

INTRODUCTION À LA PROGRAMMATION EN LANGAGE C

Safa CHEBBI

Safa.chebbi@supcom.tn

05/02/2021

PLAN DU COURS

- I. Historique**
- II. Les directives à destination du préprocesseur**
- III. Les types de base**
- IV. Variables et constantes**
- V. Les opérateurs**
- VI. Affichages et saisies**

HISTORIQUE ET CARACTÉRISTIQUES DU C

- ❑ La définition de la première version du langage C en 1988, au sein de l'institut national américain de normalisation (ANSI: American National Standards Institute)
- ❑ Fondateurs du langage C: Brian W.kernighan et Denis M. Ritchie
- ❑ Créé pour écrire UNIX
- ❑ Proche des langages haut-niveaux
- ❑ Proche du langage machine
- ❑ très apprécié pour la programmation des systèmes embarqués

STRUCTURE D'UN PROGRAMME EN C

The diagram illustrates the structure of a C program with various annotations and boxes explaining the components.

TYPE de la valeur de retour (Red box): A green arrow points from this box to the `int` in the `int main(void)` signature.

"main" : Cela signifie "principale", ses instructions sont exécutées (Blue box): A green arrow points from this box to the `main` function name in the signature.

void main(void): La fonction main ne prend aucun paramètre et ne retourne pas de valeur. (Blue box): A yellow arrow points from this box to the `void` parameter in the signature.

int main(void): La fonction main retourne une valeur entière à l'aide de l'instruction return (0 si pas d'erreur). (Blue box): A yellow arrow points from this box to the `int` return type in the signature.

int main(int argc, char *argv[]): On obtient alors des programmes auxquels on peut adresser des arguments au moment où on lance le programme. (Blue box): A yellow arrow points from this box to the `int argc, char *argv[]` parameters in the signature.

Entre accolades "{" et "}" on mettra la succession d'actions à réaliser.(Bloc) (Blue box): A red arrow points from this box to the curly braces of the `main` function body.

begin (Red text): A red arrow points from this text to the opening curly brace of the `main` function body.

end (Red text): A red arrow points from this text to the closing curly brace of the `main` function body.

Code snippet:

```
int main(void)
{
    /* corps du programme*/
    declaration des Cstes et Var ;
    instruction1 ;
    instruction2 ;
    ...
}
```

LES DIRECTIVES À DESTINATION DU PRÉPROCESSEUR

❑ Le préprocesseur effectue un prétraitement du programme source avant qu'il soit compilé au cours duquel il exécute des instructions particulières appelées directives.

❑ Directives:

- ✓ les premières lignes placées au début du programme identifiées par le caractère # en tête
- ✓ prises en compte avant la compilation du programme
- ✓ doivent être écrites à raison d'une par ligne
- ✓ contient des déclarations appropriées concernant cette fonction
- ✓ Un même fichier en-tête contient des déclarations relatives à plusieurs fonctions
- ✓ En général, il est indispensable d'incorporer `stdio.h` (bibliothèque standard permettant de gérer toute action du clavier (entrée ou sortie))

```
#include <nom-de-fichier> /* répertoire standard */  
#include "nom-de-fichier" /* répertoire courant */
```

La gestion des fichiers (**stdio.h**)
Les fonctions mathématiques (**math.h**)
Taille des type entiers (**limits.h**)
Limites des type réels (**float.h**)
Traitement de chaînes de caractères (**string.h**)
Le traitement de caractères (**ctype.h**)
Utilitaires généraux (**stdlib.h**)
Date et heure (**time.h**)

LA NOTION DE TYPE

- ❑ La mémoire centrale est un ensemble de positions binaires nommées bits.
- ❑ Les bits sont regroupés en octets (8 bits), et chaque octet est repéré par ce qu'on nomme son adresse
- ❑ L'ordinateur représente et traite les informations exprimées (codées) sous forme binaire.
- ❑ Une séquence binaire peut représenter un nombre entier, nombre réel, caractère, instruction, etc)
- ❑ Il n'est pas possible d'attribuer une signification à une information binaire tant que l'on ne connaît pas la manière dont elle a été codée.
- ❑ la notion de type permet de régler ce problème

