

Cours 1 - C

Imad Kissami

Université Mohammed VI Polytechnique - Licence S3

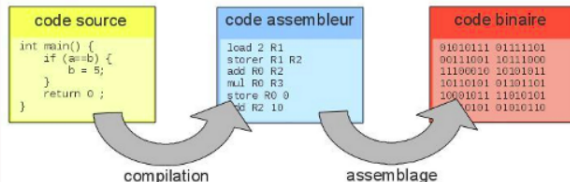
21 Octobre 2020

Historique

- Le **C** a été conçu en 1972 par Dennis Richie, au laboratoire Bell d'AT&T;
- Résultat de l'évolution de langages plus anciens.
 - **BPCL** développé en 1967 par Martin Richards;
 - **B** développé en 1970 chez AT&T par Ken Thompson.
- Langage compilé;
- Langage évolué qui permet néanmoins d'effectuer des opérations de bas niveau ("assembleur d'Unix");
- Grande efficacité et puissance;

Niveaux de langage de programmation

les niveaux de langage		
Niveau du langage de programmation	exemple de langage	exemple de code
le plus bas possible	le Binaire	0101101 00100101 11111010 00010111
bas niveau	Langage machine : Assembleur	mov eax, 39 add eax, 2 inc eax
haut niveau	Langage C	int main() { int x = 39; x = x+2; x++; }
le plus haut niveau	Le langage parlé	« prends le nombre 39 et ajoutes lui le nombre 2. Quand c'est fait, augmente le nombre obtenu de 1. »



Quel compilateur C ?

Pour développer des programmes, nous utiliserons un logiciel qui permet :

- d'éditer du code (taper le programme)
- de faire appel aux outils de compilation pour réaliser l'exécutable
- de déboguer le programme lors de son exécution

Un tel programme s'appelle un IDE : Integrated Development Environment.

Il existe de nombreux logiciels pour développer en langage C.

Quel compilateur C ?

Code::Blocks est l'IDE que nous avons choisi d'utiliser pour votre apprentissage de la programmation pour les raisons suivantes :

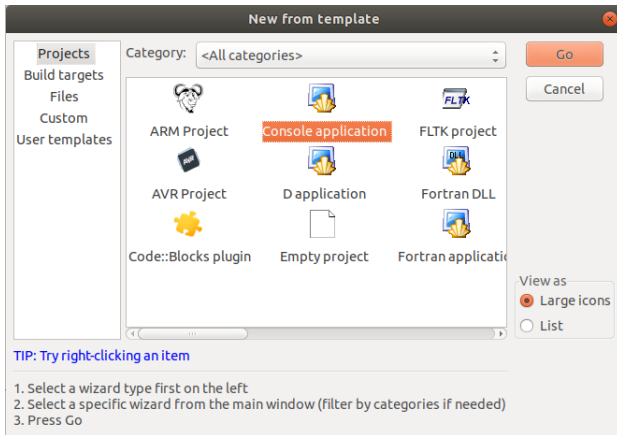
- il est libre et open source
- il est multi-plateformes (version pour Windows ou pour Linux et même pour MacOS)
- il est simple à installer
- il est de taille raisonnable (installateur de moins de 100 Mo avec les outils de compilation)
- il est simple à prendre en main
- il est performant

Le compilateur installé avec Code::Blocks est GNU GCC Compiler.

Premier programme

Créer un projet :

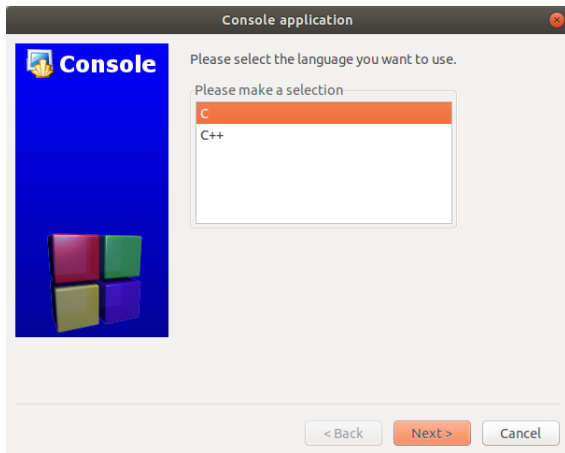
Un projet contient tous les éléments nécessaires pour compiler un programme.



