

The background of the slide is a dark blue image of a circuit board. It features intricate gold-colored traces and pads. Overlaid on the right side of the board is a pattern of binary code (0s and 1s) in a light blue/green color, arranged in a way that suggests data flow or digital logic.

Programmation 1: Fondamentaux

Introduction aux fonctions

Introduction aux Fonctions en langage C

A quoi servent les fonctions ?

Comme les procédures, les **fonctions** sont des sous-programmes qui permettent de rendre le code **réutilisable**

Alors qu'une procédure réalise un traitement sur les données en entrée, une fonction **renvoie aussi un résultat récupérable par une simple assignation de variable**, ou, par une évaluation au sein d'une expression:

```
int surface1 = surfaceCarre( longueurCote );  
if (surfaceCarre( longueurCote ) > 100 ) { .. }
```

Paramètres des fonctions

Une fonction utilise une liste de paramètres afin d'obtenir une partie des paramètres en entrée.

```
int Somme( int A, int B );
```

Remarque: D'autres données en entrée peuvent provenir de **variables** dites **globales**.

Paramètres des fonctions

Tous les paramètres de la fonction ont un type qu'il faut préciser lors de la déclaration de fonction

```
int Somme( int A, int B );
```

Déclaration des fonctions

- Les fonctions du langage C doivent **être déclarées** avant leur utilisation
- Une déclaration de fonction est aussi appelée un **prototype**
- Les prototypes peuvent être créés au début du fichier **.c** ou dans un des **fichiers d'entêtes** (header) qui possède une extension **.h**
- Dans un programme C, il existe une seule et unique fonction nommée **main()** qui ne doit pas être redéclarée. Elle sert de point d'entrée au programme.
- Les prototypes de fonction peuvent contenir des noms de paramètres fictifs afin de faciliter la compréhension de leur utilité.

```
float Somme( float val1 , float val2 );
```

Déclaration des fonctions

- Les fichiers d'entêtes .h ne doivent contenir que des déclarations et pas de code correspondant à des instructions

fichier util.h

```
float Somme( float val1 , float val2 );
```

```
float Produit(float val1, float val2);
```

```
float Quotient(float val1, float val2);
```

Déclarations des fonctions

- Les fichiers source .c doivent inclure les fichiers d'entêtes .h dont ils ont besoin

fichier prog.c

```
#include "util.h"
```


Implémentation des fonctions

En plus de sa déclaration, une fonction doit fournir le code qui va véritablement constituer son corps. Ce code se trouve dans un fichier .c

```
float Somme( float val1 , float val2 ) {  
    float tmp;  
    tmp = val1 + val2;  
    return tmp;  
}
```

Implémentation des fonctions

- Puisque la fonction renvoie une valeur, il faut lui donner un type. Le type est précisé avant le nom de la fonction.
- Dans l'exemple la fonction est de type **float**.
- Une fonction C qui ne renvoie pas de valeur est du type **void** et s'apparente donc à une procédure.

```
float Somme( float val1 , float val2 ) {  
    float tmp;  
    tmp = val1 + val2;  
    return tmp;  
}
```

Implémentation des fonctions

- Les instructions de la fonction sont regroupées dans un **bloc { }**
- la fonction renvoie la valeur grâce au mot clé **return**
- Les variables déclarées dans la liste des paramètres sont des variables locales à la fonction
- Au besoin, on peut déclarer d'autres variables locales.
- Les variables locales sont détruites quand on quitte la fonction

```
float Somme( float val1 , float val2 ) {  
    float tmp;  
    tmp = val1 + val2;  
    return tmp;  
}
```

Implémentation des fonctions

- Dans la déclaration ou l'implémentation de la fonction, les paramètres sont dits **formels**. Ils ne servent qu'à généraliser le code de la fonction

```
float Somme( float val1 , float val2 ) {  
    float tmp;  
    tmp = val1 + val2;  
    return tmp;  
}
```

