# Introduction au langage C

DAKKAR Borhen-eddine

Lycée le Corbusier

BTS SN-IR

# Table des matières

# Table des matières

# Historique

- Le langage C a été initialement développé dans les années 1970 par Ken Thompson, Dennis Ritchie et Brian Kernighan chez les Laboratories Bell.
- Il est utilisé pour créer des programmes simples et interactifs ou des programmes sophistiqués et complexes, dans le contexte d'un langage véritablement structuré.
- Ce langage a continué d'évoluer depuis sa création jusqu'à aujourd'hui. Pour cette raison, C est devenu le langage professionnel pour les programmeurs.
- Le langage C a été normalisé premièrement par l'ANSI (American National Standard Institute). C'est pour cela on parle toujours de « C ANSI » ou de « C norme ANSI ».

# Exemple introductif

## Pseudocode

**Algorithme 1 :** Somme entiers de 1 à 100

**Début**

**Entier** : n, a, b

**Réel** : somme

n ← 100

a ← 1

b ← 100

somme ← $n * (a + b)/2$

Écrire ("La somme est égale à :", somme);

**FIN**

## Programme C

```c
#include <stdio.h>
#include <stdlib.h>
//--- Somme entiers de 1 à 100 ---//
main()
{
    //--- Déclaration ---//
    int n, a, b ;
    float somme;
    //--- Initialisation ---//
    n = 100;
    a = 1;
    b = 100;
    somme = n*(a+b)/2;
    printf ("La somme est égale à : %f", somme);
}
```

# La fonction **main**

- Le programme commence par :
  #include <stdio.h>
  #include <stdlib.h>
  Pour le moment admettez simplement que ces deux ligne seront appelées au début de chaque programme.

- L'une des fonctionnalités intéressantes de C c'est la modularité. Autrment dit, nous pouvons écrire différentes fonctions où chacune effectue une tache différente.

- Pour créer le lien entre ces fonctions on utilise une fonction principale qui s'appelle main.