Premier programme

Créer un projet :

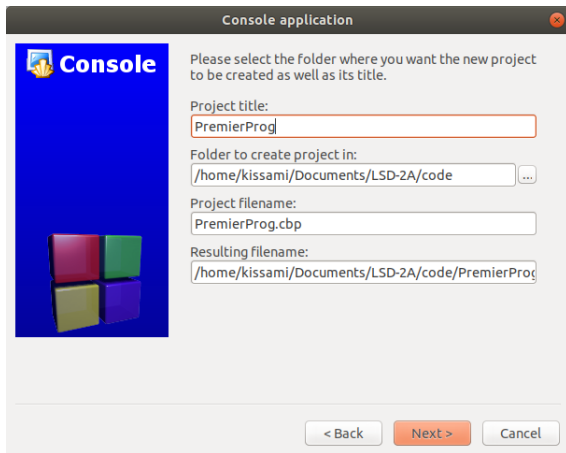
Un projet contient tous les éléments nécessaires pour compiler un programme.



Premier programme

Créer un projet :

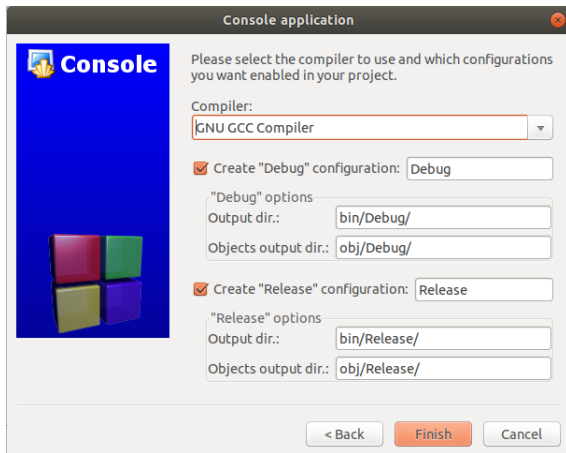
Un projet contient tous les éléments nécessaires pour compiler un programme.



Premier programme

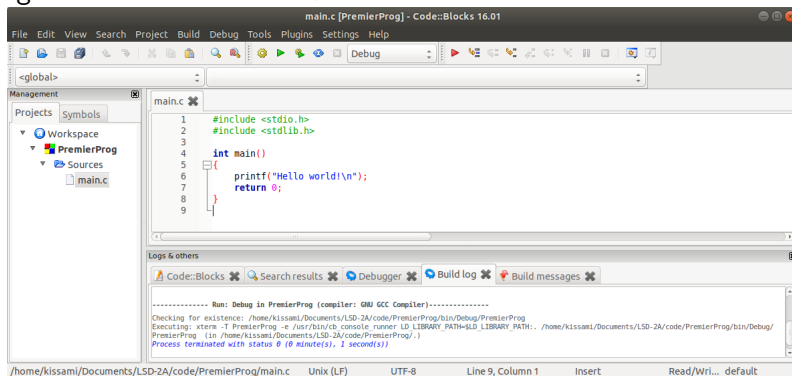
Créer un projet :

Un projet contient tous les éléments nécessaires pour compiler un programme.



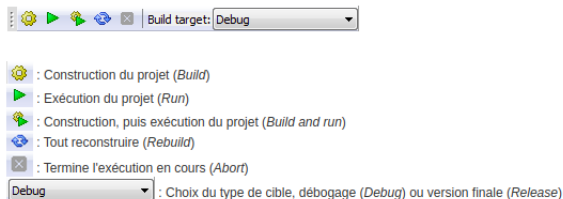
Construire et exécuter un projet

Vous pouvez éditer le fichier main.c en allant à gauche dans la fenêtre Management dans Sources — > main.c.



