Le langage C

Hervé Boisgontier

Plan Langage C

1. Les instructions de base en langage C
2. Les instructions conditionnelles
3. Les instructions itératives
4. Les instructions peu orthodoxes
5. Les tableaux
6. Les types évolués
7. Les pointeurs
8. Les fonctions
9. L'allocation dynamique de mémoire
10. Les fichiers

Plan Programmation Système

11. Langage C et Linux
12. Les fichiers
13. Les processus
14. Les signaux
15. La communication avec la mémoire partagée
16. Les sémaphores
17. Les threads

Ressources

Livre "Langage C" (2nd édition) de Frédéric DROUILLON – éditions ENI – juillet 2021

Explications rédigées dans le livre

Exercices réalisables dans le livre

Le langage C

Les instructions de base en langage C

Un exemple de programme en C

```c
#include<stdio.h>
// calcule le prix TTC d'un article
const float TVA = 20.0F/100;
int main() {
  float prixHT;
  printf("Prix HT de l'article ?\n");
  scanf(" %f", &prixHT);
  printf("Prix TTC de l'article : %.2f\n",
                  prixHT*(1+TVA));
  return 0;
}
```

Les commentaires

```
#include<stdio.h>
// calcule le prix TTC d'un article
const float TVA = 20.0F/100;
int main() {
  float prixHT;
  printf("Prix HT de l'article ?\n");
  scanf(" %f", &prixHT);
  printf("Prix TTC de l'article : %.2f\n",
                  prixHT*(1+TVA));
  return 0;
}
```

Les commentaires

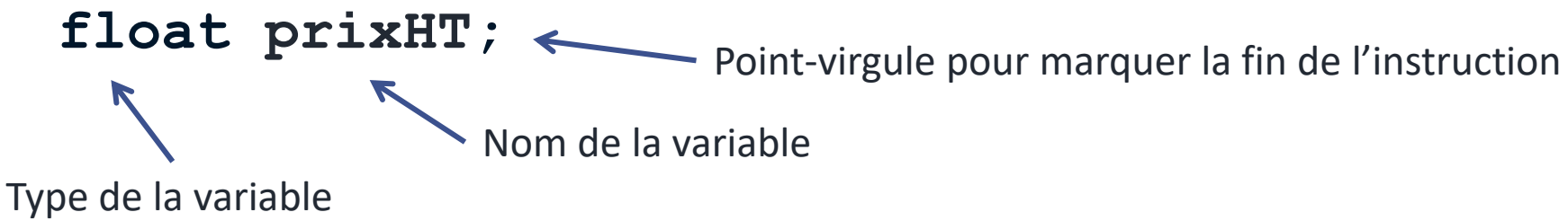
```
//calcule le prix TTC d'un article
```

- // commentaire jusqu'à la fin de la ligne
- /* commentaire entre slash étoile et
  étoile slash */

Déclaration d'une variable

```
#include<stdio.h>
// calcule le prix TTC d'un article
const float TVA = 20.0F/100;
int main() {
   float prixHT;
   printf("Prix HT de l'article ?\n");
   scanf(" %f", &prixHT);
   printf("Prix TTC de l'article : %.2f\n",
                  prixHT*(1+TVA));
   return 0;
}
```

Déclaration d'une variable



- Instruction permettant de réserver en mémoire de l'espace pour stocker une donnée
- Nom de la variable : nom sans espace commençant par une minuscule.
- Types possibles : `int`, `char`, `short`, `long`, `float`, `double`…

Les types

Types	Définition	taille	Domaine
`int`	Nombre entier positif ou négatif	taille suivant architecture	
`char`		1 octet	-128 à +127
`short`		2 octets	-32 768 à +32 767
`long`		4 octets	-2 147 483 648 à +2 147 483 647
`long long`		8 octets	$-9{,}223.10^{18}$ à $+9{,}223.10^{18}$
`unsigned int`	Nombre entier positif	taille suivant architecture	
`unsigned char`		1 octet	0 à 255
`unsigned short`		2 octets	0 à 65 535
`unsigned long`		4 octets	0 à 4 294 967 295
`unsigned long long`		8 octets	0 à $1{,}844.10^{19}$
`float`	Nombre réel positif ou négatif	4 octets	$-3{,}402.10^{38}$ à $+3{,}402.10^{38}$
`double`		8 octets	$-1{,}798.10^{308}$ à $+1{,}798.10^{308}$

et c’est tout !

et les booléens ?

- utilisation d’un entier (char) :
  - faux : `0`
  - vrai : toute autre valeur
- utilisation du type bool (avec #include <stdbool.h>)
  - faux : `false`
  - vrai : `true`

et les caractères ?

- Ce n’est que de l’affichage !

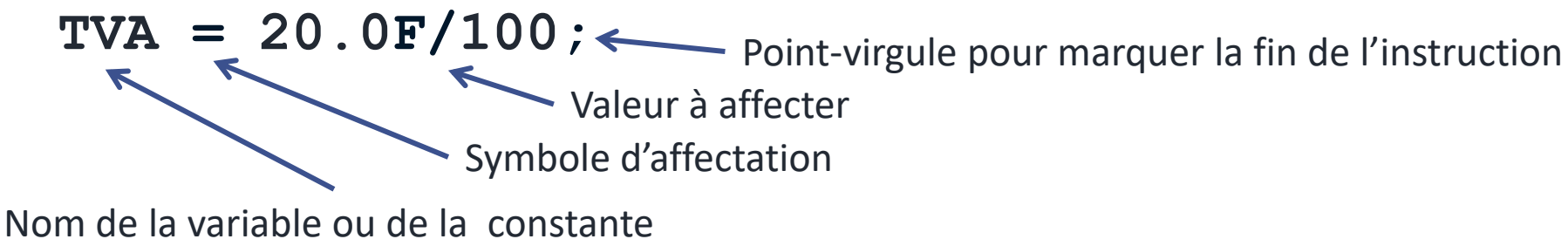
et les chaînes de caractères ?

- Patience, ce sera vu plus tard…

Affectation d'une valeur

```
#include<stdio.h>
// calcule le prix TTC d'un article
const float TVA = 20.0F/100;
int main() {
  float prixHT;
  printf("Prix HT de l'article ?\n");
  scanf(" %f", &prixHT);
  printf("Prix TTC de l'article : %.2f\n",
                  prixHT*(1+TVA));
  return 0;
}
```

Affectation d'une valeur



- Instruction permettant d'affecter une valeur à une variable ou à une constante
- Valeur à affecter : valeur ou calcul du même type que la variable ou la constante.
- Pour une constante l'affectation doit être faite en même temps que la déclaration

Classe d'allocation

Variable auto (par défaut)

Une variable locale commence à exister au moment de sa déclaration et est perdue à la fin du bloc la contenant

Variable static

La variable est initialisée qu'une seule fois. Au passage suivant, elle conserve sa valeur.

```
for(int i=0; i<10; i++) {
        static int nb=0;
        printf("%d\n", ++nb);
}
```

Littéraux entiers

Un nombre entier est par défaut un ‘int’ signé

```
int a = 9; /* entier signé en base 10 */
```

Ajout d’un suffix pour les autres

```
unsigned int b = 9U; /* entier non signé */
long c = 9L; /* entier long signé */
unsigned long d = 9UL; /* entier long non signé */
long long e = 9LL; /* entier long long signé */
```

Les valeurs littérales entières dans d'autres bases

```
int f = 0x1A7; /* entier signé en base 16 */
int g = 0107; /* entier signé en base 8 */
```

Littéraux réels

Un nombre réel est par défaut un ‘double’

```
double e = 9.25; /* double précision */
```

Ajout d’un suffix pour la simple précision

```
float f = 9.25F; /* simple précision */
```

Puissances de 10

```
double g = 9.25E17; /* 9.25 × 10^17 */
```

Littéraux caractères et chaîne de caractères

Un caractère est représenté entre simple quotte (apostrophe)

```
char h = 'E'; /* un caractère */
```

Une chaîne de caractères est représenté entre double quotte (guillemet)

```
printf("Bonjour !"); /* une chaîne de caractères */
```

Déclaration d'une constante

```
#include<stdio.h>
// calcule le prix TTC d'un article
const float TVA = 20.0F/100;
int main() {
  float prixHT;
  printf("Prix HT de l'article ?\n");
  scanf(" %f", &prixHT);
  printf("Prix TTC de l'article : %.2f\n",
                  prixHT*(1+TVA));
  return 0;
}
```

Déclaration d'une constante

```
const float TVA = 20.0F/100;
```

Mot clé indiquant la déclaration

Type de la constante

Nom de la constante

Valeur de la constante

- Comme une variable sauf que la valeur ne change jamais.
- Nom de la constante: nom sans espace écrit tout en majuscule.
- Même type de données que pour une variable

Calculs

```c
#include<stdio.h>
// calcule le prix TTC d'un article
const float TVA = 20.0F/100;
int main() {
  float prixHT;
  printf("Prix HT de l'article ?\n");
  scanf(" %f", &prixHT);
  printf("Prix TTC de l'article : %.2f\n",
                  prixHT*(1+TVA));
  return 0;
}
```

Calculs

`20.0F/100` `prixHT*(1+TVA)`

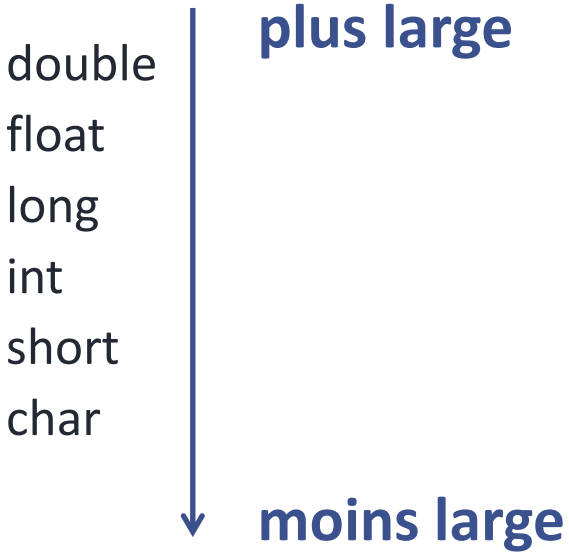
Opérateurs arithmétiques :

- `+` : addition
- `-` : soustraction
- `*` : multiplication
- `/` : division
- `%` : reste de la division entière

Calculs

`20.0F/100` `prixHT*(1+TVA)`

Calcul réalisé dans le type le plus large entre les deux opérandes



Calculs

`20.0F/100` `prixHT*(1+TVA)`

Calcul réalisé dans le type le plus large entre les deux opérandes

Possibilité de considérer une valeur dans un autre type (cast)

```
int prixTotal = 300;
int quantite = 7;
float prixUnitaire = (float)prixTotal/quantite
```

Calculs

```
1<20F/100        17!=34        22<=22
```

Opérateurs arithmétiques,

Opérateur de comparaison :

- `==` : Egale
- `!=` : Différent
- `<` : Inférieur
- `<=` : Inférieur ou égale
- `>` : Supérieur
- `>=` : Supérieur ou égale

Ils retournent **0** si c'est faux ou une valeur positive dans le cas contraire

Calculs

```
!(1<20F/100)&&(17!=34||22<=22)
```

Opérateurs arithmétiques,

Opérateur de comparaison,

Opérateurs booléens : A `op` B

- `&&` : A et B doivent être vrai pour que A `&&` B soit vrai
- `||` : soit A, soit B, soit les deux doivent être vrai pour que A `||` B soit vrai
- `!` : inverse la valeur de A

Calculs

```
~(12|5)&(27^34)  //donne 48
```

Opérateurs arithmétiques,

Opérateur de comparaison,

Opérateurs booléens,

Opérateurs bit à bit :

- **&** : Et logique
- **|** : Ou logique
- **^** : Ou exclusif
- **~** : Non logique

Calculs

```
17<<1 /* décale tous les bits vers la gauche de 1 position */
```

Opérateurs arithmétiques,

Opérateur de comparaison,

Opérateurs booléens,

Opérateurs bit à bit,

Décalage binaire :

- `<<` : Décalage à gauche
- `>>` : Décalage à droite

Calculs

```
a<<b  /* correspond à a*2^b */
43<<3 /* 43*2^3 = 43*8 = 344 */

a>>b  /* correspond à a/2^b */
43>>3 /* 43/2^3 = 43/8 = 5 */
```

Opérateur d'affectation

Calcul et affectation

```
a += b;  // correspond à a = a + b;
a -= b;  // correspond à a = a - b;
a *= b;  // correspond à a = a * b;
a /= b;  // correspond à a = a / b;
a %= b;  // correspond à a = a % b;
a |= b;  // correspond à a = a | b;
a &= b;  // correspond à a = a & b;
a <<= b; // correspond à a = a << b;
a >>= b; // correspond à a = a >> b;
```

Pré et Post Incrémentation

Pré (Dé)Incrémentation :

```
++a // correspond à a = a + 1;
--a // correspond à a = a - 1;
```

Post (Dé)Incrémentation :

```
a++ // correspond à a = a + 1;
a-- // correspond à a = a - 1;
```

Pré et Post Incrémentation

Quelle différence alors?

La pré (de)incrémentation (de)incrémente la variable avant son utilisation :

```
int a=1;
printf("%d\n", ++a);
// affiche 2
```

La post (de)incrémentation (de)incrémente la variable après son utilisation :

```
int a=1;
printf("%d\n", a++);
// affiche 1
```

Pré et Post Incrémentation

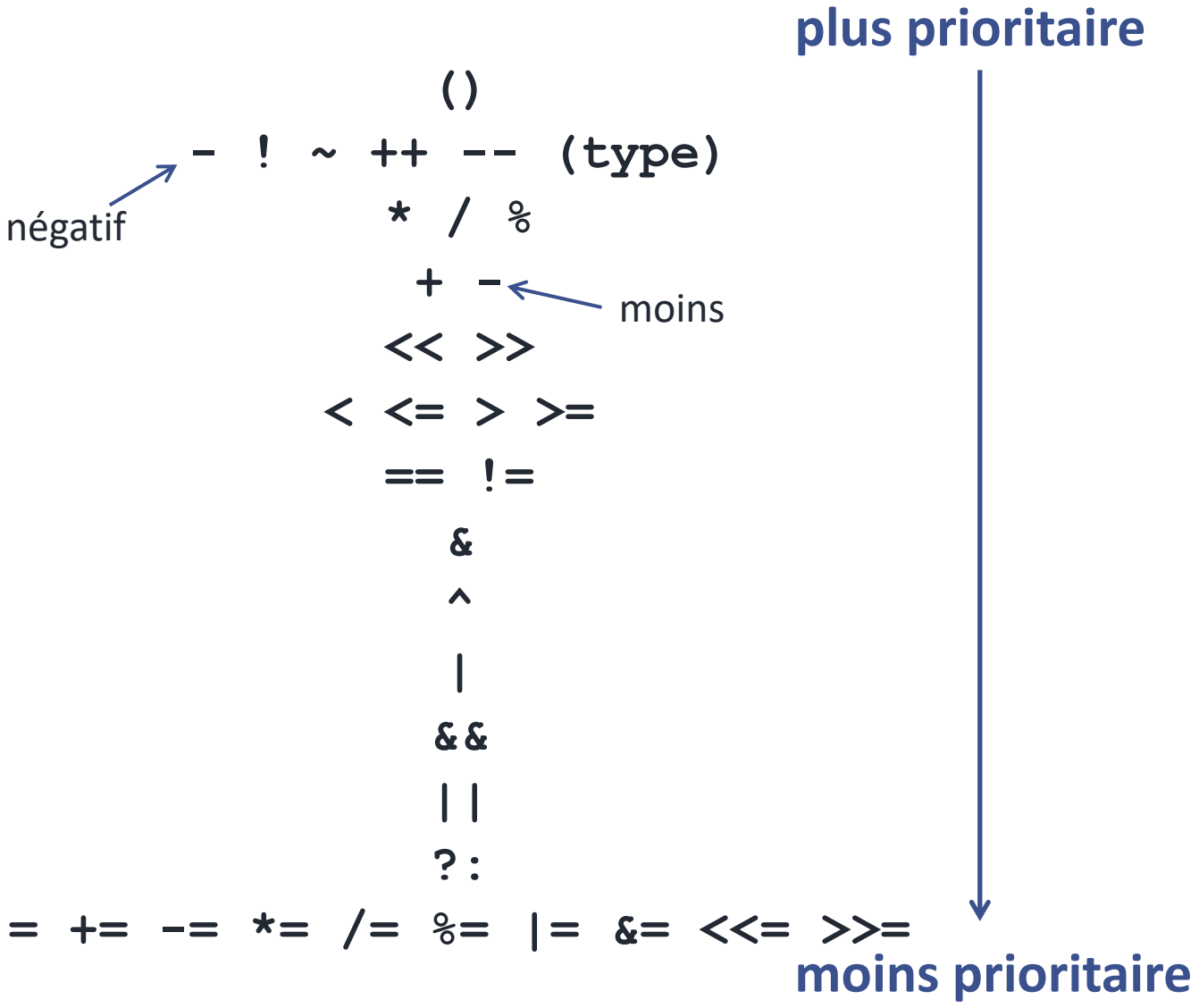
Pré Incrémentation :

```
int a=1; printf("%d\n", ++a);
// correspond à
int a=1; a=a+1; printf("%d\n", a);
```

