

C : concepts de base

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`



Plan

1 Commentaires

2 Variables

- Déclaration
- Déduction de type
- Taille
- Portée de variable

3 Entrée/sortie

- Affichage avec `printf`
- Lecture avec `scanf`
- Copie de variable

4 Pointeurs

5 Opérations sur les variables

- Opérations arithmétiques
- Opérations sur les chaînes de caractère

6 Librairie `cmath`

7 Constantes

8 Structures conditionnelles

En C++

Deux types de commentaires

Commentaire sur une seule ligne

```
// commentaire
```

© Achref EL MOUELHI ©

Commentaire sur une seule ligne

```
// commentaire
```

Raccourcis VSC

- Pour commenter : `Ctrl` + `k` ensuite `Ctrl` + `c`
- Pour décommenter : `Ctrl` + `k` ensuite `Ctrl` + `u`
- Ou aussi : `Ctrl` + `:` pour commenter et décommenter

Commentaire sur plusieurs lignes

```
/* le commentaire  
   la suite  
   et encore la suite  
*/
```

© Achref EL M.

Commentaire sur plusieurs lignes

```
/* le commentaire  
   la suite  
   et encore la suite  
*/
```

Raccourci VSC

Pour commenter ou décommenter : Alt + Shift + a

Variable ?

- Représentation d'un espace mémoire
- Permet de stocker une ou plusieurs données
- Peut avoir plusieurs valeurs différentes dans un programme

© Achref EL M

C

Variable ?

- Représentation d'un espace mémoire
- Permet de stocker une ou plusieurs données
- Peut avoir plusieurs valeurs différentes dans un programme

C est un langage fortement typé

- Il faut préciser le type de chaque variable
- Une variable peut avoir différentes valeurs mais ne peut changer de type.

Déclarer une variable

```
type nom_variable;
```

© Achret EL BOUJELHI ©

Principaux types de données numériques en C

- `short` : entier sur 2 octets, valeur entre -32 768 et 32 767
- `int` : entier, généralement sur 4 octets
- `long` : entier, généralement sur 8 octets
- `float` : nombre à virgule flottante codé sur 4 octets
- `double` : nombre à virgule flottante codé sur 8 octets

Remarques

- À tous ces types, on peut ajouter le mot-clé `unsigned` pour avoir uniquement des valeurs positives.
- Le langage **C** n'impose pas aux compilateurs un nombre exact d'octets pour chaque type mais impose un nombre minimal.
- Donc, le nombre d'octets réservé à un même type peut varier selon les compilateurs.

Types texte en C

- `char` : caractère **ASCII** codé sur 1 octet
- `char[]` : tableau de caractères pour représenter une chaîne

© Achref EL MOU

C

Types texte en C

- `char` : caractère **ASCII** codé sur 1 octet
- `char[]` : tableau de caractères pour représenter une chaîne

Type booléen en C

Le langage **C** ne possède pas de type `bool` natif (introduit en **C99** via `stdbool.h`). Avant cela, on utilisait `int` avec 0 pour `false` et 1 pour `true`.

C

Nom de variable en C

- Peut contenir une lettre, un chiffre ou un _ (underscore)
- Doit commencer par une lettre ou un _ (underscore)
- Sensible à la casse
- Ne peut contenir d'espace ou de caractères spéciaux comme #, !...
- Ne peut être un mot réservé du langage comme `for`, `int`...

Nom de variable en C

- Peut contenir une lettre, un chiffre ou un _ (underscore)
- Doit commencer par une lettre ou un _ (underscore)
- Sensible à la casse
- Ne peut contenir d'espace ou de caractères spéciaux comme #, !...
- Ne peut être un mot réservé du langage comme `for`, `int`...

Remarque

Donner un nom significatif aux variables.

