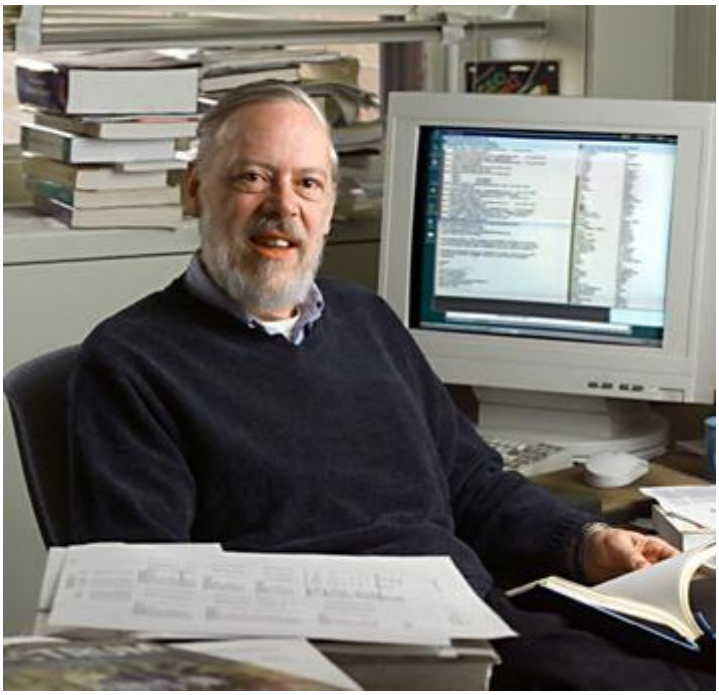
L'histoire du langage C:

Le langage C a été créé dans le courant de l'année 1972 dans les laboratoires de recherche Bell.
Essentiellement deux informaticiens l'on développé: Dennis Ritchie et Ken Thompson.
Ils se sont pour cela inspirés du langage B pour créer ce tout nouveau langage.

Ils ont aussi, par ailleurs, construit en même temps les bases d'UNIX car oui UNIX est écrit pour sa grande partie en C (avec de l'assembleur en plus).



Dennis Ritchie

Le langage C est un langage dit de "bas niveau". Cela signifie qu'il s'agit d'un

langage de programmation assez proche du langage machine (qui est le binaire).
Le langage le plus bas niveau au dessus du binaire étant l'assembleur, le langage C se trouve juste au dessus de ce dernier.

Mais rassurez-vous! Le langage C est (et heureusement) assez loin du binaire pour pouvoir y comprendre quelque chose!

Voici un exemple du premier programme que vous apprendrez à créer très bientôt:

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

int main (int argc, char** argv)
{
   printf("Hello, World!");
   return 0;
}
```

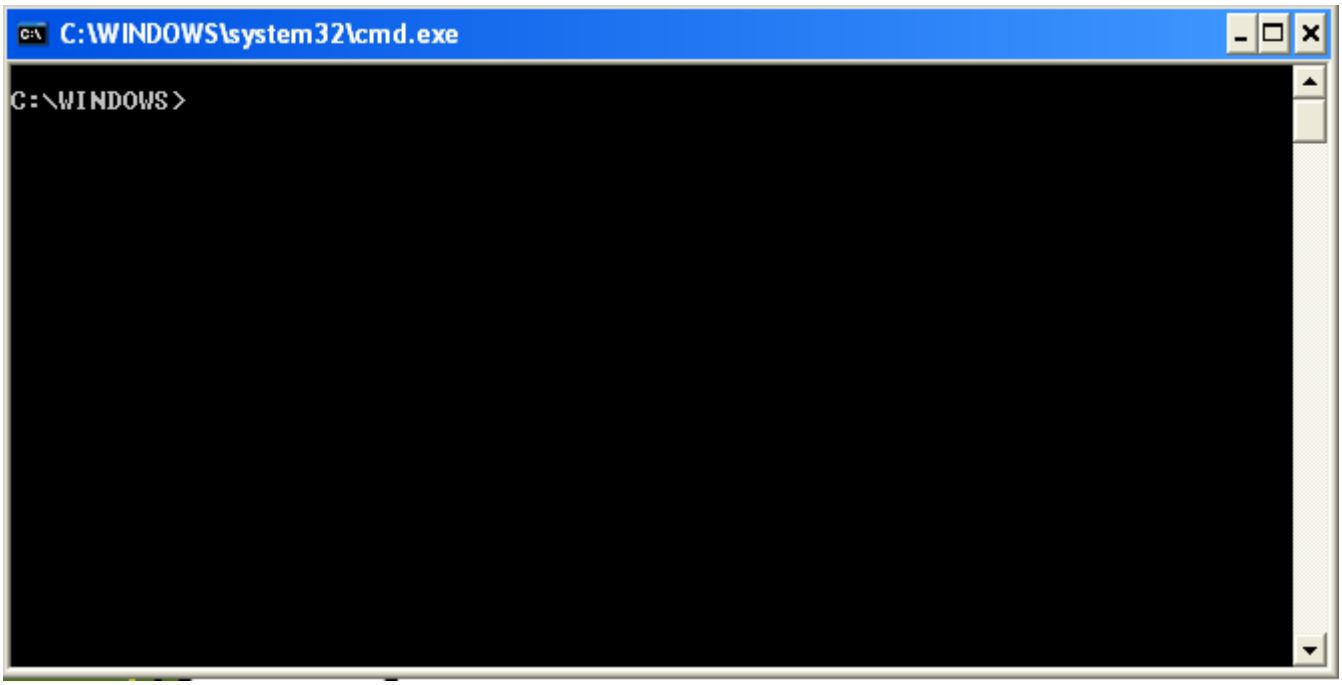
Ces lignes de code afficheront le texte Hello, World dans la console.

Mais qu'est-ce qu'une console?

La console est la fenêtre toute noire qui fait peur!
Vous pouvez la voir en cliquant sur démarrer-> exécuter -> tapez cmd et appuyez sur entrée pour windows XP.

Sinon cliquez sur démarrer -> invité de commande.

Vous devriez avoir cette fenêtre s'afficher:



Il s'agit de la console dont je viens de vous parler :)
Nos premiers programmes seront donc des programmes de ce type.
Vous allez me dire: "Oui mais moi je veux créer le nouveau Call Of Duty et ce n'est pas comme ça que je vais le réaliser!".

En effet c'est moins sympa que les programmes avec une jolie interface graphique mais je vous le répète, c'est nécessaire à l'apprentissage! Les programmes avec des couleurs et des fenêtres viendrons plus tard, une fois que vous maîtriserez les bases :)

D'ailleurs il serait bien de commencer non? Mais avant de pouvoir coder il nous manque quand même l'essentiel. Non, vous ne voyez pas? Il vous faut un IDE!

Un quoi?

Un IDE ☺ OU EDI pour Environnement de Développement Intégré. Derrière ce nom se cache un outil qui va vous faciliter grandement la vie croyez moi! Il s'agit d'un logiciel avec plusieurs fonctionnalités regroupées, car avant, au temps de Dennis Ritchie par exemple, tout se faisait à la main. Il fallait coder l'application dans ce programme ci, la compiler (vocabulaire que nous verrons juste après) dans celui la, etc...

Cela était long (même si encore de nos jours certains programmeurs utilisent encore cette méthode par nostalgie ou nécéssité) et c'est pourquoi on a eu la bonne idée d'inventer les IDE où tous les outils nécessaires sont déjà inclus et rendus beucoup plus simple d'utilisation.

Il y a une multitude d'IDE pour le langage C mais nous préférerons Code::Blocks qui est un IDE pour le langage C et C++ et qui a l'avantage d'être gratuit =)

Pour l'installer, rendez-vous sur le site officiel et téléchargez la toute dernière version en faisant bien attention de prendre la version avec mingw!

exemple: codeblocks-10.05mingw-setup.exe

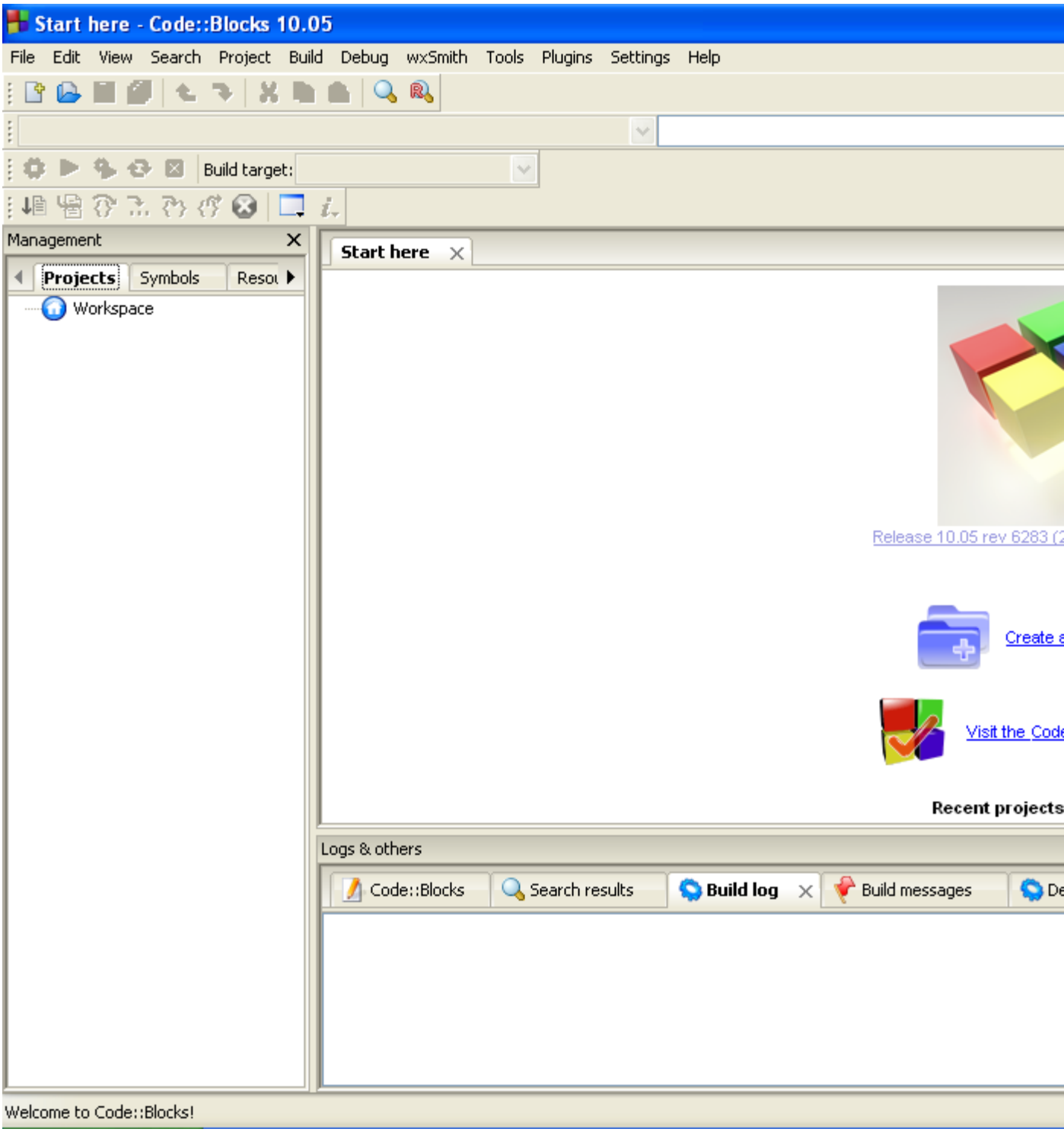
et non: codeblocks-10.05-setup.exe

Page de téléchargement sur le site officiel de Code::Blocks

Mingw est un compilateur (nous y reviendrons plus loin) si vous ne prenez pas le bon vous pourrez coder vos applications mais vous ne pourrez pas les transformer en application! Bref l'IDE ne servirait pas à grand chose.

Une fois le téléchargement terminé, installez le logiciel (ça ne devrais pas être trop dur ).
Il vous suffit de cliquer sur les boutons "suivant" quasiment à chaque fois.

Voila, maintenant lancez code::blocks, vous devriez voir apparaître ceci:

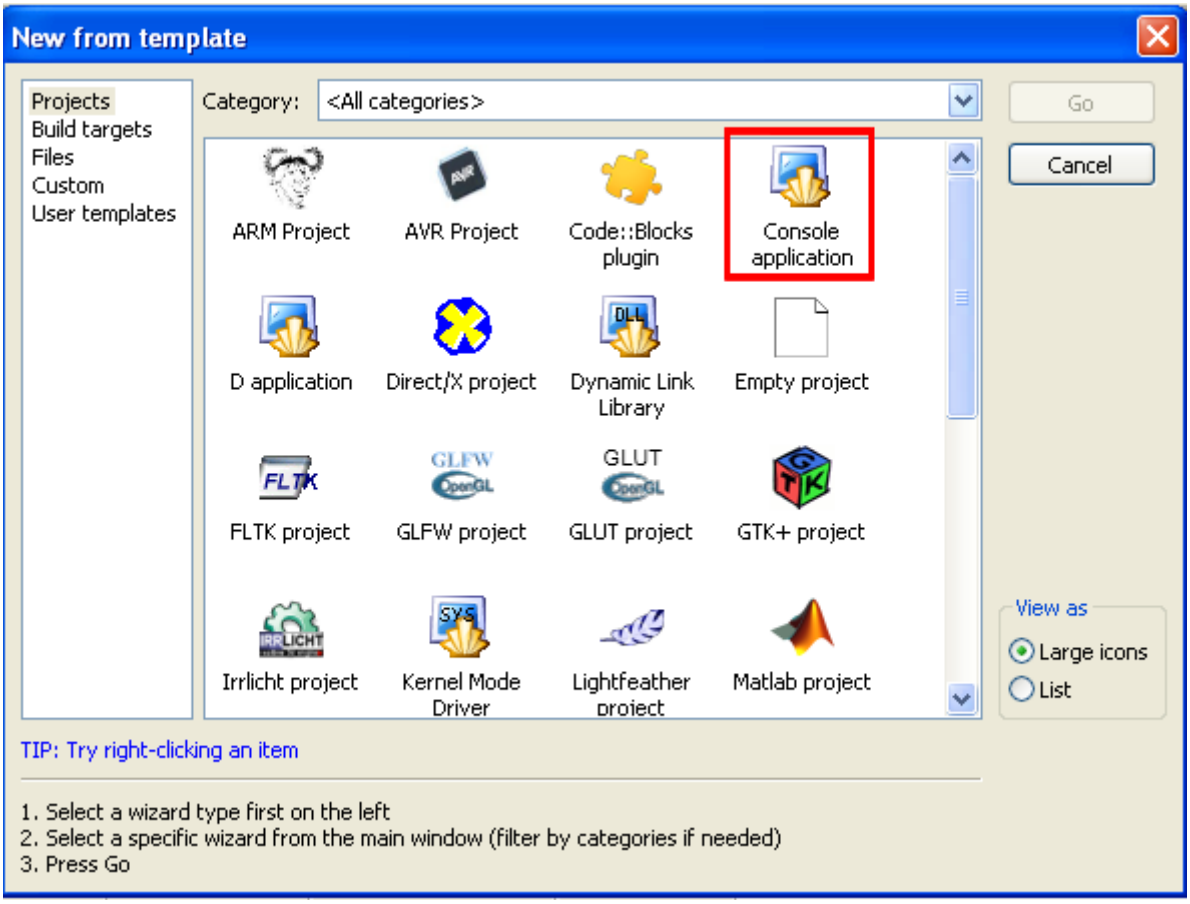


Attention! La version que j'utilise est la 10.05. Si vous utilisez une autre version il se peut qu'il y ai quelques changements mineurs dans la suite du tutoriel mais c'est sans importance =)

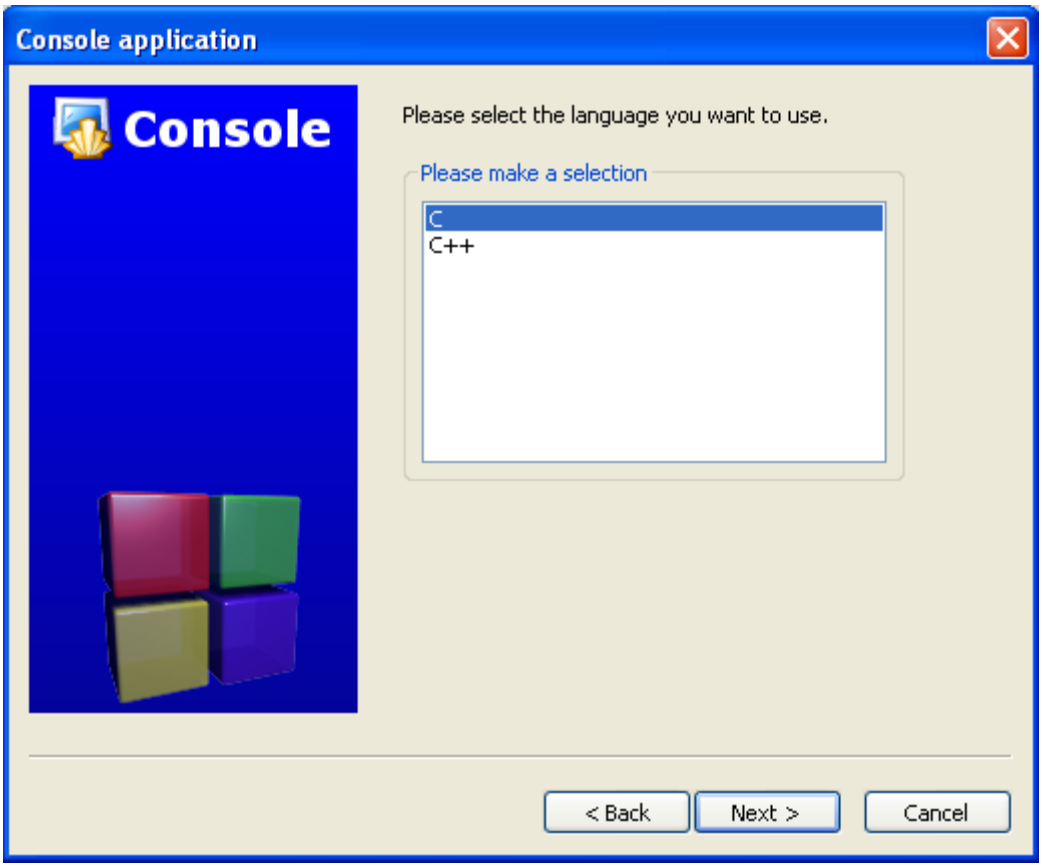
Voilà vous êtes maintenant prêt à pouvoir coder vos premières superbes applications! Une dernière chose avant cela: créer un nouveau projet.

Pour cela, cliquez sur File --> New --> Project.

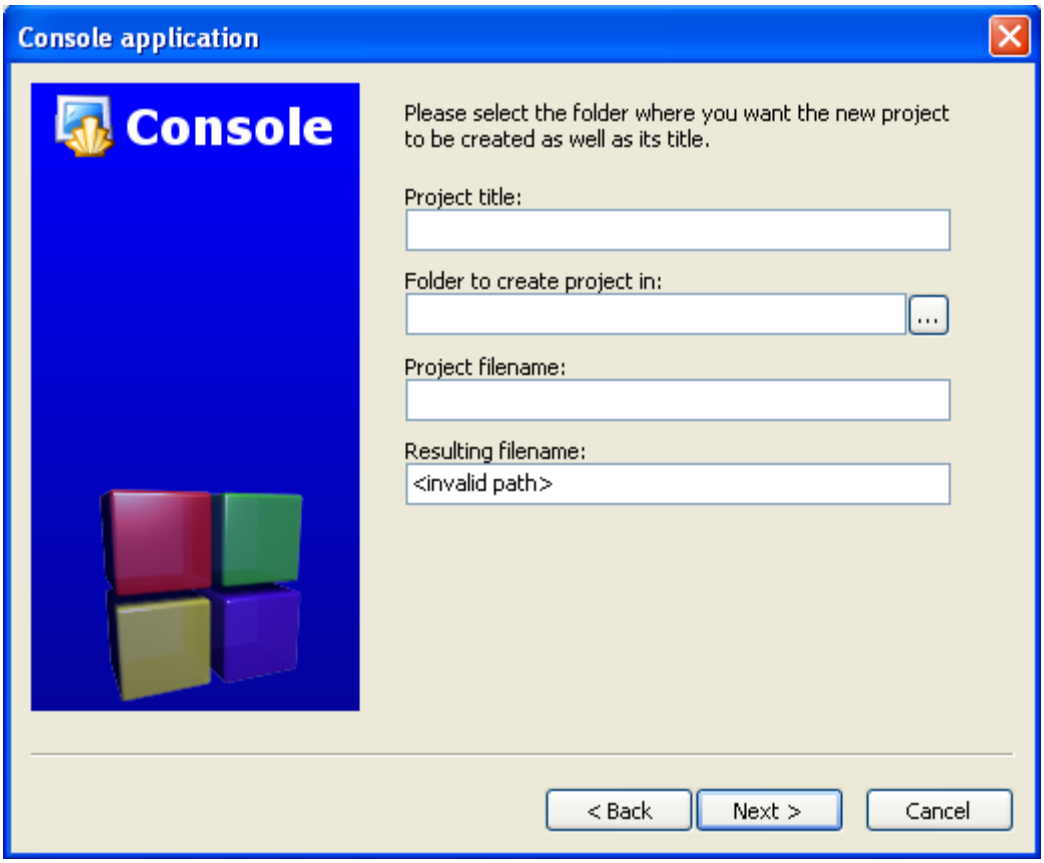
Choisissez "console application" puis cliquez sur "Go".



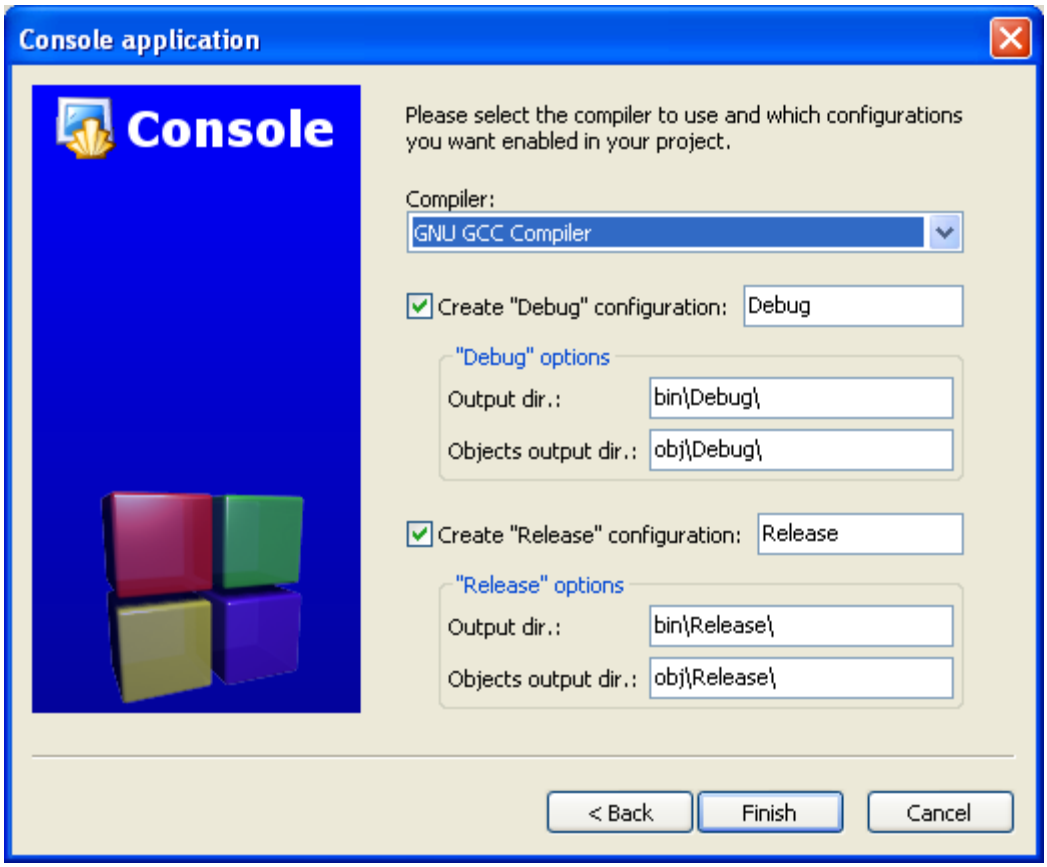
Cliquez sur Next, selectionnez C (vous pouvez aussi choisir C++ mais dans ce tutoriel nous n'aborderons que le langage C) et re-cliquez sur Next.



Maintenant choisissez le nom de votre projet dans la case "Project title", l'emplacement de votre projet en cliquant sur les "...".
Une fois ces étapes faîtes, cliquez une nouvelle fois sur Next.



Enfin, lorsque vous arrivez à cette étape:



cliquez tout simplement sur "Finish" sans **RIEN** changer!

Et voila, vous êtes maintenant près à coder!

Pour cela rendez-vous au chapitre suivant afin de commencer vraiment l'apprentissage de ce merveilleux langage de programmtion qu'est le C :)

Les bases du langage C

Nous y voici donc enfin.
Vous allez réelement pouvoir apprendre à programmer à partir d'ici :)

Je vous propose donc de commencer en douceur l'apprentissage de ce langage par l'analyse du code précédement vu, le fameux "Hello World!".

Première analyse de code C:

Lorsque vous créez un nouveau projet en C de type console, le code par défaut que code::blocks vous affiche est le suivant:

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

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

Nous allons voir pas à pas ce que signifie chaque ligne de ce code.

Les deux premières s'appellent les directives de processeur. Elles sont très utiles mais nous y reviendrons dans quelques instants et plus tard beaucoup plus en détail.

La troisième ligne de code est une ligne obligatoire pour n'importe quel programme écrit en C!

Notez d'ailleurs qu'elle peut aussi s'écrire de cette façon:

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

Nous utiliserons les deux dans ce tutoriel et nous verrons pourquoi elle peut s'écrire de deux façons différentes plus tard 

Cette ligne sert tout simplement à dire à l'ordinateur que le programme débute réelement ici. Elle est suivie d'une accolade ouvrante et nous voyons qu'à la fin du code il y en a une fermante. C'est très simple:

Tout le code de votre programme se situera obligatoirement entre ces deux accolades!

Cela s'appelle une fonction (comme en mathématiques). Tout votre code doit se situer dans cette fonction et l'ordinateur se chargera de l'éxécuter.

Une fonction est très pratique en programmation!
Elle permet de dire à l'ordinateur utilise cette fonction là! Cela permet d'éviter de recopier des centaines de lignes de code pour éxécuter la même opération... 
Une fonction est donc quelque chose qui emboîte en un nom des dizaines d'intructions!

La cinquième ligne de code, elle, sert à afficher du texte. C'est une instruction.

Explications:

Vous demandez à votre ordinateur de commencer le programme (il voit la ligne int main...).
Il exécute donc ce qui se trouve entre les accolades et voit donc printf. Il sait qu'il s'agit d'une fonction. Il va donc éxécuter printf avec les arguments que vous lui avez passé, c'est à dire: Hello world!

Vous allez me dire: Oui mais comment l'ordinateur sait qu'il s'agit d'une fonction?

C'est très simple. Vous vous rappelez des premières lignes? Les includes là (ou directives de processeur pour ceux qui ont une bonne mémoire ) et bien ils disent tout simplement à l'ordinateur:

ajoute les fichiers que je te dis avec ce programme là. Et dans ces fichiers, se trouve le code de toutes les fonctions de bases du C! Comme ça quand vous voudrez afficher du texte il vous suffira juste d'écrire printf et non les centaines de lignes super complexes de code normalement nécéssaires!

C'est ce que l'on appelle des bibliothèques!
En gros vous pouvez vous dire que vous avez énormément de chance car des programmeurs ont avant vous codé toutes ces fonctions super complexes et les ont réduite à quasiment une seule ligne pour pouvoir les utiliser simplement!

Il y a enfin la dernière ligne de code entre les accolades qui est "return 0;"

Là, c'est le plus simple! Cela dit juste à l'ordinateur que le programme est terminé et que l'on peut fermer la fenêtre 

Si on récapitule ce code:

Etape 1: On dit à l'ordinateur de se débrouiller avec les fichiers parce que on a pas envie de ré-écrire toutes les fonctions.

Etape 2: Le programme commence, c'est la fonction main.

Etape 3: On affiche le texte Hello world à l'écran, car c'est l'instrution printf.

Etape 4: Le programme est terminé, on arrête tout.

Si vous avez compris ça, vous avez de bonnes bases pour continuer! Sinon recommencez la lecture jusqu'à être sur d'avoir compris. Si un point ne vous semble vraiment pas clair, n'hésitez pas à poser une question dans questions à propos d'un tutoriel =)

Je ne sais pas si vous vous posez des questions mais en tout cas il y a encore bien au moins deux choses d'étrange dans ce programme:

-Les point-virgules à la fin de certaines lignes.
-Le " \n " après le hello world.

Commençons par les points-virgules:

En C toutes instructions se terminent par un point-virgule!

Je crois que c'est on ne peut plus clair

C'est pour cela qu'il y a un point-virgule après printf, c'est une instruction! On demande à l'ordinateur de faire cette action (afficher du texte), il s'agit donc d'une instruction! Même chose pour le return 0, on demande à l'ordinateur de terminer le programme.

Il n'y a pas de point-virgule à la fin du main car ce n'est pas une instruction! On marque juste le début du programme, mais l'ordinateur, lui, ne fait rien!

Quant aux directives de processeur, vous allez me demandez pourquoi elles n'ont pas de point-virgule alors qu'elle demande bien quelque chose, à savoir inclure des fichiers.

Nous y reviendrons dans la partie suivante intitulée "Compilation".

Je vous invite donc à cliquez sur le lien ci-dessous pour pouvoir enfin comprendre un certain nombre de chose d'un coup, et de pouvoir lancer votre premier programme écrit en C

La compilation

Nous allons ici voir comment créer, à partir du code écrit en C, différents jeux/applications.

Il s'agit de la compilation. Je vous ai dis tout à l'heure que l'ordinateur ne comprenait que le binaire. Mais je vous avait aussi dis qu'il était inutile d'apprendre le binaire car trop compliqué et d'apprendre plutôt un "vrai" langage de programmation car beaucoup plus simple.

Mais alors, si l'ordinateur ne comprend que le binaire, comment fait-on pour créer des programmes en C? La réponse c'est le compilateur qui vous la donne.

Oui le compilateur, vous savez, mingw. Mingw est un programme qui convertit le langage de programmation choisit (ici le C) en binaire en lui apportant parfois des améliorations. La compilation est donc le fait de "traduire" un langage de programmation en binaire.

Il existe un compilateur pour chaque langage de programmation différent, ce qui est tout à fait logique, car si vous prenez un traducteur anglais(langage de programmation)-français(binaire), il ne pourra pas traduire l'anglais en espagnol par exemple.

Le code source (code écrit dans un langage de programmation, ici le C) se transforme alors, grâce au compilateur, en **fichier éxécutable**.

Je vous ai dis il y a quelques instants que le compilateur apportait des optimisations/modifications dans votre code source.

Par exemple nos fameuses lignes "directives de processeur", elle disent au compilateur d'inclure à la compilation les fichiers indiqués! Comme nous le verrons plus tard dans le cours, certains éléments du langage C se font à la compilation.

Commandes sous code::blocks:

Pour **compiler** votre code source en programme éxécutable, il vous faut cliquer sur le bouton "build" de code::blocks ou en faisant le raccourcis clavier Ctrl+F9.

Cela va compiler votre code source actuel (qui doit être celui des exemples précédents (hello world)) et va vous créer le fichier éxécutable dans le dossier "bin" qui se trouve dans le répertoire de votre projet.

Pour lancer le fichier ainsi créé à partie de code::blocks, vous pouvez cliquez sur "Run" ou appuyer sur F10.

Enfin si vous voulez enchaîner les deux à la suite, appuyer sur F9 ou clqiuez sur "Build and run".

Boutons Build, Run, Build and run

Voila, c'est tout ce dont vous avez besoin de savoir pour l'instant sur la compilation à proprement parlé.
Cette partie du cours, très petite était, néanmoins nécéssaire afin de comprendre pleinement le système de compilation.

Je pense qu'il est temps de commencer les choses sérieuses, dans le chapitre suivant nous

allons voir deux choses sur le langage C: Les commentaires et les variables.

Soyez attentif, le chapitre sur les variables et l'un des plus important!

Les variables. Nous y voilà enfin. Comme je vous le disais, ce chapitre est l'un des plus important dans le domaine de la programmation C et pour bien d'autres langages d'ailleurs. Sans les variables, on ne pourrait quasiment rien faire!

Mais en fait, qu'est-ce qu'une variable?

Une variable, c'est comme une inconnue en mathématique. Vous savez les X? Et bien en programmation on travaille surtout avec des nombres...que l'on ne connait pas  .

"Une variable en programmation est une information temporaire que l'on stocke dans la mémoire vive"

D'ou son nom, (variable), car elle peut varier au cours du temps.

Derrière cette définition de dictionnaire se cache quelque chose d'assez simple et logique.

Exemple:

Prenez un jeu comme Mario. Et bien le nombre de pièces récoltées est stocké dans une variable car la valeur peut changer si Mario attrape une pièce de plus.

Il existe plusieurs sorte de variable. On peut faire l'analogie d'une variable avec une boîte.

Il y a plusieurs sortes de boîtes. Des petites, des grandes et des très grandes.

Une variable peut stocker n'importe quoi (un chiffre, une lettre, un mot, une phrase).

Les "petites" boîtes servent aux nombres.
Les "grandes" boîtes servent aux lettres.
Les "très grandes" boîtes servent aux mots/phrases (plusieurs lettres).

Une variable (en C) à besoin d'être déclarée. Cela signifie que l'on doit dire à l'ordinateur que nous réservons un espace de sa mémoire pour cette variable et que celle-ci va contenir uniquement une sorte de boîte. C'est à dire que une fois la variable déclarée, nous pouvons mettre, dans celle-ci, uniquement le type donné (une lettre, un chiffre, ...).

Une variable se déclare comme ceci en C:

Type nom;

Le type, c'est la grandeur de la boîte qui sera nécessaire.
Bien entendu, les variables ne s'appellent pas vraiment "Boîtes" mais plutôt comme ceci:

int
char
double
float
long

et bien d'autres encore....

Oui vous allez devoir les apprendre par coeur mais ne vous inquietez pas, vous n'allez probablement n'utiliser que quelques unes d'entre elles 

Ce qu'il faut retenir **pour l'instant** ce sont ces deux types là:

int
char

Le type int sert à stocker des nombres tandis que le type char à stocker des lettres et ou des mot/phrases entières.

donc **temporairement** nous allons dire que char sert a stocker des lettres et mots alors que int à stocker des chiffres.

Revenons à la déclaration des variables:

type nom;

Nous connaissons maintenant les types. Dons si vous avez suivis et que je vous demande de me déclarer une variable qui servira à stocker un chiffre, vous allez m'écrire

.....

int nom;

J'espère que vous aviez trouvé :P
Sinon revoyez ce chapitre fondamental!

Passons maintenant au nom de la variable

Le nom c'est très simple, c'est vous qui le choisissez! En respectant cependant certaines conditions:

1. Il ne doit pas y avoir deux fois une déclaration de variable sous le même nom!

2. Le nom de la variable doit obligatoirement commencer par une lettre (minuscule ou majuscule peut importe).

3. Vous pouvez utiliser des lettres majuscules, minuscules et des chiffres dans le nom.

4. Les espaces sont interdits.

5. Il ne doit pas y avoir de caractère accentué dans le nom de la variable.

6. Il ne doit y avoir aucun autre signe différent des lettres majuscules, minuscules, des chiffres à l'exception du underscore _ (tiret bas) qui est lui possible.

7. Le nom ne doit pas être le même que un déjà utilisé par le langage C (char, int, printf, if, etc...).

Exemples de déclarations de variables correctes:

