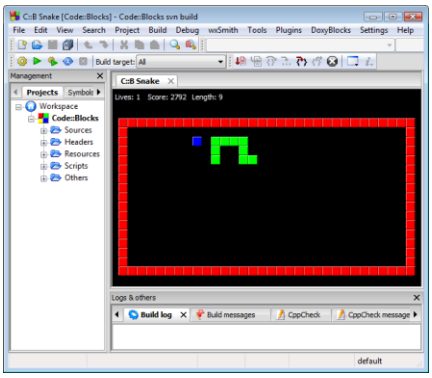
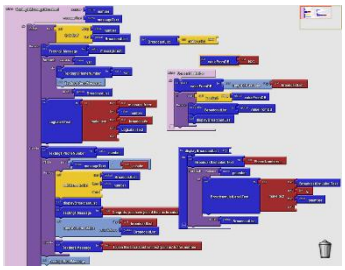


Les cahiers d'Exercices en Programmation algorythmique :



Le langage C à l'aide de CODE::BLOCK

Vous allez apprendre et entraîner ce que vous avez acquis à l'aide :

- De nombreux exercices à réaliser par vous-même
- Des corrigés et des explications Pas à Pas utiles dans votre apprentissage.

Professeur : Bérahino Justine, Kimps Bart

Professeur : Kimps Bart

AVANT-PROPOS

Ce livre est un cahier d'exercices : il vous propose des énoncés d'exercices et leurs corrigés. Vous allez apprendre le logiciel en vous entrainant à travers des exercices regroupés par thème.

Chaque énoncé vous présente l'exercice à réaliser. Vous trouverez le corrigé de chaque exercice dans ce cahier. Certaines explications peuvent-être présentes et expliquées Pas à Pas.

METHODOLOGIE

Lors de la réalisation des exercices, vous pourrez remédier à certain problème à l'aide des corrections.

Des légendes ou recommandations peuvent être présentes dans certains exercices. Celles-ci vous aideront dans vos recherches. Elles ne doivent pas être reproduites dans votre travail.

Chaque point de matière acquis dans un exercice peut être utilisé dans des exercices suivants sans explication.

Table des matières

I. Historique du C

Le langage C a connu une croissance en popularité énorme ces dernières années. On trouve ses sources en 1972, dans les laboratoires Bell, afin de développer une version portable du système d'exploitation Unix. C'est un langage de programmation structuré, mais très "près" de la machine. Publication en 1978 de "The C programming language" par Kernighan et Ritchie : définition classique du C. Le développement de compilateurs C par d'autres maisons ont rendu nécessaire la définition d'un standard précis : le standard ANSI-C.

Quelques dates :

1983 : Développement par AT&T du C++

1988: Seconde édition du livre "The C programming language".

1990: Standard ANSI-C++

II. Le succès du C

Il est dû aux faites que :

- C'est un langage universel : C n'est pas orienté vers un domaine d'applications spécifique (au contraire du FORTRAN : applications scientifiques, COBOL : applications commerciales).
- C'est un langage compact : C est basé sur un noyau de fonctions et d'opérateurs limités, permettant la formulation d'expressions simples et efficaces.
- Il est près de la machine : comme il a été développé initialement pour programmer le système UNIX, il offre des opérateurs très proches de ceux du langage machine et des fonctions qui permettent un accès simple et direct aux fonctions internes de l'ordinateur (par exemple la mémoire).
- Il est rapide puisqu'il est près de la machine.
- Il est portable : en respectant le standard ANSI-C il est possible d'utiliser le même programme sur tout autre système d'exploitation

en possession d'un compilateur C. C est devenu aujourd'hui le langage de programmation des micro-ordinateurs.

- Il est extensible : C ne se compose pas seulement de la fonction standard, le langage est animé par des bibliothèques de fonctions privées ou livrées par de nombreuses maisons de développement.

III. Inconvénients

- La possibilité d'expressions compactes entraîne le risque de se retrouver avec des programmes incompréhensibles (pour les autres, mais aussi pour nous- même), d'où la nécessité d'inclure des commentaires dans les programmes.
- C est un langage proche de la machine, il est donc dangereux. Bien qu'il soit un langage de programmation structuré, il ne nous oblige pas à adopter un style de programmation (comme, par exemple le PASCAL). Le programmateur a donc beaucoup de libertés, mais aussi des responsabilités: il doit veiller à adopter un style de programmation propre, solide et compréhensible.

IV. Les langages de programmation

Votre ordinateur est une machine étonnante et complexe. À la base, il ne comprend qu'un langage très simple constitué de 0 et de 1.

Ainsi, un message tel que celui-ci : 10100100101000110101010010101110101000110100 10 peut signifier quelque chose comme « Affiche une fenêtre à l'écran ».

S'il fallait écrire dans ce langage (qu'on appelle langage binaire), il ne faudrait pas des années pour concevoir un jeu comme Starcraft II mais plutôt des millénaires. Pour se simplifier la vie, les informaticiens ont créé des langages intermédiaires, plus simples que le binaire. Il existe aujourd'hui des centaines de langages de programmation.

Tous les langages de programmation ont le même but : vous permettre de parler à

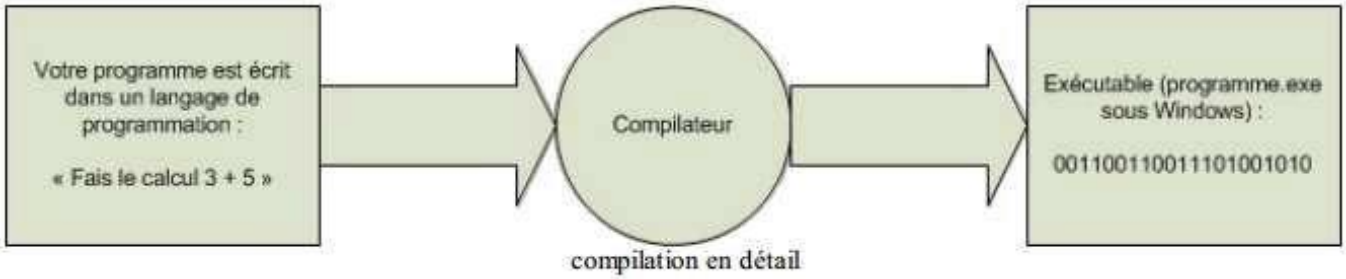
l'ordinateur plus simplement qu'en binaire.

Voici comment cela fonctionne :

- Vous écrivez des instructions pour l'ordinateur dans un langage de programmation (par exemple le C) ;
- Les instructions sont traduites en binaire grâce à un programme de « traduction » ;
- L'ordinateur peut alors lire le binaire et faire ce que vous avez demandé ! Résumons ces étapes dans un schéma (figure suivante). La compilation Le fameux« programme de traduction » s'appelle en réalité le compilateur. C'est un outil indispensable. Il vous permet de transformer votre code, écrit dans un langage de programmation, en un vrai programme exécutable.

Résumons ces notions par un schéma:

Le fameux « programme de traduction » s'appelle en réalité le compilateur. C'est un outil indispensable. Il vous permet de transformer votre code, écrit dans un langage de programmation, en un vrai programme exécutable. Reprenons le schéma précédent et utilisons un vrai vocabulaire d'informaticien (figure suivante).



compilation en détail

Voilà ce que je vous demande de retenir pour le moment : ce n'est pas bien compliqué

mais c'est la base à connaître absolument !

Les programmeurs (aussi appelés développeurs) connaissent en général plusieurs langages de programmation et non pas un seul. On se concentre rarement sur un seul langage de programmation. Bien entendu, il faut bien commencer par l'un d'eux. La bonne nouvelle, c'est que vous pouvez commencer par celui que vous voulez ! Les principes des langages sont souvent les mêmes, vous ne serez pas trop dépaysés d'un langage à l'autre.

