C4 : Structure de données. le 10/10/2005

• déclaration de variables de type complexe.

Struct COMPLEX c ;

en utilisant le renommage de type, on a :

typedef struct COMPLEX { float real ; float img ; } complex-t ;

• déclaration

complex-t c ;

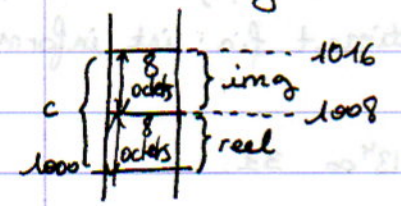
la variable c contient 2 champs qui sont réel et img pour accéder à ces champs, il est nécessaire de les sélectionner. La syntaxe de sélection est la suivante :

<variable> • <nom du champ>

exemple : c.img : sélection de la partie imaginaire.
c.real : sélection de la partie réelle.

implantation mémoire d'un enregistrement.

zone contigüe ex : c (PC 32 bits float 64)



en résumé : &c 1000.
&c.real 1000.
&c.float 1008.

opération standard sur les enregistrements.

lecture / écriture

complex-t c1 = {1.0 ; 0.0} ;

complex-t c2 ;

c2 = c1 ; équivalent à c1.real = c2.real, c2.img = c1.img ;

ATTENTION : ON NE PEUT PAS FAIRE c = c1 + c2 ;

L'addition des complexes s'écrit,

c.real = c1.real + c2.real ;

c.img = c1.img + c2.img ;

Multiplication :

C.reel = c1.reel + c2.reel - c1.img * c2.img

C.img = c1.reel * c2.img + c1.img * c2.reel

Structuration des enregistrements.

Pour éviter la répétition des déclarations; il est fréquent de d'organiser la déclaration d'enregistrement de manière hiérarchique en utilisant des enregistrements d'enregistrements.

exemple : planning.
- debut - date - j. m. a.
  - heure - h. min.
-
-
- fin - date / heure
- information - texte (pour l'ex. un entier)

```
typedef HEURE { short int heure ; short int minute; } heure_t ;
typedef DATE {short int jour ; short int mois; short int an; } date_t;
typedef DATIME { date_t day ; heure_t hour ; } datime_t;
typedef PLAN { datime_t debut; datime_t fin; int information; } plan_t;
```

exemple : lundi 10 octobre 9h30 - 13h00 31.

```
plan_t p ;
date_t date ;
date.jour = 10;
date.mois = 10;
date.an = 2005;
p.debut.day = date;
p.fin.day = date;
p.debut.hour.heure = 9;
p.debut.hour.minute = 30;
p.fin.hour.heure = 13;
p.fin.hour.minute = 00;
p.information = 31;
```

C : TABLEAU STATIQUEMENT DECLARÉ

le 10/10/05.

Les tableaux servent à conserver plusieurs données de même type. (collection de données)
L'organisation de ces données est sous forme d'une table.

Table simple (à 1 dimension)
ex: table de 10 entiers

```
int t[10];
```

chaque élément de la table est INDEXEE par un entier i,
$0 \leq i < 10$;
le premier élément t[0] ;
le dernier élément t[9] ;

Vocabulaire :

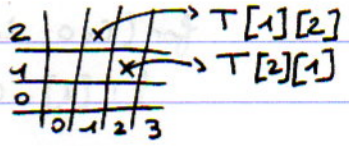
t[n]
n taille
dimension 1

2 éléments caractérisant les tableaux.
- la dimension : la dimension de l'espace se présentant le Tableau,
- la taille (par dimension) le nb de ¢ des tb par dimension.

Tableaux à plusieurs dimensions est représenté pour un espace cartésien de même dimension.

```
int T[4][3]
```

T 2 dimension de taille respective 4 et 3. nb de ¢ est 12 (4x3).
Representation sous forme d'un plan



NOTE :
attention la dimension d'un tableau correspond à l'espace representant les données et non l'espace du problème, un vecteur de dim 3 se représente par un tb de dim 1 mais de taille 3 $\vec{v} = (1,3,7)$ `float v[3];`

Algorithmique sur les tableaux :

Etude de cas :
initialisation valeur initiale aux ¢.