LES TYPES DE BASE DANS C

Dans le langage C, il y a 4 types de base, les autres types seront dérivés de ceux-ci:

Type	Signification	Exemples de valeur	Codage en mémoire	Peut être
char	Caractère unique	'a' 'A' 'z' 'Z' '\n' 'a' 'A' 'z' 'Z' '\n' Varie de -128 à 127	1 octet	signed, unsigned
int	Nombre entier	0 1 -1 4589 32000 -231 à 231 +1	2 ou 4 octets	Short, long, signed, unsigned
float	Nombre réel simple	0.0 1.0 3.14 5.32 -1.23	4 octets	
double	Nombre réel double précision	0.0 1.0E-10 1.0 - 1.34567896	8 octets	long

QUELQUES RÈGLES D'ÉCRITURES

❑ Chaque instruction en langage C se termine par un point virgule « ; ».

❑ **Les identificateurs:**

- ✓ servent à désigner les différents « objets » manipulés par le programme (variables, fonction, ...)
- ✓ formés d'une suite de caractères choisis parmi les lettres ou les chiffres. Le premier d'entre eux étant nécessairement une lettre.
- ✓ Le caractère « _ » est considéré comme une lettre. Il peut apparaître au début d'un identificateur (_total; _89)
- ✓ Les majuscules et les minuscules sont autorisées mais ne sont pas équivalentes (ligne ≠ Ligne)
- ✓ Un identificateur ne peut pas être un mot réservé du langage (exemples: int, auto, if, break, switch, return, ...)

❑ **Les séparateurs:**

- ✓ int x,y; et non intx, y
- ✓ int n,compte,total,p; ou bien plus lisiblement int n, compte, total, p;

❑ **Les commentaires:**

- ✓ Le langage C autorise la présence de commentaires dans les programmes
- ✓ Il s'agit de textes explicatifs destinés aux lecteurs du programme et qui n'ont aucune incidence sur sa compilation
- ✓ Ils sont placés entre /* et */ ou bien //
- ✓ Ils peuvent apparaître à tout endroit du programme

LES VARIABLES EN C

❑ Déclarations des variables:

❖ **Syntaxe:** Type identificateur1, identificateur2, ..., ;

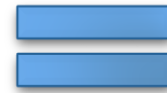
❖ **Exemple:** char c1,c2;

int i, j, var_entier;

❑ **Initialisations des variables:** Les variables doivent être déclarées avant leur utilisation dans un début de bloc (juste après{)

```
void main()  
{  
    char c;  
    int a,b;  
    c="A";  
    a=5;  
    b=10;  
}
```

est équivalent à



```
void main()  
{  
    char c="A";  
    int a=5;  
    int b=10;  
}
```

LES DÉCLARATIONS DES CONSTANTES

❑ **1^{ère} Méthode:** définition d'un symbole à l'aide de la directive de compilation `#define`

❖ **Syntaxe:** `#define identificateur valeur`

❖ **Exemple:**

```
#define PI 3.14159
void main()
{
    float perimetre, rayon = 8.7;
    perimetre = 2*rayon*PI;
    ....
}
```

Le compilateur ne réserve pas de place en mémoire

Les identificateurs s'écrivent traditionnellement en majuscules, mais ce n'est pas une obligation.

❑ **2^{ème} Méthode:** déclaration d'une variable, dont la valeur sera constante pour tout le programme

❖ **Syntaxe:** `const type identificateur=valeur;`

❖ **Exemple:**

```
void main()
{
    const float PI = 3.14159;
    const int JOURS = 5;
    float perimetre, rayon = 8.7;
    perimetre = 2*rayon*PI;
    ....
    JOURS = 3;
    ....
}
```

Le compilateur réserve de la place en mémoire (ici 4 octets).

On ne peut pas changer la valeur d'une const.