Construire et exécuter un projet

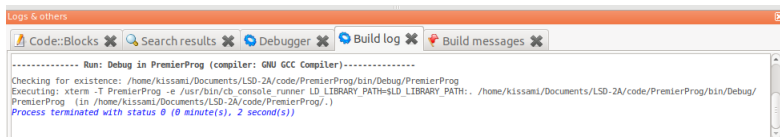
La barre d'outils Compiler :



La version finale donne un fichier exécutable plus petit et généralement plus efficace mais elle n'autorise pas le débogage.

Construire et exécuter un projet

On obtient l'affichage du résultat d'un Build dans l'onglet Build log de la fenêtre Messages.

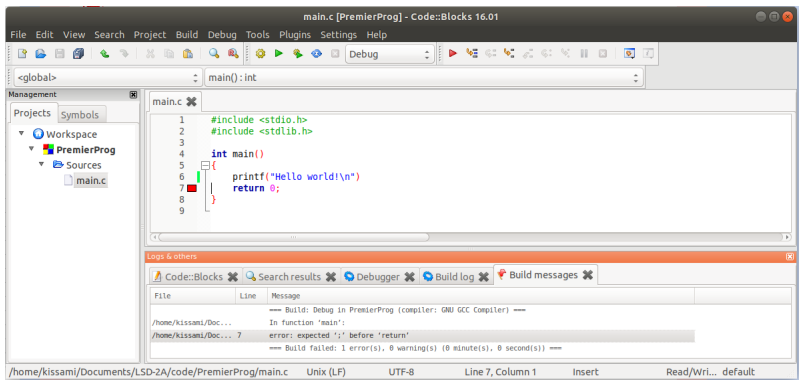


```
----- Run: Debug in PremierProg (compiler: GNU GCC Compiler)-----  
Checking for existence: /home/kissami/Documents/LSO-2A/code/PremierProg/bin/Debug/PremierProg  
Executing: xterm -T PremierProg -e /usr/bin/cb_console_runner LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.. /home/kissami/Documents/LSO-2A/code/PremierProg/bin/Debug/  
PremierProg (in /home/kissami/Documents/LSO-2A/code/PremierProg/.)  
Process terminated with status 0 (0 minute(s), 2 second(s))
```

Si on lance le programme (Run), une fenêtre s'ouvre et affiche le résultat de l'exécution.

Erreur à la compilation

Les erreurs lors de la compilation du projet sont recensées dans l'onglet Build messages de la fenêtre Messages.



L'absence de point-virgule apparaît comme une erreur. .











Débogueur

Utiliser le débogueur

Il est possible d'utiliser le débogueur soit à partir du menu Debug, soit grâce à la barre d'outils Debugger. On peut activer ou désactiver cette barre d'outils par View – > Toolbars – > Debugger.

La barre d'outils Debugger :

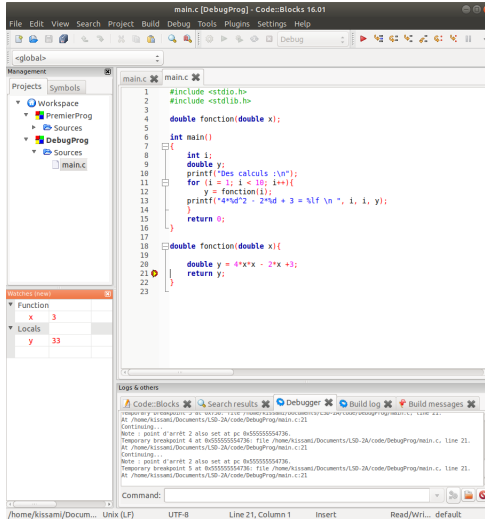


-  : Exécute en mode débogueur (*Debug/Continue*)
(Attention, si aucun point d'arrêt n'est placé, le programme s'exécute normalement).
-  : Exécute jusqu'au curseur placé dans le code (*Run to cursor*)
-  : Exécute une ligne sans entrer dans les fonctions (*Next line*)
-  : Exécute une ligne en entrant dans les fonctions (*Step into*)
-  : Exécute jusqu'à la fin de la fonction en cours (*Step out*)
-  : Exécute une instruction en assembleur (*Next instruction*)
-  : Interrompt l'exécution en cours (*Break debugger*)
-  : Termine l'exécution en cours (*Stop debugger*)
-  : Menu d'ouverture des fenêtres de débogage)
-  : Menu d'informations

L'absence de point-virgule apparaît comme une erreur. .