Programme naïf.

```
main ()
{ int T[5];
 t[0] = t[1] = t[2] = t[3] = t[4] = 0;
```

pb : le programme dépend de la taille.

```
#define N 5
 main ()
{ int T[N];
   int i;
  for (i=0; i < N; i++)
      {T[i] = 0;}
```

(Na) Classification des algorithmes :

• Classe A : classe indépendance de l'énumération.
Quelque soit l'ordre d'énumération des index, le prog. est juste.

exemple : initialisation élt à étudier : 1) l'espace d'énumération
2) les ¢ à énumérer ds cet espace.

```
 for (i=0; i<N; i++)
    {T[i] = 0;}
```

est équivalent à `for (i=N-1; i>=0; i--) {T[i] = 0;}`.

initialisation des cases paires

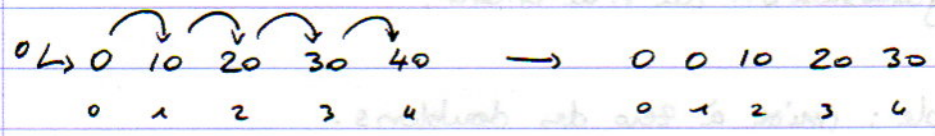
```
 for (i=0; i<N; i+=2)
   {T[i] = 0;}
```

exemple : addition de vecteurs.

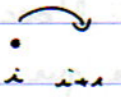
```
 float V1[N], V2[N], V3[N];
  int i;
 for (i=0; i<N; i++)
   {V3[i] = V1[i] + V2[i];}.
```

• Classe B : les algorithmes où la sequence d'énumération importe.
le résultat dépend de l'ordre de parcours.

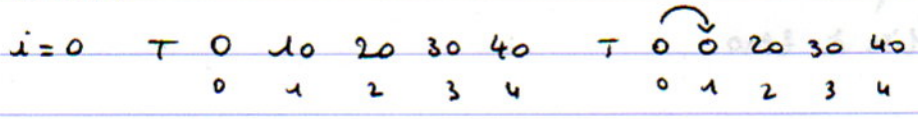
exemple : décalage à droite.

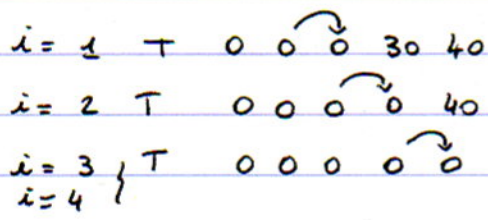



```
main ()
{ int T[N];
  int i;
  for (i=0; i<N-1; i++)
    { T[i+1] = T[i]; }
}
```



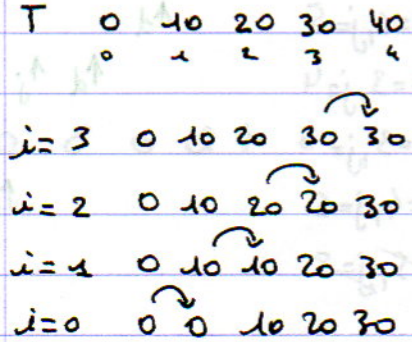
Trace :






```
/* décalage */
main ()
{ int T[N];
  int i;
  for (i=N-1, i>0, i--)
  {T[i+1] = T[i]; }
  T[0] = 0;
}
```

Trace



• Classe C : Classe des algorithmes ayant une composition de traitements.
organisation du traitement.

exemple : mise à zero des doublons.
On elimine l'occurence d'elts apparaissant plus d'une fois (on ne garde que la première apparition).

1 2 ~~1~~ 3 ~~1~~ ~~3~~ ~~3~~ 4 ~~2~~
1 2 0 3 0 0 0 4 0

hyp : les valeurs initiales sont > 0.

