

Cours de C

Eric Berthomier

`eric.berthomier@free.fr`

Laurent Signac

`signac@sic.sp2mi.univ-poitiers.fr`

Table des matières

1 Premiers pas	7
1.1 Prologue	7
1.2 Exemple de programme	7
1.3 Normalisation du programme	8
1.4 Petit mot sur ce qu'est une bibliothèque	9
1.5 Un exemple de fichier bibliothèque	9
1.6 Les différentes fonctions	10
1.7 Squelette de programme	10
1.8 Les blocs	11
1.9 Les commentaires	11
1.10 Exercices d'application	11
2 Les variables (1^e partie)	15
2.1 printf : fonction indispensable pour afficher le contenu d'une variable	15
2.2 Variable	15
2.3 Déclaration d'une variable	16
2.4 Application : exemple	16
2.5 Utilisation multiple du %	17
2.6 Exercices d'applications directes	17
2.7 Réutilisation d'une variable	17

2.8	Caractères spéciaux	17
2.9	Exercices à réaliser	18
3	Les variables (2^e partie)	24
3.1	Exercice de mise en bouche	24
3.2	Déclaration des variables	24
3.3	Saisie des variables	25
4	Les conditions	32
4.1	Exercice de mise en bouche	32
4.2	Les conditions : Si Alors Sinon	33
4.3	Opérateurs logiques	33
4.4	Opérateurs logiques purs	33
4.5	Vrai ou faux	34
4.6	Combinaison	34
4.7	Astuce	34
4.8	Les accolades	35
4.9	Exercices	35
5	Mise au point	39
5.1	Prologue	39
5.2	Exercice 1	39
5.3	Retour sur getch()	39
5.4	Boucle Faire ... Tant que (vrai)	40
5.5	Exercice 2	41
5.6	Exercice 3	41

6	Et les shadocks pompaient	45
6.1	Préambule	45
6.2	While	45
6.3	Et les Shadoks apprenaient que reprendre équivaut à apprendre	46
6.4	Fonction <code>toupper ()</code>	46
6.5	O tant que en emporte le Shadok	47
6.6	Et les Shadoks continuaient à pomper pour obtenir le résultat	47
6.7	Au clan des Shadoks, on trie, voyelles, chiffres premiers	48
7	Les boucles	54
7.1	Et les shadoks pédalèrent pendant 15 tours	54
7.2	Syntaxe	55
7.3	Notion de double boucle	56
7.4	Exercice 3 : Et les Shadoks fêtèrent Noël...	57
7.4.1	« Cône » du sapin	57
7.4.2	Affichage du tronc	57
7.4.3	Affichage des boules de Noël	57
7.5	Exercice 4 : Table Ascii	58
8	Pointeurs et Fonctions	62
8.1	Variables : pointeurs et valeurs	62
8.1.1	Les variables et la mémoire.	62
8.1.2	Pointeurs	63
8.1.3	Exercices d'application	64
8.2	Les fonctions	64
8.2.1	Définition générale	64
8.2.2	Void	65
8.2.3	Variables globales et locales	65

8.2.4	Utilisation et modification de données dans les fonctions	67
8.2.5	Piège!	68
8.2.6	Exercice 3 :	68
9	Tableaux & Chaînes de caractères	71
9.1	Tableaux	71
9.1.1	Définition	71
9.1.2	Déclaration	71
9.1.3	Utilisation	72
9.2	Chaînes de caractères	72
9.2.1	Déclaration d'une chaîne de caractères	72
9.2.2	Écriture dans une chaîne de caractères	73
9.2.3	Affichage d'une chaîne de caractères	73
9.2.4	Longueur d'un chaîne	73
9.2.5	Exercices	73
9.2.6	gets : saisie d'une chaîne de caractères	74
9.2.7	Passage d'une chaîne de caractères en paramètres	75
9.2.8	Bugs en chaînes	75
9.2.9	Exercices	75
9.3	Correction des exercices	77
9.3.1	Boucle for	77
9.3.2	Fonction de saisie (1ère partie)	78
9.3.3	Fonction de saisie (2ère partie)	79
10	Fichiers et Structures	80
10.1	Bases sur les fichiers	80
10.2	Création d'un autre fichier	83
10.3	Structures	83

10.3.1	Déclaration	84
10.3.2	Utilisation	85
10.3.3	Taille d'une structure	85
10.4	Fichiers et Structures	86
10.5	Fin et commencement	87
10.6	Correction de l'exercice	88
11	Curses	90
11.1	Concepts de base	90
11.2	Quelques fonctions d'affichage de Curses	91
11.2.1	Localisation du curseur	91
11.2.2	Affichage de caractères	91
11.2.3	Effacement de la fenêtre	91
11.2.4	Affichage d'une chaîne de caractères	92
11.2.5	Affichage formaté d'une chaîne de caractères	92
11.2.6	Boîtes	92
11.2.7	Les anciennes fonctions d'affichage	92
11.3	1 ^{re} utilisation	92
11.4	Quelques fonctions de saisie de Curses	94
11.4.1	Saisie d'un caractère	94
11.4.2	Saisie d'une chaîne de caractère	95
11.4.3	Saisie formatée	96
11.5	Affichage de caractères spéciaux	97
11.6	Les couleurs	98
11.6.1	Dénomination des couleurs :	98
11.6.2	Fonctions liées au couleurs :	98

12 Le Jeu des Allumettes	100
12.1 Enoncé de l'exercice	100
12.2 Ce qu'il faut faire	100
12.3 Aide	101
12.4 Corrections de l'exercice sans Curses	102
12.5 Corrections de l'exercice avec Curses	105
13 Le Jeu de la Vie	109
13.1 Historique	109
13.2 Règles du jeu	109
13.3 Damier torique	110
13.4 Implémentation du jeu	110
13.4.1 Rappels utiles sur ncurses	110
13.4.2 Algorithme du jeu de la vie	111
13.4.3 Structures de données utilisées	112
13.4.4 Difficultés	112
13.5 Possibilités d'amélioration	113
13.5.1 Contrôle interactif	113
13.5.2 Affichage des coordonnées	114
13.5.3 Chargement de formes prédéfinies	114
13.6 Exemples obtenus	114
13.7 Exemple de corrigé	116
A Code Ascii	119
B Bibliothèque Asciiart	122
B.1 Introduction	122
B.2 Utilisation de la bibliothèque	122
B.3 Fichier d'en-tête	123

Chapitre 1

Premiers pas

1.1 Prologue

Ce cours est destiné à vous faire connaître le C sous Linux.

1.2 Exemple de programme

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.

À l'aide de votre éditeur de texte favori, ouvrez un fichier nommé `programme1.c` (1^e programme du cours) puis tapez le texte suivant :

`programme1.c`

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

À ce moment, si vous entendez hurler, crier, pleurer, vomir ou tout autre ignominie de ce genre, c'est normal, un puriste est passé derrière votre dos. Mais nous sommes là pour apprendre et il faut bien commencer par quelque chose.

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

Ceci se fait avec le compilateur `gcc`. Dans un premier temps, nous allons continuer dans le sale et compiler salement :

En ligne de commande, tapez : `gcc -c programme1.c`

Si vous n'avez pas fait d'erreur, ceci va vous donner un fichier nommé `programme1.o`. Ce fichier est nommé le fichier objet. Le fichier que vous avez passé en paramètre au compilateur se nomme le fichier source (source de tous les bonheurs de Linux ...).

Afin de pouvoir obtenir un fichier exécutable, il nous faut transformer le fichier objet en un exécutable. Pour cela, exécutez la commande suivante :

```
gcc -o programme1 programme1.o
```

Il ne reste plus qu'à exécuter le programme :

```
./programme1
```

Et comme par magie, s'affichera alors `Bonjour` et attendra que vous appuyez sur la touche Entrée.

Code source : le code source représente le programme sous sa forme textuelle (en langage C).