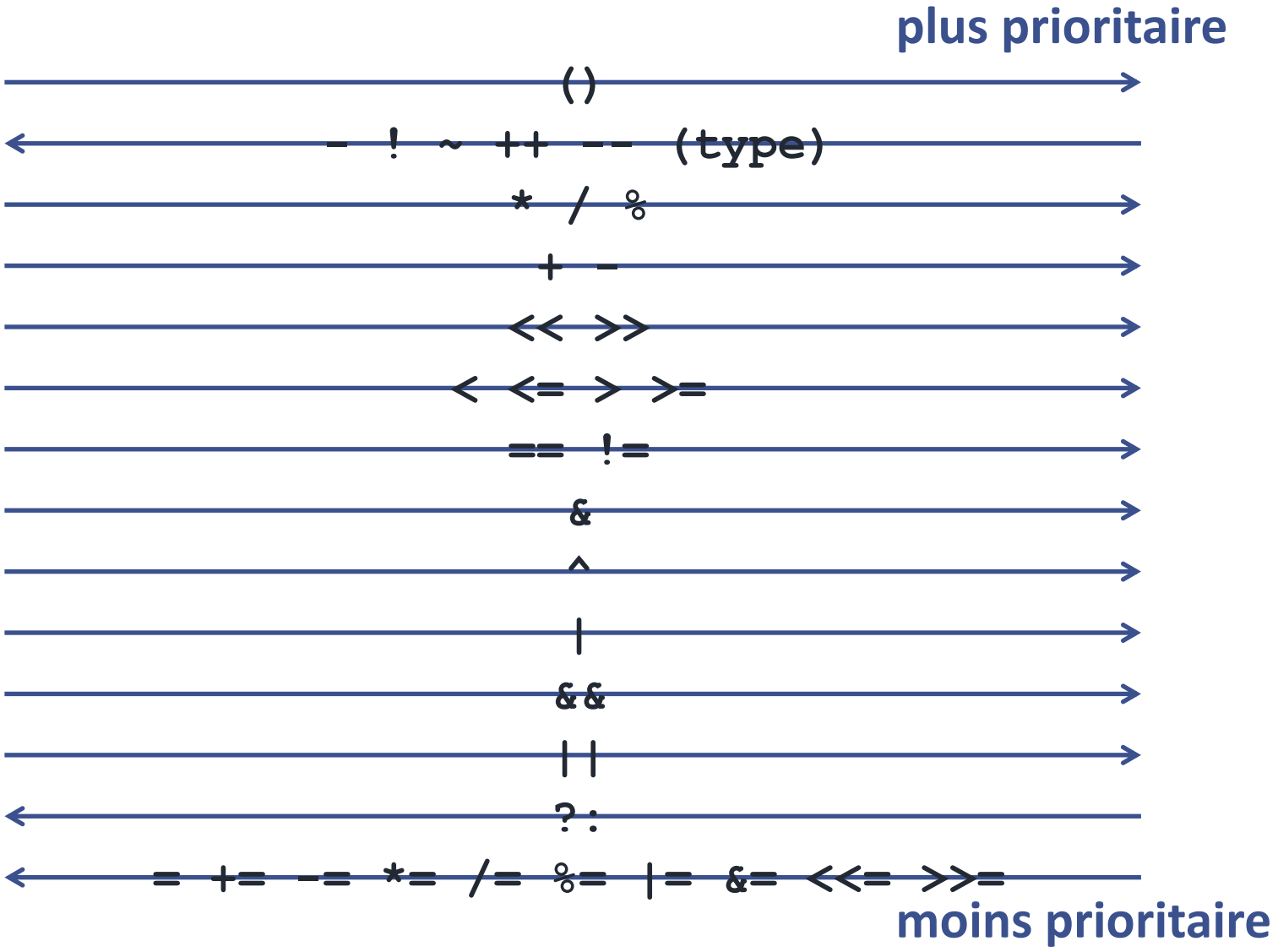
Post Incrémentation :

```
int a=1; printf("%d\n", a++);
// correspond à
int a=1; printf("%d\n", a); a=a+1;
```

Priorité des opérateurs



À égalité de priorité



À égalité de priorité

- Exemples


```
int i = 23>>1<<2; /* 23>>1 puis 11<<2 donc i=44 */
/* et non pas 1<<2 puis 23>>4 donc i=1 */
```


```
char j = -~7; /* application de ~ puis de - */
```

Affichage

```c
#include<stdio.h>
// calcule le prix TTC d'un article
const float TVA = 20.0F/100;
int main() {
  float prixHT;
  printf("Prix HT de l'article ?\n");
  scanf(" %f", &prixHT);
  printf("Prix TTC de l'article : %.2f\n",
                  prixHT*(1+TVA));
  return 0;
}
```

Affichage

```
printf("Prix HT de l'article ?\n");
printf("Prix TTC de l'article : %.2f\n",
                        prixHT*(1+TVA));
```

Mot clé indiquant l'affichage

Variables pour compléter les trous

Texte à trous à afficher

- Permet d'afficher des informations à l'utilisateur

Affichage

Les caractères spéciaux :

- `\n` : retour à la ligne
- `\"` : guillemet
- `%%` : pourcent
- `\\` : antislash
- `\t` : tabulation

Affichage

%[drapeaux][taille][.précision][préfixe]spécifieur

Les différents spécifieurs :

- `%d` : entier signé en base 10
- `%u` : entier non signé en base 10
- `%f` : réel (`%.2f` pour 2 chiffres après la virgule)
- `%c` : caractère (dans la table ASCII à cette position)
- `%x` : hexadécimal
- `%s` : chaîne de caractères (`%.2s` pour seulement 2 caractères)
- `%p` : adresse mémoire

Les préfixes :

- `hh` : char (exemple `%hhd`)
- `h` : short
- `l` : long
- `ll` : long long

Affichage

Exemples

```
printf("=%d=\n", 101);                    // =101=
printf("=%f=\n", 101.4278);               // =101.427800=
printf("=%.2f=\n", 101.4278);             // =101.43=
printf("=%c=\n", 101);                    // =e=
//car 'e' caractère n°101 dans la table ASCII
printf("=%x=\n", 101);                        // =65=
printf("=%s=\n", "cent un");                  // =cent un=
printf("=%.4s=\n", "cent un");                // =cent=
printf("=%hhu=\n", (unsigned char)255); // =255=
```

La table ASCII

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Affichage

La taille, l'alignement et les drapeaux :

`%6d` : affiche sur 6 caractères aligné à droite

```
printf("=%3d=\n", 10); // = 10=
```

`%+6d` : affiche sur 6 caractères aligné à droite avec le signe

```
printf("=%+3d=\n", 10); // =+10=
```

`%06d` : affiche sur 6 caractères avec des 0 devant

```
printf("=%03d=\n", 10); // =010=
```

`%-6d` : affiche sur 6 caractères aligné à gauche

```
printf("=%-3d=\n", 10); // =10 =
```

`%-6.2f` : affiche sur 6 caractères (point compris) aligné à gauche avec 2 chiffres après la virgule

```
printf("=%-7.2f=\n", 10.4278); // =10.43  =
```

`%*d` : affiche sur un nombre de caractères indiqué en argument aligné à droite

```
printf("=%*d=\n", 5, 10); // =   10=
```

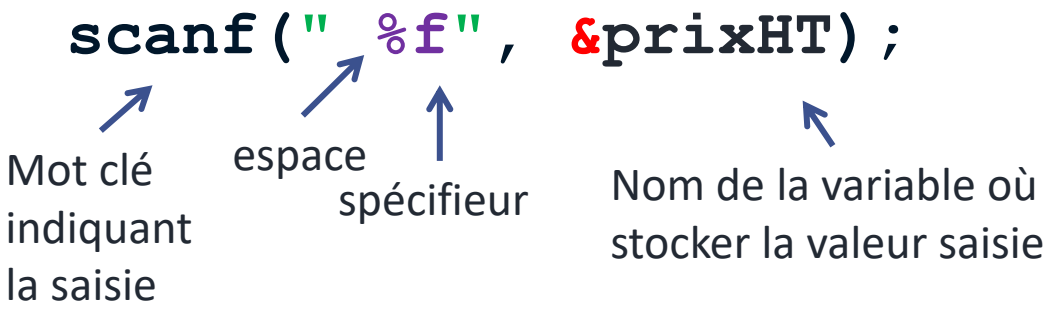
`%*.*f` : affiche sur un nombre de caractères indiqué en argument avec un nombre de chiffres après la virgule indiqué en argument

```
printf("=%*.*f=\n", 8, 3, 10.4278); // =  10.428=
```

Saisie

```c
#include<stdio.h>
// calcule le prix TTC d'un article
const float TVA = 20.0F/100;
int main() {
  float prixHT;
  printf("Prix HT de l'article ?\n");
  scanf(" %f", &prixHT);
  printf("Prix TTC de l'article : %.2f\n",
                  prixHT*(1+TVA));
  return 0;
}
```

Saisie



- Instruction permettant de stocker dans une variable la valeur saisie par l'utilisateur
- Mêmes spécifieurs que pour l'affichage
- Esperluette devant le nom de la variable

Un exemple de programme en C

```c
#include<stdio.h>
// calcule le prix TTC d'un article
const float TVA = 20.0F/100;
int main() {
  float prixHT;
  printf("Prix HT de l'article ?\n");
  scanf(" %f", &prixHT);
  printf("Prix TTC de l'article : %.2f\n",
                  prixHT*(1+TVA));
  return 0;
}
```

Saisie d'un caractère

Récupération du caractère saisi sans appui sur Entrer

`getchar() : int`

Valeur retournée

- Caractère saisi
- EOF si touche Entrer

Exemple :

```c
#include <stdio.h>

int main()
{
    char c = getchar();
    printf("%d : %c", c, c);
    return 0;
}
```

Les valeurs pseudo-aléatoires

Initialisation d'un générateur de nombres pseudo-aléatoires

```
srand(unsigned int graine)
```

Paramètre

- graine : position initiale dans les valeurs pseudo-aléatoires
  - Valeur constante : toujours la même série de valeurs (pratique pour le débuggage)
  - Valeur différente à chaque exécution : utilisation de `time(NULL)`

Génération d'une valeur pseudo-aléatoire

```
rand() : int
```

Valeur retournée

- Valeur comprise entre `0` et `RAND_MAX`

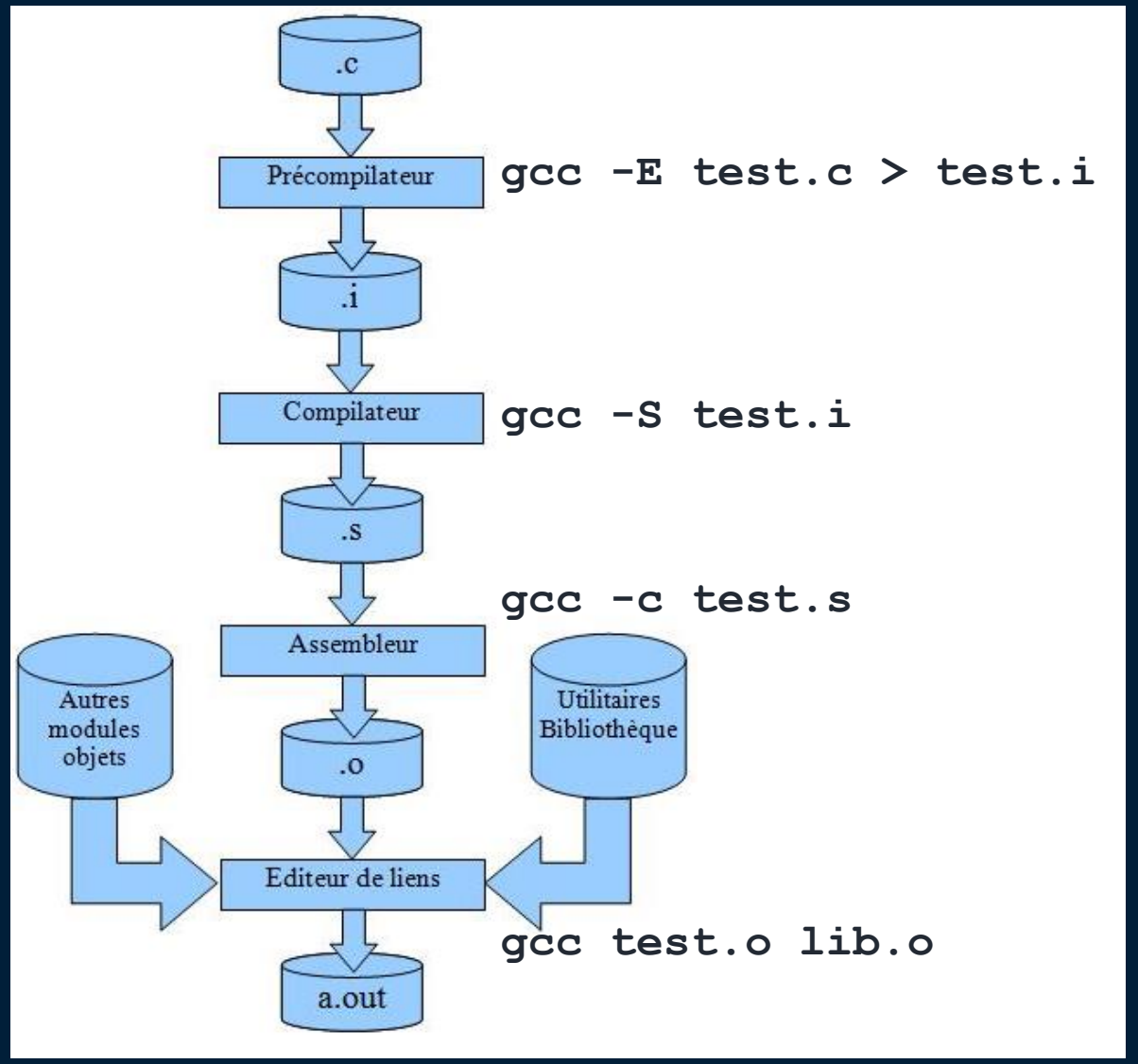
Les valeurs pseudo-aléatoires

```c
#include<stdlib.h> // pour srand() et rand()
#include<time.h>  // pour time()

int main() {
    int min = 7;
    int max = 12;
    // Initialisation du générateur de nombres aléatoires
    srand(time(NULL));
    // Valeur entière aléatoire comprise entre 0 et 9
    int a = rand()%10;
    // Valeur entière aléatoire comprise entre min et max
    int b = rand()%(max-min+1)+min;
    // Valeur réelle aléatoire comprise entre 0 et 1
    float c = (float)rand()/RAND_MAX;
    // Valeur réelle aléatoire comprise entre min et max
    float d = (float)rand()/RAND_MAX * (max-min) + min;
//...
    return 0;
}
```

II – 6
III – 4 et 6
IV – 1.3

Génération d'un exécutable

- Précompilation
  - Remplacement des macros, inclusion de fichiers…
- Compilation
  - Analyse lexicale
  - Analyse syntaxique
  - Traduction du code en assembleur
- Assemblage
  - Traduction de l'assembleur en langage machine
- Édition des liens
  - Regroupement des codes produits



Exécution

```
Prix HT de l'article ?
25
Prix TTC de l'article : 30.00
```

Exercice 1.1 :
Quels affichages ?

```c
#include<stdio.h>
int main() {
  char c = 'E';
  printf("\tAffichage d'un caractere\n");
  printf("%%c -> %c\n", c);
  printf("%%d -> %d\n", c);
  printf("%%x -> %x\n", c);

  int e = 72;
  printf("\tAffichage d'un entier\n");
  printf("%%c -> %c\n", e);
  printf("%%d -> %d\n", e);
  printf("%%x -> %x\n", e);

  float f = 19.6F/100;
  printf("\tAffichage d'un nombre reel\n");
  printf("%%f -> %f\n", f);
  printf("%%.3f -> %.3f\n", f);
  printf("%%.2f -> %.2f\n", f);
  return 0;
}
```

Exercice 1.1 :
Quels affichages ?

Exercice 1.2 : Division euclidienne

Écrire un programme qui calcul affiche le résultat et le reste de la division entière de deux valeurs saisies par l'utilisateur (short)

Affichages et saisies :

Saisissez le dividende

370

Saisissez le diviseur

24

370 / 24 = 15 et reste 10

Exercice 1.2 :
Division euclidienne

Exercice 1.3 : Les opérateurs

```c
#include<stdio.h>
int main() {
  int e = 72;
  printf("%d\n", e != 72);
  printf("%d\n", e && 72);
  printf("%d\n", !e || 0 && e);
  printf("%d\n", e & 123);
  printf("%d\n", -~e | 23);
  printf("%d\n", e ^ 23);
  printf("%d\n", e<<2);
  printf("%d\n", e>>2);
  return 0;
}
```

Le langage C

Les instructions conditionnelles

Le test if : Forme simple

```
if(valeur) {
      suite_d_instructions
}
```

ne vaut pas 0

Si la **valeur** est « vraie » alors la **suite_d_instructions** est exécutée

Exemple :

```
int age;
printf("Quel est votre age ?\n");
scanf("%d", &age);
if(age >= 18) {
      printf("Vous etes majeur !\n");
}
```

Le test if : Forme double

```
if(valeur) {
      suite_d_instructions
} else {
      autres_instructions
}
```

Si la **valeur** est « vraie » alors la **suite_d_instructions** est exécutée, dans le cas contraire ce sont les **autres_instructions** qui sont exécutées.

```
int age;
printf("Quel est votre age ?\n");
scanf("%d", &age);
if(age >= 18) {
      printf("Vous etes majeur !\n");
} else {
      printf("Vous etes mineur !\n");
}
```

Exercice 2.1 : Calculatrice 1

Créer une calculatrice simplifiée :

saisissez le premier opérande

123

saisissez l'opérateur (+ ou –)

–

saisissez le second opérande

45

123 – 45 = 78

Exercice 2.1 : Calculatrice 1

Exercice 2.2 : Calculatrice 2

```c
#include<stdio.h>
//Que fait-il ?
int main() {
  int op1, op2, res;
  char operateur;
  printf("saisissez le premier operande\n");
  scanf(" %d", &op1);
  printf("saisissez l'operateur (* ou /)\n");
  scanf(" %c", &operateur);
  printf("saisissez le second operande\n");
  scanf(" %d", &op2);
  if(operateur=='/')
    if(op2!=0)
      res = op1 / op2;
  else
    res = op1 * op2;
  printf("%d %c %d = %d\n", op1, operateur, op2, res);
  return 0;
}
```

Saisie

Plusieurs valeurs peuvent être saisies en un fois

Exercice 2.3 : Calculatrice 3

Modifier le programme de calculatrice pour pouvoir saisir l'opération en une seule fois :

saisissez l'opération :

123 – 45

123 – 45 = 78

Exercice 2.3 : Calculatrice 3

L'opérateur conditionnel

```
valeurTest?valeur1:valeur2
```

Si la **valeurTest** est « vraie » alors **valeurTest?valeur1:valeur2** vaut **valeur1** sinon **valeur2**.

Exemple :

```
printf("Il y a %d de%s.\n", nbDes, nbDes>1?"s":"");
```

À la place de

```
if(nbDe>1)
  printf("Il y a %d des.\n", nbDes);
else
  printf("Il y a %d de.\n", nbDes);
```

Exercice 2.4 : Calculatrice 4

Réécrire le programme de l’exercice précédent sans utiliser l’instruction `if`

Exercice 2.4 : Calculatrice 4

Le test switch

```
switch(variable) {
   case valeur1 : instructions; break;
   case valeur2 : instructions; break;
   case valeur3 : instructions; break;
   …
   default : instructions;
}
```

L'instruction permet de faciliter l'écriture s'il y a plus de deux choix.

