

2023/2024

Université Abdelmalek Essaâdi  
École Normale Supérieure  
Tétouan



Filière: LE - Math

# Algorithmique et Programmation

BENDAHMANE AHMED

1

## OBJECTIFS

2

- Découvrir et assimiler le vocabulaire de base du domaine
- Acquérir les bases des méthodes de programmation structurée nécessaires à l'apprentissage de tout langage de programmation
- Maîtriser les outils de l'algorithmique (schémas de programme, types et structures de données, modules)
- Acquérir les bases de la programmation structurée en langage C, norme ANSI
- Mettre en œuvre une chaîne de production de programmes (éditeur de texte, compilateur, éditeur de liens)

2

## Plan

3

- Introduction au langage C
- Structure des programmes
- Instruction de contrôle
- Opérateurs et expressions
- Variables et Constantes
- Types de données
- Tableaux et chaînes de caractères
- Pointeurs
- Les structures
- Fichiers et entrées/sorties
- La compilation et préprocesseur

3

Université Abdelmalek Essaâdi  
École Normale Supérieure  
Tétouan



2023/2024

# Introduction au langage C

4

## Programmer ?

5

- Programmer en langage C... ça veut dire quoi ?
- Est-ce que c'est bien pour commencer ?
- Est-ce que vous avez le niveau pour programmer ?
- Est-ce qu'on peut tout faire avec ?

5

## Programmer, c'est quoi ?

6

- Programmer signifie réaliser des « programmes informatiques ». Les programmes demandent à l'ordinateur d'effectuer des actions.
- Votre ordinateur est rempli de programmes en tous genres :
  - la calculatrice est un programme ;
  - votre traitement de texte est un programme ;
  - votre logiciel de « chat » est un programme ;
  - les jeux vidéo sont des programmes.

6

## Programmer, dans quel langage ?

7

- l'ordinateur ne reconnaît que le langage machine.
  - ▣ il apparaît évident qu'écrire des lignes et des lignes de 0 et de 1 serait très compliqué, voir impossible.
  - ▣ De plus la détection d'erreurs s'avèrent exclue.

7

## Programmer, dans quel langage ?

8



8

## Programmer, dans quel langage ?

8



- La première case est « Le programme est écrit dans un langage simplifié ». Ce fameux « langage simplifié » est appelé en fait **langage de haut niveau**.

9

## Programmer, dans quel langage ?

8



- La première case est « Le programme est écrit dans un langage simplifié ». Ce fameux « langage simplifié » est appelé en fait **langage de haut niveau**.
- Il existe plusieurs niveaux de langages. Plus un langage est haut niveau, plus il est proche de votre vraie langue (comme le français).

10

## Programmer, dans quel langage ?

8



- La première case est « Le programme est écrit dans un langage simplifié ». Ce fameux « langage simplifié » est appelé en fait **langage de haut niveau**.
- Il existe plusieurs niveaux de langages. Plus un langage est haut niveau, plus il est proche de votre vraie langue (comme le français).
  - ▣ Un langage de haut niveau est donc facile à utiliser, mais cela a aussi quelques petits défauts comme nous le verrons plus tard.

11

## Programmer, dans quel langage ?

9

- Il existe de nombreux langages de plus ou moins haut niveau en informatique :

12

## Programmer, dans quel langage ?

9

- Il existe de nombreux langages de plus ou moins haut niveau en informatique :
  - ▣ le C ;
  - ▣ le C++ ;
  - ▣ Java ;
  - ▣ Visual Basic ;
  - ▣ Delphi ;
  - ▣ etc.

13

## Programmer, dans quel langage ?

10

- **le code source**, c'est tout simplement le code de votre programme écrit dans un langage de haut niveau.
- **le compilateur**: le programme de traduction
- **la compilation**: la traduction
  
- il existe un compilateur différent pour chaque langage de haut niveau.
- les langages étant différents, on ne traduit pas le C de la même manière qu'on traduit le Delphi.

14

## Programmer, dans quel langage ?

11

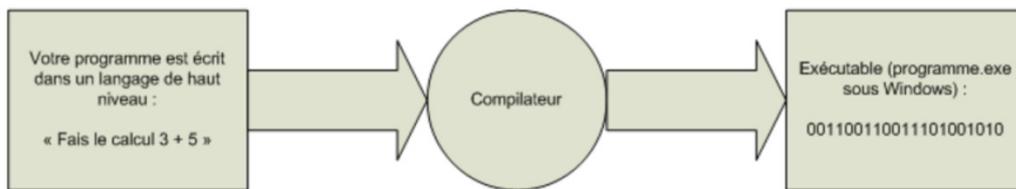
- Pour le langage C il existe plusieurs compilateurs différents ! Il y a le compilateur écrit par Microsoft, le compilateur GNU, etc.
- le programme binaire créé par le compilateur est appelé **l'exécutable**.
- les programmes (tout du moins sous Windows) ont l'extension « .exe » comme EXEcutable.

15

## Programmer, dans quel langage ?

11

- Pour le langage C il existe plusieurs compilateurs différents ! Il y a le compilateur écrit par Microsoft, le compilateur GNU, etc.
- le programme binaire créé par le compilateur est appelé **l'exécutable**.
- les programmes (tout du moins sous Windows) ont l'extension « .exe » comme EXEcutable.



16

## Pourquoi le C ?

12

- il faut bien faire un choix entre :
  - ▣ **un langage très haut niveau :**
    - facile à utiliser
    - Python, Ruby, Visual Basic ...
    - permettent d'écrire des programmes plus rapidement
    - Ils nécessitent toutefois d'être accompagnés de fichiers pour qu'ils puissent s'exécuter (comme un interpréteur) ;
  - ▣ **un langage un peu plus bas niveau :**
    - un peu plus difficiles,
    - mais avec un langage comme le C, vous allez en apprendre beaucoup plus sur la programmation et sur le fonctionnement de l'ordinateur.
    - vous serez ensuite largement capables d'apprendre un autre langage de programmation si vous le désirez.
    - vous serez donc plus autonomes.

17

## Pourquoi le C ?

13

- Il est utilisé par de nombreux développeurs (= programmeurs).
- il est possible de réaliser de nombreuses choses.
- une partie de l'OS que vous utilisez actuellement a été codée en C.
- Il est possible d'écrire ses codes sources directement depuis le bloc-note de Windows,
- pour avoir un résultat direct, il vaut mieux avoir ce que l'on appelle un IDE.  
Voyons cela dans le prochain chapitre.

18

## Historique du langage C

14

- **En 1972**, dans les ‘Bell Laboratories’, Ritchie a conçu le langage C pour développer une version portable du système d’exploitation UNIX.
- **En 1978**, le duo Kernighan/Ritchie a publié la définition classique du langage C,
- **En 1983**, le ‘American National Standards Institute’ (ANSI) chargeait une commission de mettre au point ‘une définition explicite et indépendante de la machine pour le langage C. Le résultat était le standard Ansi-C.

19

Université Abdelmalek Essaâdi  
 École Normale Supérieure  
 Tétouan



2023/2024

les Bons Outils

20

## Les outils nécessaires au programmeur

16

- De quels logiciels a-t-on besoin pour programmer ?

21

## Les outils nécessaires au programmeur

17

- **Un Compilateur** : il transformera nos lignes de codes en programme exécutable
- **Un Editeur de Texte** : il nous servira à écrire nos codes-sources.
- **un débogueur**: pour vous aider à traquer les erreurs dans le programme.

22

## Les outils nécessaires au programmeur

18

Deux possibilités :

- soit on récupère chacun de ces trois programmes séparément. C'est la méthode la plus compliquée, mais elle fonctionne.
- soit on utilise un programme « trois-en-un » (comme les liquides vaisselle, oui, oui) qui combine éditeur de texte, compilateur et débogueur.
  - ▣ Ces programmes « trois-en-un » sont appelés IDE, ou encore « Environnements de développement ».

23

## Choix de l'IDE

19

- Il en existe plusieurs pour Windows : Code::Blocks, Visual C++ Express sont les principaux.

24

## Code::Blocks

20

- Est un IDE libre et gratuit, disponible pour Windows, Mac et Linux.

25

## Code::Blocks

20

- Est un IDE libre et gratuit, disponible pour Windows, Mac et Linux.
- Disponible pour le moment qu'en anglais.

26

## Code::Blocks

20

- Est un IDE libre et gratuit, disponible pour Windows, Mac et Linux.
- Disponible pour le moment qu'en anglais.
- **Télécharger Code::Blocks**

27

## Code::Blocks

20

- Est un IDE libre et gratuit, disponible pour Windows, Mac et Linux.
- Disponible pour le moment qu'en anglais.
- **Télécharger Code::Blocks**

<http://www.codeblocks.org/downloads/binaries>

28

## Code::Blocks

20

- Est un IDE libre et gratuit, disponible pour Windows, Mac et Linux.
- Disponible pour le moment qu'en anglais.
- **Télécharger Code::Blocks**

<http://www.codeblocks.org/downloads/binaries>



File	Date	Size	Download from
codeblocks-10.05-setup.exe	27 May 2010	23.3 MB	BerliOS
codeblocks-10.05mingw-setup.exe	27 May 2010	74.0 MB	BerliOS

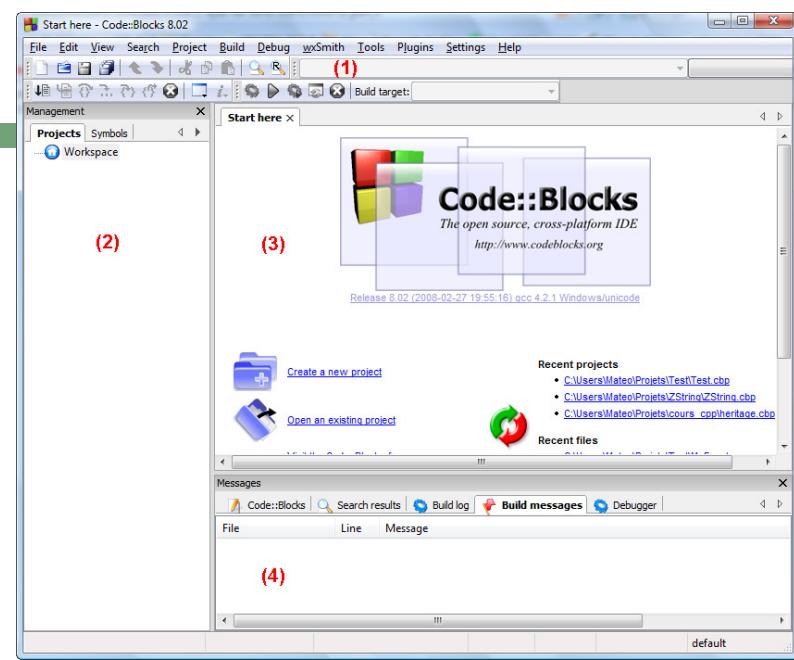


NOTE: The codeblocks-10.05mingw-setup.exe file includes the GCC compiler and GDB debugger from MinGW.

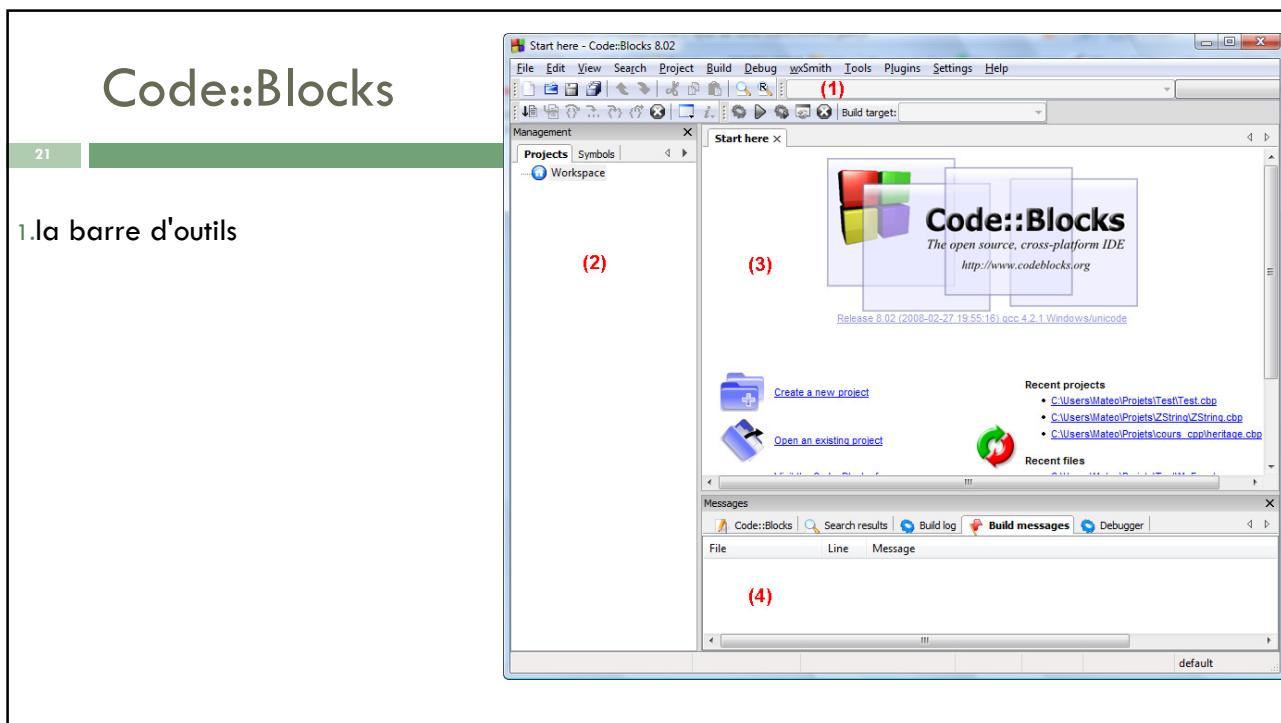
29

## Code::Blocks

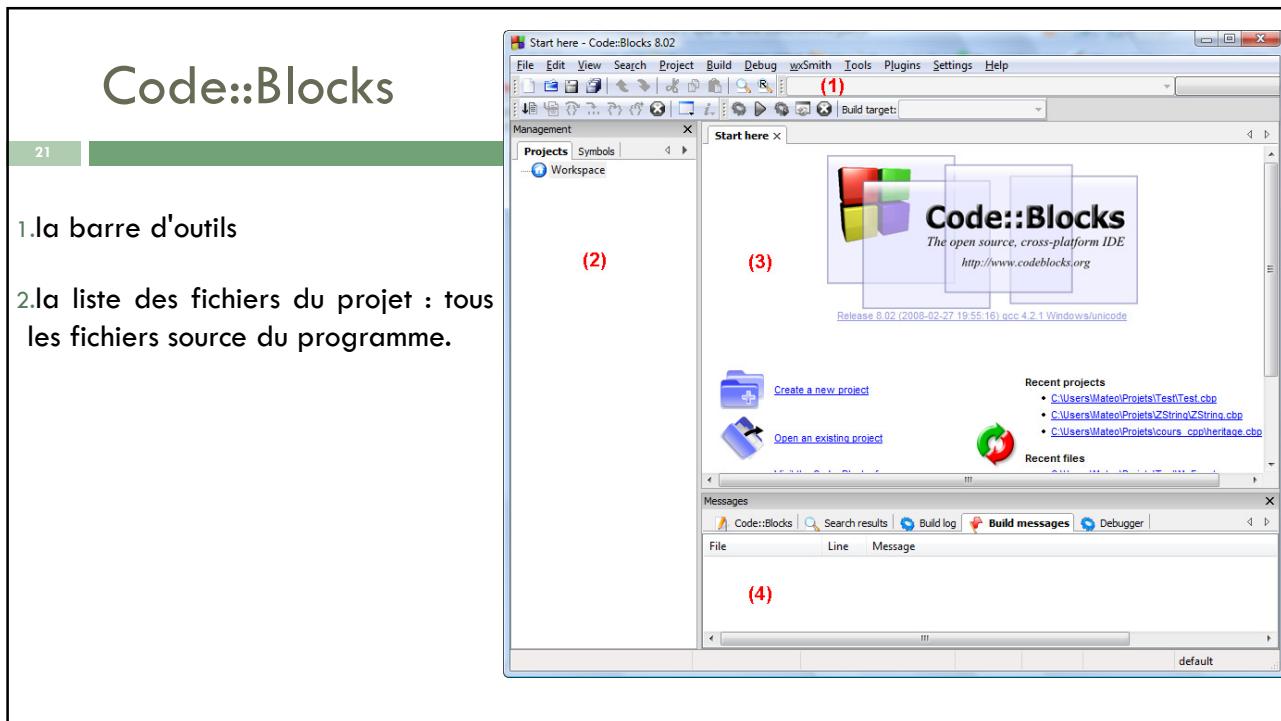
21



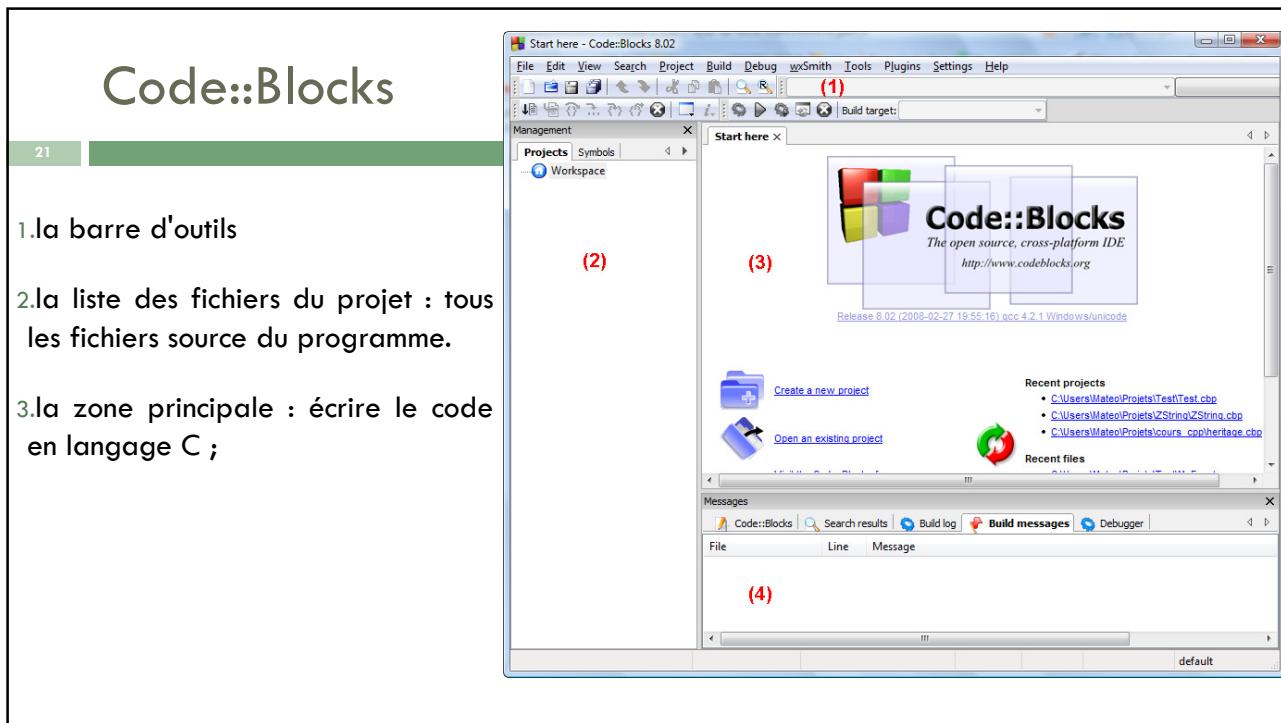
30



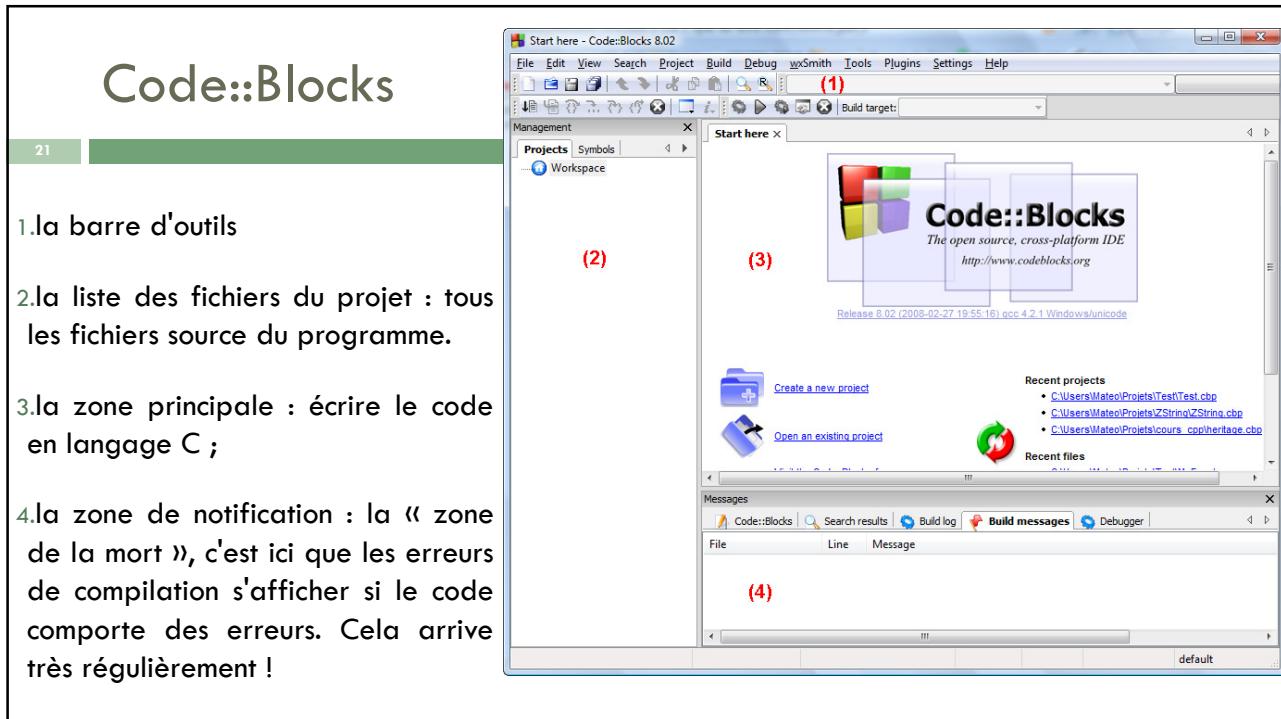
31



32



33



34

## Code::Blocks

22

A une section particulière de la barre d'outils vous trouverez quatre icônes :

35

## Code::Blocks

22

A une section particulière de la barre d'outils vous trouverez quatre icônes :

- compiler** : tous les fichiers source de votre projet sont envoyés au compilateur qui va se charger de créer un exécutable.

36

## Code::Blocks

22

A une section particulière de la barre d'outils vous trouverez quatre icônes :

- **compiler** : tous les fichiers source de votre projet sont envoyés au compilateur qui va se charger de créer un exécutable.
- **exécuter** : cette icône lance juste le dernier exécutable que vous avez compilé.

37

## Code::Blocks

22

A une section particulière de la barre d'outils vous trouverez quatre icônes :

- **compiler** : tous les fichiers source de votre projet sont envoyés au compilateur qui va se charger de créer un exécutable.
- **exécuter** : cette icône lance juste le dernier exécutable que vous avez compilé.
- **compiler & exécuter** : la combinaison des deux boutons précédents. C'est d'ailleurs ce bouton que vous utiliserez le plus souvent.

38

## Code::Blocks

22

A une section particulière de la barre d'outils vous trouverez quatre icônes :

- **compiler** : tous les fichiers source de votre projet sont envoyés au compilateur qui va se charger de créer un exécutable.
- **exécuter** : cette icône lance juste le dernier exécutable que vous avez compilé.
- **compiler & exécuter** : la combinaison des deux boutons précédents. C'est d'ailleurs ce bouton que vous utiliserez le plus souvent.
- **tout reconstruire (recompiler)** : quand vous faites compiler, Code::Blocks ne recompile en fait que les fichiers que vous avez modifiés.

39

## Code::Blocks

22

A une section particulière de la barre d'outils vous trouverez quatre icônes :

- **compiler** : tous les fichiers source de votre projet sont envoyés au compilateur qui va se charger de créer un exécutable.
- **exécuter** : cette icône lance juste le dernier exécutable que vous avez compilé.
- **compiler & exécuter** : la combinaison des deux boutons précédents. C'est d'ailleurs ce bouton que vous utiliserez le plus souvent.
- **tout reconstruire (recompiler)** : quand vous faites compiler, Code::Blocks ne recompile en fait que les fichiers que vous avez modifiés.
- Parfois vous aurez besoin de demander à Code::Blocks de vous recompiler tous les fichiers. On verra plus tard quand on a besoin de ce bouton, et vous verrez plus en détails le fonctionnement de la compilation dans un chapitre futur.

40

## Code::Blocks

22

A une section particulière de la barre d'outils vous trouverez quatre icônes :

- compiler** : tous les fichiers source de votre projet sont envoyés au compilateur qui va se charger de créer un exécutable.
  - exécuter** : cette icône lance juste le dernier exécutable que vous avez compilé. **Je vous conseille d'utiliser les raccourcis plutôt que de cliquer sur les boutons**
  - compiler & exécuter** : tous les fichiers source de votre projet sont envoyés au compilateur qui va se charger de créer un exécutable. C'est d'ailleurs ce bouton que je vous ai recommandé dans les sections précédentes. C'est tout à fait normal que les deux boutons fonctionnent de la même manière.
  - tout reconstruire** : tous les fichiers source de votre projet sont envoyés au compilateur qui va se charger de créer un exécutable. Code::Blocks ne recompile en fait que les fichiers que vous avez modifiés.
- Parfois vous aurez besoin de demander à Code::Blocks de vous recompiler tous les fichiers. On verra plus tard quand on a besoin de ce bouton, et vous verrez plus en détails le fonctionnement de la compilation dans un chapitre futur.

