Université Abdelmalek Essaâdi
École Normale Supérieure
Tétouan

2023/2024

# Les tableaux

1

## Les tableaux dans la mémoire

63

- Les tableaux sont une suite de variables de même type, situées dans un espace contigu en mémoire.

2

## Les tableaux dans la mémoire

63

- Les tableaux sont une suite de variables de même type, situées dans un espace contigu en mémoire.
- il s'agit de « **grosses variables** » pouvant contenir plusieurs nombres du même type (long, int, char, double…).

3

## Les tableaux dans la mémoire

63

- Les tableaux sont une suite de variables de même type, situées dans un espace contigu en mémoire.
- il s'agit de « **grosses variables** » pouvant contenir plusieurs nombres du même type (long, int, char, double…).
- Un tableau a une dimension bien précise. Il peut occuper 2, 3, 10, 150, 2 500 cases,

4

## Les tableaux dans la mémoire

64

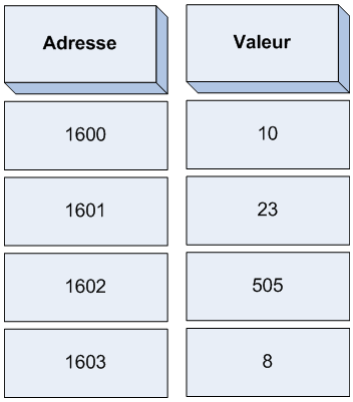
- un schéma d'un tableau de 4 cases en mémoire qui commence à l'adresse 1600.

5

## Les tableaux dans la mémoire

64

- un schéma d'un tableau de 4 cases en mémoire qui commence à l'adresse 1600.




6

## Les tableaux dans la mémoire

65

- Lorsqu'on demande à créer un tableau de 4 cases en mémoire, le programme demande à l'OS la permission d'utiliser 4 cases en mémoire.

7

## Les tableaux dans la mémoire

65

- Lorsqu'on demande à créer un tableau de 4 cases en mémoire, le programme demande à l'OS la permission d'utiliser 4 cases en mémoire.
- Ces 4 cases doivent être contiguës, c'est-à-dire les unes à la suite des autres.

8

## Les tableaux dans la mémoire

65

- Lorsqu'on demande à créer un tableau de 4 cases en mémoire, le programme demande à l'OS la permission d'utiliser 4 cases en mémoire.
- Ces 4 cases doivent être contiguës, c'est-à-dire les unes à la suite des autres.
- chaque case du tableau contient un nombre du même type.

9

## Les tableaux dans la mémoire

65

- Lorsqu'on demande à créer un tableau de 4 cases en mémoire, le programme demande à l'OS la permission d'utiliser 4 cases en mémoire.
- Ces 4 cases doivent être contiguës, c'est-à-dire les unes à la suite des autres.
- chaque case du tableau contient un nombre du même type.
- Si le tableau est de type int, alors chaque case du tableau contiendra un int.

10

## Les tableaux dans la mémoire

65

- Lorsqu'on demande à créer un tableau de 4 cases en mémoire, le programme demande à l'OS la permission d'utiliser 4 cases en mémoire.
- Ces 4 cases doivent être contiguës, c'est-à-dire les unes à la suite des autres.
- chaque case du tableau contient un nombre du même type.
- Si le tableau est de type int, alors chaque case du tableau contiendra un int.
- On ne peut pas faire de tableau contenant à la fois des int et des double

11

## Définir un tableau

66

- **Exemple**: définir un tableau de 4 int :

12

## Définir un tableau

66

- **Exemple**: définir un tableau de 4 int :

```
int tableau[4];
```

13

## Définir un tableau

66

- **Exemple**: définir un tableau de 4 int :

```
int tableau[4];
```

- accéder à chaque case du tableau:

14

## Définir un tableau

66

- **Exemple**: définir un tableau de 4 int :

```
int tableau[4];
```

- accéder à chaque case du tableau:

```
tableau[numeroDeLaCase]
```

15

## Définir un tableau

66

- **Exemple**: définir un tableau de 4 int :

```
int tableau[4];
```

- accéder à chaque case du tableau:

```
tableau[numeroDeLaCase]
```

- un tableau commence à l'indice n° 0 !

16

## Définir un tableau

66

- **Exemple**: définir un tableau de 4 int :

```
int tableau[4];
```

- accéder à chaque case du tableau:

```
tableau[numeroDeLaCase]
```

- un tableau commence à l'indice n° 0 !
- le tableau de 4 int a donc les indices 0, 1, 2 et 3.

17

## Définir un tableau

66

- **Exemple**: définir un tableau de 4 int :

```
int tableau[4];
```

- accéder à chaque case du tableau:

```
tableau[numeroDeLaCase]
```

- un tableau commence à l'indice n° 0 !
- le tableau de 4 int a donc les indices 0, 1, 2 et 3.
- Il n'y a pas d'indice 4 dans un tableau de 4 cases

18

## Définir un tableau

67

- mettre dans mon tableau des valeurs:

19

## Définir un tableau

67

- mettre dans mon tableau des valeurs:

```
int tableau[4];
tableau[0] = 10;
tableau[1] = 23;
tableau[2] = 505;
tableau[3] = 8;
```

20

## Définir un tableau

67

- mettre dans mon tableau des valeurs:

```
int tableau[4];
tableau[0] = 10;
tableau[1] = 23;
tableau[2] = 505;
tableau[3] = 8;
```

- l'adresse où se trouve tableau :

21

## Définir un tableau

67

- mettre dans mon tableau des valeurs:

```
int tableau[4];
tableau[0] = 10;
tableau[1] = 23;
tableau[2] = 505;
tableau[3] = 8;
```

- l'adresse où se trouve tableau :

```
int tableau[4];
printf("%d", tableau);
```

22

## Définir un tableau

67

- mettre dans mon tableau des valeurs:

```
int tableau[4];
tableau[0] = 10;
tableau[1] = 23;
tableau[2] = 505;
tableau[3] = 8;
```

- l'adresse où se trouve tableau :

```
int tableau[4];
printf("%d", tableau);
```

- indiquez l'indice de la case du tableau entre crochets:

23

## Définir un tableau

67

- mettre dans mon tableau des valeurs:

```
int tableau[4];
tableau[0] = 10;
tableau[1] = 23;
tableau[2] = 505;
tableau[3] = 8;
```

- l'adresse où se trouve tableau :

```
int tableau[4];
printf("%d", tableau);
```

- indiquez l'indice de la case du tableau entre crochets:

```
int tableau[4];
printf("%d", tableau[0]);
```

24

## Les tableaux à taille dynamique

68

- Le langage C existe en plusieurs versions.

25

## Les tableaux à taille dynamique

68

- Le langage C existe en plusieurs versions.
- Une version récente, appelée le C99, autorise la création de tableaux à taille dynamique

26

## Les tableaux à taille dynamique

68

- Le langage C existe en plusieurs versions.
- Une version récente, appelée le C99, autorise la création de tableaux à taille dynamique
  - c'est-à-dire de tableaux dont la taille est définie par une variable :

27

## Les tableaux à taille dynamique

68

- Le langage C existe en plusieurs versions.
- Une version récente, appelée le C99, autorise la création de tableaux à taille dynamique
  - c'est-à-dire de tableaux dont la taille est définie par une variable :

```
int taille = 5;
int tableau[taille];
```

28

## Les tableaux à taille dynamique

68

- Le langage C existe en plusieurs versions.
- Une version récente, appelée le C99, autorise la création de tableaux à taille dynamique
  - c'est-à-dire de tableaux dont la taille est définie par une variable :

```
int taille = 5;
int tableau[taille];
```

  - On a pas le droit d'utiliser une variable entre crochets pour la définition de la taille du tableau, même si cette variable est une constante

29

## Les tableaux à taille dynamique

68

- Le langage C existe en plusieurs versions.
- Une version récente, appelée le C99, autorise la création de tableaux à taille dynamique
  - c'est-à-dire de tableaux dont la taille est définie par une variable :

```
int taille = 5;
int tableau[taille];
```

  - On a pas le droit d'utiliser une variable entre crochets pour la définition de la taille du tableau, même si cette variable est une constante
    - On utilise la version C89

30

## Les tableaux à taille dynamique

68

- Le langage C existe en plusieurs versions.
- Une version récente, appelée le C99, autorise la création de tableaux à taille dynamique
  - c'est-à-dire de tableaux dont la taille est définie par une variable :

```
int taille = 5;
int tableau[taille];
```

  - On a pas le droit d'utiliser une variable entre crochets pour la définition de la taille du tableau, même si cette variable est une constante
    - On utilise la version C89
  - On doit écrire noir sur blanc le nombre correspondant à la taille du tableau :

31

## Les tableaux à taille dynamique

68

- Le langage C existe en plusieurs versions.
- Une version récente, appelée le C99, autorise la création de tableaux à taille dynamique
  - c'est-à-dire de tableaux dont la taille est définie par une variable :

```
int taille = 5;
int tableau[taille];
```

  - On a pas le droit d'utiliser une variable entre crochets pour la définition de la taille du tableau, même si cette variable est une constante
    - On utilise la version C89
  - On doit écrire noir sur blanc le nombre correspondant à la taille du tableau :

```
int tableau[5];
```

32

## Les tableaux à taille dynamique

69

- pour faire cela, on utilise une autre technique (plus sûre et qui marche partout) appelée **l'allocation dynamique**.

33

## Parcourir un tableau

70

- afficher les valeurs de chaque case du tableau.

34

## Parcourir un tableau

70

- afficher les valeurs de chaque case du tableau.

```
int main(int argc, char *argv[])
{
        int tableau[4], i = 0;
        tableau[0] = 10;
        tableau[1] = 23;
        tableau[2] = 505;
        tableau[3] = 8;
        for (i = 0 ; i < 4 ; i++)
        {
                printf("%d\n", tableau[i]);
        }
        return 0;
}
```

35

## Parcourir un tableau

70

- afficher les valeurs de chaque case du tableau.
- mettre une variable entre crochets.

```
int main(int argc, char *argv[])
{
        int tableau[4], i = 0;
        tableau[0] = 10;
        tableau[1] = 23;
        tableau[2] = 505;
        tableau[3] = 8;
        for (i = 0 ; i < 4 ; i++)
        {
                printf("%d\n", tableau[i]);
        }
        return 0;
}
```

36

## Parcourir un tableau

70

- afficher les valeurs de chaque case du tableau.

```
int main(int argc, char *argv[])
{
        int tableau[4], i = 0;
        tableau[0] = 10;
        tableau[1] = 23;
        tableau[2] = 505;
        tableau[3] = 8;
        for (i = 0 ; i < 4 ; i++)
        {
                printf("%d\n", tableau[i]);
        }
        return 0;
}
```

- mettre une variable entre crochets.
- En effet, la variable était interdite pour la création du tableau (pour définir sa taille),

37

## Parcourir un tableau

70

- afficher les valeurs de chaque case du tableau.

```
int main(int argc, char *argv[])
{
        int tableau[4], i = 0;
        tableau[0] = 10;
        tableau[1] = 23;
        tableau[2] = 505;
        tableau[3] = 8;
        for (i = 0 ; i < 4 ; i++)
        {
                printf("%d\n", tableau[i]);
        }
        return 0;
}
```

- mettre une variable entre crochets.
- En effet, la variable était interdite pour la création du tableau (pour définir sa taille),
- elle est autorisée pour « parcourir » le tableau,

38

## Parcourir un tableau

70

- afficher les valeurs de chaque case du tableau.

```
int main(int argc, char *argv[])
{
        int tableau[4], i = 0;
        tableau[0] = 10;
        tableau[1] = 23;
        tableau[2] = 505;
        tableau[3] = 8;
        for (i = 0 ; i < 4 ; i++)
        {
                printf("%d\n", tableau[i]);
        }
        return 0;
}
```

- mettre une variable entre crochets.
- En effet, la variable était interdite pour la création du tableau (pour définir sa taille),
- elle est autorisée pour « parcourir » le tableau,
- ne pas tenter d'afficher la valeur de tableau[4]

39

## Initialiser un tableau

71

```
int main(int argc, char *argv[])
{
int tableau[4], i = 0;
// Initialisation du tableau
for (i = 0 ; i < 4 ; i++)
{
tableau[i] = 0;
}
// Affichage de ses valeurs pour vérifier
for (i = 0 ; i < 4 ; i++)
{
printf("%d\n", tableau[i]);
}
return 0;
}
```

40

## Initialiser un tableau

72

- une autre façon d'initialiser un tableau

41

## Initialiser un tableau

72

- une autre façon d'initialiser un tableau
- Elle consiste à écrire

42

## Initialiser un tableau

72

- une autre façon d'initialiser un tableau
- Elle consiste à écrire `tableau[4] = {valeur1, valeur2, valeur3, valeur4}`

43

## Initialiser un tableau

72

- une autre façon d'initialiser un tableau
- Elle consiste à écrire `tableau[4] = {valeur1, valeur2, valeur3, valeur4}`

```
int main(int argc, char *argv[])
{
int tableau[4] = {0, 0, 0, 0}, i = 0;
for (i = 0 ; i < 4 ; i++)
{
printf("%d\n", tableau[i]);
}
return 0;
}
```

44

## Initialiser un tableau

73

- On peut définir les valeurs des premières cases du tableau,

45

## Initialiser un tableau

73

- On peut définir les valeurs des premières cases du tableau,
- toutes celles qu'on n'aura pas renseignées seront automatiquement mises à 0.