Le test switch

Exemple :

```
switch(operateur) {
case '+':
  res = op1 + op2; break;
case '-':
  res = op1 - op2; break;
case '*':
  res = op1 * op2; break;
case '/':
  res = op1 / op2; break;
default :
  printf("inconnu\n");
}
```

- Equivalent avec des **if** :

```
if(operateur == '+')
  res = op1 + op2;
else
  if(operateur == '-')
    res = op1 - op2;
  else
    if(operateur == '*')
      res = op1 * op2;
    else
      if(operateur == '/')
        res = op1 / op2;
      else
         printf("inconnu\n");
```

Le test switch

Plusieurs cas ensembles :

```
switch(nombre) {
case 4 :
  printf("est un nombre pair et de plus ");
case 1 :
case 9 :
  printf("est une racine parfaite\n"); break;
case 2 :
case 6 :
case 8 :
case 10 :
  printf("est un nombre pair\n"); break;
default :
  printf("est ni pair, ni une racine parfaite\n");
}
```

Exercice 2.5 : Calculatrice 5

Réaliser une calculatrice permettant d'effectuer les opérations +, -, *, / et % avec l'instruction **`switch`**.

Le langage C

Les instructions itératives

La boucle For

```
for(int i=ini; i<=fin; i++) {
      instructions
}
```

Répète les `instructions` un certain nombre de fois

Exemple :

```
for(int i=1; i<=3; i++) {
  printf("Passage n°%d\n", i);
}
```

```
Passage n°1
Passage n°2
Passage n°3
```

Opérateur d'évaluation séquentielle

```
for(i=ini, j=fin; i<=fin; i++, j--) {
      instructions
}
```

Permet d'avoir plusieurs instructions d'amorçage ou plusieurs instructions de relance

Exemple :

```
int i, j;
for(i=1, j=5; i<=j; i++, j--) {
  printf("i = %d et j = %d\n", i, j);
}
```

```
i = 1 et j = 5
i = 2 et j = 4
i = 3 et j = 3
```

La boucle While

```
amorçage
while(condition)  {
      instructions
      relance
}
```

Répète les **instructions** tant que la **condition** est « vraie »

Les instructions d'**amorçage** permettent d'initialiser la variable sur laquelle porte la **condition**.

La **relance** permet de mettre à jour la variable sur laquelle porte la **condition**.

La boucle While

```
//Amorçage
printf("Combien font 8 × 7 ?\n");
scanf("%d", &saisie);
                //Condition
while(saisie != 56) {
  //Traitement
  printf("Vous vous êtes trompe ! Réessayez…\n");
  //Relance
  scanf("%d", &saisie);
}
```

```
Combien font 8 × 7 ?
42
Vous vous êtes trompe ! Réessayez…
58
Vous vous êtes trompe ! Réessayez…
56
```

La boucle Do … While

```
do {
   (ré)affectation de la variable de
   condition instructions
} while(condition);
```

Exécute au moins une fois la boucle

```
do {
   printf("Un nombre pair, svp\n");
   scanf("%d", &nb);
} while(nb%2); // équivalent à nb%2!=0
```

Le langage C

Les instructions peu orthodoxes

Les instructions peu orthodoxes…

break

continue

goto

break

Permet de quitter la boucle dans laquelle nous nous trouvons :

```
int saisie;
for(int nbEssai=0; nbEssai<3; nbEssai++) {
  printf("code carte bleu ?\n");
  scanf(" %d", &saisie);
  if(saisie==PWD)
    break;
}
if(saisie==PWD)
  printf("Bon mot de passe\n");
else
  printf("Carte confisquee\n");
```

continue

Permet de terminer l'itération en cours sans quitter la boucle dans laquelle nous nous trouvons :

```
int beuzeu = 7;
for(int i=1; i<100; i++) {
  if(i%beuzeu==0 || i%10==7 || i/10==7) {
    printf("Beuzeu\n");
    continue;
  }
  printf("%d\n", i);
}
```

goto

- Permet de sauter vers un autre bout de code sans retour après.


```c
int main() {
  //...
  switch(operateur) {
  //...
  case '/':
    if(op2==0)
      goto erreurDiv;
    res = op1 / op2; break;
  default :
    goto erreurOp;
  }
  printf("%d %c %d = %d\n", op1, operateur, op2, res);
  return 0;
erreurDiv:
  printf("La division par 0 est interdite\n"); return 1;
erreurOp:
  printf("Operateur inconnu\n"); return 2;
}
```

Exercice 4.1

Réécrire les 3 codes précédents sans les instructions « peu orthodoxes »

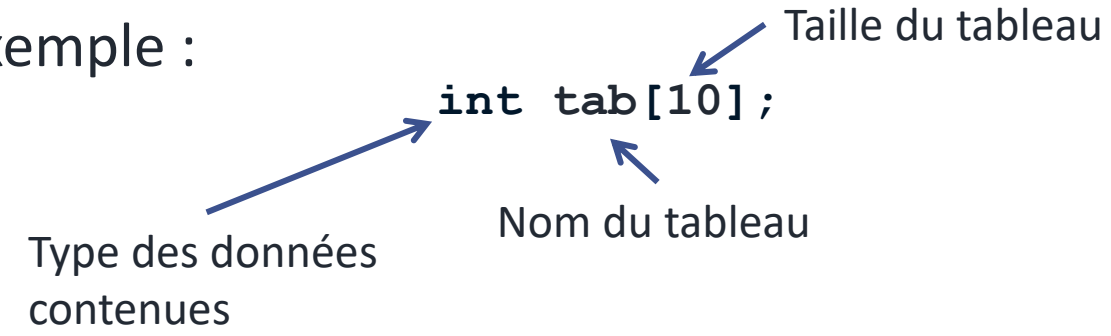
Le langage C

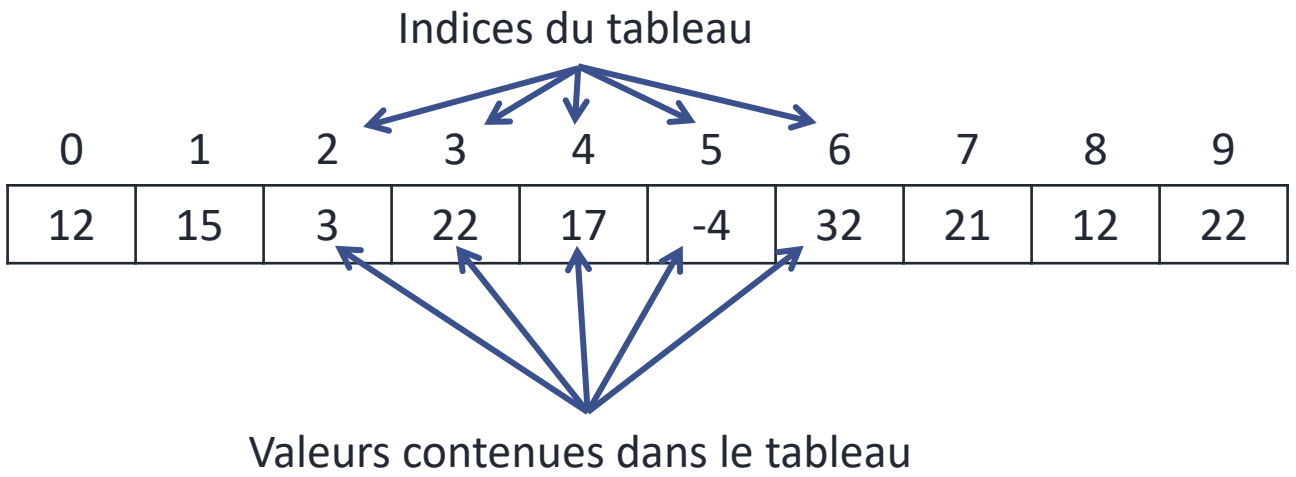
Les tableaux

Les tableaux

Un tableau est un ensemble ordonné de valeurs d'un type de données.

Exemple :





Déclaration d'un tableau

```
type nomTableau[TAILLE];
```

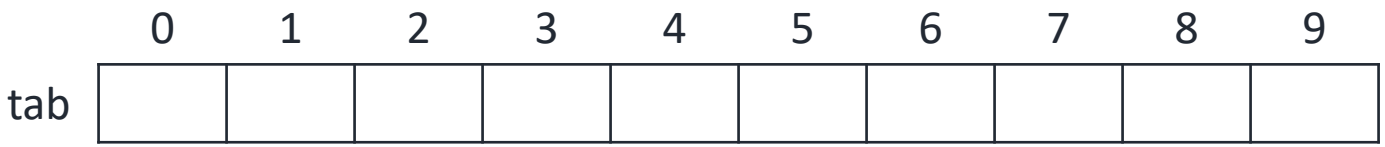
Type des données contenu dans le tableau

Nom du tableau

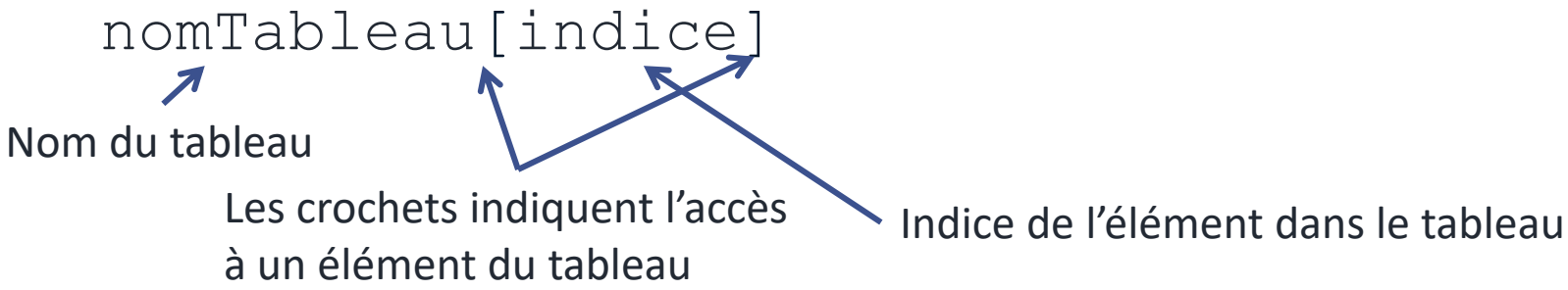
Taille du tableau

Exemple :

```
int tab[10];
```



Accès à un élément d'un tableau



Exemple accès en écriture

```
tab[3] = 42 // affectation de la valeur 42
            // dans la case n°3 du tableau tab
```

Exemple accès en lecture

```
printf("%d\n", tab[3]) // affichage de la valeur
                       // contenue dans la case n°3 de tab
```

Attention, c'est à vous de gérer la taille du tableau !

Initialisation d’un tableau

Lorsque vous déclarez un tableau, vous réservez juste des cases mémoires

Les valeurs de ces cases mémoires ne sont pas réinitialisées

Pour initialiser :

- À la déclaration
- Avec une boucle

Initialisation d'un tableau
À la déclaration

```
// Initialisation de 10 valeurs
int tab1[10] = {12, 15, 3, 22, 17, -4, 32, 21, 12, 22};
// La taille n'est pas nécessaire du coup
int tab2[] = {12, 15, 3, 22, 17, -4, 32, 21, 12, 22};
```

0	1	2	3	4	5	6	7	8	9
12	15	3	22	17	-4	32	21	12	22

```
// Si la taille est plus grande que le nombre de valeurs
// complétion avec des zéros
int tab3[10] = {12, 15, 3, 22, 17, -4};
```

0	1	2	3	4	5	6	7	8	9
12	15	3	22	17	-4	0	0	0	0

```
// Si la taille est plus grande que le nombre de valeurs
// erreur à la compilation
```

Initialisation d'un tableau Avec une boucle

```
// Déclaration du tableau
int tab[10];
// Initialisation avec les nombres de 1 à 10
for(int i=0; i<10; i++)
    tab[i] = i+1;
```

Exercice 5.1 : Initialisation d'un tableau Avec une boucle

```
// Déclaration du tableau
int tab[10];
// Initialisation avec les dix premières puissances de deux
int j=1;
for(int i=0; i<10; i++) {
    tab[i] = j;
    j<<=1;
}
// Initialisation avec les dix premiers nombres premiers
int nb=2;
int i=0;
while(i<10) {
    int j=0;
    while(j<i && nb%tab[j])
        j++;
    if(j==i)
        tab[i++] = nb;
    nb++;
}
```

Initialisation d'un tableau Avec des valeurs aléatoires

```c
#include<stdlib.h> // pour srand() et rand()
#include<time.h>  // pour time()

int main() {
    // Déclaration du tableau
    int tab[10];
    // Initialisation du générateur de nombres aléatoires
    srand(time(NULL));
    for(int i=0; i<10; i++)
        // Initialisation avec une valeur aléatoire comprise
        // entre 0 et 9
        tab[i] = rand()%10;
//...
    return 0;
}
```

Chaîne de caractères
Initialisation

Une chaîne de caractères est un tableau de caractères.

```
char chaine1[] = "Bonjour";
// Équivalent à
char chaine2[8] = {'B', 'o', 'n', 'j', 'o', 'u', 'r'};
// Équivalent à
char chaine3[] = {'B', 'o', 'n', 'j', 'o', 'u', 'r', 0};
// Équivalent à
char chaine4[] = {'B', 'o', 'n', 'j', 'o', 'u', 'r', '\0'};
// Équivalent à
char chaine5[] = {66, 111, 110, 106, 111, 117, 114, 0};

printf("%s %s %s %s %s\n",
                chaine1, chaine2, chaine3 , chaine4, chaine5);
```

Chaîne de caractères
Saisie et Affichage

Saisie

```
printf("Entrez un mot\n");
// Il faut avoir réservé l'espace mémoire nécessaire
// avant la saisie
char mot[100];

// Pas de & devant la variable
scanf(" %s", mot);
```

Affichage

```
printf("%s\n", mot);
```

Chaîne de caractères
Saisie et Affichage

Saisie

```
printf("Entrez une phrase\n");
// Il faut avoir réservé l'espace mémoire nécessaire
// avant la saisie
char message[100];
// Pour considérer uniquement le retour à la ligne
// comme délimiteur (et pas l'espace)
scanf(" %[^\n]", message);
```

Affichage

```
printf("%s\n", message);
```

Exercice 5.2 : MAJUSCULES

Écrire un programme qui transforme un texte saisi par l’utilisateur tout en majuscule et l’affiche.

Même exercice pour uniquement la première lettre de chaque mot

Exercice 5.3 : Palindrome

Un palindrome est une figure de style désignant un texte ou un mot dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche.

Exemples :

- Ici
- Kayak
- Ésope reste ici et se repose
- Karine alla en Irak
- Engage le jeu que je le gagne

Exercice 5.3 : Palindrome

Écrire un programme qui demande une chaine de caractères à l'utilisateur et lui indique si celle-ci est un palindrome.

- Les caractères non alphabétiques sont ignorés

La bibliothèque string.h

Taille d'une chaîne de caractères

```
printf("%d\n", strlen(chaine1));
```

Affectation d'une valeur (copie)

```
strcpy(chaine1, "Salut");
```

Concaténation

```
strcat(chaine1, " les ingés 3iL");
```

Comparaison

```
if(!strcmp(chaine2, chaine3))
        printf("Les chaines de caracteres sont identiques");
```

Déclaration d'un tableau à 2 dimensions

```
type tablo[DIM1][DIM2]
```

Type des données contenu dans le tableau

Nom du tableau

Taille du tableau

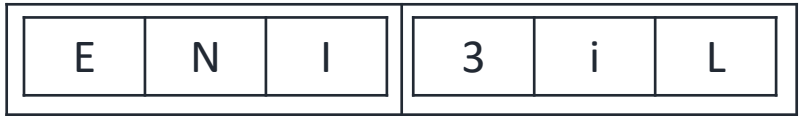
Exemple :

```
int tab[3][10]
```

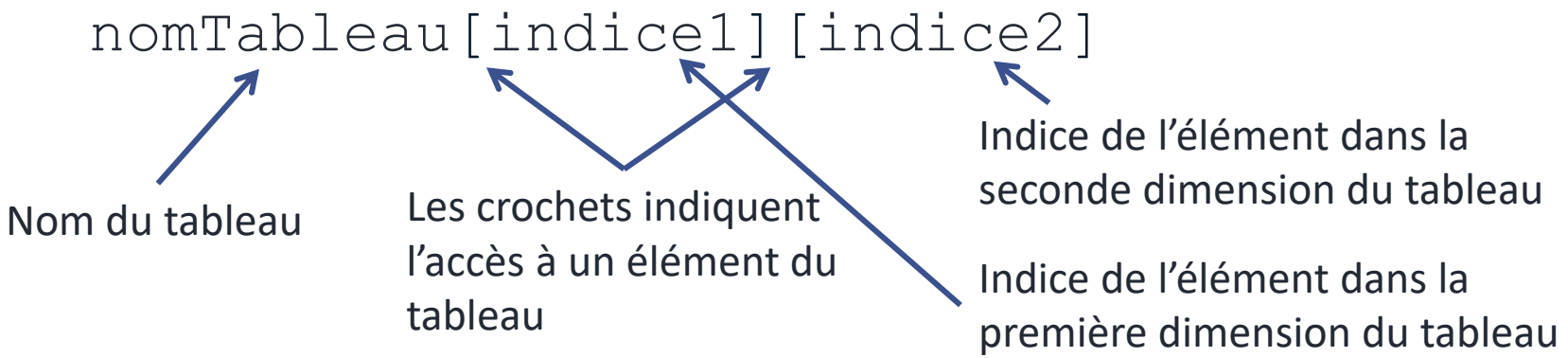
Un tableau à deux dimensions est un tableau de tableaux

```
char t[2][3]
```

E	N	I
3	i	L



Accès à un élément d'un tableau à 2 dimensions



Exemple accès en écriture

```
tab[2][8] = 4 // affectation de la valeur 4
// dans la case de coordonnée (2,8) de tab
```

Exemple accès en lecture

```
printf("%d\n",tab[3][0]) // affichage de la valeur
// contenue dans la case (3,0) de tab
```

Le langage C

Les types évolués

Énumérations

Les énumérations sont des types définissant un ensemble de constantes qui portent un nom.

Elles servent à rajouter du sens à de simples numéros, à définir des variables qui ne peuvent prendre leur valeur que dans un ensemble fini de valeurs possibles identifiées par un nom symbolique.