1.3 Normalisation du programme

Bon, je l'ai dit précédemment, nous avons fait dans le sale. Pourquoi me direz-vous ? Eh bien rendons notre compilateur bavard en lançant la commande `gcc -c -Wall programme1.c`. Observez la bornée d'insulte.

```
prog_1_1.c:5: warning: return-type defaults to 'int'
prog_1_1.c: In function 'main':
prog_1_1.c:6: warning: implicit declaration of function 'puts'
prog_1_1.c:9: warning: control reaches end of non-void function
```

Peu compréhensible et c'est normal.

En fait, l'option de compilation `-Wall` permet de « déclencher la production de messages soulignant toute technique autorisée mais discutable », en deux mots *à éviter*.

Nous allons donc normaliser ce programme.

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 pour réaliser un exposé.

Certaines bibliothèque (les plus courantes) sont incluses dans des compilateurs ce qui permet à notre programme de se compiler. 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. En général 0 signifie une terminaison sans erreur.

En lui rajoutant quelques lignes on obtient donc :

```
programme2.c
```

```

#include <stdio.h>

int main ()
{
    puts ("Bonjour");
    getchar (); /* Permet d'attendre la frappe d'une touche */
    return (0);
}

```

La valeur de retour n'est pas obligatoire mais fortement conseillée. Pour ne pas utiliser de valeur de retour, on utilise `void main()` à la place de `int main()`. Le mot clé `void` peut se traduire par « ne contenant rien ».

Attention : Dans le cas de `void main()`, on utilise `return ;` et non `return (0) ;`.

Le programme devient donc :

programme3.c

```

#include <stdio.h>

void main ()
{
    puts ("Bonjour");
    getchar ();
    return;
}

```

La normalisation n'est pas finie, malgré tout. Pour être au plus pur, il faudrait écrire :

```
int main (int argc, char** argv)
```

Mais, pour être expliquée, cette formulation à elle seule nécessiterait tout le cours de programmation (les 15 cours ;-)). Donc nous en resterons là pour la normalisation.

1.4 Petit mot sur ce qu'est une bibliothèque

À 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 « .obj » n'étend que la forme pré compilée du « .c ».

Exemple : Lorsque le compilateur C rencontre le mot `puts`, il regarde dans chacun des « .h » déclaré par l'instruction `#include` si ce mot y est défini. Il trouve celui-ci dans la bibliothèque `stdio.h`. À l'inverse, s'il ne le trouve pas, celui-ci émettra une erreur de syntaxe.

1.5 Un exemple de fichier bibliothèque

Vous trouverez ci-dessous, un extrait de la bibliothèque `stdio.h`. On y retrouve notamment la déclaration de `puts` que l'on voit dans ce cours et la déclaration de `printf` que l'on verra dans le second cours. Vous trouverez ce fichier dans le répertoire `/usr/include`.

Extrait du fichier `stdio.h` :

```
/* Write formatted output to STREAM. */
extern int fprintf __P ((FILE *__restrict __stream,
    __const char *__restrict __format, ...));
/* Write formatted output to stdout. */
extern int printf __P ((__const char *__restrict __format, ...));
/* Write formatted output to S. */
extern int sprintf __P ((char *__restrict __s,
    __const char *__restrict __format, ...));

/* Write formatted output to S from argument list ARG. */
extern int vfprintf __P ((FILE *__restrict __s,
    __const char *__restrict __format,
    _G_va_list __arg));
/* Write formatted output to stdout from argument list ARG. */
extern int vprintf __P ((__const char *__restrict __format,
    _G_va_list __arg));
/* Write formatted output to S from argument list ARG. */
extern int vsprintf __P ((char *__restrict __s,
    __const char *__restrict __format,
    _G_va_list __arg));

/* Write a string, followed by a newline, to stdout. */
extern int puts __P ((__const char *__s));
```

1.6 Les différentes fonctions

`puts` : permet d'afficher du texte suivi d'un retour à la ligne.

`getchar` : permet d'attendre la frappe d'une touche suivie d'un retour chariot ou un retour chariot seul.

`/* Commentaire*/` : permet de mettre un commentaire. On trouvera aussi `//` qui permet de mettre le reste de la ligne en commentaire.

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

1.7 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 */
    getchar(); /* Facultatif mais permet d'attendre l'appui d'une touche */
    return (0); /* Aucune erreur renvoyée */
}

```

1.8 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
}

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

```

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

1.9 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 terminent par */.

```
/* Commentaire */
```

Vous trouverez aussi :

```
// Le reste de la ligne est un commentaire
```

1.10 Exercices d'application

1. Écrire un programme qui écrit au revoir.
2. Écrire un programme qui :
 - Écrit « Salut toi, appuie sur une touche s'il te plaît »

- Attend l'appui d'une touche
 - Écrit « Merci d'avoir appuyé sur une touche »
3. Commentez le précédent programme. Par exemple :
- ```
puts ("Cours de programmation");
/* Ecrit Cours de programmation à l'écran */
```
4. Écrire 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.

programme4.c

```
#include <stdio.h>

void main()
{
 puts("Au revoir "); /* Affiche Au revoir */
 getchar ();
}
```

Ecrire un programme qui :

1. Ecrit « Salut toi, appuie sur une touche s'il te plaît »
2. Attend l'appui d'une touche
3. Ecrit « Merci d'avoir appuyé sur une touche »

programme5.c

```
#include <stdio.h>

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

 getchar (); /* 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);
}
```

Commentez le précédent programme.  
Déjà fait !

Ecrire un programme qui :

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

programme6.c

```
#include <stdio.h>

int main()
{
 puts ("Hamlet says To be or not to be, that is the question.");
 getchar();
 return (0);
}
```

## Chapitre 2

# Les variables (1<sup>e</sup> partie)

### 2.1 printf : fonction indispensable pour afficher le contenu d'une variable

programme7.c

```
#include <stdio.h>
int main ()
{
 printf ("Coucou c'est moi\n");
 /* Affiche Coucou c'est moi à l'écran */
 getchar();
 /* Attendre l'appui d'une touche */
 return (0);
}
```

On pourrait dire que la fonction `printf` est la même que l'instruction `puts` vue précédemment mais il n'en est rien... Celle-ci est beaucoup, beaucoup, beaucoup plus puissante.

*Syntaxe :* La syntaxe de `printf` est très complexe et pourrait à elle seule faire l'objet d'un cours, nous en verrons donc des applications au fur et à mesure des besoins.

### 2.2 Variable

Comme son nom l'indique une variable est quelque chose qui varie. C'est vrai mais ce n'est pas suffisant. Une variable peut être considérée comme une boîte dans laquelle on met des données que l'on peut lire ou écrire.

La manière la plus facile de lire le contenu d'une variable est la fonction `printf` que l'on a aperçue précédemment. La manière la plus simple de donner une valeur à une variable est l'opérateur mathématique `=`. Écrire dans une variable ayant déjà une valeur revient à la modifier.



Une variable ne peut contenir qu'une seule chose à la fois. Si vous mettez une seconde donnée dans la variable, la précédente est effacée.

## 2.3 Déclaration d'une variable

La déclaration d'une variable se fait simplement en écrivant :

```
<son type> <son nom>;
```

*Exemples de type de variables :*

char : caractère

int : entier

## 2.4 Application : exemple

programme8.c

```
#include <stdio.h>
int main ()
{
 int i;
 /* i : variable de type entier */
 char car;
 /* car: variable de type caractère */
 i = 65;
 /* i vaut 65 */
 car = 'E';
 /* car vaut E */
 printf ("i vaut %d.\n", i);
 /* Affiche la valeur de i */
 printf ("car vaut %c.\n", car);
 /* Affiche la valeur de car */
 getchar();
 /* Attendre l'appui d'une touche */
 return (0);
}
```

*Explications :*

1. On met dans la variable `i` la valeur 65.
2. On met dans la variable `car` la valeur de E.  
Note : En informatique, tout n'est que nombre, je dis donc la valeur de E et non E car c'est le code Ascii de E qui est sauvegardé dans cette variable. Nous reviendrons là dessus un peu plus tard.
3. `printf ("i vaut %d.\n", i);` : `%d` signifie que l'on attend une valeur entière, le programme va donc remplacer le `%d` par la valeur de `i`.
4. `printf ("car vaut %c \n", car);` : `%c` signifie que l'on attend une valeur de type caractère, le programme va donc remplacer le `%c` par la valeur de `car`. `\n` permet de réaliser un retour chariot c'est à dire un retour à la ligne.