46

## Initialiser un tableau

73

- On peut définir les valeurs des premières cases du tableau,
- toutes celles qu'on n'aura pas renseignées seront automatiquement mises à 0.

```
int tableau[4] = {10, 23};
```

47

## Initialiser un tableau

73

- On peut définir les valeurs des premières cases du tableau,
- toutes celles qu'on n'aura pas renseignées seront automatiquement mises à 0.

```
int tableau[4] = {10, 23};
```

- initialiser tout le tableau à 0:

48

## Initialiser un tableau

73

- On peut définir les valeurs des premières cases du tableau,
- toutes celles qu'on n'aura pas renseignées seront automatiquement mises à 0.

```
int tableau[4] = {10, 23};
```

- initialiser tout le tableau à 0:
  - il suffit d'initialiser au moins la première valeur à 0, et toutes les autres valeurs non indiquées prendront la valeur 0.

49

## Initialiser un tableau

73

- On peut définir les valeurs des premières cases du tableau,
- toutes celles qu'on n'aura pas renseignées seront automatiquement mises à 0.

```
int tableau[4] = {10, 23};
```

- initialiser tout le tableau à 0:
  - il suffit d'initialiser au moins la première valeur à 0, et toutes les autres valeurs non indiquées prendront la valeur 0.

```
int tableau[4] = {0}; // Toutes les cases du tableau seront initialisées à 0
```

50

## Initialiser un tableau

73

- On peut définir les valeurs des premières cases du tableau,
- toutes celles qu'on n'aura pas renseignées seront automatiquement mises à 0.

```
int tableau[4] = {10, 23};
```

- initialiser tout le tableau à 0:
  - il suffit d'initialiser au moins la première valeur à 0, et toutes les autres valeurs non indiquées prendront la valeur 0.

```
int tableau[4] = {0}; // Toutes les cases du tableau seront initialisées à 0
```

  - **int tableau[4] = {1};**

51

## Initialiser un tableau

73

- On peut définir les valeurs des premières cases du tableau,
- toutes celles qu'on n'aura pas renseignées seront automatiquement mises à 0.

```
int tableau[4] = {10, 23};
```

- initialiser tout le tableau à 0:
  - il suffit d'initialiser au moins la première valeur à 0, et toutes les autres valeurs non indiquées prendront la valeur 0.

```
int tableau[4] = {0}; // Toutes les cases du tableau seront initialisées à 0
```

  - **int tableau[4] = {1};**
    - insère les valeurs suivantes : 1, 0, 0, 0.

52

## Passage de tableaux à une fonction

74

- Pour afficher tout le contenu du tableau
  - écrire une fonction qui fait ça

53

## Passage de tableaux à une fonction

74

- Pour afficher tout le contenu du tableau
  - écrire une fonction qui fait ça
- envoie un tableau à une fonction

54

## Passage de tableaux à une fonction

74

- Pour afficher tout le contenu du tableau
  - écrire une fonction qui fait ça
- envoie un tableau à une fonction
  - Il va falloir envoyer deux informations à la fonction : le tableau (enfin, l'adresse du tableau) et aussi et surtout sa taille !

55

## Passage de tableaux à une fonction

75

- Une fonction doit être capable d'initialiser un tableau de n'importe quelle taille
  - C'est pour cela qu'il faut envoyer en plus une variable qu'on appellera par exemple **tailleTableau**.

56

## Passage de tableaux à une fonction

75

- Une fonction doit être capable d'initialiser un tableau de n'importe quelle taille
  - C'est pour cela qu'il faut envoyer en plus une variable qu'on appellera par exemple **tailleTableau**.
- le rapport entre les tableaux et les pointeurs:

57

## Passage de tableaux à une fonction

75

- Une fonction doit être capable d'initialiser un tableau de n'importe quelle taille
  - C'est pour cela qu'il faut envoyer en plus une variable qu'on appellera par exemple **tailleTableau**.
- le rapport entre les tableaux et les pointeurs:
  - si on écrit juste **tableau,** on obtient un **pointeur.** C'est un pointeur sur la première case du tableau.

58

## Passage de tableaux à une fonction

75

- Une fonction doit être capable d'initialiser un tableau de n'importe quelle taille
  - C'est pour cela qu'il faut envoyer en plus une variable qu'on appellera par exemple **tailleTableau**.
- le rapport entre les tableaux et les pointeurs:
  - si on écrit juste **tableau,** on obtient un **pointeur**. C'est un pointeur sur la première case du tableau.

```
int tableau[4];
printf("%d", tableau);
```

59

## Passage de tableaux à une fonction

75

- Une fonction doit être capable d'initialiser un tableau de n'importe quelle taille
  - C'est pour cela qu'il faut envoyer en plus une variable qu'on appellera par exemple **tailleTableau**.
- le rapport entre les tableaux et les pointeurs:
  - si on écrit juste **tableau,** on obtient un **pointeur**. C'est un pointeur sur la première case du tableau.

```
int tableau[4];
printf("%d", tableau);
```

  - on peut utiliser le symbole * pour connaître la première valeur :

60

## Passage de tableaux à une fonction

75

- Une fonction doit être capable d'initialiser un tableau de n'importe quelle taille
  - C'est pour cela qu'il faut envoyer en plus une variable qu'on appellera par exemple **tailleTableau**.
- le rapport entre les tableaux et les pointeurs:
  - si on écrit juste **tableau,** on obtient un **pointeur**. C'est un pointeur sur la première case du tableau.

```
int tableau[4];
printf("%d", tableau);
```

  - on peut utiliser le symbole * pour connaître la première valeur :

```
int tableau[4];
printf("%d", *tableau);
```

61

## Passage de tableaux à une fonction

76

- Il est aussi possible d'obtenir la valeur de la seconde case avec ***(tableau + 1) (adresse de tableau + 1).**

62

## Passage de tableaux à une fonction

76

- Il est aussi possible d'obtenir la valeur de la seconde case avec ***(tableau + 1)** **(adresse de tableau + 1)**.
- Les deux lignes suivantes sont donc identiques :

63

## Passage de tableaux à une fonction

76

- Il est aussi possible d'obtenir la valeur de la seconde case avec ***(tableau + 1)** **(adresse de tableau + 1)**.
- Les deux lignes suivantes sont donc identiques :

```
tableau[1] // Renvoie la valeur de la seconde case (la première case étant 0)
*(tableau + 1) // Identique : renvoie la valeur contenue dans la seconde case
```

64

## Passage de tableaux à une fonction

76

- Il est aussi possible d'obtenir la valeur de la seconde case avec ***(tableau + 1)** **(adresse de tableau + 1)**.
- Les deux lignes suivantes sont donc identiques :

```
tableau[1] // Renvoie la valeur de la seconde case (la première case étant 0)
*(tableau + 1) // Identique : renvoie la valeur contenue dans la seconde case
```

- tableau peut être considéré comme un pointeur.

65

## Passage de tableaux à une fonction

76

- Il est aussi possible d'obtenir la valeur de la seconde case avec ***(tableau + 1)** **(adresse de tableau + 1)**.
- Les deux lignes suivantes sont donc identiques :

```
tableau[1] // Renvoie la valeur de la seconde case (la première case étant 0)
*(tableau + 1) // Identique : renvoie la valeur contenue dans la seconde case
```

- tableau peut être considéré comme un pointeur.
  - On peut donc l'envoyer à la fonction comme on l'aurait fait avec un vulgaire pointeur :

66

## Passage de tableaux à une fonction

76

- Il est aussi possible d'obten… +
  **1) (adresse de tableau + 1**
- Les deux lignes suivantes so…

```
tableau[1] // Renvoie la valeur
*(tableau + 1) // Identique : re
```

- tableau peut être considér…
  - On peut donc l'envoyer à la… pointeur :

```
// Prototype de la fonction d'affichage
void affiche(int *tableau, int tailleTableau);
int main(int argc, char *argv[])
{
int tableau[4] = {10, 15, 3};
// On affiche le contenu du tableau
affiche(tableau, 4);
return 0;
}
void affiche(int *tableau, int tailleTableau)
{
int i;
for (i = 0 ; i < tailleTableau ; i++)
{
printf("%d\n", tableau[i]);
}
}
```

67

## Passage de tableaux à une fonction

77

- Elle prend en paramètre un pointeur sur **int** (le tableau), ainsi que la taille du tableau (très important pour savoir quand s'arrêter dans la boucle !).

68

## Passage de tableaux à une fonction

77

- Elle prend en paramètre un pointeur sur **int** (le tableau), ainsi que la taille du tableau (très important pour savoir quand s'arrêter dans la boucle !).
- Tout le contenu du tableau est affiché par la fonction via une boucle.

69

## Passage de tableaux à une fonction

77

- Elle prend en paramètre un pointeur sur **int** (le tableau), ainsi que la taille du tableau (très important pour savoir quand s'arrêter dans la boucle !).
- Tout le contenu du tableau est affiché par la fonction via une boucle.
- une autre façon d'indiquer que la fonction reçoit un tableau. Plutôt que d'indiquer que la fonction attend un **int *tableau:**

70

## Passage de tableaux à une fonction

77

- Elle prend en paramètre un pointeur sur **int** (le tableau), ainsi que la taille du tableau (très important pour savoir quand s'arrêter dans la boucle !).
- Tout le contenu du tableau est affiché par la fonction via une boucle.
- une autre façon d'indiquer que la fonction reçoit un tableau. Plutôt que d'indiquer que la fonction attend un **int *tableau:**

```
void affiche(int tableau[], int tailleTableau)
```

71

## Passage de tableaux à une fonction

77

- Elle prend en paramètre un pointeur sur **int** (le tableau), ainsi que la taille du tableau (très important pour savoir quand s'arrêter dans la boucle !).
- Tout le contenu du tableau est affiché par la fonction via une boucle.
- une autre façon d'indiquer que la fonction reçoit un tableau. Plutôt que d'indiquer que la fonction attend un **int *tableau:**

```
void affiche(int tableau[], int tailleTableau)
```

- la présence des crochets permet au programmeur de bien voir que c'est un tableau que la fonction prend, et non un simple pointeur.

72

## Passage de tableaux à une fonction

77

- Elle prend en paramètre un pointeur sur **int** (le tableau), ainsi que la taille du tableau (très important pour savoir quand s'arrêter dans la boucle !).
- Tout le contenu du tableau est affiché par la fonction via une boucle.
- une autre façon d'indiquer que la fonction reçoit un tableau. Plutôt que d'indiquer que la fonction attend un **int *tableau:**

```
void affiche(int tableau[], int tailleTableau)
```

- la présence des crochets permet au programmeur de bien voir que c'est un tableau que la fonction prend, et non un simple pointeur.
  - Cela permet d'éviter des confusions.

73

## Quelques exercices

78

**Exercice 1**

74

## Quelques exercices

78

**Exercice 1**

- Créez une fonction **sommeTableau** qui renvoie la somme des valeurs contenues dans le tableau (utilisez un return pour renvoyer la valeur).