Plusieurs traitements composent cet algorithme.
1) Reperer les doublons.
proposer des candidats à la detection des doublons.
2) Mise à zero.

```
main ()
{ int T[N];
  int j, d;
① → for (d=0; d<N; d++)
    { if (T[d] != 0) {
② →   for (j=d+1; j<N; j++)
      { if (T[d] == T[j])
          T[j] = 0; }}
    }
}
```

• Trace :

① d=0, j=? 1 2 1 3 3 (d, j)
① d=0, j=1 (d, j)
③ d=0, j=2 1 2 0 3 3
② d=0, j=3 (j)
② d=0, j=4 (j)
① d=1, j=5 (d, j)
① d=1, j=2 (d, j)
② d=1, j=3 (j)

1 2 0 3 3
② d=1, j=4 (j)
① d=2, j=5 (d, j)
① d=3, j=4 (d, j)
② d=3, j=4 1 2 0 3 0 (d)
① d=4, j=5
① ~~d=5, j=5~~

exemple : Tableau à plusieurs dimensions.
addition de 2 matrices $M = A + B$

```
main ()
{
 float A[N][M], B[N][M], C[N][M];
 int i,j;
   for (i=0; i<N; i++)
      for (j=0; j<M; j++)
        { C[i][j] = A[i][j] + B[i][j]; }
}
```

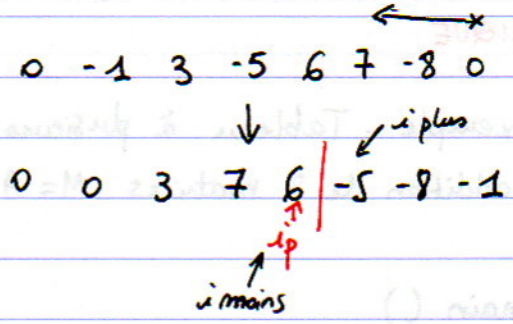
multiplication matrice vecteur $C = A\vec{v}$.

$$C_{ij} = \Sigma a_{ij} \cdot v_j$$

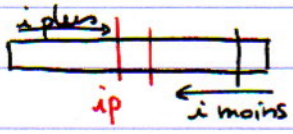
```
main ()
{ double A[N][M];
  double V[M], C[N];
  int i,j;
    for (i=0; i<N; i++)
    { C[i] = 0.0;
      for (j=0; j<N; j++)
        { C[i] += A[i][j] * V[j]; }
    }
}
```

Problème :
Soit $\mathcal{P}$ une propriété définie par un prédicat
$P : \mathbb{N} \to \mathbb{B}$ $\quad \mathbb{B} = \{F, V\}$
On veut organiser un tb tq :
$\exists ip \in [0, N]$, $\forall i$, $i < ip$ $P(T[i])$
$\forall i$, $i \geq ip$ $\neg P(T[i])$

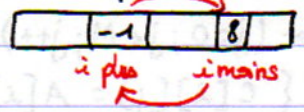
$\forall x : p(i) = i > 0$.



Algorithme.



- iplus doit parcourir des valeurs positives (≥ 0).
- imoins " " " négatives (< 0).
- Si $T[imoins] \geq 0 \wedge T[iplus] < 0$ alors on permute les élts.




```
iplus = 0 ; imoins = N-1 ;
while (iplus < imoins) {
 while (imoins > 0 && T[imoins]<0) {imoins...;}
 while (iplus < N && T[iplus] >=0) {iplus...;}
    if (iplus < imoins)
       {x = T[iplus];
        T[iplus] = T[imoins];
        T[imoins] = x;
       }
                }
```

VARIANTE

```
iplus = i = 0;
while (i<N)
 { if (T[i] >= 0)
   a = T[i];
       if (i != iplus)
       {T[i] = T[iplus];
        T[iplus] = a;
       }
     iplus++;}
  i++;
}
```