Débugueur

Utiliser le débogueur



Caractéristiques

- Toutes les **informations utiles** au programme ("**données**") sont mémorisées dans des **VARIABLES**, c'est-à-dire des emplacements mémoire accessibles en lecture et en écriture.
- L'emplacement mémoire d'une variable est créé (alloué) lors de la **définition** (parfois appelée **déclaration**) de la variable.
- En Langage C, les minuscules et les majuscules ne sont pas équivalentes et constituent des caractères différents. Par ailleurs, les caractères accentués sont interdits.
- Exemples d'identificateurs (tous différents) : moy_geom, MoyArith, resultat_1, data_4octets, somme, Somme, Prix_Article, masse_proton, ...
- Ne pas donner à une variable un nom entièrement écrit en majuscule : les noms en majuscules sont réservés aux constantes.

Créer une variable

Une variable est caractérisée par :

- son **identificateur**, c'est-à-dire son nom ;
- son **type**, qui indique l'ensemble des valeurs qui peuvent être attribuées à la variable et les opérations possibles sur la variable ;
- sa **valeur initiale**, qui peut éventuellement être indéfinie ;
- sa **classe d'allocation mémoire**, qui ne sera abordée qu'au chapitre correspondant. Sauf indication contraire, les variables seront toutes locales (règle de style). Une variable locale peut être automatique (cas par défaut) ou statique (mot-clé static).

Les différents types d'une variable

En Langage C existent trois familles de types de base :

- les types **entiers** (bâtis autour du mot-clé **int**) permettent de représenter les nombres entiers ;
- les types **réels** (mot-clé **float** ou **double**) permettent de représenter les nombres réels (parfois appelés "flottants" en informatique) ;
- le type "**octet**" (mot-clé **char**) permet de représenter les variables occupant un seul octet, entre particulier les caractères ; il s'agit en réalité d'un type entier.

Les variables de type entier

Définition	Valeurs possibles	Place occupée
<i>int</i>	dépend du logiciel et du processeur. Très utilisé par les informaticiens « purs », il est déconseillé en informatique industrielle.	2 ou 4 octets
<i>short int</i> ou <i>short</i>	-32768 à +32767	2 octets
<i>unsigned short int</i> ou <i>unsigned short</i>	0 à 65535	2 octets
<i>long int</i> ou <i>long</i>	-2147483648 à 2147483647	4 octets
<i>unsigned long int</i> ou <i>unsigned long</i>	0 à 4294967295	4 octets
<i>char</i> Très utile en II ¹ : <i>unsigned char</i>	0 à 255 ou -128 à +127 (selon le compilateur)	1 octet

Les définitions de type entier (les types en gris sont les plus utilisés)

Les variables de type réel

Définition	Gamme de valeurs	Précision	Place occupée	Exemples de constantes	commentaires
double	$+2,23.10^{-308}$ à $+1,79.10^{308}$ et 0 $-2,23.10^{-308}$ à $-1,79.10^{308}$	10^{-15}	8 octets	-1.6e-19 3.14159	à choisir sur un PC
float (obsolète sur PC)	$+1,21.10^{-38}$ à $+3,4.10^{38}$ et 0 $-1,21.10^{-38}$ à $-3,4.10^{38}$	10^{-6}	4 octets	-1.6e-19f 3.14159f	à éviter sur un PC sauf si la place est comptée. Certains μC le tolèrent (mais coûteux !).