41

## Code::Blocks Crée un nouveau projet

23

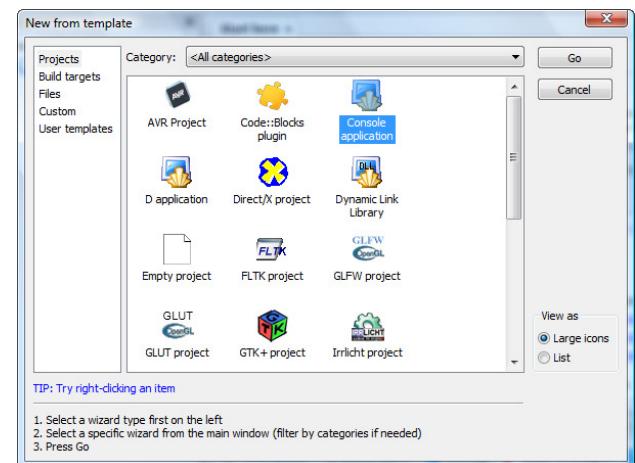
- Le menu File / New / Project

42

## Code::Blocks Crée un nouveau projet

23

- Le menu File / New / Project

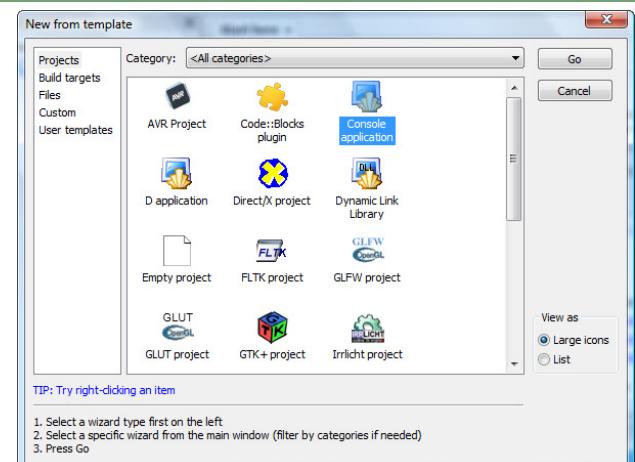


43

## Code::Blocks Crée un nouveau projet

23

- Le menu File / New / Project
- Console application

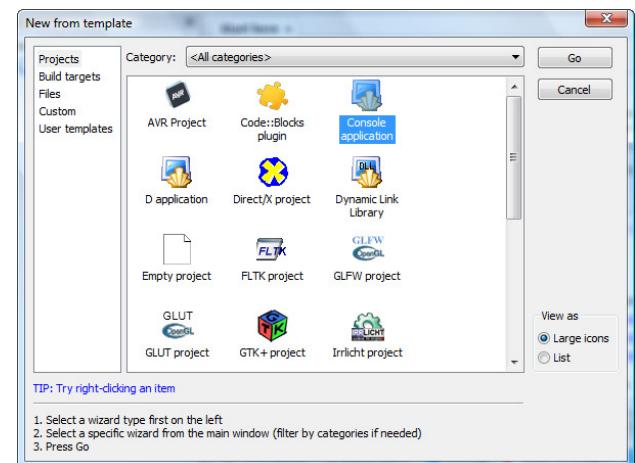


44

## Code::Blocks Crée un nouveau projet

23

- Le menu File / New / Project
- Console application
- Go

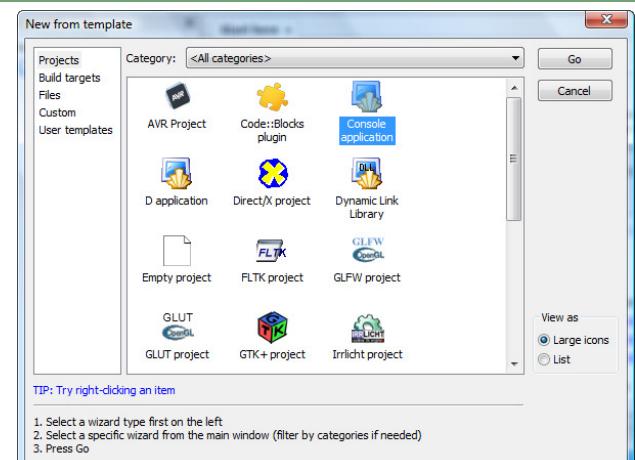


45

## Code::Blocks Crée un nouveau projet

23

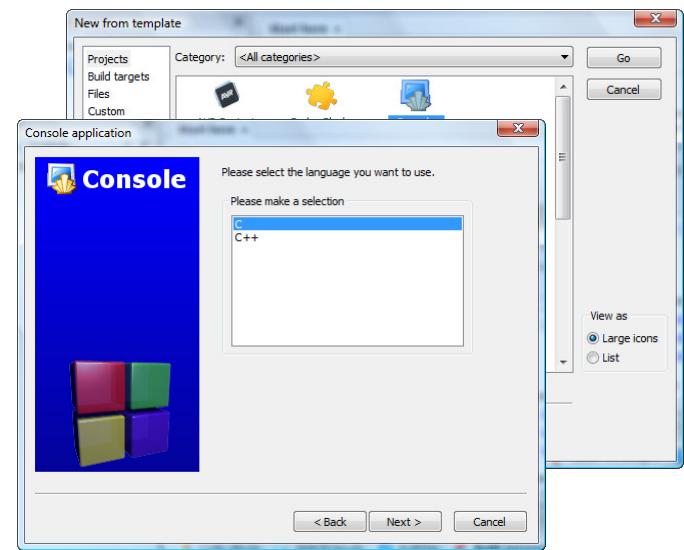
- Le menu File / New / Project
- Console application
- Go
- C



46

## Code::Blocks Crée un nouveau projet

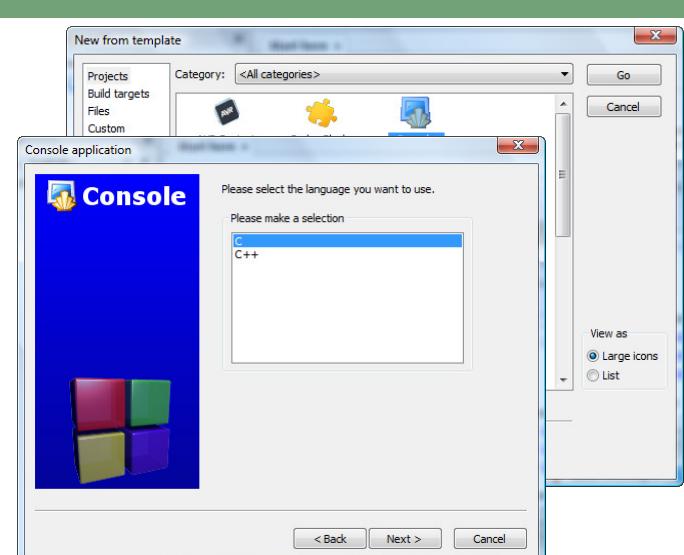
- Le menu File / New / Project
- Console application
- Go
- C



47

## Code::Blocks Crée un nouveau projet

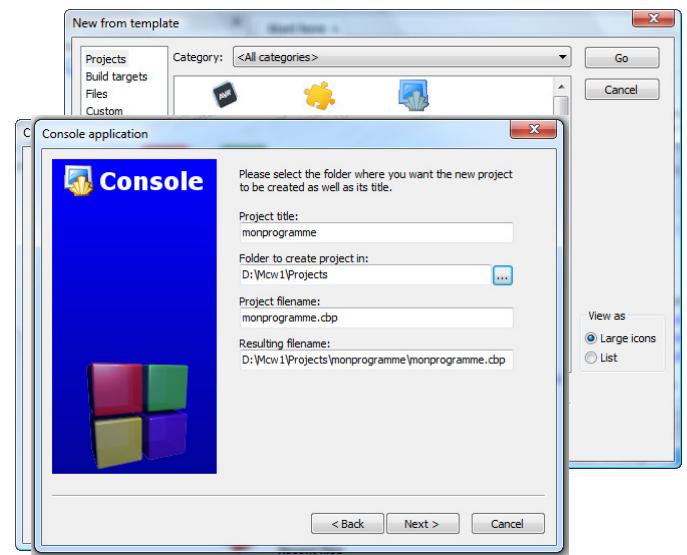
- Le menu File / New / Project
- Console application
- Go
- C
- Next



48

## Code::Blocks Crée un nouveau projet

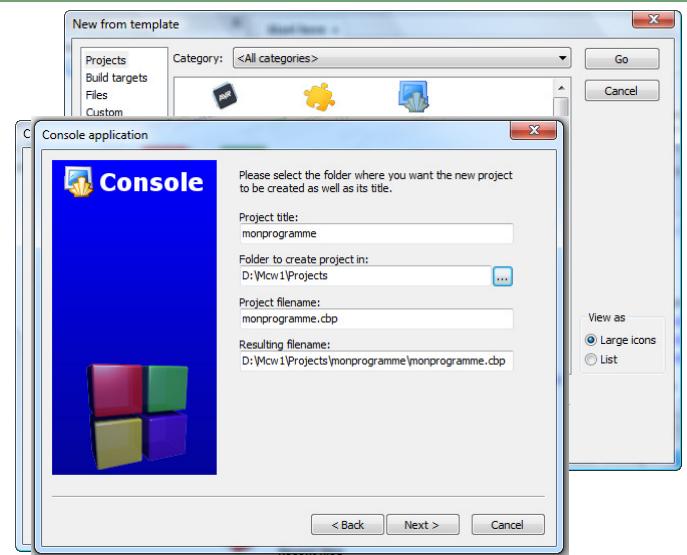
- Le menu File / New / Project
- Console application
- Go
- C
- Next



49

## Code::Blocks Crée un nouveau projet

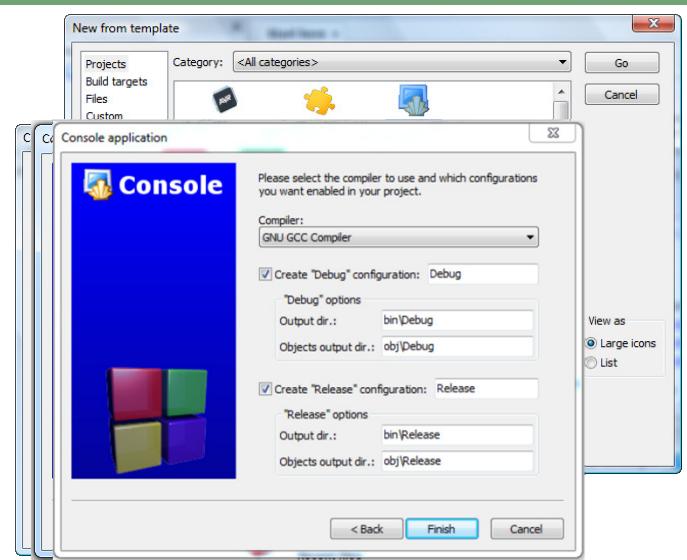
- Le menu File / New / Project
- Console application
- Go
- C
- Next
- le nom du projet et dans quel dossier les fichiers source seront enregistrés.



50

## Code::Blocks Crée un nouveau projet

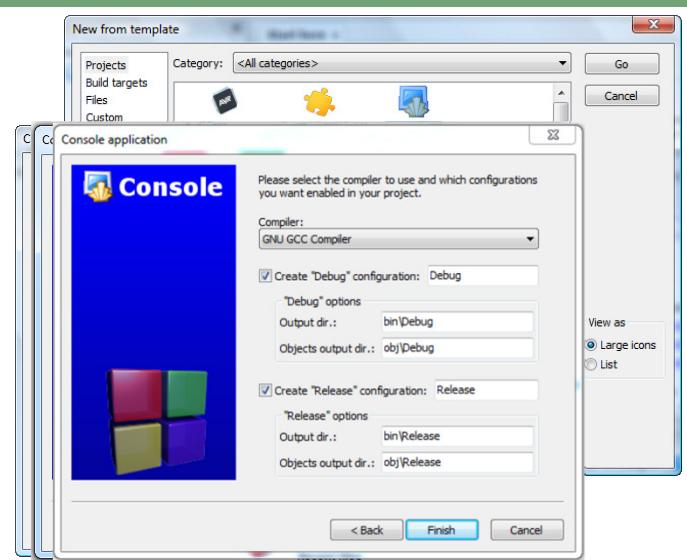
- Le menu File / New / Project
- Console application
- Go
- C
- Next
- le nom du projet et dans quel dossier les fichiers source seront enregistrés.



51

## Code::Blocks Crée un nouveau projet

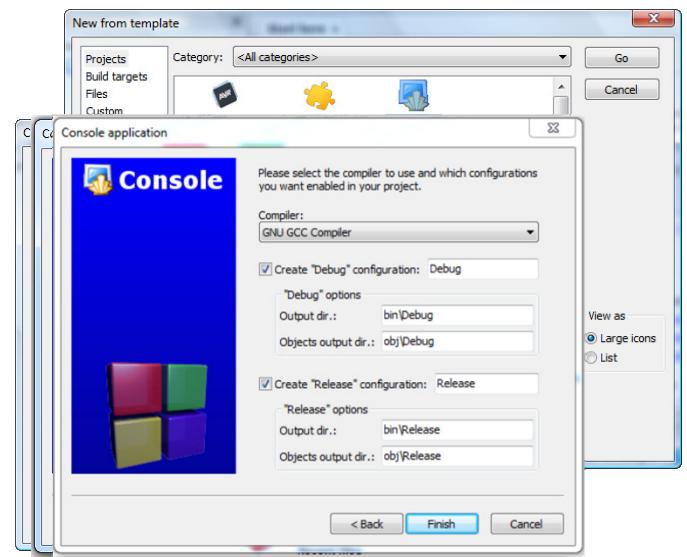
- Le menu File / New / Project
- Console application
- Go
- C
- Next
- le nom du projet et dans quel dossier les fichiers source seront enregistrés.
- la case Debug ou Release au moins soit cochée



52

## Code::Blocks Crée un nouveau projet

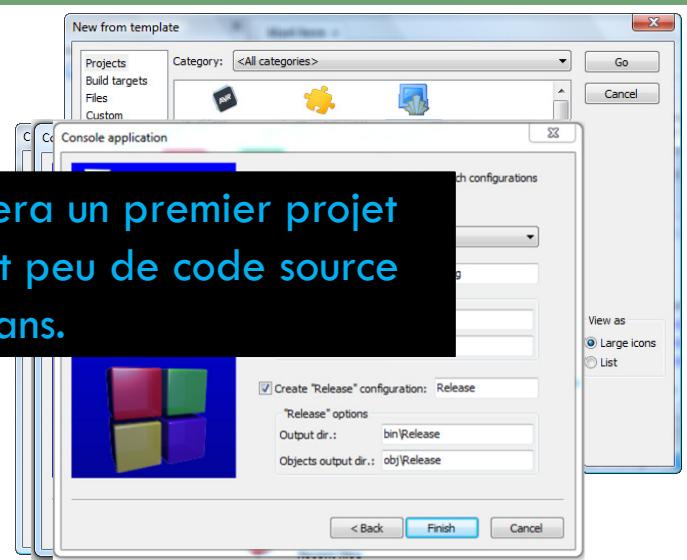
- Le menu File / New / Project
- Console application
- Go
- C
- Next
- le nom du projet et dans quel dossier les fichiers source seront enregistrés.
- la case Debug ou Release au moins soit cochée
- Finish



53

## Code::Blocks Crée un nouveau projet

- Le menu File / New / Project
- Console application
- Go
- C
- Next
- le nom du projet et dans quel dossier les fichiers source seront enregistrés.
- la case Debug ou Release au moins soit cochée
- Finish



54

## Visual C++

24

- c'est l'IDE de Microsoft ;

55

## Visual C++

24

- c'est l'IDE de Microsoft ;
- il est à la base payant, mais Microsoft a sorti une version gratuite intitulée Visual Studio Express ;

56

## Visual C++

24

- c'est l'IDE de Microsoft ;
- il est à la base payant, mais Microsoft a sorti une version gratuite intitulée Visual Studio Express ;
- il permet de programmer en C et en C++.

57

## Visual C++

24

- c'est l'IDE de Microsoft ;
- il est à la base payant, mais Microsoft a sorti une version gratuite intitulée Visual Studio Express ;
- il permet de programmer en C et en C++.
- il n'est compatible avec Windows 7 qu'à partir de la version 2010.

58

## Visual C++

24

- c'est l'IDE de Microsoft ;
- il est à la base payant, mais Microsoft a sorti une version gratuite intitulée Visual Studio Express ;
- il permet de programmer en C et en C++.
- il n'est compatible avec Windows 7 qu'à partir de la version 2010.
- télécharger Visual Studio Express:

59

## Visual C++

24

- c'est l'IDE de Microsoft ;
- il est à la base payant, mais Microsoft a sorti une version gratuite intitulée Visual Studio Express ;
- il permet de programmer en C et en C++.
- il n'est compatible avec Windows 7 qu'à partir de la version 2010.
- télécharger Visual Studio Express:

<https://www.visualstudio.com/vs/visual-studio-express/>

60

## Visual C++

24

- c'est l'IDE de Microsoft ;
- il est à la base payant, mais Microsoft a sorti une version gratuite intitulée Visual Studio Express ;
- il permet de programmer en C et en C++.
- il n'est compatible avec Windows 7 qu'à partir de la version 2010.
- télécharger Visual Studio Express:  
<https://www.visualstudio.com/vs/visual-studio-express/>
- Sélectionnez Téléchargez Community 2015

61

## Visual C++ Installation

25

62

## Visual C++ Installation

25

- Il faut vous enregistrer dans les 30 jours.

63

## Visual C++ Installation

25

- Il faut vous enregistrer dans les 30 jours.
- Connectez-vous avec votre compte Windows Live ID (équivalent du compte Hotmail ou MSN), puis répondez au petit questionnaire.

64

## Visual C++ Installation

25

- Il faut vous enregistrer dans les 30 jours.
- Connectez-vous avec votre compte Windows Live ID (équivalent du compte Hotmail ou MSN), puis répondez au petit questionnaire.
- On vous donnera à la fin une clé d'enregistrement. Vous devrez recopier cette clé dans le menu ? / Incrire le produit.

65

## Visual C++ Créer un nouveau projet

26

66

## Visual C++

### Créer un nouveau projet

26

- Fichier / Nouveau / Projet

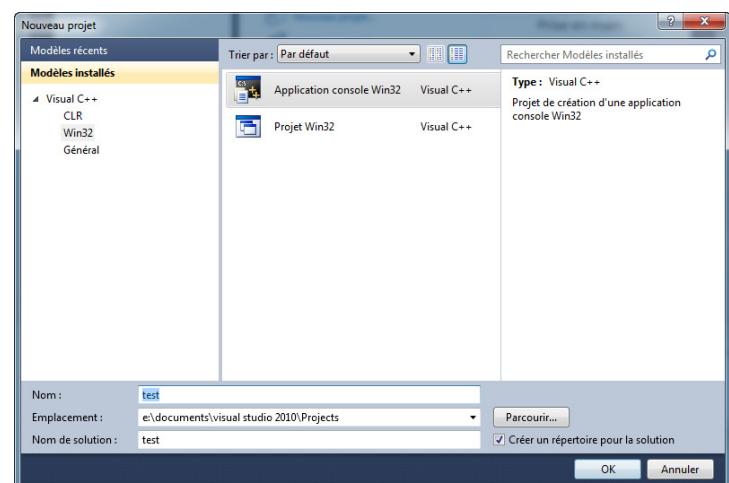
67

## Visual C++

### Créer un nouveau projet

26

- Fichier / Nouveau / Projet



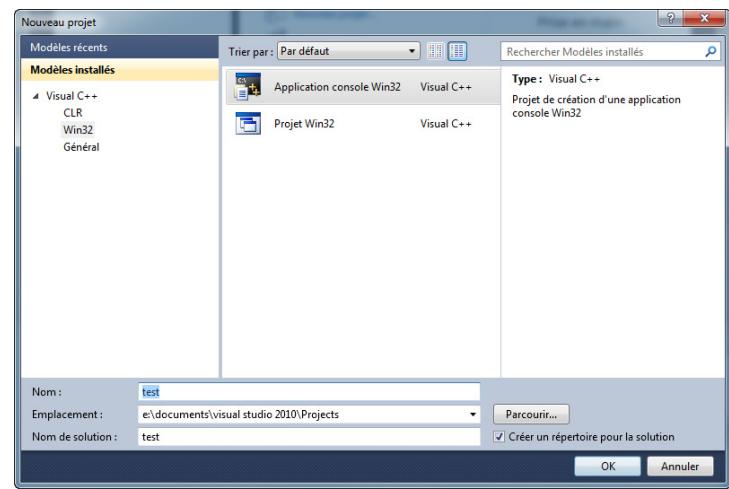
68

## Visual C++

### Créer un nouveau projet

26

- Fichier / Nouveau / Projet
- Win32 dans la colonne de gauche, puis Application console Win32 à droite



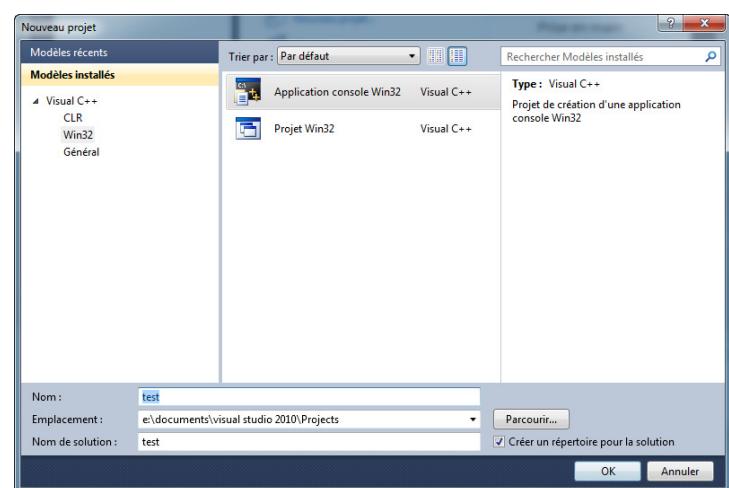
69

## Visual C++

### Créer un nouveau projet

26

- Fichier / Nouveau / Projet
- Win32 dans la colonne de gauche, puis Application console Win32 à droite
- Entrez un nom pour votre projet, par exemple test.



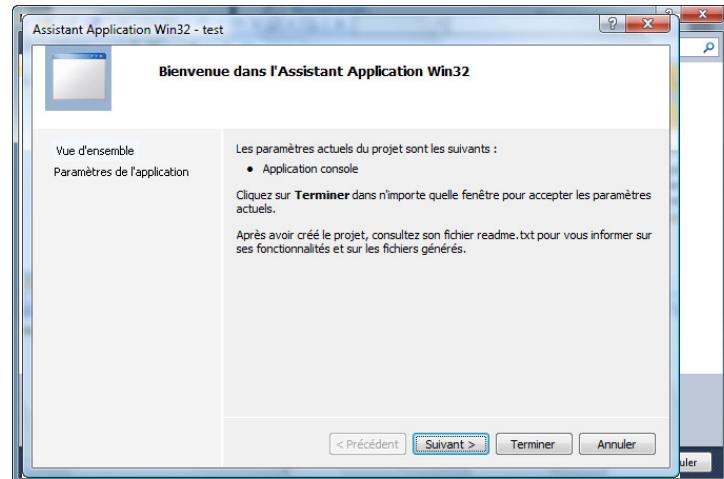
70

## Visual C++

### Créer un nouveau projet

26

- Fichier / Nouveau / Projet
- Win32 dans la colonne de gauche, puis Application console Win32 à droite
- Entrez un nom pour votre projet, par exemple test.



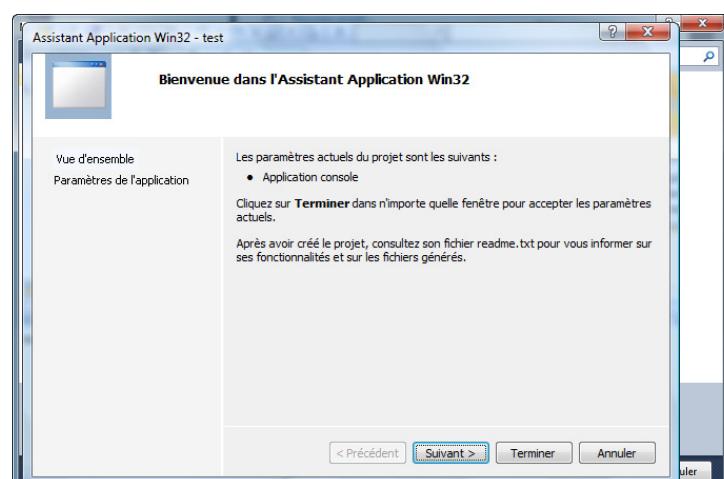
71

## Visual C++

### Créer un nouveau projet

26

- Fichier / Nouveau / Projet
- Win32 dans la colonne de gauche, puis Application console Win32 à droite
- Entrez un nom pour votre projet, par exemple test.
- cliquez sur Paramètres de l'application



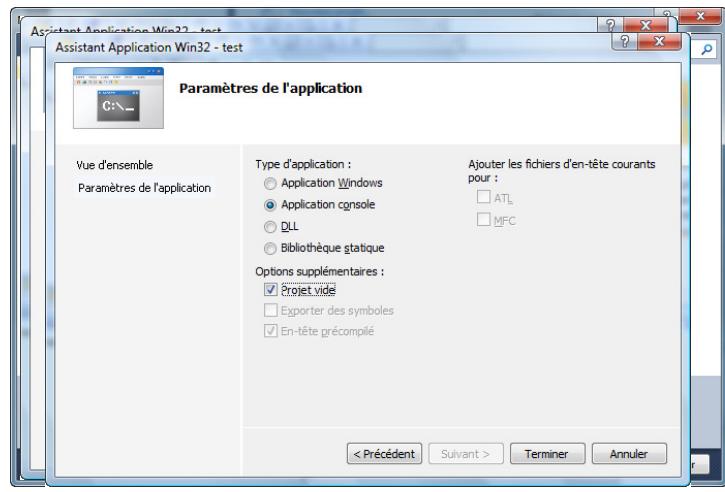
72

## Visual C++

### Créer un nouveau projet

26

- Fichier / Nouveau / Projet
- Win32 dans la colonne de gauche, puis Application console Win32 à droite
- Entrez un nom pour votre projet, par exemple test.
- cliquez sur Paramètres de l'application



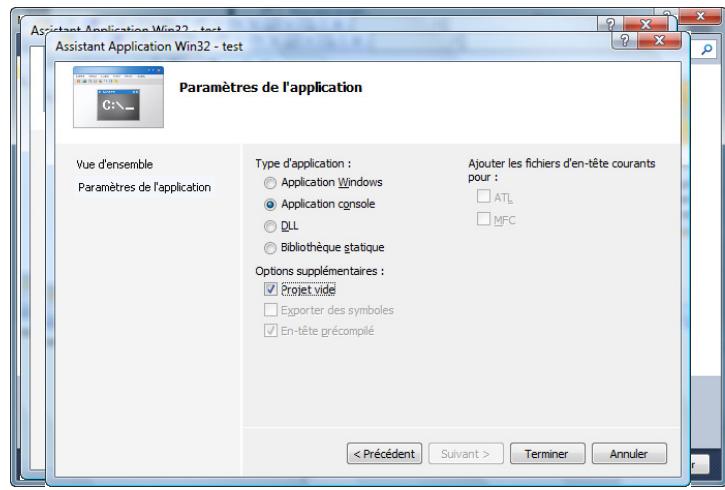
73

## Visual C++

### Créer un nouveau projet

26

- Fichier / Nouveau / Projet
- Win32 dans la colonne de gauche, puis Application console Win32 à droite
- Entrez un nom pour votre projet, par exemple test.
- cliquez sur Paramètres de l'application
- Projet vide / Terminer



74

## Visual C++

### Ajouter un nouveau fichier source

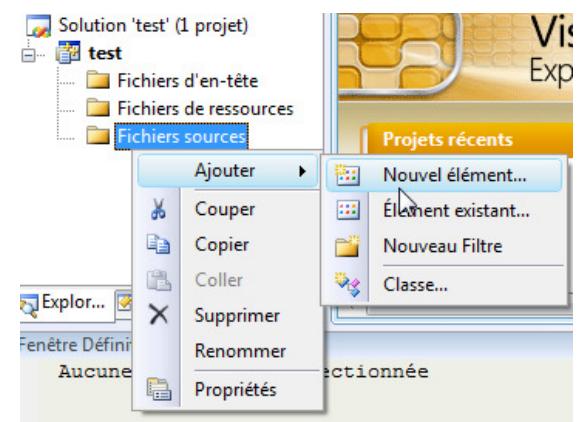
27

75

## Visual C++

### Ajouter un nouveau fichier source

27



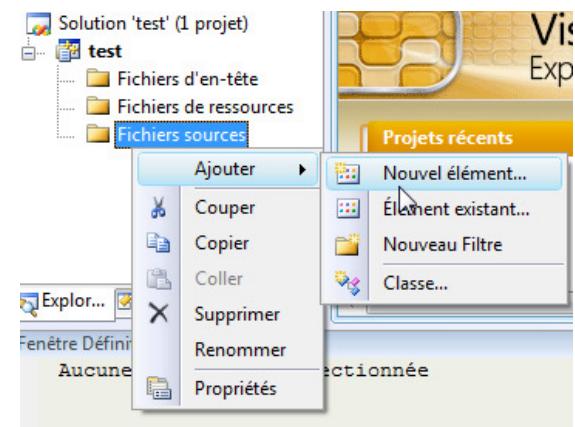
76

## Visual C++

### Ajouter un nouveau fichier source

27

- clic droit sur le dossier Fichiers source



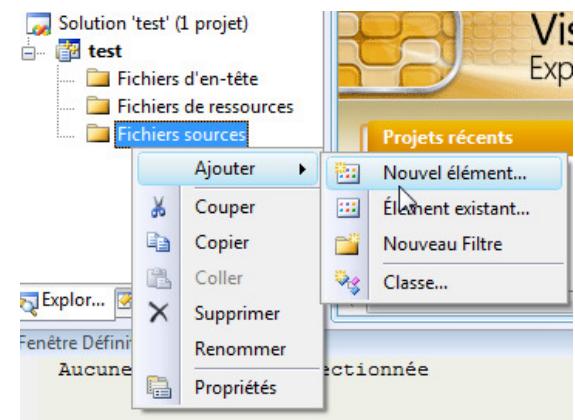
77

## Visual C++

### Ajouter un nouveau fichier source

27

- clic droit sur le dossier Fichiers source
- Ajouter / Nouvel élément



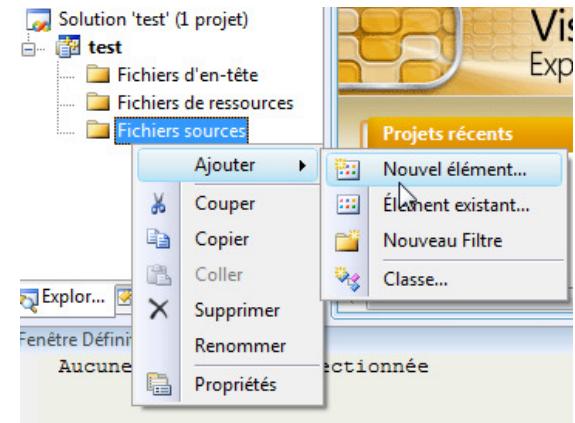
78

## Visual C++

### Ajouter un nouveau fichier source

27

- clic droit sur le dossier Fichiers source
- Ajouter / Nouvel élément
- électionnez Visual C++ à gauche puis Fichier C++ (.cpp)



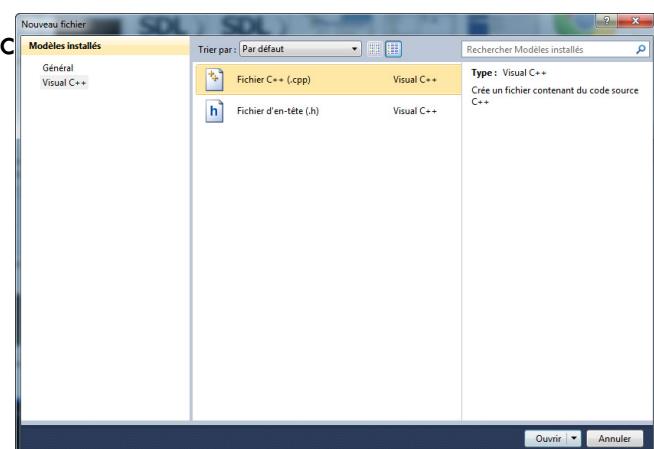
79

## Visual C++

### Ajouter un nouveau fichier source

27

- clic droit sur le dossier Fichiers source
- Ajouter / Nouvel élément
- électionnez Visual C++ à gauche puis Fichier C++ (.cpp)



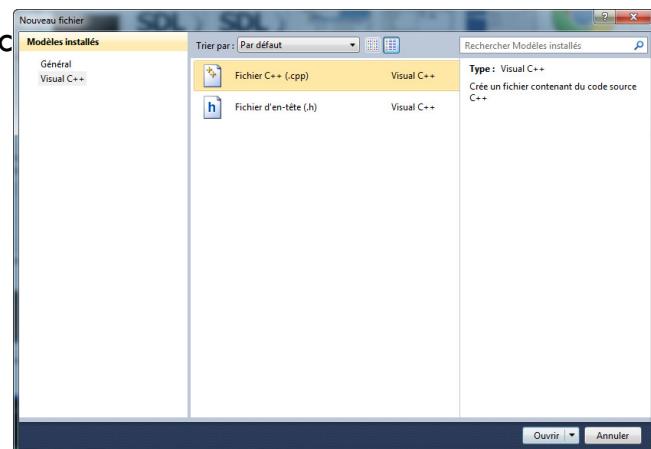
80

## Visual C++

### Ajouter un nouveau fichier source

27

- clic droit sur le dossier Fichiers source
- Ajouter / Nouvel élément
- sélectionnez Visual C++ à gauche puis Fichier C++ (.cpp)
- Entrez un nom pour votre fichier : main.c



81

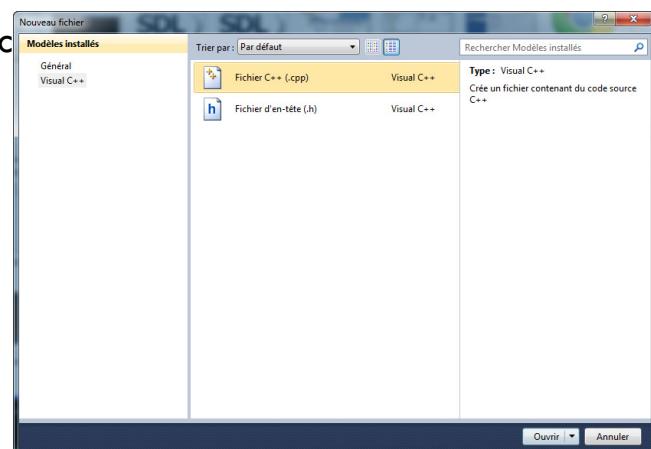
## Visual C++

### Ajouter un nouveau fichier source

27

- clic droit sur le dossier Fichiers source
- Ajouter / Nouvel élément
- sélectionnez Visual C++ à gauche puis Fichier C++ (.cpp)
- Entrez un nom pour votre fichier : main.c

Cliquez sur Ajouter



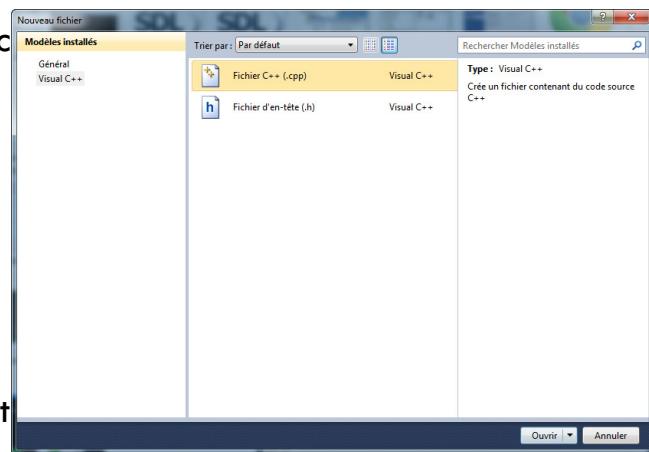
82

## Visual C++

### Ajouter un nouveau fichier source

27

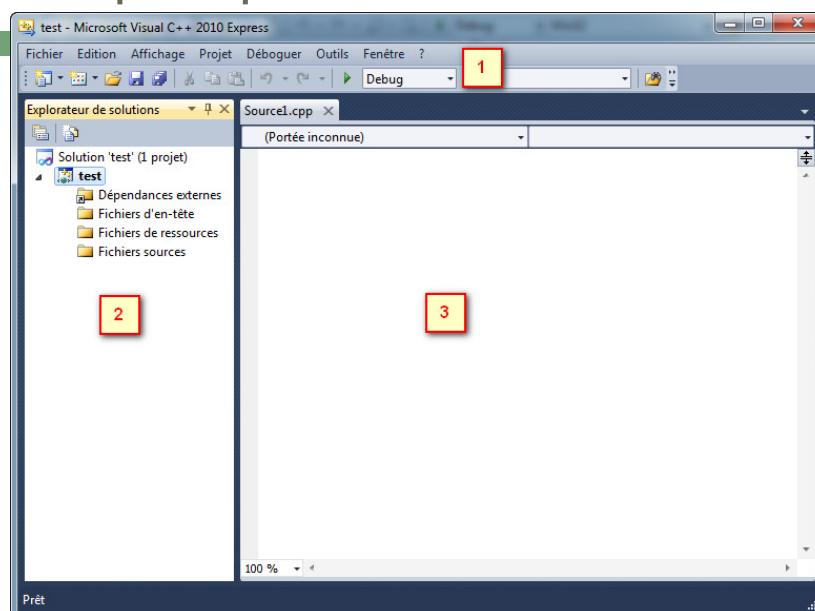
- clic droit sur le dossier Fichiers sources
- Ajouter / Nouvel élément
- sélectionnez Visual C++ à gauche puis Fichier C++ (.cpp)
- Entrez un nom pour votre fichier : main.c
- Cliquez sur Ajouter
- Un fichier vide est créé, je vous invite à l'enregistrer rapidement sous le nom de main.c



83

## La fenêtre principale de Visual

28



84

## La fenêtre principale de Visual

29

(1)

- **La barre d'outils** : tout ce qu'il y a de plus standard. Ouvrir, enregistrer, enregistrer tout, couper, copier, coller, etc.