Une énumération est un type entier

Énumérations

```c
enum titre {
  Madame = 1,
  Monsieur
};

int main() {
  enum titre votreTitre;
  printf("Quel est votre titre ?\n"
         "1. Madame\n2. Monsieur\n");
  scanf(" %d", &votreTitre);
  while(votreTitre < 1 || votreTitre > 2) {
    printf("Saisie incorrecte, ressaisissez, svp\n");
    scanf(" %d", &votreTitre);
  }
  switch(votreTitre) {
  case Monsieur : printf("Bonjour Monsieur\n"); break;
  case Madame : printf("Bonjour Madame\n"); break;
  }
  return 0;
}
```

Structures

Un agrégat d’éléments hétérogènes

Une adresse est composée

- d’un numéro ;
- d’un nom de rue ;
- d’un code postal et
- d’une ville

Structures

Déclaration d'une structure

```
struct adresse {
        int numero;
        char rue[40];
        int cp;
        char ville[40];
};
```

Déclaration d'une variable de ce type

```
struct adresse localisationENI;
```

Structures

Affectation de valeurs à une structure

```
localisationENI.cp = 44800;
strcpy(localisationENI.ville, "Saint-Herblain");
strcpy(localisationENI.rue, "Benjamin Franklin");
localisationENI.numero = 2;
```

Récupération des valeurs contenues dans une structure

```
printf("%d rue %s\n%05d %s\n",
       localisationENI.numero,
       localisationENI.rue,
       localisationENI.cp,
       localisationENI.ville);
```

Structures

```c
#include <stdio.h>
#include <string.h>
struct adresse {
  int numero;
  char rue[40];
  int cp;
  char ville[40];
};

int main() {
  struct adresse locENI;
  locENI.cp = 44800;
  strcpy(locENI.ville, "Saint-Herblain");
  strcpy(locENI.rue, "Benjamin Franklin");
  locENI.numero = 2;
  printf("%d rue %s\n%05d %s\n",
        locENI.numero, locENI.rue, locENI.cp, locENI.ville);
  return 0;
}
```

Structures

Une structure peut contenir des types de base, des tableaux, des énumérations et des structures…

```
struct personne {
        enum titre sonTitre;
        char sonNom[40];
        struct adresse sonAdresse;
        int sonTelephone;
};
```

types personnalisés

L'instruction `typedef` permet de définir de nouveaux types :

```
typedef declartionDuType nomNouveauType;
```

Exemple :

```
typedef int entier;

int main() {
        entier i = 40;
        printf("%d\n", i);
        return 0;
};
```

types personnalisés

Pour une énumération

```
typedef enum nomEnumeration {
  Madame = 1,
  Monsieur
} titre;

// ou bien
typedef enum {
  Madame = 1,
  Monsieur
} titre;
```

types personnalisés

```c
typedef enum {
  Madame = 1,
  Mademoiselle,
  Monsieur
} titre;

int main() {
  titre votreTitre;
  printf("Quel est votre titre ?\n"
         "1. Madame\n2. Monsieur\n");
  scanf(" %d", &votreTitre);
  while(votreTitre < 1 || votreTitre > 3) {
    printf("Saisie incorrecte, ressaisissez, svp\n");
    scanf(" %d", &votreTitre);
  }
  switch(votreTitre) {
  case Monsieur : printf("Bonjour Monsieur\n"); break;
  case Madame : printf("Bonjour Madame\n"); break;
  }
  return 0;
}
```

titre est maintenant un type
(plus besoin d'écrire enum)

types personnalisés

Pour une structure

```
typedef struct nomStructure {
  int numero;
  char rue[40];
  int cp;
  char ville[40];
} adresse;

// ou bien
typedef struct {
  int numero;
  char rue[40];
  int cp;
  char ville[40];
} adresse;
```

types personnalisés

```c
#include <stdio.h>
#include <string.h>
typedef struct {
  int numero;
  char rue[40];
  int cp;
  char ville[40];
} adresse;

int main() {
  adresse locENI;
  locENI.cp = 44800;
  strcpy(locENI.ville, "Saint-Herblain");
  strcpy(locENI.rue, "Benjamin Franklin");
  locENI.numero = 2;
  printf("%d rue %s\n%05d %s\n",
        locENI.numero, locENI.rue, locENI.cp, locENI.ville);
  return 0;
}
```

adresse est maintenant un type
(plus besoin d'écrire struct)

V – 1.4
V – 2.3
VI – 3.4

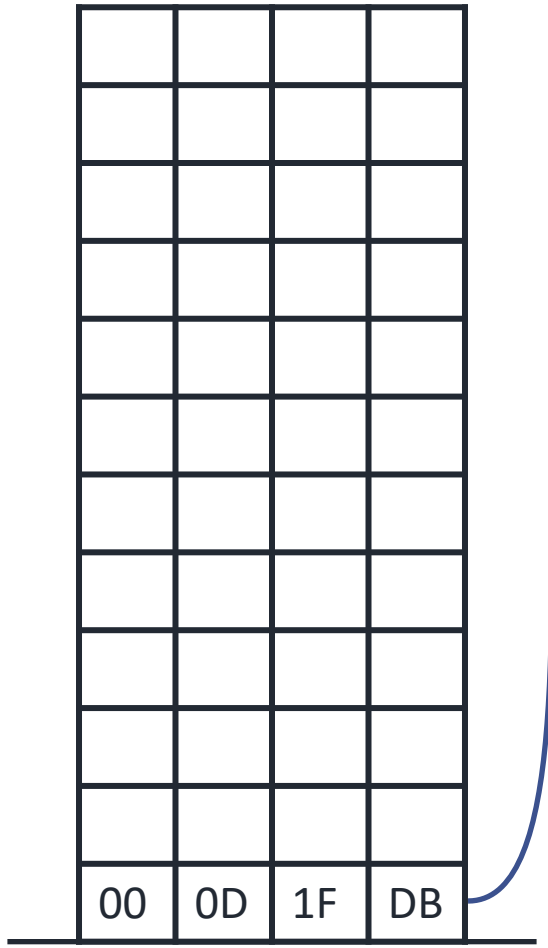
Le langage C

Les pointeurs

Les pointeurs

Un pointeur est une variable contenant l'adresse d'une autre variable.

Les pointeurs
En mémoire…



00 0D 1F D0	00	00	00	00	
00 0D 1F D4	00	00	00	00	
00 0D 1F D8	00	00	00	00	
00 0D 1F DB	00	00	26	39	9785
00 0D 1F E0	00	00	00	00	
00 0D 1F E4	00	00	00	00	
00 0D 1F E8	00	00	00	00	
00 0D 1F EB	00	00	00	00	
00 0D 1F F0	00	00	00	00	
00 0D 1F F4	00	00	00	00	
00 0D 1F F8	00	00	00	00	
00 0D 1F FB	00	00	00	00	

Déclaration d'un pointeur

Il faut ajouter une * pour indiquer la déclaration d'un pointeur vers un entier plutôt que la déclaration d'une variable entière

```
int a; // déclaration d'une variable de type entier
int* pi; // déclaration d'un pointeur vers un entier
```

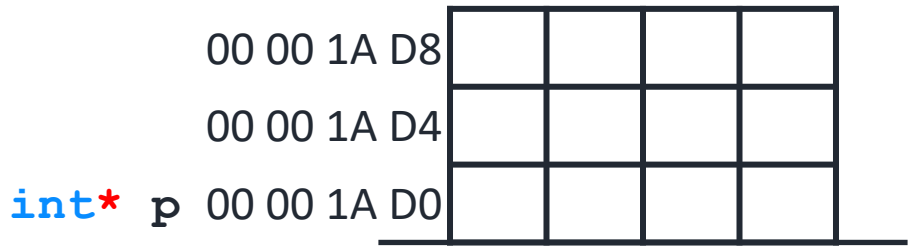
Même chose pour les autres types

```
char* pc; // déclaration d'un pointeur vers un caractère
float* pf; // déclaration d'un pointeur vers un réel
```

Déclaration d'un pointeur

Il faut ajouter une * pour indiquer la déclaration d'un pointeur vers un entier

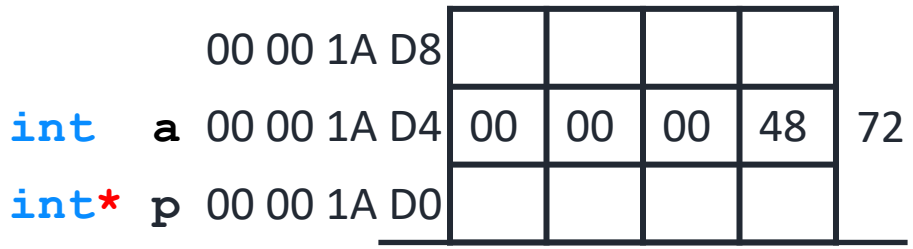
```
int* p;
```



Déclaration d'un pointeur

Il faut ajouter une * pour indiquer la déclaration d'un pointeur vers un entier

```
int* p;
int a = 72;
```



Adresse d'une variable

Il faut précéder le nom de la variable par le caractère &

```
int* p;
int a = 72;
p = &a;
```

Accès à la valeur pointée

Le caractère * placé devant une variable permet d'accéder à la valeur pointée

```
int* p;
int a = 72;
p = &a;
printf("Valeur pointee par p : %d\n", *p); // 72
```

Accès à la valeur pointée

Il est ensuite possible de modifier la variable comme d'habitude

```
int* p;
int a = 72;
p = &a;
printf("Valeur pointee par p : %d\n", *p); // 72
*p += 10;
printf("Valeur pointee par p : %d\n", *p); // 82
printf("Valeur de a: %d\n", a); // 82
```

Affichage d'une adresse

Il faut utiliser le spécifieur **%p** pour afficher l'adresse pointée

```
int* p;
int a = 72;
p = &a;
printf("Valeur pointee par p : %d\n", *p); // 72
*p += 10;
printf("Valeur pointee par p : %d\n", *p); // 82
printf("Valeur de a: %d\n", a); // 82
printf("Adresse pointee par p : %p\n", p); // 0x000D1FD4
```

Résumé

Déclaration

* : pointeur

exemple : `int* p;`

Opérateur

& se lit « adresse de »

exemple : `p = &a;`

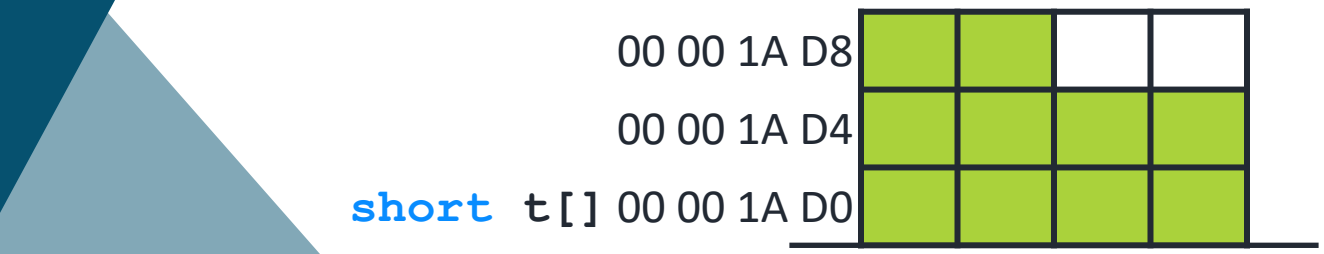
* se lit « valeur pointée par »

exemple : `*p += 10;`

Tableaux et pointeurs

Un tableau est un pointeur qui pointe sur la première valeur du tableau

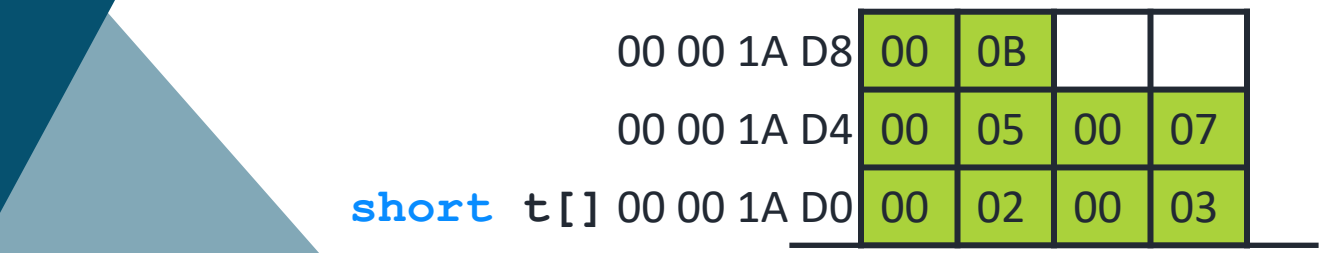
```
short t[5];
printf("%p\n", t); // 0x00001AD0
```



Tableaux et pointeurs

Un tableau est un pointeur qui pointe sur la première valeur du tableau

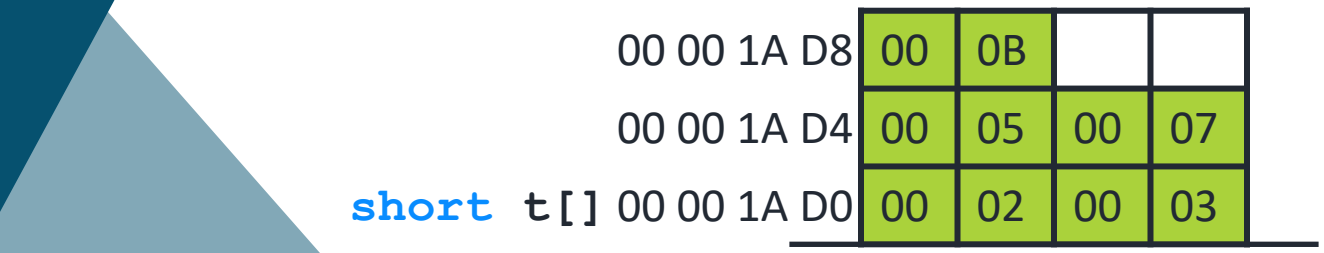
```
short t[5];
short nb=2;
int i=0;
while(i<5) {
    int j=0;
    while(j<i && nb%tab[j])
        j++;
    if(j==i)
        tab[i++] = nb;
    nb++;
}
```



Tableaux et pointeurs

Un tableau est un pointeur qui pointe sur la première valeur du tableau

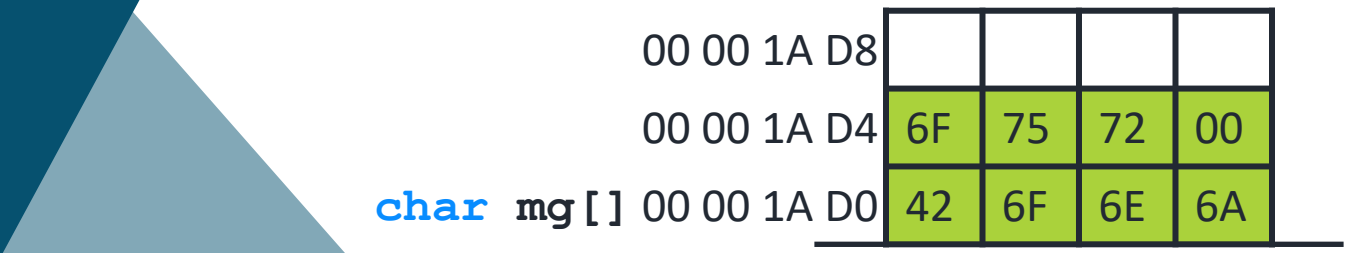
```
t[i]
// équivaut à
*(t+i)
```



Chaînes de caractères et pointeurs

Une chaîne de caractères est un pointeur qui pointe sur le premier caractère

```
char mg[] = "Bonjour";
printf("%p\n", mg); // 0x00001AD0
```



Structures et pointeurs

```c
#include <stdio.h>
#include <string.h>
typedef struct {
  int numero;
  char rue[40];
  int cp;
  char ville[40];
} adresse;
int main() {
  adresse locENI;
  adresse* ploc = &locENI;
  (*ploc).cp = 44800;
  strcpy((*ploc).ville, "Saint-Herblain");
  strcpy((*ploc).rue, "Benjamin Franklin");
  (*ploc).numero = 2;
  printf("%d rue %s\n%05d %s\n",
    (*ploc).numero, (*ploc).rue, (*ploc).cp, (*ploc).ville);
  return 0;
}
```

Structures et pointeurs

```c
#include <stdio.h>
#include <string.h>
typedef struct {
  int numero;
  char rue[40];
  int cp;
  char ville[40];
} adresse;
int main() {
  adresse locENI;
  adresse* ploc = &locENI;
  ploc->cp = 44800;
  strcpy(ploc->ville, "Saint-Herblain");
  strcpy(ploc->rue, "Benjamin Franklin");
  ploc->numero = 2;
  printf("%d rue %s\n%05d %s\n",
          ploc->numero, ploc->rue, ploc->cp, ploc->ville);
  return 0;
}
```

Exercice 7.1

Manipulation d’une valeur à travers un pointeur

- Déclarer une variable de type réel et un pointeur vers cette variable.
- Modifier la valeur de la variable en passant par le pointeur.
- Afficher la valeur, l'adresse mémoire pointée par le pointeur et l'adresse mémoire de la variable de type pointeur.

Exercice 7.2

Pointeurs et tableaux

- Créer et initialiser un tableau d'entiers de 10 cases.
- Afficher l'adresse mémoire de chaque case.
- Faire de même avec un tableau de char, de short et de double.
- Que remarquez-vous ?