V. Les outils nécessaires pour programmer

Un compilateur qui permet de traduire votre langage C en langage binaire ! Comme vous savez il existe plusieurs compilateurs.

Voici le minimum demandé par un programmeur :

- un éditeur de texte pour écrire le code source du programme. En théorie un logiciel comme le Bloc-notes sous Windows, ou « vi » sous Linux fait l'affaire. L'idéal, c'est d'avoir un éditeur de texte intelligent qui colore tout seul le code, ce qui vous permet de vous y repérer bien plus facilement ;
- un compilateur pour transformer (« compiler ») votre source en binaire ;
- un débogueur pour vous aider à traquer les erreurs dans votre programme. On n'a malheureusement pas encore inventé le « correcteur » qui corrigerait tout seul nos erreurs. Ceci dit, quand on sait bien se servir du débogueur, on peut facilement retrouver ses erreurs !

Il existe plusieurs environnements de développement. Au début, vous aurez peut- être un peu de mal à choisir celui qui vous plaît. Une chose est sûre en tout cas : vous pouvez réaliser n'importe quel type de programme, quel que soit l'IDE que vous choisissez.

VI. Choisir un IDE

Il m'a semblé intéressant de vous montrer quelques IDE parmi les plus connus. Tous sont disponibles gratuitement.

Fonctionne sous Windows, Mac et Linux.

Un des IDE que les programmeurs préfèrent s'appelle "Code::Blocks". Il est gratuit et fonctionne sur la plupart des systèmes d'exploitation. Nous allons utiliser celui-ci pour la suite du cours.

Fonctionne sous Windows uniquement.

Le plus célèbre IDE sous Windows, c'est celui de Microsoft : "Visual C++". Il existe à la base en version payante (chère !), mais il existe heureusement une version gratuite intitulée Visual C++ Express qui est vraiment très bien (il y a peu de différences avec la version payante). Il est très complet et possède un puissant module de correction des erreurs (débogage).

Fonctionne sous Mac OS X uniquement.

Sur Mac OS X, vous pouvez utiliser Xcode, généralement fourni sur le CD d'installation de Mac OS X. C'est un IDE très apprécié par tous ceux qui font de la programmation sur Mac.

VII. Télécharger Code::Blocks

Rendez-vous sur la page de téléchargement "Code::Blocks".

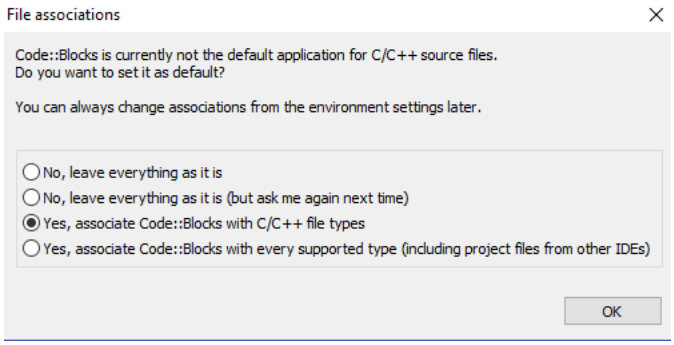
Voici un lien de téléchargement de la version Windows avec compilateur :
https://sourceforge.net/projects/codeblocks/files/Binaries/10.05/Windows/

Voici le nom du fichier SETUP : codeblocks-10.05mingw-setup.exe

De nouvelles versions existent mais le cours a été créé à partir de celle située ci-dessus.

Fermer certaines fenêtres lors du premier lancement de votre programme : Did you Know et Scripting Console.

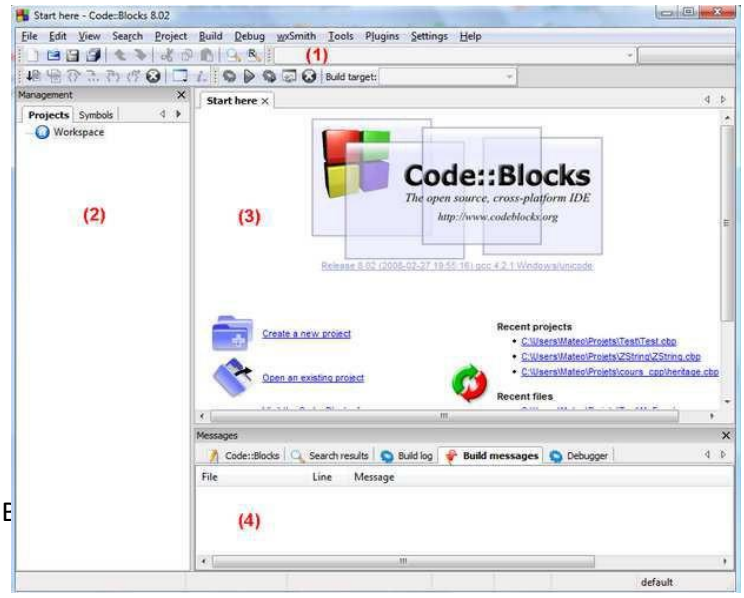
Ne pas oublier d'associer Code:Blocks au fichiers C/C++



On distingue 4 grandes sections dans la fenêtre, numérotées sur l'image :

1. La barre d'outils : elle comprend de nombreux boutons, mais seuls quelques- uns nous seront régulièrement utiles. J'y reviendrai plus loin ;

2. la liste des fichiers du projet : c'est à gauche que s'affiche la liste de tous les fichiers source de votre programme. Notez que sur cette capture d'écran que aucun projet n'a été créé, on ne voit donc pas encore de fichiers à l'intérieur de la liste. Vous verrez cette section se remplir dans cinq minutes



en lisant la suite du cours ;

3. la zone principale : c'est là que vous pourrez écrire votre code en langage C;

4. la zone de notification : aussi appelée la « zone de la mort », c'est ici que vous verrez les erreurs de compilation s'afficher si votre code comporte des erreurs. Cela arrive très régulièrement !

Intéressons-nous maintenant à une section particulière de la barre d'outils (fig. suivante). Vous trouverez les boutons suivants (dans l'ordre) :

"***Compiler***", "***Exécuter***", "***Compiler & Exécuter***" et "***Tout recompiler***".

Retenez-les, nous les utiliserons régulièrement.

Compiler : tous les fichiers source de votre projet sont envoyés au compilateur qui va se charger de créer un exécutable. S'il y a des erreurs - ce qui a de fortes chances d'arriver tôt ou tard ! -, l'exécutable ne sera pas créé et on vous indiquera les erreurs en bas de Code::Blocks ;

Exécuter : cette icône lance juste le dernier exécutable que vous avez compilé. Cela vous permettra donc de tester votre programme et de voir ainsi ce qu'il donne. Dans l'ordre, si vous avez bien suivi, on doit d'abord compiler, puis exécuter pour tester ce que ça donne. On peut aussi utiliser le troisième bouton…

Compiler & exécuter (Touche F9) : pas besoin d'être un génie pour comprendre que c'est la combinaison des deux boutons précédents. C'est d'ailleurs ce bouton que vous utiliserez le plus souvent. Notez que s'il y a des erreurs pendant la compilation (pendant la génération de l'exécutable), le programme ne sera pas exécuté. À la place, vous aurez droit à une belle liste d'erreurs à corriger !