85

## La fenêtre principale de Visual

29

(1)

- **La barre d'outils** : tout ce qu'il y a de plus standard. Ouvrir, enregistrer, enregistrer tout, couper, copier, coller, etc.
- Par défaut, il semble qu'il n'y ait pas de bouton de barre d'outils pour compiler.

86

## La fenêtre principale de Visual

29

(1)

- **La barre d'outils** : tout ce qu'il y a de plus standard. Ouvrir, enregistrer, enregistrer tout, couper, copier, coller, etc.
- Par défaut, il semble qu'il n'y ait pas de bouton de barre d'outils pour compiler.
- Vous pouvez les rajouter en faisant un clic droit sur la barre d'outils, puis en choisissant **Déboguer** et **Générer** dans la liste.

87

## La fenêtre principale de Visual

29

(1)

- **La barre d'outils** : tout ce qu'il y a de plus standard. Ouvrir, enregistrer, enregistrer tout, couper, copier, coller, etc.
- Par défaut, il semble qu'il n'y ait pas de bouton de barre d'outils pour compiler.
- Vous pouvez les rajouter en faisant un clic droit sur la barre d'outils, puis en choisissant **Déboguer** et **Générer** dans la liste.
- Si vous faites **Générer**, cela créera l'exécutable (ça signifie « **compiler** » pour Visual). Si vous faites **Déboguer / Exécuter**, on devrait vous proposer de compiler avant d'exécuter le programme.

88

## La fenêtre principale de Visual

29

(1)

- **La barre d'outils** : tout ce qu'il y a de plus standard. Ouvrir, enregistrer, enregistrer tout, couper, copier, coller, etc.
- Par défaut, il semble qu'il n'y ait pas de bouton de barre d'outils pour compiler.
- Vous pouvez les rajouter en faisant un clic droit sur la barre d'outils, puis en choisissant **Déboguer** et **Générer** dans la liste.
- Si vous faites **Générer**, cela créera l'exécutable (ça signifie « **compiler** » pour Visual). Si vous faites **Déboguer / Exécuter**, on devrait vous proposer de compiler avant d'exécuter le programme.
- **F7** permet de générer le projet, et **F5** de l'exécuter.

89

## La fenêtre principale de Visual

30

(2)

- la liste des fichiers de votre projet. Cliquez sur l'onglet Explorateur de solutions en bas, si ce n'est déjà fait.

90

## La fenêtre principale de Visual

30

(2)

- la liste des fichiers de votre projet. Cliquez sur l'onglet Explorateur de solutions en bas, si ce n'est déjà fait.
- Vous devriez voir que Visual crée déjà des dossiers pour séparer les différents types de fichiers de votre projet

91

## La fenêtre principale de Visual

30

(2)

- la liste des fichiers de votre projet. Cliquez sur l'onglet Explorateur de solutions en bas, si ce n'est déjà fait.
- Vous devriez voir que Visual crée déjà des dossiers pour séparer les différents types de fichiers de votre projet
- La partie principale : c'est là qu'on modifie les fichiers source.

92

Université Abdelmalek Essaâdi  
École Normale Supérieure  
Tétouan



2023/2024

# Premier Programme

1

Console ou fenêtre ?

32

il existe deux types de programmes, pas plus :

2

## Console ou fenêtre ?

32

il existe deux types de programmes, pas plus :

- les programmes avec **fenêtres** ;

3

## Console ou fenêtre ?

32

il existe deux types de programmes, pas plus :

- les programmes avec **fenêtres** ;
- les programmes en **console**.

4

## Console ou fenêtre ?

32

il existe deux types de programmes, pas plus :

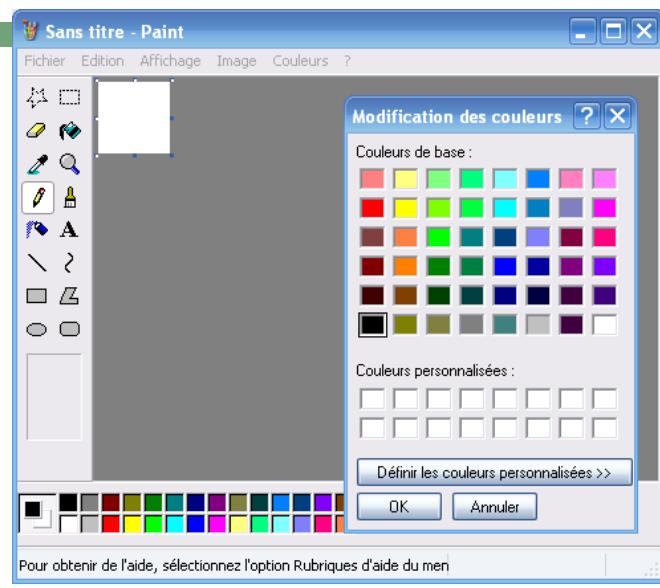
- les programmes avec **fenêtres** ;
- les programmes en **console**.

- Notre IDE nous demandait quel type de programme nous voulions créer et je vous avais dit de répondre **console**.

5

## Les programmes en fenêtres

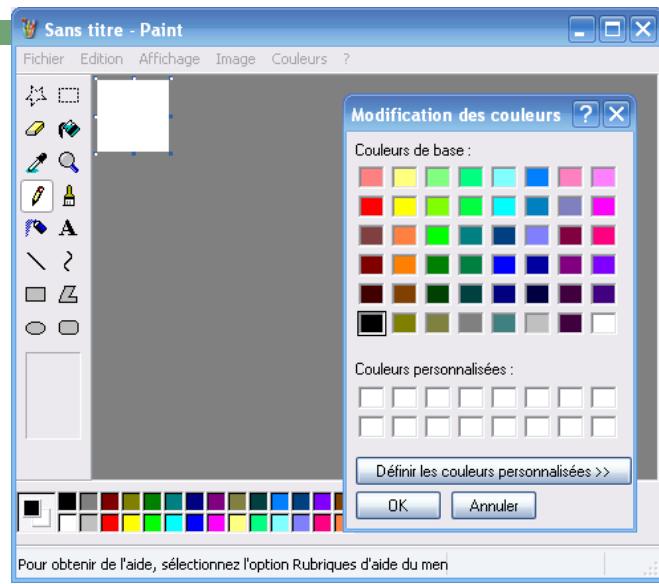
33



6

## Les programmes en fenêtres

- Pour débuter, il vaut mieux commencer par créer des programmes en console.



7

## Les programmes en console

8

## Les programmes en console

34

- Les programmes console ont été les premiers à apparaître.

9

## Les programmes en console

34

- Les programmes console ont été les premiers à apparaître.
- L'ordinateur ne gérait que le noir et blanc et il n'était pas assez puissant pour créer des fenêtres comme on le fait aujourd'hui.

10

## Les programmes en console

34

- Les programmes console ont été les premiers à apparaître.
- l'ordinateur ne gérait que le noir et blanc et il n'était pas assez puissant pour créer des fenêtres comme on le fait aujourd'hui.
- Windows a rendu l'ordinateur « grand public » principalement grâce à sa simplicité et au fait qu'il n'utilisait que des fenêtres.

11

## Les programmes en console

34

- Les programmes console ont été les premiers à apparaître.
- l'ordinateur ne gérait que le noir et blanc et il n'était pas assez puissant pour créer des fenêtres comme on le fait aujourd'hui.
- Windows a rendu l'ordinateur « grand public » principalement grâce à sa simplicité et au fait qu'il n'utilisait que des fenêtres.
- Windows est devenu tellement populaire qu'aujourd'hui beaucoup de monde a oublié ce qu'était la console.

12

## Les programmes en console

35

- **La console n'est pas morte !** En effet, Linux a remis au goût du jour l'utilisation de la console.

13

## Les programmes en console

35

- **La console n'est pas morte !** En effet, Linux a remis au goût du jour l'utilisation de la console.
- aujourd'hui on sait afficher de la couleur, tout n'est donc pas en noir et blanc comme on pourrait le croire ;

14

## Les programmes en console

35

- **La console n'est pas morte !** En effet, Linux a remis au goût du jour l'utilisation de la console.
- aujourd'hui on sait afficher de la couleur, tout n'est donc pas en noir et blanc comme on pourrait le croire ;
- la console est assez peu accueillante pour un débutant ;

15

## Les programmes en console

35

- **La console n'est pas morte !** En effet, Linux a remis au goût du jour l'utilisation de la console.
- aujourd'hui on sait afficher de la couleur, tout n'est donc pas en noir et blanc comme on pourrait le croire ;
- la console est assez peu accueillante pour un débutant ;
- c'est pourtant un outil puissant quand on sait le maîtriser.

16

## Les programmes en console

35

- **La console n'est pas morte !** En effet, Linux a remis au goût du jour l'utilisation de la console.
- aujourd'hui on sait afficher de la couleur, tout n'est donc pas en noir et blanc comme on pourrait le croire ;
- la console est assez peu accueillante pour un débutant ;
- c'est pourtant un outil puissant quand on sait le maîtriser.
  
- la console a évolué : elle peut afficher des couleurs, et rien ne vous empêche de mettre une image de fond.

17

## Les programmes en console

35

- **La console n'est pas morte !** En effet, Linux a remis au goût du jour l'utilisation de la console.
- aujourd'hui on sait afficher de la couleur, tout n'est donc pas en noir et blanc comme on pourrait le croire ;
- la console est assez peu accueillante pour un débutant ;
- c'est pourtant un outil puissant quand on sait le maîtriser.
  
- la console a évolué : elle peut afficher des couleurs, et rien ne vous empêche de mettre une image de fond.

```

2.2.5_appli.html      3.2.7.css      3.6.8.html
2.2.5.css            3.2.8.css      3.6.9.html
2.2.6_appli.html    3.2.9_appli.html ancre.html
2.2.6.css            3.2.9.css      base.php
2.3.10_appli.html   3.3.10.html    cible_formulaire.php
2.3.10.css          3.3.11.css    cible.html
2.3.11_appli.html   3.3.12.css    design1.css
2.3.11.css          3.3.13_appli.html erreur_paragraphe.html
2.3.12.css          3.3.13.css    essai2.css
2.3.13.html         3.3.14_appli.html essai1.css
2.3.14.css          3.3.14.css    images
2.3.15.html         3.3.15.css    tests_design.html
2.3.16.css          3.3.1.css     traitement.php
2.3.17.css          3.3.2.html
2.3.18.css          3.3.3.css
[root@nsl exemples]# cd ..
[root@nsl xhtml-css]# ls
animes               css.php      images      pseudoformats.php
annexes              design.php   images.php  qcma.php
autres               exemples    index.php   tableaux.php
boites_partiel.php   formatage_partiel.php intro.php  texte.php
boites_partie2.php   formatage_partie2.php  liens.php xhtml.php
conclusion.php       formulaires.php  listes.php
[root@nsl xhtml-css]#

```

18

## Les programmes en console

36

*Et sous Windows ? Il n'y a pas de console ?*

19

## Les programmes en console

36

*Et sous Windows ? Il n'y a pas de console ?*

- Démarrer / Accessoires / Invité de commandes,

20

## Les programmes en console

36

*Et sous Windows ? Il n'y a pas de console ?*

- Démarrer / Accessoires / Invite de commandes,
- Démarrer / Exécuter..., et en tapant ensuite cmd.

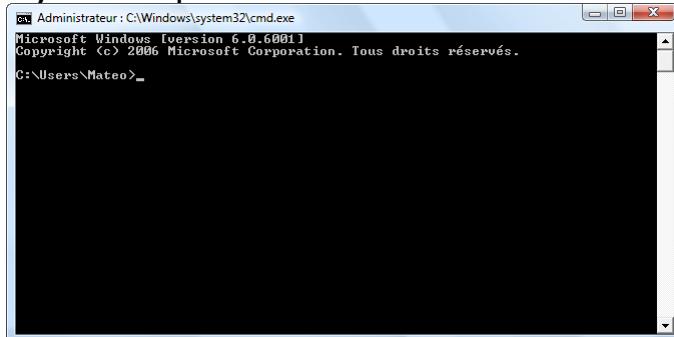
21

## Les programmes en console

36

*Et sous Windows ? Il n'y a pas de console ?*

- Démarrer / Accessoires / Invite de commandes,
- Démarrer / Exécuter..., et en tapant ensuite cmd.



22

## Un minimum de code

37

- Pour n'importe quel programme, il faudra taper un minimum de code.

23

## Un minimum de code

37

- Pour n'importe quel programme, il faudra taper un minimum de code.
- Ce code ne fera rien de particulier mais il est indispensable.

24

## Un minimum de code

37

- Pour n'importe quel programme, il faudra taper un minimum de code.
- Ce code ne fera rien de particulier mais il est indispensable.
- Il devrait servir de base pour la plupart de vos programmes en langage C.

25

## Demandez le code minimal à votre IDE

38

- Selon l'IDE que vous avez choisi, la méthode pour créer un nouveau projet n'est pas la même

26

## Demandez le code minimal à votre IDE

38

- Selon l'IDE que vous avez choisi, la méthode pour créer un nouveau projet n'est pas la même
- Code::Blocks a généré le minimum de code en langage C dont on a besoin:

27

## Demandez le code minimal à votre IDE

38

- Selon l'IDE que vous avez choisi, la méthode pour créer un nouveau projet n'est pas la même
- Code::Blocks a généré le minimum de code en langage C dont on a besoin:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

28

## Demandez le code minimal à votre IDE

38

- Selon l'IDE que vous avez choisi, la méthode pour créer un nouveau projet n'est pas la même
- Code::Blocks a généré le minimum de code en langage C dont on a besoin:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Notez qu'il y a une ligne vide à la fin de ce code. Il est nécessaire de taper sur la touche « Entrée » après la dernière accolade. Chaque fichier en C devrait normalement se terminer par une ligne vide. Si vous ne le faites pas, ce n'est pas grave, mais le compilateur risque de vous afficher un avertissement (**warning**).

29

## Demandez le code minimal à votre IDE

39

- la ligne :

```
int main()
```

30

## Demandez le code minimal à votre IDE

39

- la ligne : `int main()`
- peut aussi s'écrire :

31

## Demandez le code minimal à votre IDE

39

- la ligne : `int main()`
- peut aussi s'écrire :  
`int main(int argc, char *argv[])`

32

## Demandez le code minimal à votre IDE

39

- la ligne : `int main()`
- peut aussi s'écrire :  
`int main(int argc, char *argv[])`
- Les deux écritures sont possibles, mais la seconde (la compliquée) est la plus courante.

33

## Demandez le code minimal à votre IDE

39

- la ligne : `int main()`
- peut aussi s'écrire :  
`int main(int argc, char *argv[])`
- Les deux écritures sont possibles, mais la seconde (la compliquée) est la plus courante.
- Si vous êtes sous un autre IDE, copiez ce code source dans votre fichier [main.c](#)

34

## Demandez le code minimal à votre IDE

39

- la ligne : `int main()`

- peut aussi s'écrire :

`int main(int argc, char *argv[])`

- Les deux écritures sont possibles, mais la seconde (la compliquée) est la plus courante.
- Si vous êtes sous un autre IDE, copiez ce code source dans votre fichier `main.c`
- vous n'avez qu'un seul fichier source appelé `main.c` (le reste, ce sont des fichiers de projet générés par l'IDE).

35

## Analysons le code minimal

40

un programme console qui affiche un message à l'écran!!!!

36

## Analysons le code minimal

40

un programme console qui affiche un message à l'écran!!!!

- les deux premières lignes :

37

## Analysons le code minimal

40

un programme console qui affiche un message à l'écran!!!!

- les deux premières lignes :

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

38

## Analysons le code minimal

40

un programme console qui affiche un message à l'écran!!!!

- les deux premières lignes :

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

- des lignes spéciales que l'on ne voit qu'en haut des fichiers source.

39

## Analysons le code minimal

40

un programme console qui affiche un message à l'écran!!!!

- les deux premières lignes :

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

- des lignes spéciales que l'on ne voit qu'en haut des fichiers source.
- elles commencent par un dièse #.

40

## Analysons le code minimal

40

un programme console qui affiche un message à l'écran!!!!

- les deux premières lignes :

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

- des lignes spéciales que l'on ne voit qu'en haut des fichiers source.
- elles commencent par un dièse #.
- on les **directives de préprocesseur**

41

## Analysons le code minimal

40

un programme console qui affiche un message à l'écran!!!!

- les deux premières lignes :

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

- des lignes spéciales que l'on ne voit qu'en haut des fichiers source.
- elles commencent par un dièse #.
- on les **directives de préprocesseur**
- des lignes qui seront lues par un programme appelé **préprocesseur**, un programme qui se lance au **début de la compilation**.

42

## Analysons le code minimal

41

*elles signifient quoi, ces lignes ?*

43

## Analysons le code minimal

41

*elles signifient quoi, ces lignes ?*

- Le mot **include** en anglais signifie « inclure » en français.

44

## Analysons le code minimal

41

*elles signifient quoi, ces lignes ?*

- Le mot **include** en anglais signifie « inclure » en français.
- Ces lignes demandent d'inclure des fichiers au projet, c'est-à-dire d'ajouter des fichiers pour la compilation.

45

## Analysons le code minimal

41

*elles signifient quoi, ces lignes ?*

- Le mot **include** en anglais signifie « inclure » en français.
- Ces lignes demandent d'inclure des fichiers au projet, c'est-à-dire d'ajouter des fichiers pour la compilation.
- Ces fichiers s'appellent **stdio.h** et **stdlib.h**.

46

## Analysons le code minimal

41

*elles signifient quoi, ces lignes ?*

- Le mot **include** en anglais signifie « inclure » en français.
- Ces lignes demandent d'inclure des fichiers au projet, c'est-à-dire d'ajouter des fichiers pour la compilation.
- Ces fichiers s'appellent **stdio.h** et **stdlib.h**.
- Ces fichiers existent déjà, des fichiers source tout prêts.

47

## Analysons le code minimal

41

*elles signifient quoi, ces lignes ?*

- Le mot **include** en anglais signifie « inclure » en français.
- Ces lignes demandent d'inclure des fichiers au projet, c'est-à-dire d'ajouter des fichiers pour la compilation.
- Ces fichiers s'appellent **stdio.h** et **stdlib.h**.
- Ces fichiers existent déjà, des fichiers source tout prêts.
- On verra plus tard qu'on les appelle des **bibliothèques** (certains parlent aussi de **librairies**)

48

## Analysons le code minimal

41

*elles signifient quoi, ces lignes ?*

- Le mot **include** en anglais signifie « inclure » en français.
- Ces lignes demandent d'inclure des fichiers au projet, c'est-à-dire d'ajouter des fichiers pour la compilation.
- Ces fichiers s'appellent **stdio.h** et **stdlib.h**.
- Ces fichiers existent déjà, des fichiers source tout prêts.
- On verra plus tard qu'on les appelle des **bibliothèques** (certains parlent aussi de **librairies**)
- Ces fichiers contiennent du code tout prêt qui permet d'afficher du texte à l'écran.

49

## Analysons le code minimal

42

- Sans ces fichiers, écrire du texte à l'écran aurait été mission impossible. L'ordinateur à la base ne sait rien faire, il faut tout lui dire.

50

## Analysons le code minimal

42

- Sans ces fichiers, écrire du texte à l'écran aurait été mission impossible. L'ordinateur à la base ne sait rien faire, il faut tout lui dire.
- En résumé les deux premières lignes incluent les **bibliothèques** qui vont nous permettre (entre autres) d'afficher du texte à l'écran assez « facilement ».

51

## Analysons le code minimal

43

- La suite:

52

## Analysons le code minimal

43

- La suite:

```
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

53

## Analysons le code minimal

43

- La suite:

```
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

- c'est ce qu'on appelle **une fonction**.

54

## Analysons le code minimal

43

- La suite:

```
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

- c'est ce qu'on appelle **une fonction**.
- Un programme en langage C est constitué de fonctions, il ne contient quasiment que ça.

55

## Analysons le code minimal

43

- La suite:

```
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

- c'est ce qu'on appelle **une fonction**.
- Un programme en langage C est constitué de fonctions, il ne contient quasiment que ça.
- Pour le moment, le programme ne contient donc qu'une seule fonction.

56

## Analysons le code minimal

44

- Une fonction permet de rassembler plusieurs commandes à l'ordinateur.

57

## Analysons le code minimal

44

- Une fonction permet de rassembler plusieurs commandes à l'ordinateur.
- Regroupées dans une fonction, les commandes permettent de faire quelque chose de précis.

58

## Analysons le code minimal

44

- Une fonction permet de rassembler plusieurs commandes à l'ordinateur.
- Regroupées dans une fonction, les commandes permettent de faire quelque chose de précis.
- **Exemple:** on peut créer une fonction **ouvrir\_fichier** qui contiendra une suite d'instructions pour l'ordinateur lui expliquant comment ouvrir un fichier.

59

## Analysons le code minimal

44

- Une fonction permet de rassembler plusieurs commandes à l'ordinateur.
- Regroupées dans une fonction, les commandes permettent de faire quelque chose de précis.
- **Exemple:** on peut créer une fonction **ouvrir\_fichier** qui contiendra une suite d'instructions pour l'ordinateur lui expliquant comment ouvrir un fichier.
- une fois la fonction écrite, vous n'aurez plus qu'à dire **ouvrir\_fichier**, et l'ordinateur saura comment faire sans que vous ayez à tout répéter !

60

## Analysons le code minimal

45

- La première ligne contient le **nom** de la fonction, c'est le deuxième mot.

61

## Analysons le code minimal

45

- La première ligne contient le **nom** de la fonction, c'est le deuxième mot.
- La fonction s'appelle donc **main**. C'est un nom de fonction particulier qui signifie « **principal** ».

62

## Analysons le code minimal

45

- La première ligne contient le **nom** de la fonction, c'est le deuxième mot.
- La fonction s'appelle donc **main**. C'est un nom de fonction particulier qui signifie « **principal** ».
- **main** est la fonction principale de votre programme, **c'est toujours par la fonction main que le programme commence**.

63

## Analysons le code minimal

45

- La première ligne contient le **nom** de la fonction, c'est le deuxième mot.
- La fonction s'appelle donc **main**. C'est un nom de fonction particulier qui signifie « **principal** ».
- **main** est la fonction principale de votre programme, **c'est toujours par la fonction main que le programme commence**.
- Une fonction a un début et une fin, délimités par des accolades { et }

64

## Analysons le code minimal

45

- La première ligne contient le **nom** de la fonction, c'est le deuxième mot.
- La fonction s'appelle donc **main**. C'est un nom de fonction particulier qui signifie « **principal** ».
- **main** est la fonction principale de votre programme, **c'est toujours par la fonction main que le programme commence**.
- Une fonction a un début et une fin, délimités par des accolades { et }
- Toute la fonction main se trouve donc entre ces accolades.

65

## Analysons le code minimal

46

- La fonction main contient deux lignes :

66

## Analysons le code minimal

46

- La fonction main contient deux lignes :

```
printf("Hello world!\n");
return 0;
```

67

## Analysons le code minimal

46

- La fonction main contient deux lignes :

```
printf("Hello world!\n");
return 0;
```

- On les appelle **instructions**

68

## Analysons le code minimal

46

- La fonction main contient deux lignes :

```
printf("Hello world!\n");
return 0;
```

- On les appelle **instructions**
- Chaque instruction est une commande à l'ordinateur.

69

## Analysons le code minimal

46

- La fonction main contient deux lignes :

```
printf("Hello world!\n");
return 0;
```

- On les appelle **instructions**
- Chaque instruction est une commande à l'ordinateur.
- Chacune de ces lignes demande à l'ordinateur de faire quelque chose de précis.

70

## Analysons le code minimal

46

- La fonction main contient deux lignes :

```
printf("Hello world!\n");
return 0;
```

- On les appelle **instructions**
- Chaque instruction est une commande à l'ordinateur.
- Chacune de ces lignes demande à l'ordinateur de faire quelque chose de précis.
- Un programme, ce n'est au bout du compte rien d'autre qu'une série d'instructions : « fais ceci », « fais cela ». Vous donnez des ordres à votre ordinateur et il les exécute.

71

## Analysons le code minimal

47

- **Très important** : toute instruction se termine obligatoirement par un point-

virgule «  »

72

## Analysons le code minimal

47

- **Très important** : toute instruction se termine obligatoirement par un point-virgule « ; »
- La première ligne : `printf("Hello world!\n");` demande à afficher le message « Hello world! » à l'écran.

73

## Analysons le code minimal

47

- **Très important** : toute instruction se termine obligatoirement par un point-virgule « ; »
- La première ligne : `printf("Hello world!\n");` demande à afficher le message « Hello world! » à l'écran.
- Quand le programme arrivera à cette ligne, il va donc afficher un message à l'écran, puis passer à l'instruction suivante.

74

## Analysons le code minimal

47

- **Très important** : toute instruction se termine obligatoirement par un point-virgule « ; »
  - La première ligne : `printf("Hello world!\n");` demande à afficher le message « Hello world! » à l'écran.
  - Quand le programme arrivera à cette ligne, il va donc afficher un message à l'écran, puis passer à l'instruction suivante.
  - La deuxième ligne: `return 0;` indique qu'on arrive à la fin de la fonction **main** et demande de renvoyer la valeur 0.

75

## Analysons le code minimal

48

Pourquoi le programme renverrait-il le nombre 0 ?

76

## Analysons le code minimal

48

Pourquoi le programme renverrait-il le nombre 0 ?

- Chaque programme une fois terminé renvoie une valeur, par exemple pour dire que tout s'est bien passé.

77

## Analysons le code minimal

48

Pourquoi le programme renverrait-il le nombre 0 ?

- Chaque programme une fois terminé renvoie une valeur, par exemple pour dire que tout s'est bien passé.
- En pratique, **0** signifie « **tout s'est bien passé** » et n'importe quelle **autre valeur** signifie « **erreur** ».

78

## Analysons le code minimal

48

Pourquoi le programme renverrait-il le nombre 0 ?

- Chaque programme une fois terminé renvoie une valeur, par exemple pour dire que tout s'est bien passé.
- En pratique, **0** signifie « **tout s'est bien passé** » et n'importe quelle **autre valeur** signifie « **erreur** ».
- La plupart du temps, cette valeur n'est pas vraiment utilisée, mais il faut quand même en renvoyer une.

79

## Analysons le code minimal

48

Pourquoi le programme renverrait-il le nombre 0 ?

- Chaque programme une fois terminé renvoie une valeur, par exemple pour dire que tout s'est bien passé.
- En pratique, **0** signifie « **tout s'est bien passé** » et n'importe quelle **autre valeur** signifie « **erreur** ».
- La plupart du temps, cette valeur n'est pas vraiment utilisée, mais il faut quand même en renvoyer une.
- Le programme aurait marché sans le return 0, mais on va dire que c'est plus propre et plus sérieux de le mettre, donc **on le met**.

