Introduction au C/C++

Cours 1

Rémi Watrigant
(fortement inspiré du cours de V. Boudet, P. Giorgi et M. Joab de l'Université Montpellier 2)

Université de Nîmes

2013-2014

Plan

Prévision d'organisation :

- des **cours**
- des (TD) **TP**, dont 2 ou 3 notés (note de contrôle continu)
- 1 partiel (+ 1 rattrapage...)

Evaluation :

- 1 note de contrôle continu (moyenne des 2 ou 3 TP notés)
- 1 note d'examen terminal

Note finale (à priori) :

$$0.3 * CC + 0.6 * Exam$$

Objectifs :

- découvrir le langage C
- revoir les principes de la programmation impérative (que vous avez déjà vu en Pascal...)
- les adapter au C
- introduction aux principes de l'orienté objet en C++

Plan

Plan

assembleur : difficile

Un ordinateur comprend un seul langage : l'assembleur.

Exemple

Un programme assembleur : exemple1.s

```
1  .globl _main
2  _main:
3          pushl   %ebp
4          movl    %esp, %ebp
5          subl    $24, %esp
6          movl    $10, -12(%ebp)
7          leave
8          ret
9          .subsections_via_symbols
```

Lisibilité

Mais cette langue est difficile à apprendre pour nous.
Il nous faut donc un langage intermédiaire entre la langue naturelle et l'assembleur : un langage de haut niveau.

Exemple

L'exemple équivalent en C : exemple1.c

```
1 int main()
2 {
3     int a;
4     a=10;
5 }
```

Portabilité

L'assembleur est propre à chaque famille de processeurs. Il nous faudrait donc un programme différent sur chaque ordinateur. D'où la nécessité de **portabilité**.
On demandera donc à notre langage de haut niveau d'exister de manière identique sur toutes les architectures d'ordinateurs.

Plan

Plusieurs paradigmes de langages de programmation

langages fonctionnels : CAML, LISP

langages impératifs : C, Pascal, Fortran

langages à objets : C++, JAVA

langages logiques : Prolog

langages à balises : Html

Certains ont plusieurs caractéristiques.

Quelles sont les différences ?

- on ne programme pas de la même façon dans ces différents langages
- un programme écrit en CAML ne "fonctionne" pas en C
- les concepts théoriques à la base des langages sont différents
- Et pourtant, deux programmes écrits dans des langages différents peuvent rendre un même service à son utilisateur.

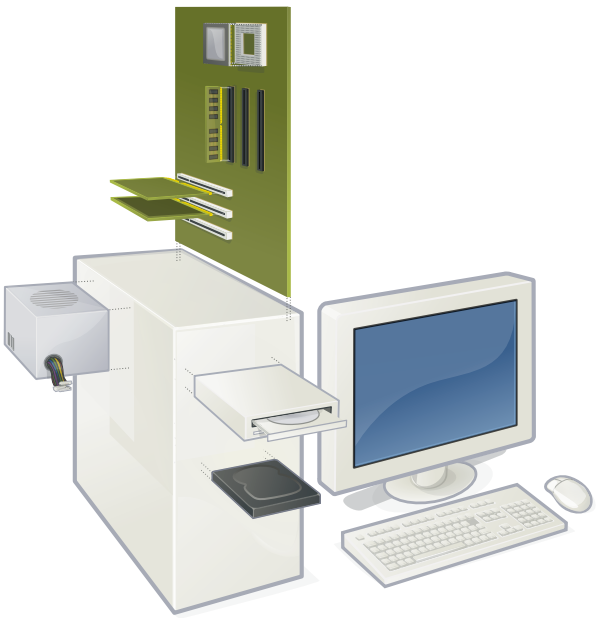
Pourquoi alors avoir plusieurs langages ?

- il est parfois plus facile de programmer certains logiciels avec certains langages et plus difficiles avec d'autres
- certains langages sont (pour certains usages)
  - plus "lents" que d'autres
  - moins "portables" que d'autres
- parfois, nous n'avons pas le choix (intégration, utilisation de bibliothèques).