Exercice 7.3

Structures et pointeurs

- Créer une structure avec un champ entier nb et un champ caractère car.
- Faire saisir un texte à l'utilisateur.
- Indiquer le nombre de fois que chaque lettre a été saisie

Exemple : bonjour -> b : 1 o : 2 n : 1 j : 1 u : 1 r : 1

Le langage C

Les fonctions

Déclaration d'une fonction

```
typeRetour nomFonction(listeParametres) {
   instructions
   return valeur
}
```

Exemple :

```
float puissance(float a, int n) {
   float p = 1;
   for(int i=1; i<=n; i++) {
      p *= a;
   }
   return p;
}
```

Attention, une fonction doit être déclarée avant son utilisation

Appel d’une fonction

```
float puiss;
puiss = puissance(10, 6);
```

Exemple

```c
// calcule a puissance n
float puissance(float a, int n) {
   float p = 1;
   for(int i=1; i<=n; i++) {
      p *= a;
   }
   return p;
}

// Saisie une valeur et un exposant et affiche
valeur^exposant
int main() {
   printf("Entrez une valeur\n");
   float val;
   scanf(" %f", &val);
   printf("Entrez l'exposant\n");
   int exp;
   scanf(" %d", &exp);
   printf("%f^%d = %f\n", val, exp, puissance(val, exp));
}
```

Exemple

```c
float puissance(float a, int n);

// Saisie une valeur et un exposant et affiche
valeur^exposant
int main() {
   printf("Entrez une valeur\n");
   float val;
   scanf(" %f", &val);
   printf("Entrez l'exposant\n");
   int exp;
   scanf(" %d", &exp);
   printf("%f^%d = %f\n", val, exp, puissance(val, exp));
}

// calcule a puissance n
float puissance(float a, int n) {
   float p = 1;
   for(int i=1; i<=n; i++) {
      p *= a;
   }
   return p;
}
```

Les procédures

Une procédure est une fonction qui ne renvoie rien (void)

```
// Affiche un caractère plusieurs fois
void afficheNFois(char carac, int n) {
   for(int i=1; i<=n; i++) {
       printf("%c", carac);
   }
   printf("\n", carac);
}

// Test la méthode afficheNFois
int main() {
   printf("Entrez un caractere\n");
   char symbole;
   scanf(" %c", &symbole);
   printf("Entrez un nombre\n");
   int nb;
   scanf(" %d", &nb);
   afficheNFois(symbole, nb);
   return 0;
}
```

Passage d'arguments

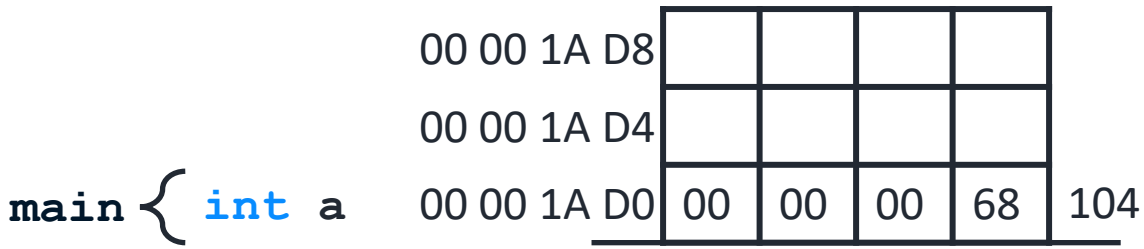
Les arguments d'une fonction sont recopiés

- Pour les types de base (**`int`**, **`char`**, **`short`**, **`long`**, **`float`**, **`double`**…), les énumérations (**`enum`**) et les structures (**`struct`**)
  - copie de la valeur
  - modification d'une copie
- Pour les types pointeurs (**`*`**) [et donc les tableaux]
  - copie du pointeur
  - modification de l'original (sauf si passé avec le mot clé **`const`**, dans ce cas, il est interdit de modifier)

Passage d'arguments
Type de base

```c
void doubleVal(int val) {
   val *= 2;
}

int main() {
   printf("Entrez un entier\n");
   int a;
   scanf(" %d", &a);
   printf("Appel a doubleVal\n");
   doubleVal(a);
   printf("Valeur de a = %d\n", a);
   return 0;
}
```

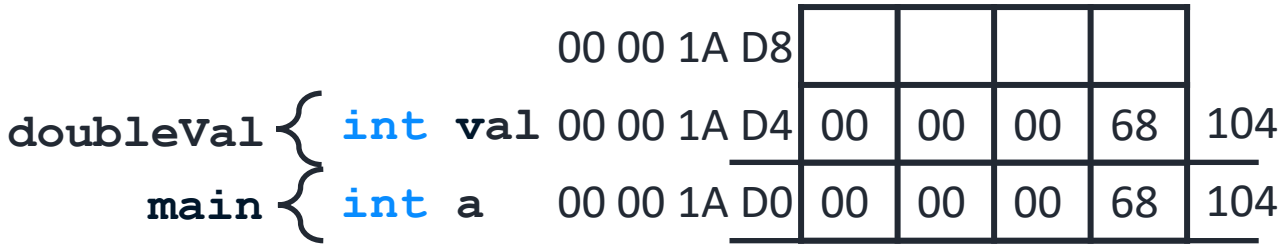
```
Entrez un entier
104
```



Passage d'arguments
Type de base

```c
void doubleVal(int val) {
   val *= 2;
}

int main() {
   printf("Entrez un entier\n");
   int a;
   scanf(" %d", &a);
   printf("Appel a doubleVal\n");
   doubleVal(a);
   printf("Valeur de a = %d\n", a);
   return 0;
}
```

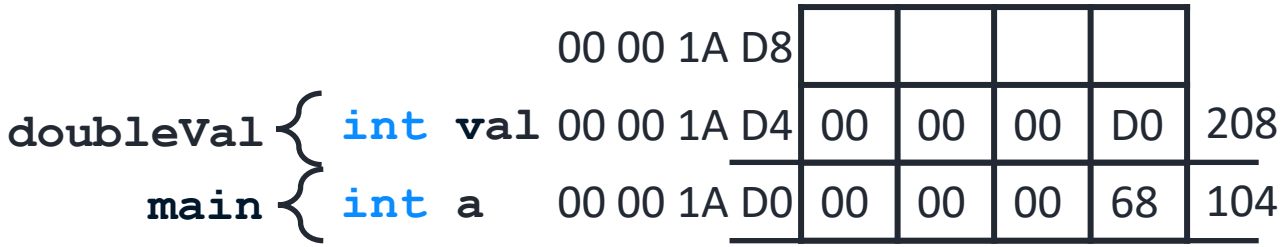
```
Entrez un entier
104
Appel a doubleVal
```



Passage d'arguments
Type de base

```c
void doubleVal(int val) {
   val *= 2;
}

int main() {
   printf("Entrez un entier\n");
   int a;
   scanf(" %d", &a);
   printf("Appel a doubleVal\n");
   doubleVal(a);
   printf("Valeur de a = %d\n", a);
   return 0;
}
```

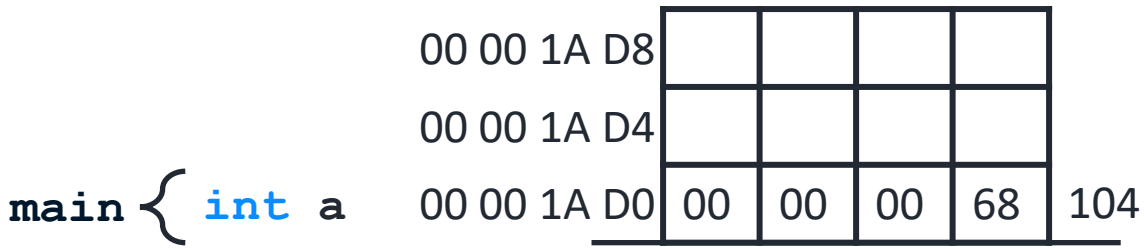
```
Entrez un entier
104
Appel a doubleVal
```



Passage d'arguments
Type de base

```c
void doubleVal(int val) {
   val *= 2;
}

int main() {
   printf("Entrez un entier\n");
   int a;
   scanf(" %d", &a);
   printf("Appel a doubleVal\n");
   doubleVal(a);
   printf("Valeur de a = %d\n", a);
   return 0;
}
```

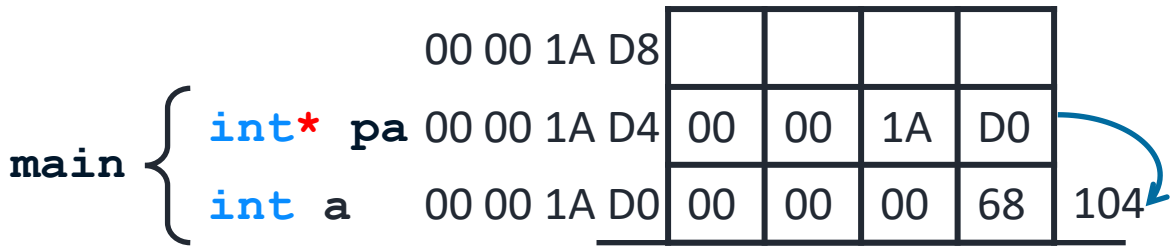
```
Entrez un entier
104
Appel a doubleVal
Valeur de a = 104
```



Passage d'arguments
Type pointeur

```c
void doubleVal(int* pv) {
    *pv *= 2;
}

int main() {
   printf("Entrez un entier\n");
   int a;
   int* pa = &a;
   scanf(" %d", &a);
   printf("Appel a doubleVal\n");
   doubleVal(pa);
   printf("Valeur de a = %d\n", a);
   return 0;
}
```

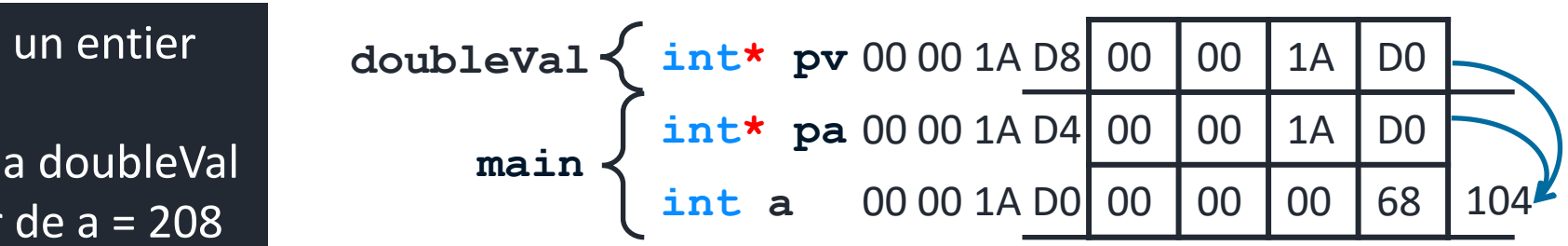
```
Entrez un entier
104
```



Passage d'arguments
Type pointeur

```
void doubleVal(int* pv) {
    *pv *= 2;
}

int main() {
   printf("Entrez un entier\n");
   int a;
   int* pa = &a;
   scanf(" %d", &a);
   printf("Appel a doubleVal\n");
   doubleVal(pa);
   printf("Valeur de a = %d\n", a);
   return 0;
}
```

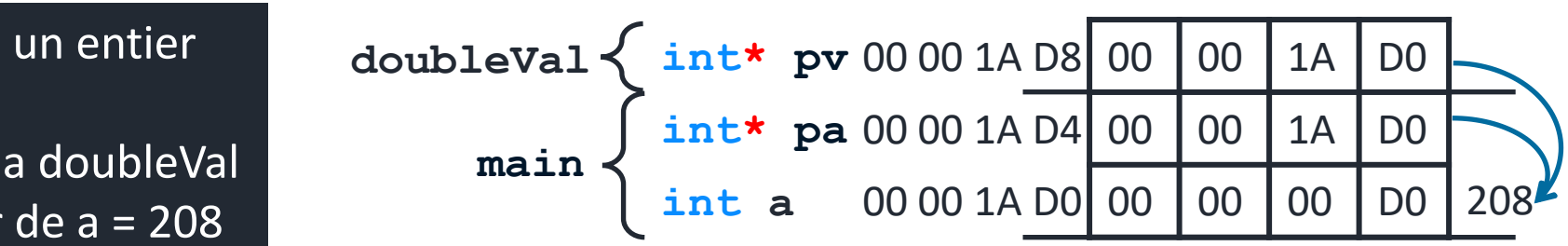
```
Entrez un entier
104
Appel a doubleVal
Valeur de a = 208
```



Passage d'arguments
Type pointeur

```c
void doubleVal(int* pv) {
    *pv *= 2;
}

int main() {
   printf("Entrez un entier\n");
   int a;
   int* pa = &a;
   scanf(" %d", &a);
   printf("Appel a doubleVal\n");
   doubleVal(pa);
   printf("Valeur de a = %d\n", a);
   return 0;
}
```

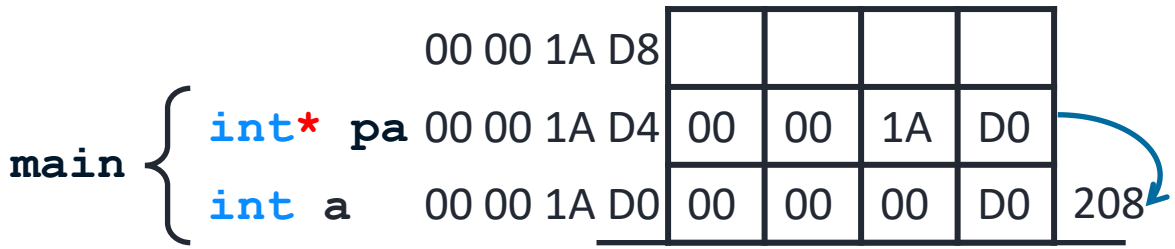
```
Entrez un entier
104
Appel a doubleVal
Valeur de a = 208
```



Passage d'arguments
Type pointeur

```c
void doubleVal(int* pv) {
    *pv *= 2;
}

int main() {
   printf("Entrez un entier\n");
   int a;
   int* pa = &a;
   scanf(" %d", &a);
   printf("Appel a doubleVal\n");
   doubleVal(pa);
   printf("Valeur de a = %d\n", a);
   return 0;
}
```

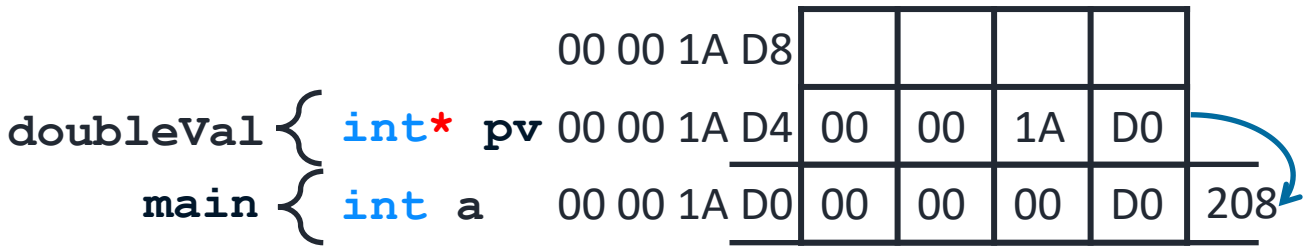
```
Entrez un entier
104
Appel a doubleVal
Valeur de a = 208
```



Passage d'arguments par référence

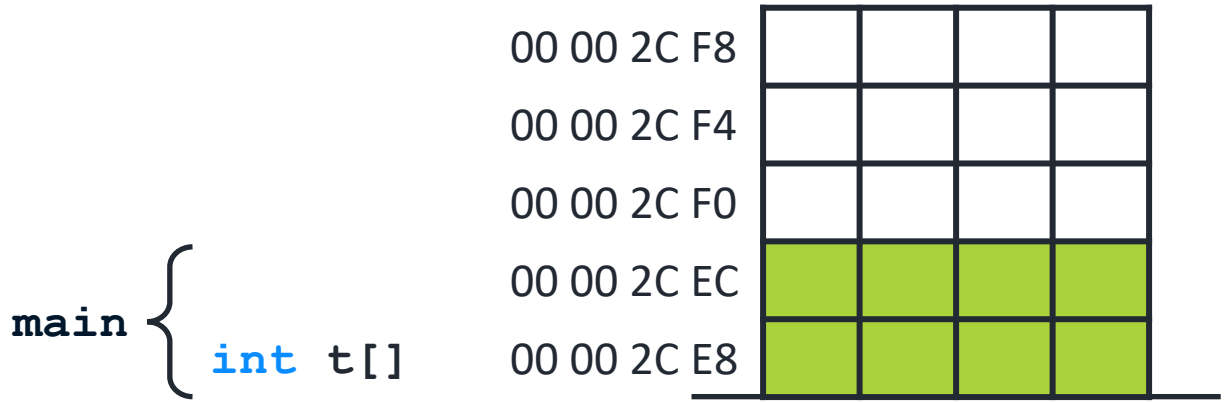
```
void doubleVal(int* pv) {
    *pv *= 2;
}

int main() {
   printf("Entrez un entier\n");
   int a;
   scanf(" %d", &a);
   printf("Appel a doubleVal\n");
   doubleVal(&a);
   printf("Valeur de a = %d\n", a);
   return 0;
}
```

```
Entrez un entier
104
Appel a doubleVal
Valeur de a = 208
```



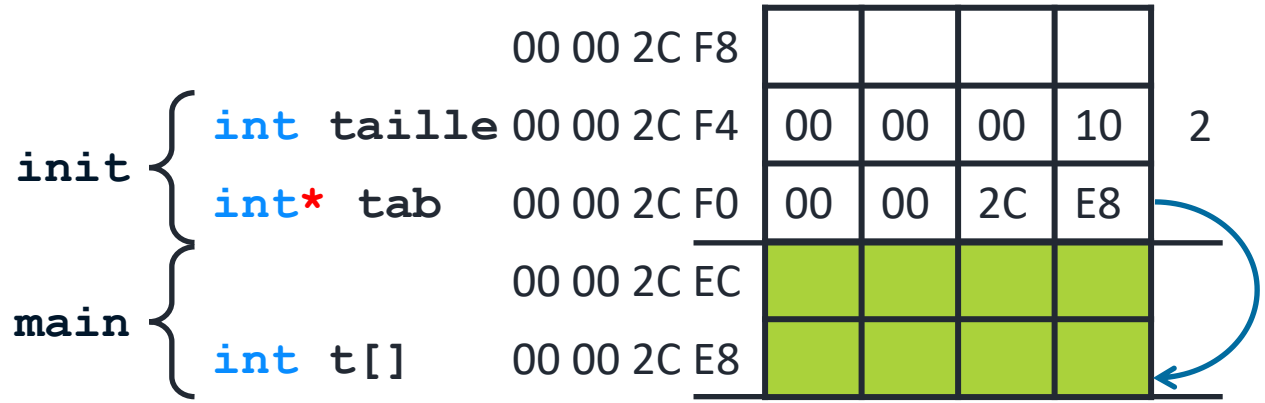
Passage d'arguments
Type tableau

```
void init(int tab[], int taille) {
   for(int i=0; i<taille; i++) {
      tab[i] = i+1;
   }
}

int main() {
   int t[2];
   init(t, 2);
   for(int i=0; i<2; i++)
      printf("%d\n", t[i]);
   return 0;
}
```



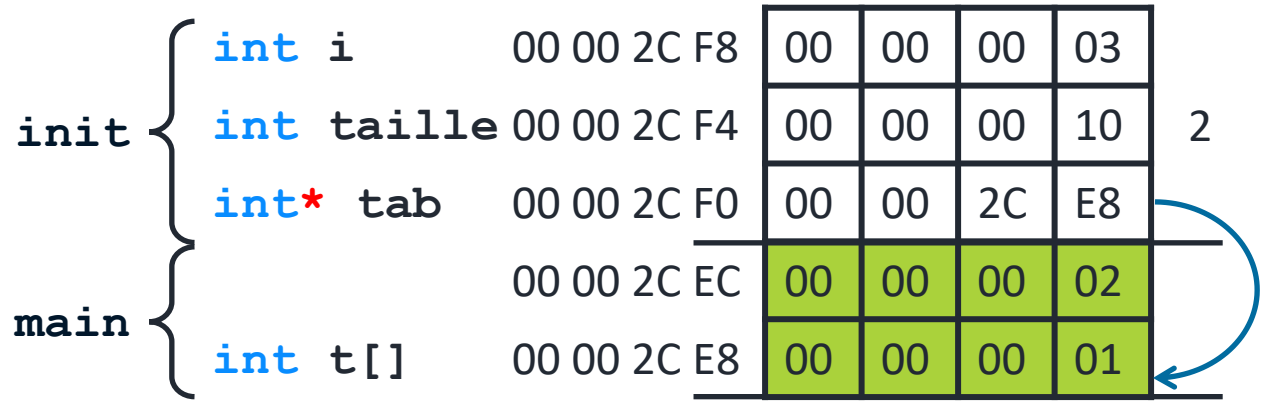
Passage d'arguments
Type tableau

```
void init(int tab[], int taille) {
   for(int i=0; i<taille; i++) {
      tab[i] = i+1;
   }
}

int main() {
   int t[2];
   init(t, 2);
   for(int i=0; i<2; i++)
      printf("%d\n", t[i]);
   return 0;
}
```