Utilisation ou Appel des fonctions

Une fois les fonctions déclarées et implémentées, on peut les utiliser dans le code du programme ou dans le code d'autres fonctions. Il s'agit de **l'appel de fonction**.

```
mySum = Somme( 10.0, 20.0); // les arguments ou paramètres sont des constantes  
float A=10.0; float B=20.0;  
mySum = Somme( A, B); // les arguments sont des variables
```

Utilisation ou Appel des fonctions

Dans cet exemple, l'appel de fonction permet de faire une copie des paramètres réels (A et B) vers les paramètres formels (val1 et val2). Il s'agit d'un **passage de paramètres dit par valeur**

Les paramètres formels et réels doivent être de même type

```
mySum = Somme( 10.0, 20.0); // les arguments ou paramètres sont des constantes  
float A=10.0; float B=20.0;  
mySum = Somme( A, B); // les arguments sont des variables
```

Utilisation ou Appel des fonctions

Les paramètres **formels** et **réels** doivent être de même type. S'ils ne le sont pas, il faut faire une conversion de type explicite à l'aide d'un **cast**, ou, tenter une conversion implicite sans faire de **cast**.

```
float  Somme(float val1, float val2);  // déclaration avec paramètres en flottants
```

```
int A=10; int B=20;  
float mySum;
```

```
mySum =  Somme( (float) A, (float) B );  // conversion explicite avec un cast (float)
```

Utilisation ou Appel des fonctions

Les modifications apportées aux paramètres passés par valeur ne sont pas conservées.

Pour modifier les variables en entrée (paramètres), il est nécessaire de passer une référence (**l'adresse**) vers ces variables plutôt que la **valeur** des variables elle-même.

En langage C, cela se fait à l'aide des opérateurs ***** et **&**.

```
void Swap(float * val1, float * val2); // déclaration des paramètres de type pointeur afin d'obtenir une
adresse
void Swap(float * val1, float * val2) {
    float tmp;
    tmp = *val1;      *val1 = *val2;      *val2=tmp;
}
float A=10;    float B=20;

Swap( &A, &B ); // appel de la fonction en passant l'adresse des variables dont on veut permuter les valeurs
```


Le cas le plus simple : appel de fonction avec passage des arguments par **valeur** :

- Déclaration d'une variable

```
int x ;
```

- Utilisation locale de la variable :

```
x = 12 ;
```

- La fonction :

- Implémentation

```
void Afficher (int valeur) {  
    printf("%d", valeur );  
}
```

Paramètre formel

- Utilisation

```
Afficher(x) ;
```

Paramètre réel

Le cas le plus simple : appel de fonction avec passage des arguments par **valeur** :

Exemple :

```
int X, Y, result ;

int somme( int A , int B) {
    return ( A + B );
}

main() {
    X=10; Y=20;
    result = somme( X, Y);
}
```

Le cas le plus simple : appel de fonction avec passage des arguments par **valeur** :

En Pseudo Code :

Déclaration fonction Somme(Val A en entier, Val B en entier) en entier

fonction somme(Val A en entier, Val B en entier) en entier
début

 Déclaration tmp en entiers

 tmp ← A + B

 retourner(tmp)

fin fonction somme

Programme principal MonProgramme

debut

 Déclaration X, Y, result en entiers

 X ← -10; Y ← -20;

 result ← somme(X, Y);

fin programme

Appel de fonction avec passage des arguments par **référence (adresse)**:

En Pseudo Code :

Déclaration fonction Swap(Ref A en entier, Ref B en entier) en ...

fonction Swap(Ref A en entier, Ref B en entier) en entier
début

 Déclaration tmp en entiers

 tmp ← A

 deref A ← deref B

 deref B ← tmp

fin fonction swap

Programme principal MonProgramme

debut

 Déclaration X, Y, result en entiers

 X ← -10; Y ← -20;

