Exercice:

Les types TA, TB, TC, TD et TE sont définis comme suit :

TA contient trois valeurs a, b, c
C : typedef enum {a, b, c} TA;

TB contient les entiers de 4 à 7
Pascal : type TB = 4.. 7;

TC contient les valeurs a et b du type TA
Pascal : type TC = a.. b;

TD contient tous les sous- ensembles de TA ({}, {a}, {b}, {c}, {a, b}, {b, c}, {a, c}, {a, b, c})
Pascal : type TD = set of TA;

TE contient des tableaux ayant 10 lignes et 2 colonnes; les éléments sont du type TA
C : typedef TA TE[10][2];
Pascal : type TE = array [1.. 10,1.. 2] of TA;

TF contient des structures ayant un champ du type Booléen et un champ du type TA
C : typedef struct { bool ch1; TA ch2; } TF;
Pascal : type TF = record ch1 : boolean; ch2 : TA end;

TG contient des fichiers dont les composantes sont du type Booléen
Pascal : type G = file of Boolean;

complétez le tableau suivant :

type	nombre de valeurs dans le type	nombre de bits nécessaires pour représenter une valeur du type
TA		
TB		
TC		
TD		
TE		
TF		
TG		

L'allocation de la mémoire aux tableaux.

char tableauCar [20];

double scores[NJudges];

8 octets/élément x 5 éléments = 40 octets

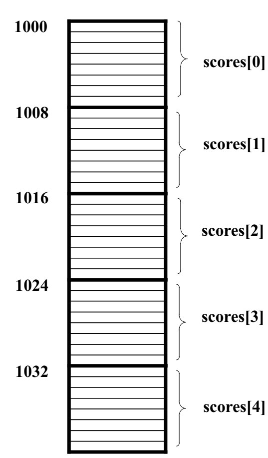
l'adresse d'un élément

scores[i]

2 x 8 + 1000 (si i = 2)

l'adresse de base

déplacement (offset)



En Pascal

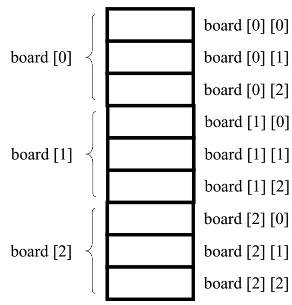
var A: **array**[inf.. sup] **of** elem;

@A[i] = @A + (i- inf) * taille(elem),

Tableaux à plusieurs dimensions.

char board [3] [3];

board [0] [0]	board [0] [1]	board [0] [2]
board [1] [0]	board [1] [1]	board [1] [2]
board [2] [0]	board [2] [1]	board [2] [2]



INF4393 - Jurek Czyzowicz

Autres langages (Pascal, Modula2, Ada, Fortran)

type TAB = **array**[inf1.. sup1, inf2.. sup2,..., infn.. supn] **of** elem;

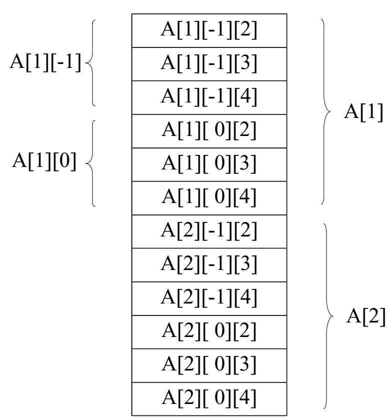
type TAB = **array**[inf1.. sup1] **of**
  array[inf2.. sup2, ..., infn.. supn] **of** elem;

type TAB = array[1.. 10, BOOLEAN, 2.. 8] of CHAR

type TAB = **array**[1.. 10] **of**
  array[BOOLEAN] **of**
    array[2.. 8] **of**
      CHAR;

INF4393 - Jurek Czyzowicz

var A, B: **array**[1.. 2,- 1.. 0, 2.. 4] **of** CHAR;



Modifications possibles:

A[1][-1][2] := B[1][-1][2]

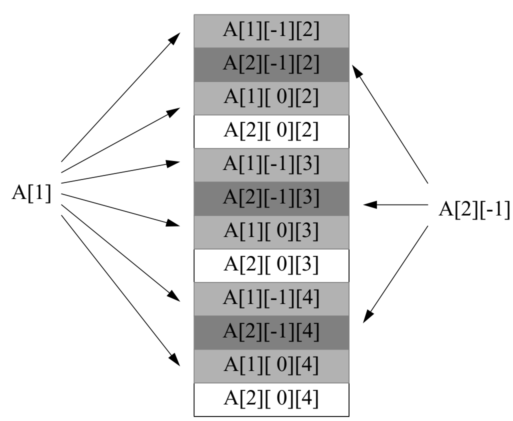
A[1][0] := A[2][-1]

A[1] := A[2]

A := B

INF4393 - Jurek Czyzowicz

Une autre représentation



var A: **array** [inf1.. sup1, inf2.. sup2] **of** elem;

