Programmation impérative

ENSIIE

Semestre 1 — 2018–19

Listes

Retour sur les ensembles

Jusqu'ici, ensembles statique

- Si trop petit $\rightsquigarrow$ problème
- Si trop grand $\rightsquigarrow$ gaspillage mémoire

Allocation mémoire par élément pendant l'exécution

- structures dynamiques

Listes

Vision abstraite :

- Soit vide
- Soit un élément et la liste qui suit
  - passage au suivant

Fonction ensemblistes

- test de vacuité
- ajout/retrait (en tête ou pas)
- appartenance
- concaténation (union)

Réalisation

Maillon :

- un élément
- un accès vers la suite de la liste

Liste = adresse d'un maillon

```
typedef struct maillon* list;

struct maillon {
  int val;
  list next;
};
```

Liste vide : adresse `NULL`

Itérations

Parcours typique d'une liste :

```
list l;
⋮
/*@ loop variant : longueur de la liste */
while (l != NULL) {
  /* travail avec l->val */
  ⋮
  l = l->next;
}
```

Propriétés

- FILO (premier entré = dernier sorti) ordre conservé
- Ajout en tête en temps constant
- Accès au i^{e} élément linéaire
- Appartenance linéaire
- Concaténation linéaire

Utilisations listes

Listes (concret) candidates pour ?

- piles
- Accès sommet en temps constant
- ...

Garder à l'esprit :

- allocation coûteuse
- faire le bon choix

Circularité

```
list l, last;
l = NULL;
add(42, &l);
last = l;
add(666, &l);
add(27, &l);
add(256, &l);
last->next = l->next;
```

$\Rightarrow$ Faire attention : constructeurs (type abstrait)