1 Premiers Pas

1.1 Prologue

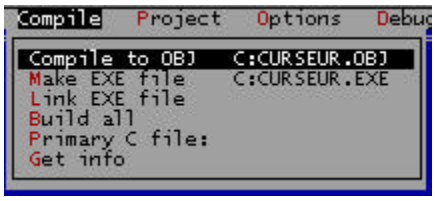
Ce cours utilise comme compilateur le Borland C 2.0. Celui-ci est disponible sur le Web.

1.2 Exemple de programme

```
main ()
{
        puts ("Bonjour");
}
```

Voici pour exemple un premier programme qui fonctionne malgré le fait qu'il ne soit pas normalisé. Celui-ci affiche le mot bonjour à l'écran.

Une fois le texte du programme frappé, il faut le compiler (Compile), c'est à dire en analyser la syntaxe.



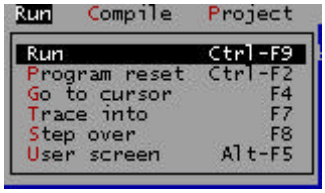
On remarquera plusieurs types de commandes dans le menu Compile :

1. *Compile* qui compile le programme c'est à dire en analyse la syntaxe et produit un pseudo-code non interprétable par l'ordinateur (.obj)
2. *Make* qui construit un exécutable .exe qui permet au programme d'être exécuter comme n'importe quel autre programme.
3. *Link* qui permet de lier plusieurs pseudo-codes entre eux et d'en créer un

4. *Build all* qui permet de retraduire tous les codes source d'un projet (ensemble de programmes liés les uns aux autres).

Code source : le code source représente le programme sous sa forme de langage C, c'est à dire ce que vous tapez dans l'éditeur de texte du compilateur.

Pour notre part, nous n'utiliserons que Compile, le reste se faisant durant la exe lors de l'exécution de la commande Run.



1.3 Exécution du programme (Run)

On remarque que :

- Pour voir ce qu'affiche le programme il est nécessaire d'utiliser la commande UserScreen dans le menu Windows.
- L'exécution du programme de nombreuses fois fait apparaître le mot Bonjour plusieurs fois.

1.4 Correction du programme

Nous allons normaliser le programme. En fait, à sa base, le langage C n'est qu'un ensemble de bibliothèques à partir desquelles le compilateur trouve les fonctions et les applications qui lui permettent de créer un programme exécutable. Exactement ce que vous faites lorsque vous recherchez dans une encyclopédie.

Certaines bibliothèques sont inclues dans des compilateurs ce qui permet à notre programme de s'exécuter. Normalement, **puts** a besoin de la bibliothèque **stdio.h**. Pour ajouter une bibliothèque, il suffit d'ajouter ***#include <nom de la bibliothèque>*** en début de programme.

Le second point à corriger est l'absence de valeur de retour. La valeur de retour permet à un programme ou à l'utilisateur de savoir si le programme que l'on exécute s'est correctement terminé. 0 signifie une terminaison sans erreur.

En lui rajoutant quelques fonctionnalités on obtient donc :

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

int main ()
{
        clrscr ();      /* Efface l'écran */
        puts ("Bonjour");
        getch ();       /* Attendre */
        return (0);
}
```

Bibliothèques

Corps du programme

La valeur de retour n'est pas obligatoire, pour ne pas utiliser de valeur de retour on utilise **void main ()** à la place de int main (). **void** peut se traduire par "ne contenant rien".
Attention: dans ce cas, on utilise **return;** et non return (0).
Le programme devient donc :

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

void main ()
{
        clrscr ();      /* Efface l'écran */
        puts ("Bonjour");
        getch ();       /* Attendre */
        return;         /* Facultatif car c'est la dernière ligne (fin)*/
}
```

1.5 Petit mot sur ce qu'est une bibliothèque

A l'instar de l'étudiant qui recherche dans des livres, on peut dire que le ".h" représente l'index du livre et le ".c" le contenu du chapitre concerné, le ".o" ou ".

Exemple

Lorsque le compilateur C rencontre le mot clrscr, il regarde dans chacun des ".h" déclaré par l'instruction #include si ce mot y est défini. Il trouve celui-ci dans la conio.h et remplace donc ce mot par le code qui lui est associé au moment de la compilation. A l'inverse, s'il ne le trouve pas, celui-ci émettra une erreur de syntaxe.