```
int essaiVariable;

int bonjour;

int essai365;

int machin_bizarre;
```

Exemples de déclarations de variables incorrectes:

```
int 36bonjour;

int char;

int éssai;

int essai variable;
```

int variable-jeu;

Donc quelque soit le type de la variable, elle se déclare toujours de la même façon.

Vous savez donc maintenant déclarer une variable, c'est pourquoi je vous invite à lire la suite qui cette fois-ci traitera des opérations qu'une variable peut éxécuter.

Bref que du nouveau et c'est à partir du prochain chapitre que vous pourrez vraiment commencer la programmation et vous verrez que vous ferrez des choses déjà assez sympa à la fin du chapitre suivant =)

Travaillons avec les variables

Pour l'instant vous savez déclarer une variable mais sa serait quand même pas mal de s'en servir 

Nous allons voir, dans un premier temps, les propriétés de chaque type de variable, puis nous allons commencer à effectuer des opérations sur celles-ci et même commencer à les afficher à l'écran 

Les types de variables:

A la base, les différents types de variables étaient prévus pour économiser le maximum de mémoire.

Il faut savoir qu'un ordinateur ne peut stocker que des chiffres.

Et bien voici une liste (non exhaustive) qui indique les types de variables, et leur limite (car oui il y a une limite de taille pour chaque type de variable):

int permet de stocker des nombres allant de -2 147 483 648 à 2 147 483 647.

char permet de stocker des nombre allant de -128 à 127.

long permet de stocker des nombres allant de -2 147 483 648 à 2 147 483 647.

float permet de stocker des nomrbes allant de -3.4 x 10 puissance 38 à 3.4 x 10 puissance 38.

double permet de stocker des nombres allant de -1.7 x 10 puissance 308 à 1.7 x 10 puissance 30.

Je vous rassure tout de suite, il ne sert à rien d'apprendre par coeur cette liste! Il faut juste connaître le nom des types de variables écrit en rouge.

Nous allons commencer par expliquer le type int. Comme nous venons de le voir, il peut stocker de très grands nombres et c'est pour cela que je vous conseille de l'utiliser quasiment à chaque fois que vous avez besoin d'utilisez des nombres dans votre programme.

Attention! Le type int ne peut stocker que des nombres entiers et en aucun cas des nombres décimaux (nombres à virgule)!

Char permet de stocker des nombres mais il est plus utilisé pour stocker...des lettres!
Car pour un ordinateur une lettre est un chiffre! Et donc un mot, un ensemble de chiffres.
Il permettera donc de stocker une ou plusieurs lettres.

Double permet de stocker de très grands nombres, mais en plus de cela, vous pouvez y stocker des nombres à virgule (par ailleurs dans un nombre décimal, la virgule est en fait un point)!

Vous pouvez donc en conclure que si vous souhaitez stocker un nombre entier, utilisez int. Pour un nombre décimal utilisez double et enfin pour une lettre ou un groupe de lettres utilisez char.

Affecter une valeur à une variable:

Pour cela rien de plus simple, il suffit d'utiliser le signe égal. Par exemple:

```
int nombreDeVie;
nombreDeVie = 3;
```

Cela créer une variable de type int qui vaudra 3.

Remarquez que là j'ai "détaillé" l'opération, mais j'aurais aussi bien pu écrire:

int nombreDeVie = 3;

directement.

Cela revient strictement au même!

Vous pouvez aussi affecter la valeur d'une variable dans une autre en faisant comme ceci:

int NombreDeVie1 = 2;

int NombreDeVie2 = NombreDeVie1;

Cela créer une variable "NombreDeVie2" qui vaut la même valeur que "NombreDeVie1".

Pour le type double, cela est identique:

double NombreDeVie = 0;

double machin = NombreDeVie;

Les variables "NombreDeVie" et "machin" vaudront toutes les deux zéros.

Vous voyez, c'est super simple! Là où c'est plus marrant, c'est pour le type char...^^

Le type char permet de stocker un seul caractère. Mais nous verrons plus loin que l'on peut y stocker en fait quasiment n'importe quelle grandeur de texte (en trichant un peu :P).

Le type char se déclare comme les autres type de variable:

char Lettre;

La différence est qu'elle peut contenir un caractère (notez bien que ce caractère peut être un chiffre comme une lettre ou encore un caractère spécial comme un slash, un tiret, etc..).

Pour affecter un caractère à cette variable, il suffit de faire comme ceci:

```
char Lettre;

Lettre = 'T';
```

Encore une fois, j'insiste sur le fait que l'on peut aussi écrire:

```
char Lettre = 'T';
```

directement.

Attention à bein mettre les apostrophes! Sinon, cela voudrait dire que vous voulez affecter une autre variable à cette variabel ci!

mais char peut, comme je vous l'ai dis, contenir aussi un chiffre:

```
char variable = 32;
```

Voilà, maintenant vous savez affecter n'importe quelle valeur à une variable.

Afficher le contenu d'une variable:

Il serait peut être intéressant de savoir maintenant comment afficher une variable, car cela est indispensable pour quasiment n'importe quel jeu/application.

Je vous donne donc la fonction permettant ceci:

printf

Et oui! C'est la même que tout à l'heure! Nous la verrons plus en détail dans le prochain chapitre ☺

Pour afficher un type int

```
printf("%d", test);
```

le %d signal à la fonction printf que nous voulons afficher une variable de type int.

La variable à afficher est celle juste après la virgule (nommée dans mon exemple "test").

Si vous exécutez donc ce code ci:

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

int main()
{
 int variable;
 variable = 56;
 printf("%d", variable);

 return 0;
}
```

Vous vous retrouverez avec un programme qui affiche 56:

```
56
Process returned 0 (0x0)   execution time : 0.266 s
Press any key to continue.
```

La même chose est bien sure possible avec les autres types de variables!

Je vais allez légèrement plus vite sur celles-ci car seul le "%d" change ☺

Pour afficher un type double

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

int main()
{
 double variable;
 variable = 56.3345;
 printf("%lf", variable);

 return 0;
 }
```

Pour afficher un type char

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

int main()
{
 char variable;
 variable = 'B';
 printf("%c", variable);

 return 0;
 }
```

Voilà! Vous savez à présent:

Déclarer n'importe quel type de variable.
Affecter n'importe quelle valeur à n'importe quel type de variable.
Afficher n'importe quel type de variable.

Il ne nous reste plus qu'à voir comment réaliser quelques calculs très simple sur les variables 

Pour cela c'est très facile:

Vous voulez faire une addition? Sa tombe bien, c'est pas compliqué 

```
int Variable1 = 12;
int Variable2 = 8;
int Variable3;
Variable3 = Variable1 + Variable2;
```

Vous venez d'additionnner Variable1 avec Variable2 pour stocker le résultat dans Variable3.
Vous pouvez bien sur utiliser des nombres sans passer par des variables:

```
int Variable1 = 36;

int Variable = 8 + Variable1;
```

Pareil, pour réaliser une soustraction sauf que l'on utilise le signe moins (logique ). pour une multiplication, on utilise l'étoile * et enfin pour une division on utilise le slash /.