## 2.5 Utilisation multiple du %

Le code « %x » signifie que le compilateur C doit remplacer ce code par la valeur correspondante (qui lui est fourni dans la suite de l'instruction) en la transformant dans le type x. Cette transformation est appelé un cast.

*Exemple :*

```
int i;
i =65;
printf ("Le caractère %d est %c",i,i);
```

nous donnera l'affichage suivant :

Le caractère 65 est A.

Le %d est remplacé par la valeur numérique de i c'est à dire 65.

Le %c est remplacé par la valeur alphanumérique (ASCII) de i c'est à dire A (cf. Table Ascii en Annexe).

## 2.6 Exercices d'applications directes

1. En utilisant ce qui a été fait précédemment, faites afficher les valeurs 70, 82, 185 et 30.
2. En utilisant ce qui a été fait précédemment, faites afficher, les caractères c, o, u, C, O, U.

## 2.7 Réutilisation d'une variable

On peut réutiliser une variable autant de fois que l'on veut, la précédente valeur étant effacée :

- `i = 3 ; i = 5 ; i = 7 ;` donnera au final pour valeur de `i` la valeur 7.

- `car = 'E' ; car = 'G' ; car = 'h' ;` donnera au final pour valeur de `car` la valeur de 'h'.

## 2.8 Caractères spéciaux

Certains caractères utilisés par la fonction `printf` (« % » par exemple) ou même tout simplement pour déclarer une variable ( ' pour les caractères par exemple) obligent à utiliser le caractère de suffixe \ pour pouvoir être affichés.

*Exemple :* Pour afficher un % avec `printf` j'écrirai :

```
printf("La réduction était de 20 \%) ;
```

Pour déclarer un caractère avec la valeur ' (prononcée *quote* en informatique et non pas apostrophe (français)), on écrira :

```
char car ; car = '\\" ;
```

## 2.9 Exercices à réaliser

1. Faire un programme qui réalise l'affichage suivant en utilisant les variables.  
Aide : Sans retour chariot, on affiche à la suite.

```
Coucou
17
```

2. De la même façon, réaliser un programme qui réalise l'affichage suivant :

```
C
O
U
Cou
1
2
3
456
C'est rigolo
```

*Rappel :* Pour mettre une variable `c` égale à `'` on écrit `c='\"`

3. Ecrire un programme qui écrit :

```
« Hamlet says "To be or not to be, that is the question." »
avec les " bien sûr.
```

*Rappel :* Pour pouvoir afficher un caractère de syntaxe `C`, par exemple `"`, on utilise le caractère `\` comme préfixe à ce caractère. Pour obtenir un `"`, on utilise donc `\"`.

## Corrigés des exercices du chapitre 2

Faites afficher, en utilisant ce qui a été fait précédemment, les valeurs 70, 82, 185 et 30.

programme9.c

```
#include <stdio.h>

int main ()
{
 int i,a,b,c;
 i=70;
 a=82;
 b=185;
 c=30;

 printf ("i vaut %d.\n",i);
 printf ("a vaut %d.\n",a);
 printf ("b vaut %d.\n",b);
 printf ("c vaut %d.\n",c);
 getchar();

 return (0);
}
```

Faites afficher, en utilisant ce qui a été fait précédemment, les caractères c, o, u, C, O, U.

programme10.c

```
#include <stdio.h>

int main ()
{
 char a,b,c,d,e,f;
 a='c';
 b='o';
 c='u';
 d='C';
 e='O';
 f='U';

 printf ("a vaut %c.\n",a);
 printf ("b vaut %c.\n",b);
 printf ("c vaut %c.\n",c);
 printf ("d vaut %c.\n",d);
 printf ("e vaut %c.\n",e);
 printf ("f vaut %c.\n",f);
 printf ("a, b, c, d, e, f valent : %c, %c, %c, %c, %c, %c.\n",a,b,c,d,e,f);

 getchar();
 return (0);
}
```

Faire un programme qui réalise l'affichage :

Coucou

17

programmell.c

```
#include <stdio.h>

int main ()
{
 char car = 'C';
 int nbre = 17;

 printf ("%c", car);
 car = 'o';
 printf ("%c", car);
 car = 'u';
 printf ("%c", car);
 car = 'c';
 printf ("%c", car);
 car = 'o';
 printf ("%c", car);
 car = 'u';
 printf ("%c", car);

 printf ("\n%d\n", nbre);

 /* Attente */
 getchar();

 return (0);
}
```

Faire un programme qui réalise l'affichage :

```
C
O
U
Cou
1
2
3
456
C'est rigolo
```

programme12.c

```
#include <stdio.h>

int main ()
{
 char car = 'C';
 int nbre = 1;

 car = 'C';
 printf ("\n%c", car);
 car = 'O';
 printf ("\n%c", car);
 car = 'U';
 printf ("\n%c", car);
 car = 'C';
 printf ("\n%c", car);
 car = 'o';
 printf ("%c", car);
 car = 'u';
 printf ("%c", car);

 printf ("\n%d", nbre);
 nbre = 2;
 printf ("\n%d", nbre);
 nbre = 3;
 printf ("\n%d", nbre);
 nbre = 456;
 printf ("\n%d", nbre);
 car = 'C';
```

```
 printf ("\n%c", car);
 car = '\\';
 printf ("%c", car);
 car = 'e';
 printf ("%c", car);
 car = 's';
 printf ("%c", car);
 car = 't';
 printf ("%c", car);
 car = ' ';
 printf ("%c", car);
 car = 'r';
 printf ("%c", car);
 car = 'i';
 printf ("%c", car);
 car = 'g';
 printf ("%c", car);
 car = 'o';
 printf ("%c", car);
 car = 'l';
 printf ("%c", car);
 car = 'o';
 printf ("%c\n", car);
 /* Attente */
 getchar();

 return (0);
}
```

Faire un programme qui écrit :

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

programme13.c

```
#include <stdio.h>

int main ()
{
 printf ("Hamlet says : \"To be or no to be, that is the question.\\n\");

 /* Attente */
 getchar();
 return (0);
}
```



## Chapitre 3

# Les variables (2<sup>e</sup> partie)

Durant tout ce chapitre, nous aurons pour but simple, mais complet et complexe dans sa programmation, de faire une malheureuse calculatrice.

*Attention :* Afin de pouvoir utiliser la bibliothèque mathématique du C (`#include <math.h>`), il est nécessaire d'ajouter au moment de l'édition de liens **-lm** ce qui nous donne :

```
gcc -o monprog monprog.o -lm
```

### 3.1 Exercice de mise en bouche

Ecrire un programme qui :

- écrit « Calculatrice : » et saute 2 lignes ;
- écrit « Valeur de a : » et saute 1 ligne ;
- attend l'appui d'une touche ;
- écrit « Valeur de b : » et saute 1 ligne ;
- attend l'appui d'une touche ;
- écrit « Valeur de a + b : » ;

Pas à pas, nous allons maintenant réaliser notre petit programme de calculatrice.

### 3.2 Déclaration des variables

Complétez le programme en :

- déclarant 2 variables `a` et `b` de type `int` (entier) ;
- assignant à ces deux variables les valeurs 36 et 54 ;
- faisant afficher le résultat de la somme de `a + b` (et non pas en faisant afficher 90 !).

Pour faire afficher le résultat il est possible d'utiliser la fonction `printf` directement ou indirectement en utilisant une troisième variable. Pour plus de facilité, nous allons utiliser un affichage direct. Celui-ci

s'effectue de la façon suivante :

```
printf ("Valeur de a + b : %d", a+b) ;
```

Le %d est remplacé par la valeur de a+b.

### 3.3 Saisie des variables

Si une calculatrice se limitait à exécuter la somme de deux nombres fixes, le boulier serait encore de mise.