75

## Quelques exercices

78

**Exercice 1**

- Créez une fonction **sommeTableau** qui renvoie la somme des valeurs contenues dans le tableau (utilisez un return pour renvoyer la valeur).
- le prototype de la fonction à créer :

76

## Quelques exercices

78

**Exercice 1**

- Créez une fonction **sommeTableau** qui renvoie la somme des valeurs contenues dans le tableau (utilisez un return pour renvoyer la valeur).
- le prototype de la fonction à créer :

```
int sommeTableau(int tableau[], int tailleTableau);
```

77

## Quelques exercices

78

**Exercice 1**

- Créez une fonction **sommeTableau** qui renvoie la somme des valeurs contenues dans le tableau (utilisez un return pour renvoyer la valeur).
- le prototype de la fonction à créer :

```
int sommeTableau(int tableau[], int tailleTableau);
```

**Exercice 2**

78

## Quelques exercices

78

**Exercice 1**

- Créez une fonction **sommeTableau** qui renvoie la somme des valeurs contenues dans le tableau (utilisez un return pour renvoyer la valeur).
- le prototype de la fonction à créer :

```
int sommeTableau(int tableau[], int tailleTableau);
```

**Exercice 2**

- Créez une fonction **moyenneTableau** qui calcule et renvoie la moyenne des valeurs.

79

## Quelques exercices

78

**Exercice 1**

- Créez une fonction **sommeTableau** qui renvoie la somme des valeurs contenues dans le tableau (utilisez un return pour renvoyer la valeur).
- le prototype de la fonction à créer :

```
int sommeTableau(int tableau[], int tailleTableau);
```

**Exercice 2**

- Créez une fonction **moyenneTableau** qui calcule et renvoie la moyenne des valeurs.
- Prototype :

80

## Quelques exercices

78

**Exercice 1**

- Créez une fonction **sommeTableau** qui renvoie la somme des valeurs contenues dans le tableau (utilisez un return pour renvoyer la valeur).
- le prototype de la fonction à créer :

```
int sommeTableau(int tableau[], int tailleTableau);
```

**Exercice 2**

- Créez une fonction **moyenneTableau** qui calcule et renvoie la moyenne des valeurs.
- Prototype :

```
int moyenneTableau(int tableau[], int tailleTableau);
```

81

## Quelques exercices

79

**Exercice 3**

- Créez une fonction **copierTableau** qui prend en paramètre deux tableaux. Le contenu du premier tableau devra être copié dans le second tableau.

82

## Quelques exercices

79

**Exercice 3**

- Créez une fonction **copierTableau** qui prend en paramètre deux tableaux. Le contenu du premier tableau devra être copié dans le second tableau.
- Prototype :

83

## Quelques exercices

79

**Exercice 3**

- Créez une fonction **copierTableau** qui prend en paramètre deux tableaux. Le contenu du premier tableau devra être copié dans le second tableau.
- Prototype :

```
void copie(int tableauOriginal[], int tableauCopie[], int tailleTableau);
```

84

## Quelques exercices

79

**Exercice 3**

- Créez une fonction **copierTableau** qui prend en paramètre deux tableaux. Le contenu du premier tableau devra être copié dans le second tableau.
- Prototype :

```
void copie(int tableauOriginal[], int tableauCopie[], int tailleTableau);
```

**Exercice 4**

85

## Quelques exercices

79

**Exercice 3**

- Créez une fonction **copierTableau** qui prend en paramètre deux tableaux. Le contenu du premier tableau devra être copié dans le second tableau.
- Prototype :

```
void copie(int tableauOriginal[], int tableauCopie[], int tailleTableau);
```

**Exercice 4**

- Créez une fonction **maximumTableau** qui aura pour rôle de remettre à 0 toutes les cases du tableau ayant une valeur supérieure à un maximum. Cette fonction prendra en paramètres le tableau ainsi que le nombre maximum autorisé (**valeurMax**). Toutes les cases qui contiennent un nombre supérieur à **valeurMax** doivent être mises à 0.

86

## Quelques exercices

79

**Exercice 3**

- Créez une fonction **copierTableau** qui prend en paramètre deux tableaux. Le contenu du premier tableau devra être copié dans le second tableau.
- Prototype :

```
void copie(int tableauOriginal[], int tableauCopie[], int tailleTableau);
```

**Exercice 4**

- Créez une fonction **maximumTableau** qui aura pour rôle de remettre à 0 toutes les cases du tableau ayant une valeur supérieure à un maximum. Cette fonction prendra en paramètres le tableau ainsi que le nombre maximum autorisé (**valeurMax**). Toutes les cases qui contiennent un nombre supérieur à **valeurMax** doivent être mises à 0.
- Prototype :

87

## Quelques exercices

79

**Exercice 3**

- Créez une fonction **copierTableau** qui prend en paramètre deux tableaux. Le contenu du premier tableau devra être copié dans le second tableau.
- Prototype :

```
void copie(int tableauOriginal[], int tableauCopie[], int tailleTableau);
```

**Exercice 4**

- Créez une fonction **maximumTableau** qui aura pour rôle de remettre à 0 toutes les cases du tableau ayant une valeur supérieure à un maximum. Cette fonction prendra en paramètres le tableau ainsi que le nombre maximum autorisé (**valeurMax**). Toutes les cases qui contiennent un nombre supérieur à **valeurMax** doivent être mises à 0.
- Prototype :

```
void maximumTableau(int tableau[], int tailleTableau, int valeurMax);
```

88

## Quelques exercices

80

**Exercice 5**

- Créez une fonction **ordonnerTableau** qui classe les valeurs d'un tableau dans l'ordre croissant. Ainsi, un tableau qui vaut {15, 81, 22, 13} doit à la fin de la fonction valoir {13, 15, 22, 81}.
- Prototype :

```
void ordonnerTableau(int tableau[], int tailleTableau);
```

89

## Les tableaux à deux dimensions

81

- il est possible en C de créer des tableaux à deux dimensions, aussi appeler matrice
- on rajoute bien évidemment la deuxième paire de crochets pour signaler qu'on utilisera une matrice.

90

## Les tableaux à deux dimensions

81

- il est possible en C de créer des tableaux à deux dimensions, aussi appeler matrice
- on rajoute bien évidemment ler qu'on utilisera une matrice.

```
#include <stdio.h>
#include <stdlib.h>
void afficherTableau(int tableau[2][2]);
int main(void)
{
    int tableau[2][2] = {{10, 20} , {15, 35}};
    afficherTableau(tableau);

      return 0;
}
void afficherTableau(int tableau[2][2])
{
    /* ... */
}
```

91

## Les tableaux à deux dimensions

82

- récupérer les valeurs du tableau

82

```c
#include <stdio.h>
#include <stdlib.h>
void afficherTableau(int tableau[2][2]);
int main(void)
{
    int tableau[2][2] = {{10, 20} , {15, 35}};
    afficherTableau(tableau);

      return EXIT_SUCCESS; /* Equivalent à return 0 sous Windows, permet un code portable */
}
void afficherTableau(int tableau[2][2])
{
    int i = 0;
    int j = 0;

    for (i = 0; i < 2; i++) {
        for(j = 0; j < 2; j++) {
            printf("Tableau[%d][%d] = %d\n", i, j, tableau[i][j]);
        }
    }
}
```

93

## Les tableaux à deux dimensions

83

- Ecrire la fonction LIRE_DIM à quatre paramètres L, LMAX, C, CMAX qui lit les dimensions L et C d'une matrice à deux dimensions. Les dimensions L et C doivent être inférieures à LMAX respectivement CMAX.

94

## Les tableaux à deux dimensions

83

- Ecrire la fonction LIRE_DIM à quatre paramètres L, LMAX, C, CMAX qui lit les dimensions L et C d'une matrice à deux dimensions. Les dimensions L et C doivent être inférieures à LMAX respectivement CMAX.
- Ecrire la fonction LIRE_MATRICE à quatre paramètres MAT, L, C, et CMAX qui lit les composantes d'une matrice MAT du type int et de dimensions L et C.

95

## Les tableaux à deux dimensions

83

- Ecrire la fonction LIRE_DIM à quatre paramètres L, LMAX, C, CMAX qui lit les dimensions L et C d'une matrice à deux dimensions. Les dimensions L et C doivent être inférieures à LMAX respectivement CMAX.
- Ecrire la fonction LIRE_MATRICE à quatre paramètres MAT, L, C, et CMAX qui lit les composantes d'une matrice MAT du type int et de dimensions L et C.
- Implémenter les fonctions en choisissant bien le type des paramètres et utiliser un dialogue semblable à celui de LIRE_TAB.

96

## Les tableaux à deux dimensions



- Ecrire la fonction ECRIRE_MATRICE à quatre paramètres MAT, L, C et CMAX qui affiche les composantes de la matrice de dimensions L et C.

97

## Les tableaux à deux dimensions



- Ecrire la fonction ECRIRE_MATRICE à quatre paramètres MAT, L, C et CMAX qui affiche les composantes de la matrice de dimensions L et C.
- Ecrire la fonction ADDITION_MATRICE qui effectue l'addition des matrices suivante:

98

## Les tableaux à deux dimensions

84

- Ecrire la fonction ECRIRE_MATRICE à quatre paramètres MAT, L, C et CMAX qui affiche les composantes de la matrice de dimensions L et C.
- Ecrire la fonction ADDITION_MATRICE qui effectue l'addition des matrices suivante:
  - MAT1 = MAT1 + MAT2

99

## Les tableaux à deux dimensions

84

- Ecrire la fonction ECRIRE_MATRICE à quatre paramètres MAT, L, C et CMAX qui affiche les composantes de la matrice de dimensions L et C.
- Ecrire la fonction ADDITION_MATRICE qui effectue l'addition des matrices suivante:
  - MAT1 = MAT1 + MAT2
  - Choisir les paramètres nécessaires et écrire un petit programme qui teste la fonction ADDITION_MATRICE.

100

Université Abdelmalek Essaâdi
École Normale Supérieure
Tétouan

2024/2025

# Les chaînes de caractères

1

## Les chaînes de caractères

2

- Une « chaîne de caractères », c'est un nom programmatiquement correct pour désigner… du texte.

2

## Les chaînes de caractères

2

- Une « chaîne de caractères », c'est un nom programmatiquement correct pour désigner… du texte.
  - du texte que l'on peut retenir sous forme de variable en mémoire.
  - on pourrait stocker le nom de l'utilisateur.

3

## Le type char

3

- le type **char** permet de stocker des nombres compris entre -128 et 127:

4

## Le type char

3

- le type **char** permet de stocker des nombres compris entre -128 et 127:
  - Si ce type **char** permet de stocker des nombres, il faut savoir qu'en **C** on l'utilise rarement pour ça.

5

## Le type char

3

- le type **char** permet de stocker des nombres compris entre -128 et 127:
  - Si ce type **char** permet de stocker des nombres, il faut savoir qu'en **C** on l'utilise rarement pour ça.
  - En général, même si le nombre est petit, on le stocke dans un **int.**

6

## Le type char

3

- le type **char** permet de stocker des nombres compris entre -128 et 127:
  - Si ce type **char** permet de stocker des nombres, il faut savoir qu'en **C** on l'utilise rarement pour ça.
  - En général, même si le nombre est petit, on le stocke dans un **int.**
  - ça prend un peu plus de place en mémoire, mais aujourd'hui, la mémoire, ce n'est vraiment pas ce qui manque sur un ordinateur.

7

## Le type char

3

- le type **char** permet de stocker des nombres compris entre -128 et 127:
  - Si ce type **char** permet de stocker des nombres, il faut savoir qu'en **C** on l'utilise rarement pour ça.
  - En général, même si le nombre est petit, on le stocke dans un **int.**
  - ça prend un peu plus de place en mémoire, mais aujourd'hui, la mémoire, ce n'est vraiment pas ce qui manque sur un ordinateur.
  - Le type **char** est en fait prévu pour stocker… une lettre !

8

## Le type char

4

- la mémoire ne peut stocker que des nombres

9

## Le type char

4

- la mémoire ne peut stocker que des nombres
  - une table qui fait la conversion entre les nombres et les lettres.

10

## Le type char

4

- la mémoire ne peut stocker que des nombres
  - une table qui fait la conversion entre les nombres et les lettres.
  - par exemple : le nombre 65 équivaut à la lettre A.

11

## Le type char

4

- la mémoire ne peut stocker que des nombres
  - une table qui fait la conversion entre les nombres et les lettres.
  - par exemple : le nombre 65 équivaut à la lettre A.
  - Le langage C permet de faire très facilement la traduction lettre <=> nombre correspondant.