Voilà, vous savez donc maintenant réaliser:

Des déclarations de variables.
Affecter une valeur à une variable.
Réaliser des opérations mathématiques sur des variables.
Afficher des variables.

Et tout cela en langage C!

Allez, avouez que c'est pas si dur que sa! 

Surtout comprenez bien ce chapitre, il est fondamental. Pas la peine d'aller plus loins si vous ne le comprenez pas vous serez perdu!

Si un point est mal expliqué, n'hésitez pas à le signaler dans questions à propos d'un tutoriel en y expliquant la difficulté, je vous y aiderais le plus vite possible 

Dans le prochain chapitre de ce cours, nous verrons les flux d'entrées et de sorties 

Les flux d'entrées et de sorties

Ou autrement dit, comment interagir avec l'utilisateur.

Il faut savoir qu'il y a deux choses principales lorsque l'on veux communiquer avec l'utilisateur.

Il faut:

-Afficher des choses à l'écran.
-Demander des saisies de l'utilisateur.

Nous allons voir dans ce chapitre ces deux choses là 

Tout d'abord, expliquons pourquoi appelle-t-on cela flux d'entrées et de sorties:

Simplement:

-Les flux d'entrées représentent les saisies de l'utilisateur.
-Les flux de sorties les affichages à l'écran.

Les flux de sorties ou: comment afficher quelque chose à l'écran?

Pour afficher du texte à l'écran, nous allons utiliser la fonction:

printf

Une fonction est, en résumé, une instruction qui demande à l'ordinateur de réaliser quelque chose. Ici printf demande d'afficher du texte. Nous verrons les fonctions plus en détail plus tard 

Pour afficher du texte avec prinf, il faut lui dire quel texte afficher, et cela ce fait comme ceci:

```
printf("Hello world!");
```

Attention de ne surtout pas oublier le point-virgule à la fin!

Ici, nous affichons à l'écran "Hello World" (sans les guillemets).

Voici donc le code complet que vous pouvez tester:

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

int main()
{
    printf("Hello world!");

    return 0;
}
```

Je vous rappelle qu'il faut appuyer sur F9 pour compiler et lancer le programme avec code:block

Cela affichera normalement à l'écran:

Appuyez sur une touche pour terminer le programme

Regardons précisément ce qu'il y a d'affiché:

-->Hello world!

C'est le texte que l'on a demandé d'afficher.

-->Process returned 0 <0x0>

Cela montre que le programme s'est terminé sans erreurs.

-->execution time : 0.281 s

C'est le temps que le programme a mis avant de se terminer. Ici il a donc mis 0.281s à afficher un hello world (le temps doit peut être différent chez vous puisque cela dépend de l'ordinateur).

-->Press any key to continue.

En français "Appuyez sur n'importe quelle touche pour continuer", dans ce programme, le "continuer" se transformerait plutôt en "quitter".

Appuyez sur une touche pour fermer le programme.

Regardez maintenant dans le dossier Bin/Debug de votre projet, vous devriez avoir nomalement votre fichier éxécutable!

Lancez-le et là,..... c'est le drame :P

En effet votre programme se ferme aussitôt après avoir été lancé et par conséquant, impossible de voir votre beau et magnifique hello world!

Ne vous inquiétez pas! Le programme fait son travail! Vous lui demandez d'afficher du texte avec printf, ce qu'il fait. Vous lui mettez ensuite return 0; ce qui signifie arrête le programme!

Il est donc normal que vous ne puissiez pas avoir le texte et que le programme se ferme aussitôt!

Pour remédier à cela et, donc faire une pause, il vous suffit de mettre une de ces deux instructions avant le return 0; :

```
system("pause");
```

ou

```
getchar();
```

Attention! La fonction system ne fonctionne que sur windows et n'est pas recommandée, car il y a une faille de sécurité dans la conception de cette fonction qui permet de faire pas mal de chose malveillantes. A utiliser le moins possible donc et avec prudence.

PS: En réalité getchar() n'est pas une fontion de pause, elle attend simplement l'appui d'une touche.

Nous allons donc choisir ici d'utiliser getchar qui attent que l'utilisateur appuie sur une touche:

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

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

Ce qui va résoudre le problème de la pause. Voilà vous pouvez maintenant distribuer à vos amis votre premier programme sans qu'il ne se ferme tout de suite

Bon OK il n'affiche que du texte mais c'est un bon début

Maintenant voyons plus en détails la fonction printf(); :

Essayez tout d'abord d'afficher le texte "Bonjour à tous!" (sans les guillemets). Allez, c'est facile! Voici tout de même le code (j'espère que vous l'aviez trouvé :P) :

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

int main()
{
    printf("Bonjour à tous!");
    getchar();
    return 0;
}
```

Et maintenant lancer-le, ce qui vous donne.....:

```
Bonjour ó tous!
```

Mais pourquoi il affiche ò alors qu'on lui demande d'afficher à? @_@

Le problème vient du fait que la console windows...ne gère pas les accents! Donc il va falloir ne pas utiliser d'accents dans la console

Donc si vous entrer un texte avec des accents pour les afficher à l'écran, vous risquez de vous retrouver avec des caractères spéciaux dans la console.

Il nous reste encore deux choses à voir avec la fonction printf:

1) Les sauts de lignes et autres commandes de mises en forme.
2) L'affichage de variables.

Les sauts de lignes et autres commandes de mises en forme

Imaginons que vous souhaitez faire un saut de ligne avant le return 0; de code::block comme indiqué en rouge ici:

au lieu de

et bien c'est possible (je pense que vous vous en doutiez légèrement )! Vous pouvez même faire autant de sauts de lignes dans la console que vous souhaitez!

Pour cela, il suffit d'écrire \n à l'endroit où vous souhaitez faire un saut de ligne. Par exemple, si vous souhaitez faire le même affichage que l'illustration d'exemple ci-dessus, il vous suffit d'écrire:

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

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

\n est considéré comme un caractère à part entière. On dit que c'est un carctère d'échappement. Voici une petite liste des principaux caractères d'échappements en C:

\n nouvelle ligne

\t tabulation horizontale

\v tabulation verticale

\b retour en arrière

\a signal sonore

\\ backslash

\' apostrophe

\" guillemet

L'affichage de variables

C'est certainement la chose qui va nous servir le plus: afficher des variables.

Imaginons que vous vouliez afficher, par exemple, le nombre de vie d'un joueur. Ce chiffre est stocké dans une variable puisqu'il peut changer.

Pour afficher une variable avec printf(); en C, tout dépend de son type.
On n'affichera donc pas de la même manière un type int d'un type char.
Nous allons donc voir un par un les différents modes d'affichages mais rassurez vous, ils fonctionnent quasiment tous de la même manière, c'est à dire comme ceci:

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

int main()
{
    printf("type d'affichage", variable);
    return 0;
}
```

Le type d'affichage est un symbole qui commence toujours par '%' suivi d'une lettre correspondant au type de la variable que vous souhaitez afficher. 

Voici les diférents symboles permettant l'affichage de variables:

int --> %d (d pour décimal)

char --> %s (s pour string (chaine en anglais))

double --> %f

float --> %f

long --> %ld

Donc si vous souhaitez afficher une variable de type int, voici comment s'y prendre:

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

int main()
{
    int ma_variable = 45;
    printf("Ma variable vaut %d", ma_variable);
    return 0;
}
```

Cela affichera:

```
Ma variable vaut 45
```

Vous voyez, c'est pas si compliqué que cela  Essayez de recopier sans aide et de mémoire ce petit code en entier en essayant de comprendre ce que vous faîtes 

Dans le chapitre suivant, nous allons voir la gestion des entrées et nous pourrons enfin vraiment communiquer avec l'utilisateur et commencer... les TP 

La gestion des flux d'entrées

La gestion des flux d'entrées est un nom barbare pour désigner...une saisie du clavier (oui, vive l'informatique et les mots compliqués ).

Elle permet donc de récupérer ce que l'utilisateur à rentré au clavier, c'est donc super utile!

Nous allons donc utiliser une nouvelle fonction, j'ai nommé..... scanf(); !

Elle s'utilise comme ceci:

```
scanf("typeDeVariable", AdresseVariable);
```

typeDeVariable est le même simbole que pour printf();, donc lorsque vous connaissez les simboles de printf();, vous connaissez ceux de scanf();!

La deuxième chose à donner (le fait de donner une chose à une fonction s'appelle donner un argument, mais nous verrons tout cela un peu plus tard ) est l'adresse de la variable.

Cela sert à stocker ce que l'utilisateur rentre dans la variable que vous souhaitez!

Mais attention! Il faut donner l'adresse de votre variable, et non la variable! Je vous expliquerais tout cela plus loin car c'est assez difficile à maitriser du premier coup.

Sachez juste que pour donner l'adresse d'une variable on utilise le symbole '&' suivit de la variable concernée et cela pour chaque type de variable sauf le type char.

Par exemple, si l'on souhaite récupérer un nombre que l'utilisateur nous donne, il faut faire comme ceci:

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

int main()
{
   int variable;
   printf("Veuillez rentrer un nombre: ");

   scanf("%d", &variable); /*On récupère la saisie de l'utilisateur et on la stocke dans la
```

```
variable "variable" de type int.*/

                printf("Vous avez rentre le nombre %d", variable);

                return 0;
            }
```

Voyons donc ce que fait ce programme:

1) On créer une variable se nommant "variable".
2) On demande d'afficher le texte "Veuillez rentrer un nombre" à l'écran.
3) On demande de mettre le nombre saisit dans la variable de type int se nommant "variable".
4) On demande d'afficher la variable.

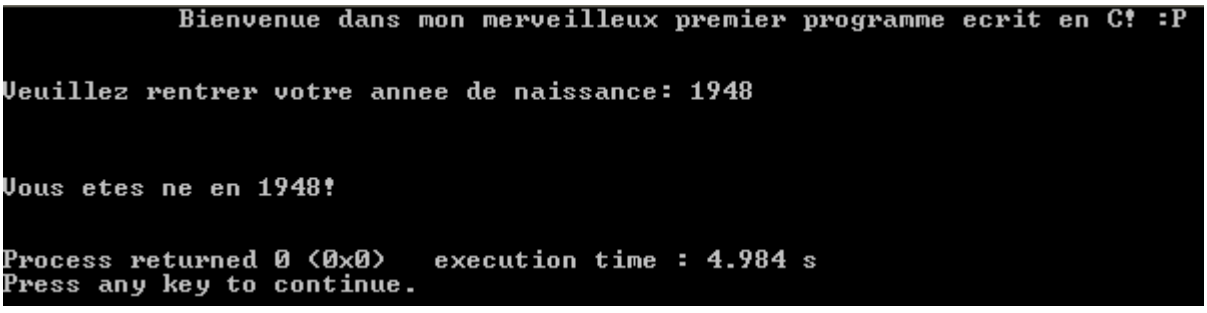
Je le répète, pour donner l'adresse mettez simplement l'esperluette '&' devant le nom de la variable qui doit contenir la saisie de l'utilisateur.

Pour l'instant n'essayez pas ceci avec le type char, on va pour le moment se contenter de demander des nombres ☺

Tenez, faites ce petit exercice pour vous entrainer:

Ecrivez un programme qui affiche un message de bienvenue. Indiquez ensuite à l'utilisateur de rentrer son année de naissance. Enfin, une fois son entrée validée, affichez une phrase comme "Vous êtes né(e) en X!" (où X est remplacé par l'année).

Voici une image du programme en éxécution pour que vous puissiez bien voir la chose ☺:



Trois, deux, un, ...Codez!

.....

C'est fini? Allez quoi c'était pas compliqué

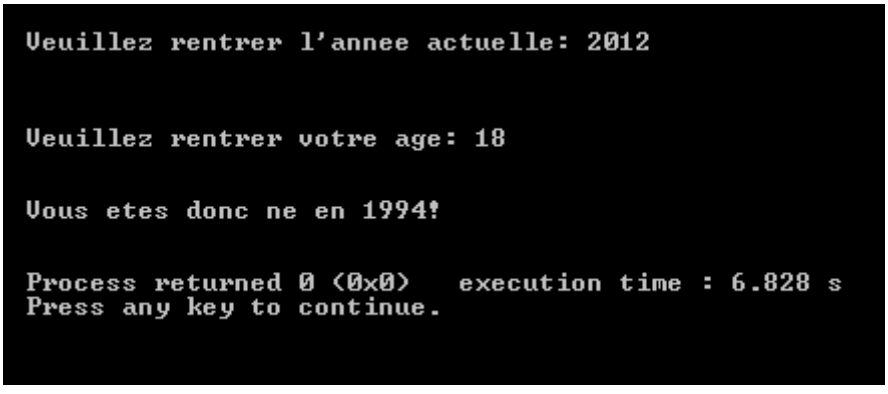
Bon, voici le code pour ceux qui voudrait le voir:

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

int main()
{
 int AnneeDeNaissance = 0;
 printf("        Bienvenue dans mon merveilleux premier programme ecrit en C! :Pnnn");
 printf("Veuillez rentrer votre annee de naissance: ");
 scanf("%d", &AnneeDeNaissance);
 printf(" \n \n \nVous etes ne en %d! \n \n", AnneeDeNaissance);
 return 0;
 }
```

J'espère que vous aviez trouvé sinon revoyez le chapitre jusqu'à temps d'avoir réussi sans model ce petit exercice

Vous pouvez aussi faire un deuxième exercice dans lequel vous demandez l'année actuelle, puis l'âge du spectateur pour que votre programme calcule son année de naissance!

Voici ce que cela donnerait:



Et voici le code pour vous corriger ou pour voir comment faire si vous êtes bloqué (ce qui doit être

évité, car il n'y a rien de dur là dedans):

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

int main(int argc, char *argv[])
{
  int AnneeActuelle = 0;
  int Age = 0;
  int AnneeTrouvee = 0;

     printf("Veuillez rentrer l'annee actuelle: ");
     scanf("%d", &AnneeActuelle);
     printf(" \n \n \nVeuillez rentrer votre age: ");
     scanf("%d", &Age);

     AnneeTrouvee = AnneeActuelle-Age;

     printf(" \n \nVous etes donc ne en %d!nn", AnneeTrouvee);

  return 0;
 }
```

Voilà, ce chapitre est maintenant terminé. J'éspère qu'il a été suffisamment clair et précis sinon n'hésitez pas à poser des questions ici

Vous voilà déjà armé de bonnes bases pour continuer. Le prochain chapitre sera certainement le plus simple de ce tutoriel car il parlera des commentaires, chose très utile en programmation

Je vous invite donc à cliquer une fois de plus sur le lien ci-dessous pour découvrir une nouvelle facette de la programmation

Les commentaires

Ce chapitre sera court, simple et très rapide à comprendre bref du vrai bonheur!

Avant d'apprendre à créer des commentaires nous allons voir leurs utilités ainsi que qu'est-ce qu'un commentaire.

Qu'est-ce qu'un commentaire?

Dans votre code, vous aimeriez peut être écrire du texte pour vous rappeler que telle instruction fait ceci, l'autre cela, etc..
Cependant, vous ne souhaitez pas que ces informations apparaissent à l'écran.
C'est ce que l'on appelle des commentaires.

Grâce à eux vous pouvez commenter votre code sans que rien ne change.

Pour l'instant vous ne faîtes qu'apprendre le C et créer des programmes assez simples mais imaginez à la fin de ce tutoriel ce que vous saurez/pourrez faire!

Mettons nous à rêver un instant:

Vous programmez un super jeu et vous avez écrit des milliers de lignes de code! Seulement voilà, vous ne vous y retrouvez plus et vous ne saurez plus qu'elle fonction fait quoi, etc... **bref vous serez perdu.**

Comme nous venons de le voir les commentaires sont très utiles. Il ne nous reste plus maintenant qu'à savoir en créer, et vous allez voir, c'est super simple 

Comment créer des commentaires?

Déjà ce qu'il faut savoir c'est que contrairement aux variables par exemple, vous pouvez mettre tous les signes que vous souhaitez dans les commentaires (lettres accentuées, espaces, etc...)

Il existe deux types de commentaires:

Les commentaires unilignes --> Vous pouvez écrire un commentaire juste sur la ligne désignée.

Les commentaires multilignes --> Vous pouvez écrire des romans d'autant de pages que vous le souhaitez avec (bon faut pas abuser un :P).

Les commentaires unilignes

Pour créer un commentaire uniligne, il suffit de marquer deux slashs consécutifs puis de

marquer votre commentaire. Vous verrez que sous code::block le commentaire prend une couleur grisée.

Comme ceci:

```
//On inclue les librairies (bibliothèques).
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello world"); //Affiche le texte "Hello world" à l'écran.

   //Quitte le programme.
   return 0;
}
```

Danc cet exemple la présence de commentaires ne sert à rien tellement le code semble évident, il s'agissait juste pour moi de vous montrer l'utilisation des commentaires unilignes ☺

Attention!
Ce commentaire ci:

```
//Un exemple de commentaire,
blablabla.....
```

est faux!

Avec le commentaire uniligne il est interdit de faire des retours à la ligne, c'est à dire que tous doit tenir sur une seule ligne!
Pour palier cet inconvénient, les programmeurs ont inventés le commentaire multiligne.

Le commentaire multiligne

Pour ce type de commentaire, il suffit d'écrire un slash suivi d'une étoile puis d'écrire son commentaire sur plusieurs lignes si on le souhaite et de "fermer" le commentaire en écrivant cette fois une étoile suivit d'un slash comme ceci:

```
/*On inclue les librairies
(bibliothèques) permetant le
bon fonctionnement du programme*/

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

```c
int main(int argc, char *argv[])
{
    printf("Hello world"); /*Affiche le texte "Hello world"
                             à l'écran.*/

    /*Quitte le programme.*/
    return 0;
}
```

Comme vous pouvez le voir, il n'y a rien de difficile ☺

 Ne mettez pas de commentaires dans d'autres commentaires, cela n'est pas recommandé du tout! 

Exemple (**code non correct**):

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

int main(int argc, char *argv[])
{
    printf("Hello world"); /*Affiche le texte "Hello world"
                             /*autre commentaire*/ à l'écran.*/
    return 0;
}
```

C'est tout pour les commentaires ☺Vous étiez prévenu, ce chapitre sera l'un des moins difficile mais c'est par contre l'un des plus utile! Ne sous-estimez pas les commentaires sans pour autant en écrire plus que de lignes de code!

Dans le prochain chapitre, nous verrons notre premier "vrai" TP! Un programme qui convertit les degrés Celsius en degrés Fahrenheit ☺

TP: Convertion celsius-Fahrenheit

Bienvenue dans ce premier "vrai" TP où nous allons pouvoir réutiliser tout ce que nous avons vu depuis le début ☺

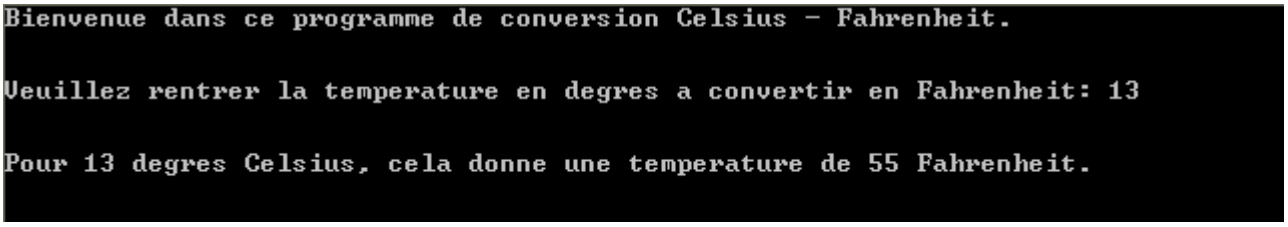
Je vous conseille de régulièrement vous entraîner de votre côté sans l'aide de ce cours à

programmer car c'est en faisant des erreurs et en programmant que vous vous améliorerez.

Ce TP sera extrêmement simple. Il s'agira de créer un programme permettant de convertir les degrés Celsius en Fahrenheit.

Durant ce TP je ne vous aiderais sur aucun point si ce n'est que pour vous donner la formule de convertion

Sur ce je vous invite à regarder ce qu'il va falloir réaliser seul:



Allez quoi c'est pas compliqué! :P

Bon je vous donne la formule pour convertir des degrés Celsius en degrés Fahrenheit :

$$°F = ((9 \times °C) / 5) + 32$$

Donc par exemple si vous voulez trouver combien font 30 degrés Celsius il faut faire l'opération:

$$((9 \times 30) / 5) + 32 = 86$$

Donc 30 degrés Celsius font 86 degrés Fahrenheit

Allez! 3-2-1... Codez!

.....

Fini! J'espère que vous avez réussi car ce n'était pas très compliqué

Mais pas de panique! Si vous n'avez pas réussi, recommencez en prenant votre temps

Je vous propose ma solution, mais il faut savoir qu'il n'y a pas vraiment, en programmation, de bonnes ou mauvaises solutions. Le but étant que le programme fonctionnne correctement en utilisant le moins de ressources possibles 

Voici mon code:

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

int main()
    {
int degres = 0;

printf("Bienvenue dans ce programme de conversion Celsius - Fahrenheit. \n \n \n
  Veuillez"      "rentrer la temperature en degres a convertir en Fahrenheit: ");

scanf("%d", &degres);

printf("nnPour %d degres Celsius, cela donne une temperature de %d Fahrenheit."
"nnn", degres, ((9*degres)/5)+32);
return 0;
}
```

Voilà  Il n'y avait pour ce TP que quelques façon de réaliser ce programme. On pouvait stocker aussi le résultat de la convertion et afficher la variable mais pour les vraiment "puristes" on pouvait "économiser" une variable en écrivant directement l'opération dans le printf();.