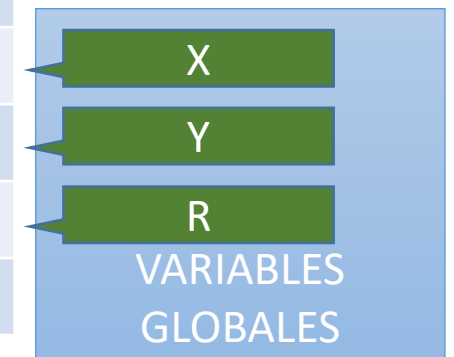
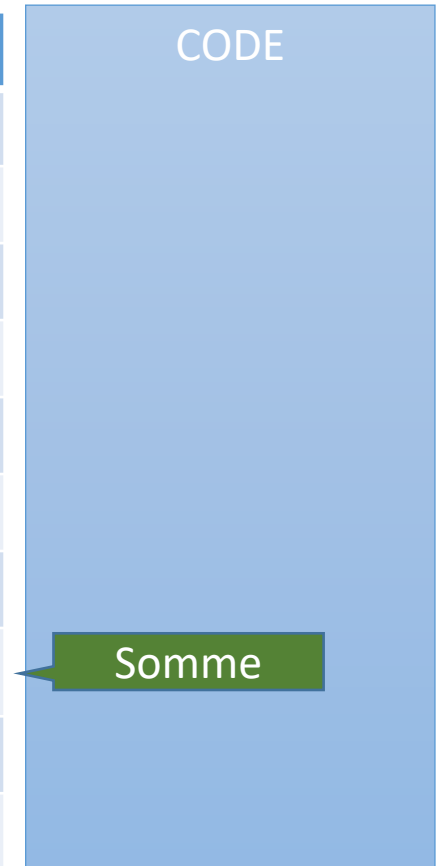
 result ← Swap(X, Y);

fin programme

EIP	0X0008
ESP	0xFFFC
EBP	0xFFFC
EAX	
registre	contenu
CPU	

0xFFF0	
0xFFF4	
0xFFF8	
0xFFFC	
adresse	contenu
PILE	

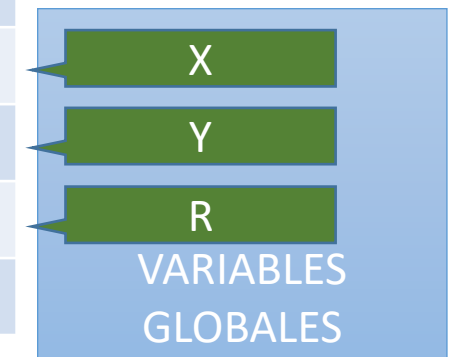
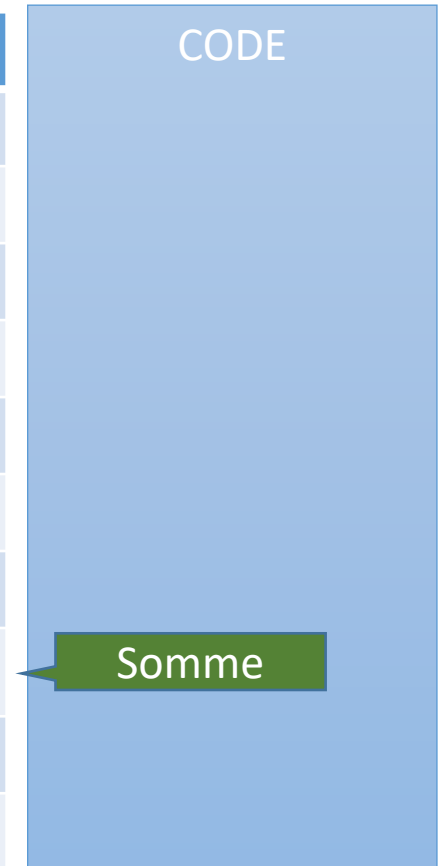
adresse	contenu
0x0000	
0x0008	Push X
0x0020	Push Y
0x0028	Call Somme
0x0030	ADD ESP,8
0x0038	MOV [0x1008] , EAX
..	
0x0500	MOV EAX, [0xFFFF8]
0x0508	ADD EAX, [0xFFFF4]
0x0520	RET
..	
0x1000	10
0x1004	20
0x1008	
..	