Contenu d'un ordinateur

Entre autres choses, un ordinateur contient :

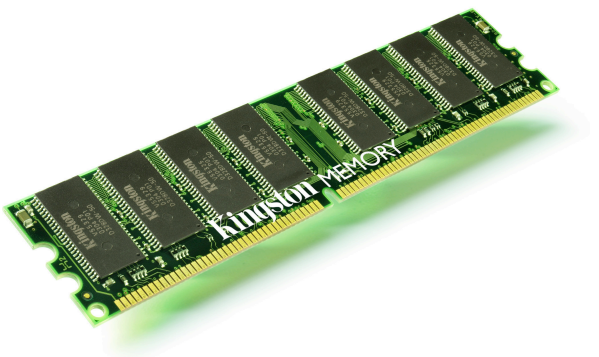
- Une mémoire
- Des unités de calcul sur les entiers
- Des unités de calcul sur les réels



Mémoire d'un ordinateur

Comment est organisée la mémoire d'un ordinateur ?

- Mémoire divisée en case (bit)
- Case identifiée par un numéro (adresse)



On peut donc voir la mémoire d'un ordinateur comme un tableau.

Mémoire d'un ordinateur

Comment est organisée la mémoire d'un ordinateur ?

A partir de maintenant, on représentera la mémoire d'un ordinateur comme un grand tableau.

Contenu de la mémoire

La mémoire d'un ordinateur (comme tout composant électronique) ne possède que deux états : 0 ou 1.
Tous les objets manipulés, des plus simples (booléens, entiers) aux plus complexes (un labyrinthe, une carte routière, un film) sont traduits en 0 et en 1.

Exemple

La représentation en mémoire de 63576473 est

$$00000011110010100001100110011001$$

celle de $3,141592$ peut-être

$$11001001000011111101101010100010001000010110100011000$$

Contenu de la mémoire

Que remarque-t-on sur les exemples précédents : la place en mémoire dépend de ce qu'on souhaite représenter
Pour tous les objets que nous manipulerons, il nous faudra préciser de quoi il s'agit : on parlera de type
Les types que nous détaillerons plus loin seront : `int`, `float`, `double`, `char`...

Contenu de la mémoire

Posons nous la question inverse : supposons que la mémoire contienne les bits suivants :

$$00100001011011100110100101100110$$

On voudrait afficher de manière lisible ce que cela représente.

- l'entier 560884070

Contenu de la mémoire

Posons nous la question inverse : supposons que la mémoire contienne les bits suivants :

$$00100001011011100110100101100110$$

On voudrait afficher de manière lisible ce que cela représente.

- l'entier 560884070
- le réel 0.000000000000000008077703024915880348382542924801 1

Contenu de la mémoire

Posons nous la question inverse : supposons que la mémoire contienne les bits suivants :

$$00100001011011100110100101100110$$

On voudrait afficher de manière lisible ce que cela représente.

- l'entier 560884070
- le réel 0.000000000000000008077703024915880348382542924801 1
- les caractères "FIN !"

Lors de l'affichage, il faudrait également préciser la nature des objets : on parlera plus tard de format.

Programmation impérative

Le programmeur décrit dans son programme :

- ce que doit faire la machine "pas à pas"
- dans quel ordre le faire

Le programmeur dispose d'un certain nombre d'instructions pour donner ses ordres à la machine :

- des instructions de contrôles du déroulement du programme
- des instructions de manipulations de la mémoire
  - en donnant un nom à une zone mémoire : variable
  - en accédant directement à une case mémoire : pointeur
  - puis en modifiant ces objets par des affectations