Une choses qui vous a peut être choqué dans mon code, c'est l'affichage.

J'ai écris comme ceci:

```
printf("debut du texte"
        "fin du texte");
```

et bien cela est correct! Et oui, c'est comme si on écrivait le texte de cette façon:

printf("debut du texte fin du texte");

Voilà, le TP étant terminé, cela vous a permis de réaliser votre premier programme vraiment "utile" et d'apprendre une nouvelle chose sur la fonction printf();.

Nous allons voir dans le chapitre suivant les conditions en langage C. Les conditions sont une notion essentielle à comprendre, car sans cela vous ne pourrez jamais rien faire d'autre que des petit programmes comme nous venons de le faire (et nous ce que l'on veux c'est réaliser des super jeux :P)

Allez en route pour le sixième chapitre de ce tutoriel sur le langage C! 

Les conditions

Les conditions sont la base des programmes un temps soit peu évolués. Elles permettent de ne pas faire toujours la même chose et d'éxécuter une partie de code que si la condition est remplie.

Prenons un exemple. Nous demandons à l'utilisateur de rentrer son âge. Nous voudrions afficher "Tu es un adulte!" ou "Tu es encore un enfant!" selon l'âge rentré (si il a plus de 18 ans, on affichera la première phrase, sinon la deuxième).

Voyons maintenant comment créer une condition:

```
if (condition)
{
  //Code si la condition est vraie.
}
```

Vous voyez par vous même que ce n'est pas très dur. En plus le "if" veut dire "si" en anglais, ce qui donne littéralement:

Si [la condition] alors je fais ce qu'il y a entre accolades.

Passons maintenant à la condition à proprement parlé.

Nous pouvons demander, en C plusieurs choses en conditions, les voici:

-Si égal.
-Si supérieur.
-Si inférieur.

-Si inférieur ou égal.
-Si supérieur ou égal.
-Si différent de.

Nous allons les voir un par un (rassurez vous il n'y a qu'un tout petit détails qui change à chaque fois).

Si égal:

```
int mavariable = 25;

  if (mavariable == 25)
  {
    //Code si la condition est vraie.
  }
```

Dans cet exemple on regarde si la variable "mavariable" vaut 25. Ici il est évident que oui puisque nous la déclarons juste au dessus, mais imaginez que "mavariable" soit inconnue et que ça soit le spectateur qui la modifie.

 ATTENTION 
Il y a bien deux signes égal !

Si supérieur à:

```
int mavariable = 25;

  if (mavariable > 25)
  {
    //Code si la condition est vraie.
  }
```

Ce code regarde si "mavariable" est supérieure à 25. Si oui il éxécute ce qu'il y a entre accolades, si non il ne fait rien.

Si inférieur à:

Je pense que vous pouvez le deviner tout seul:

```
int mavariable = 25;

  if (mavariable < 25)
  {
    //Code si la condition est vraie.
  }
```

On vérifie si "mavariable" est plus petit que 25.

Je pense que vous avez compris le principe. Maintenant je vous donne les signes et je passe les commentaires.

Si inférieur ou égal à:

```
int mavariable = 25;

if (mavariable <= 25)
{
  //Code si la condition est vraie.
}
```

Si supérieur ou égal à:

```
int mavariable = 25;

if (mavariable >= 25)
{
  //Code si la condition est vraie.
}
```

Si différent de:

```
int mavariable = 25;

if (mavariable != 25)
{
  //Code si la condition est vraie.
}
```

Vous voyez, c'est super simple! Un petit exemple que vous pouvez tester:

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

int main()
{
   printf("Veuillez rentrer votre age: ");

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

```
    if(age >= 18)
    {
    printf("Ah! Vous avez %d ans et vous etes donc majeur!", age);
    }

    if(age < 18)
    {
    printf("Ah! Vous avez %d ans et vous etes donc mineur!", age);
    }

    return 0;
}
```

Ce qui donne:

```
Veuillez rentrer votre age: 45

Ah! Vous avez 45 ans et vous etes donc majeur!
```

Il y a cependant, quelques choses encore à voir sur les conditions, comme:

-Le else.
-Le else if.
-Le switch.
-Plusieurs conditions dans une.
-Les booléens.

Allez comme je sens que vous trépignez d'impatience à l'annonce de ce menu je vais tout vous expliquer!

Else:

Else veut dire "sinon" en anglais. Il est très pratique car permet de rendre son code beaucoup plus lisible et beaucoup moins complexe. Il suffit de placer, après avoir fait un if, un else et ,entre accolades, le code à exécuter si la première condition n'est pas remplie.

Reprenons le même exemple que tout à l'heure mais en rajoutant le else:

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

int main()
{
    printf("Veuillez rentrer votre age: ");

    int age = 0;
```

```
    scanf("%d", &age);

    if(age >= 18)
    {
    printf("Ah! Vous avez %d ans et vous etes donc majeur!", age);
    }

    else
    {
    printf("Ah! Vous avez %d ans et vous etes donc mineur!", age);
    }

    return 0;
}
```

C'est quand même beaucoup plus lisible! ☺

Esle if:

On peut le traduire littéralement par "sinon si".

On pourrait le mettre comme ceci dans un code:

Si l'âge est égal à 24 ans alors fait ça.

Sinon si l'âge est égal à 25 ans fait ceci.

Bref c'est un peut un if amélioré.

Le switch:

Bon pour l'instant c'est correct, nous n'avons pas marqué beacoup de conditions. Mais imaginez si vous deviez faire énormément de conditions d'affilé pour faire à chaque fois une action différente. Cela donnerait quelque chose comme:

```
if(machin == 2)
{
}
if(machin == 3)
{
}
if(machin == 4)
{
}
if(machin == 5)
{
```

```
}
if(machin == 6)
{
}
.....
```

Je vous l'accorde ce code est complétement stupide mais c'est juste pour cerner le problème.

Le switch permet de faire une zoli' présentation de toutes ces conditions là. Voyons ce que cela donnerait avec le switch. Je vous montre et je vous explique tout après 

```
switch(machin)
{
   case 2:
   break;

   case 3:
   break;

   case 4:
   break;

   case 5:
   break;

   case 6:
   break;
}
```

Il suffit donc, pour faire un switch, d'écrire "switch" puis entre parenthèse le nom de la variable à tester et entre les accolades mettre un case X: (où X est ici un chiffre mais peut très bien être une lettre ou autre chose) puis finir par mettre un break.

Le break sert juste à indiquer que la condition s'arrête ici (un peu comme l'accolade de fin dans les conditions "classiques").

Si vous voulez faire une condition dans un switch qui dirait "si aucune condition n'est vraie alors fait ça" il suffit de mettre au lieu d'un case un default comme ceci:

```
switch(machin)
{
   case 2:
   break;

   case 3:
   break;
```

```
        case 4:
        break;

        case 5:
        break;

        case 6:
        break;

        default:
//On met ici le code à éxécuter si aucune des autres conditions n'est vraie.
        break;

    }
```

Voilà, vous savez tout du switch 

Deux conditions dans une:

Imaginons maintenant que vous souhaitez vérifier si la personne est majeur ET si oui si la variable "test" est égale à 10 (ici c'est évident, mais il peut arriver que l'on ne connaisse pas dès le départ la valeur d'une variable).

On ferait avec nos connaissances actuelles quelque chose ressemblant à ça:

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

int main()
{
    printf("Veuillez rentrer votre age: ");

    int age = 0;
    int test = 10;

    scanf("%d", &age);

    if(age >= 18)
    {
        if(test == 10)
        {
    printf("Inscription acceptee!");
        }
    }

    else
```

```
    {
    printf("Ah! Vous avez %d ans et vous etes donc mineur! Inscription refusee!", age);
    }

    return 0;
}
```

Mais il est lourd de devoir écrire une condition dans une autre. En plus il n'y a, ici, que deux conditions, mais imaginez plus....

Mais je vais maintenant vous montrer que l'on peut fusionner les conditions.

Voici exactement le même code mais amélioré avec des conditions multiples dans une:

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

int main()
{
    printf("Veuillez rentrer votre age: ");

    int age = 0;
    int test = 10;

    scanf("%d", &age);

    if(age >= 18 && test == 10)
    {
    printf("Inscription acceptee!");
    }

    else
    {
    printf("Ah! Vous avez %d ans et vous etes donc mineur! Inscription refusee!", age);
    }

    return 0;
}
```

Je vous l'accorde ce n'est pas très utile pour l'instant. Mais croyez moi, c'est super utile quant il y a plusieurs variables à tester en même temps. Vous verrez de quoi je veux parler à partir du moment où nous apprendrons les booléééens, cela deviendra beaucoup plus utile 

Passons aux explications:

Vous pouvez, dans une condition, faire ceci:

```
if(condition1 leSigne condition2 .....)
```

La "condition1" est une condition comme nous l'avons déjà vu, exemple: age == 18 (qui vérifie pour rappel que la variable age est bien égale à 18).

"Condition2" est exactemement la même chose.

Maintenant passons au signe entre les deux conditions (notez que vous pouvez mettre autant de conditions que vous le souhaitez dans une seule! Il suffit de mettre un des signes que nous allons voir entre chacune d'entre elles).

le signe && représente le "et".
le signe ! représente le "si ... faux".
le signe || représente le "ou".

Par exemple si vous souhaitez exécuter une instruction seulement si variable1 est égale à 10 et que variable2 est inférieure à 3 vous allez faire comme ceci:

```
if(variable1 == 10 && variable2 < 3)
{
    //Instruction.
}
```

Et c'est exactement le même fonctionnement pour le "ou". Le "si..faux" est un peu plus spécial, c'est d'ailleurs pour cela que nous allons voir ici son fonctionnement. Mais avant ça, nous allons nous intéresser aux variables booléennes (rassurez vous c'est très simple ).

Une variable booléenne est une variable qui ne peut prendre que deux et seulement deux valeurs différentes. On s'en sert énormément en programmation pour dire vrai ou faux mais elle peut très bien vouloir dire autre chose comme noir et blanc, ouvert fermé, etc..

Imaginons pour notre exemple que le nombre 0 vaut faux et que le nombre 1 vaut vrai.

Avec une condition comme ceci:

```
int variable = 1;

if(variable)
{
   //Intruction.
}
```

Vous allez certainement me dire que cela ne veut rien dire, mais pourtant...si!

Regardez: la condition dit: "si la variable est vrai".

Si vous vouliez dire si la variable n'est pas vrai alors il faudrait écrire:

```
int variable = 1;

if(!variable)
{
   //Intruction.
}
```

Allez c'est pas si compliqué que ça la programmation! 

Vous êtes maintenant des dieux avec les conditions! Entraînez-vous car ce chapitre est primordial! Il était certes long et conséquent mais rassurez vous la suite n'est pas plus dur! Juste quelques choses de nouvelles à chaque chapitre  . Nous allons maintenant voir dans le chapitre suivant les opérations mathématiques qui sont elles aussi très importantes.

Les opérations mathématiques

Nous allons, dans ce court chapitre, voir en détail les opérations mathématiques possibles en C.

Nous allons, pour cela, utiliser une nouvelle bibliothèque (qui sera donc à inclure au projet) qui se nomme math.h (oui c'est un nom très original ). Cette bibliothèque permet d'utiliser des fonctions mathématiques complexes très simplement sans avoir besoin de tout ré-écrire (je vous rappelle que c'est le principe d'une bibliothèque ).

Mais avant de continuer, il faut tout d'abord inclure cette bibliothèque (elle est fournie par défaut avec code::blocks comme stdlib et stdio). Pour cela, il suffit d'écrire cette ligne-ci:

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

Si vous ne connaissez pas les fonctions citées ci-dessous, pas de panique! Elles ne nous seront probablement pas utiles sauf si vous souhaitez réaliser une calculatrice scientifique par exemple 

Revenons aux fonctions possibles de cette librairie. Elle permet de réaliser les opérations:

Sinus
Cosinus
Tangente
Arc sinus

Arc cosinus
Arc tangente
Exponentielle
Logarithme
Logarithme de base 10
Puissance
Racine carré
Arrondi supérieur
Arrondi inférieur
Valeur absolue d'un nombre
Multiplication par 2, le tout à la puissance indiquée

Et encore bien d'autres  Nous allons ici détailler les principales en donnant à chaque fois un exemple d'utilisation.

Sinus: sin

Cette fonction renvoie un double. Elle prend en paramètre un double exprimé en radian.

Exemple:

```
double resultat;
        //La variable "resultat" vaudra le résultat de l'opération.
resultat = sin(valeur à calculer);
```

Cosinus: cos

Cette fonction renvoie un double. Elle prend en paramètre un double exprimé en radian.

Exemple:

```
double resultat;
        //La variable "resultat" vaudra le résultat de l'opération.
resultat = cos(valeur à calculer);
```

Tangente: tan

Cette fonction renvoie un double. Elle prend en paramètre un double exprimé en radian.

Exemple:

```
double resultat;
        //La variable "resultat" vaudra le résultat de l'opération.
resultat = tan(valeur à calculer);
```

Puissance: pow

Cette fonction renvoie un double. Elle prend comme premier paramètre le nombre à élever à la puissance indiquée par le second paramètre.

Exemple:

```
double resultat;
        //La variable "resultat" vaudra le résultat de l'opération.
resultat = pow(nombre, puissance);
```

Racine carré: sqrt

Cette fonction renvoie un double. Elle prend comme paramètre un double.

Exemple:

```
double resultat;
        //La variable "resultat" vaudra le résultat de l'opération.
resultat = sqrt(valeur à calculer);
```

Comme vous pouvez le voir, ces fonctions sont très simples d'utilisation  Je ne peux pas toutes vous les présenter car il y en a beaucoup trop et la plupart ne nous servirons probablement jamais 

Nous allons maintenant voir dans la partie suivante, les raccourcis mathématiques utilisés en programmation C. Ce chapitre sera très bref et assez simple 
C'est partit! 

Les raccourcis mathématiques

Cette partie sera donc très simple  Il s'agit juste de vous montrer les raccourcis mathématiques possibles en C.

Ces raccourcis sont très pratiques car ils rendent le code beaucoup plus lisible, sont plus clair et plus rapides à écrire.

Nous allons commencer par voir le système d'incrémentation/décrémentation puis nous verrons les raccourcis d'opérations mathématiques 

L'incrémentation

L'incrémentation, c'est le fait d'ajouter "1" à une variable. Oui je sais, pourquoi un mot aussi compliqué pour quelque chose d'aussi simple :P

Avant pour ajouter "1" à une variable, vous deviez faire:

```
int maVariable = 2;
maVariable = maVariable + 1;  //Après ces opérations, "maVariable" vaut 3.
```

Et bien je vais vous apprendre quelque chose: c'est du temps de perdu 

Il revient strictement au même d'écrire:

```
int maVariable = 2;
maVariable++;  //Après ces opérations, "maVariable" vaut 3.
```

Les deux signes "+" disent à la variable (ici "maVariable") de s'incrémenter, c'est à dire de s'ajouter "1".

Avouez que c'est quand même beaucoup plus pratique 

La décrémentation

La décrémentation est l'exacte inverse de l'incrémentation. C'est à dire que c'est le fait de soustraire "1" à une variable.

Avant pour soustraire "1" à une variable, vous deviez faire:

```
int maVariable = 2;
maVariable = maVariable - 1; //On incrémente la variable "maVariable".
//Après ces opérations, "maVariable" vaut 1.
```

Et bien maintenant c'est tout simplement:

```
int maVariable = 2;
```

```
        maVariable--;  //On décrémente la variable "maVariable".
//Après ces opérations, "maVariable" vaut 1.
```

Donc pour résumer:

-Les deux signes moins "--" après une variable servent à enlever "1" à cette dernière.
-Les deux signes plus "++" après une variable servent à ajouter "1" à cette dernière.

Voilà, vous êtes maintenant des pros de l'incrémentation et de la décrémentation :P

(Et en plus c'est la classe de dire incrémentation au lieu de "ajouter un" :P).

Les raccourcis d'opérations mathématiques

Nous allons voir 4 raccourcis mathématiques qui sont les mêmes, il y a juste un signe qui change à chaque fois 

Le principe

Imaginons que vous souhaitez faire l'opération suivante:

```
int valeur = 5;
valeur = valeur + 4; //La variable "valeur" vaut donc 9 (5 + 4).
```

Et bien lorsqu'une même variable est placée des deux cotés du signe égal dans la même configuration que ci-dessus, on peut écrire l'opération comme ceci:

```
int valeur = 5;
valeur += 4; //Revient strictement au même que d'écrire: valeur = valeur + 4;
```

Ce qu'il faut, c'est bien comprendre ce principe. Une fois compris, il suffit de changer le signe "+" devant le égal par:

L'étoile "*" pour faire une multiplication.
Le signe "-" pour faire une soustraction.
Le signe "/" pour faire une division.

Vous verrez que cette méthode d'écriture raccourcie est **très utilisée** dans le monde de la programmation. Car il faut bien comprendre une chose (et c'est un principe de base en programmation, quelque soit le langage):

Moins on met de code, mieux c'est.

Et c'est vrai. C'est bien plus simple/agréable à lire et permet de s'y retrouver plus facilement.

Voilà, un nouveau chapitre se termine, un chapitre assez simple☺

Le chapitre suivant est consacré aux boucles, une autre notion essentielle du langage C☺

Les boucles

Tout d'abord qu'est ce qu'une boucle, et à quoi cela sert-il?

Une boucle est un mot clé du langage C qui permet d'exécuter une ou plusieurs instructions en continue tant qu'une certaine condition est remplie.

Une boucle simple se créer de cette façon en C:

```
while (condition)
{
    //Instructions à exécuter.
}
```

Entre les accolades se trouvent les instructions à exécuter. La condition est une simple condition comme nous l'avons vu dans le chapitre sur les conditions.

Exemple, la boucle while:

```
int continuer =  0;


while(continuer <= 10) //Tant que continuer n'est pas supérieur ou égal à 10.
{
   printf("D");
   continuer++;
}
```

Cet exemple affichera donc 10 fois la lettre D en majuscule et ce à la suite comme le montre l'image suivante:

Comme vous pouvez le voir, la boucle while n'est pas compliquée. Regardons maintenant la boucle do...while.

```
int continuer =  0;

do
{
    printf("D");
    continuer++;

}while(continuer <= 10)
```

la boucle do...while est la même que la while à la différence près que cette boucle s'exécute une première fois sans vérifier de condition. Une fois la premiere exécution passée, elle vérifie la condition. Si celle-ci est vraie, alors on refait une nouvelle fois la boucle et ce, tant que la condition est vraie.

Si la condition est fausse, on sort de la boucle.

Vous pouvez vous apercevoir que c'est quand même super simple😄

Il ne nous reste plus maintenant qu'à voir la boucle for.

La boucle for est juste une boucle while mais en très condensée.

Reprenons le code qui affiche les 10 "D" à la suite:

```
int continuer =  0;

while(continuer <= 10)
{

    printf("D");
    continuer++;
}
```

Voici maintenant comment elle s'écrirait avec une boucle for:

```
int continuer;

for(continuer = 0; continuer <= 10; continuer++)
{
    printf("D");
}
```

Comme vous pouvez le voir, l'initialisation de la variable "continuer" se fait dans la boucle for.
En second paramètre, il y a la condition qui fait que l'on ne sort pas de la boucle (ici: tant que "continuer" est inférieur ou égal à 10) puis l'action à effectuer à chaque tour de boucle (ici incrémenter "continuer").

Enfin, tout ces paramètres sont séparés par un point virgule ";".

Cependant, je n'aime pas trop cette boucle, je préfère la boucle while. C'est pourquoi je n'utiliserais que très rarement la bouce for dans la suite de mon tutoriel, même si il m'a semblé important de quand même vous la montrer

Une dernière chose:

Faites très attention aux boucles infinies!

Voici quelques exemples de boucles qui ne se terminent jamais:

Exemple 1:

```
int continuer = 1;
while(continuer) //C'est un booléen je vous le rappel
{
    printf("a");
}
```

Exemple 2:

```
for(;;) //Oui, c'est bizarre mais c'est correct
{
    printf("a");
}
```

Et voilà! Vous savez tout des boucles
Avant d'aller plus loin, regardez si vous comprenez tout le cours jusqu'à cet endroit là car c'est fondamental!
Les chapitres suivants vont êtres légèrement plus corsés mais rien d'insurmontable, c'est pourquoi il est obligatoire d'avoir compris le début de ce tutoriel

Si vraiment une chose ne vous semble pas clair, je vous rappelle que vous pouvez poser votre question ici, vous aurez une réponse le plus rapidement possible

Nous allons maintenant voir un nouveau chapitre qui sera, lui, consacré aux fonctions

L'apparition des fonctions

Nous allons voir dans ce chapitre la notion de fonction ainsi que comment en créer et apprendre à s'en servir, bref que du bonheur!

Commençons tout de suite par voir ce qu'est une fonction et comment en créer une

La notion de fonction

Les fonctions permettent au programmeur d'organiser son code pour lui simplifier la vie et d'éviter de répéter des tâches inutiles. Si vous devez par exemple répéter beaucoup de fois dans votre code une certaine action prenant pas mal de ligne, la création d'une fonction serait utile car elle permettrait de ne pas tout réécrire à chaque fois mais juste une seule ligne.

Prenons un exemple:

```
/*Nous verrons juste après comment en créer, pour l'instant comprenez que ceci est une fonction qui exécute tout ce qui se trouve entre accolades*/

int FaireSauterJoueur()
{
    //Plein d'opérations supers-complexes pour faire sauter le joueur.
}


int main() //Le programme débute.
{
    FaireSauterJoueur(); //On a juste à appeler la fonction. Super pratique
    return 0;
```

```
}
```

Comme vous pouvez le voir, il suffit d'écrire juste une ligne à la place de plusieurs centaines  C'est donc beaucoup plus clair 

Voyons maintenant comment en créer une.

La création d'une fonction

En premier lieu, un point de vocabulaire: On ne dit pas vraiment "créer" une fonction mais plutôt "déclarer" (comme pour les variables).

Voici comment se présente la déclaration d'une fonction:

```
type nomDeLaFonction()
{
    //Instructions.
    return variable;
}
```

Voyons maintenant tout cela en détails:

-Le type est le type de retour de la fonction. Je m'explique: une fonction est souvent utilisée pour renvoyer quelque chose, par exemple la fonction pow (fonction qui calcule une puissance)
retourne un nombre (le résultat de l'opération) qui est de type double.
Et bien le type est tout simplement le type de retour de la fonction. Si vous souhaitez par exemple retourner un résultat dans un int, le type de la fonction sera de type int tout simplement et c'est exactement la même chose pour double, float, char, int, long etc.. 

-Le nomDeLaFonction est simplement le nom que vous souhaitez lui donner. Il est préférable de lui donner un nom plutôt explicite. Ce nom doit respecter les mêmes règles que pour les variables, à savoir je vous le rappelle: pas d'espaces, pas de caractères spéciaux, pas d'accents, etc... 

-Viennent ensuite les parenthèses juste après le nom de la fonction. Il s'agit de l'endroit où l'on peut placer des éventuels paramètres (nous allons revenir dessus juste après avoir terminé l'explication sur comment créer des fonctions ).

-Enfin, il y a les accolades et tout ce qui se trouve dedans. Ce sont les instructions à exécuter.

-Pour terminer, il y a le return variable qui renvoie la variable désignée du type choisi au départ 

Les paramètres

Les paramètres se situent entre les parenthèses, séparés chacun par une virgule.

Les paramètres sont des variables que l'on donne à une fonction pour qu'elle puisse les utiliser. Si on ne lui donne pas et que l'on essaye de quand même les utiliser, cela va générer une erreur.

Il faut indiquer à chaque fois le type demandé ainsi que le nom que l'on va attribuer à cette variable.

Exemple:

```
type nomDeLaFonction(int variable1, double variable2, float variable3) //Comme ceci.
{
    //Instructions.