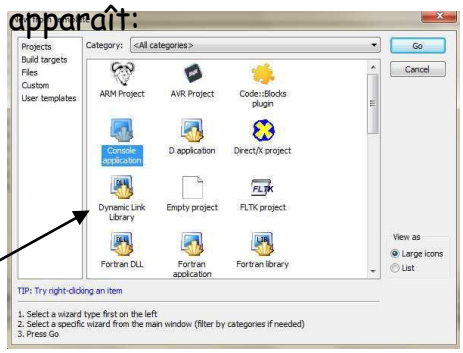
Tout recompiler : quand vous faites compiler, Code::Blocks ne recompile en fait que les fichiers que vous avez modifiés et non les autres. Parfois - je dis bien parfois - vous aurez besoin de demander à Code::Blocks de vous recompiler tous les fichiers. On verra plus tard quand on a besoin de ce bouton, et vous verrez plus en détails le fonctionnement

de la compilation dans un chapitre futur. Pour l'instant, on se contente de savoir le minimum nécessaire pour ne pas tout mélanger. Ce bouton ne nous sera donc pas utile de suite.

VIII. Création d'un nouveau projet

Pour créer un nouveau projet, c'est très simple :

Cliquez sur le menu [File/New/Project]. Une fenêtre apparaît:



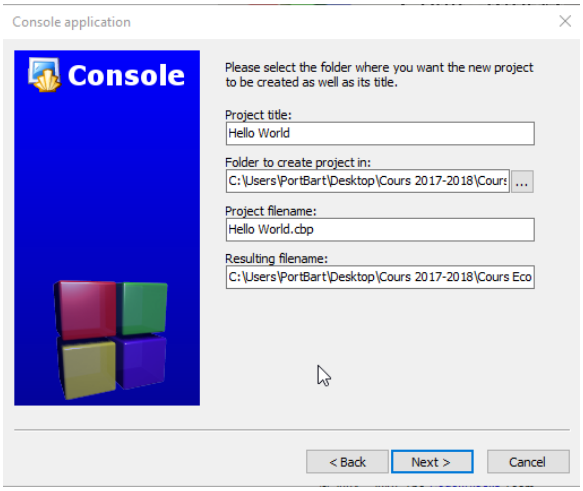
Choisissez Console application et cliquez sur le bouton GO. La fenêtre suivante apparaît:

Cliquez sur le bouton Next. La fenêtre suivante apparaît. Choisissez C et cliquez sur le bouton Next.

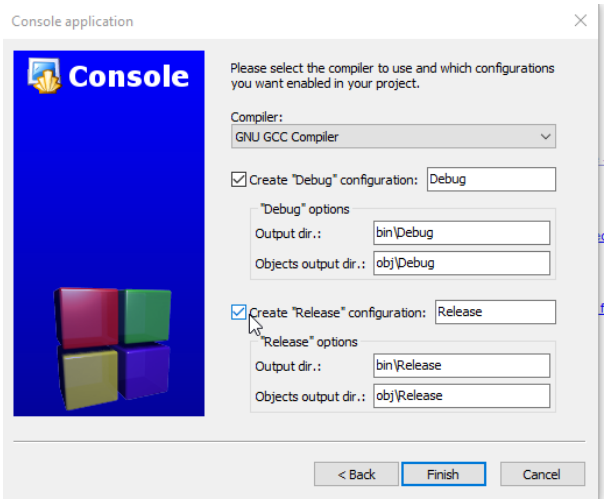
Donnez un nom au projet, sélectionner le dossier dans lequel vous voulez l'enregistrer. J'ai créé dans mon cas un dossier Exercices dans le dossier Cours Code::Block.

À vous de gérer votre espace.

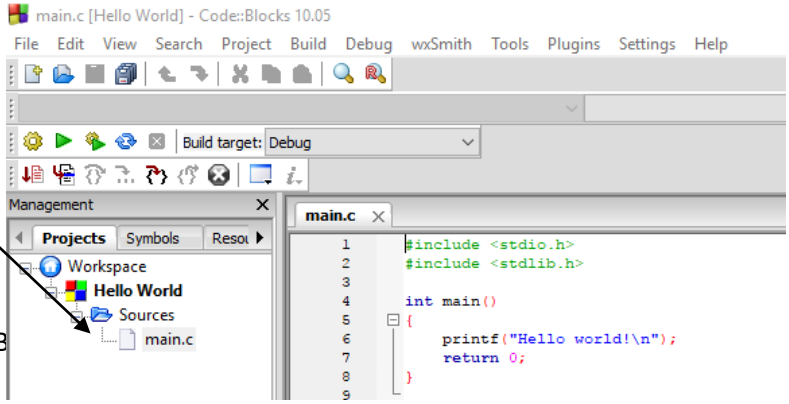
Cliquez sur le bouton Next.



Rien n'est à modifier ici, vous pouvez donc cliquer sur Finish.



Voici votre premier projet :



Pour n'importe quel programme, il faudra taper un minimum de code. Ce code ne fera rien de particulier mais il est indispensable. C'est ce « code minimum » que nous allons découvrir maintenant. Il devrait servir de base pour la plupart de vos programmes en langage C.

Code::Blocks a donc généré le minimum de code en langage C dont on a besoin.

Le voici :

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

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

```

Remarque : 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).

Notez également que la ligne : int main() peut s'écrire int main(int argc, char *argv[]).

Les deux écritures sont possibles, mais la seconde (la compliquée) est la plus courante. J'aurai donc tendance à utiliser plutôt cette dernière dans les prochains chapitres. En ce qui nous concerne, que l'on utilise l'une ou l'autre des écritures, ça ne changera rien pour nous. Inutile donc de s'y attarder, surtout que nous n'avons pas encore le niveau pour analyser ce que ça signifie.

Si la console ne s'affiche pas, vous pouvez ajouter cette ligne avant la commande return.

Getchar();

Si vous avez un souci de compilation : Cliquer dans le menu sur Settings – Compiler and debugger – Toolchain executable – Cliquer sur Auto-detect. Il configure l'endroit où il trouve le dossier de compilation.

IX. Analyse de notre premier projet (le code)

Commençons par les deux premières lignes qui se ressemblent beaucoup :

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

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

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. Il y a deux lignes, donc deux fichiers inclus. Ces fichiers s'appellent stdio.h et stdlib.h.

Ces fichiers existent déjà, des fichiers source tout prêts. On les appelle des bibliothèques (certains parlent de librairies mais c'est un anglicisme). En gros, ces fichiers contiennent du code tout prêt qui permet d'afficher du texte à l'écran. 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. Bref, en résumé les deux premières lignes incluent les bibliothèques qui vont nous permettre (entre autres) d'afficher du texte à l'écran assez « facilement ».

Et la suite