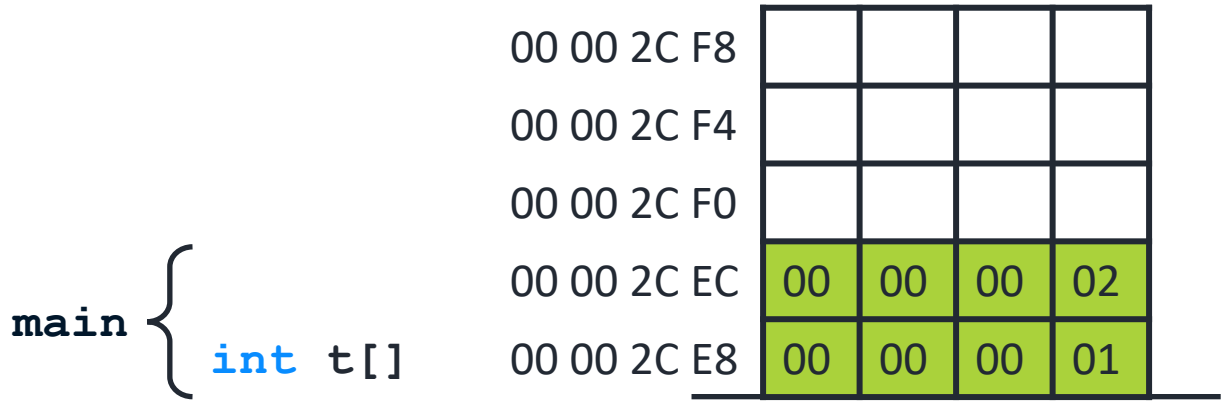
Passage d'arguments
Type tableau

```
void init(int tab[], int taille) {
   for(int i=0; i<taille; i++) {
      tab[i] = i+1;
   }
}

int main() {
   int t[2];
   init(t, 2);
   for(int i=0; i<2; i++)
      printf("%d\n", t[i]);
   return 0;
}
```



Passage d'arguments
Type tableau

```
void init(int tab[], int taille) {
   for(int i=0; i<taille; i++) {
      tab[i] = i+1;
   }
}

int main() {
   int t[2];
   init(t, 2);
   for(int i=0; i<2; i++)
      printf("%d\n", t[i]);
   return 0;
}
```



Passage d'arguments
Argument constant

```
void afficheTableau(int const tab[], int taille) {
   for(int i=0; i<taille; i++)
      printf("%d\n", tab[i]);
}

int main() {
   int tab[10];
   int j=1;
   for(int i=0; i<10; i++) {
      tab[i] = j;
      j<<=1;
   }
   afficheTableau(tab, 10);
   return 0;
}
```

La fonction main

Il est possible de récupérer les arguments de la ligne de commande grâce aux arguments de la fonction main

```
int main(int argc, char* argv[]) {
   printf("nom du programme : %s\n", argv[0]);
   printf("nombre d'arguments : %i\n", argc);
   for(int i=1; i<argc; i++)
      printf("argument n°%i : %s\n", i, argv[i]);
   return 0;
}
```

```
#> ./testMain Bonjour 3iL 01
nom du programme : ./testMain
nombre d'arguments : 4
argument n°1 : Bonjour
argument n°2 : 3iL
argument n°3 : 01
```

Séparation du code en plusieurs fichiers

Signature des fonctions dans un fichier *fct.h*

```
#ifndef FCT_H
#define FCT_H
int doubleVal(int);
#endif
```

Corps des fonctions dans un fichier *fct.c*

```
#include "fct.h"
int doubleVal(int val) {
   return 2*val;
}
```

Fichier principal *test.c*

```
#include "fct.h"
#include <stdio.h>
int main() {
   printf("%i\n", doubleVal(7));
   return 0;
}
```

Séparation du code en plusieurs fichiers

Signature des fonctions dans un fichier *fct.h*

```
#pragma once

int doubleVal(int);
```

Corps des fonctions dans un fichier *fct.c*

```
#include "fct.h"
int doubleVal(int val) {
   return 2*val;
}
```

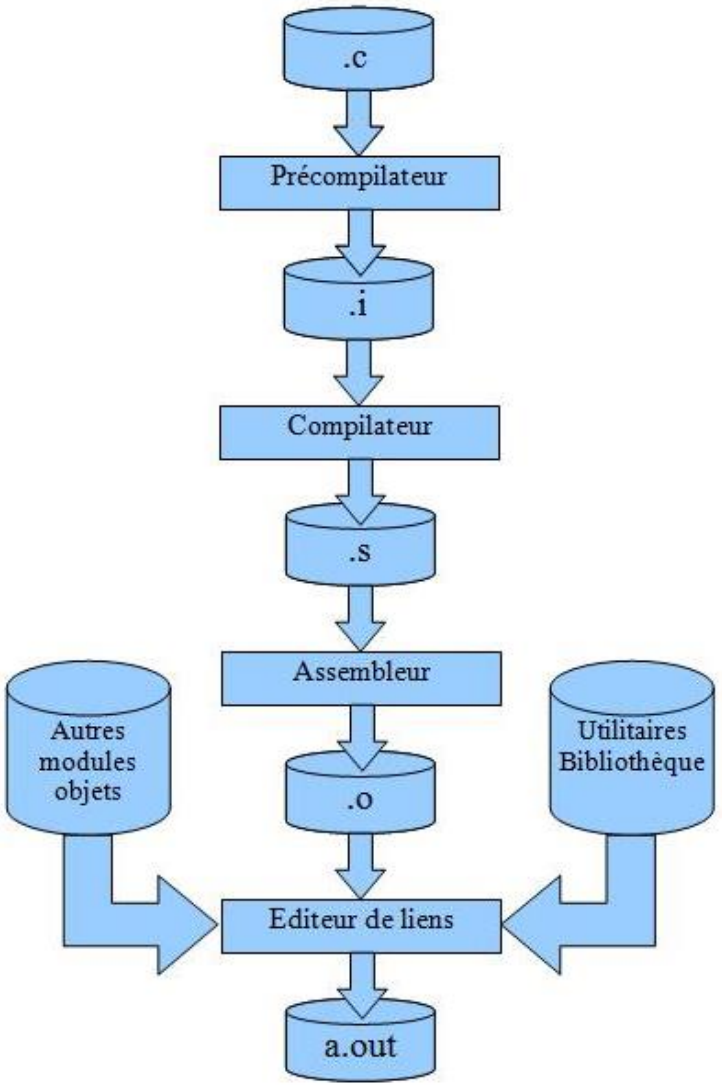
Fichier principal *test.c*

```
#include "fct.h"
#include <stdio.h>
int main() {
   printf("%i\n", doubleVal(7));
   return 0;
}
```

Séparation du code en plusieurs fichiers

Compilation séparée

```
#>  gcc -c fct.c
#>  gcc -c test.c
#>  gcc -o test test.o fct.o
#>  ./test
14
```



IV – 7.10
VI – 4.5
VIII – 8

Le langage C

L'allocation dynamique de mémoire

Allocation dynamique de mémoire

Bibliothèque nécessaire

```
#include <stdlib.h>
```

Allouer de la mémoire

```
void* malloc(size_t taille)
size_t sizeof(type)

void* realloc(void* pointeur, size_t taille)
```

Libérer de la mémoire

```
void free(void* pointeur)
```

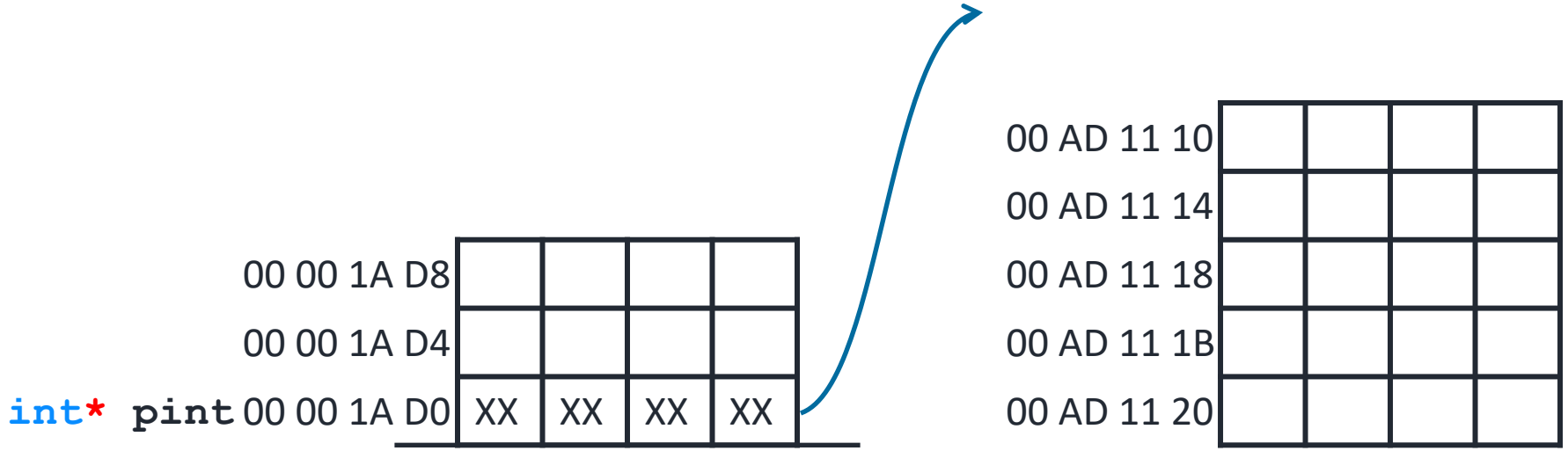
Allocation dynamique de mémoire
Réservation d'une case

```c
#include <stdlib.h>
#include <stdio.h>

int main() {
   // Déclaration d'un pointeur sur entier
   int* pint;
   // Allocation de cases mémoire pour accueillir un entier
   pint = (int*) malloc(sizeof(int));
   if(pint != NULL) {
      *pint = 17;
      printf("%d\n", *pint);
      // Liberation des cases mémoires allouées
      free(pint);
      return 0;
   } else {
      printf("Impossible d'allouer de la memoire\n");
      return 1;
   }
}
```

Allocation dynamique de mémoire
Réservation d'une case

Avant son initialisation un pointeur pointe n'importe où…

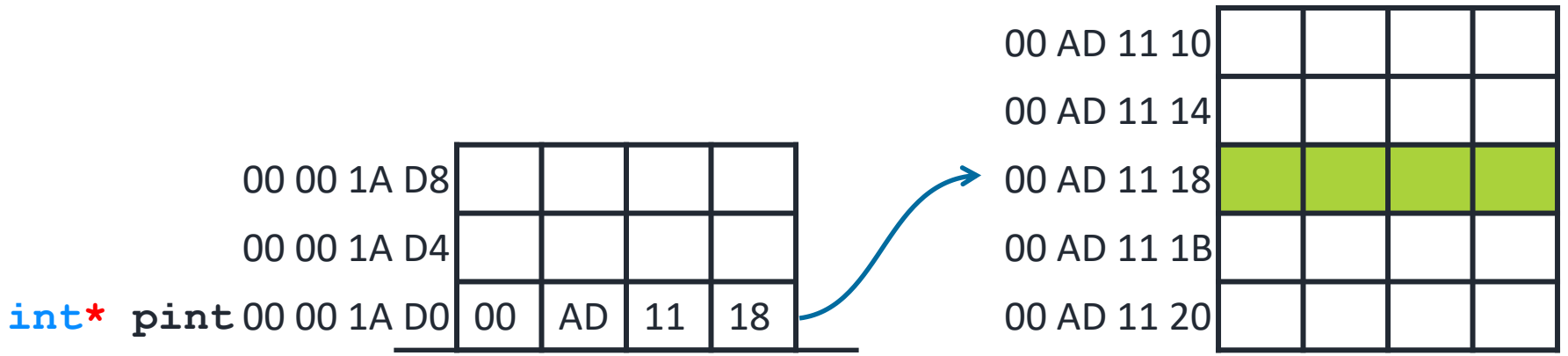
```
// Déclaration d'un pointeur sur entier
int* pint;
```



Allocation dynamique de mémoire
Réservation d'une case

Après son initialisation un pointeur pointe sur le début de la zone mémoire allouée pour accueillir la ou les valeurs

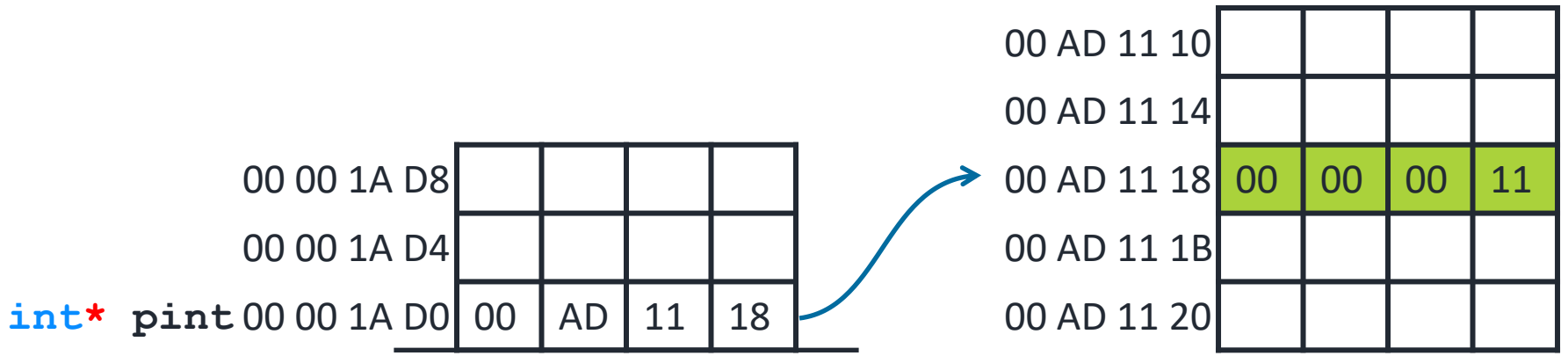
```
// Allocation de cases mémoire pour accueillir un entier
pint = (int*) malloc(sizeof(int));
```



Allocation dynamique de mémoire
Réservation d'une case

Il est alors possible d'utiliser l'espace alloué

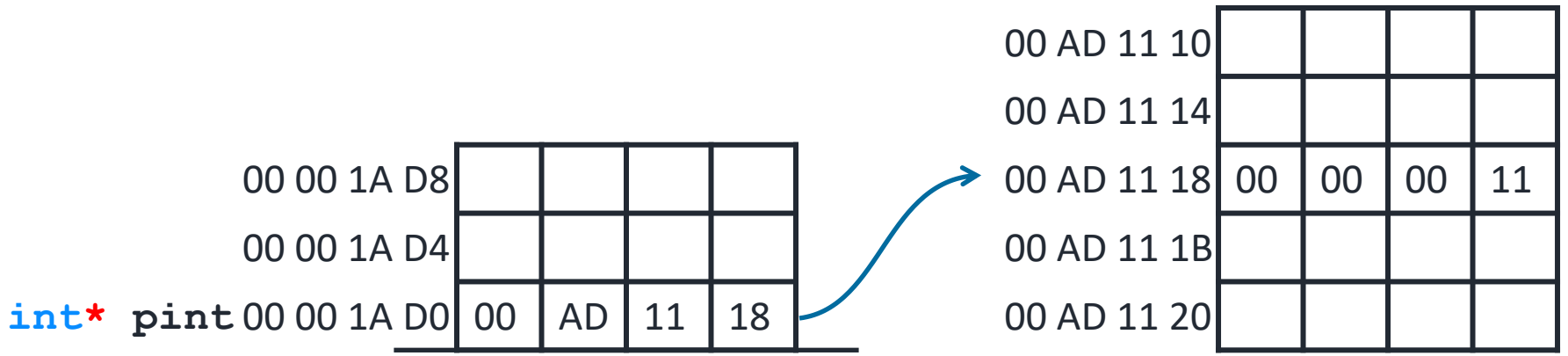
```
*pint = 17;
```



Allocation dynamique de mémoire
Réservation d'une case

L'appel à free() permet d'indiquer que l'espace est à nouveau disponible et peut être utilisé pour une nouvelle allocation

```
// Libération des cases mémoires allouées
free(pint);
```



Allocation dynamique de mémoire Réservation de plusieurs cases

```c
#include <stdlib.h>
#include <stdio.h>
#include "tableaux.h"
int main() {
   // Déclaration d'un pointeur sur entier
   int* tab;
   // Allocation de cases mémoire pour accueillir dix entiers
   tab = (int*) malloc(10*sizeof(int));
   if(tab != NULL) {
      initTableau(tab, 10);
      afficheTableau(tab, 10);
      // Liberation des cases mémoires allouées
      free(tab);
      return 0;
   } else {
      printf("Impossible d'allouer de la memoire\n");
      return 1;
   }
}
```