LES OPÉRATEURS EN C (1)

❑ **Les opérateurs arithmétiques:** Le langage C propose les opérateurs suivants:

+	addition
-	soustraction
*	multiplication
/	division (entière et rationnelle!)
%	modulo (reste d'une division entière)

Priorité des opérateurs: (*, /, %)
puis (+ et -)

le modulo (%) ne peut porter que sur des entiers

Le quotient de deux entiers fournit un entier:
(5/2) vaut 2 (5.0/2.0) vaut 2.5

❑ **Les opérateurs relationnels:** C permet de comparer des expressions à l'aide d'opérateurs classiques de comparaisons

==	égal à
!=	différent de
<, <=, >, >=	plus petit que, ...

Le résultat de la comparaison est:
✓ 0 si le résultat de la comparaison est faux
✓ 1 si le résultat de la comparaison est vrai

Priorité des opérateurs:
(<, <=, >, >=) puis (== et !=)

LES OPÉRATEURS EN C (2)

❑ **Les opérateurs logiques:** Ces opérateurs produisent un résultat numérique (int)

&&	et logique (and)
	ou logique (or)
!	négation logique (not)

Exemple: $(a < b) \ \&\& \ (c < d)$ prend la valeur 1 si les deux expressions sont vraies et 0 sinon

Priorité des opérateurs:
! puis && puis ||

$(!n) \iff (n == 0)$

$a < b \ \&\& \ c < d \iff (a < b) \ \&\& \ (c < d)$

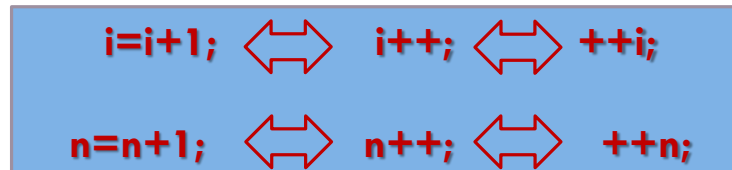
❑ **L'opérateur d'affectation :** permet d'affecter une valeur à une variable

- ✓ $i=5;$ \Rightarrow Affectation de la valeur 5 à i \Rightarrow la valeur de i après affectation: 5
- ✓ La priorité de l'opérateur d'affectation est faible par rapport aux opérateurs arithmétiques et de comparaison
- ✓ $c=b+3 \iff c=(b+3)$
- ✓ $c+5=x;$ n'a pas de sens

LES OPÉRATEURS EN C (3)

❑ Les opérateurs d'incrémentation et de décrémentation:

- ✓ `int i=5; x = i++;` passe d'abord la valeur de `i` à `x` et incrémente après (`x :5 ; i :6`)
- ✓ `int i=5; x = i-- ;` passe d'abord la valeur de `i` à `x` et décrémente après (`x :5, i :4`)
- ✓ `int i=5; x = ++i ;` incrémente d'abord et passe la valeur incrémentée à `x` (`i :6, x :6`)
- ✓ `int i=5; x = --i ;` décrémente d'abord et passe la valeur décrémentée à `x` (`i :4, x :4`)



❑ Les opérateurs d'affectation élargie:

- ✓ `i++;` remplace `i=i+1;`
- ✓ `i=i+k;` remplace `i+=k;`
- ✓ `a=a*b;` remplace `a*=b;`
- ✓ `+=; -=; *=; /=; %=;` ces opérateurs permettent de condenser l'écriture de certaines instructions

LES CONVERSIONS DE TYPE: CASTING

❑ Conversions forcées par une affectation:

Lors d'une affectation, toutes les conversions sont acceptées par le compilateur

=> problème de perte de précision

❑ Conversions forcées:

- ✓ Le programmeur peut forcer la conversion d'une expression quelconque dans un type de son choix
- ✓ **Exemple:** `int n=5, p=2; (double) (n/p);` => Aura comme valeur celle de `n/p` convertie en double

```
#include<stdio.h>
void main()
{
    int n;
    float x=2.3;
    n=x+5.3;
    printf("%d", n);
}
```

⇒ ce programme affiche 7

```
#include<stdio.h>
void main()
{
    char A=3;
    int B=4;
    float C;
    C = A/B;
    printf("%f",C);
}
=> ce programme affiche 0.000000
```

```
#include<stdio.h>
void main()
{
    char A=3;
    int B=4;
    float C;
    C = (float) A/B;
    printf("%f",C);
}
=> ce programme affiche 0.750000
```

