Introduction à la Programmation des Algorithmes

4.1. Langage C – Listes chaînées

François Fleuret
https://fleuret.org/11x001/

Il arrive fréquemment que l'on veuille définir une structure de données récursive.

Par exemple, on peut considérer qu'une liste d'entiers est:

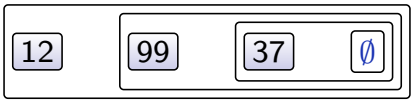
- soit vide,
- soit composée d'un entier et d'une autre liste.

La liste $\{12, 99, 37\}$ peut ainsi être vue comme composée de l'entier 12 et de la liste $\{99, 37\}$, elle même composée de 99 et de la liste $\{37\}$, elle même composée de 37 et $\{\}$.

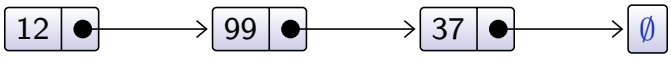
Il est donc naturel de vouloir définir une structure ayant des propriétés du même type qu'elle, comme une liste qui peut être composée d'une valeur et d'une liste.

En C, l'utilisation des pointeurs permet de représenter cette récursivité.

La liste $\{12, 99, 37\}$ peut être représentée par:



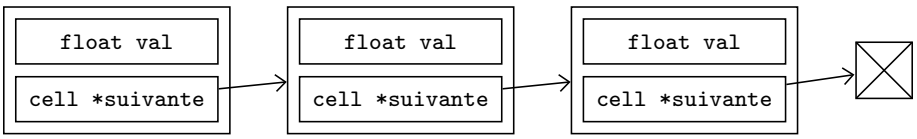
Mais en réalité la représentation de ce type d'objets en mémoire se fait toujours avec des références.



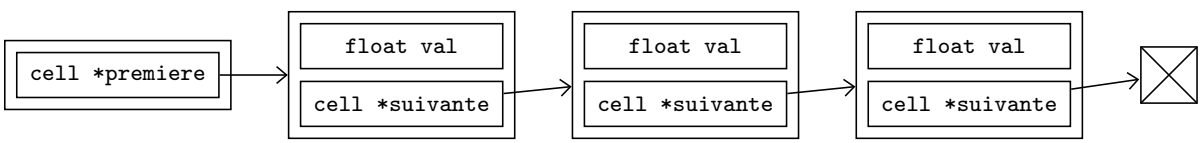
Exemple de liste chaînée

Une **liste chaînée** est constituée d'une succession de cellules, chacune contenant une valeur, et une référence vers la cellule qui lui succède, ou 0 s'il n'y en a pas.

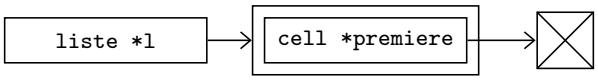
```c
struct Cell;

struct Cell {
  float val;
  struct Cell *suivante;
};

typedef struct Cell cell;
```



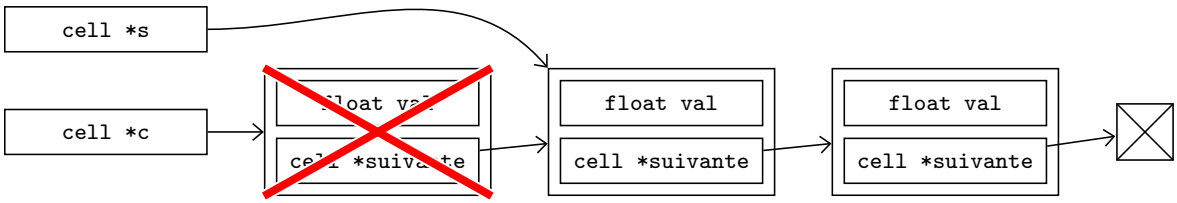

```c
struct Cell;

struct Cell {
  float val;
  struct Cell *suivante;
};

typedef struct Cell cell;

typedef struct {
  cell *premiere;
} liste;
```