Plan

Histoire

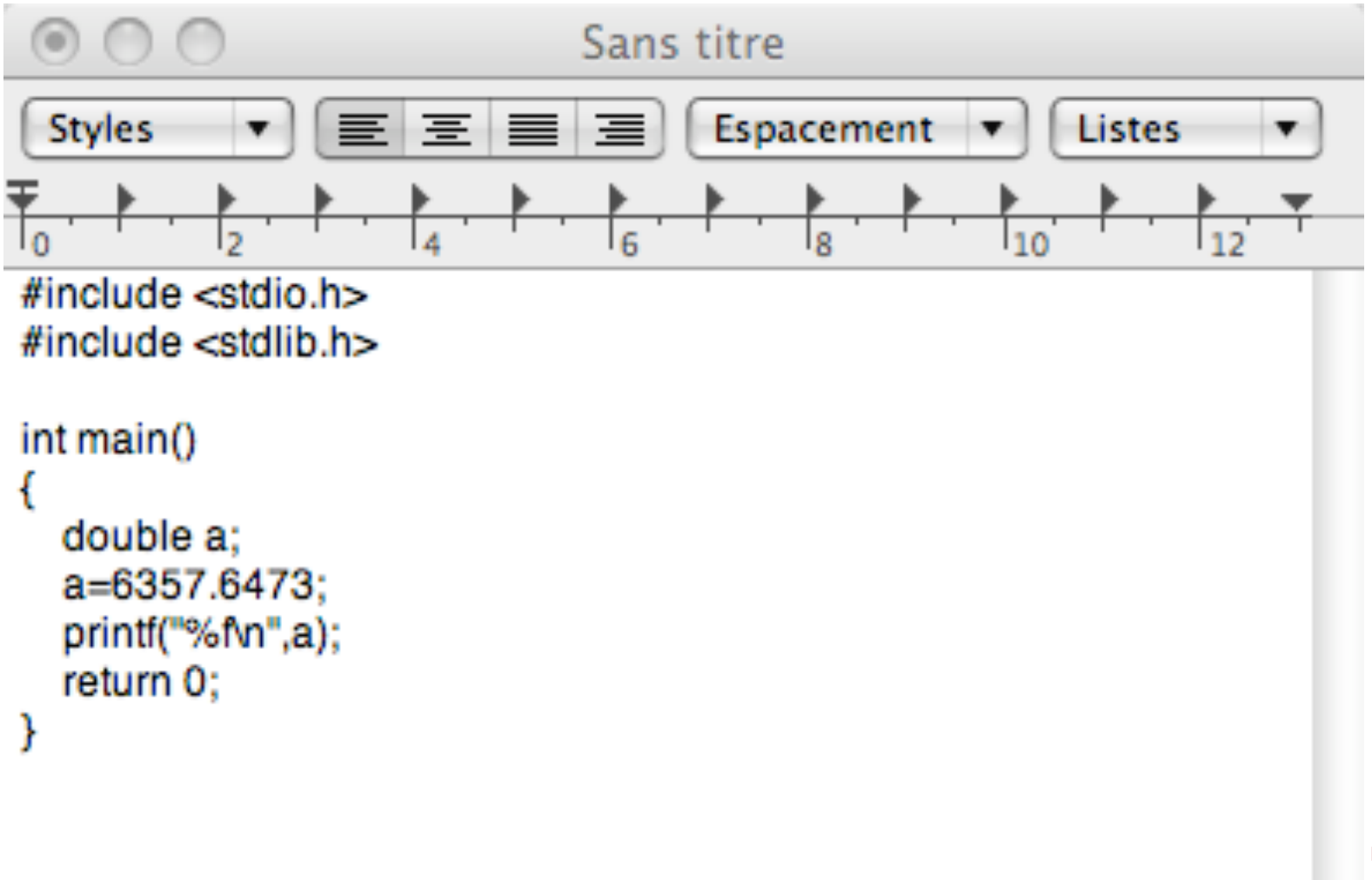
- apparu au cours de l'année 1972 dans les Laboratoires Bell
- développé en même temps que Unix par Dennis Ritchie et Ken Thompson
- En 1989, normalisation du langage C par l'Institut national américain de normalisation (ANSI) ANSI C ou C89
- en 1990 norme également adoptée par l'Organisation internationale de normalisation (C ISO)

Importance

- la majorité des grands systèmes d'exploitation entre les années 1975 et 1993 ont été développés en C.
- extrêmement utilisé dans des domaines comme
  - la programmation embarquée sur microcontrôleurs,
  - les calculs intensifs,
  - l'écriture de systèmes d'exploitation
  - tous les modules où la rapidité de traitement est importante
- a inspiré de nombreux langages récents : C++, Java, Javascript, PHP...

Exemple de programme C

Voici un exemple de programme C écrit dans un éditeur de texte quelconque.



Plan

Le C : un langage compilé

Trois étapes avant d'avoir un programme utilisable :

1. Ecrire le code source dans un éditeur de texte (fichier en langage C)
2. Compiler ce code source
3. Executer le produit de la compilation qui est un programme exécutable

A quoi sert la compilation ?

