
Val (x)	L[x]
Suc (x)	X+1
Tete (l)	a

Représentation contiguë des listes: Exemple

Algorithme abstrait

x = tete(l)
Tantque *x /= indéfini* **faire**
Trait(x),
x=suc(x),
Fait

Algorithme concret

x=a
Tantque x<b **faire**
Traiter(T[x])
x=x+1
Fait

Représentation des listes: Chaînée

- **Représentation chaînée:** Elle consiste à chaîner les éléments en conservant pour chaque élément, l'adresse de son suivant dans la liste.

  De manière schématique, on peut représenter de manière chaînée une liste L dont le type des éléments est ELT de la manière suivante:

Niveaux abstrait	**Niveau concret**
L	Pointeur sur un élément
Un élément	Une structure composée: -Un champ valeur contenant la valeur de l'élément (ELT) -Un champ suivant contenant l'adresse de l'élément suivant.
P	ensemble de pointeurs ou d'adresses
x	un pointeur sur un élément
Val (x)	X-> valeur
Suc (x)	x->suivant
Tete (L)	L

Représentation chaînée des listes: Exemple

Algorithme abstrait

x = tete(l)
Tantque *x /= indéfini* **faire**
Trait(x),
x=suc(x),
Fait

Algorithme concret

x=l
Tantque x/=NULL **faire**
Traiter(x)
x=x->suivant
Fait

Conclusion

La notion de type abstrait de données nous conduit à concevoir les programmes en deux étapes, chacune correspondant à un niveau d'abstraction donné :

- **Niveau logique (niveau abstrait): algorithme**

Les structures de données doivent être manipulées à travers les opérations qui leurs sont associées. Par exemple, pour les listes, les opérations sont ajoutert, ajouterq, …

- **Niveau physique (niveau concret): pseudo code ou programme**

Les structures de données du niveau logique sont représentées par des types concrets proposés par le langage de programmation cible, et les opérations sont représentées par des sous-programmes.