

ALGORITHMIQUE & STRUCTURES DE DONNÉES STATIQUES 1 (ING. INFO.)

(VERSION 2023-24)

Pr. Boulef M.

Programme :

ch0 : Généralités

ch1 : Notions de base

ch2 : Structures de contrôle

ch3 : Tableaux unidimensionnels

ch4 : Tableaux bidimensionnels

ch5 : Chaînes de caractères

ch6 : Actions paramétrées.

Bibliographie

1. Mounira Belmesk et Nacera Bensaou, Algorithmes et structures de données, Khawarysm, 1991.
2. Claude Delannoy, Programmer en langage C, Eyrolles, 2007.
3. Ivor Horton, Beginning C, 5th Edition (Expert's Voice in C), Apress, 2013.
4. Jean-Claude Faure, Emploi des ordinateurs, Dunod 1971.
5. Internet : Par exemple,

OpenClassrooms (ex. site du zéro),
<https://openclassrooms.com/fr/courses>

Developpez.com,
<https://general.developpez.com/cours/>

Stackoverflow, etc.
<https://stackoverflow.com/>



6. d'autres refs que j'ai oubliées.

GÉNÉRALITÉS

0.1 PRÉSENTATION D'UN ORDINATEUR

Les composants d'un ordinateur sont en général :

1. un processeur central;
2. une mémoire centrale (dite mémoire vive ou RAM);
3. un clavier;
4. un moniteur;
5. un lecteur DVD;
6. des interfaces de connexion à des périphériques tels que : imprimante, scanner, réseaux, ... etc.

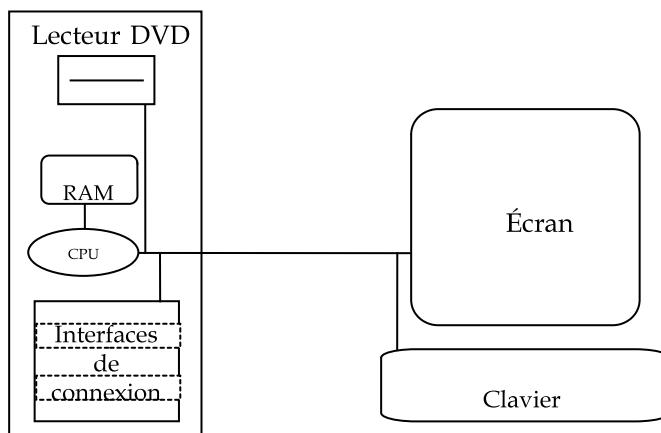


Figure 0.1 Un ordinateur.

Ces composants forment la partie hardware de l'ordinateur est peut être schématisé par le diagramme suivant:

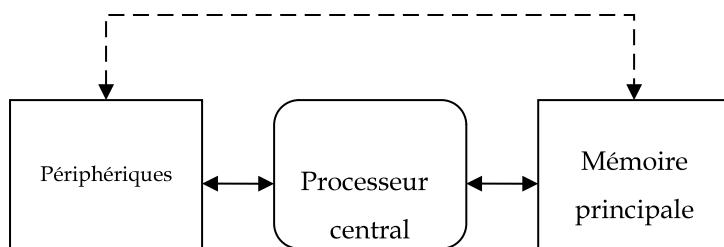


Figure 0.2 Architecture de Von Neumann

Dans ce schéma les composants suivants apparaissent:

1. Le processeur central ou CPU: Exécute les instructions de programmes en *langage machine*.
2. La mémoire principale: contient les **programmes** spécifiant les instructions à exécuter et les **informations** ou **données** traitées par ces instructions.
3. Les périphériques ou organes d'entrée-sortie: utilisés pour introduire programmes et données à la mémoire principale et pour communiquer les résultats des activités réalisées.
4. Des bus reliant les précédents organes.

0.2 SOFTWARE ET HARDWARE

L'ensemble des composants hardware est sans aucune utilité (le mot anglais hardware veut dire quincaillerie). Pour devenir utile, il faut lui associer une autre partie dite software. En effet, l'utilisateur d'un ordinateur demande essentiellement deux types de services :

1. Créer et nommer des fichiers ; conserver ces fichiers dans des supports de stockage (ex. [flash-disque](#)) ; transférer de l'information entre les fichiers et les organes d'entrée-sortie (écran, clavier, imprimante).
2. Exécuter des programmes existants déjà ou introduits à l'ordinateur; les données sont introduites au clavier ou prélevées dans un fichier ; les résultats sont affichés à l'écran, listées sur imprimante ou copiées dans un fichier.