```
4   int main()
5   {
6       printf("Hello world!\n");
7       return 0;
8   }
9
```

Ce que vous voyez là, 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, notre programme ne contient donc qu'une seule fonction.

Une fonction permet grosso modo de rassembler plusieurs commandes à l'ordinateur. Regroupées dans une fonction, les commandes permettent de faire quelque chose de précis. Par exemple, on peut créer une fonction "ouvrir_fichier" qui contiendra une suite d'instructions pour l'ordinateur lui expliquant comment ouvrir un fichier. L'avantage, c'est qu'une fois la fonction écrite, vous n'aurez plus qu'à dire "ouvrir_fichier", et votre ordinateur saura comment faire sans que vous ayez à tout répéter ! Notre 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. Si vous avez bien suivi, notre fonction main contient deux lignes :

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

Ces lignes à l'intérieur d'une fonction ont un nom. 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

Remarque : Toute instruction se termine obligatoirement par un point-virgule « ; ». C'est d'ailleurs comme ça qu'on reconnaît ce qui est une instruction et ce qui n'en est pas une. Si vous oubliez de mettre un point-virgule à la fin d'une instruction, votre programme ne

compilera pas !

La première ligne : printf("Hello world!\n"); demande à afficher le message « Hello world! » à l'écran. Quand votre programme arrivera à cette ligne, il va donc afficher un message à l'écran, puis passer à l'instruction suivante.

Passons à l'instruction suivante justement :

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

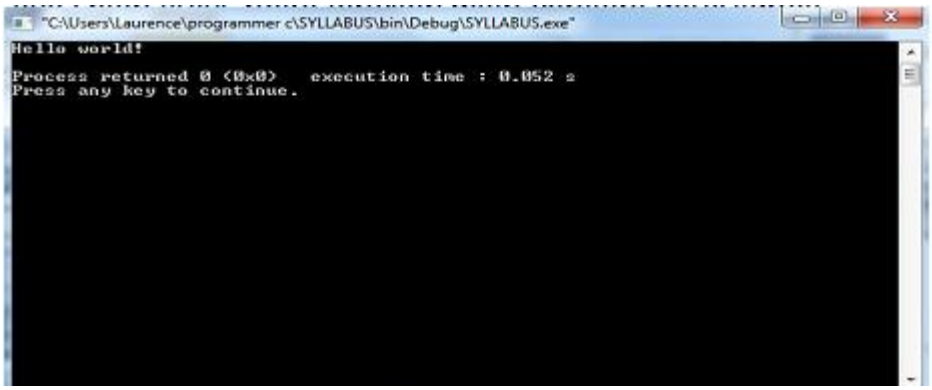
Eh bien ça, en gros, ça veut dire que c'est fini. Cette ligne indique qu'on arrive à la fin de notre fonction main et demande de renvoyer la valeur 0. Pourquoi le programme renvoi 0 car chaque programme une fois terminé renvoie une valeur, par exemple pour dire que tout s'est bien passé. En pratique, 0 signifie "tout s'est bien déroulé" 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.

Voici un schéma qui reprend l'ensemble :

X. Testons notre premier programme

Cliquer sur le bouton 

Attendre car le programme est pour la première fois compilé et voici l'affichage du programme :



Le programme affiche « Hello world! » (sur la première ligne).

Les lignes en dessous ont été générées par Code::Blocks et indiquent que le programme s'est bien exécuté et combien de temps s'est écoulé depuis le lancement.

On vous invite à appuyer sur n'importe quelle touche du clavier pour fermer la fenêtre.

XI. Modifier le message à afficher à l'écran

À partir de maintenant, on va modifier nous-mêmes le code de ce programme minimal. Afficher le message « **Bonjour** » à l'écran. Comme tout à l'heure, une console doit s'ouvrir. Le message « **Bonjour** » doit s'afficher dans la console.

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

Ce sera en fait assez simple. Si vous partez du code qui a été donné plus haut, il vous suffit simplement de remplacer « **Hello world!** » par « **Bonjour** » dans la ligne qui fait

appel à **printf**. Comme je vous le disais plus tôt, **printf** est une instruction. Elle commande à l'ordinateur : « **Affiche-moi ce message à l'écran** ».

Modifier donc votre premier programme : le message à afficher.

Voici le code source que vous devez voir à l'écran :

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

int main()
{
    printf("Bonjour");

    return 0;
}
```

On a donc 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.

XII. Les caractères spéciaux

Les caractères spéciaux sont des lettres spéciales qui permettent d'indiquer qu'on veut aller à la ligne, faire une tabulation, etc. Ils sont faciles à reconnaître : 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. Voici deux caractères spéciaux courants que vous aurez probablement besoin d'utiliser, ainsi que leur signification :

\n : retour à la ligne (= « Entrée ») ;

\t : tabulation

Dans notre cas, pour faire une entrée, il suffit de taper \n pour créer un retour à la

ligne. Si je veux donc faire un retour à la ligne juste après le mot « Bonjour », je devrais taper :

`Printf ("Bonjour\n");`

Tout ce que vous écrirez à la suite du `\n` sera placé sur la deuxième ligne.

Entraînez-vous en écrivant :

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

Voici le résultat :

Start here | main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      printf("Bonjour\nAu Revoir\n");
7      return 0;
8  }
```

XIII. Les commentaires

Cela signifie 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 par-ci par-là. Cela permet de vous retrouver au milieu d'un de vos codes source plus tard. 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. Enfin, ça va me permettre à moi d'ajouter des annotations dans les codes source de ce cours. Et de mieux vous expliquer à quoi peut servir telle ou telle ligne de code. Il y a plusieurs manières d'insérer un commentaire. Tout dépend de la longueur du commentaire que vous voulez écrire.

Si votre commentaire est court et qu'il tient sur une seule ligne, il ne fait que quelques mots. Dans ce cas, vous devez taper un double slash (//) suivi de votre commentaire.

Exemple en modifiant votre code :

Start here | main.c

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

int main()
{
    printf("Bonjour\nAu Revoir\n");// Ceci est un commentaire
    return 0;
}
```

Si votre commentaire est long et que vous avez besoin d'écrire plusieurs phrases qui tiennent sur plusieurs lignes. Dans ce cas, vous devez 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 (*/). V

Exemple en modifiant votre code :

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

int main()
{
    printf("Bonjour\nAu Revoir\n");// Ceci est un commentaire
    return 0; /* Ceci est
        un commentaire
        sur plusieurs lignes */

}
```

XIV. Les variables et les opérations

Les variables permettent à l'ordinateur de retenir des nombres, les mettre en mémoire. Nous allons donc apprendre à stocker ces nombres dans la mémoire.

Expliquons les différents types de mémoire :

- 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 RAM ;
- le disque dur : que vous connaissez sûrement, c'est là qu'on enregistre les fichiers.

J'ai classé les mémoires du plus rapide à la plus lente. Mais la mémoire la plus rapide est la plus petite. En fait, en programmation, on va surtout travailler avec la mémoire vive. On verra aussi que l'on travaillera avec le disque dur, pour lire et créer des fichiers. Quant à la mémoire cache et aux registres, on n'y touchera pas.

Il faut ajouter une dernière chose très importante : 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 !

La RAM ne peut stocker que des nombres. Mais comment fait-on alors pour retenir des mots ? En fait, même les lettres ne sont que des nombres pour l'ordinateur ! Une phrase est une simple succession de nombres. Il existe un tableau réalisant la correspondance entre les nombres et les lettres. C'est un tableau informant par exemple : le nombre 67 correspond à la lettre Y.

Revenons à notre schéma. Les choses sont en fait très simples : si l'ordinateur veut retenir le nombre 5, 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 !

Voilà en gros comment ça fonctionne.

Définir une variable

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 (ouf !) : 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. Voilà déjà un souci de moins.

Donner un nom à ses variables

En langage C, chaque variable doit donc avoir un nom mais il faut respecter des quelques règles.

- 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 » _ (qui ressemble à un trait de soulignement). C'est le seul caractère différent des lettres et chiffres autorisé ;
- Vous n'avez pas le droit d'utiliser des accents (é à ê etc.) ;
- Enfin, et c'est très important à savoir, le langage C fait la différence entre les majuscules et les minuscules. Pour votre culture, sachez qu'on dit que c'est un langage qui « respecte la casse ». Donc, du coup, les variables largeur,

LARGEUR ou encore LArgEuR sont trois variables différentes en langage C, même si pour nous ça a l'air de signifier la même chose !

Les types de variables

Comme il a été dit, un ordinateur ne sait traiter que des nombres.

Lorsque vous créez une variable, vous allez donc devoir « INDIQUER SON TYPE »

Voici les principaux types 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	-1×10^{37}	1×10^{37}
double	-1×10^{37}	1×10^{37}

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…

Remarque : Attention avec les nombres décimaux! Votre 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 e 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

En résumé, on fera surtout la distinction entre nombres entiers et flottants :

- pour un nombre entier, on utilisera le plus souvent int;
- pour un nombre flottant, on utilisera généralement double.

Déclarer une variable

Une déclaration de variable, c'est très simple, il suffit dans l'ordre de :

- 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.

Exemple (à lire) : je veux créer ma variable nommée plusgrand de type int,
je tape la ligne suivante :

```
6       int plusgrand;
```

Quelques autres exemples :

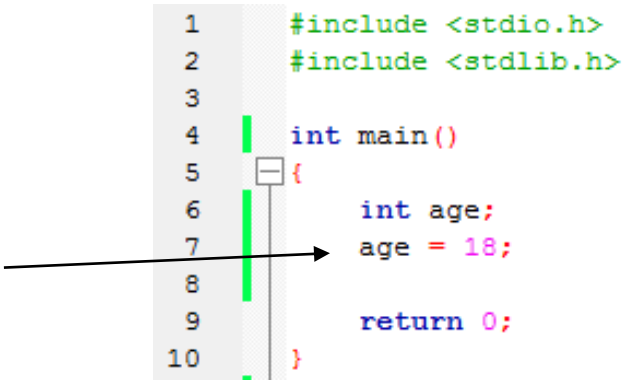
```
6       int plusgrand;
7       double totalargent;
8       unsigned int nombredelivres;
```

Remarque :
Une variable est toujours déclarée en début de programme.

Si vous avez plusieurs variables, vous pouvez les créer sur une même ligne, comme :
int age, plusgrand, pluspetit ;

Affecter une valeur à une variable

Exemple :



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

int main()
{
    int age;
    age = 18;

    return 0;
}
```

Là rien ne s'affiche à l'écran, tout se passe dans la mémoire.