80

## Analysons le code minimal

49

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

} Directives de préprocesseur

```
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

} Instructions } Fonction

81

## Tester le programme

50

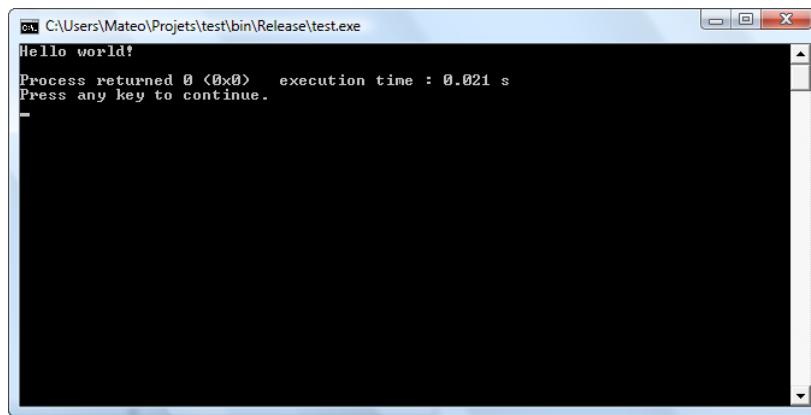
- compiler le projet, puis l'exécuter: cliquez sur l'icône **Build & Run**

82

## Tester le programme

50

- compiler le projet, puis l'exécuter: cliquez sur l'icône **Build & Run**



83

## Écrire un message à l'écran

51

- afficher le message « Bonjour » à l'écran.

84

## Écrire un message à l'écran

51

- afficher le message « Bonjour » à l'écran.

Comment fait-on pour choisir le texte qui s'affiche à l'écran ?

85

## Écrire un message à l'écran

51

- afficher le message « Bonjour » à l'écran.

Comment fait-on pour choisir le texte qui s'affiche à l'écran ?

- il suffit simplement de remplacer « **Hello world!** » par « **Bonjour** » dans la ligne qui fait appel à **printf**.

86

## Écrire un message à l'écran

51

- afficher le message « Bonjour » à l'écran.

Comment fait-on pour choisir le texte qui s'affiche à l'écran ?

- il suffit simplement de remplacer « **Hello world!** » par « **Bonjour** » dans la ligne qui fait appel à **printf**.
- printf** est une instruction. Elle commande à l'ordinateur : « **Affiche-moi ce message à l'écran** ».

87

## Écrire un message à l'écran

51

- afficher le message « Bonjour » à l'écran.

Comment fait-on pour choisir le texte qui s'affiche à l'écran ?

- il suffit simplement de remplacer « **Hello world!** » par « **Bonjour** » dans la ligne qui fait appel à **printf**.
- printf** est une instruction. Elle commande à l'ordinateur : « **Affiche-moi ce message à l'écran** ».
- Il faut savoir que **printf** est en fait une fonction qui a déjà été écrite par d'autres programmeurs avant nous.

88

## Écrire un message à l'écran

52

Cette fonction, où se trouve-t-elle ? Moi je ne vois que la fonction **main** !

89

## Écrire un message à l'écran

52

Cette fonction, où se trouve-t-elle ? Moi je ne vois que la fonction **main** !

Vous vous souvenez de ces deux lignes ?

90

## Écrire un message à l'écran

52

Cette fonction, où se trouve-t-elle ? Moi je ne vois que la fonction **main** !

- Vous vous souvenez de ces deux lignes ?

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

91

## Écrire un message à l'écran

52

Cette fonction, où se trouve-t-elle ? Moi je ne vois que la fonction **main** !

- Vous vous souvenez de ces deux lignes ?
- elles permettaient d'ajouter des **bibliothèques** dans votre programme.

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

92

## Écrire un message à l'écran

52

Cette fonction, où se trouve-t-elle ? Moi je ne vois que la fonction **main** !

- Vous vous souvenez de ces deux lignes ?

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

- elles permettaient d'ajouter des **bibliothèques** dans votre programme.
- Les **bibliothèques** sont en fait des fichiers avec des tonnes de **fonctions toutes prêtes à l'intérieur**.

93

## Écrire un message à l'écran

52

Cette fonction, où se trouve-t-elle ? Moi je ne vois que la fonction **main** !

- Vous vous souvenez de ces deux lignes ?

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

- elles permettaient d'ajouter des **bibliothèques** dans votre programme.
- Les **bibliothèques** sont en fait des fichiers avec des tonnes de **fonctions toutes prêtes à l'intérieur**.
- Ces fichiers-là (**stdio.h** et **stdlib.h**) contiennent la plupart des fonctions de base dont on a besoin dans un programme.

94

## Écrire un message à l'écran

52

Cette fonction, où se trouve-t-elle ? Moi je ne vois que la fonction **main** !

- Vous vous souvenez de ces deux lignes ?

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

- elles permettaient d'ajouter des **bibliothèques** dans votre programme.
- Les **bibliothèques** sont en fait des fichiers avec des tonnes de **fonctions toutes prêtes à l'intérieur**.
- Ces fichiers-là (**stdio.h** et **stdlib.h**) contiennent la plupart des fonctions de base dont on a besoin dans un programme.
- **stdio.h** en particulier contient des fonctions permettant d'afficher des choses à l'écran (comme **printf**) mais aussi de demander à l'utilisateur de taper quelque chose.

95

## Écrire un message à l'écran

53

- Dans la fonction **main**, on fait appel à la fonction **printf**.

96

## Écrire un message à l'écran

53

- Dans la fonction **main**, on fait appel à la fonction **printf**.
- C'est une fonction qui en appelle une autre (ici, main appelle **printf**).

97

## Écrire un message à l'écran

53

- Dans la fonction **main**, on fait appel à la fonction **printf**.
- C'est une fonction qui en appelle une autre (ici, main appelle **printf**).
- En langage C : une fonction contient des instructions qui appellent d'autres fonctions, et ainsi de suite.

98

## Écrire un message à l'écran

53

- Dans la fonction **main**, on fait appel à la fonction **printf**.
- C'est une fonction qui en appelle une autre (ici, main appelle **printf**).
- En langage C : une fonction contient des instructions qui appellent d'autres fonctions, et ainsi de suite.
  
- Pour faire appel à une fonction, c'est simple : il suffit d'écrire son nom, suivi de deux parenthèses, puis un point-virgule.

99

## Écrire un message à l'écran

53

- Dans la fonction **main**, on fait appel à la fonction **printf**.
- C'est une fonction qui en appelle une autre (ici, main appelle **printf**).
- En langage C : une fonction contient des instructions qui appellent d'autres fonctions, et ainsi de suite.
  
- Pour faire appel à une fonction, c'est simple : il suffit d'écrire son nom, suivi de deux parenthèses, puis un point-virgule.

```
printf();
```

100

## Écrire un message à l'écran

54

- Il faut indiquer quoi écrire à l'écran.

101

## Écrire un message à l'écran

54

- Il faut indiquer quoi écrire à l'écran.
- Pour faire ça, il faut donner à la fonction printf le texte à afficher.

102

## Écrire un message à l'écran

54

### □ Il faut indiquer quoi écrire à l'écran.

- Pour faire ça, il faut donner à la fonction printf le texte à afficher.
  - Pour ce faire, ouvrez des guillemets à l'intérieur des parenthèses et tapez le texte à afficher entre ces guillemets.

103

## Écrire un message à l'écran

54

### □ Il faut indiquer quoi écrire à l'écran.

- Pour faire ça, il faut donner à la fonction printf le texte à afficher.
  - Pour ce faire, ouvrez des guillemets à l'intérieur des parenthèses et tapez le texte à afficher entre ces guillemets.

```
printf("Bonjour");
```

104

## Écrire un message à l'écran

54

□ Il faut indiquer quoi écrire à l'écran.

- Pour faire ça, il faut donner à la fonction printf le texte à afficher entre ces guillemets.
- Pour ce faire, ouvrez des guillemets à l'intérieur des parenthèses de la fonction printf.

```
printf("Bonjour");
```

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

int main()
{
    printf("Bonjour");
    return 0;
}
```

105

## Écrire un message à l'écran

54

□ Il faut indiquer quoi écrire à l'écran.

- Pour faire ça, il faut donner à la fonction printf le texte à afficher entre ces guillemets.
- Pour ce faire, ouvrez des guillemets à l'intérieur des parenthèses de la fonction printf.

```
printf("Bonjour");
```

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

int main()
{
    printf("Bonjour");
    return 0;
}
```

- deux instructions qui commandent dans l'ordre à l'ordinateur :

106

## Écrire un message à l'écran

54

□ Il faut indiquer quoi écrire à l'écran.

- Pour faire ça, il faut donner à la fonction printf le texte à afficher entre ces guillemets.
- Pour ce faire, ouvrez des guillemets à l'intérieur des parenthèses de la fonction printf.

```
printf("Bonjour");
```

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

int main()
{
    printf("Bonjour");
    return 0;
}
```

□ deux instructions qui commandent dans l'ordre à l'ordinateur :

- affiche « Bonjour » à l'écran ;

107

## Écrire un message à l'écran

54

□ Il faut indiquer quoi écrire à l'écran.

- Pour faire ça, il faut donner à la fonction printf le texte à afficher entre ces guillemets.
- Pour ce faire, ouvrez des guillemets à l'intérieur des parenthèses de la fonction printf.

```
printf("Bonjour");
```

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

int main()
{
    printf("Bonjour");
    return 0;
}
```

□ deux instructions qui commandent dans l'ordre à l'ordinateur :

- affiche « Bonjour » à l'écran ;
- la fonction main est terminée, renvoie 0. Le programme s'arrête alors.

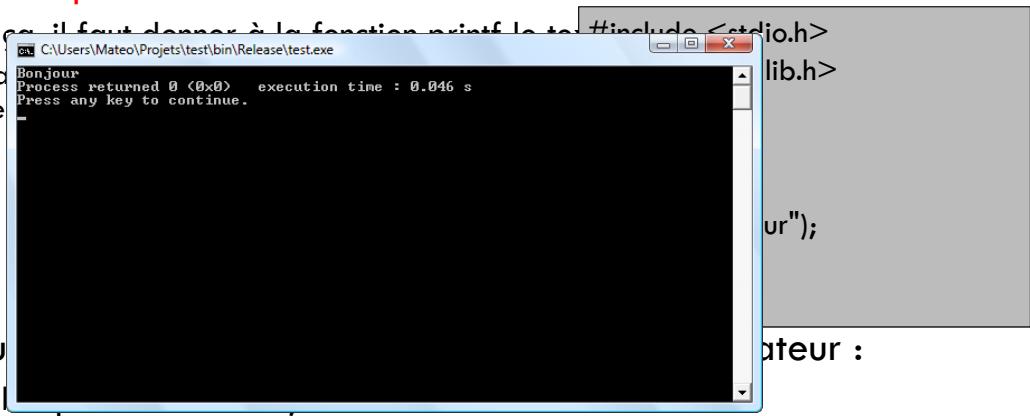
108

## Écrire un message à l'écran

54

Il faut indiquer quoi écrire à l'écran.

- Pour faire ça, il faut donner à la fonction `printf` le texte à afficher et l'appeler :
- Pour ce faire, il faut appeler la fonction `printf` avec le texte à afficher en paramètre.



deux instructions sont nécessaires pour écrire à l'écran :

- affiche « Bonjour »
- la fonction main est terminée, renvoie 0. Le programme s'arrête alors.

109

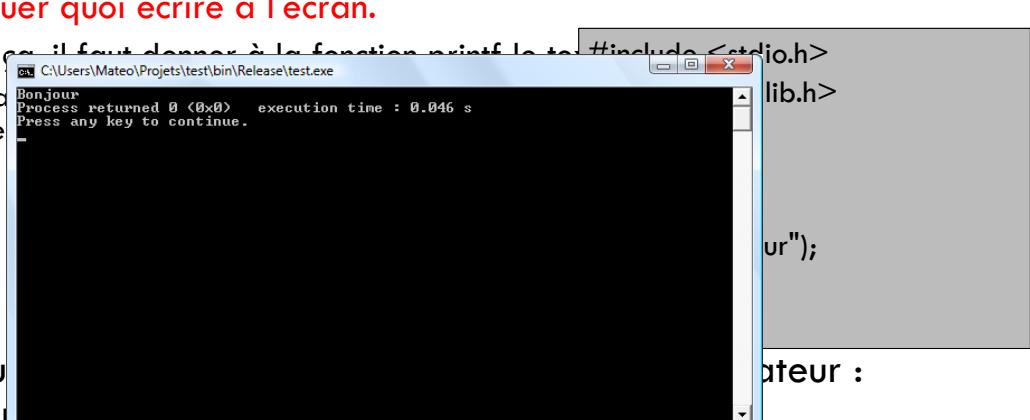
## Écrire un message à l'écran

54

la ligne du « Bonjour » est un peu collée avec le reste du texte

Il faut indiquer quoi écrire à l'écran.

- Pour faire ça, il faut donner à la fonction `printf` le texte à afficher et l'appeler :
- Pour ce faire, il faut appeler la fonction `printf` avec le texte à afficher en paramètre.



deux instructions sont nécessaires pour écrire à l'écran :

- affiche « Bonjour »
- la fonction main est terminée, renvoie 0. Le programme s'arrête alors.

110

## Les caractères spéciaux

55

- Les caractères spéciaux sont des lettres spéciales qui permettent d'indiquer qu'on veut aller à la ligne, faire une tabulation, etc.

111

## Les caractères spéciaux

55

- Les caractères spéciaux sont des lettres spéciales qui permettent d'indiquer qu'on veut aller à la ligne, faire une tabulation, etc.
- c'est un ensemble de deux caractères. Le premier d'entre eux est toujours un anti-slash ( \ ), et le second un nombre ou une **lettre**.

112

## Les caractères spéciaux

55

- Les caractères spéciaux sont des lettres spéciales qui permettent d'indiquer qu'on veut aller à la ligne, faire une tabulation, etc.
- c'est un ensemble de deux caractères. Le premier d'entre eux est toujours un anti-slash ( \ ), et le second un nombre ou une **lettre**.
- deux caractères spéciaux courants que vous aurez probablement besoin d'utiliser:

113

## Les caractères spéciaux

55

- Les caractères spéciaux sont des lettres spéciales qui permettent d'indiquer qu'on veut aller à la ligne, faire une tabulation, etc.
- c'est un ensemble de deux caractères. Le premier d'entre eux est toujours un anti-slash ( \ ), et le second un nombre ou une **lettre**.
- deux caractères spéciaux courants que vous aurez probablement besoin d'utiliser:
  - \n : retour à la ligne (= « Entrée ») ;
  - \t : tabulation.

114

## Les caractères spéciaux

56

- faire un retour à la ligne juste après le mot « Bonjour »:

115

## Les caractères spéciaux

56

- faire un retour à la ligne juste après le mot « Bonjour »:

```
printf("Bonjour\n");
```

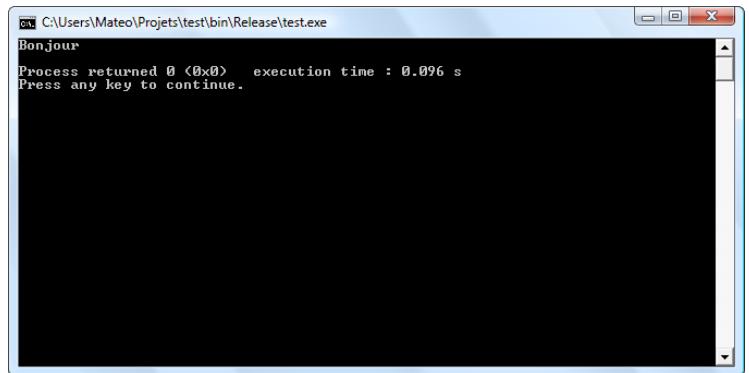
116

## Les caractères spéciaux

56

- faire un retour à la ligne juste après le mot « Bonjour »:

```
printf("Bonjour\n");
```



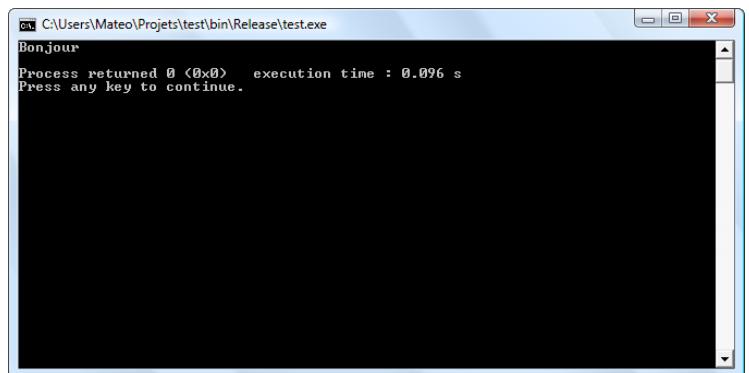
117

## Les caractères spéciaux

56

- faire un retour à la ligne juste après le mot « Bonjour »:

```
printf("Bonjour\n");
```



- Test...

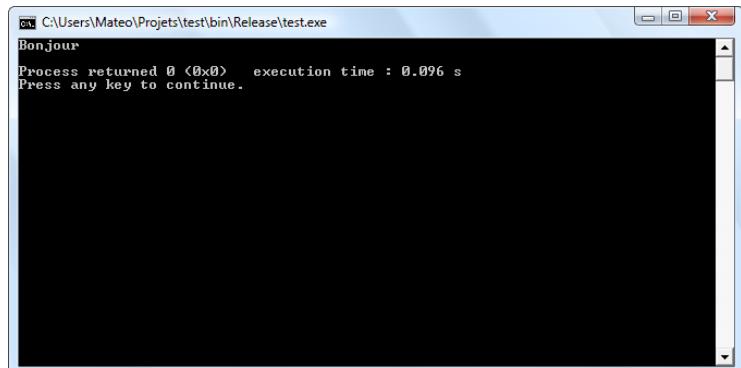
118

## Les caractères spéciaux

56

- faire un retour à la ligne juste après le mot « Bonjour »:

```
printf("Bonjour\n");
```



- Test...

```
printf("Bonjour\nAu Revoir\n");
```

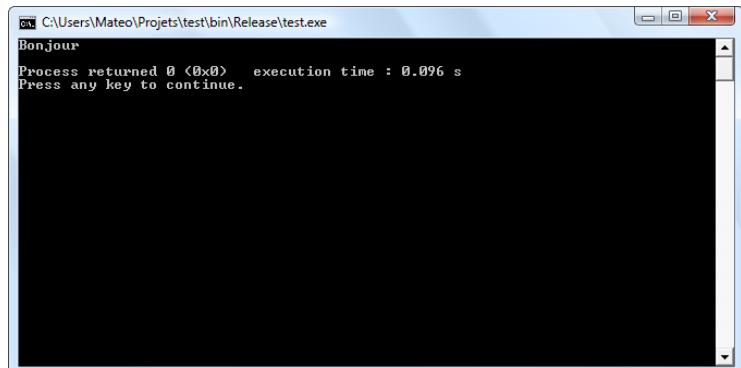
119

## Les caractères spéciaux

56

- faire un retour à la ligne juste après le mot « Bonjour »:

```
printf("Bonjour\n");
```



- Test...

```
printf("Bonjour\nAu Revoir\n");
```

- Afficher Bonjour Gérard

120

## Les commentaires

57

- Quel que soit le langage de programmation, on a la possibilité d'ajouter des commentaires à son code.

121

## Les commentaires

57

- Quel que soit le langage de programmation, on a la possibilité d'ajouter des commentaires à son code.

Qu'est-ce que ça veut dire, « commenter » ?

122

## Les commentaires

57

- Quel que soit le langage de programmation, on a la possibilité d'ajouter des commentaires à son code.  
**Qu'est-ce que ça veut dire, « commenter » ?**
- taper du texte au milieu de votre programme pour indiquer ce qu'il fait, à quoi sert telle ligne de code, etc.

123

## Les commentaires

57

- Quel que soit le langage de programmation, on a la possibilité d'ajouter des commentaires à son code.  
**Qu'est-ce que ça veut dire, « commenter » ?**
- taper du texte au milieu de votre programme pour indiquer ce qu'il fait, à quoi sert telle ligne de code, etc.
- C'est vraiment quelque chose d'indispensable car, même en étant un génie de la programmation, on a besoin de faire quelques annotations.

124

## Les commentaires

58

permet :

125

## Les commentaires

58

permet :

- de vous retrouver au milieu d'un de vos codes source plus tard. On ne dirait pas comme ça, mais on oublie vite comment fonctionnent les programmes qu'on a écrits.

126

## Les commentaires

58

permet :

- de vous retrouver au milieu d'un de vos codes source plus tard. On ne dirait pas comme ça, mais on oublie vite comment fonctionnent les programmes qu'on a écrits.
- Si vous faites une pause ne serait-ce que quelques jours, vous aurez besoin de vous aider de vos propres commentaires pour vous retrouver dans un gros code ;

127

## Les commentaires

58

permet :

- de vous retrouver au milieu d'un de vos codes source plus tard. On ne dirait pas comme ça, mais on oublie vite comment fonctionnent les programmes qu'on a écrits.
- Si vous faites une pause ne serait-ce que quelques jours, vous aurez besoin de vous aider de vos propres commentaires pour vous retrouver dans un gros code ;
- Si vous donnez votre projet à quelqu'un d'autre (qui ne connaît a priori pas votre code source), cela lui permettra de se familiariser avec bien plus rapidement ;

128

## Les commentaires

59

- Il y a plusieurs manières d'insérer un commentaire. Tout dépend de la longueur du commentaire que vous voulez écrire.

129

## Les commentaires

59

- Il y a plusieurs manières d'insérer un commentaire. Tout dépend de la longueur du commentaire que vous voulez écrire.
- **commentaire est court** : il tient sur une seule ligne, il ne fait que quelques mots.

130

## Les commentaires

59

- Il y a plusieurs manières d'insérer un commentaire. Tout dépend de la longueur du commentaire que vous voulez écrire.
- **commentaire est court** : il tient sur une seule ligne, il ne fait que quelques mots.
  - ▣ taper un double slash (//) suivi de votre commentaire:

131

## Les commentaires

59

- Il y a plusieurs manières d'insérer un commentaire. Tout dépend de la longueur du commentaire que vous voulez écrire.
- **commentaire est court** : il tient sur une seule ligne, il ne fait que quelques mots.
  - ▣ taper un double slash (//) suivi de votre commentaire:

```
printf("Bonjour"); // Cette instruction affiche Bonjour à l'écran
```

132

## Les commentaires

60

- **commentaire est long** : écrire plusieurs phrases qui tiennent sur plusieurs lignes.

133

## Les commentaires

60

- **commentaire est long** : écrire plusieurs phrases qui tiennent sur plusieurs lignes.
- taper un code qui signifie « **début de commentaire** » et un autre code qui signifie « **fin de commentaire** » :

134

## Les commentaires

60

- **commentaire est long** : écrire plusieurs phrases qui tiennent sur plusieurs lignes.
- taper un code qui signifie « **début de commentaire** » et un autre code qui signifie « **fin de commentaire** » :
  - pour indiquer le début du commentaire : tapez un slash suivi d'une étoile (`/*`) ;

135

## Les commentaires

60

- **commentaire est long** : écrire plusieurs phrases qui tiennent sur plusieurs lignes.
- taper un code qui signifie « **début de commentaire** » et un autre code qui signifie « **fin de commentaire** » :
  - pour indiquer le début du commentaire : tapez un slash suivi d'une étoile (`/*`) ;
  - pour indiquer la fin du commentaire : tapez une étoile suivie d'un slash (`*/`).

136