EIP	0X0020
ESP	0xFFFF8
EBP	0xFFFFC
EAX	
registre	contenu
CPU	

0xFFFF0	
0xFFFF4	
ESP → 0xFFFF8	10
0xFFFFC	
adresse	contenu
PILE	

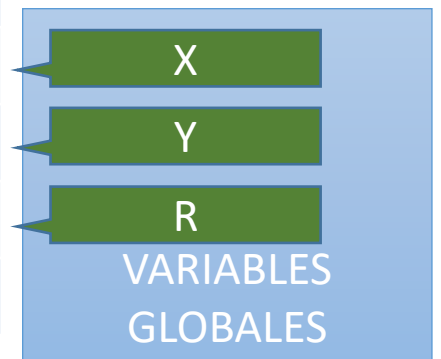
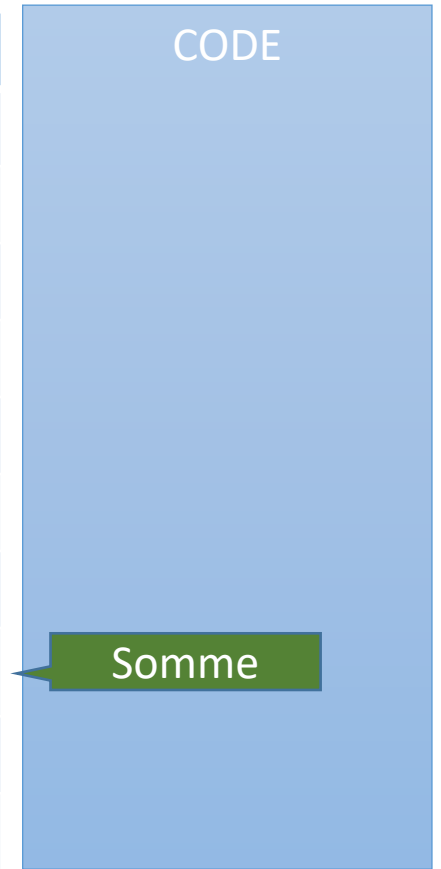
adresse	contenu
0x0000	
0x0008	Push X
0x0020	Push Y
0x0028	Call Somme
0x0030	ADD ESP,8
0x0038	MOV [0x1008] , EAX
..	
0x0500	MOV EAX, [0xFFFF8]
0x0508	ADD EAX, [0xFFFF4]
0x0520	RET
..	
0x1000	10
0x1004	20
0x1008	
..	