Les contenus de A et de B
restent inchangés;
seulement les valeurs
utilisées dans les
calculs sont converties

CLASSES DE PRIORITÉ DES OPÉRATEURS

Priorité 1 (la plus forte):	()
Priorité 2:	! ++ --
Priorité 3:	* / %
Priorité 4:	+ -
Priorité 5:	< <= > >=
Priorité 6:	== !=
Priorité 7:	&&
Priorité 8:	
Priorité 9 (la plus faible):	= += -= *= /= %=

QUELQUES FONCTIONS PRÉDÉFINIES DE LA BIBLIOTHÈQUE MATH.H

COMMANDE C	EXPLICATION	
exp(X)	fonction exponentielle	e^x
log(X)	logarithme naturel	$\ln(X)$, $X > 0$
log10(X)	logarithme à base 10	$\log_{10}(X)$, $X > 0$
pow(X,Y)	X exposant Y	X^Y
sqrt(X)	racine carrée de X	pour $X > 0$
fabs(X)	valeur absolue de X	$ X $
floor(X)	arrondir en moins	int(X)
ceil(X)	arrondir en plus	
fmod(X,Y)	reste rationnel de X/Y (même signe que X)	pour X différent de 0
sin(X) cos(X) tan(X)	sinus, cosinus, tangente de X	
asin(X) acos(X) atan(X)	arcsin(X), arccos(X), arctan(X)	
sinh(X) cosh(X) tanh(X)	sinus, cosinus, tangente hyperboliques de X	

AFFICHAGES ET SAISIES (1)

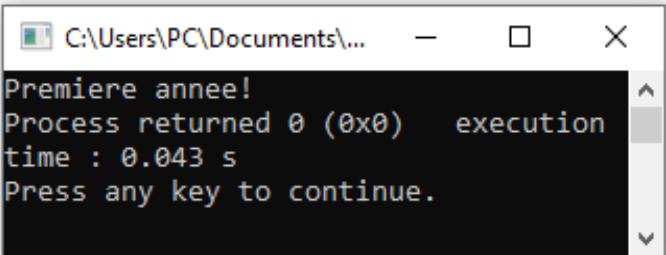
□ Affichage avec C:

- ✓ Appelle une fonction prédéfinie (fournie avec le langage et qu'on n'a pas besoin d'écrire) nommée printf.
- ✓ Cette fonction reçoit un argument « le message qu'on désire l'affichage sur le console »
- ✓ Les guillemets servent à délimiter une « chaîne de caractères » (suite de caractères).

Exemple 1

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

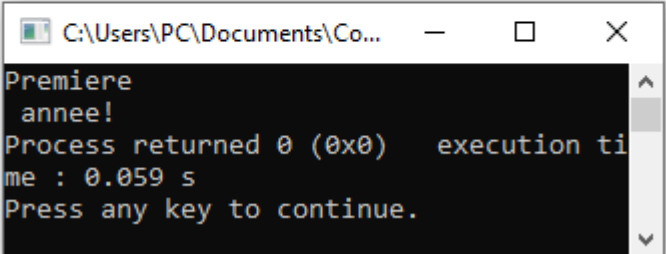
int main()
{
    printf("Premiere annee!");
}
```



Exemple 2

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

int main()
{
    printf("Premiere \n annee!");
}
```



\n: permet un
retour à la
ligne

AFFICHAGES ET SAISIES (2)

Exemple 3

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

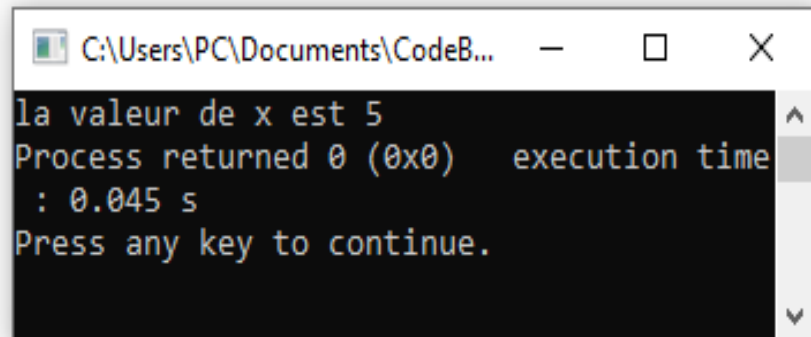
```
int main()
```

```
{
```

```
    int x=5;
```

```
    printf("la valeur de x est %d",x);
```

```
}
```



```
la valeur de x est 5
Process returned 0 (0x0)   execution time
: 0.045 s
Press any key to continue.
```