- Transformer un code source lisible par un être humain en un langage exécutable par une machine
- Les programmes que vous écrirez ne sont pas directement exécutables
- Le résultat de la compilation est soit :
  - un code exécutable...
  - des messages d'erreurs (qu'il faut lire).

A quoi sert la compilation (2) ?

La compilation sert à faire le lien entre votre programme et d'autres codes sources :

- un programme peut être composé de plusieurs codes sources
- un programme peut utiliser des bibliothèques
  - bouts de programme déjà faits
  - réutilisation de vos programmes

Un exemple de Compilation

Pour compiler l'exemple précédent (exemple1.c), on tape dans un terminal :

Exemple

```
gcc -Wall -o Exemple1 exemple1.c
```

Cette instruction compile mon programme C et crée un fichier exécutable nommé Exemple1.
Pour tester le programme, on tape dans un terminal

Exemple

```
./Exemple1
```

Plan

Les erreurs

Très vaste sujet...
Principalement, deux types d'erreurs :

- Erreur de programmation
- Erreur de conception

Parfois les deux en même temps

Erreurs de programmation

- il suffit de réparer ces "erreurs" et de continuer
- exemples :
  - erreur de syntaxe (comme une faute d'orthographe)
  - appeler une fonction qui n'existe pas
  - mauvaise gestion de la mémoire (ex : accès à la case 1000 d'un tableau de 5 cases)
  - et encore pleins d'autres...

Erreurs de conceptions

- le programme peut compiler mais ne pas donner le résultat voulu
- erreur dans l'élaboration de la méthode
- exemples :
  - division par zéro
  - boucle infinie
  - pas du tout le résultat voulu (...)
- $\Rightarrow$ besoin de tests

Que fait le compilateur en cas d'erreurs ?

Le compilateur

- signale certaines erreurs de programmations (mais pas toutes ! ! !)
- ne signale aucune erreur de conception :
  - la machine fait ce que vous lui dites de faire
  - elle ne sait pas que vous vous trompez de méthode
  - la machine ne fait qu'exécuter vos ordres : c'est vous le chef

Exemple

Prenons le programme suivant : erreur.c

```
1 int main()
2 {
3 printf("Bonjour le monde\n")
4 return 0;
5 }
```

Exemple

On le compile : gcc -Wall -o Erreur erreur.c

```
1 erreur.c: In function ?main?:
2 erreur.c:3: error: syntax error before ?return?
3 erreur.c:4: warning: control reaches end of non-void
4 function
```

A la lecture des erreurs, on comprend qu'il y a une erreur avant le `return`.
Effectivement, il manque un ';'.

Les erreurs à l'exécution

- ce peut être des erreurs de programmation et/ou de conception
- non détectées à la compilation
- plus sournoisement :
  - le programme peut marcher correctement pendant très longtemps et produire soudainement une erreur
  - Explication possible : un cas très rare a eu lieu
  - Exemple : une division par 0

Exemple

Prenons le programme suivant : erreur2.c

```
1 int main()
2 {
3 int a=13;
4 int b=0;
5 int c;
6 c=a/b;
7 printf("%d\n",c);
8 return 0;
9 }
```

La compilation ne détecte aucune erreur, mais si on exécute le programme, on obtient le message suivant :

Floating point exception

Plan

Qu'est-ce qu'un IDE

Nous avons vu qu'il nous fallait pour programmer :