Exemple

```
int x;
```

© Achref EL MOUELHI ©

Exemple

```
int x;
```

Déclaration + initialisation [copy initialization]

```
int x = 5;
```

Exemple

```
int x;
```

Déclaration + initialisation [copy initialization]

```
int x = 5;
```

L'initialisation suivante est incorrecte

```
int x = NULL;
```

Une variable non initialisé contient une valeur indéterminée

```
int x;  
printf("%d",x);  
// affiche 417 (par exemple)
```

C

On peut utiliser `typeof` pour déduire un type

```
int x;  
typeof(x) y;
```

© Achref EL MOUELHI ©

C

On peut utiliser `typeof` pour déduire un type

```
int x;  
typeof(x) y;
```

`x` et `y` sont toutes les deux de type `int`.

C

On peut utiliser `typeof` pour déduire un type

```
int x;  
typeof(x) y;
```

x et y sont toutes les deux de type int.

On peut donc lui affecter un entier

```
y = 3;
```

C

On peut utiliser `typeof` pour déduire un type

```
int x;  
typeof(x) y;
```

x et y sont toutes les deux de type int.

On peut donc lui affecter un entier

```
y = 3;
```

Mais on ne peut pas lui affecter une valeur d'un autre type

```
y = "bonjour"; // Erreur
```

Pour connaître le nombre d'octets réservés à une variable (qui varie selon le type)

```
int x = 5;  
printf("%lu", sizeof(x));  
// affiche 4
```

© Achref EL MOUL

Pour connaître le nombre d'octets réservés à une variable (qui varie selon le type)

```
int x = 5;
printf("%lu", sizeof(x));
// affiche 4
```

sizeof fonctionne aussi avec les types

```
printf("Taille de char : %lu\n", sizeof(char));
printf("Taille de int : %lu\n", sizeof(int));
printf("Taille de float : %lu\n", sizeof(float));
printf("Taille de double : %lu\n", sizeof(double));
```

Remarque

Le type de la valeur de retour de `sizeof` est `size_t` : unsigned int codé sur 64 bits.

Portée (scope) de variable

- locale : déclarée dans une fonction (`main` par exemple)
- globale : déclarée en dehors de toute fonction

Dans ce programme, `l` est une variable locale et `g` est globale

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

```
int g = 10;
```

```
int main()
```

```
{
```

```
    int l = 2;
```

```
    return 0;
```

```
}
```

C

Dans un programme, on peut déclarer une variable locale portant le nom d'une variable globale

```
#include <stdio.h>

int g = 10;

int main()
{
    int g = 2;
    printf("%d", g);
    // affiche 2
    return 0;
}
```

C

Pour accéder au contenu de la variable globale si on a une variable locale avec le même nom

```
#include <stdio.h>

int g = 10;

int main()
{
    int g = 2;
    printf("%d", ::g); // Erreur en C, pas de `::`
    return 0;
}
```

Remarques

- Une variable locale non-initialisée peut contenir une valeur indéterminée.
- Une variable globale non-initialisée contient
 - 0 si la variable est numérique (`int`, `float`, `double`...)
 - `NULL` si la variable est un pointeur
 - caractère vide pour `char`

Opérations d'entrée/sortie

- Pour afficher dans la console, on utilise `printf`.
- Pour lire une donnée saisie dans la console, on utilise `scanf`.

C

Pour afficher le contenu d'une variable

```
int x = 5;  
printf("%d", x);  
// affiche 5
```

© Achref EL MOUELHI ©

C

Pour afficher le contenu d'une variable

```
int x = 5;  
printf("%d", x);  
// affiche 5
```

Pour afficher plusieurs variables

```
int x = 5, y = 3;  
printf("%d %d", x, y);  
// affiche 5 3
```

C

Pour afficher le contenu d'une variable

```
int x = 5;  
printf("%d", x);  
// affiche 5
```

Pour afficher plusieurs variables

```
int x = 5, y = 3;  
printf("%d %d", x, y);  
// affiche 5 3
```

Ajout d'un retour à la ligne

```
int x = 5, y = 3;  
printf("%d\n%d\n", x, y);  
// affiche 5 puis 3 sur une ligne suivante
```

C

Formatage avec `printf` et `scanf` en C

- `%d` : entier décimal (`int`)
- `%f` : nombre flottant (`float`, `double`)
- `%c` : caractère (`char`)
- `%s` : chaîne de caractères (`char *`)
- `%x`, `%X` : entier hexadécimal
- `%o` : entier octal
- `%u` : entier non signé (`unsigned int`)
- `%lu` : entier long non signé (`unsigned long int`)
- `%p` : adresse mémoire (pointeur)
- ...