EIP	0X0028
ESP	0xFFFF4
EBP	0xFFFFC
EAX	
registre	contenu
CPU	

0xFFFF0	
ESP → 0xFFFF4	20
0xFFFF8	10
0xFFFFC	
adresse	contenu
PILE	

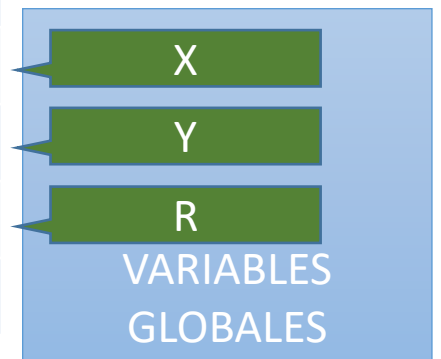
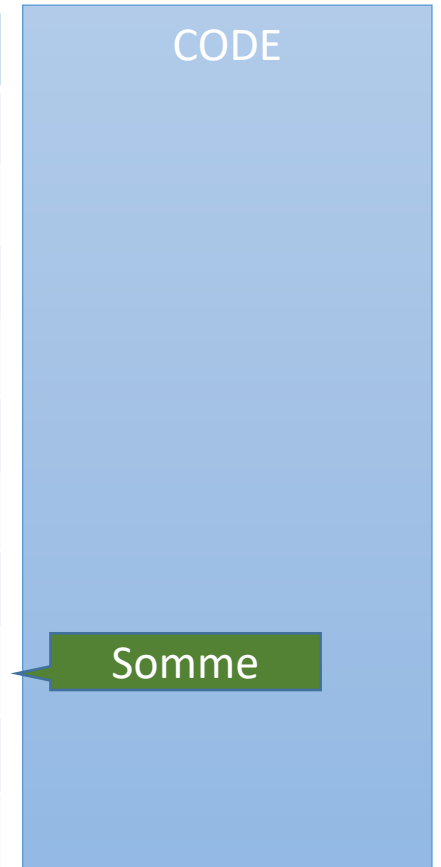
adresse	contenu
0x0000	
0x0008	Push X
0x0020	Push Y
0x0028	Call Somme
0x0030	ADD ESP,8
0x0038	MOV [0x1008] , EAX
..	
0x0500	MOV EAX, [0xFFFF8]
0x0508	ADD EAX, [0xFFFF4]
0x0520	RET
..	
0x1000	10
0x1004	20
0x1008	
..	



EIP	0X0500
ESP	0xFFFF4
EBP	0xFFFFC
EAX	10
registre	contenu
CPU	

0xFFFF0	0x0030
ESP → 0xFFFF4	20
0xFFFF8	10
0xFFFFC	
adresse	contenu
PILE	

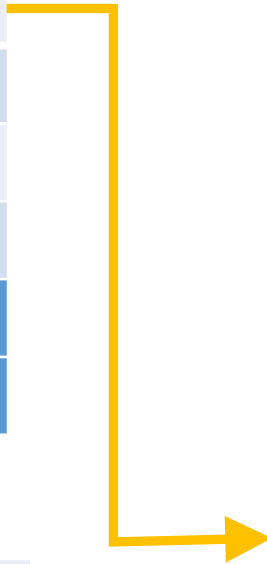
adresse	contenu
0x0000	
0x0008	Push X
0x0020	Push Y
0x0028	Call Somme
0x0030	ADD ESP,8
0x0038	MOV [0x1008] , EAX
..	
0x0500	MOV EAX, [0xFFFF8]
0x0508	ADD EAX, [0xFFFF4]
0x0520	RET
..	
0x1000	10
0x1004	20
0x1008	
..	



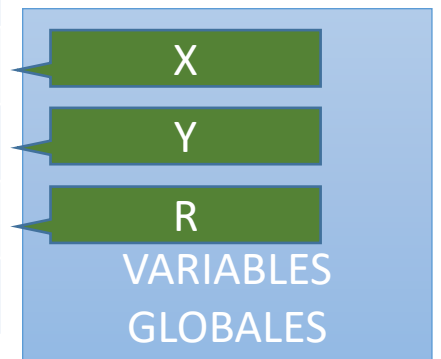
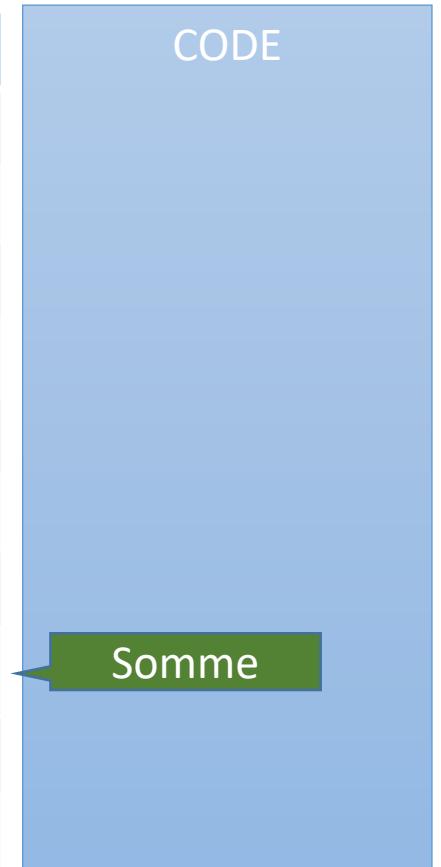
EIP	0X0508
ESP	0xFFFF0
EBP	0xFFFFC
EAX	10
registre	contenu
CPU	