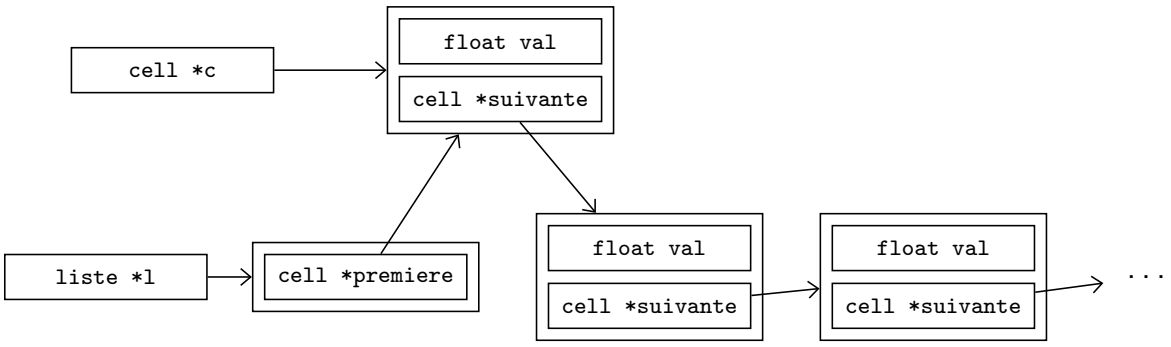
```c
liste *cree_liste() {
  liste *l = malloc(sizeof(liste));
  l->premiere = 0;
  return l;
}
```




```c
void detruit_cellules(cell *c) {
  if(c) {
    cell *s = c->suivante;
    free(c);
    detruit_cellules(s);
  }
}

void detruit_liste(liste *l) {
  detruit_cellules(l->premiere);
  free(l);
}
```



```c
void ajoute_valeur(liste *l, float x) {
  cell *c;
  c = malloc(sizeof(cell));
  c->val = x;
  c->suivante = l->premiere;
  l->premiere = c;
}
```




```c
liste *l = cree_liste();

for(int k = 0; k < 10; k++) ajoute_valeur(l, k);

for(cell *c = l->premiere; c; c = c->suivante)
  printf("%f\n", c->val);

detruit_liste(l);
```

affiche

```
9.000000
8.000000
7.000000
6.000000
5.000000
4.000000
3.000000
2.000000
1.000000
0.000000
```

```c
int longueur_cellules(cell *c) {
  if(c) {
    return 1 + longueur_cellules(c->suivante);
  } else return 0;
}

int longueur_liste(liste *l) {
  return longueur_cellules(l->premiere);
}
```



```c
liste *l = cree_liste();

for(int k = 0; k < 10; k++) ajoute_valeur(l, k);

printf("%d\n", longueur_liste(l));

detruit_liste(l);
```

affiche

10

```c
int longueur_cellules(cell *c, int l) {
  if(c) {
    return longueur_cellules(c->suivante, l + 1);
  } else return l;
}

int longueur_liste(liste *l) {
  return longueur_cellules(l->premiere, 0);
}
```



```c
int longueur_liste(liste *l) {
  int n = 0;
  for(cell *c = l->premiere; c; c = c->suivante) n++;
  return n;
}
```

```c
float somme_cellules(cell *c, float s) {
  if(c) {
    return somme_cellules(c->suivante, s + c->val);
  } else return s;
}

float somme_liste(liste *l) {
  return somme_cellules(l->premiere, 0);
}
```



```c
liste *fusionne_listes(liste *l, liste *m) {
  cell *d = l->premiere;
  if(d) {
    while(d->suivante) d = d->suivante;
    d->suivante = m->premiere;
    free(m);
    return l;
  } else {
    free(l);
    return m;
  }
}
```

```c
liste *l = cree_liste();

for(int k = 0; k < 10; k++) ajoute_valeur(l, k);

liste *l2 = cree_liste();

for(int k = 0; k < 10; k++) ajoute_valeur(l2, 1.0 / (1 + k));

printf("%f\n", somme_liste(l) + somme_liste(l2));

l = fusionne_listes(l, l2);
printf("%f\n", somme_liste(l));

detruit_liste(l);
```