    /*Nous pouvons donc, dans cette fonction, utiliser les variables "variable1", "variable2" et "variable3".*/

    return variable;
}
```

Vous savez à présent tout des fonctions "simples"  Voyons maintenant comment les utiliser:

-Tout d'abord, il faut déclarer la fonction **avant le main**, comme ceci:

```
/*On place la déclaration de la fonction juste avant le main (le main qui est d'ailleurs lui même une fonction).*/

void Fonction()
{
     //Plein d'opérations supers-complexes.
}

int main() //Le programme débute.
{
    Fonction(); //On a juste à appeler la fonction. Super pratique
    return 0;
}
```

Utiliser une fonction

Pour utiliser une fonction, il suffit d'écrire son nom suivi des parenthèses avec à l'intérieur les différents paramètres de la fonction. Si celle-ci n'a pas de paramètres, on met quand même les parenthèses mais en les laissant vides 

Pour nous habituer aux fonctions, nous allons créer ensemble une fonction qui fera l'addition des deux nombres passés en paramètre (c'est un exemple).

Tout d'abord, admettons que notre fonction renverra un nombre (logique) de type int et appelons notre fonction "CalulerSomme". Elle se présentera donc comme ceci:

```
int CalculerSomme()
{

}
```

Il ne manque plus qu'à indiquer les noms des paramètres comme ceci:

```
int CalculerSomme(int nombre1, int nombre2)
{

}
```

et à ajouter l'opération nécessaire à l'intérieur:

```
int CalculerSomme(int nombre1, int nombre2)
{
/*On écrit directement l'opération dans le return, bien que l'on aurait pu la stocker dans une variable et retourner cette dernière.*/
    return nombre1 + nombre2;
}
```

Ce qui nous donne le programme final:

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

int CalculerSomme(int nombre1, int nombre2)
{
```

```
/*On écrit directement l'opération dans le return, bien que l'on aurait pu la stocker dans une variable et retourner cette dernière.*/
     return nombre1 + nombre2;
}

int main()
{
    int resultatOperation;

//Le resultat est stocké dans la variable resultatOperation.
    resultatOperation = CalculerSomme(3, 45);

    printf("L'addition de 3 + 45 est egale a %d.", resultatOperation);  //On affiche le résultat.
}
```

Remarquez bien que comme une fonction retourne quelque chose, le simple fait d'écrire:

```
CalculerSomme(3, 45);
```

correspond à un nombre (ici le résultat de la somme de 3 par 45). Nous pouvons donc écrire directement la fonction dans le printf comme ceci:

```
    printf("L'addition de 3 + 45 est egale a %d.",CalculerSomme(3, 45));
```

Cela est tout à fait correct et permet d'utiliser une variable en moins

Voilà, j'espère que tout cela est à présent clair pour vous

Voyons maintenant les fonctions plus spéciales (il n'y en a que deux).

Les fonctions spéciales

Voici deux types de fonctions spéciales:

-La fonction **main**. Vous la connaissez depuis le début, mais elle mérite que l'on s'y intéresse un peu plus en détails. C'est une fonction comme les autres qui renvoie un int.
Elle prend comme paramètres argc et argv. Le premier paramètre contient le nombre de paramètres envoyé à cette fonction et argv contient tous les paramètres.

Vous allez me dire: Oui mais on ne lui donne pas de paramètre nous!
C'est exact, mais le système d'exploitation lui oui!
Je ne vais pas m'aventurer dans des explications plus approfondies, il faut pour cela que vous maitrisiez les tableaux, notion que nous n'avons pas encore vu

-La fonction de type **void**. C'est un type de fonction qui permet de ne rien renvoyer! Imaginez que vous écrivez une fonction ne servant, par exemple, qu'à écrire du texte à l'écran. Il ne sert à rien de renvoyer quelque chose. Vous pouvez donc écrire une fonction de type **void**.

Exemple:

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

/*Notez qu'une fonction de type void peut très bien avoir des arguments comme tout autre type de fonction ☺*/

void AfficherTexte()
{
    printf("Hello world!");
}

int main()
{
    AfficherTexte(); //On appele la fonction.
    return 0;
}
```

Voilà pour les deux types de fonctions "spéciales".

Les prototypes

Jusqu'ici nous avons vu qu'il fallait impérativement déclarée les foncitons **avant** le main sans quoi cela ne marchait pas. Et bien bonne nouvelle, on peut s'en passer!

Il suffira maintenant de déclarer son prototype avant la fonction main et de mettre la fonction n'importe où (même dans un autre fichier, chose que nous verrons très bientôt).

Reprenons l'exemple précédent:

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

void AfficherTexte() //La fonction est déclarée avant le main.
{
    printf("Hello world!");
}

int main()
```

```
{
    AfficherTexte(); //On appele la fonction.
    return 0;
}
```

Pour créer le prototype d'une fonction, c'est extrêmement simple:

1) Vous prenez la déclaration de la fonction.
2) Vous rajoutez un ";" à la fin de celle-ci.

Et c'est tout!
Essayons avec le code du dessus:

1) `void AfficherTexte()`
2) `void AfficherTexte();`

Ça va ce n'est pas trop compliqué 

Il suffit donc de mettre cette ligne avant le main puis la fonction n'importe où (ici nous allons, pour l'exemple, la mettre après le main):

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

void AfficherTexte();

int main()
{
    AfficherTexte(); //On appele la fonction.
    return 0;
}

void AfficherTexte()
{
    printf("Hello world!");
}
```

Et voilà! Vous savez maintenant absolument tout des fonctions! Le prochain chapitre traitera du préprocessing et qui sera très simple 

Retour sur le préprocessing

Bienvenue dans un nouveau chapitre qui sera une nouvelle fois assez simple 

Nous allons tout d'abord commencer par une notion très importante: qu'est-ce que le préprocessing?

Le préprocessing est l'étape où le compilateur remplace les directives de processeurs (d'où son nom).

Vous avez depuis le début croisé des directives de processeur (les includes) et bien en réalité, il y en a d'autres! Nous allons toutes les voir dans ce chapitre (rassurez-vous, il n'y en a qu'à peine 3 ☺).
Le préprocessing intervient donc au début de la compilation. Par exemple, les includes sont remplacés par les fichiers indiqués et le tout rassemblé dans un seul et même fichier. ☺

Avant de commencer, rappelez-vous d'une chose:

N'importe quelle directive de processeur commence obligatoirement par un #!

Voyons maintenant les différentes sortes de directives de processeur.☺

Include

Les includes permettent d'inclure (nan sans blagues😄) des fichiers en plus de celui de base. On s'en sert pour inclure des bibliothèques contenant des fonctions déjà codées (comme math.h, stdlib.h, etc..) ou pour inclure ses propres fichiers grâce à la programmation modulaire, notion expliquée dans le chapitre suivant☺

Pour rappel, les includes s'utilisent de cette façon:

```
#include <LeNomDuFichier>
```

!Attention!
Lorsque le fichier à inclure est situé dans le dossier de votre IDE (c'est-à-dire installé, comme les bibliothèques de base (stdlib, stdio, math, windows, etc..)) il faut encadrer le nom du fichier à inclure de chevrons <> comme vous avez eu l'habitude de le faire depuis le début☺

Cependant, lorsqu'il s'agit d'un fichier se trouvant dans le dossier de votre projet, il ne se met plus entre chevrons mais entre guillemets "" comme ceci:

```
#include "LeNomDuFichier"
```

Attention donc☺

Je pense qu'il n'est pas trop nécessaire de s'éterniser sur les includes étant donné que vous savez déjà vous en servir ;)

Voyons donc dès à présent une nouvelle directive de processeur, j'ai nommé ... define!

Define

Define est extrêmement pratique en programmation! Il permet de remplacer par un mot, une instruction, un nombre, un mot, bref ce que vous voulez!

Prenons un exemple:

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

#define Dit_Bonjour printf("Hello world!");

int main()
{
    Dit_Bonjour
    return 0;
}
```

Le nom du define doit respecter les mêmes règles que pour les variables et les fonctions!

Remarquez qu'il n'y a pas de point virgule à la fin de Dit_Bonjour car ce n'est pas une instruction mais juste une indication à l'IDE au moment de la compilation

Ce qu'il faut bien comprendre c'est que le Dit_Bonjour est remplacé par print("Hello world!");

Vous avez compris cela, vous avez tout compris

Vous pouvez aussi mettre plusieurs lignes de code dans les defines en procédent comme ceci:

```
#define blablabla\
        blablabla\
        blablabla
```

Il suffit de mettre un antislash à la fin de chaque ligne sauf à la dernière

Exemple:

```c
#define Dit_Bonjour printf("Hello");\
                    printf(" et bienvenue");\
                    printf(" sur ");\
                    printf("creanet.wifeo.com!");

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

int main()
{
        Dit_Bonjour
        return 0;
}
```

Ce qui, au final, donne bien:

```
Hello et bienvenue sur creanet.wifeo.com!
Process returned 0 (0x0)   execution time : 0.203 s
Press any key to continue.
```

Pensez aussi qu'une define peut très bien être aussi une valeur comme:

```
#define ValeurPI 3.14159265
```

Voilà, vous maîtrisez les defines! Il ne nous reste plus qu'une sorte de directive de processeur à voir, les conditions.

Les conditions existent aussi dans le monde des directives de processeur

Sauf qu'ici elles sont encore plus simples que simples Elles permettent d'effectuer ou non certaines actions au moment de la compilation du programme. Par exemple (c'est ce que nous allons voir ici) en fonction du système d'exploitation.

Une condition en "langage" préprocesseur se fait comme ceci:

```
#if condition
   //Code à compiler si la condition est vraie.

#elif conditionB
   //Sinon si cette condition est vrai, compiler ce code.

#endif //Fin de la condition
```

Avouez que c'est quand même assez simple La condition peut-être n'importe quoi, mais c'est souvent la même chose, à savoir: faire une compilation en fonction du système d'exploitation.

Par exemple, si vous souhaitez mettre dans votre code une instruction spécialement pour les ordinateurs Apple, il vous faut écrire cela:

```
#if __APPLE_
    //Instructions, inclusions, bref ce que vous voulez.
#endif
```

Voilà, j'espère une nouvelle fois avoir été clair N'hésitez pas à poser une question si vous n'avez pas compris qu'elque chose

Si toutefois vous avez compris, vous m'en voyez ravi et je vous invite à continuer votre apprentissage vers un nouveau chapitre de ce gros tutoriel pour découvrir comment "partitionner" vos fichiers sources, chose bien pratique pour organiser un projet

La programmation modulaire

La programmation modulaire est le simple fait de séparer en plusieurs fichiers votre code source. Cela est vivement recommandé car cela le rend beaucoup plus clair.
Vous allez voir qu'il n'y a vraiment rien de compliqué à cela.

Les extensions de fichiers source

Pour cela il n'y a aucune règle:

Vous choisissez l'extension de fichier et le nom de fichier que vous souhaitez!

Cependant, par convention, on nomme souvent les fichiers sources C en .c ou .h (h pour header).

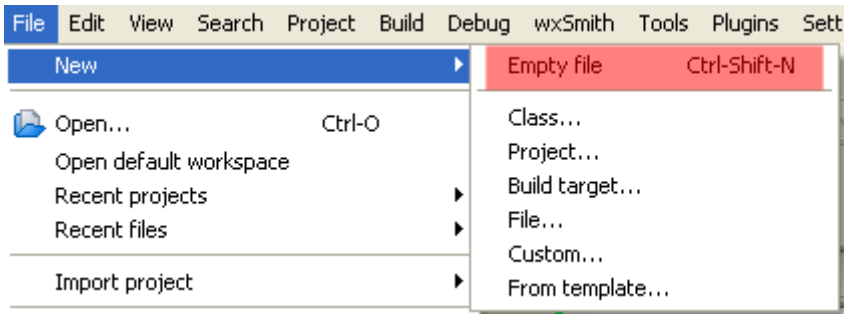
Personnellement je ne créer que des .h pour mes projets mais je sais que beaucoup créer des .c et des .h. En conclusion faites comme vous le souhaitez mais surtout organisez-vous de façon à ce que cela soit le plus clair pour vous

Créer un fichier

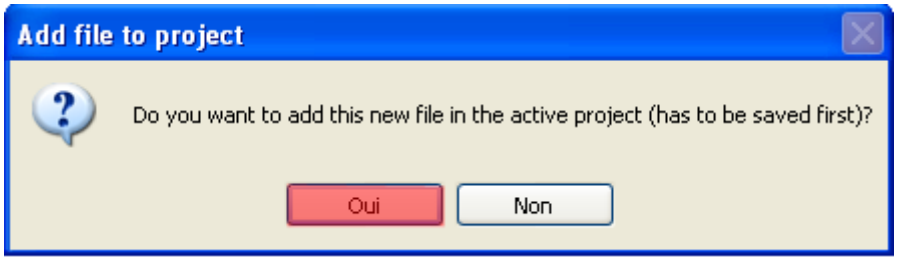
Je vais vous montrer ici comment créer un nouveau fichier pour votre projet Il vous suffit pour cela de suivre une à une les étapes

Tout d'abord ouvrez votre projet (c'est un bon début). Dans mon cas il s'agit d'un projet exécutant un simple "hello world"

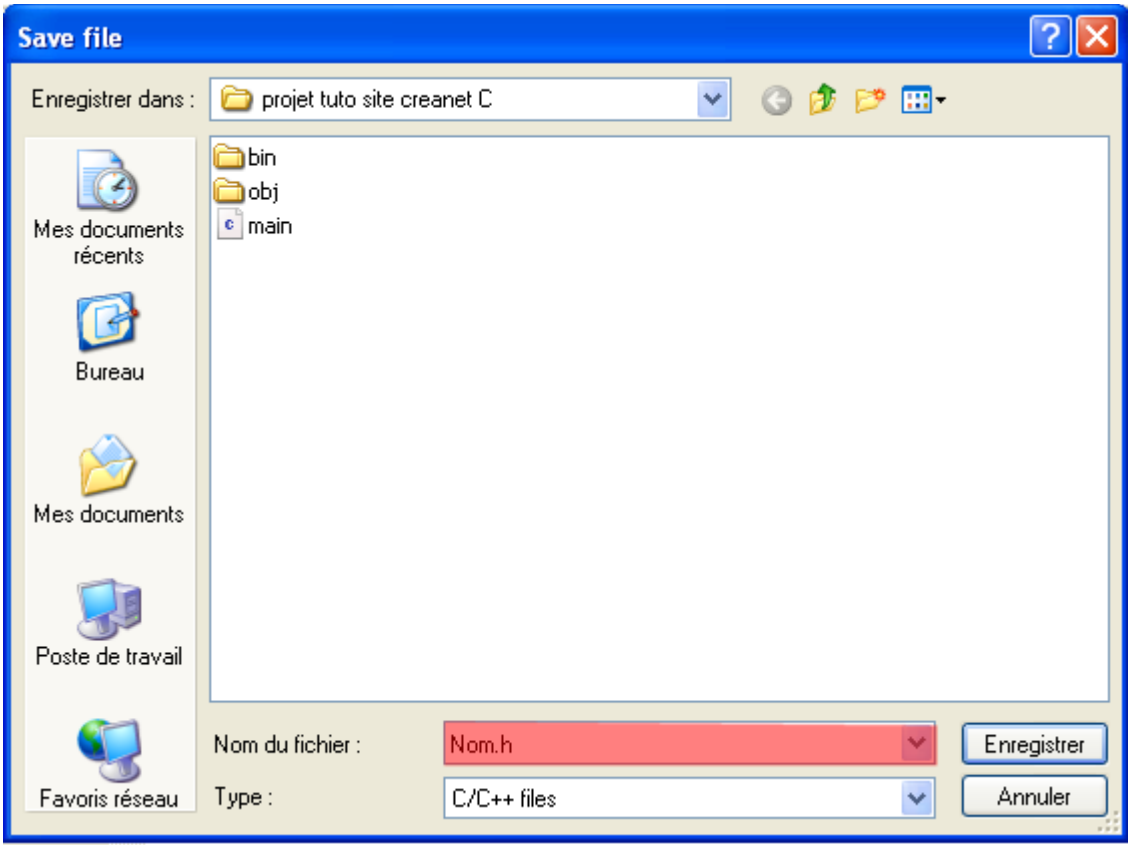
Puis cliquez sur File --> New -- > Empty file comme le montre l'image ci-dessous:



Dans la fenêtre qui s'affiche, cliquez sur "yes":



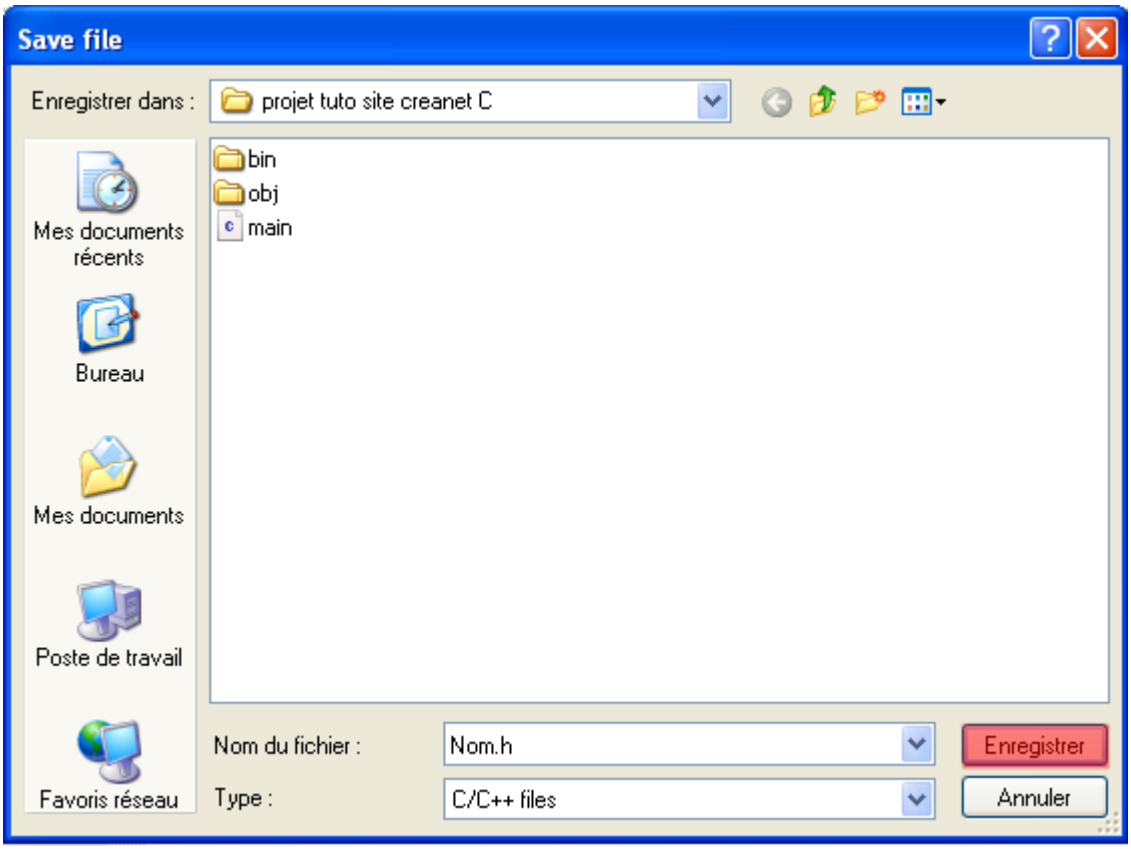
Ensuite cette fenêtre devrait s'afficher:



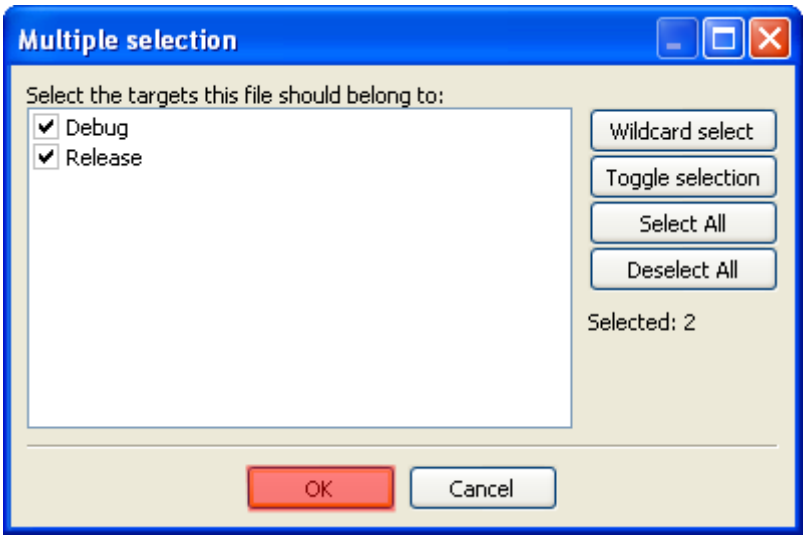
Écrivez le nom de votre nouveau fichier avec l'extension que vous souhaitez dans la case indiquée par l'image ci-dessus☺

Pour cet exemple, je vais l'appeler "essai.h".