Pour saisir une variable, il est nécessaire d'utiliser la fonction `scanf`. La fonction `scanf` s'utilise de la façon suivante :

**Saisie de la valeur a :**

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

Comme pour `printf`, on reconnaît le %d pour la saisie d'un nombre entier. Le & devant le a signifie que l'on va écrire dans la variable a.

*Aïe...* En fait &a signifie « l'adresse mémoire de a ». La fonction `scanf` va donc écrire dans l'emplacement mémoire (la petite boîte contenant la variable) de a. Si l'on oublie le &, on écrira chez quelqu'un d'autre, à une autre adresse. Et là ça peut faire mal, très mal...Mais ceci est une autre histoire.

Nous allons maintenant saisir les variables a et b. Pour exemple, je mets ici le code pour la saisie de a, la saisie de b reste à faire par vos soins à titre d'exercice.

```
/* Saisie de la valeur de a */
printf ("Valeur de a :\n");
scanf ("%d", &a) ;
```

Il est conseillé d'initialiser les variables avant de les utiliser. Au début du programme, après leur déclaration, assignez à a et à b la valeur 0.

**Exemple :**

```
a = 0 ;
```

Si tout s'est bien passé, vous avez maintenant entre les mains une super calculatrice qui réalise des additions !

*N.B.* Appuyez sur la touche `Enter` (Entrée) après avoir tapé votre valeur.

Pour aller plus rapidement, il est possible d'initialiser une variable dans le même temps que sa déclaration. Pour cela, rien de plus simple, ajoutez à la fin de la déclaration = suivi de la valeur d'initialisation :

**Exemple :**

```
int i = 10 ;
```

**Evolution du logiciel : exercice** *Aïe aïe aïe*, dur dur d'être informaticien. J'ai envie de faire une super calculatrice et je me dis qu'en fait la simplicité qui a eu lieu tout à l'heure n'est pas forcément adaptée à l'évolution désirée.

Déclarer une troisième valeur de type `int` (penser à l'initialiser à 0) que l'on nommera bêtement `s` comme somme. Une fois les valeurs de a et b saisies, initialisez `s` avec la valeur de a + b. Comme en mathématiques élémentaires. Affichez la valeur de `s`. On devrait avoir les mêmes résultats qu'auparavant, bien sûr.

**Exercices d'application :** Réalisez 2 petits programmes qui font :

- la soustraction de 2 nombres ;
- la multiplication de 2 nombres.

Un nouveau type : les nombres à virgule ou flottants (`float`). Le type `float` permet de déclarer un nombre à virgule. Transformez les 3 précédents programmes en utilisant le type `float` au lieu du type `int` et `%f` pour saisir les valeurs de `a` et `b`, et pour afficher le résultat.

*Petite aide :* Ce petit morceau de programme saisit la valeur de `a` et la fait afficher :

```
float a;
printf ("Saisie de a :");
scanf ("%f",&a);
printf ("\na vaut : %f",a);
getchar ();
```

Pour un affichage plus agréable il est possible de *fixer le nombre de chiffres après la virgule* de la façon suivante : `%.[nombre de chiffres après la virgule]f`

**Exemple :** `printf ( "%.2f", a );`

*Ouah, les 4 opérations!!!* Créer un quatrième programme qui réalise la division de deux nombres. Tester avec une division par 0!!! La solution à ce petit problème sera vu dans le cours sur les conditions (`if`).

**Exercices à réaliser** Compléter les 4 programmes afin de réaliser les opérations suivantes :

```
s = a + b + c
s = a -b-c
s = a / b / c
s = a * b * c
```

où `a`,`b` et `c` sont des nombres à virgules (`float`).

La fonction `abs` permet d'obtenir la valeur absolue d'un nombre entier. Utiliser cette fonction pour calculer la valeur absolue de `(a-b)`.

**Exemple d'utilisation de `abs` :** `S = abs (a-b) ;`

La fonction `ceil` permet d'obtenir l'arrondi entier supérieur d'un nombre réel. Utiliser cette fonction pour calculer l'arrondi supérieur de `(a / b)`.

**Exemple d'utilisation de `ceil` :** `S = ceil (a/b) ;`

## Corrigés des exercices du chapitre 3

Soustraction de 2 nombres

programme14.c

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

int main ()
{
 int a,b; /* Déclaration des variables */
 int d; /* Différence entre les 2 nombres */

 a=0; /* Initialisation des variables */
 b=0;

 printf("Calculatrice :\n\n");
 printf("Valeur de a : ");
 scanf("%d",&a);
 printf("\n");
 printf("Valeur de b : ");
 scanf("%d",&b);

 d=a-b;
 printf("Valeur de a-b : %d\n",d); /* Affichage de la différence */

 getchar ();
 return (0);
}
```

## Multiplication de 2 nombres

programme15.c

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

int main ()
{
 int a,b; /* Déclaration des variables */
 int m; /* Résultat de la multiplication */
 a=0; /* Initialisation des variables */
 b=0;

 printf("Calculatrice :\n\n");
 printf("Valeur de a : ");
 scanf("%d",&a);
 printf("\n");
 printf("Valeur de b : ");
 scanf("%d",&b);

 m = a*b;
 printf("Valeur de a*b : %d\n", m);
 /* Affichage de la multiplication */

 getchar ();
 return (0);
}
```

Transformez les 3 précédents programmes avec le type `float` Aucune difficulté dans cet exercice, je ne mettrai ici la correction que de la multiplication, les autres opérations se réalisent sur le même schéma.

programme16.c

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

int main ()
{
 float a,b; /* Déclaration des variables */
 float m; /* Résultat de la multiplication */

 a=0; /* Initialisation des variables */
 b=0;

 printf("Calculatrice :\n\n");
 printf("Valeur de a : ");
 scanf("%f",&a);
 printf("\n");
 printf("Valeur de b : ");
 scanf("%f",&b);

 m = a*b;
 printf("Valeur de a*b : %f\n", m);
 /* Affichage de la multiplication */

 getchar ();
 return (0);
}
```

Pour l'addition :

```
m = a + b;
printf ("Valeur de a+b : %f", m);
```

Pour la soustraction :

```
m = a - b;
printf ("Valeur de a-b : %f", m);
```

Calculer la somme  $a + b + c$

programme17.c

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

int main ()
{
 float a,b,c,s; /* Déclaration des variables */

 a=0; /* Initialisation des variables */
 b=0;
 c=0;
 s=0;

 printf("Saisie de a : ");
 scanf("%f",&a);
 printf("Saisie de b : "); /* Saisie variables flottantes */
 scanf("%f",&b);
 printf("Saisie de c : ");
 scanf("%f",&c);

 s = a+b+c; /* Calcul de la somme */
 printf("Valeur de s : %.2f\n",s); /* Affichage de la somme */

 getchar ();
 return (0);
}
```

Calculer la différence  $a - b - c$

```
d = a-b-c; /* Calcul de la différence */
printf("Valeur de d : %.2f\n",d); /* Affichage de la différence */
```

Calculer la multiplication  $a * b * c$

```
m = a*b*c; /* Calcul de la multiplication */
printf("Valeur de m : %.2f\n",m); /* Affiche la multiplication */
```

Calculer la division  $a / b / c$

```
d = a/b/c; /* Calcul de la division */
printf("Valeur de d : %.2f\n",d); /* Affichage de la division */
```

Calculer la valeur absolue de  $a - b$

```
r=abs(a-b); /* Calcul de la valeur absolue */
printf("Valeur de r : %f\n",r); /* Affiche la valeur absolue */
```

Calculer l'arrondi de  $a + b$

```
c=ceil(a/b); /* Calcul de l'arrondi */
printf("Valeur de c : %f\n",c); /* Affichage de l'arrondi */
```

Il aurait été possible d'utiliser `%d` du fait que l'arrondi est un nombre entier !



# Chapitre 4

## Les conditions

### 4.1 Exercice de mise en bouche

Ecrire un programme qui met en application le théorème de Pythagore pour calculer l'hypothénuse d'un triangle rectangle.