@A[k1, k2] =
    @A + [(k1- inf1) * l2 + (k2- inf2)] * taille(elem)

où l2 = sup2 - inf2 + 1

var A : **array** [inf1.. sup1, inf2.. sup2, inf3.. sup3] **of** elem;

@A[k1, k2, k3] =
    @A + [((k1- inf1)* r2 + (k2- inf2))* r3 + (k3- inf3)] * taille(elem)

où r2 = sup2 - inf2 + 1,
r3 = sup3 - inf3 + 1

@A[k1, k2, k3] =
    @A + (k1- inf1) * taille(tranche)
        + (k2- inf2) * taille(ligne)
        + (k3- inf3) * taille(elem)

où taille(ligne) = (sup3 - inf3 + 1) * taille(elem),
taille(tranche) = (sup2 - inf2 + 1) * taille(ligne)
    = (sup2 - inf2 + 1) * (sup3 - inf3 + 1) * taille(elem)

Tableaux dynamiques

- introduits en Algol 60

- les bornes des tableaux peuvent changer durant l'exécution

Exemple:

... read(K, L, M);
begin
    integer array A[K: 10; L: M];
    A[I+ 5, J] := ...

Descriptif du tableau: (les informations permettant de calculer l'adresse d'une variable indexée au cours de l'exécution)

@A
K
10
L
M
l2=M-L+1

integer array A[inf1: sup1; inf2: sup2; inf3: sup3];

@A
inf1
sup1
inf2
sup2
inf3
sup3
taille(ligne) = sup3 - inf3 + 1
taille(tranche) = (sup3 - inf3 + 1) * (sup2 - inf2 + 1)

Ensembles

Exemple (Pascal):

type
    Couleurs = (Blanc, Jaune, Rouge, Bleu, Vert, Noir);
    EnsembleDeCouleurs = **set of** Couleurs;

var E1, E2 : EnsembleDeCouleurs;

E1 := [Blanc..Rouge, Noir];

31		6	5	4	3	2	1	0
0	...	0	1	0	0	1	1	1
			Noir			Rouge	Jaune	Blanc

E1 = [Jaune, Rouge, Bleu, Vert];
E2 = [Bleu, Noir];

	A∪B	**A − B**	**A∩B**
A:	011110	011110	011110
B:	101000	101000	101000
	111110	010110	001000
	[Jaune..Noir]	[Jaune, Rouge, Vert]	[Bleu]

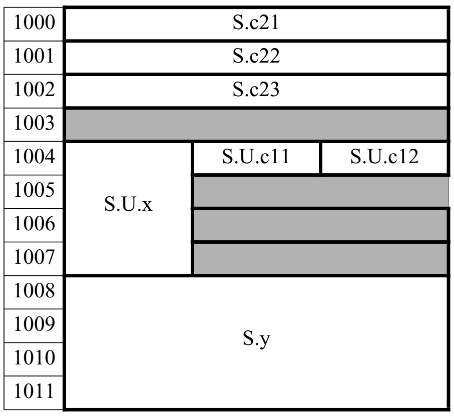
Structures

```
union un {
    float x;
    char c11, c12;
};

struct st {
    char c21, c22, c23;
    u U;
    float y;
};
```

int - 4 octets
float - 4 octets avec un alignement de 4
char - 1 octet

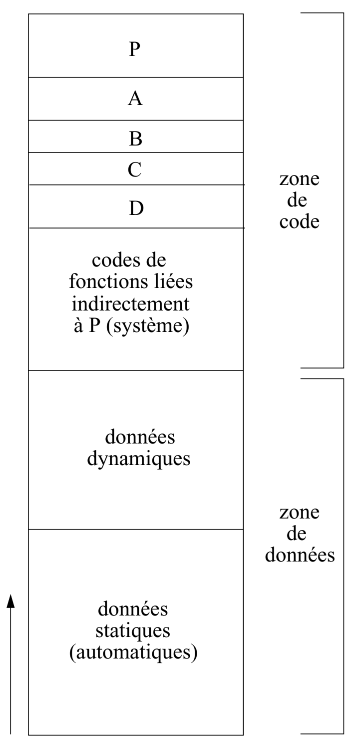
st S /* alloué à l'addresse 1000 */;



Temps de compilation et temps d'exécution.

Organisation de la mémoire machine durant l'exécution d'un programme

```
program P (...)
    var P1, P2;
    procedure A(A1)
        var A2;
        procedure B
            var B1, B2;
        {
        . . .
        } B;
    {
    B
    } A;
    procedure C(C1, C2)
        procedure D
            var D1;
        {
        . . .
        C
        . . .
        D
        . . .
        } D;
    {
    D
    } C;
{
A
}
```

Segment de données

Un segment de la pile correspondant à un appel d'une procédure (ou une fonction) qui contient les données de cette procédure et d'autres informations d'organisation.

Il contient

- l'adresse de retour (au niveau du zone de code) du sous-programme appelant,

- lien statique (l'adresse du segment de données du sous-programme imbriquant directement le sous-programme appelé,
- lien dynamique (l'adresse du segment de données du sous-programme appellant)

- les variables statiques du sous-programme

- les paramètres du sous-programme