Aucun de ces traitements n'est possible sans le software. De façon générale le software se divise en deux parties: le système d'exploitation et les logiciels d'application.

Un système d'exploitation, tel que Linux et Windows, permet de réaliser les deux fonctions suivantes:

1. Gérer les travaux de l'utilisateur en dissimulant à celui-ci les détails de fonctionnement.
2. Gérer les ressources de l'ordinateur en contrôlant et affectant celles-ci aux différentes applications (programmes) en exécution de façon raisonnable.

Les logiciels d'applications permettent à l'utilisateur de réaliser des tâches spécifiques telles que traitement de texte (Word, Gedit), programmation (Delphi, Builder), navigation dans le web (FireFox, IEExplorer), etc.

0.3 REPRÉSENTATION DE L'INFORMATION SUR ORDINATEURS

Les ordinateurs bien qu'ils puissent traiter du texte, projeter des vidéos, faire tourner des jeux, etc ; ils ne sont capables que d'une seule chose: faire des calculs. La vérité est que, lorsqu'un ordinateur traite du texte, de l'image, du son ou de la vidéo, il ne traite que des nombres codés en binaires.

0.3.1 Information binaire

Une information binaire ne peut avoir que deux états: ouvert ou fermé, libre ou occupé, blanc ou noir, droite ou gauche, vrai ou faux, etc.

Un ordinateur possède différents supports physiques capables de stocker de l'information binaire: La mémoire vive (ou RAM) est formée de millions de composants électroniques qui peuvent retenir ou relâcher une charge électrique. La surface d'un disque dur, d'une bande ou d'une disquette est recouverte de particules métalliques qui peuvent, grâce à un aimant, être orientées dans un sens ou dans l'autre. Sur un CD-ROM, on trouve un long sillon étroit irrégulièrement percé de trous.

0.3.2 Représentation d'une information binaire

On symbolise une information binaire, quel que soit son support physique, sous la forme de 1 et de 0.

0.4 PRISE EN CHARGE DES PROGRAMMES PAR ORDINATEUR

Les applications sont des programmes à l'origine écrits dans un langage de programmation de haut niveau proche du langage humain, tel que Pascal et C. Pour être exécutés, ces programmes sont transformés par des outils logiciels. Ces outils qui peuvent être séparés ou intégrés sont décrits comme suit:

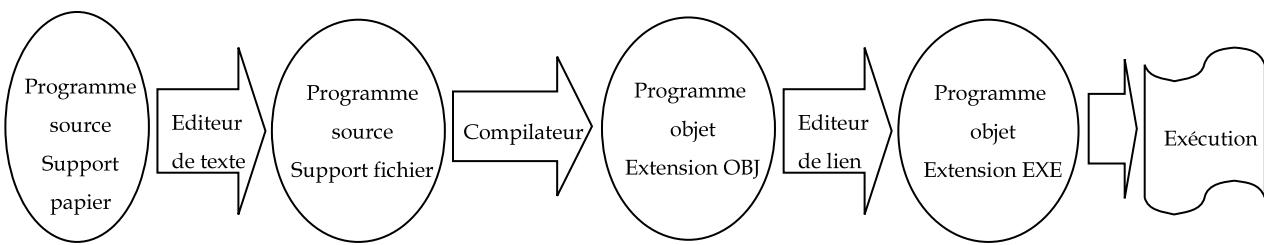


Figure 0.3

Vie d'un programme utilisateur

0.4.1 Editeur de texte

C'est un programme permettant de saisir du texte et de le sauvegarder dans un fichier de type texte. Il sert à écrire le code source du programme utilisateur écrit dans un langage de haut niveau.

Exemple: Notepad (bloc note) de Windows.

0.4.2 Compilateur

C'est un programme permettant de traduire le programme source écrit dans un langage évolué, en un programme binaire compris par la machine. Le programme binaire obtenu est appelé **programme objet**.