**Rappel :** Dans un triangle rectangle, l'hypothénuse (le plus grand côté) peut se calculer en appliquant la formule suivante :

où a et b sont les longueurs des côtés adjacents à l'angle droit.

$$\text{hypothénuse} = \sqrt{a^2 + b^2}$$

**Notes :** – La racine carrée s'obtient par l'utilisation de la fonction `sqrt` (valeur) contenue dans la bibliothèque `math.h`. (`#include <math.h>`)  
–  $a^2$  peut s'obtenir par `a*a`.

**Méthodologie :**

1. Rechercher les variables nécessaires et les déclarer dans le programme.
2. Faire saisir a au clavier.
3. Faire saisir b au clavier.
4. Effectuer l'opération de la racine carrée et afficher le résultat.

*Attention :* Afin de pouvoir utiliser la bibliothèque mathématique du C (`#include <math.h>`), il est nécessaire d'ajouter au moment de l'édition de liens **-lm** ce qui nous donne :

```
gcc -o monprog monprog.o -lm
```

## 4.2 Les conditions : Si Alors Sinon

```
if (condition vraie)
{
 instructions 1
}
else
{
 instructions 2
}
```

```
si (condition vraie)
{
 alors
 faire instructions 1
}
sinon
{
 faire instructions 2
}
```

Les conditions s'expriment avec des opérateurs logiques.

## 4.3 Opérateurs logiques

Ils servent à comparer deux nombres entre eux.

| Libellé           | Opérateur |
|-------------------|-----------|
| Inférieur         | <         |
| Supérieur         | >         |
| Egal              | ==        |
| Différent         | !=        |
| Inférieur ou égal | <=        |
| Supérieur ou égal | >=        |

## 4.4 Opérateurs logiques purs

Ce sont des opérateurs logiques permettant de combiner des expressions logiques.

| Libellé   | Opérateur |
|-----------|-----------|
| Et (and)  | &&        |
| Ou (or)   |           |
| Non (not) | !         |

| se nomme en informatique un pipe (prononcer païpe).

## 4.5 Vrai ou faux

La valeur `Vrai` peut être assimilée à la valeur numérique 1 ou à toute valeur non nulle.

La valeur `Faux` peut être assimilée à la valeur numérique 0.

L'opérateur `Ou (||)` correspond alors à une addition.

|      |      |      |   |   |   |
|------|------|------|---|---|---|
| Ou   | Vrai | Faux | + | 1 | 0 |
| Vrai | Vrai | Vrai | 1 | 2 | 1 |
| Faux | Vrai | Faux | 0 | 1 | 0 |

L'opérateur `Et (&&)` correspond alors à une multiplication

|      |      |      |   |   |   |
|------|------|------|---|---|---|
| Et   | Vrai | Faux | * | 1 | 0 |
| Vrai | Vrai | Faux | 1 | 1 | 0 |
| Faux | Faux | Faux | 0 | 0 | 0 |

*On notera que `!Vrai = Faux` et `!Faux = Vrai`.*

## 4.6 Combinaison

Toutes les opérations logiques peuvent se combiner entre elles. La seule condition d'utilisation d'un `si (if)` avec de telles combinaisons est de l'entourer de `()`.

Exemple : `if ((car == 'a') || (car == 'A'))`

## 4.7 Astuce

Vous verrez souvent ce type de code écrit :

```
if (er)
{
 /* Alors faire quelque chose */
}
```

En appliquant ce qui a été vu précédemment, on en déduit que ce code signifie que

```
si (er != 0) /* si er différent de 0 */
{
 /* Alors faire quelque chose */
}
```

## 4.8 Les accolades

Les accolades entourant les blocs d'instructions d'une condition peuvent être omises si le bloc n'est constitué que d'une seule instruction.

Exemple :

```
if (car == 'b')
 printf ("car vaut b.");
else
 printf ("car est différent de b.");
```

## 4.9 Exercices

1. Faites saisir une variable de type entier et indiquez à l'utilisateur si celle-ci est strictement positif ou strictement négatif ou nulle.

Aide :

```
if (a>0)
 printf ("Valeur positive");
else
 printf ("Valeur négative");
```

2. Faites saisir une variable de type caractère et indiquez à l'utilisateur si celle-ci est une voyelle ou une consonne. On considèrera que le caractère saisi est en minuscule.

## Corrections des exercices du chapitre 4

Écrire un programme qui met en application le théorème de Pythagore.

programme18.c

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

int main ()
{
 float a; /* base du triangle */
 float b; /* côté du triangle rectangle */
 float p; /* valeur de l'hypoténuse (p pour Pythagore !) */

 /* Initialisation des variables pour palier aux erreurs */
 a = 0;
 b = 0;

 /* Saisie de a */
 printf ("Valeur de la base : ");
 scanf ("%f",&a);

 /* Saisie de b */
 printf ("Valeur du côté : ");
 scanf ("%f",&b);

 /* Calcul par Pythagore */
 p = sqrt (a*a + b*b);

 /* Affichage du résultat */
 printf ("L'hypoténuse mesure : %.2f\n",p);

 /* Attendre avant de sortir */
 getchar ();

 return (0);
}
```

Tester le signe d'une valeur saisie au clavier.

programme19.c

```
#include <stdio.h>

int main ()
{
 /* Valeur que l'on va saisir */
 int a = 0;

 /* Saisie de a */
 printf("Saisie de a : ");
 scanf("%d",&a);

 /* Test condition a<0 */
 if (a<0)
 {
 printf("la variable a est négative.\n");
 }
 else
 {
 /* Test condition a>0 */
 if (a>0)
 {
 printf("la variable a est positive\n");
 }
 /* Sinon a est nulle */
 else
 {
 printf("la variable a est nulle\n");
 }
 }

 getchar ();
 return (0);
}
```

Tester si un caractère saisi au clavier est une consonne ou une voyelle.

programme20.c

```
#include <stdio.h>

int main ()
{
 /* Valeur que l'on va saisir */
 char car;

 /* Saisie du caractère a */
 printf("Saisie du caractère : ");
 scanf("%c",&car);

 /* Test condition car voyelle minuscule */
 if ((car == 'a') || (car == 'e') || (car == 'i') || (car == 'o') ||
 (car == 'u') || (car == 'y'))
 {
 printf("la variable car est une voyelle.\n");
 }
 else
 {
 printf("la variable car est une consonne.\n");
 }

 getchar ();
 return (0);
}
```

# Chapitre 5

## Mise au point

### 5.1 Prologue

L'objet de ce cours est de réaliser un petit break dans l'apprentissage du C et de s'attacher à voir ce que l'on est capable de réaliser avec le peu de moyen que l'on a.

Ce cours sera donc constitué de 3 exercices de difficulté croissante avec apprentissage d'une nouvelle fonction et d'un exercice complet de programmation.

### 5.2 Exercice 1

Réaliser un programme qui saisisse un nombre et indique à l'utilisateur si celui-ci est plus grand ou plus petit qu'un autre nombre fixé par le programme.

*Exemple :*

```
si (nombre_saisi < 10)
alors "plus petit"
```

Reprendre l'exercice du chapitre 4 qui disait si un nombre est strictement positif, strictement négatif ou nul.

### 5.3 Retour sur `getchar()`

La fonction `getchar ()` permet d'attendre la frappe d'un caractère au clavier, de le lire et de le renvoyer. 2 utilisations peuvent être faites de `getchar ()`, la première est celle permettant d'attendre la frappe d'une touche sans se soucier de sa valeur, la seconde est celle permettant de lire un caractère au clavier.



Exemples :

1. Attente  
`getchar ();`
2. Saisie d'un caractère  
`char car;`  
`car = getchar ();`

A chaque fois, `getchar ()` effectue le même traitement :

- Attendre la frappe d'une touche au clavier suivi d'un retour chariot (Entrée).
- Renvoyer le caractère frappé.

Dans le 1<sup>er</sup> cas, ce caractère n'est simplement pas récupéré.

## 5.4 Boucle Faire ... Tant que (vrai)

`Do ... while`, traduit par Faire ... Tant que permet de réaliser une suite d'instructions tant qu'une condition ou un ensemble de conditions est rempli.

Exemple :

programme21.c