Si on n'affecte pas un nombre à une variable, la variable n'a pas de valeur

XV. Les constantes

Bien souvent, nous avons besoin d'utiliser une variable dont on voudrait qu'elle garde la même valeur pendant toute la durée du programme. C'est-à-dire qu'une fois déclarée, vous voudriez que votre variable conserve sa valeur et que personne n'ait le droit de changer ce qu'elle contient.

Ces variables particulières sont appelées « constantes », justement parce que leur valeur reste constante.

Voici un exemple de déclaration de constante :

```
6       const int age = 17;
```

Comment afficher le contenu d'une variable

On utilise la commande `printf`, sauf que l'on rajoute un symbole spécial à l'endroit où on veut afficher la valeur de la variable.

Exemple :

```
7       printf("votre age %d\n", age);
```

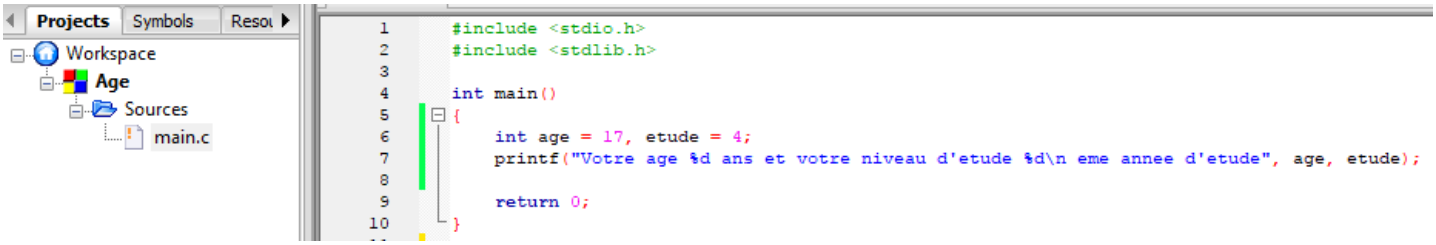
Ce « symbole spécial » est en fait un `%` suivi d'une lettre.
Cette lettre permet d'indiquer ce que l'on doit afficher.
« `d` » signifie que l'on souhaite afficher un `int`.

Voici d'autres format :

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

On peut afficher plusieurs variables dans un même `printf`.

Exemple :
Créer un nouveau projet en C appelé Age.
Effacez les lignes Hello World et écrivez les commandes ci-dessous :



Compiler et executer votre projet

Récupérer une saisie

On va demander aux utilisateurs de taper un nombre dans la console. Ce nombre, on va le récupérer et le stocker dans une variable.

Pour cela, on utilisera la fonction « scanf ».

Exemple dans un nouveau projet en C appelé Demande Age.

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

int main()
{
    int age = 0; // On initialise en premier la variable à 0*/
    printf("Quel age avez-vous ? ");
    scanf("%d", &age); //On demande d'entrer ton age avec scanf
    printf("Ah ! Vous avez donc %d ans !\n\n", age);
    return 0;
}
```

On doit mettre le %d entre guillemets "%d".

Par ailleurs, il faut mettre le symbole & devant le nom de la variable qui va recevoir la valeur.

XVI. Les calculs

Dans ce point, nous allons apprendre à réaliser des calculs qu'un ordinateur sait Faire.

Les calculs de base

Addition, Soustraction, Multiplication, Division.

Actuellement, par faciliter, vous pouvez créer des projets comme ceux créés précédemment et effacer la ligne Hello World.

Voici donc deux exemples :

Projet Addition

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

int main()
{
    int resultat = 0;
    resultat = 5 + 3;
    printf("5 + 3 = %d", resultat);
    return 0;
}
```

Projet Division

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

int main()
{
    double resultat = 0;
    resultat = 5.0 / 2.0;
    printf("5 / 2 = %f", resultat);
    return 0;
}
```

Le modulo

Le modulo est une opération mathématique qui permet d'obtenir le reste d'une division.

Le modulo se représente par le signe %. Voici quelques exemples de modulos :

- 5 % 2 = 1
- 14 % 3 =2
- 4 % 2 = 0

Le modulo 5 % 2 est le reste de la division 5 / 2, c'est-à-dire 1.

L'ordinateur calcule que 5 = 2 * 2 + 1 (c'est ce 1, le reste, que le modulo renvoie).

De même, 14 % 3, le calcul est 14 = 3 * 4 + 2 (modulo renvoie le 2).

Enfin, pour 4 % 2, la division tombe juste, il n'y a pas de reste, donc modulo renvoie 0.

Projet Modulo

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

int main()
{
    int resultat = 0;
    resultat = 14 % 3;
    printf("14 modulo 3 = %d", resultat);
    return 0;
```

N'oubliez pas que l'opération modulo est %.

Des calculs entre variables

Essayons maintenant de s'entrainer à faire des calculs entre plusieurs variables.

Exemple avec ce Projet Variable1 :

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

int main()
{
    int resultat = 0, nombre1 = 0, nombre2 = 0;
    /*On demande l'introduction du nombre 1 et 2 à l'utilisateur*/
    printf("Entrez le nombre 1 :\n");
    scanf("%d", &nombre1);
    printf("Entrez le nombre 2 :\n");
    scanf("%d", &nombre2);
    //On réalise un calcul
    resultat = nombre1 + nombre2;
    //Et on affiche l'addition à l'écran
    printf("Voici le resultat de l'addition : %d + %d = %d\n", nombre1, nombre2, resultat);

    return 0;
}
```

L'incrémentation

On peut ajouter 1 à une variable.

À l'aide de l'incrémentation la valeur de la variable augmente de 1 en 1.

Exemple : nombre = nombre + 1 ;

On peut noter également :

Nombre++ ;

La décrémentation

C'est l'inverse, on enlève 1 à une variable.

Exemple : nombre = nombre – 1 ;

Ou

Nombre-- ;

Les autres raccourcis :

nombre = nombre + 2 ;

nombre *= 2 ; (si le nombre vaut 5 au départ, il vaudra 10 après cette instruction).

```
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)
```

XVII. 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>
```

XVIII. Les conditions

Définition d'une structure conditionnelle

On appelle structure conditionnelle les instructions qui permettent de tester si une condition est vraie ou non. Ces structures conditionnelles peuvent être associées à des structures qui se répètent suivant la réalisation de la condition, on appelle ces structures des structures de boucle.

La condition if…else

Les conditions permettent de tester des variables.

Les signes indispensable

Voici la liste des opérateurs dit de comparaison, à retenir :

Symbole	Signification
==	est égal à
>	est supérieur à
<	est inférieur à
>=	est supérieur ou égal à
<=	est inférieur ou égal à
!=	est différent de

Remarque : Il y a bien deux symboles = = pour tester l'égalité.