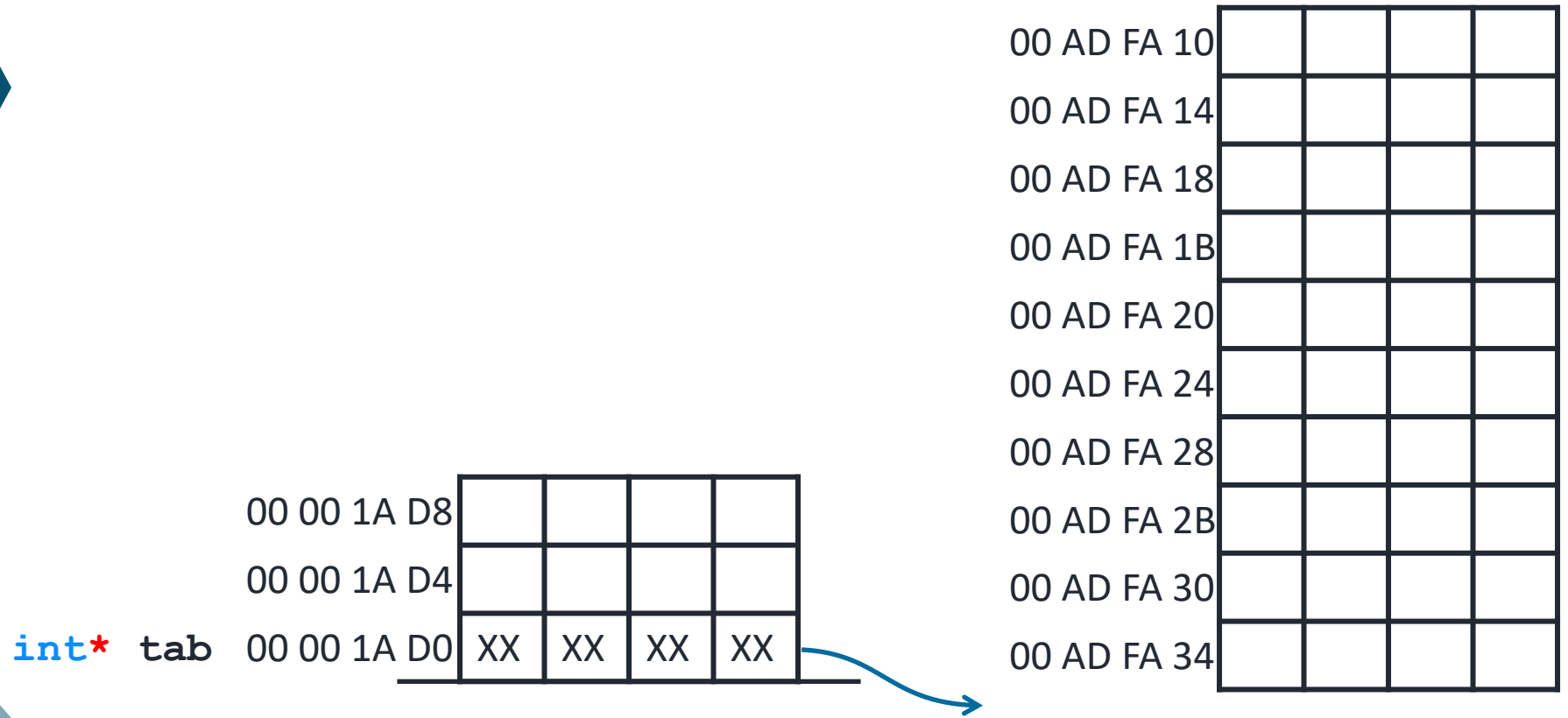
Allocation dynamique de mémoire Réservation de plusieurs cases

```
// fichier tableau.h
#pragma once
void initTableau(int*, int);
void afficheTableau(const int*, int);

// fichier tableau.c
#include "tableaux.h"
#include <stdio.h>
void initTableau(int* tab, int taille) {
   int j=1;
   for(int i=0; i<taille;  i++, j<<=1)
      tab[i] = j;
}
void afficheTableau(const int* tab, int taille) {
   for(int i=0; i<taille;  i++)
      printf("%d\n", tab[i]);
}
```

Allocation dynamique de mémoire Réservation de plusieurs cases

Avant son initialisation un pointeur pointe n'importe où…

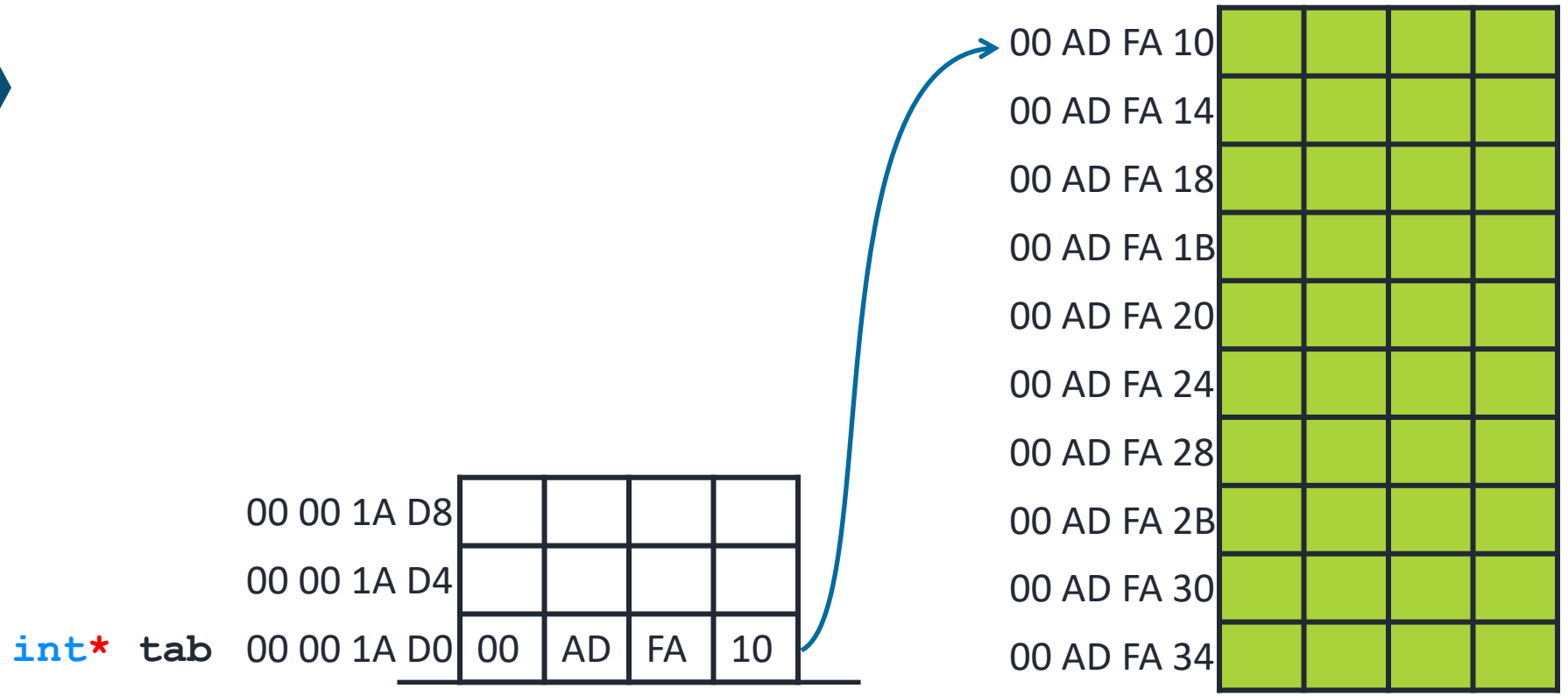
```
// Déclaration d'un pointeur sur entier
int* tab;
```



Allocation dynamique de mémoire Réservation de plusieurs cases

Après son initialisation un pointeur pointe sur le début de la zone mémoire allouée pour accueillir la ou les valeurs

```
// Allocation de cases mémoire pour accueillir dix
// entiers
tab = (int*) malloc(10*sizeof(int));
```



IX – 1.10
IX – 2.7
IX – 3.3

Le langage C

Les fichiers

Fichiers

Ouverture

```
FILE* fichier fopen(const char* chemin, const char* mode);
```

fichier : pointeur vers le fichier

chemin : relatif ou absolu

mode :

chaine	accès	positionnement	si fichier existe	si fichier n'existe pas
r	lecture	début du fichier	ouverture	erreur
w	écriture	début du fichier	réinitialisation	création
a	écriture	fin du fichier	ouverture	création
r+	lect./écr.	début du fichier	ouverture	erreur
w+	lect./écr.	début du fichier	réinitialisation	création
a+	lect./écr.	fin du fichier	ouverture	création

rb, wb, ab, rb+, wb+ et ab+ pour un fichier binaire

Fichiers

Fin du fichier

```
int feof(FILE* fichier);
```

retourne si le curseur est à la fin du fichier

Fermeture

```
int fclose(FILE* fichier);
```

retourne

- **0** si pas d'erreur
- **EOF** sinon

Fichiers

Lecture

```
int fscanf(FILE* fichier, const char* chaine, ...);
```

exemple :

```
fscanf(fic, " %d", &a);
```

Écriture

```
int fprintf(FILE* fichier, const char* chaine, ...);
```

exemple :

```
fprintf(fic, "valeur = %d", a);
```

Fichiers

```c
int main() {
   int tab[] = {66, 111, 110, 106, 111, 117, 114, 0};
   FILE* out = fopen("test.txt", "w");
   if(out == NULL) {
      printf("Probleme d'ouverture du fichier\n");
   } else {
      for(int i=0; i<8; i++) {
         int nbEcrit = fprintf(out, "%d\n", tab[i]);
         if(nbEcrit != sizeof(int))
            printf("Probleme d'ecriture dans le fichier\n");
      }
   }
   fclose(out);
   return 0;
}
```

Fichiers

```c
int main() {
   int* tab = (int*) malloc(8*sizeof(int));
   FILE* in = fopen("test.txt", "r");
   if(in == NULL) {
      printf("Probleme d'ouverture du fichier\n");
   } else {
      for(int i=0; i<8; i++) {
         int nbLu = fscanf(in, " %d", &tab[i]);
         if(nbLu != 1)
            printf("Probleme de lecture dans le fichier\n");
      }
   }
   fclose(in);
   for(int i=0; i<8; i++)
      printf("%d\n", tab[i]);
   free(tab);
   return 0;
}
```

Fichiers binaires

Lecture

```
size_t fread(void* elem,
             size_t taille,
             size_t nbElem,
             FILE* fichier);
```

Écriture

```
size_t fwrite(const void* elem,
              size_t taille,
              size_t nbElem,
              FILE* fichier);
```

Fichiers binaires

```
int main() {
   int tab[] = {66, 111, 110, 106, 111, 117, 114, 0};
   FILE* out = fopen("test.dat", "wb");
   if(out == NULL) {
      printf("Probleme d'ouverture du fichier\n");
   } else {
      int nbEcrit = fwrite(tab, sizeof(int), 8, out);
      if(nbEcrit != 8)
         printf("Probleme d'ecriture dans le fichier\n");
   }
   fclose(out);
   return 0;
}
```

Fichiers binaires

```
int main() {
   int* tab = (int*) malloc(8*sizeof(int));
   FILE* in = fopen("test.dat", "rb");
   if(in == NULL) {
      printf("Probleme d'ouverture du fichier\n");
   } else {
      int nbLu = fread(tab, sizeof(int), 8, in);
      if(nbLu != 8)
         printf("Probleme de lecture dans le fichier\n");
   }
   fclose(in);
   for(int i=0; i<8; i++)
      printf("%d\n", tab[i]);
   free(tab);
   return 0;
}
```

Positionnement dans un fichier

Où en sommes-nous ?

```
long ftell(FILE* fichier);
```

Se repositionner à une position donnée

```
long fseek(FILE* fichier, long pos, int reference);
```

reference :

- SEEK_SET : début du fichier
- SEEK_CUR : position du curseur
- SEEK_END : fin du fichier

Se repositionner au début du fichier

```
void rewind(FILE* fichier);
```

Programmation système

Langage C et Linux

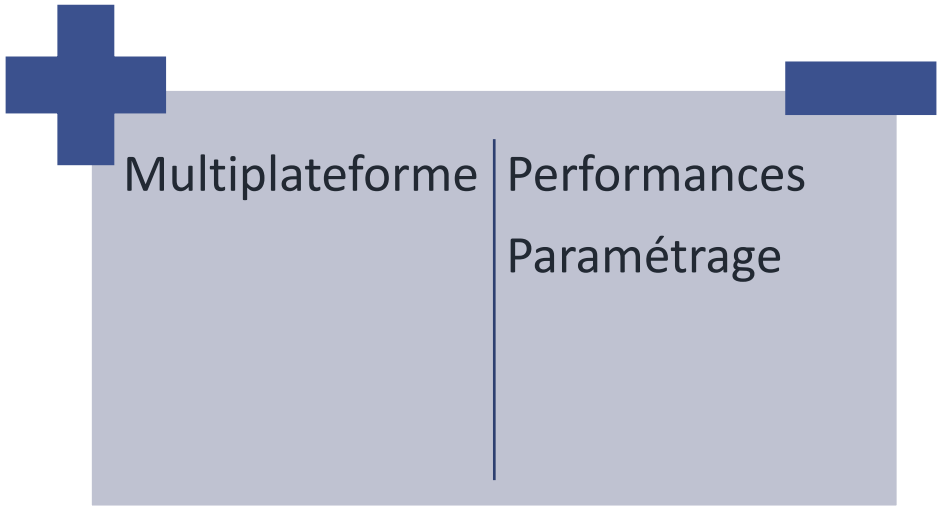
Ressources

Livre "Programmation Système" (nouvelle édition) de Philippe BANQUET – éditions ENI – septembre 2019

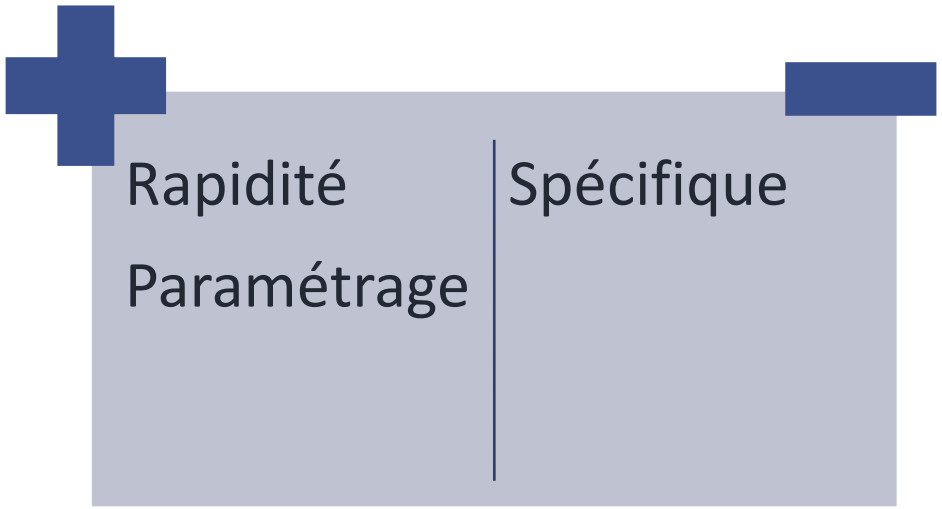
Explications rédigées dans le livre

Les appels système

Appel standard



Appel système



Programmation système

Les fichiers

Ouverture d'un fichier

```
#include <fcntl.h>
int open(const char *path, int oflag [, mode_t mode]);
```

Arguments

path	Chemin d'accès du fichier à ouvrir. S'il s'agit d'un lien symbolique, c'est le fichier cible qui sera ouvert.
oflag	Mode [O_RDONLY, O_WRONLY ou O_RDWR et options d'ouverture [O_APPEND, O_CREAT…]
mode	Permissions en cas de création [S_I{R|W|X}{USR|GRP|OTH}]

Valeur de retour

-1	Erreur, code erreur positionné dans la variable errno
>=0	Succès, descripteur associé au fichier ouvert

Exemple d'ouverture d'un fichier

```
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
// Ouverture en lecture du fichier dont on passe le chemin d'accès

int main(int argc, char *argv[]) {
   int fd = open(argv[1], O_RDONLY);
   if(fd == -1) {
      perror("Open ");
      return errno;
   }
   return 0;
}
```

Exemple de création d'un fichier

```
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>

int main(int argc, char *argv[]) {
  int fd;
  fd = open(argv[1], O_RDONLY|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR|S_IRGRP);
  if(fd == -1) {
     perror("Open ");
     return errno;
  }
  return 0;
}
```

Fermeture d'un fichier

```
#include <unistd.h>
int close(int fd);
```

Arguments

fd	descripteur associé au fichier

Valeur de retour

-1	Erreur, code erreur positionné dans la variable errno
0	Succès

Lecture dans un fichier

```
#include <unistd.h>
ssize_t read(int fd, void* buffer, size_t count);
```

Arguments

fd	Descripteur du fichier, ouvert en lecture ou en lecture/écriture
buffer	Adresse de la zone de stockage des octets lus
count	Nombre maximum d'octets à lire

Valeur de retour

-1	Erreur, code erreur positionné dans la variable errno
0	Fin de fichier
>0	Nombre d'octets effectivement lus

Exemple de lecture d'un fichier

```
#include<fcntl.h> #include<stdio.h> #include<errno.h> #include<unistd.h>
#define MAXLU 65536
int main(int argc, char *argv[]) {
  ssize_t nblus;
  char buff[MAXLU + 1];
  int fd = open(argv[1], O_RDONLY);
  if(fd == -1) { perror("Open "); return errno; }
  while(nblus = read(fd, buff, MAXLU)) {
     if(nblus == (ssize_t) -1) { perror("Read "); return errno; }
     buff[nblus] = '\0';
     printf("%s",buff);
  }
  close(fd);
  return 0;
}
```

The mask

Le masque correspond aux droits non donnés par défaut lors de la création d'un fichier

Exemple : 0022 -> pas d'écriture et pas d'exécution pour le groupe et les autres utilisateurs

```
#include <sys/stat.h>
mode_t umask(mode_t mask)
```

Arguments

Nouvelle valeur du masque avec S_I{R|W|X}{USR|GRP|OTH}

Valeur de retour

Valeur du masque avant la modification

Exemple de création d'un fichier avec des droits supérieurs au masque

```
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <sys/stat.h>

int main(int argc, char *argv[]) {
  mask(0);
  int fd = open(argv[1], O_RDONLY|O_CREAT|O_EXCL,
    S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH);
  if(fd == -1) {
     perror("Open ");
     return errno;
  }
  return 0;
}
```

Exercice 12.1

La création d'un fichier avec les droits souhaités

Créer un fichier avec le nom et les droits choisis par l'utilisateur contenant

```
# !/bin/bash
echo "Hello World !"
```

Exercice 12.2

Le compte est bon

Compter le nombre de caractères présents dans un fichier dont le nom est passé sur la ligne de commande

Programmation système

Les processus

Le numéro du processus

```
#include <unistd.h>
pid_t getpid(); // numéro du processus
pid_t getppid(); // numéro de processus de son parent
```