12

## Le type char

4

- la mémoire ne peut stocker que des nombres
  - une table qui fait la conversion entre les nombres et les lettres.
  - par exemple : le nombre 65 équivaut à la lettre A.
  - Le langage C permet de faire très facilement la traduction lettre <=> nombre correspondant.
  - Pour obtenir le nombre associé à une lettre, il suffit d'écrire cette lettre entre apostrophes : 'A'

13

## Le type char

4

- la mémoire ne peut stocker que des nombres
  - une table qui fait la conversion entre les nombres et les lettres.
  - par exemple : le nombre 65 équivaut à la lettre A.
  - Le langage C permet de faire très facilement la traduction lettre <=> nombre correspondant.
  - Pour obtenir le nombre associé à une lettre, il suffit d'écrire cette lettre entre apostrophes : 'A'
  - À la compilation, 'A' sera remplacé par la valeur correspondante.

14

## Le type char

5

- Testez :

```
int main(int argc, char *argv[])
{
        char lettre = 'A';
        printf("%d\n", lettre);
        return 0;
}
```

15

## Le type char

5

- Testez :

```
int main(int argc, char *argv[])
{
        char lettre = 'A';
        printf("%d\n", lettre);
        return 0;
}
```

- la lettre A majuscule est représentée par le nombre 65. B vaut 66, C vaut 67, etc.

16

## Le type char

5

- Testez :

```
int main(int argc, char *argv[])
{
        char lettre = 'A';
        printf("%d\n", lettre);
        return 0;
}
```

- la lettre A majuscule est représentée par le nombre 65. B vaut 66, C vaut 67, etc.
- Testez avec des minuscules

17

## Le type char

5

- Testez :

```
int main(int argc, char *argv[])
{
        char lettre = 'A';
        printf("%d\n", lettre);
        return 0;
}
```

- la lettre A majuscule est représentée par le nombre 65. B vaut 66, C vaut 67, etc.
- Testez avec des minuscules
- la lettre 'a' n'est pas identique à la lettre 'A'

18

## Le type char

5

- Testez :

```
int main(int argc, char *argv[])
{
        char lettre = 'A';
        printf("%d\n", lettre);
        return 0;
}
```

- la lettre A majuscule est représentée par le nombre 65. B vaut 66, C vaut 67, etc.
- Testez avec des minuscules
- la lettre 'a' n'est pas identique à la lettre 'A'
- l'ordinateur faisant la différence entre les majuscules et les minuscules (on dit qu'il « respecte la casse »).

19

## Le type char

6

20

## Le type char

6

- La plupart des caractères « de base » sont codés entre les nombres 0 et 127.

21

## Le type char

6

- La plupart des caractères « de base » sont codés entre les nombres 0 et 127.
- Une table fait la conversion entre les nombres et les lettres : la table ASCII (prononcez « Aski »).

22

## Le type char

6

- La plupart des caractères « de base » sont codés entre les nombres 0 et 127.
- Une table fait la conversion entre les nombres et les lettres : la table ASCII (prononcez « Aski »).
- Le site AsciiTable.com est célèbre pour proposer cette table mais ce n'est pas le seul,

23

## Le type char

6

- La plupart des caractères « de base » sont codés entre les nombres 0 et 127.
- Une table fait la conversion entre les nombres et les lettres : la table ASCII (prononcez « Aski »).
- Le site AsciiTable.com est célèbre pour proposer cette table mais ce n'est pas le seul,
- on peut aussi la retrouver sur Wikipédia et bien d'autres sites encore.

24

## Afficher un caractère

7

- La fonction **printf**

25

## Afficher un caractère

7

- La fonction **printf**
- on doit utiliser le symbole **%c**

26

## Afficher un caractère

7

- La fonction **printf**
- on doit utiliser le symbole **%c**

```
int main(int argc, char *argv[])
{
        char lettre = 'A';
        printf("%c\n", lettre);
        return 0;
}
```

27

## Afficher un caractère

7

- La fonction **printf**
- on doit utiliser le symbole **%c**

```
int main(int argc, char *argv[])
{
        char lettre = 'A';
        printf("%c\n", lettre);
        return 0;
}
```

- On peut aussi demander à l'utilisateur d'entrer une lettre en utilisant le **%c** dans un **scanf** :

28

## Afficher un caractère

7

- La fonction **printf**
- on doit utiliser le symbole **%c**

```
int main(int argc, char *argv[])
{
        char lettre = 'A';
        printf("%c\n", lettre);
        return 0;
}
```

- On peut aussi demander à l'utilisateur d'entrer une lettre en utilisant le **%c** dans un **scanf** :

```
int main(int argc, char *argv[])
{
        char lettre = 0;
        scanf("%c", &lettre);
        printf("%c\n", lettre);
        return 0;
}
```

29

## Le type char

8

30

## Le type char

8

- le type **char** permet de stocker des nombres allant de -128 à 127, **unsigned char** des nombres de 0 à 255 ;

31

## Le type char

8

- le type **char** permet de stocker des nombres allant de -128 à 127, **unsigned char** des nombres de 0 à 255 ;
- il y a une table que votre ordinateur utilise pour convertir les lettres en nombres et inversement, la table ASCII ;

32

## Le type char

8

- le type **char** permet de stocker des nombres allant de -128 à 127, **unsigned char** des nombres de 0 à 255 ;
- il y a une table que votre ordinateur utilise pour convertir les lettres en nombres et inversement, la table ASCII ;
- on peut donc utiliser le type **char** pour stocker UNE lettre ;

33

## Le type char

8

- le type **char** permet de stocker des nombres allant de -128 à 127, **unsigned char** des nombres de 0 à 255 ;
- il y a une table que votre ordinateur utilise pour convertir les lettres en nombres et inversement, la table ASCII ;
- on peut donc utiliser le type **char** pour stocker UNE lettre ;
- **'A'** est remplacé à la compilation par la valeur correspondante (65 en l'occurrence). On utilise donc les apostrophes pour obtenir la valeur d'une lettre.

34

## Les chaînes sont des tableaux de char

9

- une chaîne de caractères n'est rien d'autre qu'un tableau de type **char**

35

## Les chaînes sont des tableaux de char

9

- une chaîne de caractères n'est rien d'autre qu'un tableau de type **char**

```
char chaine[5];
```

36

## Les chaînes sont des tableaux de char

9

- une chaîne de caractères n'est rien d'autre qu'un tableau de type **char**

```
char chaine[5];
```

- on met dans **chaine[0]** la lettre **'S'**, dans **chaine[1]** la lettre **'a'**

37

## Les chaînes sont des tableaux de char

9

- une chaîne de caractères n'est rien d'autre qu'un tableau de type **char**

```
char chaine[5];
```

- on met dans **chaine[0]** la lettre **'S'**, dans **chaine[1]** la lettre **'a'**
- on peut ainsi former une chaîne de caractères, c'est-à-dire du texte.

38

## Les chaînes sont des tableaux de char

10

- la façon dont la chaîne est stockée en mémoire

| Adresse | Valeur |
|---|---|
| 18000 | 'S' |
| 18001 | 'a' |
| 18002 | 'l' |
| 18003 | 'u' |
| 18004 | 't' |

39

## Les chaînes sont des tableaux de char

10

- la façon dont la chaîne est stockée en mémoire
- un tableau qui prend 5 cases en mémoire pour représenter le mot « Salut ».

| Adresse | Valeur |
|---|---|
| 18000 | 'S' |
| 18001 | 'a' |
| 18002 | 'l' |
| 18003 | 'u' |
| 18004 | 't' |

40

## Les chaînes sont des tableaux de char

10

- la façon dont la chaîne est stockée en mémoire
- un tableau qui prend 5 cases en mémoire pour représenter le mot « Salut ».
- les lettres entre apostrophes pour indiquer que c'est un nombre qui est stocké, et non une lettre

| Adresse | Valeur |
|---|---|
| 18000 | 'S' |
| 18001 | 'a' |
| 18002 | 'l' |
| 18003 | 'u' |
| 18004 | 't' |

41

## Les chaînes sont des tableaux de char

10

- la façon dont la chaîne est stockée en mémoire
- un tableau qui prend 5 cases en mémoire pour représenter le mot « Salut ».
- les lettres entre apostrophes pour indiquer que c'est un nombre qui est stocké, et non une lettre
  - En réalité, dans la mémoire, ce sont bel et bien les nombres correspondant à ces lettres qui sont stockés.

| Adresse | Valeur |
|---|---|
| 18000 | 'S' |
| 18001 | 'a' |
| 18002 | 'l' |
| 18003 | 'u' |
| 18004 | 't' |

42

## Les chaînes sont des tableaux de char

11

- une chaîne de caractères ne contient pas que des lettres !

43

## Les chaînes sont des tableaux de char

11

- une chaîne de caractères ne contient pas que des lettres !
- Une chaîne de caractère **doit impérativement contenir un caractère spécial à la fin de la chaîne,** appelé « caractère de fin de chaîne ».

44

## Les chaînes sont des tableaux de char

11

- une chaîne de caractères ne contient pas que des lettres !
- Une chaîne de caractère **doit impérativement contenir un caractère spécial à la fin de la chaîne,** appelé « caractère de fin de chaîne ».
  - ce caractère s'écrit **'\0'.**

45

## Les chaînes sont des tableaux de char

11

- une chaîne de caractères ne contient pas que des lettres !
- Une chaîne de caractère **doit impérativement contenir un caractère spécial à la fin de la chaîne,** appelé « caractère de fin de chaîne ».
  - ce caractère s'écrit **'\0'.**
  - pour que l'ordinateur sache quand s'arrête la chaîne !

46

## Les chaînes sont des tableaux de char

11

- une chaîne de caractères ne contient pas que des lettres !
- Une chaîne de caractère **doit impérativement contenir un caractère spécial à la fin de la chaîne,** appelé « caractère de fin de chaîne ».
  - ce caractère s'écrit **'\0'.**
  - pour que l'ordinateur sache quand s'arrête la chaîne !
  - Le caractère **\0** permet de dire : « Stop, c'est fini, y'a plus rien à lire après

47

## Les chaînes sont des tableaux de char

11

- une chaîne de caractères ne contient pas que des lettres !
- Une chaîne de caractère **doit impérativement contenir un caractère spécial à la fin de la chaîne,** appelé « caractère de fin de chaîne ».
  - ce caractère s'écrit **'\0'.**
  - pour que l'ordinateur sache quand s'arrête la chaîne !
  - Le caractère **\0** permet de dire : « Stop, c'est fini, y'a plus rien à lire après
- pour stocker le mot « Salut » (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 **char,** mais de 6 !

48

## Les chaînes sont des tableaux de char

11

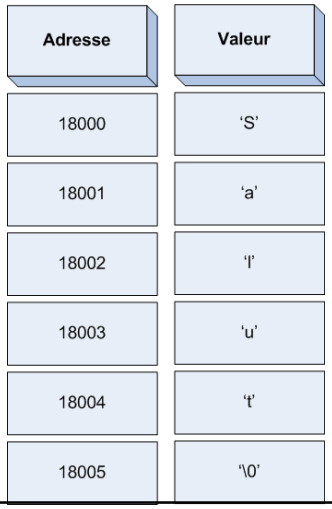
- une chaîne de caractères ne contient pas que des lettres !
- Une chaîne de caractère **doit impérativement contenir un caractère spécial à la fin de la chaîne**, appelé « caractère de fin de chaîne ».
  - ce caractère s'écrit **'\0'**.
  - pour que l'ordinateur sache quand s'arrête la chaîne !
  - Le caractère **\0** permet de dire : « Stop, c'est fini, y'a plus rien à lire après
- pour stocker le mot « Salut » (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 **char**, mais de 6 !
- Oublier le caractère de fin **\0** est une source d'erreurs

49

## Les chaînes sont des tableaux de char

12

- le schéma correct de la représentation de la chaîne de caractères « Salut » en mémoire.




50

## Les chaînes sont des tableaux de char

13

- le caractère **\0** comme un avantage.

51

## Les chaînes sont des tableaux de char

13

- le caractère **\0** comme un avantage.
  - grâce à lui, on n'aura pas à retenir la taille du tableau car il indique que le tableau s'arrête à cet endroit.

52

## Les chaînes sont des tableaux de char

13

- le caractère **\0** comme un avantage.
  - grâce à lui, on n'aura pas à retenir la taille du tableau car il indique que le tableau s'arrête à cet endroit.
  - on pourra passer le tableau de char à une fonction sans avoir à ajouter à côté une variable indiquant la taille du tableau.

53

## Les chaînes sont des tableaux de char

13

- le caractère **\0** comme un avantage.
  - grâce à lui, on n'aura pas à retenir la taille du tableau car il indique que le tableau s'arrête à cet endroit.
  - on pourra passer le tableau de char à une fonction sans avoir à ajouter à côté une variable indiquant la taille du tableau.
  - cela n'est valable que pour les chaînes de caractères (c'est-à-dire le type char*, qu'on peut aussi écrire char[]).