Cliquez ensuite sur enregistrer:

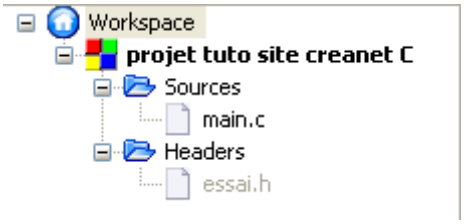


Puis a l'apparition de cette dernière fenêtre:



ne cliquez que sur "OK" **sans rien changer**.

Et voilà! Votre fichier est créé, félicitation Vous devriez le voir apparaître dans la partie de gauche de votre IDE comme ceci:

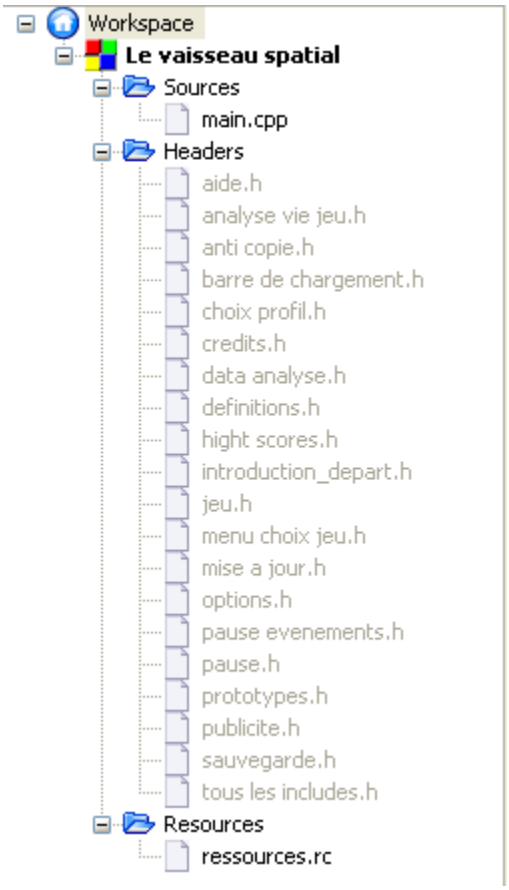


Maintenant qu'il est créé, il vous faut l'ajouter Pour cela, il suffit d'écrire dans le main:

```
#include "LeNomDuFichier"
```

Vous pouvez maintenant écrire du code dans votre nouveau fichier sous la forme de fonction par exemple et l'utiliser dans le main! Un "vrai" programme est souvent composé d'une multitude de fichiers. Pour ma part, je pousse un peu à l'extrême cette méthode puisque j'utilise un fichier par fonction Le principal étant de s'y retrouver, adopter votre propre façon de programmer

Voici une photo du projet de notre premier jeu nommé Space Fight (je ne vous montre que l'arborescence des fichiers et non les fichiers en eux-même un):



Comme vous le voyez, les noms de fichiers sont assez explicites de façon encor une fois à s'y retrouver

Une chose à laquelle il faut faire bien attention:

Une variable déclarée dans une fonction ne peut être utilisée dans une autre fonction sauf si elle est donnée en argument.

Cela veut dire que si vous créer une variable dans le main, vos autres fichiers **ne pourront pas accéder à cette variable. C'est ce que l'ont appelle la portée des variables.**

Vous pouvez utiliser, pour remédier à cela, des variables dites **globales**. Ceci est à éviter mais je vais quand même vous montrer comment elles se présentent

Il suffit de mettre ses variables avant le main comme ceci par exemple:

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

```c
int variableGlobale = 36; /*Cette variable peut être utilisée dans le fichier "monFichier.h". Notez que vous pouvez déclarer une variable globale elle même dans un fichier à part*/

int main()
{
   printf("Hello world!");
}
```

Il n'est donc pas recommandé d'utiliser des variables globales dans vos programmes, car cela rend le code moins lisible (on doit aller chercher partout une variable dans son code) et peut aussi causer des bugs si deux instructions utilisent en même temps la variable
Je vous recommande donc de ne pas utiliser de variables globales

Nous voilà arrivés à terme de ce chapitre sur la programmation moduaire. Je vous invite donc à cliquer sur la suite du cours pour faire un petit TP

TP: Administration du FBI

Avant d'aller plus loin dans ce cours sur le langage C, il est préférable de faire une petite pause sur un tout petit TP pour voir si vous avez tout compris

Le but est de demander à l'utilisateur le mot de passe (en chiffres) pour accéder au contrôle du FBI (c'est fictif un)

Une image étant plus parlante qu'un long discours, voivi ce qu'il faut faire:

Si l'utilisateur se trompe 5 fois de suite, il vous faut afficher ceci:

```
Veuillez rentrer le mot de passe (chiffres): 458542
Ce n'est pas le bon mot de passe !


Veuillez tapez le bon mot de passe :0254695
Ce n'est pas le bon mot de passe !


Veuillez tapez le bon mot de passe :4857521
Ce n'est pas le bon mot de passe !


Veuillez tapez le bon mot de passe :23546984
Ce n'est pas le bon mot de passe !


Veuillez tapez le bon mot de passe :15236548



        Mais, ...Vous n'etes pas l'administrateur!Pirate! Bandit! Escroc!
                                 Hors de ma vue!


Appuyez sur une touche pour continuer...
```

Si par contre il trouve le mot de passe alors là:

```
----------------------------ADMINISTRATION DU FBI-------------------------------


Veuillez rentrer le mot de passe (chiffres): 785210054

Bienvenue au FBI!
```

 J'expère que le but de ce TP est clair. Si c'est le cas.... 3 - 2 - 1 - Codez!

.....

Vous avez fini? Si ce n'est pas le cas essayez encore  Cela ne sert à rien de sauter des chapitres pour être perdu après, le mieux est d'y aller progressivement 

Si toute fois vous êtes bloqué (ou vous avez terminé ) je vous propose ma solution:

```
//Le mot de passe est 785210054.

#include <stdio.h>
```

```c
#include <stdlib.h>

int main()
{
   int motdepasse = 785210054; /*Il sufit de changer le 785210054 par un autre nombre pour
                    changer le mot de passe.*/

   int compteur = 1;
   int motdepasseentre;

   printf("----------------------------ADMINISTRATION DU FBI-----------------------------");

   printf("Veuillez rentrer le mot de passe (chiffres): ");
   scanf("%d", &motdepasseentre);

   while (motdepasseentre != motdepasse && compteur != 5)
   {
    printf("Ce n'est pas le bon mot de passe !");
    printf("Veuillez tapez le bon mot de passe :");
    scanf("%d", &motdepasseentre);
    compteur++;

   if(compteur == 5)
   {
      printf("      Mais, ...Vous n'etes pas l'administrateur!"
           "Pirate! Bandit! Escroc!                    Hors de ma vue!");
         system("pause");
         return 0;
   }

   }

   printf("Bienvenue au FBI!");

   return 0;
}
```

Je vous rappelle qu'il ne s'agit pas de la seule solution possible 

Voilà, ce mini-tp étant terminé, je vous invite à cliquer sur la suite qui traîtera des tableaux 

Les tableaux

Les tableaux sont très utiles. Ils permettent, de façon assez simple, de gérer plusieurs variables de même type. De plus, au chapitre suivant, vous saurez comment créer du texte avec le type

char et ce grâce aux tableaux! Il est donc important de comprendre ce chapitre

Prenons un exemple:

Imaginons que vous ayez besoin de 5 variables de type int. Vous devez faire ceci:

```
int variable0;
int variable1;
int variable2;
int variable3;
int variable4;
```

Là, c'est juste pour 5 variables, imaginez pour plus...
Et bien voici maintenant la même chose sous forme de tableau:

```
int tableau[5]; //Nous définissons un tableau de 5 int.
```

Remarquez que "tableau" peut être nommé comme vous le souhaitez

Pour accéder à une variable, il suffit de faire ceci:

```
tableau[indice];
```

Où indice est un nombre (où une variable, c'est pareil).

Par exemple, si nous voulons accéder à la deuxième variable du tableau, il vous faut écrire:

```
tableau[1]; //Non il n'y  pas d'erreur, lisez la suite
```


Attention! Un tableau commence toujours par l'indice numéro 0!

Ce qui veut dire que dans le cas présent, il n'y a pas d'indice numéro 5!

Donc la troisième variable du tableau est bien à l'indice numéro 2 (0, 1, 2).

Il faut impérativement comprendre cela car c'est une source d'erreurs très très fréquente!

Voyons maintenant comment affecter une valeur à une case du tableau:

```
tableau[indice] = valeur;
```

Et oui, c'est super simple Si vous voulez que la case du tableau à l'indice numéro 3 ait pour valeur 46 il suffit d'écrire ceci:

```
tableau[2] = 46; //Case numéro 3. (Elle vaut maintenant 46).
```

Comprenez bien que tableau[2] est considéré comme une variable à part entière de type int. Si bien qu'il est parfaitement correct d'écrire ceci pour afficher une case d'un tableau:

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

Prenons un code complet pour l'exemple:

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

int main()
{
   int monTableau[5]; //Tableau de 5 int (0, 1, 2, 3, 4).

   monTableau[2] = 46;

   printf("montableau[2] vaut %d", monTableau[2]);

   return 0;
}
```

Ce qui donne bien:

```
montableau[2] vaut 46
```

Les boucles

Ce qui est intéressant avec les boucles, c'est que l'on peut parcourir des tableaux avec une variable et non "manuellement"

Ce que je veux dire par là c'est: essayez d'afficher toutes les valeurs de ce tableau:

```
int tableau[100];

tableau[0] = 45;
tableau[1] = 84;
tableau[2] = 36;
tableau[3] = 78;
```

```
tableau[4] = 12;
tableau[5] = 29;
tableau[6] = 30;

.....
```

Vous allez faire quelque chose comme un printf à chaque fois:

```
printf("%d", tableau[0]);
printf("%d", tableau[1]);
printf("%d", tableau[2]);
printf("%d", tableau[3]);
printf("%d", tableau[4]);
```

Ba, c'est un peu long quoi Alors qu'il suffit d'une boucle pour tout afficher:

```
#define TAILLE_TABLEAU 5 /*Petit rappel sur les defines, qui peuvent servir pour initialiser la taille d'un tableau.*/

int i = 0;
int tableau[TAILLE_TABLEAU];

tableau[0] = 45;
tableau[1] = 84;
tableau[2] = 36;
tableau[3] = 78;
tableau[4] = 12;

while(i != TAILLE_TABLEAU)
{
    printf("%d ", tableau[i]);
    i++;
}
```

Ce qui nous donne:

```
45 84 36 78 12
```

C'est beaucoup plus rapide

À propos de la déclaration de la taille d'un tableau:

Vous ne pouvez pas déclarer la taille d'un tableau avec une variable comme ceci:

```
int variable = 5;
```

```
int tableau[variable];
```

Ceci est incorrect (cela s'appelle un tableau à taille dynamique, incorrect en langage C)

Certaines personnes vont peut-être demander pourquoi on peut mettre un define mais pas une variable pour déclarer un tableau. La réponse est qu'**un define n'est pas une variable.**
Il s'agit juste, je vous le rappelle, d'un "mot-clé" remplacé à la compilation par un nombre, des instructions, etc...

Grâce aux boucles, vous pouvez aussi initialiser les tableaux facilement comme ceci:

```
#define TAILLE_TABLEAU 5 /*Petit rappel sur les defines, qui peuvent servir pour initialiser la taille d'un tableau.*/

int i = 0;
int tableau[TAILLE_TABLEAU];


while(i != TAILLE_TABLEAU)
{
    tableau[i] = 0;
    i++;
}
```

Une autre manière d'initialiser les tableaux est sous cette forme:

```
int tableau[TAILLE_TABLEAU] = {0}; //Initialise toutes les cases du tableau à zéro.
```

Notez que le fait de faire ceci:

```
int tableau[TAILLE_TABLEAU] = {36, 45, 78};
```

Initialise la première case (n°0) à 36, la seconde (n°1) à 45, la troisième (n°2) à 78 et tout le reste à zéro. 

Attention! Le fait de marquer ceci:

```
int tableau[TAILLE_TABLEAU] = {36};
```

N'initialise pas toutes les cases à 36 mais seulement la première!
Les autres seront à zéro!

Voilà, maintenant que vous connaissez les tableaux, nous allons (enfin) voir les chaines de caractères. C'est-à-dire que nous allons, dans le chapitre suivant, apprendre à manipuler du texte!

Allez c'est parti!

Les chaines de caractères

À partir d'ici nous allons enfin pouvoir commencer à manipuler du texte

J'ai préféré ne pas vous montrer comment manipuler du texte tout de suite comme les nombres car c'est "légèrement" plus difficile et vous n'aviez pas les connaissances adaptées

Mais maintenant vous voilà près! Nous allons tout d'abord voir comment créer une chaine de caractère, comment cela fonctionne, en demander une à l'utilisateur, puis enfin découvrir quelques fonctions qui permettront de manipuler ces chaînes plus facilement
Here we go!

Créer une chaine de caractères

Nous avions vu, il y a déjà quelque temps, comment créer une variable pouvant contenir une lettre. Il fallait écrire ceci:

```
char variable = 'A';
```

Jusque là, rien de nouveau. Mais ce n'est pas un hasard si nous avons vu les tableaux juste avant ce chapitre, car en fait:

une chaine de caractère n'est qu'un tableau de char

Ce qui veut dire que comme char sert à stocker des lettres, une chaine de caractères sera un tableau de lettres.

Voici donc un exemple:

```
char variable[10]; //On créer un tableau de 10 char.

//On met dans chaque case du tableau une lettre.
    variable[0] = 'C';
    variable[1] = 'r';
    variable[2] = 'e';
```

```
variable[3] = 'a';
variable[4] = 'n';
variable[5] = 'e';
variable[6] = 't';
```

En réalité, ceci n'est qu'à moitié correcte, car pour que l'ordinateur sache quand la chaine s'arrête, il faut placer un antislash suivi d'un zero \0 (remarquez que ce signe est considéré comme un seul et unique caractère comme n'importe quel autre signe).

Il faut donc écrire pour que cela soit correct:

```
char variable[10]; //On créer un tableau de 10 char.

//On met dans chaque case du tableau une lettre.
variable[0] = 'C';
variable[1] = 'r';
variable[2] = 'e';
variable[3] = 'a';
variable[4] = 'n';
variable[5] = 'e';
variable[6] = 't';
variable[7] = '\0';
```

Pour afficher le texte, il suffit d'écrire:

```
printf("%s", variable);
```

Ce qui donne bien:

```
Creanet
```

Vous avez réussi à afficher du texte! Bon on savait déjà le faire OK :P
Maintenant écrivez ceci:

```
char variable[10] = "Creanet";
```

et afficher la variable à l'aide de printf ... Cela revient strictement au même! 

Donc lors de la déclaration d'une variable de type char, si vous souhaitez placer du texte dedans en utilisant les tableaux, vous pouvez marquer directement le texte comme je viens de vous le montrer plutôt qu'un fastidieux:

```
variable[0] = 'C';
variable[1] = 'r';
variable[2] = 'e';
variable[3] = 'a';
  .................
```

Mais je vous le rappelle:

Cela ne marche que pour l'initialisation de la variable!

De plus, vous pouvez écrire ceci à l'initialisation:

```
char variable[] = "Creanet";
```

La taille du tableau est calculée automatiquement et même le caractère de fin de chaine est pris en compte

Elle n'est pas belle la vie?

Voyons maintenant comment demander à l'utilisateur une chaine de caractère.

Saisie d'une chaine de caractères

Tout simplement comme ceci:

```
char votreChaine[10];
scanf("%s", votreChaine); /*Ne mettez pas le signe '&' devant un tableau de type char! Le prochain chapitre rendera cela plus clair*/
```

et voilà, ce que l'utilisateur à rentré est maintenant stocké dans le variable "votreChaine"


Attention

Si l'utilisateur rentre plus de texte que de cases possibles dans votre tableau le programme va planter (buffer overflow pour ceux qui connaissent). Il est donc impératif de bien prendre une taille de tableau en conséquence.

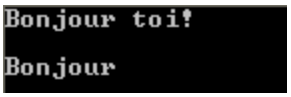
Les limites de scanf

Scanf est, certes, assez simple à utiliser mais pose néanmoins quelques problèmes et notamment de sécurité. Comme nous le verrons dans le chapitre sur la sécurité de vos applications. Nous allons tout de même l'utiliser, mais de façon contrôlée.

Par exemple, essayez ce code qui affiche ce que l'on a rentré:

```
char votreChaine[10];
scanf("%s", votreChaine); //On demande une chaine.
printf("%s", votreChaine); //On affiche cette chaine.
```

Essayez de rentrer "Bonjour toi!" (sans les guillemets). Cela nous donne:



O.O

Oui il n'y a que le "Bonjour" d'affiché! Donc deux choses à retenir:

1) On fait attention à la taille du buffer!
2) On ne marque pas d'espace!

Nous verrons donc, comme je vous l'ai dit, dans le chapitre sur la sécurité des applications, pourquoi cela se déroule de cette façon et surtout comment l'éviter

Les différentes fonctions de gestion de chaines de caractères

Avant toute chose, il vous faut inclure une bibliothèque Celle-ci se nomme string.h. Il faut donc écrire ceci:

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

Voyons maintenant les fonctions de manipulation de chaines de caractères les plus utiles

La longueur d'une chaine

Il vous sera peut être utile, parfois, de connaître la taille d'une chaine de caractères en particulier. Pour cela, il vous faut utiliser la fonction:

```
strlen(votreChaine);
```

Cette fonction revoie un type size_t (c'est un type créé spécialement pour cela, mais faites comme si c'était un int)

Exemple d'utilisation:

```
char maVariable[] = "Bonjour!";
printf("maVariable contient %d caracteres", sizeof(maVariable));
```

Tout simplement

Comparer deux chaines

La fonction permettant de regarder si deux chaines sont identiques est:

strcmp(chaine1, chaine2);

Cette fonction renvoie 0 si les chaines sont identiques et autre chose sinon.
On peut donc l'utiliser comme ceci:

```
char chaine1[] = "Exemple";
char chaine2[] = "Exemple";

if(strcmp(chaine1, chaine2) == 0)
{
    printf("Ce sont les memes chaines!");
}

else
{
    printf("Ces chaines ne sont pas pareilles!");
}
```

Ecrire dans une chaine

Si vous souhaitez, par exemple, écrire dans une chaîne une phrase suivie d'une variable par exemple comme ceci:

A ok! Vous avez donc X ans! //Où X est l'âge.
Et bien c'est possible grâce à la fonction sprintf 

sprintf(chaineDestination, "Du texte", ...);

chaineDestination est la chaine de caractères qui va stocker la phrase formée au complet. Veillez donc bien à sa taille  Le deuxième paramètre de cette fonction est le texte à écrire dans la variable précédemment citée. Enfin, les trois petits points ne sont pas obligatoires, ce sont des paramètres optionnels. Comme pour printf, si vous mettez dans le texte le signe "%d" et, à la place des trois petits points, une variable de type int, celle ci s'affichera dans le texte. Voici un exemple pour mieux comprendre:

```
char variableDestination[100];
int age = 45;