Exemple: gcc comprenant le compilateur C du projet GNU sous UNIX.

0.4.3 Editeur de liens

C'est un programme qui permet d'assembler le code objet du programme utilisateur et celui des procédures et fonctions prédéfinies en un seul et unique programme exécutable.

Cette étape réalisée, le programme utilisateur est enfin prêt à être pris en charge par le SE.

0.5 APPROCHE ALGORITHMIQUE A LA PROGRAMMATION

La programmation procure un moyen de communiquer avec un ordinateur car le langage naturel est trop compliqué pour qu'il soit compris par une machine dépourvue d'intelligence. Donc apprendre un langage de programmation est nécessaire. Cependant, il y a plusieurs langages programmation pour lesquels il est difficile de dire que l'un est meilleur que les autres dans tous ses aspects. Heureusement, tous ces langages possèdent des points en commun qui seront récapitulés dans ce que nous appelons langage algorithmique. Par la suite, pour écrire un programme, on commence par écrire un algorithme en utilisant le langage algorithmique ; ensuite le passage au programme se fera par remplacement de toute instruction algorithmique par son équivalent du langage de programmation choisi.

NOTIONS DE BASE

```

Algorithme < nom de l'algorithme > ;
    < Partie déclarations >
Debut
    < Partie actions >
Fin.

```

Structure d'un algorithme

- Les types de base sont :
 - entier
 - reel
 - caractere : par exemple a,b,c ..., A,B,C, ..., 0,1, ..., 9,+,-,:, ... (voir table ASCII)
 - bool : pour booléen acceptant deux valeurs uniquement vrai ou faux.
- Les actions de base sont :
 - La lecture : lire(<variables>)
 - L'affectation : <variable> \leftarrow <expression> (on peut utiliser \leftarrow ou $:=$)
 - L'écriture : ecrire(<expressions>)

Exemple :

```

algorithme simple ;
    var A,B,somme:entier ;
debut
    ecrire('Donner 2 nombres') ;
    Lire(A,B) ;
    somme  $\leftarrow$  A+B ;
    Ecrire('Le resultat :',somme) ;
fin.

```

Que fait cet algo ?

La traduction de cet algo en C donne :

```

#include <stdio.h>

int main(){
    int A,B,somme ;
    printf("Donner deux nombres") ;
    scanf("%d %d",&A,&B) ;
    somme =A+B ;
    printf("Le resultat est %d",somme) ;
    return 0 ;
}

```

Remarques et propriétés

- Compilation : Une instruction aussi simple que $C \leftarrow A + B$ requiert être traduite en:

```

LOAD R0, adr A      ; charge A dans le registre R0
LOAD R1, adr B      ; charge B dans le registre R1
ADD R0, R1           ; R0  $\leftarrow$  R0 + R1
STORE R0, adr C     ; stocke R0 à l'adresse de C

```

- En algo on suppose que la casse n'est pas importante mais en C elle l'est.
- Les opérations arithmétiques pour les réels et les entiers sont : +,-,*,/. Appliquée entre deux entiers / donne un réel. Appliquées entre un réel et un entier elles donnent un réel.
- On supposera en algorithmique qu'il n'y a pas de borne pour les entiers et les réels.
- Un **identificateur** est une chaîne alphanumérique pouvant utiliser _ mais commençant par un alphabétique et est différente de tout mot réservé (algorithme, var, debut, fin, etc.).
- Pour l'évaluation d'expressions, on supposera l'ordre de priorités décroissant suivant :
 1. NON, - unaire ;
 2. Opérateurs multiplicatifs : *, /, ET ;
 3. additifs : +, - binaire, OU ;
 4. relationnels : <, >, <=, >=, <>, = ;

Les conflits sont évalués de gauche à droite.
- On ne peut affecter un réel à un entier, mais on peut les comparer ;
- On peut comparer deux caractères et le résultat dépend de leurs codes ASCII.

Notions de bases en C

Algorithmique	C
var x : entier ;	int x ;
var y : reel ;	float y ;
var z : caractère ;	char z ;
var b : bool ;	#include <stdbool.h> bool b ;
lire(x) ;	scanf(``%d``,&x) ;

L'instruction conditionnelle

Si <condition> Alors <action>
fsi ;

Si <condition> Alors <action 1>
Sinon <action 2>
fsi ;

- Le délimiteur de bloc fsi doit être mis qu'il y ait une ou plusieurs instructions dans le bloc. Par conséquent, pour la variante composée, la mise d'un ';' avant sinon n'est pas interdite.