1.6 Un exemple de fichier bibliothèque

Vous trouverez ci-dessous, un extrait de la bibliothèque stdio.h. On y retrouve notamment la définition de **puts** que l'on voit dans ce cours et la définition de **printf** que l'on verra dans le 2nd cours.

Extrait du fichier stdio.h

```
/*      stdio.h

        Definitions for stream input/output.

        Copyright (c) Borland International 1987,1988
        All Rights Reserved.
*/
#if     !defined(__STDIO_DEF_)
#define __STDIO_DEF_
int     _Cdecl printf   (const char *format, ...);
int     _Cdecl puts     (const char *s);
#endif
```

1.7 Les différentes fonctions

puts : permet d'afficher du texte.
clrscr : permet d'effacer l'écran.
getch : permet d'attendre la frappe d'une touche.
/* Commentaire*/ : permet de mettre un commentaire.

Notre programme efface l'écran puis affiche bonjour et attend que l'on appuie sur une touche afin que l'on puisse voir ce qu'il a écrit.

1.8 Squelette de programme

On peut définir le squelette d'un programme C de la façon suivante :

```
/* Déclaration des bibliothèques */

int main ()
  {
```

```
  /* Déclaration des variables */          cf. chapitre 2

  /* Corps du programme */
  getch () ;        /* Facultatif mais permet de voir ce qui s'est produit à l'écran */
                    /* En attendant l'appui d'une touche */
  return (0) ;      /* Aucune erreur renvoyée */
}
```

1.9 Les blocs

La partie de programme située entre deux accolades est appelée un bloc. Je conseille de prendre l'habitude de faire une tabulation après le retour à la ligne qui suit l'accolade. Puis retirer cette tabulation après l'accolade fermante du bloc. Ainsi, on obtient :

```
{
        Tabulation
        Tout le code est frappé à cette hauteur
}
```

Bloc de programme

Retrait de la tabulation
Tout le texte est maintenant frappé à cette hauteur.

Cette méthode permet de contrôler la fermeture de toutes les accolades et leurs correspondances.

1.10 Les commentaires

Commenter signifie qu'une personne ne connaissant pas le langage C doit pouvoir lire le programme et le comprendre.

Les commentaires sont indispensables dans tout bon programme.

Les commentaires peuvent être placés à n'importe quel endroit dans le programme. Ils commencent par **/*** et se termine par ***/**.

```
/* Commentaire */
```

1.11 Exercices d'application

Ecrire un programme qui écrit au revoir.

Ecrire un programme qui :

- Ecrit « Salut toi, appuie sur une touche s'il te plaît
- Attend l'appui d'une touche
- Efface l'écran
- Ecrit « Merci d'avoir appuyé sur une touche

Commentez le précédent programme.
Exemple :

```
puts ("Cours de programmation" ) ;
/* Ecrit 'Cours de programmation' à l'écran */
```

Ecrire un programme qui écrit
« Hamlet says To be or not to be, that is the question. »

Corrigés des exercices du chapitre 1

! ***Ecrire un programme qui écrit au revoir***

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

main()
{
   clrscr();              /* Efface l'écran */
   puts("Au revoir ");    /* Affiche Au revoir */
   getch ();
}
```

! ***Ecrire un programme qui :***

- Ecrit « Salut toi, appuie sur une touche s'il te plaît
- Attend l'appui d'une touche
- Efface l'écran
- Ecrit « Merci d'avoir appuyé sur une touche

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

int main ()
{
   clrscr ();  /* Efface l'écran */

   puts ("Salut toi, appuie sur une touche s'il te plaît");
   /* Affiche le message Salut toi, … s'il te plaît */

   getch ();   /* Attend la frappe d'une touche */

   puts ("Merci d'avoir appuyé sur une touche");
   /* Affiche le message Merci d'avoir appuyé sur une touche */

   return 0;   /* Revient */
}
```

! ***Commentez le précédent programme.***

! ***Ecrire un programme qui :***

Ecrit : « Hamlet says To be or not to be, that is the question. »

```
#include <stdio.h>
#include <conio.h>
int main()
{
   clrscr ();
   puts ("Hamlet says To be or not to be, that is the question.");
   getch ();
   return 0;
 }
```