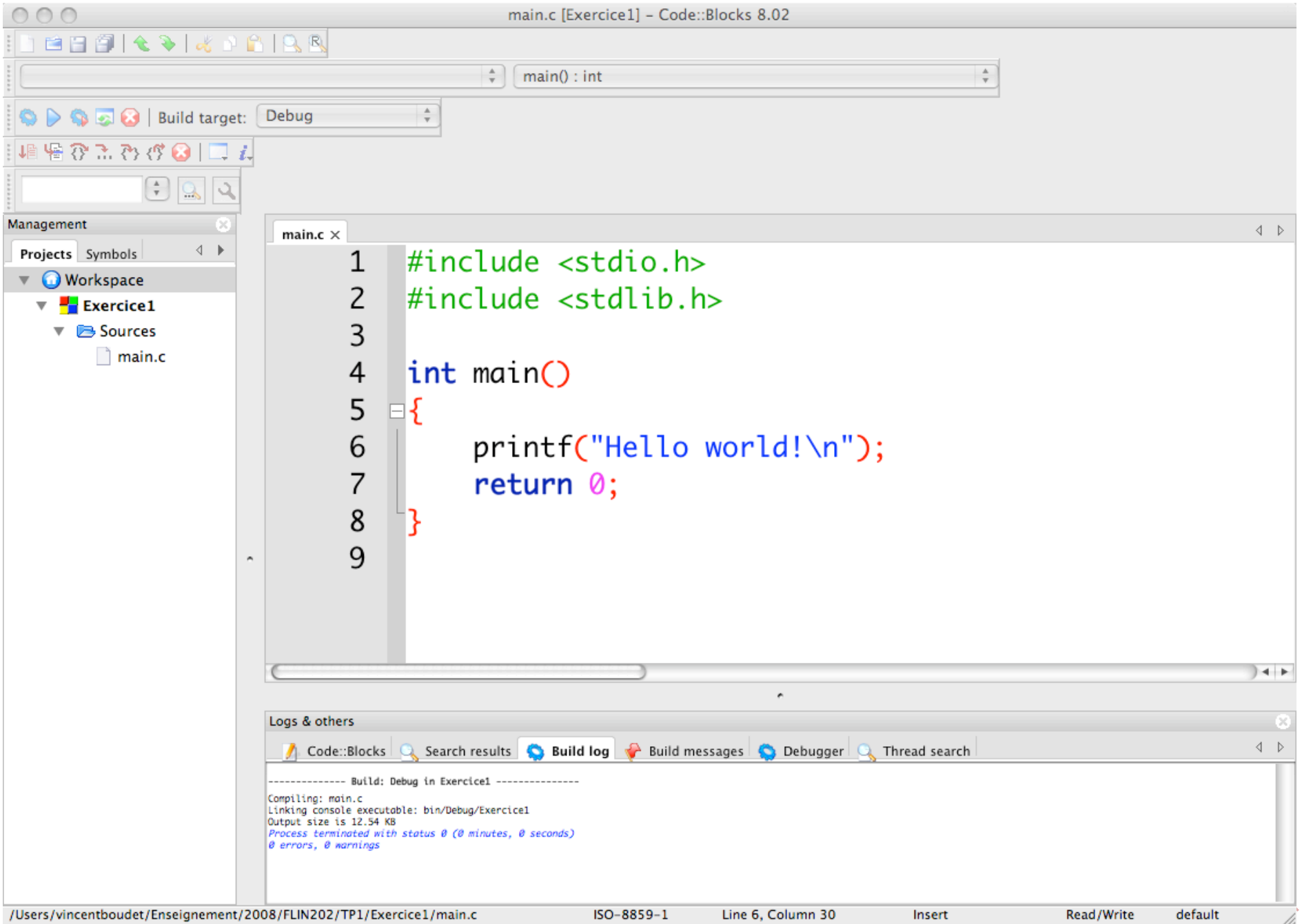
- un éditeur de texte
- un compilateur
- un outil pour gérer et traiter les erreurs

Un IDE (Integrated Development Environment, en français Environnement de développement intégré) contient tous ces éléments.

Code::Blocks

C'est un IDE disponible sur plusieurs systèmes (Windows, Linux, Mac OS), gratuit et complet.
Il est disponible là : *http ://www.codeblocks.org*

Regardons attentivement la capture d'écran :


main.c [Exercice1] - Code::Blocks 8.02
main() : int
Build target: Debug
Management
Projects
Symbols
Workspace
Exercice1
Sources
main.c
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6 printf("Hello world!\n");
7 return 0;
8 }
9
Logs & others
Code::Blocks
Search results
Build log
Build messages
Debugger
Thread search
---------- Build: Debug in Exercice1 ----------
Compiling: main.c
Linking console executable: bin/Debug/Exercice1
Output size is 12.54 KB
Process terminated with status 0 (0 minutes, 0 seconds)
0 errors, 0 warnings
/Users/vincentboudet/Enseignement/2008/FLIN202/TP1/Exercice1/main.c
ISO-8859-1
Line 6, Column 30
Insert
Read/Write
default