Exemples :

1. Si A < B Alors min ← A
 Sinon min ← B ;
 fsi ;
 Ecrire (min) ;

2. Calcul du minimum avec un seul Si simple.

Instruction conditionnelle en C

En C les blocs sont délimités par {} (en cas d'une seule action, on peut les omettre mais c'est déconseillé).

Les correspondances algo-langage C sont :

Algo	C
Si <condition> Alors <action> fsi ;	if(<cond>){ <action> }

<pre> Si < condition > Alors <action1> sinon <action2> fsi ; </pre>	<pre> if(<cond>){ <action1> }else{ <action2> } </pre>
---	---

Conseil : En C, même s'il y a une seule instruction mettre des accolades.

Exemples

1. Produit de deux entiers. Traduire en C.
2. Valeur absolue d'un entier donné. Traduire en C.

LES BOUCLES

2.1 Boucle « TANT QUE »

```
Tant que <condition> faire  
    < action >  
    ftq ;
```

Exemple : Algorithme d'écriture répétée

```
Lire (n) ;  
I ← 1  
Tant que I <= n faire  
    Ecrire ('Bonjour') ;  
    I ← I + 1 ;  
ftq ;
```

2.2 ACTION RÉPÉTITIVE « POUR »

```
Pour < comp > ← < val_init > à < val_finale >  
    faire  
        < action >  
    fpour ;
```

Exemple :

```
nbrFois=4 ;  
Pour i ← 1 à nbrFois faire  
    Lire (note) ; Ecrire ('note obtenue :', note) ;  
fpour ;
```

2.3 ACTION RÉPÉTITIVE « RÉPÉTER ... JUSQU'À »

```
Répéter  
    < action >  
Jusqu'à < condition > ;
```

Exemple

```
Lire (n) ; i ← 1 ;  
Répéter  
    Ecrire ('je suis dans la boucle') ;  
    i ← i + 1 ;  
jusqu'à (i > n) ;  
Ecrire ('je suis sorti') ;
```

STRUCTURES DE CONTRÔLE EN C

En C les blocs sont délimités par { } (en cas d'une seule action, on peut les omettre mais c'est déconseillé).

Les correspondances algo-langage C sont:

Algo	C
Tant que <condition> faire <action> ftq ;	While (<condition>){ <action> }
Pour <comp> ← <val_init> à <val_fin> faire <action> fpour ;	for(<comp>=<val_init>;<comp> <= <val_fin> ;<comp>++){ <action> }
Répéter <action> Jusqu'à <condition> ;	do{ <action> }while(!<condition>);

Exemples

1. Somme de n éléments entiers. Traduire en C (utiliser le tableau de correspondances Algo-C)
2. Produits de n éléments entiers.
3. Somme des n premiers éléments positifs.

TABLEAUX UNIDIMENSIONNELS

- Un tableau unidimensionnel est déclaré par:

var <identificateur>: Tableau (<taille>) <type de base> ;

ou `<taille>` est une constante entière positive indiquant le nombre d'éléments du tableau.

- Un élément est désigné par `<identificateur>[<indice>]` ou `<identificateur>(<indice>)`.

<indice> peut prendre une valeur entière positive entre 1 et <taille> (et non pas entre 0 et <taille>-1 comme en C).

Exemples :

- Inversion de tableau avec et sans tableau supplémentaire.

- Somme des éléments de rang pair d'un tableau d'entiers:

```

Algorithme somPosPair ;
Const tailleMax=100 ;
Var
    N,I : Integer ;
    Vect : Tableau (tailleMax) Entier ;
Début
    Lire(N) ; {N <=tailleMax}
    Pour I  $\leftarrow$  1 à N faire
        Lire(Vect(i)) ; {on peut faire la somme ici mais ...}
        fpour ;
        somIndPair $\leftarrow$ 0 ;
        Pour I  $\leftarrow$  1 à N/2 faire
            somIndPair  $\leftarrow$  somIndPair+Vect(i*2)) ;
        fpour ;
        ecrire(somIndPair) ;
Fin.