`printf("la valeur de x est %d", x) ;`

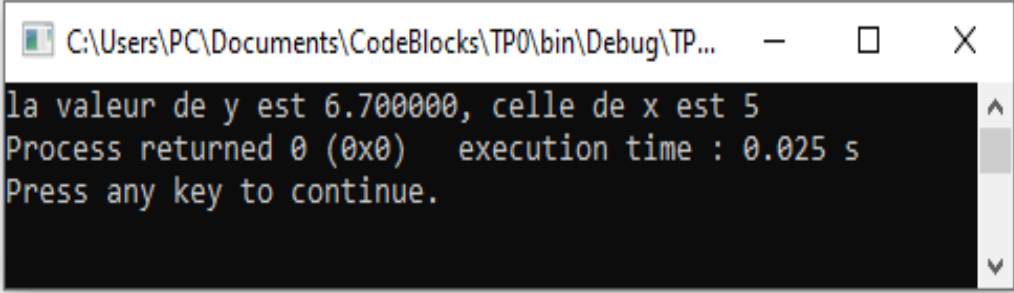
- ✓ printf reçoit deux arguments : "la valeur de x est %d" et x
- ✓ Le % indique que le caractère suivant est, non plus du texte à afficher tel quel, mais un « code de format »
- ✓ %d est remplacé par la valeur de x.
- ✓ Il faut toujours veiller à accorder le code de format au type de la valeur correspondante (%d pour int; %f pour float; %p pour @mémoire, %c pour char, &...)
- ✓ Printf(« %p »,&x);

AFFICHAGES ET SAISIES (3)

Exemple 4

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

int main()
{
    int x=5;
    float y=6.7;
    printf("la valeur de y est %f, celle de x est %d",y,x);
}
```



C:\Users\PC\Documents\CodeBlocks\TP0\bin\Debug\TP... — □ ×

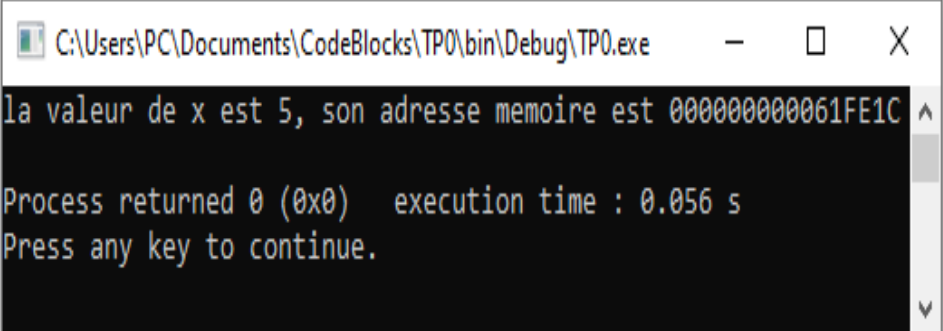
la valeur de y est 6.700000, celle de x est 5
Process returned 0 (0x0) execution time : 0.025 s
Press any key to continue.

Type entier : int ; %d
Type réel : float ; %f

Exemple 5

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

int main()
{
    int x=5;
    printf("la valeur de x est %d, son adresse memoire est %p",x,&x);
}
```



C:\Users\PC\Documents\CodeBlocks\TP0\bin\Debug\TP0.exe — □ ×

la valeur de x est 5, son adresse memoire est 000000000061FE1C
Process returned 0 (0x0) execution time : 0.056 s
Press any key to continue.