Un if simple

L'avantage d'une condition est qu'elle permet de ne pas réaliser systématiquement toutes les instructions mais seulement dans certains cas prévus par le programmeur. Dans ce contexte nous allons commencer par étudier l'instruction If.

L'instruction de test ou instruction If (if = si en anglais) permet de ne faire exécuter certaines instructions que dans le cas où une condition est remplie.

Voici la syntaxe de If :

```
If ( < expression >) < instruction >;
```

La valeur entre parenthèse représente la condition du If. C'est d'elle que dépend ou non l'exécution de l'instruction qui suit. Si la valeur de la condition est supérieure à zéro alors l'instruction qui lui est associée sera exécutée. En revanche si la valeur est nulle (= 0) alors l'instruction ne sera pas exécutée.

Voilà un exemple pour illustrer If dans un programme. Vous pouvez créer un projet nommé `Condition1` :

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

int main()
{
    long x = 5;
    long y = 3;

    /* si  x est supérieur à y */
    if (x > y)
        printf("X est superieur a Y");

    return 0;
```

Explication : On initialise deux variables x et y. On remarque que x et bien supérieur à y. On fait ensuite appel à un test if : si la variable x et supérieur à la variable y alors affiche " X est supérieur à Y ".

Ici la variable x est supérieur à y donc le message s'affichera. Mais maintenant si vous remplacez la valeur de la variable x par 2, le message ne s'affichera plus car la condition sera fausse.

Voici un autre exemple appliquant la remarque ci-dessus. Vous pouvez créer un projet nommé `Condition2` . Les deux cas sont ici pris en compte.

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

int main()
{
    long x = 2;
    long y = 7;

    /* si  x est supérieur à y */
    if (x > y)
        printf("X est superieur a Y\n");
    if (x <y)
        printf("X est inferieur a Y\n");

    return 0;
}
```

Voici le même programme mais avec une variante dans le second comparateur.

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

int main()
{
    long x = 2;
    long y = 7;

    /* si  x est supérieur à y */
    if (x > y)
        printf("X est superieur a Y\n");
    if (!(x >y))
        printf("X est inferieur a Y\n");

    return 0;
}
```

Fichier enregistré avec le même nom.

On remarque l'opérateur de négation " ! ". On pourrait traduire cela par :
Si x n'est pas supérieur à y.
Le " ! " sert à insérer une négation dans notre condition.

Cela peut s'avérer très utile ...

Le Else pour dire « sinon »

En français, nous allons écrire ceci :

Si la variable

vaut ça, ALORS

fais ceci,

SINON fais

cela.

Premier exemple avec le code précédent. Nous allons un peu le modifier et toujours l'enregistrer avec le même nom de projet.

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

int main()
{
    long x = 2;
    long y = 7;

    /* si  x est supérieur à y */
    if (x > y)
    {
        printf("X est superieur a Y\n");
    }
     else
     {
        printf("X est inferieur a Y\n");
     }
return 0;
}
```

Exemple AgeSup1

On demande l'âge, si l'âge est supérieur ou égal à 18 alors affiche « Vous êtes majeur » sinon affiche « Tu es mineur ».

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

int main()
{
    long age = 0;

    printf("Quel est votre age ? ");
    scanf("%ld", &age);

    /*Si age = 18 ou (ou = comme opérateur ||) si age est > à 18 */
    if (age == 18 || age > 18)
        {
            printf("Vous etes majeur");
        }
    else
    {
            printf("tu es mineur");
    }
return 0;
}
```

Le Else if pour dire « sinon si »

En français, nous allons écrire ceci :

Si la variable vaut ça ALORS fais ceci,

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

Exemple AgeSup2

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

int main()
{
    long age = 0;

    printf("Quel est votre age ? ");
    scanf("%ld", &age);

    /*Si age = 18 ou (ou = comme opérateur ||) si age est > à 18 */
    if (age < 18)
        {
            printf("Vous etes mineur");
        }
    else if (age ==18)
    {
            printf("Tu es tout juste majeur");
    }
    /*si age est > 18 et (&&) age est < 100*/
    else if (age >18 && age <100)
    {
        printf("Vous etes majeur");
    }

    else
    {
        printf("Tu dois etre tres age ?");
    }
return 0;
}
```

La condition Switch

Alors, pour éviter d'avoir à faire des répétitions dans les tests, ils ont inventé une autre structure que le `if... else`. Cette structure particulière s'appelle `switch`.

Voici un Exemple nommé Switch1 :

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

int main()
{
    int age = 0;

    printf("Quel age avez-vous ? ");
    scanf("%d", &age);

    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;
    }
}
```

Vous devez utiliser une instruction `break;` obligatoirement à la fin de chaque cas. Si vous ne le faites pas, alors l'ordinateur ira lire les instructions en dessous censées être réservées aux autres cas ! L'instruction `break;` commande en fait à l'ordinateur de « sortir » des accolades.

Gérer un menu avec un switch

En console, pour faire un menu, on fait des `printf` qui affichent les différentes options possibles. Chaque option est numérotée, et l'utilisateur doit entrer le numéro du menu qui l'intéresse.

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

int main(int argc, char *argv[])
{
  int choixMenu;

  printf("=== Menu ===\n\n");
  printf("1. Royal Cheese\n");
  printf("2. Mc Deluxe\n");
  printf("3. Mc Bacon\n");
  printf("4. Big Mac\n");
  printf("\nVotre choix ? ");
  scanf("%d", &choixMenu);
  printf("\n");

  switch (choixMenu)
  {
    case 1:
       printf("Vous avez choisi le Royal Cheese. Bon choix !");
       break;

       case 2:
       printf("Vous avez choisi le Mc Deluxe. Berk, trop de sauce...");
       break;

       case 3:
       printf("Vous avez choisi le Mc Bacon. Bon, ca passe encore ca ;o)");
       break;

       case 4:
       printf("Vous avez choisi le Big Mac. Vous devez avoir tres faim !");
       break;

       default:
       printf("Vous n'avez pas rentre un nombre correct. Vous ne mangerez rien du tout !");
       break;
  }
  printf("\n\n");
  return 0;
}
```

XVIV. Les boucles

Qu'est-ce qu'une boucle ?

C'est une technique permettant de répéter les mêmes instructions plusieurs fois.

Voici un schéma :

La boucle while

C'est aussi simple que cela.whilesignifie « Tant que ». On dit donc à l'ordinateur « Tant que la condition est vraie, répète les instructions entre accolades ».

Syntaxe :

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

Voici un exemple que vous pouvez créer. Nom : Boucle1

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

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

Cette boucle while se répète donc tant que l'utilisateur n'a pas tapé 47.

Autre exemple avec un compteur. Nom : Boucle2

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

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

Comment ça marche exactement ?

1. Au départ, on a une variable compteur initialisée à 0. Elle vaut donc 0 au début du programme.
2. La boucle while ordonne la répétition TANT QUE compteur est inférieur à 10. Comme compteur vaut 0 au départ, on rentre dans la boucle.
3. On affiche la phrase « Salut les Zeros ! » via un printf.
4. On **incrémente** la valeur de la variable compteur, grâce à compteur++;.compteur valait 0, elle vaut maintenant 1.
5. On arrive à la fin de la boucle (accolade fermante) : on repart donc au début, au niveau du while. On refait le test du while: « *Est-ce que* compteur *est toujours inférieure à 10 ?* ». Ben oui, compteur vaut 1 ! Donc on recommence les instructions de la boucle.

Maintenant, il serait intéressant de voir la valeur de la variable. Exercice nommé Boucle3.

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

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

La boucle do… while

La seule chose qui change en fait par rapport à while, c'est la position de la condition. Au lieu d'être au début de la boucle, la condition est à la fin.
Exemple : Boucle4

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

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

La boucle for

Quelles différences ?