```
#include <stdio.h>

int main ()
{
 char car;
 int sortie;

 do
 {
 printf ("Tapez S pour sortir !\n");

 /* On saisit un caractère */
 car = getchar ();

 /* On le compare pour savoir si l'on peut sortir */
 sortie = ((car == 's') || (car == 'S'));
 } while (!sortie);

 return (0);
}
```

Rappel :

- Un nombre entier vaut la valeur logique vraie si celui-ci est différent de 0.
- Un nombre entier vaut la valeur logique faux si celui-ci est égal à 0.
- `||` signifie un ou logique (or).

*Remarque :* Cette utilisation n'est pas très belle, le retour chariot utilisé pour la saisie du caractère étant renvoyé et interprété nous donne un affichage double. Malgré cela, au niveau de ce cours, nous nous en contenterons.

## 5.5 Exercice 2

Tapez l'exemple précédent, aménagez le, comprenez le, puis transformez le afin que l'on sorte de la boucle uniquement lorsque l'utilisateur a tapé le nombre 10.

*Attention :* La saisie d'un nombre ne se fait pas par `getchar` mais par `scanf` (cf. Chapitre 3).

## 5.6 Exercice 3

Voici un petit exemple de programme qui permet d'obtenir des nombres aléatoires entre 0 et 100.

programme22.c

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

int main()
{
 int nb_alea; /* Nombre aléatoire */

 srand (time (NULL)); /* Initialisation des nombres aléatoires */

 /* Le nombre aléatoire est stocké dans une variable puis affiché */
 nb_alea = (int) (((float) rand () / RAND_MAX) * 100);
 printf ("%d",nb_alea);

 getchar ();
 return (0);
}
```

`srand (time (NULL))` permet d'initialiser le système aléatoire.

`rand ()` permet d'obtenir un nombre entre 0 et `RAND_MAX`.

`(float) rand ()` permet de dire au langage C que l'on désire réaliser des calculs avec des nombres à virgules (flottants).

`((float) rand () / RAND_MAX)` permet d'obtenir un nombre entre 0 et 1, qui multiplié par 100 nous donne un nombre entre 0 et 100.

En vous aidant de ce petit programme et de ce qui a été fait précédemment, réalisez un petit jeu qui :

1. Initialise un nombre entre 0 et 100;
2. Tente de faire deviner ce nombre à l'utilisateur en lui indiquant s'il est plus petit ou plus grand.

## Corrigés des exercices du chapitre 5

programme23.c

```
#include <stdio.h>

int main ()
{
 int nb_choisi = 33;
 int nb_saisi = 0;

 printf ("Votre nombre : ");
 scanf ("%d",&nb_saisi);

 if (nb_choisi < nb_saisi)
 printf ("Mon nombre est plus petit.\n");
 else
 {
 if (nb_choisi == nb_saisi)
 printf ("Mon nombre est égal.\n");
 else
 printf ("Mon nombre est plus grand.\n");
 }

 /* Attente */
 getchar ();

 return (0);
}
```

programme24.c

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

int main ()
{
 int valeur;

 do
 {
 printf ("Votre nombre : ");
 scanf ("%d",&valeur);
 }
 while (valeur != 10);

 return (0);
}
```

programme25.c

```
#include <stdio.h>
#include <stdlib.h> /* pour les valeurs aléatoires */
#include <time.h>

int main ()
{
 int nb_hasard = 0;
 int votre_nb = 0;

 srand (time (NULL));
 nb_hasard = (int) (((float) rand () / RAND_MAX) * 100); /* Nombre entre 0 et 100

 do
 {
 printf("Votre nombre : ");
 scanf("%d",&votre_nb);

 if (nb_hasard < votre_nb)
 printf ("\nMon nombre est plus petit\n");
 else
 {
 if (nb_hasard > votre_nb)

 /* il peut être aussi égal ... */
 printf ("\nVotre nombre est plus grand\n");
 }
 }
 while (votre_nb != nb_hasard);

 printf ("Trouvé\n");
 getchar ();

 return (0);
}
```

## Chapitre 6

# Et les shadocks pompaient

### 6.1 Préambule

Du fait que la fonction `getchar()` attend l'appui de la touche `Enter`, nous allons utiliser une bibliothèque créée pour le cours. Cette bibliothèque se nomme `ASCIART` et est détaillée en annexe.

Elle donne entre autres accès à la fonction `getch()` qui permet la saisie d'un caractère sans attente de retour chariot.

Pour l'utiliser, il nous faut :

- ajouter `#include "asciiart.h"` à notre programme (ce fichier est livré avec la bibliothèque);
- linker le programme avec `asciiart.o: gcc -o mon_prog mon_prog.o asciiart.o`

Vous pouvez utiliser cette fonction dans les exercices, chaque fois que vous souhaitez attendre l'appui d'une touche. Une fonction similaire (`getche()`) fait le même travail mais affiche le caractère Entrée.

On remarquera que le `#include` est différent de celui utilisé précédemment, les `< . . . >` ont été remplacés par des `" "`.

L'explication en est simple, les bibliothèques du compilateur sont insérées via `#include <nom de la bibliothèque>` et les personnelles sont insérées via `#include "nom de la bibliothèque"`. A noter que dans le cas d'une bibliothèque personnelle, celle-ci doit se situer dans le même répertoire que le programme source (du moins on le considèrera dans le cadre de ce cours).

### 6.2 While

De la même façon que `do { ... } while(condition) ;`, il est possible d'utiliser :

```
while (condition == Vrai) { ... }
```

*Exemple :*

```
char car = ' ';
while ((car != 's') && (car != 'S'))
{
 car = getche ();
}
```

*Attention :* `while (condition) ;` signifie que tant que condition est vrai, on revient à la même ligne. Si condition est toujours vraie, on tombe alors dans un piège. Si condition est fausse, on passe alors immédiatement à la ligne/commande suivante.

### 6.3 Et les Shadoks apprennent que reprendre équivaut à apprendre

*Exercice 1* Traduisez en langage C, complétez avec les variables nécessaires, compilez, exécutez, comprenez :

```
Faire
 Saisir une touche
Tant Que (touche != S) et (touche != s)
```

*Exercice 2* Traduisez en langage C, complétez avec les variables nécessaires, compilez, exécutez, comprenez :

```
Faire
 Saisir un nombre
Tant Que (nombre != 10)
```

*Attention :* La saisie d'un nombre ne se fait pas par la fonction `getch ()` mais par la fonction `scanf` (reportez vous au chapitre correspondant pour plus de précisions).

### 6.4 Fonction `toupper ()`

Le problème de la comparaison de la minuscule et de la majuscule de l'exercice 1 peut être contourné par l'utilisation de la fonction `toupper (#include <ctype.h>)` qui transforme un caractère minuscule en majuscule.

La fonction `toupper` s'utilise de la façon suivante :

```
int main ()
{
 char car;
 char car_min;

 car_min = 'a'
 car = toupper (car_min);
 printf ("%c", car);

 return (0);
}
```

Le programme précédent affichera A.

## 6.5 O tant que en emporte le Shadok

*Exercice 3* Ecrire le programme :

Tant que je ne tape pas un nombre impair compris entre 1 et 10 je recommence la saisie d'un nombre

*Exercice 4* Ecrire le programme :

Tant que je ne tape pas une voyelle je recommence la saisie d'une touche

## 6.6 Et les Shadoks continuaient à pomper pour obtenir le résultat

Dans les exercices qui suivent, la notion de compteur intervient. Un compteur est une variable numérique que l'on décrémente (-1) ou incrémente (+1) suivant nos besoins.

*Exemples :*