```
/*
Ci-dessous, ce sont des directives de préprocesseur. Ces lignes permettent d'ajouter des fichiers
au projet, fichiers que l'on appelle bibliothèques. Grâce à ces bibliothèques, on disposera de
fonctions toutes prêtes pour afficher par exemple un message à l'écran.
*/
#include <stdio.h>
#include <stdlib.h>
/*
Ci-dessous, vous avez la fonction principale du programme, appelée main. C'est par cette
fonction que tous les programmes commencent. Ici, ma fonction se contente d'afficher Bonjour à
l'écran.
*/
int main()
{
    printf("Bonjour"); // Cette instruction affiche Bonjour à l'écran
    return 0;           // Le programme renvoie le nombre 0 puis s'arrête
}
```

137

## Les commentaires

61

- Lors de la compilation, tous les commentaires seront ignorés.

138

## Les commentaires

61

- Lors de la compilation, tous les commentaires seront ignorés.
- Ces commentaires n'apparaîtront pas dans le programme final, ils servent seulement aux programmeurs.

139

## Les commentaires

61

- Lors de la compilation, tous les commentaires seront ignorés.
- Ces commentaires n'apparaîtront pas dans le programme final, ils servent seulement aux programmeurs.
- Normalement, on ne commente pas chaque ligne du programme.

140

## Les commentaires

61

- Lors de la compilation, tous les commentaires seront ignorés.
- Ces commentaires n'apparaîtront pas dans le programme final, ils servent seulement aux programmeurs.
- Normalement, on ne commente pas chaque ligne du programme.
- Le mieux est de commenter plusieurs lignes à la fois, c'est-à-dire d'indiquer à quoi sert une série d'instructions histoire d'avoir une idée.

141

## Les commentaires

61

- Lors de la compilation, tous les commentaires seront ignorés.
- Ces commentaires n'apparaîtront pas dans le programme final, ils servent seulement aux programmeurs.
- Normalement, on ne commente pas chaque ligne du programme.
- Le mieux est de commenter plusieurs lignes à la fois, c'est-à-dire d'indiquer à quoi sert une série d'instructions histoire d'avoir une idée.

**Une citation tirée de chez IBM :**

Si après avoir lu uniquement les commentaires d'un programme vous n'en comprenez pas le fonctionnement, jetez le tout !

142

Université Abdelmalek Essaâdi  
École Normale Supérieure  
Tétouan



2023/2024

# Les Variables

1

## Une affaire de mémoire

63

un ordinateur a plusieurs types de mémoire !

Pourquoi un ordinateur aurait-il plusieurs types de mémoire ? Une seule mémoire aurait suffi, non ?

2

1

## Une affaire de mémoire

63

un ordinateur a plusieurs types de mémoire !

Pourquoi un ordinateur aurait-il plusieurs types de mémoire ? Une seule mémoire aurait suffi, non ?

- Non: le problème c'est qu'on a besoin d'avoir une mémoire à la fois **rapide** (pour récupérer une information très vite) et **importante** (pour stocker beaucoup de données).

3

## Une affaire de mémoire

63

un ordinateur a plusieurs types de mémoire !

Pourquoi un ordinateur aurait-il plusieurs types de mémoire ? Une seule mémoire aurait suffi, non ?

- Non: le problème c'est qu'on a besoin d'avoir une mémoire à la fois **rapide** (pour récupérer une information très vite) et **importante** (pour stocker beaucoup de données).
- jusqu'ici nous avons été incapables de créer une mémoire qui soit à la fois très rapide et importante.

4

## Une affaire de mémoire

63

un ordinateur a plusieurs types de mémoire !

Pourquoi un ordinateur aurait-il plusieurs types de mémoire ? Une seule mémoire aurait suffi, non ?

- Non: le problème c'est qu'on a besoin d'avoir une mémoire à la fois **rapide** (pour récupérer une information très vite) et **importante** (pour stocker beaucoup de données).
- jusqu'ici nous avons été incapables de créer une mémoire qui soit à la fois très rapide et importante.
- Plus exactement, la mémoire rapide coûte cher, on n'en fait donc qu'en petites quantités.

5

## Les différents types de mémoire

64

de la plus rapide à la plus lente :

6

## Les différents types de mémoire

64

de la plus rapide à la plus lente :

- **les registres** : une mémoire ultra-rapide située directement dans le processeur ;
- **la mémoire cache** : elle fait le lien entre les registres et la mémoire vive ;
- **la mémoire vive** : c'est la mémoire avec laquelle nous allons travailler le plus souvent ;
- **le disque dur** : c'est là qu'on enregistre les fichiers.

7

## Les différents types de mémoire

64

de la plus rapide à la plus lente :

- **les registres** : une mémoire ultra-rapide située directement dans le processeur ;
  - **la mémoire cache** : elle fait le lien entre les registres et la mémoire vive ;
  - **la mémoire vive** : c'est la mémoire avec laquelle nous allons travailler le plus souvent ;
  - **le disque dur** : c'est là qu'on enregistre les fichiers.
- 
- la mémoire la plus rapide était la plus petite, et la plus lente la plus grosse.

8

## Les différents types de mémoire

65

- en programmation, on va surtout travailler avec **la mémoire vive**.

9

## Les différents types de mémoire

65

- en programmation, on va surtout travailler avec **la mémoire vive**.
- On verra aussi comment lire et écrire sur le **disque dur**, pour lire et créer des fichiers.

10

## Les différents types de mémoire

65

- en programmation, on va surtout travailler avec **la mémoire vive**.
- On verra aussi comment lire et écrire sur le **disque dur**, pour lire et créer des fichiers.

Quant à la mémoire cache et aux registres, on n'y touchera pas du tout !

11

## Les différents types de mémoire

65

- en programmation, on va surtout travailler avec **la mémoire vive**.
- On verra aussi comment lire et écrire sur le **disque dur**, pour lire et créer des fichiers.

Quant à la mémoire cache et aux registres, on n'y touchera pas du tout !

C'est l'ordinateur qui s'en occupe.

12

## Les différents types de mémoire

66

- seul le disque dur retient tout le temps les informations qu'il contient.

13

## Les différents types de mémoire

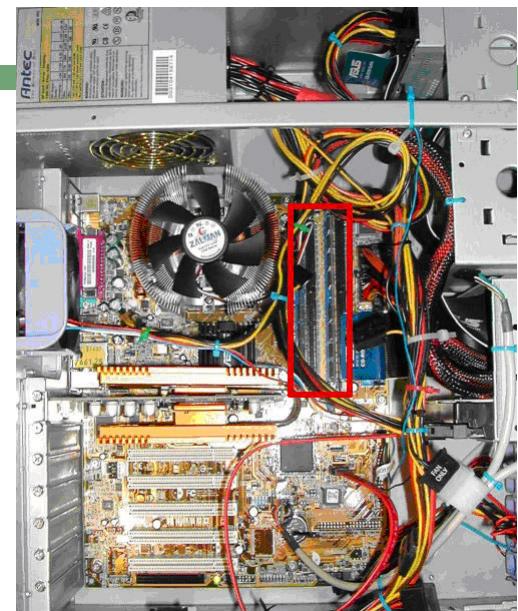
66

- seul le disque dur retient tout le temps les informations qu'il contient.
- toutes les autres mémoires (registres, mémoire cache, mémoire vive) sont des mémoires temporaires : **lorsque vous éteignez votre ordinateur, ces mémoires se vident !**

14

## La mémoire vive

67



15

## Le schéma de la mémoire vive

68

- un schéma du fonctionnement de la mémoire vive

16

## Le schéma de la mémoire vive

68

- un schéma du fonctionnement de la mémoire vive

Adresse	Valeur
0	145
1	3.8028322
2	0.827551
3	3901930
...	...
3 448 765 900 126 (et des poussières)	940.5118

17

## Le schéma de la mémoire vive

69

deux colonnes.

- les **adresses** :

18

## Le schéma de la mémoire vive

69

deux colonnes.

□ **les adresses :**

- une adresse est un nombre qui permet à l'ordinateur de se repérer dans la mémoire vive.

19

## Le schéma de la mémoire vive

69

deux colonnes.

□ **les adresses :**

- une adresse est un nombre qui permet à l'ordinateur de se repérer dans la mémoire vive.
- On commence à l'adresse 0 (au tout début de la mémoire) et on finit à l'adresse 3 448 765 900 126... en fait on ne connaît pas le nombre d'adresses qu'il y a dans la RAM, il y en a beaucoup.

20

## Le schéma de la mémoire vive

69

deux colonnes.

□ **les adresses :**

- une adresse est un nombre qui permet à l'ordinateur de se repérer dans la mémoire vive.
- On commence à l'adresse 0 (au tout début de la mémoire) et on finit à l'adresse 3 448 765 900 126... en fait on ne connaît pas le nombre d'adresses qu'il y a dans la RAM, il y en a beaucoup.
- En plus ça dépend de la quantité de mémoire vive que vous avez. Plus vous avez de mémoire vive, plus il y a d'adresses, donc plus on peut stocker de choses.

21

## Le schéma de la mémoire vive

69

deux colonnes.

□ **les adresses :**

- une adresse est un nombre qui permet à l'ordinateur de se repérer dans la mémoire vive.
- On commence à l'adresse 0 (au tout début de la mémoire) et on finit à l'adresse 3 448 765 900 126... en fait on ne connaît pas le nombre d'adresses qu'il y a dans la RAM, il y en a beaucoup.
- En plus ça dépend de la quantité de mémoire vive que vous avez. Plus vous avez de mémoire vive, plus il y a d'adresses, donc plus on peut stocker de choses.
- À chaque adresse, on peut stocker une **valeur** (un nombre) : l'ordinateur stocke dans la mémoire vive ces nombres pour pouvoir s'en souvenir par la suite.

22

## Le schéma de la mémoire vive

69

deux colonnes.

□ **les adresses :**

- une adresse est un nombre qui permet à l'ordinateur de se repérer dans la mémoire vive.
- On commence à l'adresse 0 (au tout début de la mémoire) et on finit à l'adresse 3 448 765 900 126... en fait on ne connaît pas le nombre d'adresses qu'il y a dans la RAM, il y en a beaucoup.
- En plus ça dépend de la quantité de mémoire vive que vous avez. Plus vous avez de mémoire vive, plus il y a d'adresses, donc plus on peut stocker de choses.
- À chaque adresse, on peut stocker une **valeur** (un nombre) : l'ordinateur stocke dans la mémoire vive ces nombres pour pouvoir s'en souvenir par la suite.
- On ne peut stocker qu'un nombre par adresse !

23

## Le schéma de la mémoire vive

69

deux colonnes.

**RAM ne peut stocker que des nombres.**

□ **les adresses :**

- une adresse est un nombre qui permet à l'ordinateur de se repérer dans la mémoire vive.
- On commence à l'adresse 0 (au tout début de la mémoire) et on finit à l'adresse 3 448 765 900 126... en fait on ne connaît pas le nombre d'adresses qu'il y a dans la RAM, il y en a beaucoup.
- En plus ça dépend de la quantité de mémoire vive que vous avez. Plus vous avez de mémoire vive, plus il y a d'adresses, donc plus on peut stocker de choses.
- À chaque adresse, on peut stocker une **valeur** (un nombre) : l'ordinateur stocke dans la mémoire vive ces nombres pour pouvoir s'en souvenir par la suite.
- On ne peut stocker qu'un nombre par adresse !

24

## Le schéma de la mémoire vive

70

- si l'ordinateur veut retenir le nombre 5 (qui pourrait être le nombre de vies qu'il reste au personnage d'un jeu), il le met quelque part en mémoire où il y a de la place et note l'adresse correspondante (par exemple **3 062 199 902**).
- Plus tard, lorsqu'il veut savoir à nouveau quel est ce nombre, il va chercher à la « case » mémoire n° **3 062 199 902** ce qu'il y a, et il trouve la valeur... 5 !

25

## Déclarer une variable

71

une variable, c'est quoi ?

26

## Déclarer une variable

71

une variable, c'est quoi ?

- c'est une petite information temporaire qu'on stocke dans la RAM.

27

## Déclarer une variable

71

une variable, c'est quoi ?

- c'est une petite information temporaire qu'on stocke dans la RAM.
- On dit qu'elle est « variable » car c'est une valeur qui peut changer pendant le déroulement du programme.

28

## Déclarer une variable

71

une variable, c'est quoi ?

- c'est une petite information temporaire qu'on stocke dans la RAM.
- On dit qu'elle est « variable » car c'est une valeur qui peut changer pendant le déroulement du programme.
- Par exemple, notre nombre 5 de tout à l'heure (le nombre de vies restant au joueur) risque de diminuer au fil du temps. Si ce nombre atteint 0, on saura que le joueur a perdu.

29

## Déclarer une variable

72

- En langage C, une variable est constituée de deux choses :

30

## Déclarer une variable

72

- En langage C, une variable est constituée de deux choses :
  - ▣ **une valeur** : c'est le nombre qu'elle stocke, par exemple 5 ;
  - ▣ **un nom** : c'est ce qui permet de la reconnaître.

31

## Déclarer une variable

72

- En langage C, une variable est constituée de deux choses :
  - ▣ **une valeur** : c'est le nombre qu'elle stocke, par exemple 5 ;
  - ▣ **un nom** : c'est ce qui permet de la reconnaître.
- En programmant en C, on n'aura pas à retenir l'adresse mémoire : à la place, on va juste indiquer des noms de variables. C'est le compilateur qui fera la conversion entre le nom et l'adresse.

32

## Donner un nom à ses variables

73

- chaque variable doit donc avoir un nom.

33

## Donner un nom à ses variables

73

- chaque variable doit donc avoir un nom.
- pour la variable qui retient le nombre de vies, on aimerait bien l'appeler « **Nombre de vies** » ou quelque chose du genre.

34

## Donner un nom à ses variables

73

- chaque variable doit donc avoir un nom.
- pour la variable qui retient le nombre de vies, on aimerait bien l'appeler « **Nombre de vies** » ou quelque chose du genre.
- il y a quelques contraintes.** Vous ne pouvez pas appeler une variable n'importe comment :

35

## Donner un nom à ses variables

73

- chaque variable doit donc avoir un nom.
- pour la variable qui retient le nombre de vies, on aimerait bien l'appeler « **Nombre de vies** » ou quelque chose du genre.
- il y a quelques contraintes.** Vous ne pouvez pas appeler une variable n'importe comment :
  - il ne peut y avoir que des minuscules, majuscules et des chiffres (**abcABC012**) ;

36

## Donner un nom à ses variables

73

- chaque variable doit donc avoir un nom.
- pour la variable qui retient le nombre de vies, on aimerait bien l'appeler « **Nombre de vies** » ou quelque chose du genre.
- **il y a quelques contraintes.** Vous ne pouvez pas appeler une variable n'importe comment :
  - il ne peut y avoir que des minuscules, majuscules et des chiffres (**abcABC012**) ;
  - votre nom de variable doit commencer par une lettre ;

37

## Donner un nom à ses variables

73

- chaque variable doit donc avoir un nom.
- pour la variable qui retient le nombre de vies, on aimerait bien l'appeler « **Nombre de vies** » ou quelque chose du genre.
- **il y a quelques contraintes.** Vous ne pouvez pas appeler une variable n'importe comment :
  - il ne peut y avoir que des minuscules, majuscules et des chiffres (**abcABC012**) ;
  - votre nom de variable doit commencer par une lettre ;
  - les espaces sont interdits. À la place, on peut utiliser le caractère « **underscore** » **\_**. C'est le seul caractère différent des lettres et chiffres autorisé ;

38

## Donner un nom à ses variables

73

- chaque variable doit donc avoir un nom.
- pour la variable qui retient le nombre de vies, on aimerait bien l'appeler « **Nombre de vies** » ou quelque chose du genre.
- **il y a quelques contraintes.** Vous ne pouvez pas appeler une variable n'importe comment :
  - il ne peut y avoir que des minuscules, majuscules et des chiffres (**abcABC012**) ;
  - votre nom de variable doit commencer par une lettre ;
  - les espaces sont interdits. À la place, on peut utiliser le caractère « underscore » **\_**. C'est le seul caractère différent des lettres et chiffres autorisé ;
  - vous n'avez pas le droit d'utiliser des accents (**éàê** etc.).

39

## Donner un nom à ses variables

74

- le langage C fait la différence entre les majuscules et les minuscules.

40

## Donner un nom à ses variables

74

- le langage C fait la différence entre les majuscules et les minuscules.
  - on dit que c'est un langage qui « **respecte la casse** ».

41

## Donner un nom à ses variables

74

- le langage C fait la différence entre les majuscules et les minuscules.
  - on dit que c'est un langage qui « **respecte la casse** ».
- les variables **largeur**, **LARGEUR** ou encore **LArgEuR** sont trois variables différentes

42

## Donner un nom à ses variables

74

- le langage C fait la différence entre les majuscules et les minuscules.
  - on dit que c'est un langage qui « **respecte la casse** ».
- les variables **largeur**, **LARGEUR** ou encore **LArgEuR** sont trois variables différentes
- noms de variables corrects : **nombreDeVies**, **prenom**, **nom**,  
**numero\_de\_telephone**, **numeroDeTelephone**.

43

## Les types de variables

75

- L'ordinateur ne sait traiter que des nombres.

44

## Les types de variables

75

- L'ordinateur ne sait traiter que des nombres.
- Lorsque vous créez une variable, vous allez donc devoir **indiquer son type**.

45

## Les types de variables

75

- L'ordinateur ne sait traiter que des nombres.
- Lorsque vous créez une variable, vous allez donc devoir **indiquer son type**.
- les principaux types de variables existant en langage C :

46

## Les types de variables

75

- L'ordinateur ne sait traiter que des nombres.
- Lorsque vous créez une variable, vous allez donc devoir **indiquer son type**.
- les principaux types de variables existant en langage C :

Nom du type	Minimum	Maximum
signed char	-127	127
int	-32 767	32 767
long	-2 147 483 647	2 147 483 647
float	-1 x10 <sup>37</sup>	1 x10 <sup>37</sup>
double	-1 x10 <sup>37</sup>	1 x10 <sup>37</sup>

47

## Les types de variables

76

- Les trois premiers types (**signed char, int, long**) permettent de stocker des nombres entiers : 1, 2, 3, 4...

48

## Les types de variables

76

- Les trois premiers types (**signed char, int, long**) permettent de stocker des nombres entiers : 1, 2, 3, 4...
- Les deux derniers (**float, double**) permettent de stocker des nombres décimaux (appelés nombres flottants) : 13.8, 16.911...

49

## Les types de variables

76

- Les trois premiers types (**signed char, int, long**) permettent de stocker des nombres entiers : 1, 2, 3, 4...
- Les deux derniers (**float, double**) permettent de stocker des nombres décimaux (appelés nombres flottants) : 13.8, 16.911...
- **Attention:** avec les nombres décimaux l'ordinateur ne connaît pas la virgule, il utilise le point. Vous ne devez donc pas écrire 54,9 mais plutôt 54.9 !

50

## Les types de variables

76

- Les trois premiers types (**signed char, int, long**) permettent de stocker des nombres entiers : 1, 2, 3, 4...
- Les deux derniers (**float, double**) permettent de stocker des nombres décimaux (appelés nombres flottants) : 13.8, 16.911...
- **Attention:** avec les nombres décimaux l'ordinateur ne connaît pas la virgule, il utilise le point. Vous ne devez donc pas écrire 54,9 mais plutôt 54.9 !
- Pour les types entiers (signed char, int, long...), il existe d'autres types dits **unsigned** (non signés) qui eux ne peuvent stocker que des **nombre positifs**. Pour les utiliser, il suffit d'écrire le mot **unsigned** devant le type :

51

## Les types de variables

76

- Les trois premiers types (**signed char, int, long**) permettent de stocker des nombres entiers : 1, 2, 3, 4...
- Les deux derniers (**float, double**) permettent de stocker des nombres décimaux (appelés nombres flottants) : 13.8, 16.911...
- **Attention:** avec les nombres décimaux l'ordinateur ne connaît pas la virgule, il utilise le point. Vous ne devez donc pas écrire 54,9 mais plutôt 54.9 !
- Pour les types entiers (signed char, int, long...), il existe d'autres types dits **unsigned** (non signés) qui eux ne peuvent stocker que des **nombre positifs**. Pour les utiliser, il suffit d'écrire le mot **unsigned** devant le type :
  - **unsigned char** 0 à 255

52

## Les types de variables

76

- Les trois premiers types (**signed char, int, long**) permettent de stocker des nombres entiers : 1, 2, 3, 4...
- Les deux derniers (**float, double**) permettent de stocker des nombres décimaux (appelés nombres flottants) : 13.8, 16.911...
- **Attention:** avec les nombres décimaux l'ordinateur ne connaît pas la virgule, il utilise le point. Vous ne devez donc pas écrire 54,9 mais plutôt 54.9 !
- Pour les types entiers (signed char, int, long...), il existe d'autres types dits **unsigned** (non signés) qui eux ne peuvent stocker que des **nombres positifs**. Pour les utiliser, il suffit d'écrire le mot **unsigned** devant le type :
  - unsigned char 0 à 255
  - unsigned int 0 à 65 535

53

## Les types de variables

76

- Les trois premiers types (**signed char, int, long**) permettent de stocker des nombres entiers : 1, 2, 3, 4...
- Les deux derniers (**float, double**) permettent de stocker des nombres décimaux (appelés nombres flottants) : 13.8, 16.911...
- **Attention:** avec les nombres décimaux l'ordinateur ne connaît pas la virgule, il utilise le point. Vous ne devez donc pas écrire 54,9 mais plutôt 54.9 !
- Pour les types entiers (signed char, int, long...), il existe d'autres types dits **unsigned** (non signés) qui eux ne peuvent stocker que des **nombres positifs**. Pour les utiliser, il suffit d'écrire le mot **unsigned** devant le type :
  - unsigned char 0 à 255
  - unsigned int 0 à 65 535
  - unsigned long 0 à 4 294 967 295

54

## Les types de variables

77

- les **unsigned** sont des types qui ont le défaut de ne pas pouvoir stocker de nombres négatifs, mais l'avantage de pouvoir stocker des nombres deux fois plus grands

55

## Les types de variables

77

- les **unsigned** sont des types qui ont le défaut de ne pas pouvoir stocker de nombres négatifs, mais l'avantage de pouvoir stocker des nombres deux fois plus grands
- signed char s'arrête à 127, tandis que unsigned char s'arrête à 255 par exemple.

56

## Les types de variables

78

Pourquoi avoir créé trois types pour les nombres entiers ? Un seul type aurait été suffisant, non ?

57

## Les types de variables

78

Pourquoi avoir créé trois types pour les nombres entiers ? Un seul type aurait été suffisant, non ?

- on a créé à l'origine plusieurs types pour **économiser de la mémoire**.

58

## Les types de variables

78

Pourquoi avoir créé trois types pour les nombres entiers ? Un seul type aurait été suffisant, non ?

- on a créé à l'origine plusieurs types pour **économiser de la mémoire**.
  - quand on dit à l'ordinateur qu'on a besoin d'une variable de type **char**, on prend **moins d'espace en mémoire** que si on avait demandé une variable de type **int**.

59

## Déclarer une variable

79

- créez un nouveau projet console que vous appellerez « **variables** ».

60

## Déclarer une variable

79

- créez un nouveau projet console que vous appellerez « **variables** ».
- déclarer une variable, c'est-à-dire demander à l'ordinateur la permission d'utiliser un peu de mémoire.

61

## Déclarer une variable

79

- créez un nouveau projet console que vous appellerez « **variables** ».
- déclarer une variable, c'est-à-dire demander à l'ordinateur la permission d'utiliser un peu de mémoire.
- Une déclaration de variable, c'est très simple maintenant que vous savez tout ce qu'il faut. Il suffit dans l'ordre :

62

## Déclarer une variable

79

- créez un nouveau projet console que vous appellerez « **variables** ».
- déclarer une variable, c'est-à-dire demander à l'ordinateur la permission d'utiliser un peu de mémoire.
- Une déclaration de variable, c'est très simple maintenant que vous savez tout ce qu'il faut. Il suffit dans l'ordre :
  - d'indiquer le **type** de la variable que l'on veut créer ;

63

## Déclarer une variable

79

- créez un nouveau projet console que vous appellerez « **variables** ».
- déclarer une variable, c'est-à-dire demander à l'ordinateur la permission d'utiliser un peu de mémoire.
- Une déclaration de variable, c'est très simple maintenant que vous savez tout ce qu'il faut. Il suffit dans l'ordre :
  - d'indiquer le **type** de la variable que l'on veut créer ;
  - d'insérer un **espace** ;

64

## Déclarer une variable

79

- créez un nouveau projet console que vous appellerez « **variables** ».
- déclarer une variable, c'est-à-dire demander à l'ordinateur la permission d'utiliser un peu de mémoire.
- Une déclaration de variable, c'est très simple maintenant que vous savez tout ce qu'il faut. Il suffit dans l'ordre :
  - d'indiquer le **type** de la variable que l'on veut créer ;
  - d'insérer un **espace** ;
  - d'indiquer le **nom** que vous voulez donner à la variable ;

65

## Déclarer une variable

79

- créez un nouveau projet console que vous appellerez « **variables** ».
- déclarer une variable, c'est-à-dire demander à l'ordinateur la permission d'utiliser un peu de mémoire.
- Une déclaration de variable, c'est très simple maintenant que vous savez tout ce qu'il faut. Il suffit dans l'ordre :
  - d'indiquer le **type** de la variable que l'on veut créer ;
  - d'insérer un **espace** ;
  - d'indiquer le **nom** que vous voulez donner à la variable ;
  - et enfin, de ne pas oublier le **point-virgule**.

66

## Déclarer une variable

79

- créez un nouveau projet console que vous appellerez « **variables** ».
- déclarer une variable, c'est-à-dire demander à l'ordinateur la permission d'utiliser un peu de mémoire.
- Une déclaration de variable, c'est très simple maintenant que vous savez tout ce qu'il faut. Il suffit dans l'ordre :
  - d'indiquer le **type** de la variable que l'on veut créer ;
  - d'insérer un **espace** ;
  - d'indiquer le **nom** que vous voulez donner à la variable ;
  - et enfin, de ne pas oublier le **point-virgule**.

```
int nombreDeVies;
```

67

## Déclarer une variable

80

- autres exemples

```
int noteDeMaths;
double sommeArgentRecue;
unsigned int nombreDeLecteursEnTrainDeLireUnNomDeVariableUnPeuLong;
```

68

## Déclarer une variable

80

- autres exemples

```
int noteDeMaths;
double sommeArgentRecue;
unsigned int nombreDeLecteursEnTrainDeLireUnNomDeVariableUnPeuLong;
```

- faire les déclarations de variables au début des fonctions

69

## Déclarer une variable

80

- autres exemples

```
int noteDeMaths;
double sommeArgentRecue;
unsigned int nombreDeLecteursEnTrainDeLireUnNomDeVariableUnPeuLong;
```

- faire les déclarations de variables au début des fonctions

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) // Équivalent de int main()
{
    int nombreDeVies;
    return 0;
}
```

70

## Déclarer une variable

81

- plusieurs variables du **même type** à déclarer:

71

## Déclarer une variable

81

- plusieurs variables du **même type** à déclarer:

- ▣ inutile de faire une ligne pour chaque variable.

72

## Déclarer une variable

81

- plusieurs variables du **même type** à déclarer:
  - inutile de faire une ligne pour chaque variable.
  - Il suffit de séparer les différents noms de variables par des **virgules** sur la même ligne : **int nombreDeVies, niveau, ageDuJoueur;**

73

## Affecter une valeur à une variable

82

- donner une valeur à la variable **nombreDeVies**:

74

## Affecter une valeur à une variable

82

- donner une valeur à la variable **nombreDeVies**: `nombreDeVies = 5;`

75

## Affecter une valeur à une variable

82

- donner une valeur à la variable **nombreDeVies**: `nombreDeVies = 5;`

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

int main(int argc, char *argv[])
{
    int nombreDeVies;
    nombreDeVies = 5;
    return 0;
}
```

76

## Affecter une valeur à une variable

82

- donner une valeur à la variable **nombreDeVies**: `nombreDeVies = 5;`

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

int main(int argc, char *argv[])
{
    int nombreDeVies;
    nombreDeVies = 5;
    return 0;
}
```

- une petite case de mémoire vient de prendre la valeur 5.

77

## Affecter une valeur à une variable

83

- changer la valeur

78

## Affecter une valeur à une variable

83

- changer la valeur

```
int nombreDeVies;  
nombreDeVies = 5;  
nombreDeVies = 4;  
nombreDeVies = 3;
```

79

## La valeur d'une nouvelle variable

84

Quand on déclare une variable, quelle valeur a-t-elle au départ ?  
Y a-t-il une valeur par défaut (par exemple 0) ?

80

## La valeur d'une nouvelle variable

84

Quand on déclare une variable, quelle valeur a-t-elle au départ ?

Y a-t-il une valeur par défaut (par exemple 0) ?

- Non, non et non, il n'y a pas de valeur par défaut.

81

## La valeur d'une nouvelle variable

84

Quand on déclare une variable, quelle valeur a-t-elle au départ ?

Y a-t-il une valeur par défaut (par exemple 0) ?

- Non, non et non, il n'y a pas de valeur par défaut.
- l'emplacement est réservé mais la valeur ne change pas.

82

## La valeur d'une nouvelle variable

84

Quand on déclare une variable, quelle valeur a-t-elle au départ ?

Y a-t-il une valeur par défaut (par exemple 0) ?

- Non, non et non, il n'y a pas de valeur par défaut.
- l'emplacement est réservé mais la valeur ne change pas.
- On n'efface pas ce qui se trouve dans la « case mémoire ».

83

## La valeur d'une nouvelle variable

84

Quand on déclare une variable, quelle valeur a-t-elle au départ ?

Y a-t-il une valeur par défaut (par exemple 0) ?

- Non, non et non, il n'y a pas de valeur par défaut.
- l'emplacement est réservé mais la valeur ne change pas.
- On n'efface pas ce qui se trouve dans la « case mémoire ».
- La variable prend la valeur qui se trouvait là avant dans la mémoire, et cette valeur peut être n'importe quoi !

84

## La valeur d'une nouvelle variable

85

- Si cette zone de la mémoire n'a jamais été modifiée, la valeur est peut-être 0.

85

## La valeur d'une nouvelle variable

85

- Si cette zone de la mémoire n'a jamais été modifiée, la valeur est peut-être 0.
- Mais vous n'en êtes pas sûrs, il pourrait très bien y avoir le nombre 363 ou 18 à la place, c'est-à-dire un reste d'un vieux programme qui est passé par là avant !

86

## La valeur d'une nouvelle variable

85

- Si cette zone de la mémoire n'a jamais été modifiée, la valeur est peut-être 0.
- Mais vous n'en êtes pas sûrs, il pourrait très bien y avoir le nombre 363 ou 18 à la place, c'est-à-dire un reste d'un vieux programme qui est passé par là avant !
- Il faut donc faire très attention à ça si on veut éviter des problèmes par la suite.

87

## La valeur d'une nouvelle variable

85

- Si cette zone de la mémoire n'a jamais été modifiée, la valeur est peut-être 0.
- Mais vous n'en êtes pas sûrs, il pourrait très bien y avoir le nombre 363 ou 18 à la place, c'est-à-dire un reste d'un vieux programme qui est passé par là avant !
- Il faut donc faire très attention à ça si on veut éviter des problèmes par la suite.
  - Le mieux est d'initialiser la variable dès qu'on la déclare.
  - combiner la déclaration et l'affectation d'une variable dans la même instruction :

```
int nombreDeVies = 5;
```

- L'avantage, c'est que vous êtes sûrs après que cette variable contient une valeur correcte, et pas du n'importe quoi.

88

## Les constantes

86

- on ait besoin d'utiliser une variable dont on voudrait qu'elle garde la même valeur pendant toute la durée du programme.

89

## Les constantes

86

- on ait besoin d'utiliser une variable dont on voudrait qu'elle garde la même valeur pendant toute la durée du programme.
- La variable conserve sa valeur et que personne n'ait le droit de changer ce qu'elle contient.

90

## Les constantes

86

- on ait besoin d'utiliser une variable dont on voudrait qu'elle garde la même valeur pendant toute la durée du programme.
  - La variable conserve sa valeur et que personne n'ait le droit de changer ce qu'elle contient.
  - Ces variables particulières sont appelées constantes

91

## Les constantes

86

- on ait besoin d'utiliser une variable dont on voudrait qu'elle garde la même valeur pendant toute la durée du programme.
  - La variable conserve sa valeur et que personne n'ait le droit de changer ce qu'elle contient.
  - Ces variables particulières sont appelées constantes
- Pour déclarer une constante: il faut utiliser le mot **const** juste devant le type quand vous déclarez votre variable.

92

## Les constantes

86

- on ait besoin d'utiliser une variable dont on voudrait qu'elle garde la même valeur pendant toute la durée du programme.
- La variable conserve sa valeur et que personne n'ait le droit de changer ce qu'elle contient.
- Ces variables particulières sont appelées constantes
- Pour déclarer une constante: il faut utiliser le mot **const** juste devant le type quand vous déclarez votre variable.
- il faut obligatoirement lui donner une valeur au moment de sa déclaration

93

## Les constantes

86

- on ait besoin d'utiliser une variable dont on voudrait qu'elle garde la même valeur pendant toute la durée du programme.
- La variable conserve sa valeur et que personne n'ait le droit de changer ce qu'elle contient.
- Ces variables particulières sont appelées constantes
- Pour déclarer une constante: il faut utiliser le mot **const** juste devant le type quand vous déclarez votre variable.
- il faut obligatoirement lui donner une valeur au moment de sa déclaration

```
const int NOMBRE_DE_VIES_INITIALES = 5;
```

94

## Les constantes

86

- on ait besoin d'utiliser une variable dont on voudrait qu'elle garde la même valeur pendant toute la durée du programme.
  - • par convention on écrit les noms des constantes entièrement en majuscules.
  - • permet de distinguer facilement les constantes des variables.
- Pour déclarer une constante: il faut utiliser le mot **const** juste devant le type quand vous déclarez votre variable.
- il faut obligatoirement lui donner une valeur au moment de sa déclaration

```
const int NOMBRE_DE_VIES_INITIALES = 5;
```

95

## Afficher le contenu d'une variable

87

- la fonction **printf**

96

## Afficher le contenu d'une variable

87

- la fonction **printf**

```
printf("Il vous reste %d vies");
```

97

## Afficher le contenu d'une variable

87

- la fonction **printf**

```
printf("Il vous reste %d vies");
```

- un symbole spécial à l'endroit où l'on veut afficher la valeur de la variable:

98

## Afficher le contenu d'une variable

87

- la fonction **printf**

```
printf("Il vous reste %d vies");
```

- un symbole spécial à l'endroit où l'on veut afficher la valeur de la variable:
  - **%** suivi d'une **lettre**

99

## Afficher le contenu d'une variable

87

- la fonction **printf**

```
printf("Il vous reste %d vies");
```

- un symbole spécial à l'endroit où l'on veut afficher la valeur de la variable:
  - **%** suivi d'une **lettre**
  - cette **lettre** permet d'indiquer ce que l'on doit afficher.

100

## Afficher le contenu d'une variable

87

- la fonction **printf**

```
printf("Il vous reste %d vies");
```

- un symbole spécial à l'endroit où l'on veut afficher la valeur de la variable:
  - **%** suivi d'une **lettre**
  - cette **lettre** permet d'indiquer ce que l'on doit afficher.
  - **'d'** signifie que l'on souhaite afficher un **int**.

101

## Afficher le contenu d'une variable

88

- Il existe plusieurs autres possibilités:

102

## Afficher le contenu d'une variable

88

- Il existe plusieurs autres possibilités:

Format	Type attendu
"%d"	int
"%ld"	long
"%f"	float
"%lf"	double

103

## Afficher le contenu d'une variable

89

- On a indiqué qu'à un endroit précis on voulait afficher un nombre entier, **mais on n'a pas précisé lequel !**

104

## Afficher le contenu d'une variable

89

- On a indiqué qu'à un endroit précis on voulait afficher un nombre entier, **mais on n'a pas précisé lequel !**
  - Il faut indiquer à la fonction **printf** quelle est la variable dont on veut afficher la valeur.

105

## Afficher le contenu d'une variable

89

- On a indiqué qu'à un endroit précis on voulait afficher un nombre entier, **mais on n'a pas précisé lequel !**
  - Il faut indiquer à la fonction **printf** quelle est la variable dont on veut afficher la valeur.
  - taper le nom de la variable après les guillemets et après avoir rajouté une virgule

106

## Afficher le contenu d'une variable

89

- On a indiqué qu'à un endroit précis on voulait afficher un nombre entier, **mais on n'a pas précisé lequel !**
- Il faut indiquer à la fonction **printf** quelle est la variable dont on veut afficher la valeur.
- taper le nom de la variable après les guillemets et après avoir rajouté une virgule

```
printf("Il vous reste %d vies", nombreDeVies);
```

107

## Afficher le contenu d'une variable

89

- On a indiqué qu'à un endroit précis on voulait afficher un nombre entier, **mais on n'a pas précisé lequel !**
  - Il faut indiquer à la fonction **printf** quelle est la variable dont on veut afficher la valeur.
  - taper le nom de la variable après les guillemets et après avoir rajouté une virgule
- ```
printf("Il vous reste %d vies", nombreDeVies);
```
- Le **%d** sera remplacé par la variable indiquée après la virgule

108

## Afficher le contenu d'une variable

90

Test...