Les définitions de réels (le type en gris est le plus utilisé)

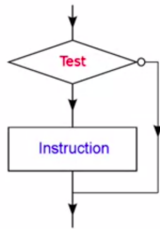
Assignations

- `int x` définit une **variable**
(ici, un nombre, on parlera plus tard des types)
- L'assignation :
 $x = 12;$
 $x = y + 3;$
 $x = x - 2;$
- Les opérations arithmétiques : $+$ $-$ $/$ $*$ $\%$

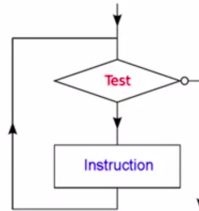
printf et scanf

format	conversion en	écriture
%d	int	décimale signée
%ld	long int	décimale signée
%u	unsigned int	décimale non signée
%lu	unsigned long int	décimale non signée
%o	unsigned int	octale non signée
%lo	unsigned long int	octale non signée
%x	unsigned int	hexadécimale non signée
%lx	unsigned long int	hexadécimale non signée
%f	double	décimale virgule fixe
%lf	long double	décimale virgule fixe
%e	double	décimale notation exponentielle
%le	long double	décimale notation exponentielle
%g	double	décimale, représentation la plus courte parmi %f et %e
%lg	long double	décimale, représentation la plus courte parmi %lf et %le
%c	unsigned char	caractère
%s	char*	chaîne de caractères

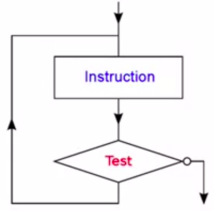
Structures de contrôle de base



Test: if

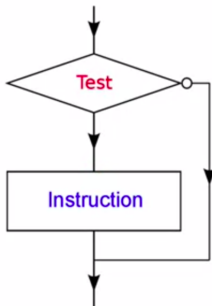


Boucles: while



do...while

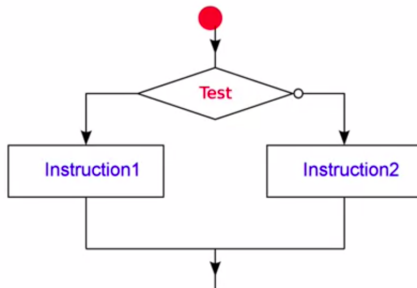
Le test de base



```
if (Condition)
{
    Instruction;
}
```

```
if (Condition){
    Instruction;
}
```


Le test complet



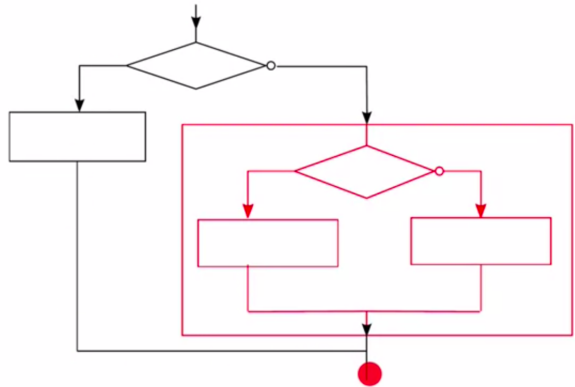
```
if (Condition)
{
    Instruction1;
}
else
{
    Instruction2;
}
```

Le test complet

```

if (x == 0) {
    // ...
}
else if (x == 1) {
    // ...
} else {
    ///...
}

```



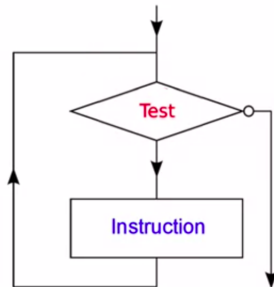
La boucle for

```
for (i=0 ; i<10 ; i++) {  
    // instructions  
}
```

Trois paramètres :

- **l'initialisation** : une instruction qui s'exécute une fois au début de la boucle
- **la condition** : la condition pour que la boucle continue
- **l'incrémement** : une instruction qui s'exécute à chaque itération

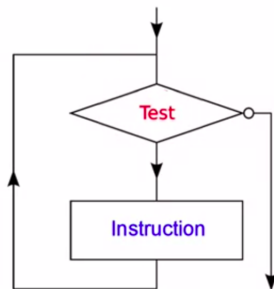
while : la boucle avec le test initial



```
while (Condition)
{
    Instruction;
}
```

```
while (Condition){
    Instruction;
}
```

while : la boucle avec le test initial

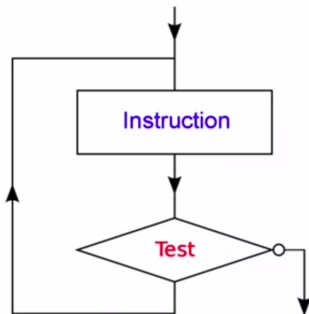


```
while (Condition)
{
    Instruction;
}
```

```
while (Condition){
    Instruction;
}
```