```
int i;

i++; /* Incrémente le compteur i */
i--; /* Décrémente le compteur i */
i++; /* revient à la même chose que i=i+1; */
i--; /* revient à la même chose que i = i-1; */
```

Dans les exemples précédents, le nombre de caractères entrés peut donc être comptabilisé en ajoutant 1 à une variable à chaque fois qu'une touche est frappée.

*Exercice 5* Ecrire le programme :

Tant que je n'ai pas saisi 10 caractères, je recommence la saisie d'une touche



*Exercice 6* Ecrire le programme :

Tant que je n'ai pas saisi 10 nombres, je recommence la saisie d'un nombre

## **6.7 Au clan des Shadoks, on trie, voyelles, chiffres premiers**

*Exercice 7* Ecrire le programme :

Tant que je n'ai pas saisi 10 voyelles, je recommence la saisie d'une touche  
On prendra soin d'indiquer à l'utilisateur combien de voyelles il lui reste à entrer.

*Exercice 8* Ecrire le programme :

Tant que je n'ai pas saisi 10 chiffres premiers (1,3,5,7), je recommence la saisie d'un chiffre  
On prendra soin d'indiquer à l'utilisateur combien de chiffres premiers il lui reste à entrer.

## Corrigés des exercices du chapitre 6

### Exercice 1

programme26.c

```
#include <stdio.h>
#include "asciiarth"

int main ()
{
 char car = 0;
 printf ("Tapez S pour arrêter ...\n");
 do
 {
 car = getch ();
 }
 while ((car != 's') && (car != 'S'));
 return (0);
}
```

### Exercice 2

programme27.c

```
#include <stdio.h>

int main ()
{
 int nbre = 0;
 printf ("Tapez 10 pour arrêter ...\n");
 do
 {
 scanf ("%d", &nbre);
 }
 while (nbre != 10);
 return (0);
}
```

### Exercice 3

programme28.c

```
#include <stdio.h>

int main ()
{
 int nbre = 0;

 printf ("Tapez un chiffre impair pour arrêter ...\n");
 while ((nbre != 1) && (nbre != 3) && (nbre != 5) && (nbre != 7) && (nbre != 9))
 {
 scanf ("%d", &nbre);
 }
 return (0);
}
```

ou bien :

programme29.c

```
#include <stdio.h>

int main ()
{
 int nbre = 0;

 printf ("Tapez un chiffre impair pour arrêter ...\n");
 while ((nbre < 0) || (nbre > 10) || (nbre%2==0))
 {
 scanf ("%d", &nbre);
 }
 return (0);
}
```

*Exercice 4*

programme30.c

```
#include <stdio.h>
#include <ctype.h>
#include "asciiart.h"

int main ()
{
 char car = ' ';

 printf ("Tapez une voyelle pour arrêter ...\n");

 while ((car != 'A') && (car != 'E') && (car != 'I')
 && (car != 'O') && (car != 'U') && (car != 'Y'))
 {
 car = getch ();
 car = toupper (car);
 }
 return (0);
}
```

*Exercice 5*

programme31.c

```
#include <stdio.h>
#include "asciiart.h"

int main ()
{
 char car = ' ';
 int nbre = 0;

 printf ("Tapez 10 caractères pour arrêter ...");

 do
 {
 car = getch ();
 nbre ++;
 }
 while (nbre != 10);
 return (0);
}
```

*Exercice 6*

programme32.c

```
#include <stdio.h>

int main ()
{
 int nbre = 0;
 int nb_nbre = 0;

 printf ("Tapez 10 nombres pour arrêter ...");

 do
 {
 scanf ("%d",&nbre);
 nb_nbre ++;
 }
 while (nb_nbre != 10);

 return (0);
}
```

*Exercice 7*

programme33.c

```
#include <ctype.h>
#include "asciiart.h"

int main ()
{
 char car;
 int nb_nbre = 10;

 printf ("Tapez encore %d voyelles pour arrêter...\n",nb_nbre);
 do
 {
 car=getch();
 car=toupper(car);
 if (car=='A' || car=='E' || car=='I' || car=='O' || car=='U')
 {
 nb_nbre--;
 printf ("Tapez encore %d voyelles pour arrêter...\n",nb_nbre);
 }
 }
 while (nb_nbre != 0);

 return (0);
}
```

*Exercice 8*

programme34.c

```
#include <ctype.h>
#include "asciiart.h"

int main ()
{
 int nb_nbre = 10;
 int nbre;

 printf ("Tapez encore %d chiffres premiers...\n",nb_nbre);
 do
 {
 scanf("%d",&nbre);
 if (nbre==1 || nbre==2 || nbre==3 || nbre==5 || nbre==7)
 {
 nb_nbre--;
 printf ("Tapez encore %d chiffres premiers arrêter...\n",nb_nbre);
 }
 }
 while (nb_nbre != 0);

 return (0);
}
```

# Chapitre 7

## Les boucles

### 7.1 Et les shadoks pédalèrent pendant 15 tours

Pour faire effectuer un certain nombre de fois une tâche, on utilise l'instruction `for` de la façon suivante (avec `i`, une variable de type entier (`int` par exemple)).

```
for (i=point de départ; i<point d'arrivée; i=i+pas)
{
 instruction(s) répétée(s);
}
```

Par souci de simplicité, nous dirons simplement que la formule suivante :

```
for (i=0; i<15; i++)
{
 instr;
}
```

signifie que l'on va exécuter `instr` pour `i` variant de 0 à 14 (<15) c'est à dire 15 fois.

*Exemple :*

programme35.c

```
#include "asciiart.h"
#include <stdio.h>
int main ()
{
 int i;
 for (i=0; i<15; i++)
 {
 printf ("Je me répète pour i valant %d\n",i);
 }
 printf ("Je me suis répétée... 15 fois\n");
 getch ();
 return (0);
}
```

## 7.2 Syntaxe

De la même façon que le `if`, le `for` ne nécessite pas d'accolades si le nombre d'instructions à répéter est de 1.

*Exemple :* On peut utiliser cette fonctionnalité dans le programme précédent en remplaçant :

```
for (i=0; i<15; i++)
{
 printf ("Je me répète pour i valant %d\n",i);
}
```

par

```
for (i=0; i<15; i++)
 printf ("Je me répète pour i valant %d\n",i);
```

*Exercice 1* Utilisez une boucle pour `i` variant de 0 à 7 inclus pour afficher : « Ceci est la couleur `i` » où `i` est remplacé par sa valeur et où la phrase est écrite avec la couleur `i`.

*Aide :* La fonction `textcolor(i)` de notre bibliothèque `asciiart` change la couleur courante :

```
textcolor (i);
printf ("Ceci est la couleur %d",i);
```

La fonction `textreset()` de la bibliothèque permet de remettre les valeurs par défaut (à employer avant de terminer un programme).



## 7.3 Notion de double boucle

Il est possible de remplacer les instructions par une boucle afin de réaliser une double boucle. On obtient donc :

```
Pour i allant de ... à ...
{
 ...
 Pour j allant de ... à ...
 {
 ...
 }
}
```

*Exemple :*

programme36.c

```
#include "asciiart.h"
#include <stdio.h>
int main ()
{
 int i;
 int j;
 for (i=0; i<5; i++)
 {
 printf ("\nJe suis dans la boucle i, i vaut %d\n",i);

 for (j=3; j>0; j--)
 {
 printf ("Je suis dans la boucle j, j vaut %d\n",j);
 }
 }
 getch ();
 return (0);
}
```

*Exercice 2* En utilisant la double boucle, écrire un programme qui écrit une étoile, passe à la ligne, écrit deux étoiles, passe à la ligne, écrit trois étoiles... jusqu'à cinq étoiles afin d'obtenir ceci :

```
*
**


```

## 7.4 Exercice 3 : Et les Shadoks fêtèrent Noël...

### 7.4.1 « Cône » du sapin

À l'aide d'une double boucle, réalisez un cône pour dessiner le haut du sapin en vert sur 10 lignes. Vous devriez obtenir ceci :

```

 *


```

*Aide :*

- Sur la ligne n<sup>o</sup> 1, on affiche 9 espaces puis 1 étoile ;
- Sur la ligne n<sup>o</sup> 2, on affiche 8 espaces puis 3 étoiles ;
- Sur la ligne n<sup>o</sup> 3, on affiche 7 espaces puis 5 étoiles ;
- Sur la ligne n<sup>o</sup> i, on affiche 10-i espaces puis 2\*i-1 étoiles.

On obtient donc par exemple pour l'affichage des étoiles sur la ligne i :