```

Tri de tableaux

Le tri d'un tableau consiste à ordonner ses éléments dans un ordre croissant.

```

Algorithme triSimple ;
var
    n,i,j,tmp: entier ;
    vect : tableau (50) Entier ;
Debut
    {Lecture}
    ecrire('Donner n <=50 :') ;
    lire(n) ;
    pour i ← 1 jusqu'a n faire
        lire(Vect(i)) ;
    fpour ;
    {Tri }
    pour i ← 1 jusqu'a n-1 faire
        pour j ← i+1 jusqu'a n faire
            si vect(j) < vect(i) Alors
                tmp ← vect(i) ;
                vect(i) ←vect(j) ;
                vect(j) ←tmp ;
            fsi ;
            fpour ;
        fpour ;
    {Affichage}
    pour i ← 1 jusqu'a n faire
        ecrire(vect(i)) ;
    fpour ;
fin.

```

Tableaux unidim. en C

Algorithmique	C
<pre> vect : tableau (10) entier ; ecrire(``Donner n<=10``) ; lire(n) ; pour i ← 1 jusqu'a n faire ecrire(``Donner un element du tableau``) ; lire(vect[i]) ; fpour ; </pre>	<pre> int vect[10] ; printf(``Donner n<=10``) ; scanf(``%d``,&n) ; for(i=0 ; i<n;i++){ printf(``Donner un element du tableau``) ; scanf(``%d``,&vect[i]) ; } </pre>

TABLEAUX BIDIMENSIONNELS

- Un tableau bidimensionnel est déclaré par:
`Var <identificateur>: Tableau (<nombre lignes>,<nombre colonnes>) <type de base> ;`
 ou `<nombre lignes>` et `<nombre colonnes>` sont des constantes entières positives.
- Un élément est désigné par `<identificateur>[<indice ligne>,<indice colonne>]`. On peut utiliser les `()` au lieu des `[]`.
- L'indice de ligne peut prendre une valeur entière positive entre 1 et `<nombre lignes>`.
- L'indice de colonne peut prendre une valeur entière positive entre 1 et `<nombre colonnes>`.

Exemple :

```

Algorithme manipMat ;
const NbrLignMax=10,NbrColMax=10 ;
var n,m,i,j : entier;
    mat : tableau (NbrLignMax,NbrColMax) entier ;
Debut
    Ecrire('Donner le nombre de lignes (≤',NbrLignMax,',') et celui de colonnes(≤',NbrColMax,','):' );
    Lire(n,m) ;
    pour i ← 1 jusqu'à n faire
        pour j ← 1 jusqu'à m faire
            Lire(mat(i,j)) ;
            fpour ;
        fpour ;
    pour i ← 1 jusqu'à n faire
        somLigne=0 ;
        pour j ← 1 jusqu'à m faire
            somLigne=somLigne+mat(i,j) ; {on peut réaliser ces traitements en lecture mais ...}
            fpour ;
        ecrire(somLigne) ;
    fpour ;
Fin.

```

Correspondances algo C

Algorithmique	C
<code>mat : tableau (10,10) entier ;</code> <code>ecrire(``Donner n<=10 et m<=10``) ;</code> <code>lire(n,m) ;</code> <code>pour i ← 1 jusqu'à n faire</code> <code> pour j ← 1 jusqu'à m faire</code> <code> ecrire(``Donner un element :``) ;</code> <code> lire(vect[i]) ;</code> <code> fpour ;</code> <code>fpour ;</code>	<code>int mat[10][10] ;</code> <code>printf(``Donner n<=10 et m<=10``) ;</code> <code>scanf(``%d %d``,&n,&m) ;</code> <code>for(i=0 ; i<n; i++){</code> <code> for(j=0 ; j<m; j++){</code> <code> printf(``Donner un element :``) ;</code> <code> scanf(``%d``,&mat[i][j]) ;</code> <code> }</code>

Exemples:

- Somme des éléments d'une matrice. Traduire en C.
- Somme de chaque ligne. Traduire en C.
- Somme de chaque colonne.
- Somme des éléments de la première diagonale d'une matrice quelconque.
- Somme des éléments de la deuxième diagonale d'une matrice quelconque.