La structure **while** est une boucle. Elle peut durer indéfiniment !
L'instruction peut s'exécuter **0, 1 ou plusieurs** fois.

while : la boucle avec le test final

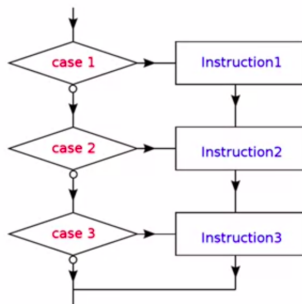


```
do  
{  
    Instruction;  
} while (Condition);
```

```
do {  
    Instruction;  
} while (Condition);
```

La structure **do...while** est une boucle. Elle peut durer indéfiniment !
L'instruction peut s'exécuter **1 ou plusieurs** fois.

switch...case : un branchement conditionnel



```
switch (expression) {  
    case (1) : Instruction1;  
    case (2) : Instruction2;  
    case (3) : Instruction3;  
}
```

La structure **switch..case.** permet d'effectuer des tests et branchements selon une liste de valeur d'une variable.

Instructions d'échappement : "continue"

Les instructions d'échappement permettent de rompre le déroulement séquentiel d'une suite d'instructions. L'instruction continue, permet de passer prématurément au tour de boucle suivant.

- Exemple :

```
for ( int i=0 ; i<3 ; i++)
{
    printf(" Debut de la boucle %d\n", i);
    if( i < 2 ) continue ;
    printf("J'utilise l'instruction continue\n");
}
```

Résultat :

Debut de la boucle 0

Debut de la boucle 1

Debut de la boucle 2

J'utilise l'instruction continue

Instructions d'échappement : "break"

L'instruction `break`; permet de quitter la boucle ou l'aiguillage le plus proche

- Exemple :

```
for ( int i=0 ; i < 3 ; i++)  
{  
    printf("Debut de la boucle %d\n", i);  
    if( i == 2){  
        printf("Je sors de la boucle\n");  
        break ;  
    }  
}
```

Résultat :

```
Je suis dans la boucle 0  
Je suis dans la boucle 1  
Je suis dans la boucle 2  
Je sors de la boucle
```

Instructions d'échappement : "return"

`return` [expression]

Cette instruction permet de sortir de la fonction qui la contient :

- si elle est suivie d'une expression, la valeur de celle-ci est transmise à la fonction appelante après avoir été convertie, si nécessaire et si possible, dans le type de celui de la fonction ;
- sinon la valeur retournée est indéterminée.
- Exemple :

```
int plus_grand(){  
    if(x > y) return x;  
    else return y;  
}
```

Instructions d'échappement : "exit"

Un programme peut être interrompu au moyen de la fonction exit.

- Exemple :

```
for( int i = 0 ; i < 10 ; i++ )  
{  
    printf(" Debut_tour_%d\n", i);  
    if( i == 2 )  
        exit (1);  
}
```

Résultat :

Debut tour 0

Debut tour 1

Debut tour 2

Exercice

Exercice 1 :

Calculer parmi les entiers de 1 à 100 :

- la somme des entiers pairs,
- la somme des carrés des entiers impairs,
- la somme des cubes de ces entiers.

Exercices

Exercice -corrigé:

```
#include "stdio.h"

int main()
{
    int pairs, carres_impairs, cubes;

    pairs = carres_impairs = cubes = 0;
    // Boucle de calcul.
    for( int i=1; i<=100; i++ )
    {
        cubes += i*i*i;
        // "i" est-il pair ou impair?
        i%2 ? carres_impairs += i*i : (pairs += i);
    }
    // Impression des resultats.
    printf("%d\n", pairs);
    printf("%d\n", carres_impairs);
    printf("%d\n", cubes);

    return 0;
}
```