Introduction à la Programmation des Algorithmes

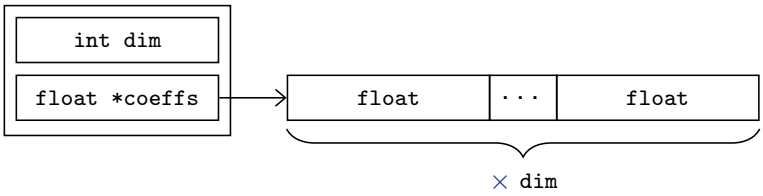
3.4. Langage C – Types et allocation dynamique

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

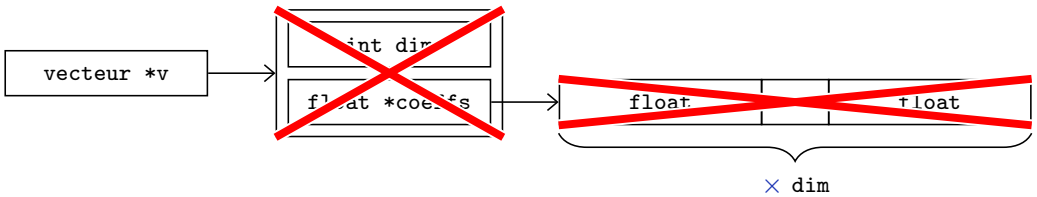
Comme nous l'avons vu, un pointeur ne fournit pas d'information sur la zone qu'il pointe, à part l'adresse du premier octet qui la compose.

La création de structures de données complexes se fait en combinant des pointeurs et des champs de données.

```c
typedef struct {
  int dim;
  float *coeffs;
} vecteur;
```




```c
vecteur *cree_vecteur(int dim) {
  vecteur *v;
  v = malloc(sizeof(vecteur));
  v->dim = dim;
  v->coeffs = malloc(sizeof(float) * dim);
  return v;
}

void detruit_vecteur(vecteur *v) {
  free(v->coeffs);
  free(v);
}
```



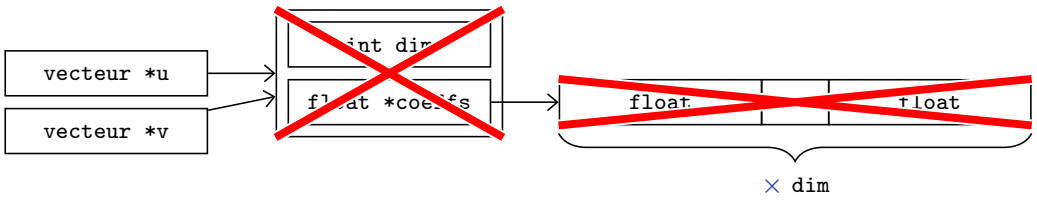
```c
void randomize_vecteur(vecteur *v) {
  for(int k = 0; k < v->dim; k++)
    v->coeffs[k] = 2 * (float) rand() / (float) RAND_MAX - 1;
}

float norme_l2_vecteur(vecteur *v) {
  float s = 0;
  for(int k = 0; k < v->dim; k++)
    s += v->coeffs[k] * v->coeffs[k];
  return sqrt(s);
}

int main(void) {
  vecteur *v = cree_vecteur(100);
  randomize_vecteur(v);
  printf("norme=%f\n", norme_l2_vecteur(v));
  detruit_vecteur(v);
  return 0;
}
```


L'opérateur d'affectation = se limite à faire une copie du pointeur sans dupliquer la variable pointée.

```c
int main(void) {
  vecteur *u, *v;
  v = cree_vecteur(100);
  u = v;
  detruit_vecteur(u);
  detruit_vecteur(v);
  return 0;
}
```

```
*** Error in `./a.out': double free or corruption
```

```c
int main(void) {
  vecteur *u, *v;
  u = cree_vecteur(100);
  v = u;
  detruit_vecteur(u);
  detruit_vecteur(v);
  return 0;
}
```