CHAÎNES DE CARACTÈRES

Une chaîne de caractères est une suite de caractères ASCII (*American Standard Code for Information Interchange*) terminée par le caractère de code ASCII nul '\0'.

Exemple :

```
Algorithme IllustrChaine;
    Var      s,s1,s2:chaine(10);
              b: bool;
debut
    s1 ← 'Bonjour';
    s2 ← 'Bonsoir';
    ecrire(s1 < s2); { vrai, car 'j' < 's'}
    ecrire(s1 = 'bonjour'); { faux, car 'B' < 'b'. En effet, 66 < 98 }
    ecrire(' Bonsoir' > 'bonjour'); { faux, car ' ' < 'b'. En effet, 32 < 98}
    ecrire(s1 + s2) ; {affiche BonjourBonsoir }
    s ← s1+s2 ; ecrire(s) ; {affiche BonjourBo car '\0' inclus}
    ecrire(longueur(s)); {affiche 9}
    s1← 'Non'; ecrire(longueur(s)); {affiche 3}
fin.
```

- **Déclaration :** var <identifiant> : chaine(<taille maximale>) ;
où <taille maximale> est une constante entière positive égale au nombre maximum de caractères ('\0' compris).
- **Constantes :** Une constante chaîne de caractères est mise entre cotes simples. Exemple:

'je suis une chaine constante'
- **Actions de base :** On peut lire ou écrire une variable chaîne comme on peut lui affecter une autre.

Exemple :

```
Algorithme manipChaine ;
    var      maChaine1 :chaine(20);
              maChaine2 :chaine(10);
debut
    lire(maChaine1) ; {mettra '\0' automatiquement}
    maChaine2 ← maChaine1 ; {idem}
    ecrire(maChaine2) ;
fin.
```

- Si la chaîne lue ou affectée contient plus de caractères que la taille maximale de la chaîne réceptrice, cette dernière contiendra les <taille maximale>-1 premiers caractères puis '\0' uniquement ; i.e. les caractères en plus sont ignorés.
- On peut affecter un caractère à une variable chaîne et le résultat est une chaîne composée du caractère affecté suivi du caractère '\0'.

- **Dégroupage :** On peut accéder à un élément (qui est de type caractère) de la chaîne de caractères par son indice mis entre () ou [].

Exemple :

```
var maChaine : chaine(10) ;
debut
    maChaine ← 'bonjour' ;
Fin.
```

Après l'affectation, maChaine(1) contient le caractère 'b' et maChaine(8) contient '\0' ;

- Opérateurs :**
 - On peut comparer deux chaînes de caractères par les opérateurs <, >, <=, >=, =, <> et le résultat booléen dépendra de l'ordre lexicographique déterminé par le code ASCII de leurs caractères.
 - L'opérateur + appliquée sur deux chaînes (une chaîne et un caractère ou deux caractères) permet de les concaténer (sans incidence sur ces chaînes).
- Exemple :** maChaine1 ← maChaine2+maChaine3 ;
- Fonctions prédefinies :**
 - longueur(maChaine) : donne la longueur courante de maChaine sans compter '\0'.
 - ord(<caractere>) : donne le code ASCII du caractère <caractere>.
 - car(numeroCode) : donne le caractère ayant le code ASCII numeroCode.

Chaines de caractères en C

Algorithmique	C	Observation
var maChaine :chaine(20);	char maChaine[20] ;	Ajouter #include <string.h>
lire(maChaine) ;	scanf(`%s`, maChaine) ;	En C attention au débordement.
maChaine2 ← maChaine1 ;	strcpy(maChaine2,maChaine1)	idem
ecrire(maChaine) ;	printf(`%s`, maChaine) ;	
longueur(maChaine)	strlen(maChaine)	
maChaine2 < maChaine1	strcmp(maChaine2,maChaine1)<0	
maChaine2 > maChaine1	strcmp(maChaine2,maChaine1)>0	
maChaine2 = maChaine1	strcmp(maChaine2,maChaine1)==0	
maChaine1←maChaine1+maChaine2 ;	strcat(maChaine1,maChaine2) ;	Modifie maChaine1, attention au débordement

Exemples

- Tri d'un tableau de noms sans espace et de même casse. Traduire en C.
- Tri d'un tableau de noms sans espace.
- Nombre de mots d'une phrase sans ponctuation