54

## Les chaînes sont des tableaux de char

13

- le caractère **\0** comme un avantage.
  - grâce à lui, on n'aura pas à retenir la taille du tableau car il indique que le tableau s'arrête à cet endroit.
  - on pourra passer le tableau de char à une fonction sans avoir à ajouter à côté une variable indiquant la taille du tableau.
  - cela n'est valable que pour les chaînes de caractères (c'est-à-dire le type char*, qu'on peut aussi écrire char[]).
  - Pour les autres types de tableaux, on est toujours obligés de retenir la taille du tableau quelque part.

55

## Création et initialisation de la chaîne

14

- la méthode manuelle mais peu efficace :

56

## Création et initialisation de la chaîne

14

- la méthode manuelle mais peu efficace :

```
char chaine[6]; // Tableau de 6 char pour stocker S-a-l-u-t + le \0
chaine[0] = 'S';
chaine[1] = 'a';
chaine[2] = 'l';
chaine[3] = 'u';
chaine[4] = 't';
chaine[5] = '\0';
```

57

## Création et initialisation de la chaîne

15

- Pour faire un **printf** il faut utiliser le symbole **%s** (**s** comme *string*}, qui signifie « chaîne » en anglais).

58

## Création et initialisation de la chaîne

15

- Pour faire un **printf** il faut utiliser le symbole **%s** (**s** comme *string*}, qui signifie « chaîne » en anglais).

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
        char chaine[6]; // Tableau de 6 char pour stocker S-a-l-u-t + le \0
        // Initialisation de la chaîne (on écrit les caractères un à un en mémoire)
        chaine[0] = 'S';
        chaine[1] = 'a';
        chaine[2] = 'l';
        chaine[3] = 'u';
        chaine[4] = 't';
        chaine[5] = '\0';
        printf("%s", chaine); // Affichage de la chaîne grâce au %s du printf
        return 0;
}
```

59

## Création et initialisation de la chaîne

16

- il existe **une méthode plus simple :**

60

## Création et initialisation de la chaîne

16

- il existe **une méthode plus simple :**

```
int main(int argc, char *argv[])  {
        char chaine[] = "Salut"; // La taille du tableau chaine est automatiquement calculée
        printf("%s", chaine);
        return 0;
}
```

61

## Création et initialisation de la chaîne

16

- il existe **une méthode plus simple :**

```
int main(int argc, char *argv[])  {
        char chaine[] = "Salut"; // La taille du tableau chaine est automatiquement calculée
        printf("%s", chaine);
        return 0;
}
```

- une variable de type char[].

62

## Création et initialisation de la chaîne

16

- il existe **une méthode plus simple :**

```
int main(int argc, char *argv[])  {
        char chaine[] = "Salut"; // La taille du tableau chaine est automatiquement calculée
        printf("%s", chaine);
        return 0;
}
```

- une variable de type char[].
- on aura pu écrire aussi char*

63

## Création et initialisation de la chaîne

16

- il existe **une méthode plus simple :**

```
int main(int argc, char *argv[])  {
        char chaine[] = "Salut"; // La taille du tableau chaine est automatiquement calculée
        printf("%s", chaine);
        return 0;
}
```

- une variable de type char[].
- on aura pu écrire aussi char*
- Il y a toutefois un défaut : ça ne marche que pour l'initialisation !

64

## Création et initialisation de la chaîne

16

- il existe **une méthode plus simple :**

```
int main(int argc, char *argv[])  {
        char chaine[] = "Salut"; // La taille du tableau chaine est automatiquement calculée
        printf("%s", chaine);
        return 0;
}
```

- une variable de type char[].
- on aura pu écrire aussi char*
- Il y a toutefois un défaut : ça ne marche que pour l'initialisation !
  - on ne pout pas écrire plus loin dans le code :

65

## Création et initialisation de la chaîne

16

- il existe **une méthode plus simple :**

```
int main(int argc, char *argv[])  {
        char chaine[] = "Salut"; // La taille du tableau chaine est automatiquement calculée
        printf("%s", chaine);
        return 0;
}
```

- une variable de type char[].
- on aura pu écrire aussi char*
- Il y a toutefois un défaut : ça ne marche que pour l'initialisation !
  - on ne pout pas écrire plus loin dans le code : `chaine = "Salut";`

66

## Création et initialisation de la chaîne

16

- il existe **une méthode plus simple** :

```
int main(int argc, char *argv[])  {
        char chaine[] = "Salut"; // La taille du tableau chaine est automatiquement calculée
        printf("%s", chaine);
        return 0;
}
```

- une variable de type char[].
- on aura pu écrire aussi char*
- Il y a toutefois un défaut : ça ne marche que pour l'initialisation !
  - on ne pout pas écrire plus loin dans le code : `chaine = "Salut";`
  - Cette technique est donc à réserver à l'initialisation. Après cela, il faudra écrire les caractères manuellement un à un en mémoire

67

## Affichage de chaînes de caractères

17

- **puts** est idéale pour écrire une chaîne constante ou le contenu d'une variable dans une ligne isolée.

68

## Affichage de chaînes de caractères

17

- **puts** est idéale pour écrire une chaîne constante ou le contenu d'une variable dans une ligne isolée.
- Syntaxe:

69

## Affichage de chaînes de caractères

17

- **puts** est idéale pour écrire une chaîne constante ou le contenu d'une variable dans une ligne isolée.
- Syntaxe:
  - **puts( <Chaîne> )**

70

## Affichage de chaînes de caractères

17

- **puts** est idéale pour écrire une chaîne constante ou le contenu d'une variable dans une ligne isolée.
- Syntaxe:
  - **puts( <Chaîne> )**
- Effet:

71

## Affichage de chaînes de caractères

17

- **puts** est idéale pour écrire une chaîne constante ou le contenu d'une variable dans une ligne isolée.
- Syntaxe:
  - **puts( <Chaîne> )**
- Effet:
  - puts écrit la chaîne de caractères désignée par <Chaîne> sur stdout et provoque un retour à la ligne. En pratique,

72

## Affichage de chaînes de caractères

17

- **puts** est idéale pour écrire une chaîne constante ou le contenu d'une variable dans une ligne isolée.
- Syntaxe:
  - **puts( <Chaîne> )**
- Effet:
  - puts écrit la chaîne de caractères désignée par <Chaîne> sur stdout et provoque un retour à la ligne. En pratique,
  - **puts(TXT);** est équivalent à **printf("%s\n",TXT);**

73

## Affichage de chaînes de caractères

18

- Exemples

74

## Affichage de chaînes de caractères

18

- Exemples
  - char TEXTE[] = "Voici une première ligne.";
  - puts(TEXTE);
  - puts("Voici une deuxième ligne.");

75

## Récupération d'une chaîne via un scanf

19

- le symbole **%s.**

76

## Récupération d'une chaîne via un scanf

19

- le symbole **%s**.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.

77

## Récupération d'une chaîne via un scanf

19

- le symbole **%s**.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- Si on lui demande son prénom, il s'appelle peut-être Luc (3 caractères),

78

## Récupération d'une chaîne via un scanf

19

- le symbole **%s**.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- Si on lui demande son prénom, il s'appelle peut-être Luc (3 caractères),
- S'il s'appelle Jean-Edouard (beaucoup plus de caractères)

79

## Récupération d'une chaîne via un scanf

19

- le symbole **%s**.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- Si on lui demande son prénom, il s'appelle peut-être Luc (3 caractères),
- S'il s'appelle Jean-Edouard (beaucoup plus de caractères)
  - il n'y a pas 36 solutions.

80

## Récupération d'une chaîne via un scanf

19

- le symbole **%s**.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- Si on lui demande son prénom, il s'appelle peut-être Luc (3 caractères),
- S'il s'appelle Jean-Edouard (beaucoup plus de caractères)
  - il n'y a pas 36 solutions.
  - Il va falloir créer un tableau de **char** très grand, suffisamment grand pour pouvoir stocker le prénom.

81

## Récupération d'une chaîne via un scanf

19

- le symbole **%s**.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- Si on lui demande son prénom, il s'appelle peut-être Luc (3 caractères),
- S'il s'appelle Jean-Edouard (beaucoup plus de caractères)
  - il n'y a pas 36 solutions.
  - Il va falloir créer un tableau de **char** très grand, suffisamment grand pour pouvoir stocker le prénom.
  - on va donc créer un char[100].

82

## Récupération d'une chaîne via un scanf

19

- le symbole **%s**.
- Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- Si on lui demande son prénom, il s'appelle peut-être Luc (3 caractères),
- S'il s'appelle Jean-Edouard (beaucoup plus de caractères)
  - il n'y a pas 36 solutions.
  - Il va falloir créer un tab… stocker le prénom.
  - on va donc créer un cha…

```c
int main(int argc, char *argv[])
{
        char prenom[100];
        printf("Comment t'appelles-tu petit Zero ? ");
        scanf("%s", prenom);
        printf("Salut %s, je suis heureux de te rencontrer !", prenom);
        return 0;
}
```

83

## Récupération d'une chaîne via un scanf

20

- l'utilisation de **scanf** pour la lecture de chaînes de caractères est seulement conseillée si on est forcé de lire un nombre fixé de mots en une fois.

84

## Récupération d'une chaîne via un scanf

20

- l'utilisation de **scanf** pour la lecture de chaînes de caractères est seulement conseillée si on est forcé de lire un nombre fixé de mots en une fois.
- **gets** est idéal pour lire une ou plusieurs lignes de texte (p.ex. des phrases) terminées par un retour à la ligne.

85

## Récupération d'une chaîne via un scanf

20

- l'utilisation de **scanf** pour la lecture de chaînes de caractères est seulement conseillée si on est forcé de lire un nombre fixé de mots en une fois.
- **gets** est idéal pour lire une ou plusieurs lignes de texte (p.ex. des phrases) terminées par un retour à la ligne.
- Syntaxe:

86

## Récupération d'une chaîne via un scanf

20

- l'utilisation de **scanf** pour la lecture de chaînes de caractères est seulement conseillée si on est forcé de lire un nombre fixé de mots en une fois.
- **gets** est idéal pour lire une ou plusieurs lignes de texte (p.ex. des phrases) terminées par un retour à la ligne.
- Syntaxe:
  - gets( <Chaîne> )

87

## Récupération d'une chaîne via un scanf

20

- l'utilisation de **scanf** pour la lecture de chaînes de caractères est seulement conseillée si on est forcé de lire un nombre fixé de mots en une fois.
- **gets** est idéal pour lire une ou plusieurs lignes de texte (p.ex. des phrases) terminées par un retour à la ligne.
- Syntaxe:
  - gets( <Chaîne> )
- Effet:

88

## Récupération d'une chaîne via un scanf

20

- l'utilisation de **scanf** pour la lecture de chaînes de caractères est seulement conseillée si on est forcé de lire un nombre fixé de mots en une fois.
- **gets** est idéal pour lire une ou plusieurs lignes de texte (p.ex. des phrases) terminées par un retour à la ligne.
- Syntaxe:
  - gets( <Chaîne> )
- Effet:
  - **gets** lit une ligne de de caractères de stdin et la copie à l'adresse indiquée par <Chaîne>. Le retour à la ligne final est remplacé par le symbole de fin de chaîne '\0'.

89

## Récupération d'une chaîne via un scanf

20

- l'utilisation de **scanf** pour la lecture de chaînes de caractères est seulement conseillée si on est forcé de lire un nombre fixé de mots en une fois.
- **gets** est idéal pour lire une ou plusieurs lignes de texte (p.ex. des phrases) terminées par un retour à la ligne.
- Syntaxe:
  - gets( <Chaîne> )
- Effet:
  - **gets** lit une ligne de de caractères de stdin et la copie à l'adresse indiquée par <Chaîne>. Le retour à la ligne final est remplacé par le symbole de fin de chaîne '\0'.

Exemple

90

## Récupération d'une chaîne via un scanf

20

- l'utilisation de **scanf** pour la lecture de chaînes de caractères est seulement conseillée si on est forcé de lire un nombre fixé de mots en une fois.
- **gets** est idéal pour lire une ou plusieurs lignes de texte (p.ex. des phrases) terminées par un retour à la ligne.
- Syntaxe:
  - gets( <Chaîne> )
- Effet:
  - **gets** lit une ligne de de caractères de stdin et la copie à l'adresse indiquée par <Chaîne>. Le retour à la ligne final est remplacé par le symbole de fin de chaîne '\0'.

Exemple

- int MAXI = 1000;
- char LIGNE[MAXI];
- gets(LIGNE);

91

## Fonctions de manipulation des chaînes

21

- la bibliothèque **string.h** fournit une pléthore de fonctions dédiées aux calculs sur des chaînes.

92

## Fonctions de manipulation des chaînes

21

- la bibliothèque **string.h** fournit une pléthore de fonctions dédiées aux calculs sur des chaînes.
- inclure **string.h**

93

## Fonctions de manipulation des chaînes

21

- la bibliothèque **string.h** fournit une pléthore de fonctions dédiées aux calculs sur des chaînes.
- inclure **string.h**

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

94

## strlen : calculer la longueur d'une chaîne

22

- **strlen** est une fonction qui calcule la longueur d'une chaîne de caractères (sans compter le caractère **0** ).

95

## strlen : calculer la longueur d'une chaîne

22

- **strlen** est une fonction qui calcule la longueur d'une chaîne de caractères (sans compter le caractère **0** ).
- on doit lui envoyer un seul paramètre : la chaîne de caractères. Cette fonction retourne la longueur de la chaîne.

96

## strlen : calculer la longueur d'une chaîne

23

- le prototype des fonctions

97

## strlen : calculer la longueur d'une chaîne

23

- le prototype des fonctions