Il faut programmer une **copie** d'un objet pointé.

```c
vecteur *clone_vecteur(vecteur *v) {
  vecteur *u = cree_vecteur(v->dim);
  for(int k = 0; k < v->dim; k++)
    u->coeffs[k] = v->coeffs[k];
  return u;
}

int main(void) {
  vecteur *u, *v;
  v = cree_vecteur(100);
  randomize_vecteur(v);
  u = clone_vecteur(v);
  detruit_vecteur(u);
  detruit_vecteur(v);
  return 0;
}
```

```c
vecteur *somme_vecteur(vecteur *v, vecteur *w) {
  if(v->dim != w->dim) abort();
  vecteur *u = clone_vecteur(v);
  for(int k = 0; k < v->dim; k++)
    u->coeffs[k] += w->coeffs[k];
  return u;
}

float produit_scalaire(vecteur *v, vecteur *w) {
  if(v->dim != w->dim) abort();
  float s = 0;
  for(int k = 0; k < v->dim; k++)
    s += v->coeffs[k] * w->coeffs[k];
  return s;
}
```



```c
int main(void) {
  vecteur *v1, *v2, *v3;
  v1 = cree_vecteur(100);
  v2 = cree_vecteur(100);
  randomize_vecteur(v1);
  randomize_vecteur(v2);
  v3 = somme_vecteur(v1, v2);
  printf("%f %f\n",
         produit_scalaire(v1, v3) + produit_scalaire(v2, v3),
         produit_scalaire(v3, v3));
  detruit_vecteur(v1);
  detruit_vecteur(v2);
  detruit_vecteur(v3);
  return 0;
}
```

affiche

```
86.744965 86.744965
```

Contrairement à des langages plus récents, en particuliers ses extensions "orientées objets" C++ et Objective-C, le langage C ne permet pas facilement de gérer la destruction automatique de variables allouées dynamiquement.

Il n'y a pas de moyen simple de faire fonctionner ce code sans fuite de mémoire:

```
1  int main(void) {
2    vecteur *v1, *v2;
3    v1 = cree_vecteur(100);
4    v2 = cree_vecteur(100);
5    randomize_vecteur(v1);
6    randomize_vecteur(v2);
7    s = produit_scalaire(v1, somme_vecteur(v1, v2));
8    detruit_vecteur(v1);
9    detruit_vecteur(v2);
10   return 0;
11 }
```

Le vecteur créé par `somme_vecteur` ligne 7 n'est jamais libéré.

La manipulation dynamique de la mémoire permet même de créer des types dont la taille est dynamique.

Imaginons que l'on veuille récupérer des mesures $X_0, X_1, \ldots$ (températures, vitesses) sans savoir à l'avance combien il y en aura, et dont on voudra calculer la moyenne $\hat{\mathbb{E}}(X)$ et la variance $\hat{\mathbb{V}}(X)$.

Avec les tableaux que nous avons vus jusqu'ici, nous devrions allouer une zone mémoire de la taille maximum possible.

Une alternative plus élégante consiste à agrandir dynamiquement le tableau quand c'est nécessaire.

```c
typedef struct {
  int nb, nb_max;
  float *mesures;
} echantillon;

echantillon *cree_echantillon() {
  echantillon *e = malloc(sizeof(echantillon));
  e->nb = 0;
  e->nb_max = 10;
  e->mesures = malloc(sizeof(float) * e->nb_max);
  return e;
}

void detruit_echantillon(echantillon *e) {
  free(e->mesures);
  free(e);
}
```

Le champ `nb_max` contient la taille du tableau `mesures`, et `nb` combien de valeurs ont déjà été stockées. La taille de 10 initiale ligne 9 est un choix arbitraire.


```c
void ajoute_mesure(echantillon *e, float x) {
  if(e->nb == e->nb_max) {
    float *mesures = malloc(sizeof(float) * 2 * e->nb_max);
    for(int k = 0; k < e->nb; k++) mesures[k] = e->mesures[k];
    free(e->mesures);
    e->mesures = mesures;
    e->nb_max *= 2;
  }
  e->mesures[e->nb++] = x;
}
```

On teste si le tableau est plein ligne 2, si c'est le cas on alloue un tableau deux fois plus grands ligne 3, on y copie les valeurs stockées lignes 4, libère l'ancien tableau ligne 5, et met à jour les champs lignes 6–7.