1. Vous noterez que l'on n'a pas initialisé la variable compteur à 0 dès sa déclaration (on aurait pu le faire).
2. Il y a beaucoup de choses entre les parenthèses après le for (nous allons détailler ça après).
3. Il n'y a plus decompteur++; dans la boucle.

Intéressons-nous à ce qui se trouve entre les parenthèses, car c'est là que réside tout l'intérêt de la boucle for. Il y a trois instructions condensées, chacune séparée par un point-virgule.

1. La première est **l'initialisation** : cette première instruction est utilisée pour préparer notre variable compteur. Dans notre cas, on initialise la variable à 0.
2. 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.
3. Enfin, il y a **l'incrémentation** : 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 incrémentation, mais on peut aussi faire une décrémentation (variable--) ou encore n'importe quelle autre opération (variable += 2;pour avancer de 2 en 2 par exemple).

Bref, comme vous le voyez la boucleforn'est rien d'autre qu'un condensé. Sachez vous en servir, vous en aurez besoin plus d'une fois !

Voici un exemple Boucle6.

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

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

En résumé

1. Les **boucles** sont des structures qui nous permettent de répéter une série d'instructions plusieurs fois.
2. Il existe plusieurs types de boucles :while,do… whileetfor. Certaines sont plus adaptées que d'autres selon les cas.
3. La boucle for est probablement celle qu'on utilise le plus dans la pratique. On y fait très souvent des incrémentations ou des décrémentations de variables.

XX. TP : Plus ou Moins, votre premier jeu

Le principe est le suivant.

1. L'ordinateur tire au sort un nombre entre 1 et 100.
2. Il vous demande de deviner le nombre. Vous entrez donc un nombre entre 1 et 100.
3. 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.
4. Puis l'ordinateur vous redemande le nombre.
5. … Et il vous indique si le nombre mystère est supérieur ou inférieur.
6. 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.

Pour générer un nombre aléatoire, on utilise la fonction rand(). Cette fonction génère un nombre au hasard. Mais nous, on veut que ce nombre soit compris entre 1 et 100 par exemple (si on ne connaît pas les limites, ça va devenir trop compliqué).

Mais il faut aussi initialiser le générateur de nombres aléatoires avec srand(time(NULL))

MAX et MIN sont des constantes, le premier est le nombre maximal (100) et le second le nombre minimal (1). Les constantes doivent être définies au début du programme.

Les bibliothèques à inclure

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

Bon, vous trouverez la solution à la page suivante mais avec les informations ci-dessus, vous allez normalement trouver la solution seul.

Solution :

Nom du projet TP1.

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

int main ( int argc, char** argv )
{
    int nombreMystere = 0, nombreEntre = 0;
    const int MAX = 100, MIN = 1;
    // Génération du nombre aléatoire
    srand(time(NULL));
    nombreMystere = (rand() % (MAX - MIN + 1)) + MIN;
    /* La boucle du programme. Elle se répète tant que l'utilisateur n'a pas trouvé le nombre mystère */
    do
    {
        // On demande le nombre
        printf("Quel est le nombre ? ");
        scanf("%d", &nombreEntre);
        // On compare le nombre entré avec le nombre mystère
        if (nombreMystere > nombreEntre)
            printf("C'est plus !\n\n");
        else if (nombreMystere < nombreEntre)
            printf("C'est moins !\n\n");
        else
            printf ("Bravo, vous avez trouve le nombre mystere !!!\n\n");
    }
    while (nombreEntre != nombreMystere);

    return 0;
}
```

XXI. Les fonctions

Nous allons apprendre à structurer nos programmes en petits bouts… un peu comme si on jouait aux Legos.
Tous les gros programmes en C sont en fait des assemblages de petits bouts de code, et ces petits bouts de code sont justement ce qu'on appelle… des fonctions !

Créer et appeler 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.
On dit qu'une fonction possède une entrée et une sortie. La fig. suivante représente une fonction schématiquement.

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

1. L'entrée: on fait « rentrer » des informations dans la fonction (en lui donnant des informations avec lesquelles travailler).
2. Les calculs : grâce aux informations qu'elle a reçues en entrée, la fonction travaille.
3. 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.

Créer et appeler une fonction

Exemple de fonction : ConversionEF

On commence par une qui convertit les euros en francs. Pour ceux d'entre vous qui ne connaîtraient pas ces monnaies sachez que 1 euro = 40,3 francs belge. Cette fonction prend une variable en entrée de type double et retourne une sortie de type double car on va forcément manipuler des nombres décimaux.

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

double conversion(double euros)/*Voici la fonction avec un
                                paramètre euros*/
 {
    double francs = 0;
    francs = 40.3 * euros;
    return francs;
 }

int main()
{
    printf("10 euros = %fFB\n", conversion(10));
    printf("50 euros = %fFB\n", conversion(50));
    printf("100 euros = %fFB\n", conversion(100));
    printf("200 euros = %fFB\n", conversion(200));
    return 0;
}
```

Appel de la fonction

Autre exemple : ConversionEF2

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

double conversion(double euros)/*Voici la fonction avec un
                                paramètre euros*/
 {
    double francs = 0;
    francs = 40.3 * euros;
    return francs;
 }

int main()
{
    int nombreEntre = 0, nombreFB = 0;

    printf("Entrez un nombre... "); // 3
    scanf("%d", &nombreEntre); // 4

    nombreFB = conversion(nombreEntre);

    printf("La conversion en FB de ce nombre est %d\n", nombreFB); /

    return 0;
}
```

Exercice : Réalise en programme nommé ConversionFE qui convertit à l'inverse des FB en Euros.

Voici d'autres exercices afin de vous familiarisez au langage

Exercice : Punition

C'est 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.

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

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;
}
```

Exercice : Aire d'un rectangle nommé ARectangle

L'aire d'un rectangle est facile à calculer : largeur * hauteur.
Notre fonction nommée ARectangl eva prendre deux paramètres : la largeur et la hauteur. Elle renverra l'aire.

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

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;
}
```

Exercice : Menu

Ce code est plus intéressant et concret. On crée 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.

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

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 sucres a la sauce de myrtilles enrobee de chocolat\n");
        printf("3 : Escalope de kangourou saignante 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;
}
```

En résumé

1. Les fonctions s'appellent entre elles. Ainsi, lemainpeut appeler des fonctions toutes prêtes telles queprintfouscanf, mais aussi des fonctions que nous avons créées.
2. Une fonction récupère en entrée des variables qu'on appelle **paramètres**.
3. Elle effectue certaines opérations avec ces paramètres puis retourne en général une valeur à l'aide de l'instructionreturn.