```
size_t strlen(const char* chaine);
```

98

## strlen : calculer la longueur d'une chaîne

23

- le prototype des fonctions

```
size_t strlen(const char* chaine);
```

- **size_t** est un type spécial qui signifie que la fonction renvoie un nombre correspondant à une taille.

99

## strlen : calculer la longueur d'une chaîne

23

- le prototype des fonctions

```
size_t strlen(const char* chaine);
```

- **size_t** est un type spécial qui signifie que la fonction renvoie un nombre correspondant à une taille.
- stocker la valeur renvoyée par strlen dans une variable de type int (l'ordinateur convertira de size_t en int automatiquement).

100

## strlen : calculer la longueur d'une chaîne

23

- le prototype des fonctions

```
size_t strlen(const char* chaine);
```

- **size_t** est un type spécial qui signifie que la fonction renvoie un nombre correspondant à une taille.
- stocker la valeur renvoyée par strlen dans une variable de type int (l'ordinateur convertira de size_t en int automatiquement).

```
int main(int argc, char *argv[]) {
        char chaine[] = "Salut";
        int longueurChaine = 0;
        // On récupère la longueur de la chaîne dans longueurChaine
        longueurChaine = strlen(chaine);
        printf("La chaine %s fait %d caracteres de long", chaine, longueurChaine);
        return 0;
}
```

101

## strlen : calculer la longueur d'une chaîne

24

- écrire une fonction similaire à **strlen**

102

## strlen : calculer la longueur d'une chaîne

```
int longueurChaine(const char* chaine);
int main(int argc, char *argv[]) {
        char chaine[] = "Salut";
        int longueur = 0;
        longueur = longueurChaine(chaine);
        printf("La chaine %s fait %d caracteres de long", chaine, longueur);
        return 0; }
int longueurChaine(const char* chaine) {
        int nombreDeCaracteres = 0;
        char caractereActuel = 0;
        do {
                caractereActuel = chaine[nombreDeCaracteres];
                nombreDeCaracteres++;
        }
        while(caractereActuel != '\0'); // On boucle tant qu'on n'est pas arrivé à l'\0
        nombreDeCaracteres--; // On retire 1 caractère de long pour ne pas compter le caractère \0
        return nombreDeCaracteres;
}
```

103

## strlen : calculer la longueur d'une chaîne

25

104

## strlen : calculer la longueur d'une chaîne

25

- La fonction **longueurChaine** fait une boucle sur le tableau **chaine.**

105

## strlen : calculer la longueur d'une chaîne

25

- La fonction **longueurChaine** fait une boucle sur le tableau **chaine.**
- Elle stocke les caractères un par un dans **caractereActuel.**

106

## strlen : calculer la longueur d'une chaîne

25

- La fonction **longueurChaine** fait une boucle sur le tableau **chaine**.
- Elle stocke les caractères un par un dans **caractereActuel**.
- Dès que **caractèreActuel** vaut **'\0'**, la boucle s'arrête.

107

## strlen : calculer la longueur d'une chaîne

25

- La fonction **longueurChaine** fait une boucle sur le tableau **chaine**.
- Elle stocke les caractères un par un dans **caractereActuel**.
- Dès que **caractèreActuel** vaut **'\0'**, la boucle s'arrête.
- À chaque passage dans la boucle, on ajoute 1 au nombre de caractères qu'on a analysés.

108

## strlen : calculer la longueur d'une chaîne

25

- La fonction **longueurChaine** fait une boucle sur le tableau **chaine**.
- Elle stocke les caractères un par un dans **caractereActuel**.
- Dès que **caractèreActuel** vaut **'\0'**, la boucle s'arrête.
- À chaque passage dans la boucle, on ajoute 1 au nombre de caractères qu'on a analysés.
- À la fin de la boucle, on retire 1 caractère au nombre total de caractères qu'on a comptés.

109

## strlen : calculer la longueur d'une chaîne

25

- La fonction **longueurChaine** fait une boucle sur le tableau **chaine**.
- Elle stocke les caractères un par un dans **caractereActuel**.
- Dès que **caractèreActuel** vaut **'\0'**, la boucle s'arrête.
- À chaque passage dans la boucle, on ajoute 1 au nombre de caractères qu'on a analysés.
- À la fin de la boucle, on retire 1 caractère au nombre total de caractères qu'on a comptés.
- Enfin, on retourne **nombreDeCaracteres**

110

## strcpy : copier une chaîne dans une autre

26

- La fonction **strcpy** (comme « **string copy** ») permet de copier une chaîne à l'intérieur d'une autre.

111

## strcpy : copier une chaîne dans une autre

26

- La fonction **strcpy** (comme « **string copy** ») permet de copier une chaîne à l'intérieur d'une autre.
- Son prototype est :

112

## strcpy : copier une chaîne dans une autre

26

- La fonction **strcpy** (comme « **string copy** ») permet de copier une chaîne à l'intérieur d'une autre.
- Son prototype est :

```
char* strcpy(char* copieDeLaChaine, const char* chaineACopier);
```

113

## strcpy : copier une chaîne dans une autre

26

- La fonction **strcpy** (comme « **string copy** ») permet de copier une chaîne à l'intérieur d'une autre.
- Son prototype est :

```
char* strcpy(char* copieDeLaChaine, const char* chaineACopier);
```

- Cette fonction prend deux paramètres :

114

## strcpy : copier une chaîne dans une autre

26

- La fonction **strcpy** (comme « **string copy** ») permet de copier une chaîne à l'intérieur d'une autre.
- Son prototype est :

```
char* strcpy(char* copieDeLaChaine, const char* chaineACopier);
```

- Cette fonction prend deux paramètres :
  - **copieDeLaChaine** : c'est un pointeur vers un **char*** (tableau de **char**). C'est dans ce tableau que la chaîne sera copiée ;

115

## strcpy : copier une chaîne dans une autre

26

- La fonction **strcpy** (comme « **string copy** ») permet de copier une chaîne à l'intérieur d'une autre.
- Son prototype est :

```
char* strcpy(char* copieDeLaChaine, const char* chaineACopier);
```

- Cette fonction prend deux paramètres :
  - **copieDeLaChaine** : c'est un pointeur vers un **char*** (tableau de **char**). C'est dans ce tableau que la chaîne sera copiée ;
  - **chaineACopier** : c'est un pointeur vers un autre tableau de **char**. Cette chaîne sera copiée dans **copieDeLaChaine**.

116

## strcpy : copier une chaîne dans une autre

27

- La fonction renvoie un pointeur sur **copieDeLaChaine,** ce qui n'est pas très utile.

117

## strcpy : copier une chaîne dans une autre

27

- La fonction renvoie un pointeur sur **copieDeLaChaine,** ce qui n'est pas très utile.
- En général, on ne récupère pas ce que cette fonction renvoie.

118

## strcpy : copier une chaîne dans une autre

27

- La fonction renvoie un pointeur sur **copieDeLaChaine,** ce qui n'est pas très utile.
- En généra

```c
int main(int argc, char *argv[])
{
/* On crée une chaîne "chaine" qui contient un peu de texte
et une copie (vide) de taille 100 pour être sûr d'avoir la place
pour la copie */
char chaine[] = "Texte", copie[100] = {0};
strcpy(copie, chaine); // On copie "chaine" dans "copie"
// Si tout s'est bien passé, la copie devrait être identique à chaine
printf("chaine vaut : %s\n", chaine);
printf("copie vaut : %s\n", copie);
return 0;
}
```

119

## strcpy : copier une chaîne dans une autre

28

- la chaîne **copie** est assez grande pour accueillir le contenu de **chaine.**

120

## strcpy : copier une chaîne dans une autre

28

- la chaîne **copie** est assez grande pour accueillir le contenu de **chaine.**
- Si on définie **copie[5]**

121

## strcpy : copier une chaîne dans une autre

28

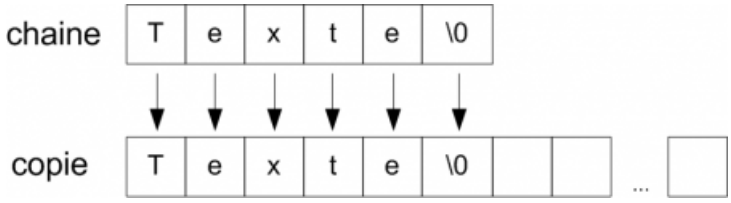
- la chaîne **copie** est assez grande pour accueillir le contenu de **chaine.**
- Si on définie **copie[5]**
- la copie a fonctionné comme sur la fig. suivante :

122

## strcpy : copier une chaîne dans une autre

28

- la chaîne **copie** est assez grande pour accueillir le contenu de **chaine.**
- Si on définie **copie[5]**
- la copie a fonctionné comme sur la fig. suivante :

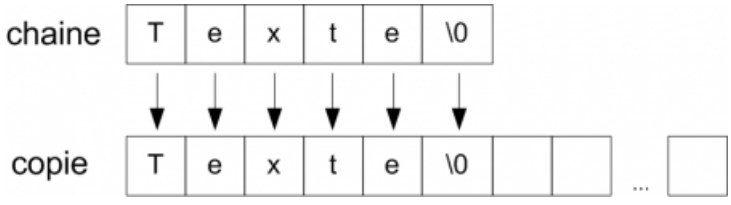



123

## strcpy : copier une chaîne dans une autre

28

- la chaîne **copie** est assez grande pour accueillir le contenu de **chaine.**
- Si on définie **copie[5]**
- la copie a fonctionné comme sur la fig. suivante :



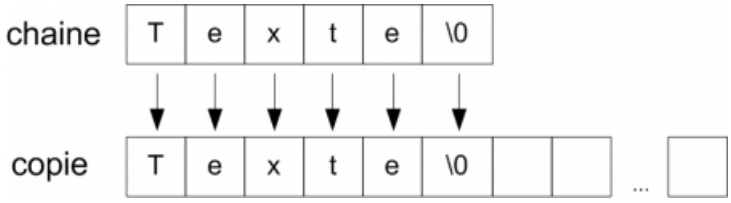

- La chaîne **copie** contient de nombreux caractères inutilisés

124

## strcpy : copier une chaîne dans une autre

28

- la chaîne **copie** est assez grande pour accueillir le contenu de **chaine.**
- Si on définie **copie[5]**
- la copie a fonctionné comme sur la fig. suivante :




- La chaîne **copie** contient de nombreux caractères inutilisés
- L'avantage de créer un tableau un peu plus grand, c'est que de cette façon la chaîne **copie** sera capable de recevoir d'autres chaînes peut-être plus grandes dans la suite du programme.

125

## strcat : concaténer 2 chaînes

29

- Cette fonction ajoute une chaîne à la suite d'une autre.

126

## strcat : concaténer 2 chaînes

29

- Cette fonction ajoute une chaîne à la suite d'une autre.
- Exemple : Deux variables :

127

## strcat : concaténer 2 chaînes

29

- Cette fonction ajoute une chaîne à la suite d'une autre.
- Exemple : Deux variables :

```
chaine1 = "Salut "
chaine2 = "Mateo21"
```

128

## strcat : concaténer 2 chaînes

29

- Cette fonction ajoute une chaîne à la suite d'une autre.
- Exemple : Deux variables :