Exemple :

```
int age = 25;
printf("Mon âge est : %d\n", age);
// affiche Mon âge est : 25

float pi = 3.1415;
printf("Valeur de pi : %.2f\n", pi);
// affiche Valeur de pi : 3.14
```

C

Pour lire un entier

```
int x;  
printf("Saisir un entier : ");  
scanf("%d", &x);  
printf("La valeur saisie est %d\n", x);
```

© Achref EL MOUELHI ©

C

Pour lire un entier

```
int x;  
printf("Saisir un entier : ");  
scanf("%d", &x);  
printf("La valeur saisie est %d\n", x);
```

Pour lire un caractère

```
char c;  
printf("Saisir un caractere : ");  
scanf(" %c", &c);  
printf("La valeur saisie est %c\n", c);
```

C

Pour lire un entier

```
int x;  
printf("Saisir un entier : ");  
scanf("%d", &x);  
printf("La valeur saisie est %d\n", x);
```

Pour lire un caractère

```
char c;  
printf("Saisir un caractere : ");  
scanf(" %c", &c);  
printf("La valeur saisie est %c\n", c);
```

Pour lire plusieurs caractères

```
char c1, c2;  
printf("Saisir deux caracteres : ");  
scanf("%c %c", &c1, &c2);  
printf("%c %c\n", c1, c2);
```

Comment copier le contenu d'une variable dans une autre ?

```
int x = 5;  
int y = x;  
printf("x = %d, y = %d\n", x, y);  
// affiche x = 5, y = 5
```

© Achref EL MOU

Comment copier le contenu d'une variable dans une autre ?

```
int x = 5;  
int y = x;  
printf("x = %d, y = %d\n", x, y);  
// affiche x = 5, y = 5
```

Modifier l'un ne modifie pas l'autre

```
y = 2;  
printf("x = %d, y = %d\n", x, y);  
// affiche x = 5, y = 2
```

Pointeur ?

- Variable qui stocke l'adresse d'une autre variable.
- Il est possible d'avoir plusieurs pointeurs vers une même variable.
- Un pointeur $\equiv$ une variable contenant une adresse mémoire.

C

Pour déclarer un pointeur, on utilise l'opérateur *

```
int *z;
```

© Achref EL MOUELHI ©

C

Pour déclarer un pointeur, on utilise l'opérateur *

```
int *z;
```

Ou aussi

```
int * z;
```

C

Pour déclarer un pointeur, on utilise l'opérateur *

```
int *z;
```

Ou aussi

```
int * z;
```

Ou encore

```
int* z;
```

C

Pour déclarer un pointeur, on utilise l'opérateur *

```
int *z;
```

Ou aussi

```
int * z;
```

Ou encore

```
int* z;
```

Les trois écritures sont équivalentes.

L'opérateur `&` permet de récupérer l'adresse mémoire d'une variable

```
int x = 5;
int *ptr = &x;
printf("%p\n", (void*)&x);
// affiche l'adresse de x
```

Les deux variables `x` et `ptr` pointent vers la même adresse mémoire

```
int x = 5;
int *ptr = &x;

printf("%d\n", x);
// affiche 5

printf("%p\n", (void*)&x);
// affiche l'adresse de x

printf("%d\n", *ptr);
// affiche 5

printf("%p\n", (void*)ptr);
// affiche l'adresse de x
```

Modifier la valeur pointée par le pointeur modifie la variable d'origine

```
*ptr = 2;

printf("%d\n", x);
// affiche 2

printf("%p\n", (void*)&x);
// affiche la même adresse

printf("%d\n", *ptr);
// affiche 2

printf("%p\n", (void*)ptr);
// affiche la même adresse
```

On peut changer la valeur d'un pointeur mais pas sa constante mémoire directement (extcolorredLe code suivant génère une erreur)

```
int t = 10;  
ptr = &t; // OK  
*ptr = t; // OK  
&ptr = t; // ERREUR
```

Exercice 1 : sans tester, qu'affiche le programme suivant ?

```
int a = 2, b = 5;  
int *c = &a;  
*c = b;  
printf("%d\n%d\n%d\n", *c, b, a);
```

© Achref EL M...