# La fonction **main**

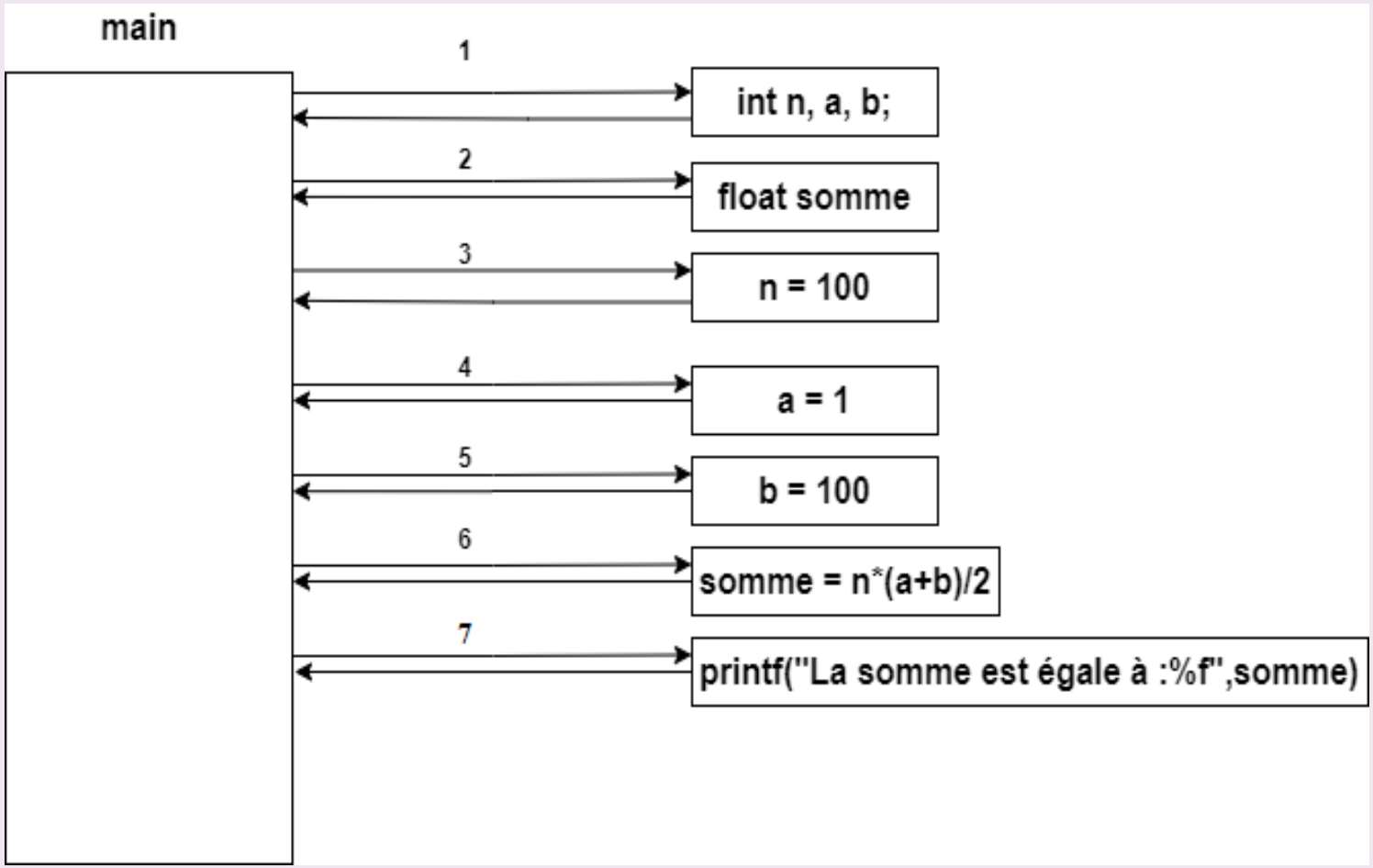
- Tout programme C doit avoir une fonction appelée main (). Elle indique aux fonctions et aux instructions le séquence dans laquelle ils doivent opérer.

- Les accolades, { et }, déterminent le début et fin du corps de la fonction. Les instructions entre accolades déterminent ce que fait la fonction.

- Chaque instruction à l'intérieur de la fonction doit se terminer par un point-virgule ";".

# La fonction **main**

## Déroulement du programme



# La fonction **printf**

- **printf** est une fonction prédéfinie fournie avec le langage et donc que vous n'avez pas à écrire vous-même.
- Les guillemets " " servent à délimiter une chaîne de caractères (suite de caractères).
- La notation \n est conventionnelle : elle représente un caractère de fin de ligne.
- Ecrire **printf ("Bonjour\n") ;** affiche sur l'écran **Bonjour** et saute la ligne.
- Les caractères %f signifie que le caractère suivant n'est pas un texte à afficher mais une valeur (ici la valeur de somme - flottante-).

# La fonction **scanf**

- **scanf** est une deuxième fonction prédéfinie fournie par le langage C.
- Le rôle de cette fonction est de lire une information au clavier.
- L'appel de la fonction scanf se fait comme suit :

```
scanf ("%f", &x) ;
```

.

- Le premier argument est exprimé sous forme d'une chaîne de caractères "%f". Il correspond au type de la variable que nous voulons lire, ici une variable flottante.
- Le deuxième paramètre est le nom de la variable précédé par le symbole esperluette "&". Ce dernier permet d'obtenir l'adresse de la variable. .
- **x** représente la varible que nous voulons lire.