```
chaine1 = "Salut "
chaine2 = "Mateo21"
```

- Si on concatène **chaine2** dans **chaine1,** alors **chaine1** vaudra "**Salut Mateo21**".

129

## strcat : concaténer 2 chaînes

29

- Cette fonction ajoute une chaîne à la suite d'une autre.
- Exemple : Deux variables :

```
chaine1 = "Salut "
chaine2 = "Mateo21"
```

- Si on concatène **chaine2** dans **chaine1,** alors **chaine1** vaudra "**Salut Mateo21**".
- Quant à **chaine2,** elle n'aura pas changé et vaudra donc toujours "**Mateo21**".

130

## strcat : concaténer 2 chaînes

29

- Cette fonction ajoute une chaîne à la suite d'une autre.
- Exemple : Deux variables :

```
chaine1 = "Salut "
chaine2 = "Mateo21"
```

- Si on concatène **chaine2** dans **chaine1,** alors **chaine1** vaudra "**Salut Mateo21**".
- Quant à **chaine2,** elle n'aura pas changé et vaudra donc toujours "**Mateo21**".
- Seule **chaine1** est modifiée.

131

## strcat : concaténer 2 chaînes

30

- le prototype :

132

## strcat : concaténer 2 chaînes

30

- le prototype : `char* strcat(char* chaine1, const char* chaine2);`

133

## strcat : concaténer 2 chaînes

30

- le prototype : `char* strcat(char* chaine1, const char* chaine2);`
- **chaine2** ne peut pas être modifiée car elle est définie comme constante dans le prototype de la fonction.

134

## strcat : concaténer 2 chaînes

30

- le prototype : `char* strcat(char* chaine1, const char* chaine2);`
- **chaine2** ne peut pas être modifiée car elle est définie comme constante dans le prototype de la fonction.
- La fonction retourne un pointeur vers **chaine1**

135

## strcat : concaténer 2 chaînes

30

- le prototype : `char* strcat(char* chaine1, const char* chaine2);`
- **chaine2** ne peut pas être modifiée car elle est définie comme constante dans le prototype de la fonction.
- La fonction retourne un pointeur vers **chaine1**

```
int main(int argc, char *argv[])
{ /* On crée 2 chaînes. chaine1 doit être assez grande pour accueillir
        le contenu de chaine2 en plus, sinon risque de plantage */
        char chaine1[100] = "Salut ", chaine2[] = "Mateo21";
        strcat(chaine1, chaine2); // On concatène chaine2 dans chaine1
        // Si tout s'est bien passé, chaine1 vaut "Salut Mateo21"
        printf("chaine1 vaut : %s\n", chaine1);
        // chaine2 n'a pas changé :
        printf("chaine2 vaut toujours : %s\n", chaine2);
        return 0;
}
```

136

## strcat : concaténer 2 chaînes

31

137

## strcat : concaténer 2 chaînes

31

- Vérifiez absolument que **chaine1** est assez grande pour qu'on puisse lui ajouter le contenu de **chaine2,**

138

## strcat : concaténer 2 chaînes

31

- Vérifiez absolument que **chaine1** est assez grande pour qu'on puisse lui ajouter le contenu de **chaine2,**
- sinon vous ferez un débordement en mémoire qui peut conduire à un plantage.

139

## strcat : concaténer 2 chaînes

31

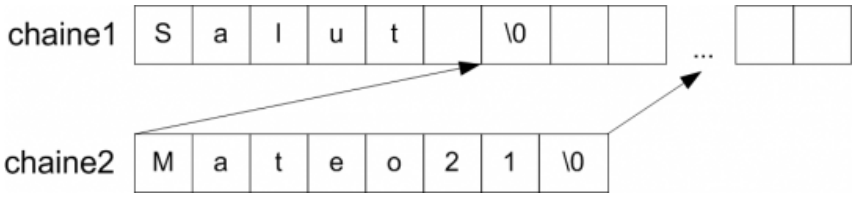
- Vérifiez absolument que **chaine1** est assez grande pour qu'on puisse lui ajouter le contenu de **chaine2,**
- sinon vous ferez un débordement en mémoire qui peut conduire à un plantage.
- le fonctionnement de la concaténation.

140

## strcat : concaténer 2 chaînes

31

- Vérifiez absolument que **chaine1** est assez grande pour qu'on puisse lui ajouter le contenu de **chaine2,**
- sinon vous ferez un débordement en mémoire qui peut conduire à un plantage.
- le fonctionnement de la concaténation.




141

## strcmp : comparer 2 chaînes

32

- **strcmp** compare 2 chaînes entre elles.
- le prototype :

142

## strcmp : comparer 2 chaînes

32

- **strcmp** compare 2 chaînes entre elles.
- le prototype :

```
int strcmp(const char* chaine1, const char* chaine2);
```

143

## strcmp : comparer 2 chaînes

32

- **strcmp** compare 2 chaînes entre elles.
- le prototype :

```
int strcmp(const char* chaine1, const char* chaine2);
```

- les variables **chaine1** et **chaine2** sont comparées.

144

## strcmp : comparer 2 chaînes

32

- **strcmp** compare 2 chaînes entre elles.
- le prototype : `int strcmp(const char* chaine1, const char* chaine2);`
- les variables **chaine1** et **chaine2** sont comparées.
- aucune d'elles n'est modifiée car elles sont indiquées comme constantes.

145

## strcmp : comparer 2 chaînes

32

- **strcmp** compare 2 chaînes entre elles.
- le prototype : `int strcmp(const char* chaine1, const char* chaine2);`
- les variables **chaine1** et **chaine2** sont comparées.
- aucune d'elles n'est modifiée car elles sont indiquées comme constantes.
- **strcmp** renvoie :

146

## strcmp : comparer 2 chaînes

32

- **strcmp** compare 2 chaînes entre elles.
- le prototype : `int strcmp(const char* chaine1, const char* chaine2);`
- les variables **chaine1** et **chaine2** sont comparées.
- aucune d'elles n'est modifiée car elles sont indiquées comme constantes.
- **strcmp** renvoie :
  - 0 si les chaînes sont identiques ;

147

## strcmp : comparer 2 chaînes

32

- **strcmp** compare 2 chaînes entre elles.
- le prototype : `int strcmp(const char* chaine1, const char* chaine2);`
- les variables **chaine1** et **chaine2** sont comparées.
- aucune d'elles n'est modifiée car elles sont indiquées comme constantes.
- **strcmp** renvoie :
  - 0 si les chaînes sont identiques ;
  - une autre valeur (positive ou négative) si les chaînes sont différentes.

148

## strcmp : comparer 2 chaînes

33

- un code de test :

149

## strcmp : comparer 2 chaînes

33

- un code de test :

```c
int main(int argc, char *argv[])
{
	char chaine1[] = "Texte de test", chaine2[] = "Texte de test";
	if (strcmp(chaine1, chaine2) == 0) // Si chaînes identiques
	{
		printf("Les chaines sont identiques\n");
	}
	else
	{
		printf("Les chaines sont differentes\n");
	}
	return 0;
}
```

150

## strchr : rechercher un caractère

34

- La fonction **strchr** recherche un caractère dans une chaîne.
- **Prototype :**

151

## strchr : rechercher un caractère

34

- La fonction **strchr** recherche un caractère dans une chaîne.
- **Prototype :**

```
char* strchr(const char* chaine, int caractereARechercher);
```

152

## strchr : rechercher un caractère

34

- La fonction **strchr** recherche un caractère dans une chaîne.
- **Prototype** :

```
char* strchr(const char* chaine, int caractereARechercher);
```

- La fonction prend 2 paramètres :

153

## strchr : rechercher un caractère

34

- La fonction **strchr** recherche un caractère dans une chaîne.
- **Prototype** :

```
char* strchr(const char* chaine, int caractereARechercher);
```

- La fonction prend 2 paramètres :
  - **chaine** : la chaîne dans laquelle la recherche doit être faite ;
  - **caractereARechercher** : le caractère que l'on doit rechercher dans la chaîne.

154

## strchr : rechercher un caractère

34

- La fonction **strchr** recherche un caractère dans une chaîne.
- **Prototype** :

```
char* strchr(const char* chaine, int caractereARechercher);
```

- La fonction prend 2 paramètres :
  - **chaine** : la chaîne dans laquelle la recherche doit être faite ;
  - **caractereARechercher** : le caractère que l'on doit rechercher dans la chaîne.
- **caractereARechercher** est de type **int** et non de type **char**

155

## strchr : rechercher un caractère

34

- La fonction **strchr** recherche un caractère dans une chaîne.
- **Prototype** :

```
char* strchr(const char* chaine, int caractereARechercher);
```

- La fonction prend 2 paramètres :
  - **chaine** : la chaîne dans laquelle la recherche doit être faite ;
  - **caractereARechercher** : le caractère que l'on doit rechercher dans la chaîne.
- **caractereARechercher** est de type **int** et non de type **char**
  - un caractère est et restera toujours un nombre.

156

## strchr : rechercher un caractère

34

- La fonction **strchr** recherche un caractère dans une chaîne.
- **Prototype** :

```
char* strchr(const char* chaine, int caractereARechercher);
```

- La fonction prend 2 paramètres :
  - **chaine** : la chaîne dans laquelle la recherche doit être faite ;
  - **caractereARechercher** : le caractère que l'on doit rechercher dans la chaîne.
- **caractereARechercher** est de type **int** et non de type **char**
  - un caractère est et restera toujours un nombre.
- La fonction renvoie un pointeur vers le premier caractère qu'elle a trouvé,

157

## strchr : rechercher un caractère

34

- La fonction **strchr** recherche un caractère dans une chaîne.
- **Prototype** :

```
char* strchr(const char* chaine, int caractereARechercher);
```

- La fonction prend 2 paramètres :
  - **chaine** : la chaîne dans laquelle la recherche doit être faite ;
  - **caractereARechercher** : le caractère que l'on doit rechercher dans la chaîne.
- **caractereARechercher** est de type **int** et non de type **char**
  - un caractère est et restera toujours un nombre.
- La fonction renvoie un pointeur vers le premier caractère qu'elle a trouvé,
  - elle renvoie l'adresse de ce caractère dans la mémoire.
  - elle renvoie **NULL** si elle n'a rien trouvé.

158

## strchr : rechercher un caractère

35

- **Exemple** : on récupère ce pointeur dans **suiteChaine** :

159

## strchr : rechercher un caractère

35

- **Exemple** : on récupère ce pointeur dans **suiteChaine** :

```
int main(int argc, char *argv[]) {
        char chaine[] = "Texte de test", *suiteChaine = NULL;
        suiteChaine = strchr(chaine, 'd');
        if (suiteChaine != NULL) // Si on a trouvé quelque chose
        {
        printf("Voici la fin de la chaine a partir du premier d : %s", suiteChaine);
        }
        return 0;
}
```

160

## strchr : rechercher un caractère

35

- **Exemple** : on récupère ce pointeur dans **suiteChaine** :

```
int main(int argc, char *argv[]) {
        char chaine[] = "Texte de test", *suiteChaine = NULL;
        suiteChaine = strchr(chaine, 'd');
        if (suiteChaine != NULL) // Si on a trouvé quelque chose
        {
        printf("Voici la fin de la chaine a partir du premier d : %s", suiteChaine);
        }
        return 0;
}
```

- **suiteChaine** est un pointeur comme **chaine,**

161

## strchr : rechercher un caractère

35

- **Exemple** : on récupère ce pointeur dans **suiteChaine** :