Table ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	Ø	96	60	140	`	~
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	%	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	%	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	-	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	Ø	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	:	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

ACTIONS PARAMÉTRÉES

Une action paramétrée est structuration d'une ou de plusieurs actions en une seule en lui associant un identificateur qui permet de la référencer en tout point de l'algorithme.

```
Action <NomAction> [(<ListParam>)] ;
  <Partie Déclaration>
  Début
    <Partie action>
  Fin ;
```

- **Déclaration et définition :** Pour les actions paramétrées, on suppose que déclaration et définition sont confondues. Ces déclarations sont placées avant l'algorithme principal (voir exercice 1).
- **Syntaxe :** (Exemples)
 - o Une action paramétrée de type procédure avec *par1* entier transmis en entrée et *par2* entier transmis en sortie :


```
Action nomAction (E/ par1 :entier; S/ par2 :entier) ;
  Debut ... Fin ;
```

(au lieu du mot clé **Action** on peut utiliser **Procedure**)
 - o Une action paramétrée de type fonction retournant un entier avec *par1* entier transmis par valeur et *par2* entier transmis par variable :


```
Fonction nomFct (E/ par1 :entier; Ref/ par2 :entier) : entier ;
  Debut ... retourne(expression) ; Fin ;
```
- **Modes de transmission :** Quatre modes de transmission de paramètres : **Entrée** (ou par valeur), **Sortie**, **Mixte** et par **Référence** (ou par variable) spécifiés par E/, S/, ES/ et Ref/ respectivement. Pour expliquer le déroulement avec un schéma de RAM, on supposera qu'il y a création de variables locales de substitution dans l'espace pile pour les 3 premiers modes (E/, S/ et ES/).

On suppose qu'un tableau peut être transmis par valeur.

Exemple : Permutation de deux nombres

```
Action Permut(ES/ x,y : Entier) ;
Var   tmp : Entier ;
Debut
  tmp ← x ; x ← y ; y ← tmp ;
Fin ;
Algorithme permutation ;
```

Var A,B : Entier ;

Début

Lire(A,B) ; permut(A,B) ; Ecrire(A,B) ;

Fin.

Schéma de RAM associé

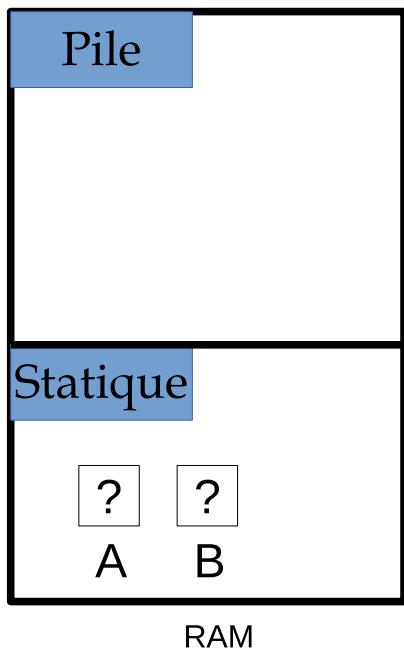
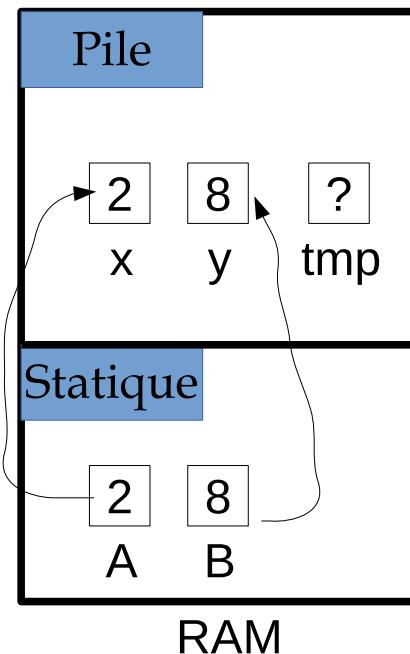
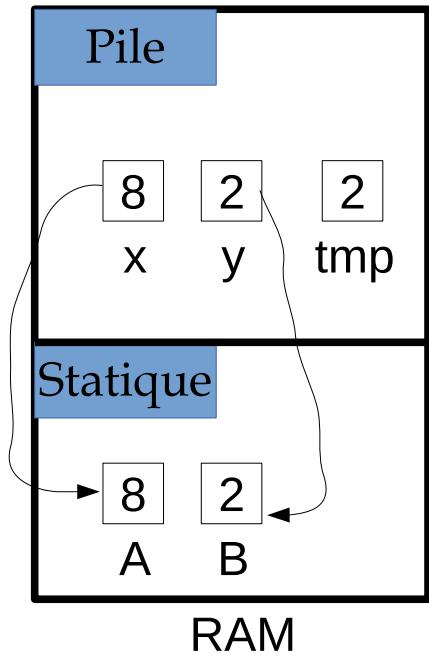