109

## Afficher le contenu d'une variable

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int nombreDeVies = 5; // Au départ, le joueur a 5 vies
    printf("Vous avez %d vies\n", nombreDeVies);
    printf("**** B A M ****\n"); // Là il se prend un grand coup sur la tête
    nombreDeVies = 4; // Il vient de perdre une vie !
    printf("Ah desole, il ne vous reste plus que %d vies maintenant !\n\n",
    nombreDeVies);
    return 0;
}
```

110

## Afficher plusieurs variables dans un même printf

91

- il suffit d'indiquer:

111

## Afficher plusieurs variables dans un même printf

91

- il suffit d'indiquer:
  - des %d ou des %f là où vous voulez.

112

## Afficher plusieurs variables dans un même printf

91

- il suffit d'indiquer:
  - ▣ des %d ou des %f là où vous voulez.
  - ▣ les variables correspondantes dans le même ordre, séparées par des virgules.

113

## Afficher plusieurs variables dans un même printf

91

- il suffit d'indiquer:
  - ▣ des %d ou des %f là où vous voulez.
  - ▣ les variables correspondantes dans le même ordre, séparées par des virgules.

```
printf("Vous avez %d vies et vous etes au niveau n° %d", nombreDeVies, niveau);
```

114

## Afficher plusieurs variables dans un même printf

91

□ il suffit d'indiquer:

- des %d ou des %f là où vous voulez.
- les variables correspondantes dans le même ordre, séparées par des virgules.

```
printf("Vous avez %d vies et vous êtes au niveau n° %d", nombreDeVies, niveau);
```

■ Test...

115

## Afficher plusieurs variables dans un même printf

91

□ il suffit d'indiquer:

- des %d ou des %f là où vous voulez.
- les variables correspondantes dans le même ordre, séparées par des virgules.

```
printf("Vous avez %d vies et vous êtes au niveau n° %d", nombreDeVies, niveau);
```

■ Test...

```
int main(int argc, char *argv[])
{
    int nombreDeVies = 5, niveau = 1;

    printf("Vous avez %d vies et vous êtes au niveau n° %d\n", nombreDeVies, niveau);
    return 0;
}
```

116

## Récupérer une saisie

92

- demander à l'utilisateur de taper un nombre dans la console.

117

## Récupérer une saisie

92

- demander à l'utilisateur de taper un nombre dans la console.
- Ce nombre, on va le récupérer et le stocker dans une variable.

118

## Récupérer une saisie

92

- demander à l'utilisateur de taper un nombre dans la console.
- Ce nombre, on va le récupérer et le stocker dans une variable.
- Pour demander à l'utilisateur d'entrer quelque chose dans la console, on va utiliser une autre fonction toute prête : **scanf**.

119

## Récupérer une saisie

92

- demander à l'utilisateur de taper un nombre dans la console.
- Ce nombre, on va le récupérer et le stocker dans une variable.
- Pour demander à l'utilisateur d'entrer quelque chose dans la console, on va utiliser une autre fonction toute prête : **scanf**.
- Vous devez

120

## Récupérer une saisie

92

- demander à l'utilisateur de taper un nombre dans la console.
- Ce nombre, on va le récupérer et le stocker dans une variable.
- Pour demander à l'utilisateur d'entrer quelque chose dans la console, on va utiliser une autre fonction toute prête : **scanf**.
- Vous devez
  - mettre un format pour indiquer ce que l'utilisateur doit entrer (un **int**, un **float**, ...).

121

## Récupérer une saisie

92

- demander à l'utilisateur de taper un nombre dans la console.
- Ce nombre, on va le récupérer et le stocker dans une variable.
- Pour demander à l'utilisateur d'entrer quelque chose dans la console, on va utiliser une autre fonction toute prête : **scanf**.
- Vous devez
  - mettre un format pour indiquer ce que l'utilisateur doit entrer (un **int**, un **float**, ...).
  - indiquer le nom de la variable qui va recevoir le nombre.

122

## Récupérer une saisie

92

- demander à l'utilisateur de taper un nombre dans la console.
- Ce nombre, on va le récupérer et le stocker dans une variable.
- Pour demander à l'utilisateur d'entrer quelque chose dans la console, on va utiliser une autre fonction toute prête : **scanf**.
- Vous devez
  - mettre un format pour indiquer ce que l'utilisateur doit entrer (un **int**, un **float**, ...).
  - indiquer le nom de la variable qui va recevoir le nombre.

```
int age = 0;
scanf("%d", &age);
```

123

## Récupérer une saisie

92

- demander à l'utilisateur de taper un nombre dans la console.
- Ce nombre, on va le récupérer et le stocker dans une variable.
- Pour demander à l'utilisateur d'entrer quelque chose dans la console, on va utiliser une autre fonction toute prête : **scanf**.
- Vous devez
  - mettre un format pour indiquer ce que l'utilisateur doit entrer (un **int**, un **float**, ...).
  - indiquer le nom de la variable qui va recevoir le nombre.
- il faut mettre le symbole **&** devant le nom de la variable qui va recevoir la valeur.

124

## Récupérer une saisie

92

- demander à l'utilisateur de taper un nombre dans la console.
  - Ce nombre, on va le récupérer et le stocker dans une variable.
  - Pour demander à l'utilisateur de taper un nombre, on va utiliser une fonction :
    - Lorsque le programme arrive à un **scanf**, il se met en pause et attend que l'utilisateur entre un nombre.
    - Ce nombre sera stocké dans la variable age.
  - Vous devez :
    - mettre un format pour indiquer ce que l'utilisateur doit entrer (un **int**, un **float**, ...).
    - indiquer le nom de la variable qui va recevoir le nombre.
- ```
int age = 0;
scanf("%d", &age);
```
- il faut mettre le symbole **&** devant le nom de la variable qui va recevoir la valeur.

125

## Récupérer une saisie

93

une petite divergence de format entre printf et scanf !

```
double poids = 0;
scanf("%lf", &poids);
```

126

## Récupérer une saisie

93

une petite divergence de format entre printf et scanf !

- pour récupérer un float, c'est le format "%f" qu'il faut utiliser.

```
double poids = 0;  
scanf("%lf", &poids);
```

127

## Récupérer une saisie

93

une petite divergence de format entre printf et scanf !

- pour récupérer un float, c'est le format "%f" qu'il faut utiliser.
- mais pour le type double c'est le format "%lf".

```
double poids = 0;  
scanf("%lf", &poids);
```

128

## Récupérer une saisie

94

□ Test...

129

## Récupérer une saisie

94

□ Test...

```
int main(int argc, char *argv[])
{
    int age = 0; // On initialise la variable à 0

    printf("Quel age avez-vous ? ");
    scanf("%d", &age); // On demande d'entrer l'âge avec scanf
    printf("Ah ! Vous avez donc %d ans !\n\n", age);

    return 0;
}
```

130

## Récupérer une saisie

94

- Test...

```
int main(int argc, char *argv[])
{
    int age = 0; // On initialise la variable à 0

    printf("Quel age avez-vous ? ");
    scanf("%d", &age); // On demande d'entrer l'âge avec scanf
    printf("Ah ! Vous avez donc %d ans !\n\n", age);

    return 0;
}
```

- si vous rentrez un nombre décimal, comme 2.9

131

## Récupérer une saisie

94

- Test...

```
int main(int argc, char *argv[])
{
    int age = 0; // On initialise la variable à 0

    printf("Quel age avez-vous ? ");
    scanf("%d", &age); // On demande d'entrer l'âge avec scanf
    printf("Ah ! Vous avez donc %d ans !\n\n", age);

    return 0;
}
```

- si vous rentrez un nombre décimal, comme 2.9
- si vous tapez des lettres au hasard (« éèydf »)

132

## En résumé

95

- Nos ordinateurs possèdent plusieurs types de mémoire. De la plus rapide à la plus lente : les registres, la mémoire cache, la mémoire vive et le disque dur.

133

## En résumé

95

- Nos ordinateurs possèdent plusieurs types de mémoire. De la plus rapide à la plus lente : les registres, la mémoire cache, la mémoire vive et le disque dur.
- Pour « retenir » des informations, notre programme a besoin de stocker des données dans la mémoire. Il utilise pour cela la mémoire vive. Les registres et la mémoire cache sont aussi utilisés pour augmenter les performances, mais cela fonctionne automatiquement, nous n'avons pas à nous en préoccuper.

134

## En résumé

95

- Nos ordinateurs possèdent plusieurs types de mémoire. De la plus rapide à la plus lente : les registres, la mémoire cache, la mémoire vive et le disque dur.
- Pour « retenir » des informations, notre programme a besoin de stocker des données dans la mémoire. Il utilise pour cela la mémoire vive. Les registres et la mémoire cache sont aussi utilisés pour augmenter les performances, mais cela fonctionne automatiquement, nous n'avons pas à nous en préoccuper.
- Dans notre code source, les variables sont des données stockées temporairement en mémoire vive. La valeur de ces données peut changer au cours du programme.

135

## En résumé

95

- Nos ordinateurs possèdent plusieurs types de mémoire. De la plus rapide à la plus lente : les registres, la mémoire cache, la mémoire vive et le disque dur.
- Pour « retenir » des informations, notre programme a besoin de stocker des données dans la mémoire. Il utilise pour cela la mémoire vive. Les registres et la mémoire cache sont aussi utilisés pour augmenter les performances, mais cela fonctionne automatiquement, nous n'avons pas à nous en préoccuper.
- Dans notre code source, les variables sont des données stockées temporairement en mémoire vive. La valeur de ces données peut changer au cours du programme.
- À l'opposé, on parle de constantes pour des données stockées en mémoire vive. La valeur de ces données ne peut pas changer.

136

## En résumé

95

- Nos ordinateurs possèdent plusieurs types de mémoire. De la plus rapide à la plus lente : les registres, la mémoire cache, la mémoire vive et le disque dur.
- Pour « retenir » des informations, notre programme a besoin de stocker des données dans la mémoire. Il utilise pour cela la mémoire vive. Les registres et la mémoire cache sont aussi utilisés pour augmenter les performances, mais cela fonctionne automatiquement, nous n'avons pas à nous en préoccuper.
- Dans notre code source, les variables sont des données stockées temporairement en mémoire vive. La valeur de ces données peut changer au cours du programme.
- À l'opposé, on parle de constantes pour des données stockées en mémoire vive. La valeur de ces données ne peut pas changer.
- Il existe plusieurs types de variables, qui occupent plus ou moins d'espace en mémoire. Certains types comme int sont prévus pour stocker des nombres entiers, tandis que d'autres comme double stockent des nombres décimaux.

137

## En résumé

95

- Nos ordinateurs possèdent plusieurs types de mémoire. De la plus rapide à la plus lente : les registres, la mémoire cache, la mémoire vive et le disque dur.
- Pour « retenir » des informations, notre programme a besoin de stocker des données dans la mémoire. Il utilise pour cela la mémoire vive. Les registres et la mémoire cache sont aussi utilisés pour augmenter les performances, mais cela fonctionne automatiquement, nous n'avons pas à nous en préoccuper.
- Dans notre code source, les variables sont des données stockées temporairement en mémoire vive. La valeur de ces données peut changer au cours du programme.
- À l'opposé, on parle de constantes pour des données stockées en mémoire vive. La valeur de ces données ne peut pas changer.
- Il existe plusieurs types de variables, qui occupent plus ou moins d'espace en mémoire. Certains types comme int sont prévus pour stocker des nombres entiers, tandis que d'autres comme double stockent des nombres décimaux.
- La fonction scanf permet de demander à l'utilisateur de saisir un nombre.

138

Université Abdelmalek Essaâdi  
École Normale Supérieure  
Tétouan



2023/2024

# Les opérateurs de calcul

1

## Les calculs de base

97

- on ne peut faire que des opérations très simples :

2

## Les calculs de base

97

- on ne peut faire que des opérations très simples :
  - ▣ addition ;
  - ▣ soustraction;
  - ▣ multiplication ;
  - ▣ division ;
  - ▣ Modulo

3

## Les calculs de base

97

- on ne peut faire que des opérations très simples :
  - ▣ addition ;
  - ▣ soustraction;
  - ▣ multiplication ;
  - ▣ division ;
  - ▣ Modulo
- Si on veut faire des opérations plus compliquées (des carrés, des puissances, des logarithmes et autres joyeusetés) il **faudra les programmer**, c'est-à-dire expliquer à l'ordinateur comment les faire.

4

## Les calculs de base

97

- on ne peut faire que des opérations très simples :
  - ▣ addition ;
  - ▣ soustraction;
  - ▣ multiplication ;
  - ▣ division ;
  - ▣ Modulo
- il existe une bibliothèque mathématique livrée avec le langage C qui contient des fonctions mathématiques toutes prêtes.
- Si on veut faire des opérations plus compliquées (des carrés, des puissances, des logarithmes et autres joyeusetés) **il faudra les programmer**, c'est-à-dire expliquer à l'ordinateur comment les faire.

5

## Les calculs de base

98

- Exemple: addition

6

## Les calculs de base

98

- Exemple: addition

```
int resultat = 0;  
resultat = 5 + 3;
```

7

## Les calculs de base

98

- Exemple: addition

```
int resultat = 0;  
resultat = 5 + 3;
```

- Affichage:

8

## Les calculs de base

98

- Exemple: addition      

```
int resultat = 0;  
resultat = 5 + 3;
```
- Affichage:      

```
printf("5 + 3 = %d", resultat);
```

9

## Les calculs de base

98

- Exemple: addition      

```
int resultat = 0;  
resultat = 5 + 3;
```
- Affichage:      

```
printf("5 + 3 = %d", resultat);
```

Opération	Signe
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo	%

10

## Les calculs de base: La division

99

11

## Les calculs de base: La division

99

- Exemple:

12

## Les calculs de base: La division

99

- Exemple:

```
int resultat = 0;  
resultat = 5 / 2;  
printf ("5 / 2 = %d", resultat);
```

13

## Les calculs de base: La division

99

- Exemple:

```
int resultat = 0;  
resultat = 5 / 2;  
printf ("5 / 2 = %d", resultat);
```

- Test...

14

## Les calculs de base: La division

99

- Exemple:

```
int resultat = 0;
resultat = 5 / 2;
printf ("5 / 2 = %d", resultat);
```

- Test...

□ Essayez le même code en transformant juste **resultat** en **double**

15

## Les calculs de base: La division

99

- Exemple:

```
int resultat = 0;
resultat = 5 / 2;
printf ("5 / 2 = %d", resultat);
```

- Test...

□ Essayez le même code en transformant juste **resultat** en **double**

□ les nombres de l'opération sont des nombres entiers, l'ordinateur répond par un nombre entier.

16

## Les calculs de base: La division

99

- Exemple:

```
int resultat = 0;
resultat = 5 / 2;
printf ("5 / 2 = %d", resultat);
```

- Test...

**Essayez le même code en transformant juste **resultat** en **double****

- les nombres de l'opération sont des nombres entiers, l'ordinateur répond par un nombre entier.
- transformer les nombres 5 et 2 de l'opération en nombres décimaux, c'est-à-dire écrire 5.0 et 2.0

17

## Les calculs de base: La division

99

- Exemple:

```
int resultat = 0;
resultat = 5 / 2;
printf ("5 / 2 = %d", resultat);
```

- Test...

**Essayez le même code en transformant juste **resultat** en **double****

- les nombres de l'opération sont des nombres entiers, l'ordinateur répond par un nombre entier.
- transformer les nombres 5 et 2 de l'opération en nombres décimaux, c'est-à-dire écrire 5.0 et 2.0

```
double resultat = 0;
resultat = 5.0 / 2.0;
printf ("5 / 2 = %f", resultat);
```

18

## Les calculs de base: Le modulo

100

- est une opération mathématique qui permet d'obtenir **le reste d'une division**.

19

## Les calculs de base: Le modulo

100

- est une opération mathématique qui permet d'obtenir **le reste d'une division**.
- se représente par le signe %.

20

## Les calculs de base: Le modulo

100

- est une opération mathématique qui permet d'obtenir le **reste d'une division**.
- se représente par le signe %.
  - $5 \% 2 = 1$  ;
  - $14 \% 3 = 2$  ;
  - $4 \% 2 = 0$ .

21

## Des calculs entre variables

101

- **Exemple:**

22

## Des calculs entre variables

101

- Exemple:**      `resultat = nombre1 + nombre2;`

23

## Des calculs entre variables

101

- Exemple:**      `resultat = nombre1 + nombre2;`
- fait la somme des variables **nombre1** et **nombre2**, et stocke le résultat dans la variable **resultat**.

24

## Des calculs entre variables

101

- Exemple:** `resultat = nombre1 + nombre2;`
- fait la somme des variables **nombre1** et **nombre2**, et stocke le résultat dans la variable **resultat**.
- Exercice:**

25

## Des calculs entre variables

101

- Exemple:** `resultat = nombre1 + nombre2;`
- fait la somme des variables **nombre1** et **nombre2**, et stocke le résultat dans la variable **resultat**.
- Exercice:**
  - écrivez un programme qui demande deux nombres à l'utilisateur. Ces deux nombres, vous les stockez dans des variables.

26

## Des calculs entre variables

101

- Exemple:** `resultat = nombre1 + nombre2;`
- fait la somme des variables **nombre1** et **nombre2**, et stocke le résultat dans la variable **resultat**.
- Exercice:**
  - écrivez un programme qui demande deux nombres à l'utilisateur. Ces deux nombres, vous les stockez dans des variables.
  - faites la somme de ces variables et stockez le résultat dans une variable appelée **resultat**.

27

## Des calculs entre variables

101

- Exemple:** `resultat = nombre1 + nombre2;`
- fait la somme des variables **nombre1** et **nombre2**, et stocke le résultat dans la variable **resultat**.
- Exercice:**
  - écrivez un programme qui demande deux nombres à l'utilisateur. Ces deux nombres, vous les stockez dans des variables.
  - faites la somme de ces variables et stockez le résultat dans une variable appelée **resultat**.
  - affichez le résultat du calcul à l'écran.

28

## Des calculs entre variables

101

□ **Exemple:**

```
resultat = nom
```

- fait la somme des variables dans la variable **resultat**.

□ **Exercice:**

- écrivez un programme qui demande à l'utilisateur de saisir deux nombres et les stockez dans des variables.
- faites la somme de ces variables et stockez le résultat dans la variable **resultat**.
- affichez le résultat du calcul.

```
int main(int argc, char *argv[])
{
    int resultat = 0, nombre1 = 0, nombre2 = 0;
    // On demande les nombres 1 et 2 à l'utilisateur :
    printf("Entrez le nombre 1 : ");
    scanf("%d", &nombre1);
    printf("Entrez le nombre 2 : ");
    scanf("%d", &nombre2);
    // On fait le calcul :
    resultat = nombre1 + nombre2;
    // Et on affiche l'addition à l'écran :
    printf ("%d + %d = %d\n", nombre1, nombre2, resultat);
    return 0;
}
```

29

## Les raccourcis

102

- il existe en C des techniques permettant de raccourcir l'écriture des opérations.

30

## Les raccourcis

102

- il existe en C des techniques permettant de raccourcir l'écriture des opérations.

Pourquoi utiliser des raccourcis ?

31

## Les raccourcis

102

- il existe en C des techniques permettant de raccourcir l'écriture des opérations.

Pourquoi utiliser des raccourcis ?

Parce que, souvent, on fait des opérations répétitives.

32

## L'incrémentation

103

- ajouter 1 à une variable:

33

## L'incrémentation

103

- ajouter 1 à une variable:

```
nombre = nombre + 1;
```

34

## L'incrémentation

103

- ajouter 1 à une variable:

```
nombre = nombre + 1;
```

- un raccourci pour cette opération qu'on appelle **l'incrémentation**:

35

## L'incrémentation

103

- ajouter 1 à une variable:

```
nombre = nombre + 1;
```

- un raccourci pour cette opération qu'on appelle **l'incrémentation**:

```
nombre++;
```

36

## La décrémentation

104

- l'inverse de l'incrémentation : on enlève 1 à une variable.

37

## La décrémentation

104

- l'inverse de l'incrémentation : on enlève 1 à une variable.
- en forme « longue » :

38

## La décrémentation

104

- l'inverse de l'incrémentation : on enlève 1 à une variable.
- en forme « longue » :

```
nombre = nombre - 1;
```

39

## La décrémentation

104

- l'inverse de l'incrémentation : on enlève 1 à une variable.
- en forme « longue » :

```
nombre = nombre - 1;
```

- en forme « raccourcie » :

40

## La décrémentation

104

- l'inverse de l'incrémentation : on enlève 1 à une variable.
- en forme « longue » :

```
nombre = nombre - 1;
```

- en forme « raccourcie » :

```
nombre--;
```

41

## Les autres raccourcis

105

- Les raccourcis fonctionnent pour toutes les opérations de base : + - \* / %
- permet d'éviter une répétition du nom d'une variable sur une même ligne.

42

## Les autres raccourcis

105

- Les raccourcis fonctionnent pour toutes les opérations de base : + - \* / %
- permet d'éviter une répétition du nom d'une variable sur une même ligne.

```
nombre *= 2;
```

43

## Les autres raccourcis

105

- Les raccourcis fonctionnent pour toutes les opérations de base : + - \* / %
- permet d'éviter une répétition du nom d'une variable sur une même ligne.

```
nombre *= 2;
```

- Test...

44

## Les autres raccourcis

105

- Les raccourcis fonctionnent pour toutes les opérations de base : + - \* / %
- permet d'éviter une répétition du nom d'une variable sur une même ligne.

```
nombre *= 2;
```

- Test...

```
int nombre = 2;

nombre += 4; // nombre vaut 6...
nombre -= 3; // ... nombre vaut maintenant 3
nombre *= 5; // ... nombre vaut 15
nombre /= 3; // ... nombre vaut 5
nombre %= 3; // ... nombre vaut 2 (car 5 = 1 * 3 + 2)
```

45

## La bibliothèque mathématique

106

- des bibliothèques « **standard** », c'est-à-dire des bibliothèques toujours utilisables.

46

## La bibliothèque mathématique

106

- des bibliothèques « **standard** », c'est-à-dire des bibliothèques toujours utilisables.
- Ce sont des bibliothèques « **de base** » qu'on utilise très souvent.

47

## La bibliothèque mathématique

106

- des bibliothèques « **standard** », c'est-à-dire des bibliothèques toujours utilisables.
- Ce sont des bibliothèques « **de base** » qu'on utilise très souvent.
- Les bibliothèques sont des ensembles de fonctions toutes prêtées.

48

## La bibliothèque mathématique

106

- des bibliothèques « **standard** », c'est-à-dire des bibliothèques toujours utilisables.
- Ce sont des bibliothèques « **de base** » qu'on utilise très souvent.
- Les bibliothèques sont des ensembles de fonctions toutes prêtes.
- Ces fonctions ont été écrites par des programmeurs avant vous.

49

## La bibliothèque mathématique

106

- des bibliothèques « **standard** », c'est-à-dire des bibliothèques toujours utilisables.
- Ce sont des bibliothèques « **de base** » qu'on utilise très souvent.
- Les bibliothèques sont des ensembles de fonctions toutes prêtes.
- Ces fonctions ont été écrites par des programmeurs avant vous.
- Exemple:**

50

## La bibliothèque mathématique

106

- des bibliothèques « **standard** », c'est-à-dire des bibliothèques toujours utilisables.
- Ce sont des bibliothèques « **de base** » qu'on utilise très souvent.
- Les bibliothèques sont des ensembles de fonctions toutes prêtes.
- Ces fonctions ont été écrites par des programmeurs avant vous.
- **Exemple:**
  - les fonctions **printf** et **scanf** de la bibliothèque **stdio.h**

51

## La bibliothèque mathématique

106

- des bibliothèques « **standard** », c'est-à-dire des bibliothèques toujours utilisables.
- Ce sont des bibliothèques « **de base** » qu'on utilise très souvent.
- Les bibliothèques sont des ensembles de fonctions toutes prêtes.
- Ces fonctions ont été écrites par des programmeurs avant vous.
- **Exemple:**
  - les fonctions **printf** et **scanf** de la bibliothèque **stdio.h**
  - il existe une autre bibliothèque, appelée **math.h**, qui contient de nombreuses fonctions mathématiques toutes prêtes.

52

## La bibliothèque mathématique

107

- faire de puissances en C

53

## La bibliothèque mathématique

107

- faire de puissances en C
- calculer un simple carré

54

## La bibliothèque mathématique

107

- faire de puissances en C
- calculer un simple carré
- on peut toujours essayer de taper **5\$^2\$** dans le programme, mais l'ordinateur ne le comprendra jamais.

55

## La bibliothèque mathématique

107

- faire de puissances en C
- calculer un simple carré
- on peut toujours essayer de taper **5\$^2\$** dans le programme, mais l'ordinateur ne le comprendra jamais.
- Il faut indiquer la bibliothèque mathématique

56

## La bibliothèque mathématique

107

- faire de puissances en C
- calculer un simple carré
- on peut toujours essayer de taper **5\$^2\$** dans le programme, mais l'ordinateur ne le comprendra jamais.
- Il faut indiquer la bibliothèque mathématique
- Pour pouvoir utiliser les fonctions de la bibliothèque mathématique, il est indispensable de mettre la directive de préprocesseur suivante en haut de votre programme :

57

## La bibliothèque mathématique

107

- faire de puissances en C
- calculer un simple carré
- on peut toujours essayer de taper **5\$^2\$** dans le programme, mais l'ordinateur ne le comprendra jamais.
- Il faut indiquer la bibliothèque mathématique
- Pour pouvoir utiliser les fonctions de la bibliothèque mathématique, il est indispensable de mettre la directive de préprocesseur suivante en haut de votre programme :

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

58

## La bibliothèque mathématique : Exemple

108

- **fabs:** retourne la valeur absolue d'un nombre.

59

## La bibliothèque mathématique : Exemple

108

- **fabs:** retourne la valeur absolue d'un nombre.

```
double absolu = 0, nombre = -27;  
absolu = fabs(nombre); // absolu vaudra 27
```

60

## La bibliothèque mathématique : Exemple

108

- **fabs:** retourne la valeur absolue d'un nombre.

```
double absolu = 0, nombre = -27;
absolu = fabs(nombre); // absolu vaudra 27
```

- Cette fonction renvoie un **double**, donc la variable **absolu** doit être de type **double**.

61

## La bibliothèque mathématique : Exemple

108

- **fabs:** retourne la valeur absolue d'un nombre.

```
double absolu = 0, nombre = -27;
absolu = fabs(nombre); // absolu vaudra 27
```

- Cette fonction renvoie un **double**, donc la variable **absolu** doit être de type **double**.
- Il existe aussi une fonction similaire appelée **abs**, située cette fois dans **stdlib.h**.

62

## La bibliothèque mathématique : Exemple

108

- **fabs:** retourne la valeur absolue d'un nombre.

```
double absolu = 0, nombre = -27;
absolu = fabs(nombre); // absolu vaudra 27
```

- Cette fonction renvoie un **double**, donc la variable **absolu** doit être de type **double**.
- Il existe aussi une fonction similaire appelée **abs**, située cette fois dans **stdlib.h**.
  - Elle renvoie donc un nombre entier de type **int** et non un double comme **fabs**.

63

## La bibliothèque mathématique : Exemple

109

- **ceil:** renvoie le premier nombre entier après le nombre décimal qu'on lui donne.

64

## La bibliothèque mathématique : Exemple

109

- **ceil:** renvoie le premier nombre entier après le nombre décimal qu'on lui donne.
- On arrondit en fait toujours au nombre entier supérieur.

65

## La bibliothèque mathématique : Exemple

109

- **ceil:** renvoie le premier nombre entier après le nombre décimal qu'on lui donne.
- On arrondit en fait toujours au nombre entier supérieur.

```
double dessus = 0, nombre = 52.71;
dessus = ceil(nombre); // dessus vaudra 53
```

66

## La bibliothèque mathématique : Exemple

109

- **ceil:** renvoie le premier nombre entier après le nombre décimal qu'on lui donne.
- On arrondit en fait toujours au nombre entier supérieur.

```
double dessus = 0, nombre = 52.71;
dessus = ceil(nombre); // dessus vaudra 53
```

- **floor:** C'est l'inverse de la fonction **Ceil**

67

## La bibliothèque mathématique : Exemple

109

- **ceil:** renvoie le premier nombre entier après le nombre décimal qu'on lui donne.
- On arrondit en fait toujours au nombre entier supérieur.
- **floor:** C'est l'inverse de la fonction **Ceil**
- **pow:** calculer la puissance d'un nombre

68

## La bibliothèque mathématique : Exemple

109

- **ceil:** renvoie le premier nombre entier après le nombre décimal qu'on lui donne.
- On arrondit en fait toujours au nombre entier supérieur.

```
double dessus = 0, nombre = 52.71;
dessus = ceil(nombre); // dessus vaudra 53
```

- **floor:** C'est l'inverse de la fonction **Ceil**
- **pow:** calculer la puissance d'un nombre

```
pow(nombre, puissance);
```

69

## La bibliothèque mathématique : Exemple

109

- **ceil:** renvoie le premier nombre entier après le nombre décimal qu'on lui donne.

- On arrondit en fait toujours au nombre entier supérieur.

```
double dessus = 0, nombre = 52.71;
dessus = ceil(nombre); // dessus vaudra 53
```

- **floor:** C'est l'inverse de la fonction **Ceil**
- **pow:** calculer la puissance d'un nombre
- **sqrt:** calcule la racine carrée d'un nombre (renvoie un double).

70

## La bibliothèque mathématique : Exemple

109

- **ceil:** renvoie le premier nombre entier après le nombre décimal qu'on lui donne.
- On arrondit en fait toujours au nombre entier supérieur.

```
double dessus = 0, nombre = 52.71;
dessus = ceil(nombre); // dessus vaudra 53
```

- **floor:** C'est l'inverse de la fonction **Ceil**
- **pow:** calculer la puissance d'un nombre

```
pow(nombre, puissance);
```

- **sqrt:** calcule la racine carrée d'un nombre (renvoie un double).