```
int main(int argc, char *argv[]) {
        char chaine[] = "Texte de test", *suiteChaine = NULL;
        suiteChaine = strchr(chaine, 'd');
        if (suiteChaine != NULL) // Si on a trouvé quelque chose
        {
        printf("Voici la fin de la chaine a partir du premier d : %s", suiteChaine);
        }
        return 0;
}
```

- **suiteChaine** est un pointeur comme **chaine,**
  - **chaine** pointe sur le premier caractère (le **'T'** majuscule),

162

## strchr : rechercher un caractère

35

- **Exemple** : on récupère ce pointeur dans **suiteChaine** :

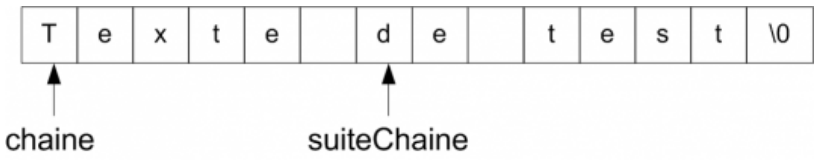
```
int main(int argc, char *argv[]) {
        char chaine[] = "Texte de test", *suiteChaine = NULL;
        suiteChaine = strchr(chaine, 'd');
        if (suiteChaine != NULL) // Si on a trouvé quelque chose
        {
        printf("Voici la fin de la chaine a partir du premier d : %s", suiteChaine);
        }
        return 0;
}
```

- **suiteChaine** est un pointeur comme **chaine,**
  - **chaine** pointe sur le premier caractère (le **'T'** majuscule),
  - **suiteChaine** pointe sur le premier caractère **'d'** qui a été trouvé dans chaine.

163

## strchr : rechercher un caractère

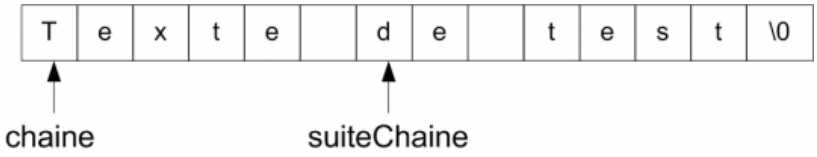
36




164

## strchr : rechercher un caractère

36

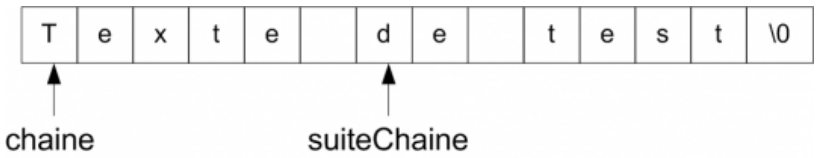



- Il existe une fonction **strrchr** strictement identique à **strchr,**

165

## strchr : rechercher un caractère

36




- Il existe une fonction **strrchr** strictement identique à **strchr,**
- renvoie un pointeur vers le dernier caractère qu'elle a trouvé dans la chaîne plutôt que vers le premier.

166

## strpbrk : premier caractère de la liste

37

- recherche un des caractères dans la liste que vous lui donnez sous forme de chaîne,

167

## strpbrk : premier caractère de la liste

37

- recherche un des caractères dans la liste que vous lui donnez sous forme de chaîne,
- contrairement à **strchr** qui ne peut rechercher qu'un seul caractère à la fois.

168

## strpbrk : premier caractère de la liste

37

- recherche un des caractères dans la liste que vous lui donnez sous forme de chaîne,
- contrairement à **strchr** qui ne peut rechercher qu'un seul caractère à la fois.
- **Exemple** :

169

## strpbrk : premier caractère de la liste

37

- recherche un des caractères dans la liste que vous lui donnez sous forme de chaîne,
- contrairement à **strchr** qui ne peut rechercher qu'un seul caractère à la fois.
- **Exemple** :
  - si on forme la chaîne "**xds**" et qu'on en fait une recherche dans "Texte de test",
  - la fonction renvoie un pointeur vers le premier de ces caractères qu'elle y a trouvé.
  - le premier caractère de "**xds**" qu'elle trouve dans "Texte de test" est le **x**,
  - **strpbrk** renverra un pointeur sur '**x**'.

170

## strpbrk : premier caractère de la liste

37

- recherche un des caractères dans la liste que vous lui donnez sous forme de chaîne,
- contrairement à **strchr** qui ne peut rechercher qu'un seul caractère à la fois.
- **Exemple** :
  - si on forme la chaîne "**xds**" et qu'on en fait une recherche dans "Texte de test",
  - la fonction renvoie un pointeur vers le premier de ces caractères qu'elle y a trouvé.
  - le premier caractère de "**xds**" qu'elle trouve dans "Texte de test" est le **x**,
  - **strpbrk** renverra un pointeur sur '**x**'.
  - **Prototype** :

171

## strpbrk : premier caractère de la liste

37

- recherche un des caractères dans la liste que vous lui donnez sous forme de chaîne,
- contrairement à **strchr** qui ne peut rechercher qu'un seul caractère à la fois.
- **Exemple** :
  - si on forme la chaîne "**xds**" et qu'on en fait une recherche dans "Texte de test",
  - la fonction renvoie un pointeur vers le premier de ces caractères qu'elle y a trouvé.
  - le premier caractère de "**xds**" qu'elle trouve dans "Texte de test" est le **x**,
  - **strpbrk** renverra un pointeur sur '**x**'.
  - **Prototype** :

```
char* strpbrk(const char* chaine, const char* lettresARechercher);
```

172

## strpbrk : premier caractère de la liste

38

- Testez la fonction :

173

## strpbrk : premier caractère de la liste

38

- Testez la fonction :

```
int main(int argc, char *argv[]) {
        char *suiteChaine;
        // On cherche la première occurrence de x, d ou s dans "Texte de test"
        suiteChaine = strpbrk("Texte de test", "xds");
        if (suiteChaine != NULL)
        {
        printf("Voici la fin de la chaine a partir du premier des caracteres trouves : %s",
suiteChaine);
        }
return 0;
}
```

174

## strpbrk : premier caractère de la liste

39

- Il faut simplement retenir la règle suivante :

175

## strpbrk : premier caractère de la liste

39

- Il faut simplement retenir la règle suivante :
  - si vous utilisez les guillemets "", cela signifie chaîne ;

176

## strpbrk : premier caractère de la liste

39

- Il faut simplement retenir la règle suivante :
  - si vous utilisez les guillemets "", cela signifie chaîne ;
  - si vous utilisez les apostrophes '', cela signifie caractère.

177

## strstr : rechercher une chaîne dans une autre

40

- Cette fonction recherche la première occurrence d'une chaîne dans une autre chaîne.

178

## strstr : rechercher une chaîne dans une autre

40

- Cette fonction recherche la première occurrence d'une chaîne dans une autre chaîne.
- Le prototype :

179

## strstr : rechercher une chaîne dans une autre

40

- Cette fonction recherche la première occurrence d'une chaîne dans une autre chaîne.
- Le prototype :

```
char* strstr(const char* chaine, const char* chaineARechercher);
```

180

## strstr : rechercher une chaîne dans une autre

40

- Cette fonction recherche la première occurrence d'une chaîne dans une autre chaîne.
- Le prototype :

```
char* strstr(const char* chaine, const char* chaineARechercher);
```

- attention à ne pas confondre : **strpbrk** recherche **UN** des caractères, tandis que **strstr** recherche toute la chaîne.

181

## strstr : rechercher une chaîne dans une autre

41

- **Exemple :**

182

## strstr : rechercher une chaîne dans une autre

41

- **Exemple :**

```
int main(int argc, char *argv[]) {
        char *suiteChaine;
        // On cherche la première occurrence de "test" dans "Texte de test" :
        suiteChaine = strstr("Texte de test", "test");
        if (suiteChaine != NULL)
        {
        printf("Premiere occurrence de test dans Texte de test : %s\n", suiteChaine);
        }
        return 0;
}
```

183

## strstr : rechercher une chaîne dans une autre

41

- **Exemple :**

```
int main(int argc, char *argv[]) {
        char *suiteChaine;
        // On cherche la première occurrence de "test" dans "Texte de test" :
        suiteChaine = strstr("Texte de test", "test");
        if (suiteChaine != NULL)
        {
        printf("Premiere occurrence de test dans Texte de test : %s\n", suiteChaine);
        }
        return 0;
}
```

- Elle renvoie un pointeur quand elle a trouvé ce qu'elle cherchait.

184

## strstr : rechercher une chaîne dans une autre

41

- **Exemple :**

```
int main(int argc, char *argv[]) {
        char *suiteChaine;
        // On cherche la première occurrence de "test" dans "Texte de test" :
        suiteChaine = strstr("Texte de test", "test");
        if (suiteChaine != NULL)
        {
        printf("Premiere occurrence de test dans Texte de test : %s\n", suiteChaine);
        }
        return 0;
}
```

- Elle renvoie un pointeur quand elle a trouvé ce qu'elle cherchait.
- Elle renvoie NULL si elle n'a rien trouvé.

185

## sprintf : écrire dans une chaîne

42

- Cette fonction se trouve dans **stdio.h**

186

## sprintf : écrire dans une chaîne

42

- Cette fonction se trouve dans **stdio.h**
  - contrairement aux autres fonctions qui étaient dans **string.h**.

187

## sprintf : écrire dans une chaîne

42

- Cette fonction se trouve dans **stdio.h**
  - contrairement aux autres fonctions qui étaient dans **string.h**.
- au lieu d'écrire à l'écran, **sprintf** écrit dans… une chaîne !

188

## sprintf : écrire dans une chaîne

42

- Cette fonction se trouve dans **stdio.h**
  - contrairement aux autres fonctions qui étaient dans **string.h**.
- au lieu d'écrire à l'écran, **sprintf** écrit dans… une chaîne !
- C'est une fonction très pratique pour mettre en forme une chaîne.

189

## sprintf : écrire dans une chaîne

42

- Cette fonction se trouve dans **stdio.h**
  - contrairement aux autres fonctions qui étaient dans **string.h**.
- au lieu d'écrire à l'écran, **sprintf** écrit dans… une chaîne !
- C'est une fonction très pratique pour mettre en forme une chaîne.
- **Exemple** :

190

## sprintf : écrire dans une chaîne

42

- Cette fonction se trouve dans **stdio.h**
  - contrairement aux autres fonctions qui étaient dans **string.h**.
- au lieu d'écrire à l'écran, **sprintf** écrit dans… une chaîne !
- C'est une fonction très p
- **Exemple :**

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
        char chaine[100];
        int age = 15;
        // On écrit "Tu as 15 ans" dans chaine
        sprintf(chaine, "Tu as %d ans !", age);
        // On affiche chaine pour vérifier qu'elle contient bien cela :
        printf("%s", chaine);
        return 0;
}
```

191

## En résumé

43

192

## En résumé

43

- Un ordinateur ne sait pas manipuler du texte, il ne connaît que les nombres. Pour régler le problème, on associe à chaque lettre de l'alphabet un nombre correspondant dans une table appelée la **table ASCII**.

193

## En résumé

43

- Un ordinateur ne sait pas manipuler du texte, il ne connaît que les nombres. Pour régler le problème, on associe à chaque lettre de l'alphabet un nombre correspondant dans une table appelée la **table ASCII**.
- Le type **char** est utilisé pour stocker une et une seule lettre. Il stocke en réalité un nombre mais ce nombre est automatiquement traduit par l'ordinateur à l'affichage.

194

## En résumé

43

- Un ordinateur ne sait pas manipuler du texte, il ne connaît que les nombres. Pour régler le problème, on associe à chaque lettre de l'alphabet un nombre correspondant dans une table appelée la **table ASCII**.
- Le type **char** est utilisé pour stocker une et une seule lettre. Il stocke en réalité un nombre mais ce nombre est automatiquement traduit par l'ordinateur à l'affichage.
- Pour créer un mot ou une phrase, on doit construire une **chaîne de caractères**. Pour cela, on utilise un **tableau de char**.

195

## En résumé

43

- Un ordinateur ne sait pas manipuler du texte, il ne connaît que les nombres. Pour régler le problème, on associe à chaque lettre de l'alphabet un nombre correspondant dans une table appelée la **table ASCII**.
- Le type **char** est utilisé pour stocker une et une seule lettre. Il stocke en réalité un nombre mais ce nombre est automatiquement traduit par l'ordinateur à l'affichage.
- Pour créer un mot ou une phrase, on doit construire une **chaîne de caractères**. Pour cela, on utilise un **tableau de char**.
- Toute chaîne de caractère se termine par un caractère spécial appelé **\0** qui signifie « fin de chaîne ».

196

## En résumé

43

- Un ordinateur ne sait pas manipuler du texte, il ne connaît que les nombres. Pour régler le problème, on associe à chaque lettre de l'alphabet un nombre correspondant dans une table appelée la **table ASCII**.
- Le type **char** est utilisé pour stocker une et une seule lettre. Il stocke en réalité un nombre mais ce nombre est automatiquement traduit par l'ordinateur à l'affichage.
- Pour créer un mot ou une phrase, on doit construire une **chaîne de caractères**. Pour cela, on utilise un **tableau de char**.
- Toute chaîne de caractère se termine par un caractère spécial appelé **\0** qui signifie « fin de chaîne ».
- Il existe de nombreuses fonctions toutes prêtes de manipulation des chaînes dans la **bibliothèque string**. Il faut inclure **string.h** pour pouvoir les utiliser.

197