%p permet d'afficher
une adresse mémoire

AFFICHAGES ET SAISIES (4)

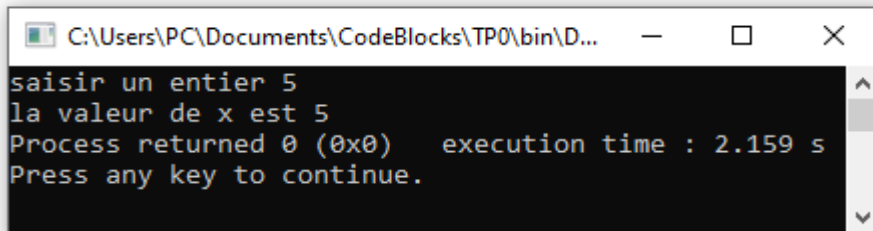
❑ Saisie avec C: `scanf (« %d », &x);`

- ✓ une fonction prédéfinie en C dont le rôle est de lire des informations au clavier.
- ✓ Comme `printf`, la fonction `scanf` possède en premier argument un format exprimé sous forme d'une chaîne de caractère: « %d »(int)
- ✓ L'argument précise dans quelle variable on souhaite placer la valeur lue
- ✓ `&x`: la fonction `scanf` range la valeur lue à l'adresse de la variable. (& est un opérateur signifiant « adresse de »)

Exemple 1

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

int main()
{
    int x;
    printf("saisir un entier");
    scanf("%d", &x);
    printf("la valeur de x est %d", x);
}
```

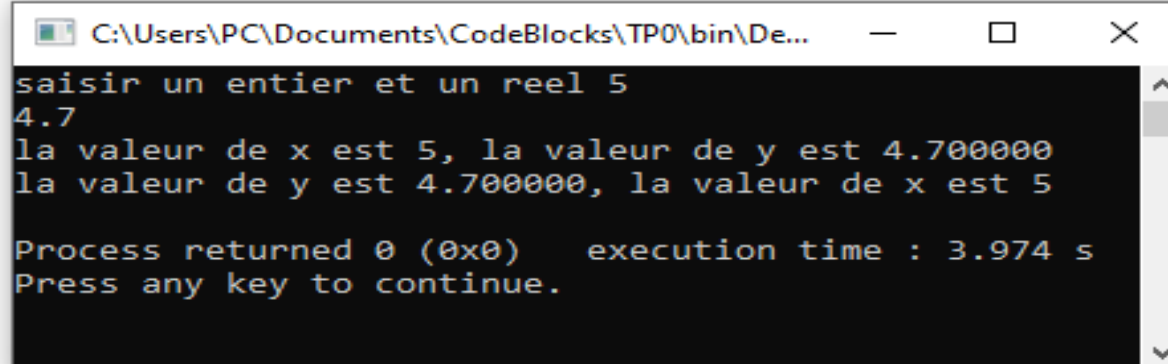


AFFICHAGES ET SAISIES (5)

Exemple 2

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

int main()
{
    int x;
    float y;
    printf("saisir un entier et un reel");
    scanf("%d%f",&x,&y);
    printf("la valeur de x est %d, la valeur de y est %f \n",x,y);
    printf("la valeur de y est %f, la valeur de x est %d \n",y,x);
}
```



```
C:\Users\PC\Documents\CodeBlocks\TP0\bin\De...
saisir un entier et un reel 5
4.7
la valeur de x est 5, la valeur de y est 4.700000
la valeur de y est 4.700000, la valeur de x est 5

Process returned 0 (0x0)   execution time : 3.974 s
Press any key to continue.
```

EXERCICES D'APPLICATION

Exercice 1

Ecrire un programme qui saisit 2 entiers et affiche successivement la somme, la différence, le produit et le quotient de ces 2 entiers.

EXERCICES D'APPLICATION

Exercice 2

Ecrire un programme qui calcule le périmètre et la surface d'un rectangle.

EXERCICES D'APPLICATION

Exercice 3

Ecrire un programme qui saisit deux entiers a et b , permute la valeur de ces deux entiers et affiche les deux entiers avant et après permutation.

EXERCICES D'APPLICATION

Exercice 4

Ecrire un programme qui calcule la somme de 5 nombres entiers introduits en clavier.

EXERCICES D'APPLICATION

Exercice 5

Ecrire un programme C qui lit en entrée trois entiers et affiche leur moyenne avec une précision de deux chiffres après la virgule.

m

%.2f