ESP


0xFFFF0	0x0030
0xFFFF4	20
0xFFFF8	10
0xFFFFC	
adresse	contenu
PILE	



adresse	contenu
0x0000	
0x0008	Push X
0x0020	Push Y
0x0028	Call Somme
0x0030	ADD ESP,8
0x0038	MOV [0x1008] , EAX
..	
0x0500	MOV EAX, [0xFFFF8]
0x0508	ADD EAX, [0xFFFF4]
0x0520	RET
..	
0x1000	10
0x1004	20
0x1008	
..	



EIP	0X0520
ESP	0xFFFF0
EBP	0xFFFFC
EAX	30
registre	contenu
CPU	

	0xFFFF0	0x0030
	0xFFFF4	20
	0xFFFF8	10
	0xFFFFC	
	adresse	contenu
PILE		

adresse	contenu
0x0000	
0x0008	Push X
0x0020	Push Y
0x0028	Call Somme
0x0030	ADD ESP,8
0x0038	MOV [0x1008] , EAX
..	
0x0500	MOV EAX, [0xFFFF8]
0x0508	ADD EAX, [0xFFFF4]
0x0520	RET
..	
0x1000	10
0x1004	20
0x1008	
..	



EIP	0X0030
ESP	0xFFFF4
EBP	0xFFFFC
EAX	30
registre	contenu
CPU	

0xFFFF0	0x0030
0xFFFF4	20
0xFFFF8	10
0xFFFFC	
adresse	contenu
PILE	

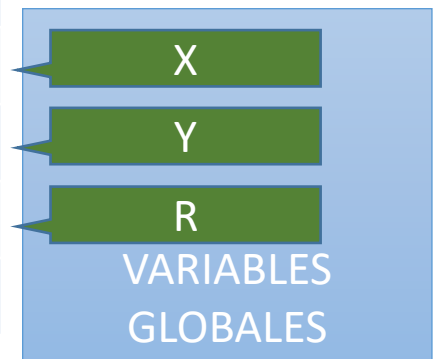
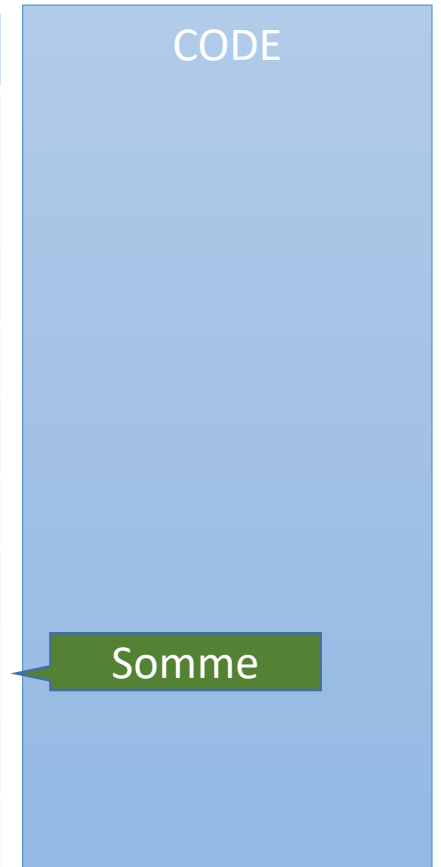
adresse	contenu
0x0000	
0x0008	Push X
0x0020	Push Y
0x0028	Call Somme
0x0030	ADD ESP,8
0x0038	MOV [0x1008] , EAX
..	
0x0500	MOV EAX, [0xFFFF8]
0x0508	ADD EAX, [0xFFFF4]
0x0520	RET
..	
0x1000	10
0x1004	20
0x1008	
..	