# Exemple

```c
#include <stdio.h>
#include <stdlib.h>
main( )
{
    printf("Je suis un élève de BTS SN\n");
    printf("Au lycée le Corbusier\n");
    printf("44 Rue Léopold Rechossière\n");
    printf("93300 Aubervilliers");
}
```

La sortie de ce programme est:
Je suis un élève de BTS SN
Au lycée le Corbusier
44 Rue Léopold Rechossière
93300 Aubervilliers

# Règles d'écriture

- Un identificateur d'une variable est formé d'une suite de caractères, des lettres ou des chiffres. Le premier d'entre eux étant nécessairement une lettre.
- Le caractère souligné ( _ underscore en anglais) est considéré comme une lettre.
- Les majuscules et les minuscules ne sont pas équivalentes en C. Ainsi, somme et Somme représentent deux mots différents.
- Les symboles /* text */ premettent de mettre "text" en commentaires dans votre programme source. Vous pouvez utiliser aussi les deux slash // pour commenter un ligne de code.

# Table des matières

# Données de base

C reconnaît trois types de données de base:

- Nombres entiers (int).
- nombres flottants (float ou double).
- caractères (char).

**Les nombres entiers :**

- Une constante entière en C est un nombre positif ou négatif sans virgule.
- Il existe trois tailles différentes d'entiers : short int, int et long int.
- La taille ne dépond pas uniquement de mot-clé considéré, mais également de la machine utilisée.
- À titre d'exemple, 16 bits nous permettent de représenter des entiers s'étendant de -32 768 à 32 767.

# Données de base

## Les nombres flottants

- Les constantes à virgule flottante et à double précision sont des nombres signés ou non signés ayant une virgule décimale.
- La différence entre les nombres à virgule flottante et à double précision est la quantité de bits de stockage qu'un ordinateur utilise pour chaque type.
- Un nombre réel sera représenté en flottant en déterminant deux quantités **M** (mantisse) et **E** (exposant) telles que la valeur : $M \times B^E$ ($1.5\ \ 10^{22}$).
- Le C prévoit trois types de flottants correspondant à des tailles différentes : float, double et long double.

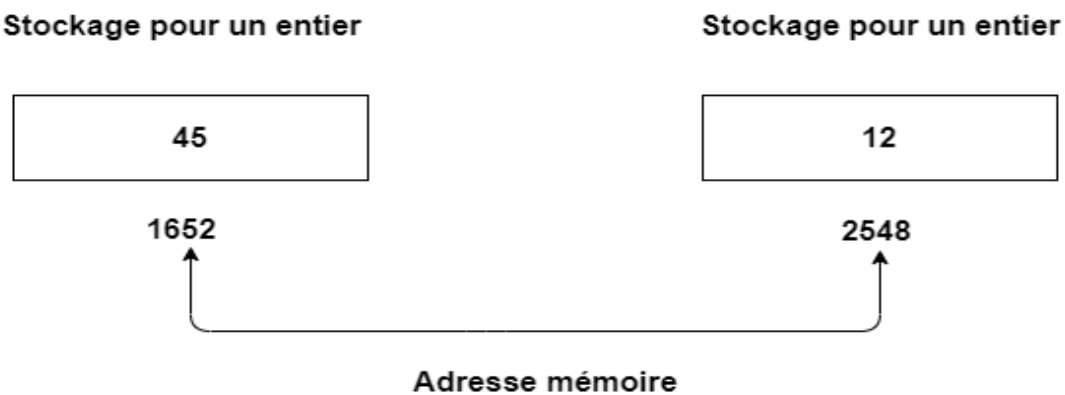
# Données de base

## Les caractères

- Les caractères sont les lettres de l'alphabet, les dix chiffres de 0 à 9 et les symboles spéciaux comme +, $ , -!, ...
- Un caractère en C s'écrit entre deux apostrophes (ou quotes) comme dans cet exemple : 'a', 'b','9'.
- Il existe des caractères non imprimables comme:
  - \r CR Retour chariot (Carriage Return)..
  - \n LF Saut de ligne (Line Feed).
  - \t HT Tabulation horizontale (Horizontal Tab).
  - \v VT Tabulation verticale (Vertical Tab).