/*On forme de toutes pièces la variable variableDestination*/
sprintf(variableDestination,"Vous avez %d ans.", age);
printf("%s", variableDestination); //On affiche cette variable.
```

Sprintf est donc très utile pour la création de chaines de caractères 

Voilà, je vous ai montré les quelques fonctions relatives aux chaines de caractères qui me semblaient utiles. Si, par la suite, nous avons besoin d'autre fonctions, je vous les indiquerais

et expliquerais

Mais rien ne vous empêche d'aller de vous même voir le fichier source string.h pour voir les différentes autres chaines de caractères (attention toutefois ce fichier ne contient que les prototypes de ces fonctions et non le code des fonctions à part entière)

Le prochain chapitre traitera des pointeurs et de la gestion des adresses
Ce ne sera pas le moment de dormir car il s'agit peut être d'un des chapitres les plus dur mais ne vous en faites pas, vous avez fais le pire si vous êtes arrivé jusqu'ici

Les pointeurs et la gestion des adresses

Nous y voici enfin Préparez-vous à comprendre pas mal de choses d'un coup, ce chapitre sera beaucoup théorique mais c'est l'un des plus important!

Les pointeurs sont **partout** en programmation. Absolument partout! Vous en avez même utilisé ,sans le savoir, dès le chapitre sur les flux d'entrées et de sortie

Le but de ce chapitre est donc de vous expliquer ce que sont les pointeurs, à quoi ils servent et surtout comment s'en servir

Les pointeurs, une histoire ... de mémoire

La mémoire vive:

Tout d'abord, redéfinnissons ensemble comment est présentée la mémoire vive et coment elle fonctionne.
Nous avons vu précedemment que la mémoire vive permettait de stocker des informations temporaires (les variables). Seulement, savez-vous comment l'odinateur sait où trouver chaque variable dans son immense mémoire vive? Non? Et bien nous allons le découvrir ici

Les adresses

Le fonctionnement de la mémoire vive est assez simple. Imaginez qu'une maison représente une variable, et que le monde entier représente la taille de mémoire vive. Vous avez vu le nombre de maison?! L'ordinateur ne pourrait jamais retrouver une variable (=maison) dans la mémoire (=le monde)! Mais heureusement, pour pallier ce problème on a inventé quelque chose de simple et d'efficace: les adresses.

C'est comme pour les maisons! Comment trouve-t-on une maison en particulier? Grâce à son adresse! Et bien comment retrouve-t-on où est stockée une variable dans la mémoire vive?

Grâce à son adresse😄 Regardez ce schéma pour mieux comprendre à quoi ressemble la mémoire vive de votre ordinateur:

Adresse	Variable
0	36
…	…
9	18
10	3,5
…	…
5 025 453 158	145

A gauche, ce sont les adresses (elles vont beaucoup plus loin que 5 025 453 158, il y en a un nombre immense☺). Et à chaque adresse correspond un endroit de libre où l'on peut y stocker une variable. Attention, j'ai bien dis y stocker **une seule variable**. Une variable n'étant qu'un chiffre (une lettre est aussi un chiffre et donc un mot, une suite de chiffre)

on ne peut stocker qu'un seul chiffre par emplacement mémoire.

Donc lorsque vous déclarez une variable, voici ce qui se passe en mémoire:

1) Le programme demande à l'ordinateur s'il peut prendre une partie de sa mémoire.

2) L'ordinateur accepte en lui indiquant l'adresse d'un emplacement mémoire qu'il pourra utiliser.

3) Le programme mémorise l'adresse.

Comprenez bien que les noms que vous donnez aux variables ne sont en réalité que des adresses. On a donné la possibilité de leur mettre des noms dans le code source juste afin de

s'y retrouver

Du coup, lorsque vous aurez, dans la suite du programme, besoin de cette variable, le programme ira tout simplement à l'adresse indiquée

Se servir des adresses

Nous allons maintenant voir comment afficher l'adresse d'une variable

Pour cela, il faut tout d'abord ... créer une variable oui! Allez on va prendre un int

```
int maVariable = 10;
```

Pour afficher la valeur de la variable, nous faisons bien ceci:

```
printf("%d", maVariable);
```

Et bien pour afficher l'adresse d'une variable, il faut faire comme ceci:

```
printf("%p", &maVariable);
```

Le symbole "%p" a été spécialement concu pour les adresses, donc préférez le aux autres classique "%d", "%lf", etc.. qui fonctionnerons quand même Voici ce que cela affiche chez moi:

```
0022FF4C
```

Cette adresse est bein un nombre, mais écrit en format hexadécimal.

Notez qu'il est tout à fait normal que vous n'ayez pas la même adresse d'affichée que moi et ne vous étonnez pas non plus si elle change une fois le programme relancé! La raison est simple: nous n'avons pas la même quantité de mémoire vive ni la même utilisation de celle ci

Le symbole "&" devant la variable doit vous rappeler quelque chose. En effet, nous l'avons déjà utilisé avec scanf sans trop vous expliquer pourquoi Il est maintenant temps de remédier à cela

Lorsque le signe "&" est présent devant une variable,

cela désigne l'adresse de cette variable et non sa valeur

Nous pouvons donc faire ceci pour mieux faire la différence:

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

int main()
{
   int maVariable = 10;

   printf("maVariable vaut %d et se situe a l'adresse %p de ma memoire vive.", maVariable, &maVariable);

   return 0;
}
```

Ce qui nous donne:

```
maVariable vaut 10 et se situe a l'adresse 0022FF4C de ma memoire vive.
```

Donc il suffit de se rappeler qu'il faut rajouter devant une variable le signe "&" pour donner son adresse.

Les pointeurs

Mais qu'est ce dont qu'un pointeur? Nous avons déjà vu de nombreux types de variables (int, char, double, float, etc...).

Et bien en fait, un pointeur n'est rien d'autre qu'une variable dans laquelle est stockée ... une adresse! Et grâce à cela, on va pouvoir faire plein de zoli' choses

Voici comment créer un pointeur: Il suffit de rajouter, collé au nom de la variable une étoile "*" comme ceci:

```c
int *monPointeur = NULL;
```

L'étoile signifie qu'il s'agit d'un pointeur. Le pointeur peut être de n'importe quel type

Enfin, on voit que le pointeur est initialisé à **NULL** (en majuscules!). NULL permet de s'assurer que le pointeur ne contient rien. Il est super-important d'initialiser une variable et d'autant plus un pointeur!

Etant donné qu'un pointeur est créé pour stocker l'adresse d'une variable voici comment lui affecter la valeur de maVariable. Tout simplement:

```c
int maVariable = 10;
```

```
int *monPointeur = &maVariable;
```

Explications:

On créer une variable valant 10. On créer ensuite un pointeur valant l'adresse de maVariable.

Du coup, si on affchiche la variable monPointeur (qui est donc un pointeur) comme une variable classique, il devrait afficher sa valeur et par conséquent ... l'adresse de maVariable!

Essayons:

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

int main()
{
   int maVariable = 10;

   int *monPointeur = NULL;
   monPointeur = &maVariable;

   printf("monPointeur vaut %p et l'adresse de maVariable est %p", monPointeur,
&maVariable);

   return 0;
}
```

Ce qui donne bien:

```
monPointeur vaut 0022FF48 et l'adresse de maVariable est 0022FF48
```

On vient d'apprendre pas mal de chose là! Mais savez-vous que l'on peut faire encore plus fort?!

En effet, il est possible de gérer une variable grâce à une autre! Et tout cela encore grâce aux pointeurs! Maintenant, ce que je vais vous expliquer est **la** chose à retenir des pointeurs. Tenez-vous bien:

On peut, comme nous venons de le voir, créer un pointeur contenant l'adresse d'une variable. Mais l'on peut aussi accéder, grâce à ce pointeur, à la variable se trouvant à cette adresse et par conséquent, la modifier!

On dit que le pointeur "monPointeur" pointe sur la variable "maVaiable".
(vocabulaire à retenir).

Pour accéder, à partir d'un pointeur, à la variable que ce dernier pointe, il suffit de mettre l'étoile devant le pointeur.

Voici un exemple:

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

int main()
{
   int maVariable = 10;

   int *monPointeur = NULL;
   monPointeur = &maVariable;

   printf("maVariable vaut %d et monPointeur pointe sur une adresse memoire ou est stocke la valeur %d.", maVariable, *monPointeur);

   return 0;
}
```

Ce qui fait apparaître à l'écran:

```
maVariable vaut 10 et monPointeur pointe sur une adresse memoire
ou est stocke la valeur 10.
```

C'est magique! Nous pouvons accéder à une variable juste avec son adresse!

Voici un petit mémo de ce qu'il faut impérativement retenir:

maVariable --> désigne la valeur contenue dans cette variable.
&maVariable --> désigne l'adresse de cette variable.

monPointeur --> désigne la valeur contenue dans ce pointeur (c'est donc une adresse).
*monPointeur --> désigne la valeur de la variable pointée par le pointeur.

Si vous maitrisez cela, vous avez tout compris des pointeurs! Sinon revoyez tout et faîtes des tests. J'ai personnelement mis beaucoup de temps à comprendre le fonctionnement des pointeurs en programmation

Passer un pointeur à une fonction

A partir de là, on va voir le vrai "plus" qu'apportent les pointeurs en programmation

Essayez tout d'abord ce code-ci:

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

void fonctionIncrementation(int maVariable);

int main()
{
   int maVariable = 39;

   printf("maVariable avant incrementation --> %d  ", maVariable);

   fonctionIncrementation(maVariable);

   printf("maVariable apres incrementation --> %d", maVariable);

   return 0;
}

void fonctionIncrementation(int maVariable)
{
   maVariable += 1;
}
```

Ce qui fait apparaître ceci:

```
maVariable avant incrementation --> 39
maVariable apres incrementation --> 39
```

Mais c'est bizarre! On donne "maVariable" à une fonction qui incrémente la variable et on trouve, à la fin, une nouvelle fois 39!? @_@

La raison en est très simple:

Lorsque vous envoyez une variable en argument à une fonction, celle-ci fait juste **une simple copie** de cette variable et travaille sur cette copie! En fin de compte, la variable passée en argument n'est jamais modifiée par la fonction ce qui fait qu'elle ne change aucunement de valeur.

Mais maintenant que vous connaissez tous les petits secrets des pointeurs, laissez-moi vous en montrer un dernier: modifier une variable avec une fonction!

Voici le code qui fonctionne:

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

```c
void fonctionIncrementation(int *maVariable); //Prototype de la fonction.

int main()
{
   int maVariable = 39; //On créer une variable qui vaut 39.

   printf("maVariable avant incrementation --> %d  ", maVariable); /*On affiche la variable et donc 39.*/

   fonctionIncrementation(&maVariable); //On donne en argument l'adresse de la variable.

   printf("maVariable apres incrementation --> %d", maVariable);

   return 0;
}

//On stocke l'adresse donnée dans un pointeur nommé monPointeur.
void fonctionIncrementation(int *monPointeur)
{
   *monPointeur += 1; //On ajoute 1 à la variable pointée par monPointeur.
}
```

J'espère que les commentaires rendant plus clair ce code 

Il suffit "juste" de donner l'adresse de la variable à la fonction puis de modifier cette variable directement avec un pointeur. La variable est donc cette fois bien modifiée et n'est plus une simple copie.

De ce fait, vous pouvez même modifier directement plusieurs variables grâce à une fonction sans utiliser de return! 

Voilà un nouveau chapitre qui se termine. Un chapitre long mais très important! n'hésitez pas à poser des questions et à relire ce chapitre ainsi qu'à faire des tests! 
Le prochain chapitre vous apprendra à créer vos propres types de variables! 

Les structures

Les structures sont des variables crées par ... vous même 

Les structures sont très utiles et permettent de faire un nombre incroyable de choses 

Prenons un exemple:

Vous programmez un jeu comme Mario sauf que dans votre jeu, il y a plusieurs personnages jouant en même temps: Personnage1, Personnage2, et Personnage3.
Vous devez donc, pour chaque personnage, créer un certain nombre de variables (pour la vie, la direction où il va, sa position sur l'écran, le nombre de pièce récupérés, etc...).

Seulement voilà: Imaginons qu'il vous faut 5 variables par personnages, il vous faudra créer 15 variables en tout! Et ce n'est pas tout! Imaginons encore qu'il vous faut, pour une raison X, ajouter un nouveau personnage!

Vous le voyez, cela ne serez pas évident et on s'emmelerait les pinceaux à savoir à quel personnage appartient telle variable etc...

Mais là où je veux en venir, c'est que, grâce aux strcutures, vous pouvez pallier ce problème très facilement! 

Voici comment se déclare une structure en langage C:

```
struct nomStructure
{
    //Variables.

}; //Oui, il y a un point virugule!
```

Vous devez impérativement mettre un point virgule après l'accolade fermente

Les variables peuvent être de type différent.

Regardons ce que cela donnerais avec mon précédent exemple de Mario:

```
struct Personnage
{
    int nombreDePieces;
    int viesRestantes;
};
```

Remarquez que vous pouvez rajouter autant de variables de n'importe quel type souhaité 

Vous venez de créer votre propre type de variable! Il vous faut maintenant savoir comment l'utiliser  Pour cela, il vous suffit de déclarer votre type de variable comme une variable classique en rajoutant le mot clé "struct" comme ceci:

```
struct Personnage
{
    int nombreDePieces;
    int viesRestantes;
};

//On déclare une variable se nommant "Personnage1" de type Personnage.
struct Personnage Personnage1;
```

Si vous ne souhaitez pas écrire le mot "struct" à chaque fois que vous créer une variable de votre nouveau type, vous pouvez utiliser "typedef" comme ceci:

```
typedef struct Personnage Personnage;

struct Personnage
{
    int nombreDePieces;
    int viesRestantes;
};

//On déclare une variable se nommant "Personnage1" de type Personnage.
Personnage Personnage1;
```

Le typedef remplace juste le "struct Personnage" par simplement "Personnage" dans tout votre code, ce qui est beaucoup plus pratique et permet de rendre le code encore plus lisible 

Utilisation d'une structure

Pour accéder aux différentes variables qui composent votre nouveau type, il vous suffit décrire le nom de votre structure suivit d'un point et enfin du nom de la variable à laquelle vous souhaitez accéder. Comme ceci:

```
typedef struct Personnage Personnage;
```

```
struct Personnage
{
    int nombreDePieces;
    int viesRestantes;
};

//On déclare une variable se nommant "Personnage1" de type Personnage.
Personnage Personnage1;

//La variable "viesRestantes" de la structure Personnage1 vaut maintenant 4.
Personnage1.viesRestantes = 4;
```

Passer une structure une fonction

Les structures étant des types de variables de la même manière qu'un int ou un char, vous pouvez créer avec vos structures des pointeurs

Rien que de vous l'avoir anoncé je sens que vous êtes heureux

Imaginons que vous souhaitez créer un pointeur de type Personnage (la définition de la structure est celle juste ci-dessus) il vous suffira d'écrire ceci:

```
Personnage *monPointeur = NULL;
```

Il faut bien comprendre qu'en créant une structure, vous créez un nouveau type de variable!

Ce petit rappel sur les pointeurs étant fait, passons maintenant à l'étape supérieure: comment passer une structure à une fonction?

Nous allons pour cela prendre un exemple qui mettra à 6 le nombre de vies restantes et à 10 le nombre de pièces. Je vous laisse regarder le code avec les commentaires, je vous explique tout cela juste après

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

//Nous définnissons ici un nouveau type de variable grâce à une structure.
typedef struct Personnage Personnage;

struct Personnage
{
    int nombreDePieces;
    int viesRestantes;
```

```
};

void fonction(Personnage *Personnage1); //Prototype de la fonction utilisée après.

int main()
{
   Personnage Personnage1; //On créer une variable de type Personnage.

   fonction(&Personnage1); //On fournit à la fonction l'adresse de la variable crée.

   printf("nombre de vie: %d nombre de pieces: %d", Personnage1.viesRestantes,
Personnage1.nombreDePieces); //On affiche les variables de la structure.

   return 0;
}


//Notre fonction.
void fonction(Personnage *Personnage1) /*On récupère l'adresse de la structure par un pointeur du
même type.*/
{
//Nous allons voir pourquoi cette écriture juste après :)
    Personnage1->nombreDePieces = 10;
    Personnage1->viesRestantes = 6;
}
```

Oui, j'ai abusé en commentaire 

Une petite précision tout d'abord:

Il est impératif de déclarer votre structure en globale (avant le main et les prototypes) car il faut indiquer au programme que vous créez un nouveau type de variable 

Revenons maintenant sur l'utilisation des structures dans une fonction. Dans cet exemple, je voulais modifier les variables. J'ai, pour cela, écrit ceci:

```
    Personnage1->nombreDePieces = 10;
    Personnage1->viesRestantes = 6;
```

Tout simplement car il s'agit d'un pointeur, et que pour faire référence **à la valeur pointée** (et non l'adresse) il faut rajouter une flèche "->" devant la variable souhaitée.

Ceci ne marche que pour les pointeurs

Nous pouvons donc en conclure que:

pour accéder à une structure déclarée dans la fonction actuelle on utilise le point "."
pour accéder à une structure donnée en paramètre, on utilise la flèche "->"

Vous voilà rodé pour les structures :P

Encore une fois, j'insiste sur le fait que si une explication ne vous semble pas claire, il vous suffit de poser une question ici, nous nous ferons un plaisir d'y répondre

Les fichiers

Nous allons, dans ce chapitre, apprendre à manipuler les fichiers du disque dur en C. Nous allons voir comment lire un fichier et ses informations, comment les modifier et comment créer des fichiers

Voyons à quoi cela peut bien servir:

Prenez le jeu Mario. Le but est de collecter le plus d'étoile possible. Pour cela il suffit d'une variable qui s'incrémentera à chaque fois que l'on gagne une étoile. Là c'est simple

Mais imaginez maintenant que le jeu dur tellement longtemps qu'il est impossible de le terminer en 5 minutes. Dans ce cas-là, le joueur aimerais peut être bien éteindre son PC ou fermer le jeu pour reprendre la partie plus tard.

Problème: **La mémoire vive sera effacée et toutes les étoiles gagnées reviendront au nombre de 0.**

Il y a de quoi faire rager le joueur Pour contourner cet inconvénient, il fallait trouver quelque chose qui reste dans la mémoire de l'ordinateur quoi qu'il arrive: les fichiers.

Nous allons donc voir comment ouvrir et fermer un fichier puis comment écrire dedans et lire son contenu

Ouverture et fermeture d'un fichier

Avant d'écrire ou de lire un fichier, encore faudrait-il savoir l'ouvrir Et c'est ce que nous allons apprendre ici

Voici le prototype de la fonction permettant d'ouvrir un fichier (Remarque: à partir de maintenant je vous montrerais les prototypes des nouvelles fonctions que l'on utilise, cela sera beaucoup plus simple ):

```
FILE* fopen(const char* nomFichier, const char* typeOuverture);
```

Nous voyons que le premier paramètre est le nom du fichier à ouvrir. Pour cela, il suffit de mettre un type char ou directement le nom comme ceci:

```
fopen("essai.txt", const char* typeOuverture);
```

Le second paramètre "typeOuverture" est aussi une chaîne de caractères dans laquelle est placé l'un des "mots" suivants:

a
a+
r
r+
w
w+

Chaque "mot" permet d'ouvrir le fichier dans un mode différent. Les voici, ici, décrit:

"a" --> Permet d'écrire à partir de la fin du fichier. *Le fichier sera créé s'il n'existe pas.*

"a+" --> Permet d'écrire et de lire à partir de la fin du fichier. *Le fichier sera créé s'il n'existe pas.*

"r" --> Permet de lire dans le fichier. *Le fichier doit avoir été créé.*

"r+" --> Permet de lire et d'écrire dans le fichier. *Le fichier doit avoir été créé.*

"w" --> Permet d'écrire dans le fichier. *Le fichier sera créé s'il n'existe pas.*

"w+" --> Permet de lire et d'écrire dans le fichier en supprimant d'abord tout son contenu. *Le fichier sera créé s'il n'existe pas.*

Si vous souhaitez créer un fichier pour écrire par la suite dedans, il vous suffit d'écrire ceci:

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

int main()
```

```
{
/*On créer un pointeur de type FILE (structure créée spécialement pour la gestion des fichiers)*/
    FILE *monFichier = NULL; //Bien penser à initialiser le pointeur à NULL.

/*On créer le fichier test.txt pour pouvoir écrire dedans par la suite.*/
    monFichier = fopen("test.txt", "a");

    return 0;
}
```

Nous avons créé un fichier dans le même répertoire que votre projet 

C'est ce que l'on appelle **le chemin relatif**.

Le chemin relatif désigne un fichier à partir de l'endroit où votre programme est ouvert.

Le chemin absolu est le chemin complet.

Pour naviguer de dossiers en dossiers en C, il suffit de mettre un slash "/" ou deux antislashs d'affilés dans la chaine de caractère correspondant au nom du fichier. Comme ceci:

```
fopen("unDossier/unDeuxieme/monFichier.txt", "a"); /*Chemin relatif, c'est-à-dire que l'on suit ce chemin depuis l'emplacement de lancement de l'éxcutable.*/


fopen("C:/Program Files/test.txt", "a"); /*Chemin absolu, c'est à dire que quel que soit l'emplacement de lancement de l'application, nous ouvrirons toujours le même fichier indiqué*/
```

Enfin, il faut, une fois toutes les modifications de faîtes, fermer le fichier. Nous allons pour cela utiliser la fonction fclose qui a pour seul paramètre le pointeur associé au fichier à fermer.

Exemple:

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

int main()
{
    FILE *monFichier = NULL;

    monFichier = fopen("test.txt", "a");

    fclose(monFichier); //On ferme le fichier préalablement ouvert.
```

```
    return 0;
}
```

Jusqu'ici, rien de bien dur normalement 

Court moment de culture générale^^:

Toutes les fonctions, les structures nécessaires à la gestion des fichiers se trouvent dans la librairie stdio.h. Il faut donc bien penser à inclure cette bibliothèque (c'est ce que l'on fait depuis le début de ce tutoriel )

Bien. Vous savez ouvrir et fermer n'importe quel fichier présent sur le disque dur. Vous savez même en créer un (avec le type d'ouverture "a").
Savez-vous aussi que l'on peut aussi supprimer un fichier? Il faut pour cela utiliser la fonction remove. Voici son prototype:

```
int remove("LefichierASupprimer");
```

Cette fonction renvoie 0 si elle se termine correctement et qu'elle a réussi à supprimer le fichier. Sinon elle renvoie -1.