```
double resultat = 0, nombre = 100;
resultat = sqrt(nombre); // resultat vaudra 10
```

71

## La bibliothèque mathématique : Exemple

110

- **sin, cos, tan:** les trois fameuses fonctions utilisées en trigonométrie (renvoient un double).

72

## La bibliothèque mathématique : Exemple

110

- **sin, cos, tan:** les trois fameuses fonctions utilisées en trigonométrie (renvoient un double).
- **exp:** calculer l'exponentielle d'un nombre (renvoie un double)

73

## La bibliothèque mathématique : Exemple

110

- **sin, cos, tan:** les trois fameuses fonctions utilisées en trigonométrie (renvoient un double).
- **exp:** calculer l'exponentielle d'un nombre (renvoie un double)
- **log:** calculer le logarithme népérien d'un nombre

74

## La bibliothèque mathématique : Exemple

110

- **sin, cos, tan:** les trois fameuses fonctions utilisées en trigonométrie (renvoient un double).
- **exp:** calculer l'exponentielle d'un nombre (renvoie un double)
- **log:** calculer le logarithme népérien d'un nombre
- **log10:** calculer le logarithme base 10 d'un nombre.



# Les conditions

1

## La condition if... else

112

- Les conditions permettent de tester des variables.
- **Exemple:** si la variable machin est égale à 50, fais ceci

2

## Plan

113

- quelques **symboles** à connaître avant de commencer,
- le test **if**,
- le test **else**,
- le test **else if**,
- **plusieurs conditions** à la fois,
- quelques erreurs courantes à éviter.

3

## Quelques symboles à connaître

114

- Ces symboles sont indispensables pour réaliser des conditions.

4

## Quelques symboles à connaître

114

- Ces symboles sont indispensables pour réaliser des conditions.

Symbol	Signification
<code>==</code>	est égal à
<code>&gt;</code>	est supérieur à
<code>&lt;</code>	est inférieur à
<code>&gt;=</code>	est supérieur ou égal à
<code>&lt;=</code>	est inférieur ou égal à
<code>!=</code>	est différent de

5

## Un if simple

115

SI la variable vaut ça,  
ALORS fais ceci.

6

## Un if simple

115

SI la variable vaut ça,  
ALORS fais ceci.

```
if /* Votre condition */  
{  
    // Instructions à exécuter si la condition est vraie  
}
```

7

## Un if simple

115

SI la variable vaut ça,  
ALORS fais ceci.

```
if /* Votre condition */  
{  
    // Instructions à exécuter si la condition est vraie  
}
```

□ Exemple:

8

## Un if simple

115

SI la variable vaut ça,  
ALORS fais ceci.

```
if /* Votre condition */
{
    // Instructions à exécuter si la condition est vraie
}
```

□ Exemple:

```
if (age >= 18)
{
    printf ("Vous etes majeur !");
}
```

9

## Un if simple

115

SI la variable vaut ça,  
ALORS fais ceci.

```
if /* Votre condition */
{
    // Instructions à exécuter si la condition est vraie
}
```

□ Exemple:

```
if (age >= 18)
{
    printf ("Vous etes majeur !");
}
```

□ Test...

10

## Un if simple

115

SI la variable vaut ça,  
ALORS fais ça.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int age = 20;

    if (age >= 18)
    {
        printf ("Vous etes majeur !\n");
    }
    return 0;
}
```

□ Exemple:

□ Test...

11

## Un if simple

116

□ sur une même ligne

```
if (age >= 18) { printf ("Vous etes majeur !"); }
```

12

## Un if simple

116

- sur une même ligne

```
if (age >= 18) { printf ("Vous etes majeur !"); }
```

- difficilement lisible

13

## Le else pour dire « sinon »

117

- si le test n'a pas marché (il est faux), on va dire à l'ordinateur d'exécuter d'autres instructions.

14

## Le **else** pour dire « sinon »

117

- si le test n'a pas marché (il est faux), on va dire à l'ordinateur d'exécuter d'autres instructions.

SI la variable vaut ça,  
ALORS fais ceci,  
SINON fais cela.

15

## Le **else** pour dire « sinon »

117

- si le test n'a pas marché (il est faux), on va dire à l'ordinateur d'exécuter d'autres instructions.

SI la variable vaut ça,  
ALORS fais ceci,  
SINON fais cela.

```
if (age >= 18) // Si l'âge est supérieur ou égal
à 18
{
    printf ("Vous êtes majeur !");
}
else // Sinon...
{
    printf ("Vous êtes mineur !");
}
```

16

## Le **else if** pour dire « sinon si »

118

- il est possible aussi de faire un « sinon si » pour faire un autre test si le premier test n'a pas marché.

17

## Le **else if** pour dire « sinon si »

118

- il est possible aussi de faire un « sinon si » pour faire un autre test si le premier test n'a pas marché.
- Le « sinon si » se place entre le **if** et le **else**

18

## Le **else if** pour dire « sinon si »

118

- il est possible aussi de faire un « sinon si » pour faire un autre test si le premier test n'a pas marché.
- Le « sinon si » se place entre le **if** et le **else**

SI la variable vaut ça ALORS fais ceci,  
SINON SI la variable vaut ça ALORS fais ça,  
SINON fais cela.

19

## Le **else if** pour dire « sinon si »

118

- il est possible aussi de faire un « sinon si » pour faire un autre test si le premier test n'a pas marché.
- Le « sinon si » se place entre le **if** et le **else**

SI la variable vaut ça ALORS fais ceci,  
SINON SI la variable vaut ça ALORS fais ça,  
SINON fais cela.

```
if (age >= 18) // Si l'âge est supérieur ou égal
à 18
{
    printf ("Vous êtes majeur !");
}
else if ( age > 4 ) // Sinon, si l'âge est au moins
supérieur à 4
{
    printf ("Bon t'es pas trop jeune quand
meme...");
}
else // Sinon...
{
    printf ("Aga gaa aga gaaa"); // Langage
bébé, vous pouvez pas comprendre
}
```

20

## Le **else if** pour dire « sinon si »

119

- Le **else** et le **else if** ne sont pas obligatoires.

21

## Le **else if** pour dire « sinon si »

119

- Le **else** et le **else if** ne sont pas obligatoires.
- Pour faire une condition, seul un **if** est nécessaire

22

## Le **else if** pour dire « sinon si »

119

- Le **else** et le **else if** ne sont pas obligatoires.
- Pour faire une condition, seul un **if** est nécessaire
- on peut mettre autant de **else if** que l'on veut:

23

## Le **else if** pour dire « sinon si »

119

- Le **else** et le **else if** ne sont pas obligatoires.
- Pour faire une condition, seul un **if** est nécessaire
- on peut mettre autant de **else if** que l'on veut:

SI la variable vaut ça,  
 ALORS fais ceci,  
 SINON SI la variable vaut ça ALORS fais ça,  
 SINON SI la variable vaut ça ALORS fais ça,  
 SINON SI la variable vaut ça ALORS fais ça,  
 SINON fais cela.

24

## Plusieurs conditions à la fois

120

- tester si l'âge est supérieur à 18 **ET** si l'âge est inférieur à 25.
- il va falloir utiliser de nouveaux symboles :

25

## Plusieurs conditions à la fois

120

- tester si l'âge est supérieur à 18 **ET** si l'âge est inférieur à 25.
- il va falloir utiliser de nouveaux symboles :

Symbole	Signification
&&	ET
	OU
!	NON

26

## Plusieurs conditions à la fois

121

Test **ET**

27

## Plusieurs conditions à la fois

121

Test **ET**

`if (age > 18 && age < 25)`

28

## Plusieurs conditions à la fois

121

- Test **ET**      `if (age > 18 && age < 25)`
- Test **OU:**

29

## Plusieurs conditions à la fois

121

- Test **ET**      `if (age > 18 && age < 25)`
- Test **OU:**  

```
if (age > 30 || argent > 100000)
{
    printf("Bienvenue chez PicsouBanque !");
}
else
{
    printf("Hors de ma vue, miserable !");
}
```

30

## Plusieurs conditions à la fois

121

- Test **ET**

```
if (age > 18 && age < 25)
```

- Test **OU:**

```
if (age > 30 || argent > 100000)
{
    printf("Bienvenue chez PicsouBanque !");
}
else
{
    printf("Hors de ma vue, miserable !");
}
```

- Pour le taper **||** sur un clavier **AZERTY** français, il faudra faire **Alt Gr + 6**.  
Sur un clavier belge, il faudra faire **Alt Gr + &**.

31

## Plusieurs conditions à la fois

122

- Test **NON**

- le point d'exclamation signifie « non ».

```
if (!(age < 18))
```

- **N'oubliez pas les deux signes ==**
- **Le point-virgule de trop**

32

## Les booléens, le cœur des conditions

123

- les conditions font intervenir quelque chose qu'on appelle les **booléens** en informatique.

33

## Les booléens, le cœur des conditions

123

- les conditions font intervenir quelque chose qu'on appelle les **booléens** en informatique.
- Exemple:

34

## Les booléens, le cœur des conditions

123

- les conditions font intervenir quelque chose qu'on appelle les **booléens** en informatique.

- Exemple:

```
if (1)
{
    printf("C'est vrai");
}
else
{
    printf("C'est faux");
}
```

35

## Les booléens, le cœur des conditions

123

- les conditions font intervenir quelque chose qu'on appelle les **booléens** en informatique.

- Exemple:

```
if (1)
{
    printf("C'est vrai");
}
else
{
    printf("C'est faux");
}
```

```
if (0)
{
    printf("C'est vrai");
}
else
{
    printf("C'est faux");
}
```

36

## Les booléens, le cœur des conditions

123

- les conditions font intervenir quelque chose qu'on appelle les **booléens** en informatique.

- Exemple:

```
if (1)
{
    printf("C'est vrai");
}
else
{
    printf("C'est faux");
}
```

```
if (0)
{
    printf("C'est vrai");
}
else
{
    printf("C'est faux");
}
```

- remplacer le 0 par n'importe quel autre nombre entier, comme 4, 15, 226, -10, -36, etc.

37

## Les booléens, le cœur des conditions

124

- un test dans un if, ce test renvoie la valeur 1 s'il est vrai, et 0 s'il est faux.

38

## Les booléens, le cœur des conditions

124

- un test dans un if, ce test renvoie la valeur 1 s'il est vrai, et 0 s'il est faux.
- Un test avec une variable: c'est une opération !

39

## Les booléens, le cœur des conditions

124

- un test dans un if, ce test renvoie la valeur 1 s'il est vrai, et 0 s'il est faux.
- Un test avec une variable: c'est une opération !

```
int age = 20;  
int majeur = 0;  
  
majeur = age >= 18;  
printf("Majeur vaut : %d\n", majeur);
```

40

## Les booléens, le cœur des conditions

124

- un test dans un if, ce test renvoie la valeur 1 s'il est vrai, et 0 s'il est faux.

- Un test avec une variable: c'est une opération !

```
int age = 20;
int majeur = 0;

majeur = age >= 18;
printf("Majeur vaut : %d\n", majeur);
```

- La variable majeur est un booléen

41

## Les booléens, le cœur des conditions

124

- un test dans un if, ce test renvoie la valeur 1 s'il est vrai, et 0 s'il est faux.

- Un test avec une variable: c'est une opération !

```
int age = 20;
int majeur = 0;

majeur = age >= 18;
printf("Majeur vaut : %d\n", majeur);
```

- La variable majeur est un booléen

- on dit qu'une variable à laquelle on fait prendre les valeurs 0 et 1 est un booléen.

42

## Les booléens dans les conditions

125

- En langage C, il n'existe pas de type de variable « **booléen** ».
- on est obligé d'utiliser un type **entier** comme **int** pour gérer les **booléens**.

```
int majeur = 1;

if (majeur)
{
    printf("Tu es majeur !");
}
else
{
    printf("Tu es mineur");
}
```

if (majeur && garcon)

43

## Les booléens dans les conditions

126

si on fait le test if (majeur == 1), ça marche aussi, non ?

44

## La condition switch

127

- le if... else peut s'avérer quelque peu... répétitif:

45

## La condition switch

127

- le if... else peut s'avérer quelque peu... répétitif:
- **Exemple:**

46

## La condition switch

127

- le if... else peut s'avérer quelque peu... répétitif:

- **Exemple:**

```
if (age == 2)
{   printf("Salut bebe !"); }
else if (age == 6)
{   printf("Salut gamin !"); }
else if (age == 12)
{   printf("Salut jeune !"); }
else if (age == 16)
{   printf("Salut ado !"); }
else if (age == 18)
{   printf("Salut adulte !"); }
else if (age == 68)
{   printf("Salut papy !"); }
else
{   printf("Je n'ai aucune phrase de prête pour ton age"); }
```

47

## Construire un switch

128

- éviter d'avoir à faire des répétitions:

48

## Construire un switch

128

- éviter d'avoir à faire des répétitions:
- **Exemple:**

49

## Construire un switch

128

- éviter d'avoir à faire des répétitions:
- **Exemple:**

```
switch (age)
{
    case 2:
        printf("Salut bebe !");
        break;
    case 6:
        printf("Salut gamin !");
        break;
    case 12:
        printf("Salut jeune !");
        break;
    case 16:
        printf("Salut ado !");
        break;
    case 18:
        printf("Salut adulte !");
        break;
    case 68:
        printf("Salut papy !");
        break;
    default:
        printf("Je n'ai aucune phrase de prete pour ton age ");
        break;
}
```

50

## Gérer un menu avec un switch

129

- Le switch est très souvent utilisé pour faire des **menus** en console.

51

## Gérer un menu avec un switch

129

- Le switch est très souvent utilisé pour faire des **menus** en console.
- **Exemple:**

52

## Gérer un menu avec un switch

129

- Le switch est très souvent utilisé pour faire des **menus** en console.

- **Exemple:**

```
==== Menu ====
1. Royal Cheese
2. Mc Deluxe
3. Mc Bacon
4. Big Mac
Votre choix ?
```

53

## Gérer un menu avec un switch

129

- Le switch est très souvent utilisé pour faire des **menus** en console.

- **Exemple:**

```
==== Menu ====
1. Royal Cheese
2. Mc Deluxe
3. Mc Bacon
4. Big Mac
Votre choix ?
```

- reproduisez ce menu

54

## Gérer un menu avec un switch

129

- Le switch est très souvent utilisé pour faire des **menus** en console.

- **Exemple:**

```
==== Menu ====
1. Royal Cheese
2. Mc Deluxe
3. Mc Bacon
4. Big Mac
Votre choix ?
```

- reproduisez ce menu
- enregistrez le choix de l'utilisateur dans une variable **choixMenu**,

55

## Gérer un menu avec un switch

129

- Le switch est très souvent utilisé pour faire des **menus** en console.

- **Exemple:**

```
==== Menu ====
1. Royal Cheese
2. Mc Deluxe
3. Mc Bacon
4. Big Mac
Votre choix ?
```

- reproduisez ce menu
- enregistrez le choix de l'utilisateur dans une variable **choixMenu**,
- faites un **switch** pour dire à l'utilisateur « tu as choisi le menu Royal Cheese » par exemple.

56

## Les ternaires : des conditions condensées

130

- Ou Expressions ternaires.

```
age = (majeur) ? 18 : 17;
```

57

## Les ternaires : des conditions condensées

130

- Ou Expressions ternaires.
- **Exemple:**

```
age = (majeur) ? 18 : 17;
```

58

## Les ternaires : des conditions condensées

130

- Ou Expressions ternaires.

- **Exemple:**

```
if (majeur)
    age = 18;
else
    age = 17;
```

```
age = (majeur) ? 18 : 17;
```

59

## Les ternaires : des conditions condensées

130

- Ou Expressions ternaires.

- **Exemple:**

```
if (majeur)
    age = 18;
else
    age = 17;
```

- La même condition en ternaire

```
age = (majeur) ? 18 : 17;
```

60

Université Abdelmalek Essaâdi  
École Normale Supérieure  
Tétouan



2023/2024

# Les boucles

1

## Qu'est-ce qu'une boucle ?

132

- une boucle est une structure qui permet de répéter les mêmes instructions plusieurs fois.

2

## Qu'est-ce qu'une boucle ?

132

- une boucle est une structure qui permet de répéter les mêmes instructions plusieurs fois.
- trois types de boucles courantes en C :

3

## Qu'est-ce qu'une boucle ?

132

- une boucle est une structure qui permet de répéter les mêmes instructions plusieurs fois.
- trois types de boucles courantes en C :
  - while
  - do... while
  - for

4

## Qu'est-ce qu'une boucle ?

133

- ce qu'il se passe dans l'ordre :

5

## Qu'est-ce qu'une boucle ?

133

- ce qu'il se passe dans l'ordre :
  1. l'ordinateur lit les instructions de haut en bas (comme d'habitude) ;

6

## Qu'est-ce qu'une boucle ?

133

### ce qu'il se passe dans l'ordre :

1. l'ordinateur lit les instructions de haut en bas (comme d'habitude) ;
2. puis, une fois arrivé à la fin de la boucle, il repart à la première instruction ;

7

## Qu'est-ce qu'une boucle ?

133

### ce qu'il se passe dans l'ordre :

1. l'ordinateur lit les instructions de haut en bas (comme d'habitude) ;
2. puis, une fois arrivé à la fin de la boucle, il repart à la première instruction ;
3. il recommence alors à lire les instructions de haut en bas...

8

## Qu'est-ce qu'une boucle ?

133

### ce qu'il se passe dans l'ordre :

1. l'ordinateur lit les instructions de haut en bas (comme d'habitude) ;
2. puis, une fois arrivé à la fin de la boucle, il repart à la première instruction ;
3. il recommence alors à lire les instructions de haut en bas...
4. ... et il repart au début de la boucle.

9

## Qu'est-ce qu'une boucle ?

133

### ce qu'il se passe dans l'ordre :

1. l'ordinateur lit les instructions de haut en bas (comme d'habitude) ;
2. puis, une fois arrivé à la fin de la boucle, il repart à la première instruction ;
3. il recommence alors à lire les instructions de haut en bas...
4. ... et il repart au début de la boucle.

10

## Qu'est-ce qu'une boucle ?

133

ce qu'il se passe dans l'ordre :

1. l'ordinateur lit les instructions de haut en bas (comme d'habitude) ;
2. puis, une fois arrivé à la fin de la boucle, il repart à la première instruction ;
3. il recommence alors à lire les instructions de haut en bas...
4. ... et il repart au début de la boucle.



11

## Qu'est-ce qu'une boucle ?

133

ce qu'il se passe dans l'ordre :

1. l'ordinateur lit les instructions de haut en bas (comme d'habitude) ;
2. puis, une fois arrivé à la fin de la boucle, il repart à la première instruction ;
3. il recommence alors à lire les instructions de haut en bas...
4. ... et il repart au début de la boucle.



Problème: si on ne l'arrête pas, l'ordinateur est capable de répéter les instructions à l'infini !!

12

## Qu'est-ce qu'une boucle ?

134

- les **conditions** !

13

## Qu'est-ce qu'une boucle ?

134

- les **conditions** !
- Quand on crée une boucle, on indique toujours une condition.

14

## Qu'est-ce qu'une boucle ?

134

### les **conditions** !

- Quand on crée une boucle, on indique toujours une condition.
- Cette condition signifiera « **Répète la boucle tant que cette condition est vraie** ».

15

## La boucle while

135

### une boucle while :

16

## La boucle while

135

- une boucle while :

```
while /* Condition */  
{  
    // Instructions à répéter  
}
```

17

## La boucle while

135

- une boucle while :

```
while /* Condition */  
{  
    // Instructions à répéter  
}
```

- « Tant que la condition est vraie, répète les instructions entre accolades ».

18

## La boucle while

135

- une boucle while :

```
while /* Condition */
{
    // Instructions à répéter
}
```

- « Tant que la condition est vraie, répète les instructions entre accolades ».

- **Exemple:**

19

## La boucle while

135

- une boucle while :

```
while /* Condition */
{
    // Instructions à répéter
}
```

- « Tant que la condition est vraie, répète les instructions entre accolades ».

- **Exemple:**

```
int nombreEntre = 0;
while (nombreEntre != 47)
{
    printf("Tapez le nombre 47 ! ");
    scanf("%d", &nombreEntre);
}
```

20

## La boucle while

136

- **Exemple:** Une boucle qui se répète un certain nombre de fois.

21

## La boucle while

136

- **Exemple:** Une boucle qui se répète un certain nombre de fois.

```
int compteur = 0;
while (compteur < 10)
{
    printf("Salut les MCWs !\n");
    compteur++;
}
```

22

## La boucle while

136

- **Exemple:** Une boucle qui se répète un certain nombre de fois.

```
int compteur = 0;
while (compteur < 10)
{
    printf("Salut les MCWs !\n");
    compteur++;
}
```

```
int compteur = 0;
while (compteur < 10)
{
    printf("La variable compteur vaut %d\n", compteur);
    compteur++;
}
```

23

## La boucle while

137

- Attention aux boucles infinies

24

## La boucle while

137

- Attention aux boucles infinies
  - ▣ Si la condition est toujours vraie, votre programme ne s'arrêtera jamais !

25

## La boucle while

137

- Attention aux boucles infinies
  - ▣ Si la condition est toujours vraie, votre programme ne s'arrêtera jamais !

```
while (1)
{
    printf("Boucle infinie\n");
}
```

26

## La boucle do... while

138

- La seule chose qui change par rapport à **while**, c'est la position de la condition.

27

## La boucle do... while

138

- La seule chose qui change par rapport à **while**, c'est la position de la condition.

```
int compteur = 0;
do
{
    printf("Salut les Zeros !\n");
    compteur++;
} while (compteur < 10);
```

28

## La boucle do... while

138

- La seule chose qui change par rapport à **while**, c'est la position de la condition.

```
int compteur = 0;
do
{
    printf("Salut les Zeros !\n");
    compteur++;
} while (compteur < 10);
```

- cette boucle s'exécutera toujours au moins une fois.

29

## La boucle do... while

138

- La seule chose qui change par rapport à **while**, c'est la position de la condition.

```
int compteur = 0;
do
{
    printf("Salut les Zeros !\n");
    compteur++;
} while (compteur < 10);
```

- cette boucle s'exécutera toujours au moins une fois.
- il y a un point-virgule à la fin !

30

## La boucle for

139

- une autre façon de faire une boucle while.
- **Exemple:**

31

## La boucle for

139

- une autre façon de faire une boucle while.
- **Exemple:**

```
int compteur = 0;
while (compteur < 10)
{
    printf("Salut les MCWs !\n");
    compteur++;
}
```

32

## La boucle for

139

- une autre façon de faire une boucle while.

- **Exemple:**

```
int compteur = 0;
while (compteur < 10)
{
    printf("Salut les MCWs !\n");
    compteur++;
}
```

- l'équivalent en boucle for :

33

## La boucle for

139

- une autre façon de faire une boucle while.

- **Exemple:**

```
int compteur = 0;
while (compteur < 10)
{
    printf("Salut les MCWs !\n");
    compteur++;
}
```

- l'équivalent en boucle for :

```
int compteur;
for (compteur = 0 ; compteur < 10 ; compteur++)
{
    printf("Salut les Zeros !\n");
}
```

34

## La boucle for

140

- trois instructions condensées:

35

## La boucle for

140

- trois instructions condensées:

■ La première est **l'initialisation** : cette première instruction est utilisée pour préparer notre variable compteur. Dans notre cas, on initialise la variable à 0.

36

## La boucle for

140

- trois instructions condensées:
  - ▣ La première est **l'initialisation** : cette première instruction est utilisée pour préparer notre variable compteur. Dans notre cas, on initialise la variable à 0.
  - ▣ La seconde est la **condition** : comme pour la boucle **while**, c'est la condition qui dit si la boucle doit être répétée ou non. Tant que la condition est vraie, la boucle **for** continue.

37

## La boucle for

140

- trois instructions condensées:
  - ▣ La première est **l'initialisation** : cette première instruction est utilisée pour préparer notre variable compteur. Dans notre cas, on initialise la variable à 0.
  - ▣ La seconde est la **condition** : comme pour la boucle **while**, c'est la condition qui dit si la boucle doit être répétée ou non. Tant que la condition est vraie, la boucle **for** continue.
  - ▣ Enfin, il y a l'**incrémantation** : cette dernière instruction est exécutée à la fin de chaque tour de boucle pour mettre à jour la variable compteur. La quasi-totalité du temps on fera une incrémantation, mais on peut aussi faire une décrémantation (**variable--**) ou encore n'importe quelle autre opération (**variable += 2**; pour avancer de 2 en 2 par exemple).

38

## TP

141

- un petit jeu « Plus ou moins ».

39

## TP

141

- un petit jeu « Plus ou moins ».
- L'ordinateur tire au sort un nombre entre 1 et 100.

40

## TP

141

- un petit jeu « Plus ou moins ».
  - ▣ L'ordinateur tire au sort un nombre entre 1 et 100.
  - ▣ Il vous demande de deviner le nombre. Vous entrez donc un nombre entre 1 et 100.

41

## TP

141

- un petit jeu « Plus ou moins ».
  - ▣ L'ordinateur tire au sort un nombre entre 1 et 100.
  - ▣ Il vous demande de deviner le nombre. Vous entrez donc un nombre entre 1 et 100.
  - ▣ L'ordinateur compare le nombre que vous avez entré avec le nombre « mystère » qu'il a tiré au sort. Il vous dit si le nombre mystère est supérieur ou inférieur à celui que vous avez entré.

42

## TP

141

- un petit jeu « Plus ou moins ».
  - ▣ L'ordinateur tire au sort un nombre entre 1 et 100.
  - ▣ Il vous demande de deviner le nombre. Vous entrez donc un nombre entre 1 et 100.
  - ▣ L'ordinateur compare le nombre que vous avez entré avec le nombre « mystère » qu'il a tiré au sort. Il vous dit si le nombre mystère est supérieur ou inférieur à celui que vous avez entré.
  - ▣ Puis l'ordinateur vous redemande le nombre.

43

## TP

141

- un petit jeu « Plus ou moins ».
  - ▣ L'ordinateur tire au sort un nombre entre 1 et 100.
  - ▣ Il vous demande de deviner le nombre. Vous entrez donc un nombre entre 1 et 100.
  - ▣ L'ordinateur compare le nombre que vous avez entré avec le nombre « mystère » qu'il a tiré au sort. Il vous dit si le nombre mystère est supérieur ou inférieur à celui que vous avez entré.
  - ▣ Puis l'ordinateur vous redemande le nombre.
  - ▣ ... Et il vous indique si le nombre mystère est supérieur ou inférieur.

44

## TP

141

- un petit jeu « Plus ou moins ».
  - ▣ L'ordinateur tire au sort un nombre entre 1 et 100.
  - ▣ Il vous demande de deviner le nombre. Vous entrez donc un nombre entre 1 et 100.
  - ▣ L'ordinateur compare le nombre que vous avez entré avec le nombre « mystère » qu'il a tiré au sort. Il vous dit si le nombre mystère est supérieur ou inférieur à celui que vous avez entré.
  - ▣ Puis l'ordinateur vous redemande le nombre.
  - ▣ ... Et il vous indique si le nombre mystère est supérieur ou inférieur.
  - ▣ Et ainsi de suite, jusqu'à ce que vous trouviez le nombre mystère.

45

## TP

141

- un petit jeu « Plus ou moins ».
  - ▣ L'ordinateur tire au sort un nombre entre 1 et 100.
  - ▣ Il vous demande de deviner le nombre. Vous entrez donc un nombre entre 1 et 100.
  - ▣ L'ordinateur compare le nombre que vous avez entré avec le nombre « mystère » qu'il a tiré au sort. Il vous dit si le nombre mystère est supérieur ou inférieur à celui que vous avez entré.
  - ▣ Puis l'ordinateur vous redemande le nombre.
  - ▣ ... Et il vous indique si le nombre mystère est supérieur ou inférieur.
  - ▣ Et ainsi de suite, jusqu'à ce que vous trouviez le nombre mystère.
  - ▣ Le but du jeu, bien sûr, est de trouver le nombre mystère en un minimum de coups.

46

## TP

141

- un petit jeu « Plus ou moins ».

- L'ordinateur tire au sort Quel est le nombre ? 50  
C'est plus !
- Il vous demande de deviner le nombre mystère. Quel est le nombre ? 75  
C'est plus !
- L'ordinateur compare le nombre que vous avez entré avec le nombre mystère. Quel est le nombre ? 85  
C'est moins !
- Puis l'ordinateur vous renvoie une réponse. Quel est le nombre ? 80  
C'est moins !
- ... Et il vous indique si le nombre mystère est plus ou moins que la dernière réponse. Quel est le nombre ? 78  
C'est plus !
- Et ainsi de suite, jusqu'à ce que vous trouviez le bon nombre. Quel est le nombre ? 79  
Bravo, vous avez trouvé le nombre mystère !!!
- Le but du jeu, bien sûr, est de trouver le nombre mystère en demandant le moins de questions possibles.



# Les fonctions

1

## Introduction

2

- Tous les gros programmes en C sont en fait des assemblages de petits bouts de code.

2

## Introduction

2

- Tous les gros programmes en C sont en fait des assemblages de petits bouts de code.
- ce qu'on appelle... des fonctions !

3

## Introduction

2

- Tous les gros programmes en C sont en fait des assemblages de petits bouts de code.
- ce qu'on appelle... des fonctions !
- un programme en C commençait par une fonction appelée **main**.

4

## Créer et appeler une fonction

3

```
#include <stdio.h>
#include <stdlib.h>} Directives de préprocesseur

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Instructions } Fonction

5

## Créer et appeler une fonction

3

```
#include <stdio.h>
#include <stdlib.h>} Directives de préprocesseur

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Instructions } Fonction

jusqu'ici nous sommes restés à l'intérieur de la fonction main

6

## Créer et appeler une fonction

4

- dans la réalité:

7

## Créer et appeler une fonction

4

- dans la réalité:

- Quasiment aucun programme n'est écrit uniquement à l'intérieur des accolades de la fonction main.

8

## Créer et appeler une fonction

4

- dans la réalité:

- Quasiment aucun programme n'est écrit uniquement à l'intérieur des accolades de la fonction main.

- découper les programmes en petits bouts:

9

## Créer et appeler une fonction

4

- dans la réalité:

- Quasiment aucun programme n'est écrit uniquement à l'intérieur des accolades de la fonction main.

- découper les programmes en petits bouts:

- Chaque « petit bout de programme » sera ce qu'on appelle une fonction.

10

## Créer et appeler une fonction

4

- dans la réalité:
  - Quasiment aucun programme n'est écrit uniquement à l'intérieur des accolades de la fonction main.
- découper les programmes en petits bouts:
  - Chaque « petit bout de programme » sera ce qu'on appelle une fonction.
- Une fonction exécute des actions et renvoie un résultat.