EIP	0X0038
ESP	0xFFFC
EBP	0xFFFC
EAX	30
registre	contenu
CPU	

0xFFF0	0x0030
0xFFF4	20
0xFFF8	10
0xFFFC	
adresse	contenu
PILE	

adresse	contenu
0x0000	
0x0008	Push X
0x0020	Push Y
0x0028	Call Somme
0x0030	ADD ESP,8
0x0038	MOV [0x1008] , EAX
..	
0x0500	MOV EAX, [0xFFFF8]
0x0508	ADD EAX, [0xFFFF4]
0x0520	RET
..	
0x1000	10
0x1004	20
0x1008	
..	



EIP	0X0040
ESP	0xFFFC
EBP	0xFFFC
EAX	30
registre	contenu
CPU	

0xFFF0	0x0030
0xFFF4	20
0xFFF8	10
0xFFFC	
adresse	contenu
PILE	

adresse	contenu
0x0000	
0x0008	Push X
0x0020	Push Y
0x0028	Call Somme
0x0030	ADD ESP,8
0x0038	MOV [0x1008] , EAX
..	
0x0500	MOV EAX, [0xFFFF8]
0x0508	ADD EAX, [0xFFFF4]
0x0520	RET
..	
0x1000	10
0x1004	20
0x1008	30
..	



	variable simple	pointeur	pointeur sur structure
déclaration	<code>int x;</code>	<code>// pointeur sur un entier</code> <code>int *debut;</code>	<code>struct personne personne1 = {"Burton",51};</code> <code>struct personne personne2 = {"Audiard",57};</code> <code>// pointeur sur une structure personne</code> <code>struct personne *realisateur;</code>
utilisation locale de la valeur	<code>x = 12;</code>	<code>debut = &x; // debut pointe sur la variable x</code> <code>*debut = 88; // modification de la valeur pointée</code> <code>// x vaut 88</code> <code>// debut contient l'adresse de la variable x</code> <code>printf("\n %d ", x);</code> <code>printf("\n %d ", *debut);</code>	<code>realisateur = &personne1;</code> <code>printf("\n Nom : %s, âge : %d", realisateur->nom, realisateur->age);</code> <code>realisateur = &personne2;</code> <code>printf("\n Nom : %s, âge : %d", realisateur->nom, realisateur->age);</code>
utilisation dans une fonction : passage par valeur	<code>void Afficher(int valeur)</code> <code>{</code> <code>printf("%d", valeur);</code> <code>}</code> <code>Afficher(x);</code>	<code>void Afficher(int valeur)</code> <code>{</code> <code>printf("%d", valeur);</code> <code>}</code> <code>Afficher(*debut);</code>	<code>void AffichePersonne(struct personne pers) {</code> <code>printf("\n%s %d\n", pers.nom, pers.age);</code> <code>}</code> <code>AffichePersonne(*realisateur);</code>
utilisation dans une fonction : passage par adresse	<code>void Doubler(int *valeur)</code> <code>{</code> <code>*valeur = *valeur * 2;</code> <code>}</code> <code>Doubler(&x);</code>	<code>void Doubler(int *valeur)</code> <code>{</code> <code>*valeur = *valeur * 2;</code> <code>}</code> <code>Doubler(debut);</code>	<code>void majuscule(struct personne *pers) {</code> <code>int i; int lg = strlen(pers->nom);</code> <code>for (i=0; i<lg; i++)</code> <code>*(pers->nom+i)=toupper(*(pers->nom+i));</code> <code>}</code> <code>majuscule(realisateur);</code>

Exercices : écrire et tester les fonctions

- Somme des n premiers entiers positifs
- Recherche de la valeur max, parmi les n premières valeurs d'un tableau
- Compter le nombre de caractères dans une chaîne de caractères
- Compter le nombre d'occurrences d'un caractère dans une chaîne de caractères