Exercice 1 : sans tester, qu'affiche le programme suivant ?

```
int a = 2, b = 5;  
int *c = &a;  
*c = b;  
printf("%d\n%d\n%d\n", *c, b, a);
```

Réponse

5
5
5

Exercice 2 : sans tester, le programme suivant affiche t-il trois adresses identiques ?

```
int a = 2, b = 5;
int *c = &a;
*c = b;
printf("%p\n%p\n%p\n", (void*)c, (void*)&b, (void*)&a);
```

© Achrel

Exercice 2 : sans tester, le programme suivant affiche t-il trois adresses identiques ?

```
int a = 2, b = 5;
int *c = &a;
*c = b;
printf("%p\n%p\n%p\n", (void*)c, (void*)&b, (void*)&a);
```

Réponse

```
0x61ff08
0x61ff04
0x61ff08
```

Pour les variables numériques (`int`, `float`...)

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

C

Exercice

Écrire un code **C** qui

- demande à l'utilisateur de saisir deux entiers a et b
- affiche le résultat de l'équation $a * x + b = 0$

Quelques raccourcis

- $i = i + 1 \Rightarrow i++;$
- $i = i - 1 \Rightarrow i--;$
- $i = i + 2 \Rightarrow i += 2;$
- $i = i - 2 \Rightarrow i -= 2;$

C

Exemple de post-incrémentation

```
int i = 2;  
int j = i++;  
printf("%d\n", i); // affiche 3  
printf("%d\n", j); // affiche 2
```

© Achref EL MOU

Exemple de post-incrémentation

```
int i = 2;  
int j = i++;  
printf("%d\n", i); // affiche 3  
printf("%d\n", j); // affiche 2
```

Exemple de pre-incrémentation

```
int i = 2;  
int j = ++i;  
printf("%d\n", i); // affiche 3  
printf("%d\n", j); // affiche 3
```

C

Utilisation des chaînes de caractères en C

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

int main() {
    char str1[] = "bon";
    char str2[] = "jour";
    char concatResult[20];

    strcpy(concatResult, str1);
    strcat(concatResult, str2);

    printf("%s\n", concatResult);
    // affiche bonjour
    return 0;
}
```

Opérations sur les chaînes de caractères

- `strlen(str)` : retourne la longueur de la chaîne `str`.
- `strcpy(dest, src)` : copie `src` dans `dest`.
- `strcat(dest, src)` : concatène `src` à la fin de `dest`.
- `strcmp(str1, str2)` : compare deux chaînes (retourne 0 si elles sont égales).
- `strchr(str, c)` : retourne un pointeur sur la première occurrence de `c` dans `str`.
- `strstr(str, substr)` : retourne un pointeur sur la première occurrence de `substr` dans `str`.
- `strncpy(dest, src, n)` : copie au maximum `n` caractères de `src` vers `dest`.

C

Exemple : Calcul de la longueur d'une chaîne

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

int main() {
    char str[] = "bonjour";
    printf("%lu\n", strlen(str));
    // affiche 7
    return 0;
}
```

C

Exemple : Trouver un caractère dans une chaîne

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

int main() {
    char str[] = "bonjour";
    char *ptr = strchr(str, 'o');
    if (ptr != NULL) {
        printf("Position: %ld\n", ptr - str);
    }
    return 0;
}
```

Exemple : Comparaison de chaînes

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

int main() {
    char str1[] = "bonjour";
    char str2[] = "bonsoir";

    int cmp = strcmp(str1, str2);
    printf("%d\n", cmp); // affiche un nombre négatif car 'j' < 's'
    return 0;
}
```