11

## Créer et appeler une fonction

4

- dans la réalité:
  - Quasiment aucun programme n'est écrit uniquement à l'intérieur des accolades de la fonction main.
- découper les programmes en petits bouts:
  - Chaque « petit bout de programme » sera ce qu'on appelle une fonction.
- Une fonction exécute des actions et renvoie un résultat.
  - C'est un morceau de code qui sert à faire quelque chose de précis.

12

## Créer et appeler une fonction

5

- On dit qu'une fonction possède une entrée et une sortie.

□

13

## Créer et appeler une fonction

5

- On dit qu'une fonction possède une entrée et une sortie.



14

## Créer et appeler une fonction

6

- Lorsqu'on appelle une fonction, il y a trois étapes.

15

## Créer et appeler une fonction

6

- Lorsqu'on appelle une fonction, il y a trois étapes.

■ **L'entrée:** on fait « rentrer » des informations dans la fonction (en lui donnant des informations avec lesquelles travailler).

16

## Créer et appeler une fonction

6

- Lorsqu'on appelle une fonction, il y a trois étapes.
- **L'entrée:** on fait « rentrer » des informations dans la fonction (en lui donnant des informations avec lesquelles travailler).
- **Les calculs :** grâce aux informations qu'elle a reçues en entrée, la fonction travaille.

17

## Créer et appeler une fonction

6

- Lorsqu'on appelle une fonction, il y a trois étapes.
- **L'entrée:** on fait « rentrer » des informations dans la fonction (en lui donnant des informations avec lesquelles travailler).
- **Les calculs :** grâce aux informations qu'elle a reçues en entrée, la fonction travaille.
- **La sortie :** une fois qu'elle a fini ses calculs, la fonction renvoie un résultat. C'est ce qu'on appelle la sortie, ou encore le retour.

18

## Créer et appeler une fonction

7

- **Exemple:** une fonction appelée triple qui calcule le triple du nombre qu'on lui donne, en le multipliant par 3:

19

## Créer et appeler une fonction

7

- **Exemple:** une fonction appelée triple qui calcule le triple du nombre qu'on lui donne, en le multipliant par 3:



La fonction « triple » multiplie le  
nombre en entrée par 3

20

## Créer et appeler une fonction

8

- Le but des fonctions est donc de simplifier le code source, pour ne pas avoir à retaper le même code plusieurs fois d'affilée.

21

## Schéma d'une fonction

9

```
type nomFonction(parametres)
{
    // Insérez vos instructions ici
}
```

22

## Schéma d'une fonction

9

```
type nomFonction(parametres)
{
    // Insérez vos instructions ici
}
```

- **type** (correspond à la sortie) : c'est le type de la fonction.

23

## Schéma d'une fonction

9

```
type nomFonction(parametres)
{
    // Insérez vos instructions ici
}
```

- **type** (correspond à la sortie) : c'est le type de la fonction.
- Comme les variables, les fonctions ont un type.

24

## Schéma d'une fonction

9

```
type nomFonction(parametres)
{
    // Insérez vos instructions ici
}
```

- **type** (correspond à la sortie) : c'est le type de la fonction.
- Comme les variables, les fonctions ont un type.
- Ce type dépend du résultat que la fonction renvoie :

25

## Schéma d'une fonction

9

```
type nomFonction(parametres)
{
    // Insérez vos instructions ici
}
```

- **type** (correspond à la sortie) : c'est le type de la fonction.
- Comme les variables, les fonctions ont un type.
- Ce type dépend du résultat que la fonction renvoie :
  - ▣ si la fonction renvoie un nombre décimal, vous mettrez sûrement double,

26

## Schéma d'une fonction

9

```
type nomFonction(parametres)
{
    // Insérez vos instructions ici
}
```

- type** (correspond à la sortie) : c'est le type de la fonction.
- Comme les variables, les fonctions ont un type.
- Ce type dépend du résultat que la fonction renvoie :
  - si la fonction renvoie un nombre décimal, vous mettrez sûrement double,
  - si elle renvoie un entier vous mettrez int ou long par exemple.

27

## Schéma d'une fonction

9

```
type nomFonction(parametres)
{
    // Insérez vos instructions ici
}
```

- type** (correspond à la sortie) : c'est le type de la fonction.
- Comme les variables, les fonctions ont un type.
- Ce type dépend du résultat que la fonction renvoie :
  - si la fonction renvoie un nombre décimal, vous mettrez sûrement double,
  - si elle renvoie un entier vous mettrez int ou long par exemple.
- il est aussi possible de créer des fonctions qui ne renvoient rien !

28

## Schéma d'une fonction

10

- Il y a donc deux sortes de fonctions :

29

## Schéma d'une fonction

10

- Il y a donc deux sortes de fonctions :

- ▣ les fonctions qui renvoient une valeur : on leur met un des types que l'on connaît (**char**, **int**, **double**, etc.) ;

30

## Schéma d'une fonction

10

- Il y a donc deux sortes de fonctions :
  - les fonctions qui renvoient une valeur : on leur met un des types que l'on connaît (**char**, **int**, **double**, etc.) ;
  - les fonctions qui ne renvoient pas de valeur : on leur met un type spécial **void** (qui signifie « vide »).

31

## Schéma d'une fonction

11

- **nomFonction** : c'est le nom de votre fonction.

32

## Schéma d'une fonction

11

- **nomFonction** : c'est le nom de votre fonction.

- Vous pouvez appeler votre fonction comme vous voulez, du temps que vous respectez les mêmes règles que pour les variables (pas d'accents, pas d'espaces, etc.).

33

## Schéma d'une fonction

11

- **nomFonction** : c'est le nom de votre fonction.

- Vous pouvez appeler votre fonction comme vous voulez, du temps que vous respectez les mêmes règles que pour les variables (pas d'accents, pas d'espaces, etc.).

- **paramètres** (correspond à l'entrée) : entre parenthèses

34

## Schéma d'une fonction

11

- **nomFonction** : c'est le nom de votre fonction.
  - Vous pouvez appeler votre fonction comme vous voulez, du temps que vous respectez les mêmes règles que pour les variables (pas d'accents, pas d'espaces, etc.).
- **paramètres** (correspond à l'entrée) : entre parenthèses
  - Vous pouvez envoyer des paramètres à la fonction.

35

## Schéma d'une fonction

11

- **nomFonction** : c'est le nom de votre fonction.
  - Vous pouvez appeler votre fonction comme vous voulez, du temps que vous respectez les mêmes règles que pour les variables (pas d'accents, pas d'espaces, etc.).
- **paramètres** (correspond à l'entrée) : entre parenthèses
  - Vous pouvez envoyer des paramètres à la fonction.
  - Ce sont des valeurs avec lesquelles la fonction va travailler.

36

## Schéma d'une fonction

11

- **nomFonction** : c'est le nom de votre fonction.
  - Vous pouvez appeler votre fonction comme vous voulez, du temps que vous respectez les mêmes règles que pour les variables (pas d'accents, pas d'espaces, etc.).
- **paramètres** (correspond à l'entrée) : entre parenthèses
  - Vous pouvez envoyer des paramètres à la fonction.
  - Ce sont des valeurs avec lesquelles la fonction va travailler.
- **Vous pouvez envoyer autant de paramètres que vous le voulez.**

37

## Schéma d'une fonction

11

- **nomFonction** : c'est le nom de votre fonction.
  - Vous pouvez appeler votre fonction comme vous voulez, du temps que vous respectez les mêmes règles que pour les variables (pas d'accents, pas d'espaces, etc.).
- **paramètres** (correspond à l'entrée) : entre parenthèses
  - Vous pouvez envoyer des paramètres à la fonction.
  - Ce sont des valeurs avec lesquelles la fonction va travailler.
- **Vous pouvez envoyer autant de paramètres que vous le voulez.**
- **Vous pouvez aussi n'envoyer aucun paramètre à la fonction**

38

## Schéma d'une fonction

12

- **Exemple:** une fonction triple

39

## Schéma d'une fonction

12

- **Exemple:** une fonction triple
  - vous envoyez un nombre en paramètre.

40

## Schéma d'une fonction

12

### **Exemple:** une fonction triple

- vous envoyez un nombre en paramètre.
- La fonction « récupère » ce nombre et en calcule le triple, en le multipliant par 3.

41

## Schéma d'une fonction

12

### **Exemple:** une fonction triple

- vous envoyez un nombre en paramètre.
- La fonction « récupère » ce nombre et en calcule le triple, en le multipliant par 3.
- Elle renvoie ensuite le résultat de ses calculs.

42

## Schéma d'une fonction

12

### **Exemple:** une fonction triple

- vous envoyez un nombre en paramètre.
- La fonction « récupère » ce nombre et en calcule le triple, en le multipliant par 3.
- Elle renvoie ensuite le résultat de ses calculs.
- vous avez les accolades qui indiquent le début et la fin de la fonction.

43

## Schéma d'une fonction

12

### **Exemple:** une fonction triple

- vous envoyez un nombre en paramètre.
- La fonction « récupère » ce nombre et en calcule le triple, en le multipliant par 3.
- Elle renvoie ensuite le résultat de ses calculs.
- vous avez les accolades qui indiquent le début et la fin de la fonction.
- À l'intérieur de ces accolades vous mettrez les instructions que vous voulez.

44

## Schéma d'une fonction

12

**Exemple:** une fonction triple

- vous envoyez un nombre en paramètre.
- La fonction « récupère » ce nombre et en calcule le triple, en le multipliant par 3.
- Elle renvoie ensuite le résultat de ses calculs.
- vous avez les accolades qui indiquent le début et la fin de la fonction.
- À l'intérieur de ces accolades vous mettrez les instructions que vous voulez.
- Pour la fonction triple, il faudra taper des instructions qui multiplient par 3 le nombre reçu en entrée.

45

## Schéma d'une fonction

12

Une fonction, c'est donc un mécanisme qui reçoit des valeurs en entrée (les paramètres) et qui renvoie un résultat en sortie.

**Exemple:** une fonction triple

- vous envoyez un nombre en paramètre.
- La fonction « récupère » ce nombre et en calcule le triple, en le multipliant par 3.
- Elle renvoie ensuite le résultat de ses calculs.
- vous avez les accolades qui indiquent le début et la fin de la fonction.
- À l'intérieur de ces accolades vous mettrez les instructions que vous voulez.
- Pour la fonction triple, il faudra taper des instructions qui multiplient par 3 le nombre reçu en entrée.

46

## Créer une fonction

13

- Exemple:

47

## Créer une fonction

13

```
int triple(int nombre)
{
    int resultat = 0;
    resultat = 3 * nombre; // On multiplie le nombre fourni par 3
    return resultat; // On retourne la variable resultat qui vaut le triple de nombre
}
```

48

## Créer une fonction

13

```
int triple(int nombre)
{
    int resultat = 0;
    resultat = 3 * nombre; // On multiplie le nombre fourni par 3
    return resultat; // On retourne la variable resultat qui vaut le triple de nombre
}
```

- la fonction est de type **int**. Elle doit donc renvoyer une valeur de type **int**.

49

## Créer une fonction

13

```
int triple(int nombre)
{
    int resultat = 0;
    resultat = 3 * nombre; // On multiplie le nombre fourni par 3
    return resultat; // On retourne la variable resultat qui vaut le triple de nombre
}
```

- la fonction est de type **int**. Elle doit donc renvoyer une valeur de type **int**.
- Entre les parenthèses, vous avez les variables que la fonction reçoit.

50

## Créer une fonction

13

```
int triple(int nombre)
Ex {
    int resultat = 0;
    resultat = 3 * nombre; // On multiplie le nombre fourni par 3
    return resultat; // On retourne la variable resultat qui vaut le triple de nombre
}
```

- la fonction est de type **int**. Elle doit donc renvoyer une valeur de type **int**.
- Entre les parenthèses, vous avez les variables que la fonction reçoit.
- La fonction triple reçoit une variable de type **int** appelée **nombre**.

51

## Créer une fonction

13

```
int triple(int nombre)
Ex {
    int resultat = 0;
    resultat = 3 * nombre; // On multiplie le nombre fourni par 3
    return resultat; // On retourne la variable resultat qui vaut le triple de nombre
}
```

- la fonction est de type **int**. Elle doit donc renvoyer une valeur de type **int**.
- Entre les parenthèses, vous avez les variables que la fonction reçoit.
- La fonction triple reçoit une variable de type **int** appelée **nombre**.
- La ligne qui donne pour consigne de « renvoyer une valeur » est celle qui contient le **return**.

52

## Créer une fonction

13

```
int triple(int nombre)
{
    int resultat = 0;
    resultat = 3 * nombre; // On multiplie le nombre fourni par 3
    return resultat; // On retourne la variable resultat qui vaut le triple de nombre
}
```

- la fonction est de type **int**. Elle doit donc renvoyer une valeur de type **int**.
- Entre les parenthèses, vous avez les variables que la fonction reçoit.
  - La fonction triple reçoit une variable de type **int** appelée **nombre**.
- La ligne qui donne pour consigne de « renvoyer une valeur » est celle qui contient le **return**.
  - Cette ligne se trouve généralement à la fin de la fonction, après les calculs.

53

## Créer une fonction

14

```
return resultat;
```

54

## Créer une fonction

14

```
return resultat;
```

- Ce code signifie pour la fonction : « **Arrête-toi là et renvoie le nombre resultat** ».

55

## Créer une fonction

14

```
return resultat;
```

- Ce code signifie pour la fonction : « **Arrête-toi là et renvoie le nombre resultat** ».
- Cette variable **resultat** DOIT être de type **int**, car la fonction renvoie un **int** comme on l'a dit plus haut.

56

## Créer une fonction

14

```
return résultat;
```

- Ce code signifie pour la fonction : « **Arrête-toi là et renvoie le nombre résultat** ».
- Cette variable **résultat** DOIT être de type **int**, car la fonction renvoie un **int** comme on l'a dit plus haut.
- La variable **résultat** est déclarée (= créée) dans la fonction triple.

57

## Créer une fonction

14

```
return résultat;
```

- Ce code signifie pour la fonction : « **Arrête-toi là et renvoie le nombre résultat** ».
- Cette variable **résultat** DOIT être de type **int**, car la fonction renvoie un **int** comme on l'a dit plus haut.
- La variable **résultat** est déclarée (= créée) dans la fonction triple.
- Cela signifie qu'elle n'est utilisable que dans cette fonction, et pas dans une autre comme la fonction main par exemple.

58

## Créer une fonction

14

```
return résultat;
```

- Ce code signifie pour la fonction : « **Arrête-toi là et renvoie le nombre résultat** ».
- Cette variable **résultat** DOIT être de type **int**, car la fonction renvoie un **int** comme on l'a dit plus haut.
- La variable **résultat** est déclarée (= créée) dans la fonction **triple**.
- Cela signifie qu'elle n'est utilisable que dans cette fonction, et pas dans une autre comme la fonction **main** par exemple.
- C'est donc une variable propre à la fonction **triple**.

59

## Créer une fonction

15

- la façon la plus courte d'écrire notre fonction **triple**

60

## Créer une fonction

15

- la façon la plus courte d'écrire notre fonction **triple**

```
int triple(int nombre)
{
    return 3 * nombre;
}
```

61

## Plusieurs paramètres

16

- il est possible de créer des fonctions acceptant plusieurs paramètres.

62

## Plusieurs paramètres

16

- il est possible de créer des fonctions acceptant plusieurs paramètres.

```
int addition(int a, int b)
{
    return a + b;
}
```

63

## Plusieurs paramètres

16

- il est possible de créer des fonctions acceptant plusieurs paramètres.

```
int addition(int a, int b)
{
    return a + b;
}
```

- Il suffit de séparer les différents paramètres par une virgule.

64

## Aucun paramètre

17

- Ces fonctions feront généralement toujours la même chose.

```
void bonjour()
{
    printf("Bonjour");
}
```

65

## Aucun paramètre

17

- Ces fonctions feront généralement toujours la même chose.
- Ces fonctions serviront juste à effectuer certaines actions, comme afficher du texte à l'écran.

```
void bonjour()
{
    printf("Bonjour");
}
```

66

## Aucun paramètre

17

- Ces fonctions feront généralement toujours la même chose.
- Ces fonctions serviront juste à effectuer certaines actions, comme afficher du texte à l'écran.
- ce sera forcément toujours le même texte puisque la fonction ne reçoit aucun paramètre susceptible de modifier son comportement !

```
void bonjour()
{
    printf("Bonjour");
}
```

67

## Aucun paramètre

17

- Ces fonctions feront généralement toujours la même chose.
- Ces fonctions serviront juste à effectuer certaines actions, comme afficher du texte à l'écran.
- ce sera forcément toujours le même texte puisque la fonction ne reçoit aucun paramètre susceptible de modifier son comportement !
- **Exemple:** une fonction bonjour qui affiche juste « Bonjour » à l'écran :

```
void bonjour()
{
    printf("Bonjour");
}
```

68

## Aucun paramètre

17

- Ces fonctions feront généralement toujours la même chose.
- Ces fonctions serviront juste à effectuer certaines actions, comme afficher du texte à l'écran.
  - ce sera forcément toujours le même texte puisque la fonction ne reçoit aucun paramètre susceptible de modifier son comportement !
  - **Exemple:** une fonction bonjour qui affiche juste « Bonjour » à l'écran :
  - la fonction ne prend aucun paramètre.

```
void bonjour()
{
    printf("Bonjour");
}
```

69

## Aucun paramètre

17

- Ces fonctions feront généralement toujours la même chose.
- Ces fonctions serviront juste à effectuer certaines actions, comme afficher du texte à l'écran.
  - ce sera forcément toujours le même texte puisque la fonction ne reçoit aucun paramètre susceptible de modifier son comportement !
  - **Exemple:** une fonction bonjour qui affiche juste « Bonjour » à l'écran :
  - la fonction ne prend aucun paramètre.
  - le type **void**

```
void bonjour()
{
    printf("Bonjour");
}
```

70

## Aucun paramètre

17

- Ces fonctions feront généralement toujours la même chose.
- Ces fonctions serviront juste à effectuer certaines actions, comme afficher du texte à l'écran.
  - ce sera forcément toujours le même texte puisque la fonction ne reçoit aucun paramètre susceptible de modifier son comportement !
  - **Exemple:** une fonction bonjour qui affiche juste « Bonjour » à l'écran :
    - la fonction ne prend aucun paramètre.
    - le type **void**
    - la fonction n'a pas non plus de **return**

```
void bonjour()
{
    printf("Bonjour");
}
```

71

## Aucun paramètre

17

- Ces fonctions feront généralement toujours la même chose.
- Ces fonctions serviront juste à effectuer certaines actions, comme afficher du texte à l'écran.
  - ce sera forcément toujours le même texte puisque la fonction ne reçoit aucun paramètre susceptible de modifier son comportement !
  - **Exemple:** une fonction bonjour qui affiche juste « Bonjour » à l'écran :
    - la fonction ne prend aucun paramètre.
    - le type **void**
    - la fonction n'a pas non plus de **return**
    - Elle ne retourne rien. Une fonction qui ne retourne rien est de type **void**.

```
void bonjour()
{
    printf("Bonjour");
}
```

72

## Appeler une fonction

18

- écrire la fonction **triple** AVANT la fonction **main**.

73

## Appeler une fonction

18

- écrire la fonction **triple** AVANT la fonction **main**.
  - ▣ Si vous la placez après, ça ne marchera pas.

74

## Appeler une fonction

18

- écrire la fonction **triple** AVANT la fonction **main**.
  - Si vous la placez après, ça ne marchera pas.
  - **Exemple:**

75

## Appeler une fonction

18

- écrire la fonction **triple** AVANT la fonction **main**.
  - Si vous la placez après, ça n'
  - **Exemple:**

```
#include <stdio.h>
#include <stdlib.h>
int triple(int nombre)
{
    return 3 * nombre;
}
int main(int argc, char *argv[])
{
    int nombreEntre = 0, nombreTriple = 0;
    printf("Entrez un nombre... ");
    scanf("%d", &nombreEntre);
    nombreTriple = triple(nombreEntre);
    printf("Le triple de ce nombre est %d\n", nombreTriple);
    return 0;
}
```

76

## Appeler une fonction

19

- l'appel de la fonction :

```
nombreTriple = triple(nombreEntre);
```

77

## Explications sous forme de schéma

20

- dans quel ordre le code est lu.

78

## Explications sous forme de schéma

20

- dans quel ordre le code est lu.
- lire la ligne numérotée 1, puis 2, puis 3...

79

## Explications sous forme de schéma

20

- dans quel ordre le code est lu.
- lire la ligne numérotée 1, puis 2, puis 3...

```

#include <stdio.h>
#include <stdlib.h>
int triple(int nombre) // 6
{
    return 3 * nombre; // 7
}
int main(int argc, char *argv[]) // 1
{
    int nombreEntre = 0, nombreTriple = 0; // 2
    printf("Entrez un nombre... "); // 3
    scanf("%d", &nombreEntre); // 4
    nombreTriple = triple(nombreEntre); // 5
    printf("Le triple de ce nombre est %d\n", nombreTriple); // 8
    return 0; // 9
}

```

80

## Explications sous forme de schéma

21

1. Le programme commence par la fonction **main**.

81

## Explications sous forme de schéma

21

1. Le programme commence par la fonction **main**.
2. Il lit les instructions dans la fonction une par une dans l'ordre.

82

## Explications sous forme de schéma

21

1. Le programme commence par la fonction **main**.
2. Il lit les instructions dans la fonction une par une dans l'ordre.
3. Il lit l'instruction suivante et fait ce qui est demandé (**printf**).

83

## Explications sous forme de schéma

21

1. Le programme commence par la fonction **main**.
2. Il lit les instructions dans la fonction une par une dans l'ordre.
3. Il lit l'instruction suivante et fait ce qui est demandé (**printf**).
4. De même, il lit l'instruction et fait ce qui est demandé (**scanf**).

84

## Explications sous forme de schéma

21

1. Le programme commence par la fonction **main**.
2. Il lit les instructions dans la fonction une par une dans l'ordre.
3. Il lit l'instruction suivante et fait ce qui est demandé (**printf**).
4. De même, il lit l'instruction et fait ce qui est demandé (**scanf**).
5. Il lit l'instruction... On appelle la fonction **triple**, on doit donc sauter à la ligne de la fonction **triple** plus haut.

85

## Explications sous forme de schéma

22

6. On saute à la fonction **triple** et on récupère un paramètre (**nombre**).

86

## Explications sous forme de schéma

22

6. On saute à la fonction **triple** et on récupère un paramètre (**nombre**).
7. On fait des calculs sur le nombre et on termine la fonction. **return** signifie la fin de la fonction et permet d'indiquer le résultat à renvoyer.

87

## Explications sous forme de schéma

22

6. On saute à la fonction **triple** et on récupère un paramètre (**nombre**).
7. On fait des calculs sur le nombre et on termine la fonction. **return** signifie la fin de la fonction et permet d'indiquer le résultat à renvoyer.
8. On retourne dans le **main** à l'instruction suivante.

88

## Explications sous forme de schéma

22

6. On saute à la fonction **triple** et on récupère un paramètre (**nombre**).
7. On fait des calculs sur le nombre et on termine la fonction. **return** signifie la fin de la fonction et permet d'indiquer le résultat à renvoyer.
8. On retourne dans le **main** à l'instruction suivante.
9. Un **return** ! La fonction **main** se termine et donc le programme est terminé.

89

## Explications sous forme de schéma

23

2) La fonction **triple** retourne (**return**) une valeur.  
Cette valeur, c'est  $3 \times$  le nombre qu'on lui a envoyé.

Cette valeur de retour est stockée dans la variable **nombreTriple** de la fonction **main**. Le signe « = » permet donc de dire « Envoie le résultat de la fonction dans cette variable ».

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

int triple(int nombre)
{
    return 3 * nombre;
}

int main(int argc, char *argv[])
{
    int nombreEntre = 0, nombreTriple = 0;

    printf("Entrez un nombre... ");
    scanf("%d", &nombreEntre);

    nombreTriple = triple(nombreEntre);
    printf("Le triple de ce nombre est %d\n", nombreTriple);

    return 0;
}
```

1) La variable **nombreEntre** est envoyée en paramètre à la fonction **triple**. Celle-ci récupère cette variable dans une autre variable qui s'appelle « **nombre** ».

Note : on aurait aussi pu mettre le même nom de variable dans les 2 fonctions. Il n'y aurait pas eu de conflit, car une variable appartient à sa fonction.

90

## Testez le programme

24

Entrez un nombre... 10  
Le triple de ce nombre est 30

91

## Testez le programme

24

Entrez un nombre... 10  
Le triple de ce nombre est 30

- Vous n'êtes pas obligés de stocker le résultat d'une fonction dans une variable !

92

## Testez le programme

24

Entrez un nombre... 10  
Le triple de ce nombre est 30

- Vous n'êtes pas obligés de stocker le résultat d'une fonction dans une variable !

```
#include <stdio.h>
#include <stdlib.h>
int triple(int nombre)
{
    return 3 * nombre;
}
int main(int argc, char *argv[])
{
    int nombreEntre = 0;
    printf("Entrez un nombre... ");
    scanf("%d", &nombreEntre);
    // Le résultat de la fonction est directement envoyé au printf et n'est pas stocké dans une variable
    printf("Le triple de ce nombre est %d\n", triple(nombreEntre));
    return 0;
}
```

93

## Exemple: Conversion euros / MAD

25

94

## Exemple: Conversion euros / MAD

25

- créer une fonction appelée conversion.

95

## Exemple: Conversion euros / MAD

25

- créer une fonction appelée conversion.
- 1 euro = 10.70185 MAD

96

## Exemple: Conversion euros / MAD

25

- créer une fonction appelée conversion.
  - ▣ 1 euro = 10.70185 MAD
  - ▣ cette fonction prend une variable en entrée de type double et retourne une sortie de type double

97

## Exemple: Conversion euros / MAD

25

- créer une fonction appelée conversion.
  - ▣ 1 euro = 10.70185 MAD
  - ▣ cette fonction prend une variable en entrée de type double et retourne une sortie de type double
  - ▣ manipuler des nombres décimaux

98

## Exemple: Conversion euros / MAD

25

- créer une fonction appelée conversion.

- 1 euro = 10.70185 MAD
- cette fonction prend une variable en entrée et renvoie une variable de sortie de type double
- manipuler des nombres décimaux

```
double conversion(double euros)
{
    double mad = 0;
    mad = 10.70185 * euros;
    return mad;
}

int main(int argc, char *argv[])
{
    printf("10 euros = %fDH\n", conversion(10));
    printf("50 euros = %fDH\n", conversion(50));
    printf("100 euros = %fDH\n", conversion(100));
    printf("200 euros = %fDH\n", conversion(200));
    return 0;
}
```

99

## Exemple: Conversion euros / MAD

26

- Écrivez une seconde fonction qui fera la conversion inverse : MAD => Euros.

100

## Exemple: La punition

27

- une fonction qui ne renvoie rien (pas de sortie).

101

## Exemple: La punition

27

- une fonction qui ne renvoie rien (pas de sortie).
- une fonction qui affiche le même message à l'écran autant de fois qu'on lui demande.

102

## Exemple: La punition

27

- une fonction qui ne renvoie rien (pas de sortie).
- une fonction qui affiche le même message à l'écran autant de fois qu'on lui demande.
  - ▣ cette fonction prend un paramètre en entrée : le nombre de fois où il faut afficher la punition.

103

## Exemple: La punition

27

- une fonction qui ne renvoie rien (pas de sortie).
- une fonction qui affiche le même message à l'écran autant de fois qu'on lui demande.
  - ▣ cette fonction prend un p  
la punition.

```
void punition(int nombreDeLignes)
{
    int i;
    for (i = 0 ; i < nombreDeLignes ; i++)
    {
        printf("Je ne dois pas recopier mon voisin\n");
    }
}
int main(int argc, char *argv[])
{
    punition(10);
    return 0;
}
```

104

## Exemple: Aire d'un rectangle

28

- Une fonction nommée aireRectangle va prendre deux paramètres : la largeur et la hauteur. Elle renverra l'aire.

105

## Exemple: Aire d'un rectangle

28

- Une fonction nommée aireRectangle va prendre deux paramètres : la largeur et la hauteur. Elle renverra l'aire.

```
double aireRectangle(double largeur, double hauteur)
{
    return largeur * hauteur;
}
int main(int argc, char *argv[])
{
    printf("Rectangle de largeur 5 et hauteur 10. Aire = %f\n", aireRectangle(5, 10));
    printf("Rectangle de largeur 2.5 et hauteur 3.5. Aire = %f\n", aireRectangle(2.5, 3.5));
    printf("Rectangle de largeur 4.2 et hauteur 9.7. Aire = %f\n", aireRectangle(4.2, 9.7));
    return 0;
}
```

106

## Exemple: Aire d'un rectangle

29

- afficher directement la largeur, la hauteur et l'aire dans la fonction

107

## Exemple: Aire d'un rectangle

29

- afficher directement la largeur, la hauteur et l'aire dans la fonction

```
void aireRectangle(double largeur, double hauteur)
{
    double aire = 0;
    aire = largeur * hauteur;
    printf("Rectangle de largeur %f et hauteur %f. Aire = %f\n", largeur, hauteur, aire);
}
int main(int argc, char *argv[])
{
    aireRectangle(5, 10);
    aireRectangle(2.5, 3.5);
    aireRectangle(4.2, 9.7);
    return 0;
}
```

108

## Exemple: Un menu

30

- une fonction menu() qui ne prend aucun paramètre en entrée.
- cette fonction se contente d'afficher le menu et demande à l'utilisateur de faire un choix.
- La fonction renvoie le choix de l'utilisateur.

109

```

E 30
int menu() {
    int choix = 0;
    while (choix < 1 || choix > 4)
    {
        printf("Menu :\n");
        printf("1 : Poulet de dinde aux escargots rotis a la sauce bearnaise\n");
        printf("2 : Concombres sures a la sauce de myrtilles enrobees de chocolat\n");
        printf("3 : Escalope de kangourou sanguinante et sa gelee aux fraises poivree\n");
        printf("4 : La surprise du Chef (j'en salive d'avance...)\n");
        printf("Votre choix ? ");
        scanf("%d", &choix);
    }
    return choix;
}
int main(int argc, char *argv[])
{
    switch (menu())
    {
        case 1:
            printf("Vous avez pris le poulet\n"); break;
        case 2:
            printf("Vous avez pris les concombres\n"); break;
        case 3:
            printf("Vous avez pris l'escalope\n"); break;
        case 4:
            printf("Vous avez pris la surprise du Chef. Vous etes un sacre aventurier dites donc !\n"); break;
    }
    return 0;
}

```

110