Dans tous les cas on stock la nouvelle valeur ligne 9.

$$\hat{\mathbb{E}}(X) = \frac{1}{N} \sum_{k=0}^{K-1} X_k$$

```c
float moyenne(echantillon *e) {
  if(e->nb == 0) abort();
  float s = 0;
  for(int k = 0; k < e->nb; k++)
    s += e->mesures[k];
  return s / e->nb;
}
```


$$\hat{\mathbb{V}}(X) = \frac{1}{N-1} \sum_{k=0}^{N-1} \left(X_k - \hat{\mathbb{E}}(X) \right)^2$$

```c
float variance(echantillon *e) {
  if(e->nb < 2) abort();
  float s = 0, m = moyenne(e), d;
  for(int k = 0; k < e->nb; k++) {
    d = e->mesures[k] - m;
    s += d * d;
  }
  return s / (e->nb - 1);
}
```

```c
int main(void) {
  echantillon *e = cree_echantillon();
  for(int k = 0; k < 10000; k++) {
    ajoute_mesure(e, (float) rand() / (float) RAND_MAX);
  }
  printf("%f %f\n", moyenne(e), variance(e));
  detruit_echantillon(e);
  return 0;
}
```

affiche

```
0.497132 0.083105
```


Chaînes de caractères

La conversion entre un octet et un caractère se fait avec une table. La plus standard étant la table ASCII ("American Standard Code for Information Interchange") qui remonte aux années 60s.

0 NUL	1 SOH	2 STX	3 ETX	4 EOT	5 ENQ	6 ACK	7 BEL	8 BS	9 HT	10 LF	11 VT	12 FF	13 CR	14 SO	15 SI
16 DLE	17 DC1	18 DC2	19 DC3	20 DC4	21 NAK	22 SYN	23 ETB	24 CAN	25 EM	26 SUB	27 ESC	28 FS	29 GS	30 RS	31 US
32 SP	33 !	34 "	35 #	36 $	37 %	38 &	39 '	40 (	41)	42 *	43 +	44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 [	92 \	93]	94 ^	95 _
96 `	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 DEL

Des conventions de codage sur plusieurs octets permettent de représenter d'autres alphabets avec des milliers de caractères. Le standard est Unicode.

La convention en langage C est qu'une chaîne de caractères est représentée comme une suite de caractères en mémoire, chacun occupant un octet, suivi du caractère de code zéro. Il indique la fin de la chaîne.

```
1  int longueur_chaine(char *s) {
2    int l = 0;
3    while(s[l] != '\0') l++;
4    return l;
5  }
```

```c
char *clone_chaine(char *s) {
  int l = longueur_chaine(s);
  char *t = malloc(sizeof(char) * (l + 1));
  for(int k = 0; k <= l; k++) t[k] = s[k];
  return t;
}

int main(void) {
  char *a = "abcdefghijklmnopqrstuvwxyz";
  printf("l=%d\n", longueur_chaine(a));
  char *b = clone_chaine(a);
  printf("b=\"%s\"\n", b);
  free(b);
  return 0;
}
```

affiche

```
l=26
b="abcdefghijklmnopqrstuvwxyz"
```



```c
char *sous_chaine(char *s, int debut, int fin) {
  int l = longueur_chaine(s);
  if(debut < 0 || debut > fin || fin > l) abort();
  char *t = malloc(sizeof(char) * (fin - debut + 1));
  for(int k = 0; k < fin - debut; k++) t[k] = s[debut + k];
  t[fin - debut + 1] = '\0';
  return t;
}

int main(void) {
  char *a = "abcdefghijklmnopqrstuvwxyz";
  char *c = sous_chaine(a, 3, 5);
  printf("c=\"%s\"\n", c);
  free(c);
  return 0;
}
```

affiche

```
c="de"
```

Nous verrons dans un prochain cours que la plupart des opérations standard de copies de zone mémoires, de remplissage avec des constantes, et de manipulation de chaînes de caractères existent déjà dans les librairies standard.

Il est [généralement] difficile d'écrire une version plus efficace et robuste que celle écrite par des programmeurs aguerris, et spécifiquement adaptée à un processeur / système.