Valeur de retour

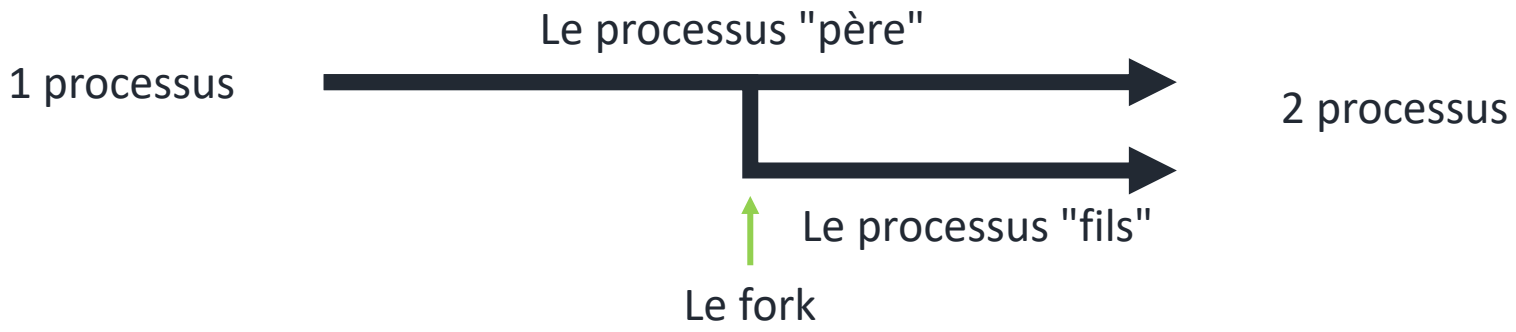
numéro du processus

Le numéro du processus

```
#include<stdio.h>
#include<unistd.h>

int main() {
    printf("numéro de ce processus : %d\n", getpid());
    printf("numéro de processus de son parent : %d\n", getppid());
    return 0;
}
```

La division en 2 processus

```
#include <unistd.h>
pid_t fork();
```



Valeur de retour

-1	Erreur, code erreur positionné dans la variable errno
0	Processus enfant
>0	Processus parent (numéro du processus de son enfant)

La division en 2 processus

```
#include <stdio.h>
#include<unistd.h>

int main()
{
   pid_t pidc = fork();
   switch(pidc) {
      case -1 : printf("erreur le fork n'a pu avoir lieu..."); break;
      case 0 : printf("enfant : %d %d %d\n", pidc, getpid(), getppid()); break;
      default : printf("père : %d %d %d\n", pidc, getpid(), getppid());
                sleep(1); break;
   }
   return 0;
}
```

L'attente de la fin d'un processus

```
#include <sys/wait.h>
pid_t wait(int * status);
pid_t waitpid(pid_t pid, int * status, int options);
```

Arguments

status	Pointeur vers une valeur entière contenant le code retour du processus
pid	Numéro du processus ou -1 pour tout enfant
options	Options d'attente

Valeur de retour

-1	Erreur, code erreur positionné dans la variable errno
>0	Identifiant du processus enfant terminé

L'attente de la fin d'un processus

```
#include<unistd.h>    #include<sys/wait.h>    #include<stdio.h>
int main() {
   pid_t pidc = fork();
   switch(pidc) {
      case -1 : printf("erreur le fork n'a pu avoir lieu..."); break;
      case 0 : sleep(1); printf("fils : j'ai fini\n"); break;
      default :
         printf("père : j'attends...\n");
         int codeRetourFils;
         waitpid(pidc, &codeRetourFils, 0);
         printf("mon fils a fini avec le code retour %d\n", codeRetourFils);
         break;
   }
   return 0;
}
```

Exécution d'un autre processus

```
#include <unistd.h>

int execve(const char *chemin,
           char *const argv[],
           char *const envp[]);
```

Arguments

chemin	Chemin d'accès au fichier exécutable à lancer
argv	Arguments à passer (avec NULL dans la dernière case)
envp	Variables d'environnement couples nom=valeur (avec NULL dans la dernière case)

Valeur de retour

-1	Erreur, code erreur positionné dans la variable errno
	Aucun retour en cas de succès

Exécution d'un autre processus

```
#include<unistd.h>
int main() {
    execve("/bin/ls", NULL, NULL);
    return 0;
}
```

Exécution d'un autre processus

```
#include <unistd.h>
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ...,
                char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],
                char *const envp[]);
```

v	Tableau pour les paramètres
l	Liste de valeurs à la place d'un tableau
p	Recherche de l'exécutable dans le PATH
e	Ajout de variables d'environnement

Exécution d'un autre processus

```
#include<unistd.h>
int main() {
    execvpe("ls", NULL, NULL);
    return 0;
}
```

Exercice 13.1

Je suis ton père…

Écrire un programme se scindant en deux processus qui affichent chacun un message :

Je suis ton père

Je suis ton fils

Exercice 13.2

La course

- Deux processus affichent chacun 1000 fois un caractère. Le premier à avoir terminé gagne.
- Afficher le résultat, par exemple : "le processus 4587 a gagné (enfant 1)"

Exercice 13.3

Le contenu des répertoires

Écrire un programme qui fait saisir à l'utilisateur le chemin d'un répertoire et affiche le contenu détaillé de celui-ci en faisant appel à la commande `ls` puis recommence à moins que l'utilisateur saisisse 0 pour mettre fin à l'exécution.

Programmation système

Les signaux

Quelques signaux

2	SIGINT	CTRL-C	
9	SIGKILL	Terminaison forcée	Non blocable
10	SIGUSR1	Utilisateur 1	
12	SIGUSR2	Utilisateur 2	
14	SIGALRM	Alarme	
18	SIGCONT	Reprise du processus	
20	SIGTSTP	CTRL-Z	

Envoi d'un signal

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

Arguments

pid	Numéro du processus vers lequel envoyer le signal
sig	Numéro du signal à envoyer

Valeur de retour

-1	Erreur, code erreur positionné dans la variable errno
0	En cas de succès

Envoi d'un signal

```
#include<stdio.h>
#include<signal.h>

int main() {
    printf("Numéro du processus ?\n");
    int numPid;
    scanf(" %d", &numPid);
    kill(SIGINT, numPid);
    return 0;
}
```

Gestion des signaux

```
#include <signal.h>

int sigaction(int numSignal,
              const struct sigaction *act, struct sigaction *oldAct);
```

Arguments

numSignal	Numéro du signal à intercepter (sauf SIGKILL et SIGSTOP)
act	Nouvelle réaction au signal
oldAct	Ancienne réaction au signal ou NULL

Structure sigaction

sa_handler	Adresse d'une fonction callback ou SIG_IGN ou SIG_DFL
sa_mask	Masque des signaux bloqués pendant l'appel à la fonction callback
sa_flags	Options de gestion du signal

Valeur de retour

-1	Erreur, code erreur positionné dans la variable errno
0	En cas de succès

Exemple

```
#include<stdio.h>  #include<unistd.h>  #include<signal.h>
void traite(int numSig) {
   printf("signal reçu :  %d\n", numSig);
   sleep(10);
}
int main() {
   struct sigaction act;
   act.sa_handler = &traite;
   sigemptyset(&act.sa_mask);
   act.sa_flags = 0;
   sigaction(SIGUSR1, &act, NULL);
   while(1) {
      sleep(1);  printf("tic\n"); sleep(1);  printf("tac\n");
   }
   return 0;
}
```

Exercice 14.1

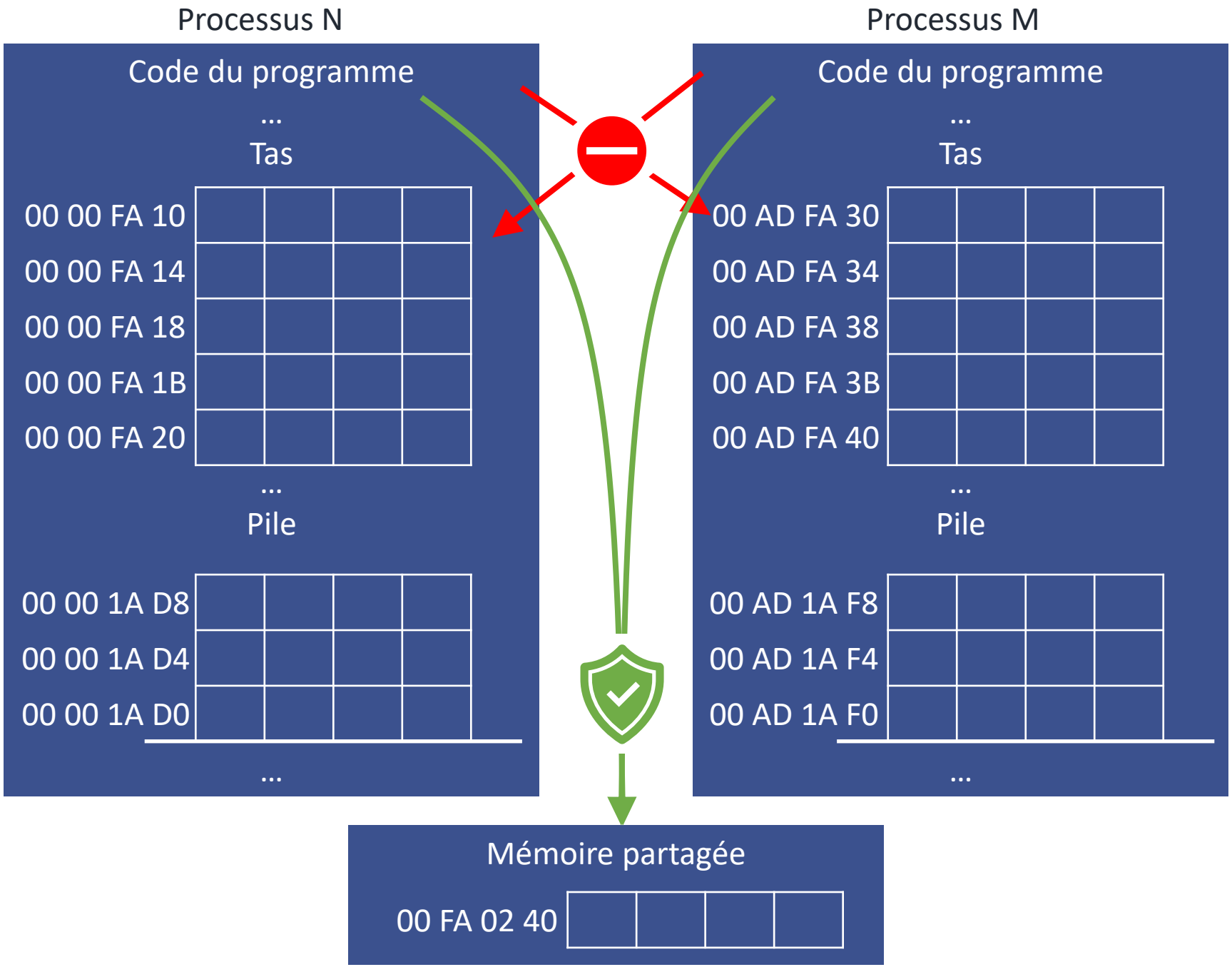
Questions express

- Écrire une fonction qui pose une question à l'utilisateur qui doit répondre dans un temps impartit.
- Écrire un programme posant 3 questions avec limite de temps

Programmation système

La communication avec la mémoire partagée

La mémoire partagée



Création d'une zone de mémoire partagée

```
#include <sys/mman.h>
#include <sys/stat.h> /* For mode constants */
#include <fcntl.h> /* For O_* constants */

int shm_open(const char* nom, int drapeaux, mode_t mode);
```

Arguments

nom	commence par / et aucun autre / autorisé
drapeaux	O_RDONLY ou O_RDWR (+ O_CREAT, O_EXCL ou O_TRUNC)
mode	droits en octal

Valeur de retour

>0	succès : numéro du descripteur
-1	erreur

Allocation de mémoire partagée

```
#include <unistd.h>

int ftruncate(int descripteur, off_t taille);
```

Arguments

descripteur	valeur retournée par shm_open()
taille	nombre d'octets à allouer

Valeur de retour

0	succès
-1	erreur

Mappage

```
#include <sys/mman.h>

void* mmap(NULL, size_t taille, int protection, MAP_SHARED,
           int descripteur, off_t decalage);
```

Arguments

taille	nombre d'octets à allouer
protection	PROT_READ ou PROT_WRITE
descripteur	valeur retournée par shm_open()
decalage	décalage par rapport au début de la zone partagée

Valeur de retour

>0	succès : pointeur vers la zone mémoire
MAP_FAILED	erreur

Exemple d'écriture

```c
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
   const char* CHEMIN = "/memoirePartagee";
   const size_t TAILLE = sizeof(int)*3;
   int mem = shm_open(CHEMIN, O_CREAT|O_RDWR, 0666);
   if(mem==-1) {perror("shm_open"); return 1; }
   if(ftruncate(mem, TAILLE) == -1) { perror("ftruncate"); return 2; }
   int* tabParatage = mmap(NULL, TAILLE, PROT_WRITE, MAP_SHARED, mem, 0);
   if(tabParatage == MAP_FAILED) { perror("mmap"); return 3; }
   tabParatage[0] = 66; tabParatage[1] = 111; tabParatage[2] = 110;
   munmap(tabParatage, TAILLE);
   close(mem);
   return 0;
}
```

Exemple de lecture

```c
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
   const char* CHEMIN = "/memoirePartagee";
   const size_t TAILLE = sizeof(int)*3;
   int mem = shm_open(CHEMIN, O_RDONLY, 0666);
   if(mem==-1) {perror("shm_open : "); return 1; }
   int* tabParatage = mmap(NULL, TAILLE, PROT_READ, MAP_SHARED, mem, 0);
   if(tabParatage == MAP_FAILED) { perror("mmap : "); return 3; }
   printf("%d - %d - %d\n", tabParatage[0], tabParatage[1], tabParatage[2]);
   munmap(tabParatage, TAILLE);
   close(mem);
   return 0;
}
```

Programmation système

Les sémaphores

Les sémaphores

Norme POSIX

« Bâton de parole »

pour éviter que plusieurs processus accèdent à la même ressource

Création ou ouverture d'un sémaphore

```
#include <fcntl.h> // constantes O_*
#include <sys/stat.h> // constantes mode
#include <semaphore.h>

sem_t* sem_open(const char *nom, int drapeaux);
```

Arguments

nom	commence par / et aucun autre / autorisé
drapeaux	O_CREAT, O_EXCL, O_RDWR, O_RDONLY

Valeur de retour

>0	succès : pointeur vers le sémaphore
SEM_FAILED	erreur

Exemple de création d'un sémaphore

```c
#include<stdio.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<semaphore.h>

int main() {
   const char* NOM_SEMAPHORE = "/semaphore";
   sem_t* sem = sem_open(NOM_SEMAPHORE, O_RDWR|O_CREAT|O_EXCL, 0666, 1);
   if(sem == SEM_FAILED) { perror("semaphore"); return 1; }
   sem_close(sem);
   return 0;
}
```

Zone exclusive

```
#include <semaphore.h>

int sem_wait(sem_t* semaphore);
// zone exclusive
int sem_post(sem_t* semaphore);
```

Arguments

semaphore	pointeur vers le sémaphore

Valeur de retour

0	succès
-1	erreur

Exemple d'utilisation d'un sémaphore

```
#include<stdio.h>  #include<unistd.h>  #include<fcntl.h>  #include<sys/stat.h>
#include<semaphore.h>  #include<sys/wait.h>

int main() {
   const char* NOM_SEMAPHORE = "/semaphore";
   sem_t* sem = sem_open(NOM_SEMAPHORE, O_RDONLY);
   if(sem == SEM_FAILED) { perror("semaphore"); return 1; }
   char* laPhrase =
            "Les semaphores permettent de synchroniser l'acces a une ressource";
   pid_t pid = fork();
   sem_wait(sem);
   for(int i=0; i<65; i++) {
      printf("%c\n", laPhrase[i]);
   }
   sem_post(sem);
   sem_close(sem);
   if(pid!=0)
      wait(NULL);
   return 0;
}
```

La suppression d’un sémaphore

Programmation système

Les threads

Les threads

Norme POSIX

Compilation avec **`-pthread`**

```
int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void *(*start_routine)(void *),
                   void *restrict arg);
```

Attributs :

- tread : Pointeur avec l'identifiant du thread
- attr : Mettre NULL
- start_routine : Fonction à exécuter par le thread
- arg : Arguments à passer à la fonction à exécuter

Retour

- 0 : fin normale
- >0 : code erreur

Exemple

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>

void* bonjour(void* donnees) {
   printf("Bonjour du thread n°%d avec l'ID %ld !\n", *(int*)donnees, pthread_self());
   return NULL;
}

int main() {
   const int NB = 10;
   pthread_t ids[10];
   int numero[10];
   for(int i=0; i<NB; i++) {
      numero[i] = i+1;
      printf("création du thread n°%d\n", numero[i]);
      if(pthread_create(&(ids[i]), NULL, bonjour, &(numero[i]))) {
         printf("création de thread impossible\n");
         return 1;
      }
   }
   sleep(10);
   return 0;
}
```

L’attente de la fin d’un thread

```
int pthread_join(pthread_t thread, void **retval);
```

Paramètres

- thread : thread attendu
- retval : Valeur de retour de la fonction effectuée par le thread

Retour

- 0 : fin normale
- >0 : code erreur

Les mutex

MutEx : mutuellement exclusif

Verrous sur des données partagées

Déclaration d'une variable de type pthread_mutex_t

Exemple :

pthread_mutex_t verrous;

L’initialisation d’un mutex

```
int pthread_mutex_init(pthread_mutex_t * mutex,
                       const pthread_mutexattr_t * attr);
```

Attributs :

- mutex : Pointeur vers le mutex
- attr : Mettre NULL

Retour

- 0 : fin normale
- >0 : code erreur

Le verrouillage avec un mutex

int pthread_mutex_lock(pthread_mutex_t *_mutex_);

int pthread_mutex_unlock(pthread_mutex_t *_mutex_);

Attributs :

- mutex : Pointeur vers le mutex

Retour

- 0 : fin normale
- >0 : code erreur

Exemple

```
#include<stdio.h>
#include<pthread.h>

int cpt = 0;
pthread_mutex_t mutex_cpt;

void* compte(void* donnees) {
   for(int i=0; i<1000000; i++) {
      pthread_mutex_lock(&mutex_cpt);
      cpt++;
      pthread_mutex_unlock(&mutex_cpt);
   }
   return NULL;
}
```

```
int main() {
   pthread_mutex_init(&mutex_cpt, NULL);
   pthread_t ids[2];
   for(int i=0; i<2; i++) {
      if(pthread_create(&(ids[i]), NULL,
                        compte, NULL)) {
         printf("problème de thread\n");
         return 1;
      }
   }
   for(int i=0; i<2; i++) {
      pthread_join(ids[i], NULL);
   }
   printf("valeur de cpt à la fin : %d\n", cpt);
   return 0;
}
```