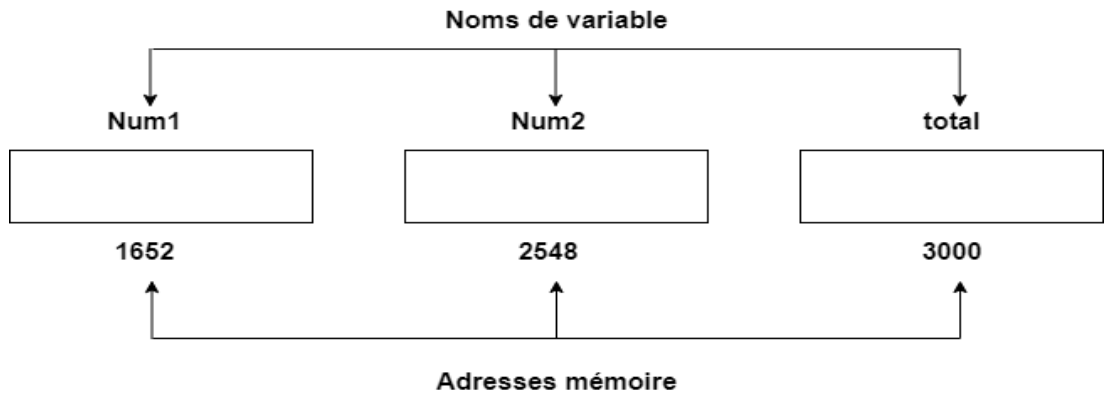
# Variables et déclarations

- Tous les nombres entiers, flottants et valeurs utilisés dans un programme sont stockés et récupérés à partir de la mémoire de l'ordinateur.
- Avant que les langages de haut niveau tels que C n'existent, les emplacements de mémoire étaient référencés par leurs adresses.
- Par exemple, pour stocker les valeurs 45 et 12 dans les emplacements mémoire 1652 et 2548 respectivement, les instructions requises sont équivalentes à:
  **mettre un 45 à l'emplacement 1652**
  **mettre un 12 à l'emplacement 2548**
- Pour ajouter ces deux nombres et enregistrer leur résultat dans l'emplacement mémoire 3000, il fallait une instruction équivalante à:
  **ajouter le contenu de l'emplacement 1652 au contenu de l'emplacement 2548 et stocker le résultat dans l'emplacement 3000**



# Variables et déclarations

- Dans langages évolué comme C, les noms symboliques sont utilisés à la place de l'adresse mémoire réelle.
- Une variable est simplement un nom donné par le programmeur à un emplacement de stockage mémoire. Le terme variable est utilisée car la valeur stockée dans la variable peut changer ou varier.
- En utilisant ces noms de variables, les opérations de stockage et d'addition précédentes peuvent être remplacées par les instructions C suivantes:
  num1 = 45;
  num2 = 12;
  total = num1 + num2



# Variables et déclarations

- Un déclaration en langage C doit spécifier le type et le nom de la variable.
  type nom_de_variable;
  où **type** désigne un type de données valide et le **nom_de_variable** est un nom choisi par l'utilisateur.
- **Exemple :**

```
int total;
```

  déclare **total** comme le nom d'une variable capable de stocker une valeur entière.
- Pour déclarer une variable de type flottant ou de type caractère, nous utilisons la syntaxe suivante:

```
float somme;//déclare somme comme variable flottante
double division;//déclare division comme variable double
char var_carectere; //déclare var_caractre comme variable caractère.
```

# Variables et déclarations

- L'instruction C la plus basique pour à la fois attribuer des valeurs aux variables et effectuer des calculs est l'instruction d'affectation, sa forme générale est la suivante:

```
variable = operande;
```

**Exemple :**

```
Longueur = 5;
Lagrgeur = 11.5;
```

- Il est possible de définir une instruction d'affectation utilisant des expressions:

```
somme = 3 + 7;
diff = 15 - 6;
pente = (y2 - y1) / (x2 - xl);
```