affiche

```
47.928970
47.928967
```



```c
void retourne_liste(liste *l) {
  cell *d, *e, *f;
  d = l->premiere;
  e = 0;
  while(d) {
    f = d->suivante;
    d->suivante = e;
    e = d;
    d = f;
  }
  l->premiere = e;
}
```

```c
liste *l = cree_liste();

for(int k = 0; k < 10; k++) ajoute_valeur(l, k);

retourne_liste(l);

for(cell *c = l->premiere; c; c = c->suivante)
  printf("%f\n", c->val);

detruit_liste(l);
```

affiche

```
0.000000
1.000000
2.000000
3.000000
4.000000
5.000000
6.000000
7.000000
8.000000
9.000000
```


Une fonction pour enlever la dernière valeur ajoutée permet d'utiliser une liste comme une pile (le dernier entré est le premier sorti).

```c
float enleve_valeur(liste *l) {
  if(!l->premiere) abort();
  cell *c = l->premiere;
  float x = c->val;
  l->premiere = c->suivante;
  free(c);
  return x;
}
```

```c
liste *l = cree_liste();

for(int k = 0; k < 5; k++) ajoute_valeur(l, k);

printf("%f\n", enleve_valeur(l));
printf("%f\n", enleve_valeur(l));
printf("\n");

for(cell *c = l->premiere; c; c = c->suivante)
  printf("%f\n", c->val);

detruit_liste(l);
```

affiche

```
4.000000
3.000000

2.000000
1.000000
0.000000
```


Exemple de liste doublement chaînée

L'exemple de liste chaînée que nous venons de voir souffre de deux défauts: les valeurs sont mémorisées en ordre inverse, et la fusion de listes demande de parcourir l'une d'entre elles.

Une solution simple à ces deux problèmes est d'utiliser une **liste doublement chaînée**, dont chaque cellule possède une référence vers la cellule qui la précède, et une vers celle qui lui succède.


```c
struct Cell;

struct Cell {
  float val;
  struct Cell *suivante, *precedente;
};

typedef struct Cell cell;

typedef struct {
  cell *premiere, *derniere;
} liste;
```

```c
liste *cree_liste() {
  liste *l = malloc(sizeof(liste));
  l->premiere = 0;
  l->derniere = 0;
  return l;
}

void detruit_cellules(cell *c) {
  if(c) {
    cell *s = c->suivante;
    free(c);
    detruit_cellules(s);
  }
}

void detruit_liste(liste *l) {
  detruit_cellules(l->premiere);
  free(l);
}
```



```c
void ajoute_valeur(liste *l, float x) {
  cell *c;
  c = malloc(sizeof(cell));
  c->val = x;
  c->suivante = 0;
  c->precedente = l->derniere;
  if(l->derniere) {
    l->derniere->suivante = c;
  } else {
    l->premiere = c;
  }
  l->derniere = c;
}
```

```c
liste *fusionne_listes(liste *l, liste *m) {
  if(l->derniere) {
    l->derniere->suivante = m->premiere;
    if(m->premiere) m->premiere->precedente = l->derniere;
    l->derniere = m->derniere;
    free(m);
    return l;
  } else {
    free(l);
    return m;
  }
}
```


Une fonction qui enlève et retourne le premier élément de la liste permet d'utiliser un liste comme une file d'attente (le premier entré est le premier sorti).

```c
float enleve_valeur(liste *l) {
  if(!l->premiere) abort();

  cell *c = l->premiere;
  l->premiere = c->suivante;

  if(c->suivante) c->suivante->precedente = 0;

  if(c == l->derniere) l->derniere = 0;

  float x = c->val;
  free(c);

  return x;
}
```

```c
liste *l = cree_liste();

for(int k = 0; k < 5; k++) ajoute_valeur(l, k);

for(int i = 0; i < 3; i++) printf("%f\n", enleve_valeur(l));

printf("\n");

for(int k = 0; k < 2; k++) ajoute_valeur(l, 100 + k);

for(int i = 0; i < 4; i++) printf("%f\n", enleve_valeur(l));

detruit_liste(l);
```

affiche

```
0.000000
1.000000
2.000000

3.000000
4.000000
100.000000
101.000000
```