XXII. Questions Quiz

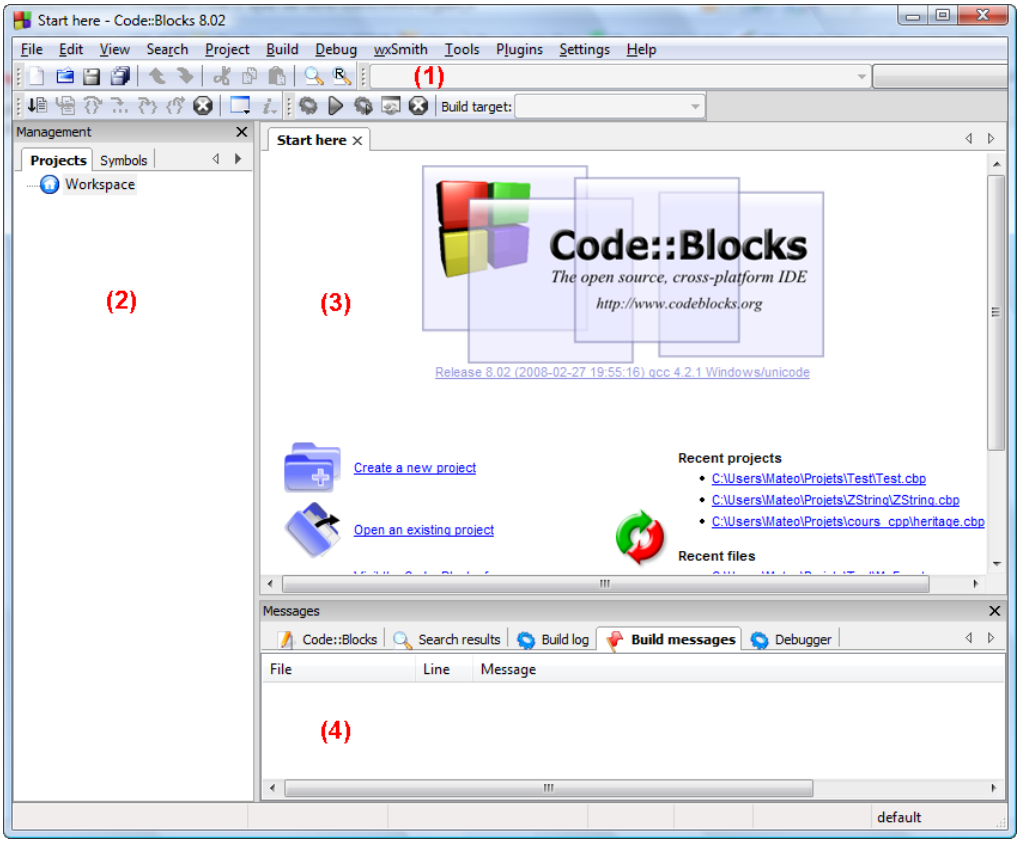
Q1. Comment s'appelle le programme chargé de traduire un code source en binaire ?

- Le diffuseur
- Le compilateur
- Le binarisateur

Q2. Lequel de ces programmes n'est *pas* un IDE ?

- Bloc-Notes
- Microsoft Visual Studio
- Code::Blocks
- Xcode

Q3. Où s'afficheront les erreurs de compilation dans Code::Blocks d'après la capture suivante ?



- Zone (1)
- Zone (2)
- Zone (3)
- Zone (4)

Q4. Que signifie le return 0 dans le code suivant ?

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

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

- Le programme a effectué 0 actions
- Le programme ne s'est pas bien exécuté
- Le programme s'est bien exécuté

Q5. Quel symbole permet d'effectuer un retour à la ligne à l'écran ?

- \t
- \n
- Il suffit d'appuyer sur la touche Entrée, voyons !

Q6. Si ma variable "compteEnBanque" est un long qui vaut 6 500 000 (soyons fous), qu'est-ce que cette ligne de code affichera à l'écran ?

```
printf("Vous avez %ld euros sur votre compte", compteEnBanque);
```

- Vous avez %ld euros sur votre compte
- Vous avez 6 500 000 euros sur votre compte
- Vous avez ld euros sur votre compte, compteEnBanque

Q7. Quelle est la seule mémoire qui n'est pas vidée lorsque l'ordinateur est éteint ?

- Registres
- Mémoire cache
- Mémoire vive
- Disque dur

Q8. Combien vaudra la variable "resultat" après cette opération ?

```
resultat = (8 / 3) - 2;
```

- -2
- 0
- 1
- 2

Q9. Quel est le problème de ce switch ?

```
switch (variable)
{
  case 5:
    printf("Salut");
  case 12:
    printf("Bonjour");
  default:
    printf("Au revoir");
}
```

- Il manque des instructions break
- Il faut mettre un point-virgule à la fin du switch
- Il faut ouvrir des accolades pour chaque "case"
- C'est "case default" et pas "default" tout court

Q10. Laquelle de ces boucles for pourrait afficher les messages suivants dans la console ?

```
Ligne n°1
Ligne n°3
Ligne n°5
Ligne n°7
```

- for (compteur = 1 ; compteur < 9 ; compteur += 2)
- for (compteur = 1 ; compteur <= 7 ; compteur++)
- for (compteur = 0 ; compteur < 9 ; compteur += 2)
- for (compteur = 1 ; compteur < 8 ; compteur++)

Q11. Combien de fois le message "Salut" sera-t-il affiché ici ?

```
int compteur = 14;

while (compteur < 15)
{
  printf("Salut\n");
}
```

- 0 fois
- 1 fois
- 14 fois
- 15 fois
- C'est une boucle infinie

Q12. Dans quel cas l'instruction return n'est pas obligatoire ?

- Quand la fonction ne prend aucun paramètre en entrée
- Quand la fonction est de type void
- Quand la fonction doit renvoyer 0

Q13. Que sont les paramètres d'une fonction ?

- Des indications sur le nom de la fonction
- Des indications sur la valeur qu'elle doit renvoyer
- Des variables qu'on lui envoie pour qu'elle puisse travailler

Q14. Quel est le problème de cette fonction qui est censée calculer le carré de la variable nombre ?
*Rappel : le carré d'un nombre, c'est nombre * nombre (le nombre multiplié par lui-même)*

```
int carre(int nombre){
   int resultat = 0;
   resultat = nombre * nombre;
}
```

- La fonction ne retourne aucune valeur
- La fonction ne marche pas car on a oublié un point-virgule quelque part
- Quel problème ? Il n'y a pas de problème !

Q15. Combien de fonctions peut comporter un programme ?

- Une seule, le *main*
- Maximum 100
- Maximum 1024
- Il n'y a pas de limite !

XXIII. Activité Partie 1 - Améliorez le jeu du Plus ou Moins

À vous de jouer ! Vous allez me montrer que vous avez compris les bases du C et que vous savez faire votre premier programme !

Vous vous rappelez du TP "Plus ou Moins" de la page 43 ? Ce jeu consiste à deviner un nombre sélectionné au hasard par l'ordinateur. Je vous avais montré comment le faire et je vous avais suggéré quelques améliorations possibles... Eh bien **c'est le moment de faire ces améliorations** !

Les améliorations à réaliser

Je vous demande en particulier de réaliser ces 3 améliorations sur le TP :

- **Faites un compteur de « coups »**. Ce compteur devra être une variable que vous incrémenterez à chaque fois que vous passez dans la boucle. Lorsque l'utilisateur a trouvé le nombre mystère, vous lui direz « Bravo, vous avez trouvé le nombre mystère en 8 coups » par exemple.
- Lorsque l'utilisateur a trouvé le nombre mystère, le programme s'arrête. Demandez si le joueur veut faire **une autre partie** !

  Il vous faudra faire une boucle qui englobera la quasi-totalité de votre programme. Cette boucle devra se répéter TANT QUE l'utilisateur n'a pas demandé à arrêter le programme. Je vous conseille de rajouter une variable booléenne du type continuerPartie initialisée à 1 au départ. Si l'utilisateur demande à arrêter le programme, vous mettrez la variable à 0 et le programme s'arrêtera.

- Créez **plusieurs niveaux de difficulté**. Au début, faites un menu qui demande le niveau de difficulté. Par exemple :

  - 1 = entre 1 et 100 ;
  - 2 = entre 1 et 1000 ;
  - 3 = entre 1 et 10000.

  Si vous faites ça, vous devrez changer votre constante MAX… Eh oui, ça ne peut plus être une constante si la valeur doit changer au cours du programme ! Renommez par exemple cette variable en nombreMaximum (vous prendrez soin

d'enlever le mot-clé const sinon ça sera toujours une constante !). La valeur de cette variable dépendra du niveau qu'on aura choisi.

Vous pouvez sans problème partir du code de base fourni en solution dans le chapitre.

Bien joué et bon courage dans la suite de tes formations.

Bibliothèque

https://openclassrooms.com/courses/apprenez-a-programmer-en-c