# Exemple

## Exemple 1

```
main()
{
    int sum;
    sum = 25;
    printf("\n Le nombre stocké dans somme est %d.",sum);
    sum = sum + 10;
    printf (" \n Maintenant le nombre stocké dans somme est %d.", sum) ;
}
```

Le programme affiche:

Le nombre stocké dans somme est 25

Maintenant le nombre stocké dans somme est 35

Les expressions d'affectation telles que

```
sum = sum + 25
```

qui utilisent la même variable des deux côtés de l'opérateur d'affectation, peuvent être écrites :

```
sum += 25
```

de même pour les autres opérations arithmétiques : $+=, *=, /=, \%=$

# Exemple

## Exemple 2

```
#include <stdio.h>
main( )
{
    int count;
    count = 0;
    printf("\nThe La valeur initiale de count est %d.", count);
    ++count;
    printf("\n Maintenant count est %d.", count);
    ++count;
    printf ( "\n Maintenant count est%d.", count);
    ++count;
    printf("\n Maintenant count est %d.", count);
    ++count;
}
```

Le programme affiche :
La valeur initiale de count est O.
Maintenant count est I.
Maintenant count est 2.
Maintenant count est 3.

# Exemple

## Exemple 3

```c
#include <stdio.h>
main()
{
    int num1, num2, num3;
    float Moyenne;
    printf("Enter les trois nombers: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    Moyenne = (num1 + num2 + num3)/3.0;
    printf("La moyenne est égale à %f", Moyenne);
}
```

Le programme affiche :
Enter les trois nombers: 22 56 73
La moyenne est égale à 50.333333

# Table des matières

# La notion de priorité

- Il est nécessaire de savoir l'ordre d'exécution des opérateurs arithmétiques.
- Prenons l'expression suivantes : $a + b * c$. Tout d'abord, il y aura l'addition de $a$ et $b$ suivi par une multiplication par $c$.
- En C les opérateurs $+$ et $-$ ont la priorité la plus élevée. On trouve ensuite, à un même niveau, les opérateurs $*$, $/$ et %.
- Il est conseillé d'utiliser les parenthèses pour gérer les priorités, en forçant le calcul préalable de l'expression qu'elles contiennent.

```
a + b * c          a + ( b * c )
- a / - b + c      ( ( - a ) / ( - b ) ) + c
a * b + c * d      ( a * b ) + ( c * d )
```

# Opérateurs relationnels

- Les opérateurs relationnels en C sont les suivants:

| | |
|---|---|
| $<$ | inférieur à |
| $<=$ | inférieur ou égal à |
| $>$ | supérieur à |
| $>=$ | supérieur ou égal à |
| $==$ | égal à |
| $!=$ | différent de |

- Il faut savoir que les opérateurs $(<, <=, >, >=)$ ont la même priorité. tandis que les opérateurs $(== et != )$ possèdent la même priorité, mais celle-ci est inférieure à celle des précédents.

- L'instruction $a < b == c < d$ est interprétée comme :
$(a < b) == (c < d)$

# Les opérateurs logiques

Il existe trois opérateurs logiquesen C :

- **et (noté &) :** l'expression $(a < b)\&(c < d)$ prend la valeur 1 (vrai) si les deux expressions $a < b$ et $c < d$ sont toutes deux vraies.
- **ou (noté ||) :** l'expression $(a < b)||(c < d)$ prend la valeur 1 (vrai) si l'une au moins des deux conditions $a < b$ et $c < d$ est vraie (de valeur 1), et prend la valeur 0 (faux) dans le cas contraire.
- **non (noté !) :** l'expression $!(a < b)$ prend la valeur 1 (vrai) si la condition $a < b$ est fausse (de valeur 0) et prend la valeur 0 (faux) dans le cas contraire.

# Références