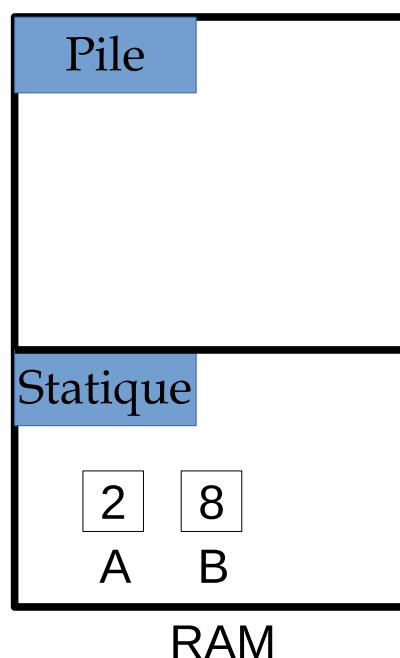
Schéma de RAM avant exécution de lire(a,b)



Après appel de l'action paramétrée



Lors du retour de l'action



Après retour à l'algorithme principal

Exercice : Utiliser le mode Ref au lieu de ES puis donner le schéma de RAM associé.

- **Visibilité**

Les variables déclarées dans l'algorithme principal sont globales. Une action paramétrée peut appeler une autre si la définition de l'appelée est réalisée avant celle de l'appelante. Dans ce cas, les variables de cette dernière sont vues dans l'appelée.

Exemple :

```
Action suivant(ES/ z:entier) ;
debut z ← z+1 ;      fin ;

Action addit(E/x,y:entier ; S/som :entier) ;
    var val:entier ;
debut
    som ← x ;
    val ← 0 ;
    Tant que val<y faire
        suivant(som) ;
        suivant(val) ;
    ftq ;
fin ;

Algorithme Additionner ;
    var A,B,somme:entier ;
debut lire(A,B) ; addit(A,B,somme);ecrire(somme) ; fin.
```

Objets déclarés dans	Visibilité
A : Additionner	A, SA1, SA2
SA1 : Addit	SA1, SA2
SA2 : suivant	SA2

- Globalité absolue (objets de A) et localité absolue (ceux de SA2).
- La spécification de globalité et localité dépend du langage.
- Éviter la modification des variables globales qui ne sont pas des paramètres localement.
- Déclaration et définition d'une action paramétrée.

Action paramétrées en C (exemple)

```
#include <stdio.h>
// Prototypes des fonctions appelées, Declaration Globale
int lireEntier(),
    max2Entiers(int nbr1, int nbr2);

// on met main en premier
int main()
{
    // Déclaration des variables locales
    int A, B;

    // Traitement avec appel des fonctions
    A = lireEntier();
    B = lireEntier();
    printf("Le maximum est %d\n", max2Entiers(A,B));
    return 0;
}
//On definit les fonctions restantes progressivement
// Définition de la fonction lireEntier
int lireEntier(void)
{
    int nbr;
    printf("Entrez un nombre entier : ");
    scanf("%d", &nbr);
    return nbr;
}

/* Définition de la fonction max2Entiers */
int max2Entiers(int nbr1, int nbr2)
{
    if (nbr1>nbr2)
        return nbr1;
    else
        return nbr2;
}
```

Exercices :

- Valeur absolue (procédure et fonction)
- Somme des éléments d'un tableau d'entiers. (procédure et fonction)
- Produit de deux matrices d'entiers.