Notez qu'il ne faut pas préalablement ouvrir le fichier à supprimer avec fopen!

Petit exemple:

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

int main()
{
    int maVariable = remove("test.txt"); //On supprime le fichier.

    if(maVariable == 0)
      printf("Le fichier test.txt a ete supprime!");

    else
       printf("Impossible de supprimer le fichier");

      return 0;
}
```

 Attention 

Une fois le fichier supprimé, celui-ci l'est vraiment

et n'est pas placé dans la corbeille!
Attention donc de ne pas supprimer n'importe quoi!

Vous pouvez aussi renommer un fichier. Pour cela on utilise la fonction rename. Voici son prototype:

```
rename("nomOriginal", "nomApresRenommage");
```

Je vous passe la nécessité d'un code complet, cette fonction s'utilise quasiment de la même façon que remove 

Ecrire dans un fichier:

La fonction permettant ceci est fprintf. Elle s'utilise exactement comme printf mais n'a pas le même prototype. Le voici:

```
fprintf(pointeurFichier, "MonTexte", ...);
```

Je m'explique:

pointeurFichier est un pointeur de type FILE et qui est associé au fichier dans lequel vous souhaitez écrire. Attention! Veillez bien à ce que votre type d'ouverture de fichier accepte l'écriture! (Par exemple "a" est correct).

Le second paramètre est le texte à écrire dans le fichier et le troisième est optionnel, il s'agit des éventuels variables que vous souhaiteriez écrire dans le fichier.

Voici un exemple pour mieux comprendre:

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

int main()
{
    FILE *monFichier = NULL;

    monFichier = fopen("test.txt", "a"); //On ouvre le fichier en mode écriture.

    fprintf(monFichier, "Hello world!");

    fclose(monFichier); //On ferme le fichier préalablement ouvert.

    return 0;
}
```

Ce code créera donc un fichier test.txt (si celui-ci n'existe pas déjà) et écrira dedans "Hello World!".

Pour écrire une variable dans un fichier, il faut faire exactement la même chose qu'avec printf 

Exemple:

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

int main()
{
    int maVariable = 36;
    FILE *monFichier = NULL;

    monFichier = fopen("test.txt", "a"); //On ouvre le fichier en mode écriture.

    fprintf(monFichier, "maVariable vaut %d", maVariable);

    fclose(monFichier); //On ferme le fichier préalablement ouvert.

    return 0;
}
```

Lire dans un fichier

Avant toutes choses, vérifiez bien que le fichier est ouvert en mode lecture et qu'il existe.

Pour vérifier si un fichier existe, il suffit de faire ceci:

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

int main()
{
    FILE *monFichier = NULL;

    monFichier = fopen("test.txt", "r"); //On ouvre le fichier en mode lecture.

    if(monFichier == NULL) //Si le fichier n'existe pas.
      printf("Impossible d'ouvrir ce fichier, celui-ci n'existe pas.");


    else //Sinon si le fichier existe, on fait ce que l'on veut et on le referme.
```

```
    {
      fclose(monFichier); //On ferme le fichier préalablement ouvert.
    }

    return 0;
}
```

Attention

Il vous faut impérativement fermer le fichier dans la condition que le fichier existe! Si vous fermez un fichier qui n'existe pas, il y aura un joli crash de votre application!

Pour maintenant lire dans un fichier, il vous suffit d'utiliser fscanf qui s'utilise de la même façon que scanf:

```
fscanf(pointeurFichier, "%d", variable);
```

Où pointeurFichier est le pointeur associé au fichier voulu.

Ensuite, il faut savoir comment le fichier est structuré. Par exemple, si le fichier contient ceci:

145

Et que vous souhaitez lire le chiffre, il vous suffit d'écrire ceci:

```
fscanf(pointeurFichier, "%d", variable);
```

Et le chiffre sera logiquement placé dans "variable".

De même, si vous avez, par exemple, deux chiffres comme ceci:

154 875

Il vous suffit d'écrire:

```
fscanf(pointeurFichier, "%d %d", variable, variable2);
```

Et de même pour les chaines de caractères 

Il y a plusieurs autres fonctions comme fgets, fgetc, etc.. mais nous les verrons en temps voulu 

J'espère une nouvelle fois m'être fait comprendre sur ce chapitre  Je le redis une nouvelle fois, il est **très** important de comprendre ce que vous faites et d'écrire du code tout seul sans

aide en apprenant de ses erreurs. Essayez de créer vous-même des petits programmes.

Le chapitre suivant sera le dernier cours sur le langage C avant l'apparition des programmes en fenêtres! C'est pourquoi il est plus que jamais important de suivre!

L'allocation dynamique

L'allocation dynamique permet de rendre un programme plus performant notamment en utilisant juste la mémoire nécessaire et non une variable "classique".

Grâce à l'allocation dynamique, il vous sera possible d'allouer vous-même de la mémoire vive sans passer par une variable.

Je pense que vous ne voyez pas trop l'intérêt mais rassurez-vous, vous allez le découvrir très bientôt

Avant toute chose, rappelez-vous de la fontion sizeof. Je vous avais dit qu'il permettait de connaître la taille d'une variable. Et bien grâce à lui, vous pouvez aussi connaître la taille d'un type de variable!

Exemple:

```
sizeof(long);
```

Ce qui vous affichera si vous utilisez printf comme ceci:

```
printf("Long: %d octets.", sizeof(long));
```

```
Long: 4 octets.
```

Ce qui veut dire que lorsque vous créez une variable de type long, 4 octets seront alloués dans la mémoire de l'ordinateur alors que vous n'avez peut-être besoin que de juste 1 octet.

Sur les ordinateurs actuels, cela ne pose plus du tout de problème tellement la quantité de mémoire vive est importante. Cependant, si vous faites de la robotique par exemple, le programme aura certainement moins de mémoire vive possible et il sera très important d'économiser au maximum cette mémoire.

Pour faire ceci, il faut impérativement maîtriser le chapitre sur les pointeurs.

Lorsque vous créez une variable, voici les opérations exécutées par le programme:

1) Le programme demande à l'ordinateur un espace de mémoire vive libre pour qu'il puisse y stocker une valeur.

2) L'ordinateur répond par l'affirmative (la seule erreur éventuellement possible serait de ne plus avoir de mémoire vive. Ce qui est, je ne vous le cache pas, extrêmement peu courant ) et fournit au programme un pointeur pointant sur la zone mémoire réservée.

3) Le programme utilise cette zone mémoire.

4) Enfin, lorsque le programme n'a plus besoin de cette zone mémoire il la rend à l'ordinateur.

Nous allons maintenant voir comment réaliser ceci en langage C 

Tout d'abord, il faut demander une zone mémoire de disponible à l'ordinateur. Pour cela, il suffit d'utiliser la fonction malloc.

Voici son prototype:

```
void *malloc(size_t size);
```

Elle prend comme seul paramètre le nombre d'octets à allouer. Cette fonction renvoie un pointeur sur la zone mémoire réservée ou renvoie NULL en cas d'erreur.

Exemple:

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

int main()
{
     int *pointeur = NULL;

    pointeur = malloc(10); //On alloue 10 octets.

     if(pointeur == NULL)
       printf("Impossible d'allouer de la memoire!");

    else
       printf("Memoire allouee avec succes!");

  return 0;
}
```

Ce code alloue à l'OS (Operating System = Système d'exploitation) 10 octets de mémoire vive.

Mais ce code n'est pas correct!

En effet, il faut **impérativement** libérer la mémoire allouer à la fin de son utilisation!

Ceci est possible grâce à la fonction free qui prend comme seul paramètre, le pointeur pointant vers la zone mémoire à libérer.

Exemple de code complet correct:

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

int main()
{
     int *pointeur = NULL;

    pointeur = malloc(10); //On alloue 10 octets.

     if(pointeur == NULL)
        printf("Impossible d'allouer de la memoire!");

    else
    {
        printf("Memoire allouee avec succes!");
        free(pointeur); //On libère la mémoire allouée.
    }

  return 0;
}
```

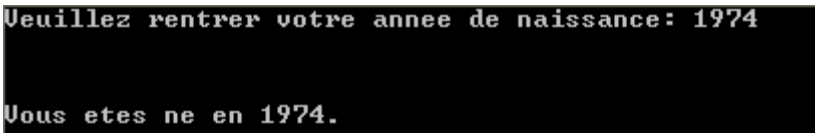
Remarquez que le free se situe bien dans la condition "else" qui indique que la zone mémoire a bien été allouée. Si vous essayez de faire un free avec un pointeur pointant sur NULL car l'opération de "malloc" n'a pas réussi, vous aurez une erreur. Attention donc.

Essayons de faire un petit programme d'exemple pour mieux se faire à l'allocation dynamique. Ce programme consistera à demander l'année de naissance à l'utlisateur et de lui redonner en utilisant les pointeurs ainsi que les allocations de mémoire (ici, l'allocation est limite inutile, mais il ne s'agit là que d'un exemple ):

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

int main()
{
    int *pointeur = NULL; //On creer un pointeur.

    printf("Veuillez rentrer votre annee de naissance: ");

    pointeur = malloc(sizeof(int)); //On alloue la "taille" d'un int.

    if(pointeur == NULL)
```

```
    printf("ERREUR DU PROGRAMME!");

    else
    {
    scanf("%d", pointeur);

    system("cls"); //On efface l'écran.

    printf("Vous etes ne en %d.", *pointeur);

    free(pointeur); //On désalloue la mémoire.
    }

    return 0;
}
```

Ce qui fait apparaître:



Pour l'instant vous pensez certainement que cela complexifie pour rien votre programme, mais je peux vous assurez que l'allocation dynamique est parfois bien utile

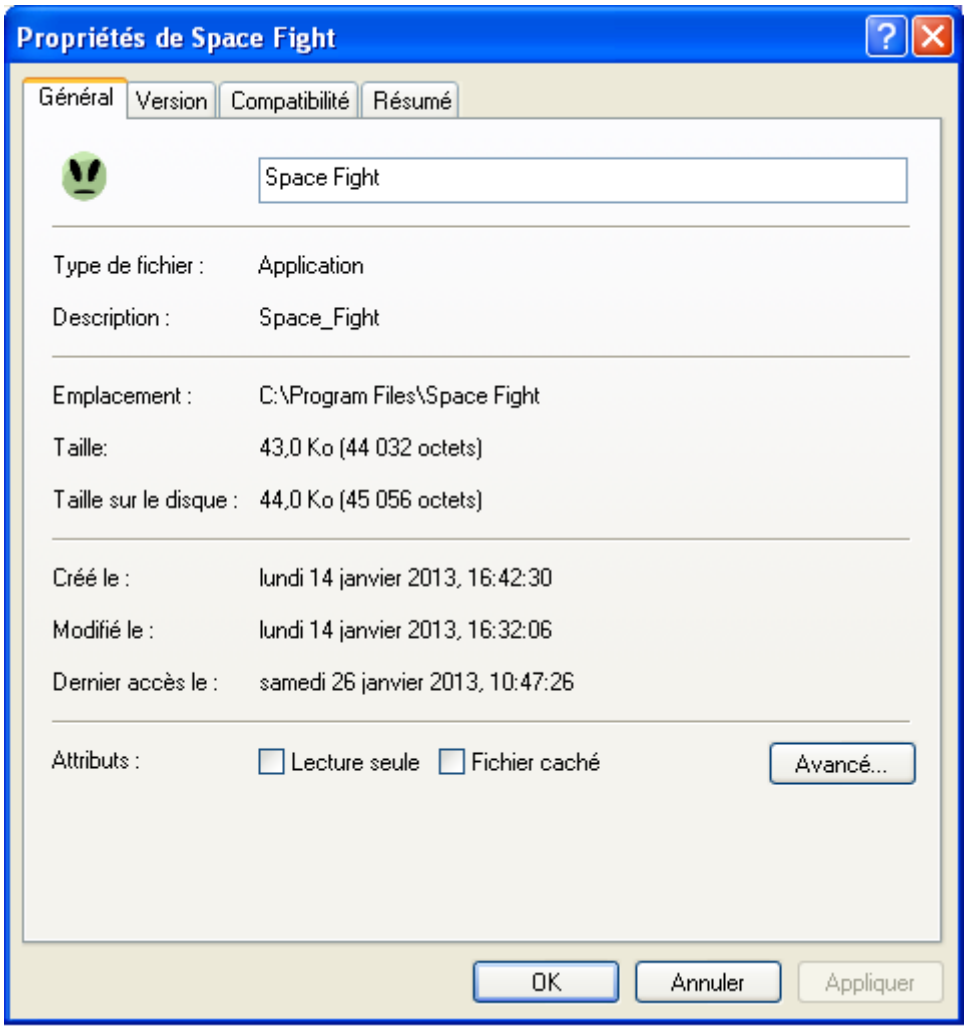
Ce cours chapitre de terminé, je vous propose de poursuivre ce cours avec, comme dernier chapitre (et oui le temps passe vite), une explication des propriétés des applications windows.

Les propriétés des exécutables

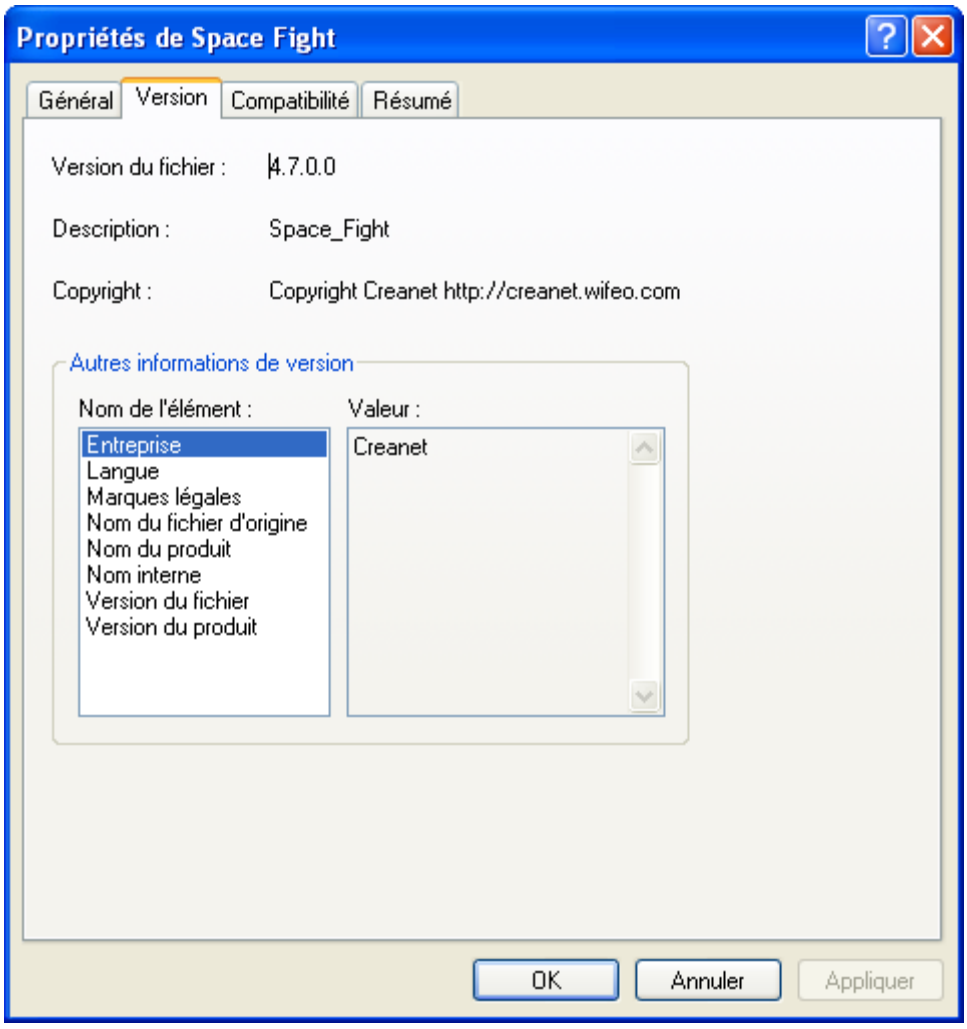
Avant toute chose je tiens à signaler que dans ce chapitre vous n'allez pas apprendre de nouvelles choses sur le langage C. Nous allons juste appliquer nos connaissances pour rajouter à nos applications des petits "plus". C'est parti!

Prenons un exemple: Space Fight.

Faisons un clique droit sur l'application et sélectionnons propriétés.
Vous arrivez à cette fenêtre-ci:



Allons maintenant visiter l'onglet "Version":



On peut voir, dans cet onglet, la version du fichier (ici 4.7.0.0), les copyrights (Creanet http://creanet.wifeo.com).

Enfin, dans "Autres informations de version", on peut trouver plein d'autres renseignements comme le nom d'origine, un teste de copyright, etc...

Et toutes ces descriptions/crédits sont écrits par le programmeur de l'application! Et ces informations sont inchangeables!

Je vais donc vous apprendre dans ce chapitre à créer vos propres propriétés d'application comme présenté ci-dessus.☺

Tout d'abord, il vous faut créer un fichier ressource. Pour cela, suivez la procédure classique de la création d'un fichier mais mettez l'extension .rc au nom du fichier. Par exemple dans mon cas, mon fichier se nommera ressources.rc☺ Enfin, rajouter cet include dans votre fichier main:

```
#include <winver.h>
```

Vous êtes prêt! Je vais à présent vous mettre le code et vous indiquer où remplir les champs. Je n'expliquerais pas ici le code car il s'agit d'API Windows et non de C

```
#define VOS_NT_WINDOWS32    0x00040004L
#define VFT_APP             0x00000001L

1 VERSIONINFO

FILEVERSION 4,7,0,0 //Version de votre programme
PRODUCTVERSION 0,1,0,0
FILETYPE VFT_APP
{
   BLOCK "StringFileInfo"
      {
         BLOCK "040904E4"
         {
         VALUE "CompanyName", "Nom de votre société"
         VALUE "FileVersion", "Version n° X"
         VALUE "FileDescription", "Description de l'exécutable"
         VALUE "InternalName", "Nom complet de l'application"
         VALUE "LegalCopyright", "Copyright"
         VALUE "LegalTrademarks", "Texte de crédits"
         VALUE "OriginalFilename", "Nom original"
         VALUE "ProductName", "Nom de l'application"
         VALUE "ProductVersion", "Version n° X"
         }
      }

   BLOCK "VarFileInfo"
   {
      VALUE "Translation", 0x0409, 1252
   }
}
```

Il vous suffit d'inscrire ce code-ci dans votre fichier ressource et de la compléter comme indiqué.

Compilez...votre programme à des propriétés! Mais ce n'est pas tout! Vous pouvez rajouter encore une chose à votre application pour qu'elle fasse plus "pro".

Il s'agit...de l'icône!

Mettre un icône sur une application

Vous voyez quand c'est simple? Et bien là, c'est encore plus simple que simple. Facile quoi

Il vous suffit de mettre tout en haut de votre fichier ressource cette ligne-ci:

```
1 ICON "icone.ico"
```

Le numéro indique le numéro de la ressource de votre programme. Ici c'est le numéro 1 car c'est la seule "vraie ressource". Le mot "ICON" indique que l'on veut stocker dans l'exécutable un icône, et cet icône est indiqué dans la chaîne de caractères suivante.

Le premier icône de stocké dans une application sera forcément l'icône que prendra l'application à la place de l'icône application par défaut qui est:

Par exemple, l'application Space Fight aura pour icône:

Attention! Il faut impérativement que votre image soit au format icône (.ico)!
Sans cela, la compilation de votre programme échouera!

Voilà, c'est la fin de ce cours petit chapitre hors sujet qui traitait des ressources des applications Je souhaitais vous montrer ceci car il m'a fallu beaucoup de temps pour trouver, enfin, comment faire

Je vous laisse maintenant, lire la conclusion de cette première partie intitulée "Le langage C".

Conclusion

Vous voici maintenant arrivé au terme de l'apprentissage du langage C à travers ce tutoriel

Attention, vous n'avez pas encore **tout** vu! J'ai volontairement passé certaines notions du langage C qui sont pour moi "secondaires". J'ai pour l'instant voulu vous inculquer les bases de la programmation

Grâce à ces bases, vous pouvez commencer à créer des petits programmes déjà assez sympa

J'ai choisi de vous apprendre le langage C, car c'est l'un des langages les plus utilisés!
De plus, celui-ci est multi-plateforme, pas trop compliqué, et permet pourtant de contrôler parfaitement n'importe quelle architecture d'ordinateur!

Il a même été utilisé pour créer une grande partie du système d'exploitation Linux!

Nous allons voir dans la deuxième partie de ce très gros tutoriel, comment créer des programmes en couleurs et fenêtrés!

Vous pourrez commencer à créer de "vrai" jeux 

Pour terminer cette conclusion, je ne ferais que citer Jacob Navia (créateur du compilateur lcc-win32):

"Si tu veux écrire un logiciel destiné à durer un certain temps, n'apprends pas "le language du jour" : apprends le C."

Tutoriel écrit en entier par Neyort pour http://creanet.wifeo.com

Tutoriel Partie 1 : Le langage C