C

Exemple : Remplacement d'une sous-chaîne

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

void replaceSubstring(char *str, int pos, int len,
    char *replacement) {
    char buffer[50];
    strncpy(buffer, str, pos);
    buffer[pos] = '\0';
    strcat(buffer, replacement);
    strcat(buffer, str + pos + len);
    strcpy(str, buffer);
}

int main() {
    char str[50] = "bonjour";
    replaceSubstring(str, 2, 2, "soir");
}
```

Librairie `math`

= { fonctions mathématiques prédéfinies }

© Achref EL MOU

Librairie `math`

= { fonctions mathématiques prédéfinies }

Pour l'utiliser, il faut commencer par l'inclure

```
#include <math.h>
```

Quelques fonctions de `math`

- `abs(x)` : retourne la valeur absolue de `x`
- `pow(x, y)` : retourne la `x` puissance `y`
- `max(x, y)` : retourne le max de `x` et `y`
- `min(x, y)` : retourne le min de `x` et `y`
- `sqrt(x)` : retourne la racine carré de `x`
- `floor(x)`, `ceil(x)` et `round(x)` : retournent l'arrondi de `x`
- ...

Exemple avec `sqrt`

```
int x = 9;  
printf("%f", sqrt(x));  
// affiche 3.0
```

Exemple avec `floor(x)`, `ceil(x)` **et** `round(x)`

```
printf("%f", round(1.9)); // affiche 2
printf("%f", round(1.5)); // affiche 2
printf("%f", round(1.4)); // affiche 1
printf("%f", ceil(1.9)); // affiche 2
printf("%f", ceil(1.5)); // affiche 2
printf("%f", ceil(1.4)); // affiche 2
printf("%f", floor(1.9)); // affiche 1
printf("%f", floor(1.5)); // affiche 1
printf("%f", floor(1.4)); // affiche 1
```

Constante ?

- Élément qui ne peut changer de valeur
- Déclaré avec les mot-clé `const` ou `#define`

© Achref EL MOUELHI

Constante ?

- Élément qui ne peut changer de valeur
- Déclaré avec les mot-clé `const` ou `#define`

Déclaration d'une constante

```
const double PI = 3.1415;
```

Constante ?

- Élément qui ne peut changer de valeur
- Déclaré avec les mot-clé `const` ou `#define`

Déclaration d'une constante

```
const double PI = 3.1415;
```

L'instruction déclenche l'erreur suivante : **error : assignment of read-only variable 'pi'**

```
PI = 5;
```

Dans le contexte global, la déclaration pourrait être simplifiée ainsi

```
#define PI 3.14159
```

Exécuter si une condition est vraie

```
if (condition)
{
    ...
}
```

C

Exemple

```
#include <stdio.h>

int main() {
    int x = 3;
    if (x > 0)
    {
        printf("%d est strictement positif\n", x);
    }
    return 0;
}
```

C

Exemple

```
#include <stdio.h>

int main() {
    int x = 3;
    if (x > 0)
    {
        printf("%d est strictement positif\n", x);
    }
    return 0;
}
```

Pour les conditions, on utilise les fonctions ou les opérateurs de comparaison

Opérateurs de comparaison

- `==` : pour tester l'égalité
- `!=` : pour tester l'inégalité
- `>` : supérieur à
- `<` : inférieur à
- `>=` : supérieur ou égal à
- `<=` : inférieur ou égal à

Opérateurs de comparaison

- `==` : pour tester l'égalité
- `!=` : pour tester l'inégalité
- `>` : supérieur à
- `<` : inférieur à
- `>=` : supérieur ou égal à
- `<=` : inférieur ou égal à

En **C**, on ne peut comparer deux valeurs de type incompatible.

Exercice

Écrire un code **C** qui demande à l'utilisateur de saisir un entier positif et qui affiche ensuite sa parité (sans `else`).

Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc `else`)

```
if (condition)
{
    ...
}
else
{
    ...
}
```

C

Exemple

```
#include <stdio.h>

int main() {
    int x = 3;
    if (x > 0)
    {
        printf("%d est strictement positif\n", x);
    }
    else
    {
        printf("%d est strictement négatif ou nul\n", x);
    }
    return 0;
}
```

C

Exercice

Écrire un programme **C** qui permet de déterminer si une chaîne de caractères saisie par l'utilisateur contient un nombre pair ou impair de caractères.

Exercice

Écrire un programme qui demande à l'utilisateur de saisir l'indice d'un mois (entier compris entre 1 et 12) et qui retourne le nombre de jours de ce mois

- si l'entier est égal à 1, 3, 5, 7, 8, 10 ou 12 le programme affiche 31
- sinon si l'entier est égal à 4, 6, 9 ou 11 le programme affiche 30
- sinon si l'entier est égal à 2, le programme demande à l'utilisateur de saisir l'année et lui retourne 29 si l'année est bissextile, 28 sinon (voir https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile)
- pour toute autre valeur, le programme affiche une erreur

C

Boucle `while` : à chaque itération, on teste si la condition est vraie avant d'accéder aux traitements

```
while (condition) {  
    // Instructions  
}
```

© Achref EL

C

Boucle `while` : à chaque itération, on teste si la condition est vraie avant d'accéder aux traitements

```
while (condition) {  
    // Instructions  
}
```

Attention aux boucles infinies, vérifiez que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

C

Exemple

```
int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

© Achref EL MOU

C

Exemple

```
int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

Le résultat est

```
0
1
2
3
4
```

La Boucle `do ... while` **exécute le bloc au moins une fois avant de vérifier la condition**

```
do {  
    // Instructions  
} while (condition);
```

© Achref EL

La Boucle `do ... while` exécute le bloc au moins une fois avant de vérifier la condition

```
do {  
    // Instructions  
} while (condition);
```

Attention aux boucles infinies, vérifiez que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

C

Exemple

```
int i = 0;
do {
    printf("%d\n", i);
    i++;
} while (i < 5);
```

© Achref EL MOU

C

Exemple

```
int i = 0;
do {
    printf("%d\n", i);
    i++;
} while (i < 5);
```

Le résultat est

```
0
1
2
3
4
```

C

Boucle `for`

```
for (initialisation; condition; incr\ementation) {  
    // Instructions  
}
```

© Achref EL M...

Boucle `for`

```
for (initialisation; condition; incr\ementation) {  
    // Instructions  
}
```

Attention aux boucles infinies si vous modifiez la valeur du compteur à l'intérieur de la boucle.

C

Exemple

```
for (int i = 0; i < 5; i++) {  
    printf("%d\n", i);  
}
```

© Achref EL MOUËL

C

Exemple

```
for (int i = 0; i < 5; i++) {  
    printf("%d\n", i);  
}
```

Le résultat est

```
0  
1  
2  
3  
4
```

C

Exemple avec break

```
#include <stdio.h>

int main() {
    int j = 5;
    do {
        printf("%d\n", j);
        if (j == 3) {
            break;
        }
        j--;
    } while (j > 0);
    return 0;
}
```

C

Exemple avec `break`

```
#include <stdio.h>

int main() {
    int j = 5;
    do {
        printf("%d\n", j);
        if (j == 3) {
            break;
        }
        j--;
    } while (j > 0);
    return 0;
}
```

Résultat

5
4

C

Exemple avec `continue`

```
#include <stdio.h>

int main() {
    int j = 5;
    while (j > 0) {
        j--;
        if (j == 3) {
            continue;
        }
        printf("%d\n", j);
    }
    return 0;
}
```

C

Exemple avec `continue`

```
#include <stdio.h>

int main() {
    int j = 5;
    while (j > 0) {
        j--;
        if (j == 3) {
            continue;
        }
        printf("%d\n", j);
    }
    return 0;
}
```

Résultat

```
4
2
1
0
```

C

Exemple avec goto

```
#include <stdio.h>

int main() {
    int limit = 6, compteur = 0;
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < i + 1; j++) {
            printf("*");
            compteur++;
            if (compteur == limit) {
                goto exit;
            }
        }
        printf("\n");
    }
exit:
    return 0;
}
```

Remarque

`break`, `goto` et `continue` rendent le code difficile à lire, à l'exception de l'utilisation de `break` dans `switch`, ils ne doivent pas être utilisés.