```
for (j=0; j<((2*i)-1); j++)
{
 printf ("*");
}
```

### 7.4.2 Affichage du tronc

Pour poursuivre le sapin, il nous faut maintenant dessiner le tronc. Ecrire la suite du programme en dessinant le tronc à l'aide du caractère @. Vous devriez obtenir ceci (en brun) :

```

@@@
@@@
@@@

```

### 7.4.3 Affichage des boules de Noël

On peut rajouter des boules de Noël dans le sapin en modifiant le dessin du feuillage et en remplaçant aléatoirement certains caractères @ verts par un O de couleur différente.

*Aide :* On peut tirer un nombre aléatoire entre 0 et 9 ainsi (voir chapitre 5) :

```
int i;
srand (time (NULL));
i=rand()%10;
```

On pourra donc remplacer la ligne qui trace un élément de feuillage :

```
printf("@");
```

par :

```
if (rand()%10==0)
{
 textcolor(rand()%8);
 printf("O");
}
else
{
 textcolor(2);
 printf("@");
}
```

## 7.5 Exercice 4 : Table Ascii

Les codes Ascii (i.e. les nombres qui représentent les caractères en informatique) vont de 0 à 255. Ecrire un programme qui fait afficher sur des lignes successives puis de façon propre, par exemple un tableau sur 8 colonnes, les codes Ascii avec les caractères qui leur correspondent (On pourra commencer l'afficher à partir du code 32).

*Aide :* Pour faire afficher le caractère associé à un code Ascii, on écrit :

```
printf ("%03d : %c", code_ascii, code_ascii);
```

*Exemple :*

```
int i = 65;
printf ("%03d : %c", i, i); // affichera 065 : A
```

## Corrigés des exercices du chapitre 7

### Exercice 1

programme37.c

```
#include <stdio.h>
#include <stdlib.h>
#include "asciiart.h"
int main ()
{
 int i;
 for (i=0; i<=7; i++)
 {
 textcolor (i);
 printf ("Ceci est la couleur %d\n", i);
 }
 getch ();
 return(0);
}
```

### Exercice 2

programme38.c

```
#include <stdio.h>
#include <stdlib.h>
#include "asciiart.h"
int main ()
{
 int i;
 int j;
 for (i=1; i<=5; i++)
 {
 for (j=1; j<=i; j++)
 {
 printf ("*");
 }
 printf ("\n");
 }
 getch ();
 return(0);
}
```

Exercice 3

programme39.c

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

int main ()
{
 int i;
 int j;

 /* Initialisation des nombres aléatoires */
 srand (time (NULL));

 textcolor(2);
 for (i=1; i<=10; i++)
 {
 for (j=0; j<10-i; j++)
 {
 printf(" ");
 }
 for (j=1; j<= (i*2-1); j++)
 {
 if (rand()%10==0)
 {
 textcolor(rand()%8);
 printf("O");
 }
 else
 {
 textcolor(2);
 printf("*");
 }
 }
 printf ("\n");
 }
 textcolor (3);
 for (i=1; i<=3; i++)
 {
 printf (" @@@\n");
 }
 textreset ();
 getch ();
 return (0);
}
```

#### Exercise 4

programme40.c

```
#include <stdio.h>
#include <stdlib.h>
#include "asciiart.h"
int main ()
{
 int i;
 int j;

 for (i=4; i<32; i++)
 {
 for(j=0; j<8; j++)
 {
 printf ("%03d %c ",i*8+j,i*8+j);
 }
 printf("\n");
 }
 getch ();
 return (0);
}
```

programme41.c

```
#include <stdio.h>
#include <stdlib.h>
#include "asciiart.h"
int main (void)
{
 int i;
 int j;
 int car = 32;

 printf (" 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15\n");
 for (i=3; i<=16; i++)
 {
 for (j=1; j<=16; j++)
 {
 printf (" %c",car);
 car ++;
 }
 printf ("\n");
 }
 getch ();
 return (0);
}
```

# Chapitre 8

## Pointeurs et Fonctions

Peu d'exercices dans ce cours dans lequel l'important est de comprendre les nombreux exemples cités.

### 8.1 Variables : pointeurs et valeurs

#### 8.1.1 Les variables et la mémoire.

Une variable (par exemple `char car`) est en fait une petite zone mémoire ici de 1 octet que l'on s'alloue et où l'on va ranger les informations. Chaque type de variable utilise au moins 1 octet. Le type `int` varie selon les compilateurs, pour nous il utilise 4 octets c'est à dire 4 cases mémoire. Nous avons vu précédemment qu'il était possible de voir le contenu d'une variable avec la fonction `printf` et qu'il était possible de mettre une valeur dans une variable avec l'opérateur `=` ou la fonction `scanf`.

*Exemple :*

```
char car = 'C';
printf ("%c", car);
```

Représentons-nous la mémoire comme une rue remplie de maisons. Chaque case mémoire est représentée par une maison. Chaque maison porte un numéro, son adresse postale, pour une case mémoire, on parlera d'adresse mémoire. L'adresse est unique pour chaque maison. Cependant, rien ne vous empêche de vous tromper intentionnellement ou non de maison. De la même façon, une adresse mémoire d'une case mémoire est unique mais rien ne vous empêche de vous tromper intentionnellement ou non de case mémoire.

*Exemple de la vie courante :*

```
Mr Leneuf adresse : 99, grand rue
```

*Exemple en C :*

```
char car adresse : 0x001F (soit 31 en décimal)
```

0x signifie adresse hexadécimale (en base 16).

En langage C, l'adresse mémoire est accessible en faisant précéder la variable de l'opérateur &.

*Exemple :*

```
char car = 'C';
int nbre = 12;

printf ("Adresse de car : %lu [%lx]", &car, &car);
printf ("Adresse de nbre : %lu [%lx]", &nbre, &nbre);
```

On ajoute "l" devant le u (%lu) et devant le x (%lx) afin de faire afficher un entier long (> 65536). u signifie un nombre non signé. *Les crochets ne sont là que pour des raisons d'esthétique.*

### 8.1.2 Pointeurs

Pour utiliser les adresses mémoire des variables à la place des variables elles-mêmes, on utilise les pointeurs. Les pointeurs sont définis par un type et se déclarent de la façon suivante :

```
type* variable;
ou
type *variable;
```

Le signe \* indique l'utilisation d'un pointeur i.e. l'utilisation d'une adresse mémoire. Le type permet simplement de savoir comment le compilateur C doit interpréter l'adresse mémoire lors de sa lecture. Nous en verrons des exemples lors de l'utilisation des chaînes de caractères.

Pour enregistrer une valeur dans une adresse mémoire on écrit :

```
*variable = <valeur>
```

\*variable signifie contenu de l'adresse mémoire.



*Exemple :*

programme42.c

```
#include <stdio.h>

int main ()
{
 char car='C';
 char* ptr_car = NULL;

 printf ("Avant, le caractère est : %c\n",car);
 ptr_car = &car; /* ptr_car = adresse de car */
 ptr_car = 'E'; / on modifie le contenu de l'adresse mémoire */
 printf ("\nAprès le caractère est : %c\n",car); /* i.e. on a modifié car */

 return (0);
}
```

NULL signifie l'adresse 0. Il indique que l'adresse mémoire est invalide.

Explicitons cet exemple à l'aide de notre rue bordée de maison.

```
ptr_car = &car; Signifie que l'on m'a donné l'adresse postale de M. car.
*ptr_car = 'E'; Signifie que je rentre chez M. car et que j'y dépose le caractère E.
printf ("%c",car); Signifie que l'on va lire le contenu de car pour l'afficher.
```

### 8.1.3 Exercices d'application

*Exercice 1* Réaliser un programme équivalent qui change la valeur en K.

*Exercice 2* Réaliser un programme équivalent qui change une valeur numérique (int) de 10 à 35.

## 8.2 Les fonctions

### 8.2.1 Définition générale

Une fonction est un petit bloc de programme qui à l'image d'une industrie va créer, faire ou modifier quelque chose.

Un bloc de programme est mis sous la forme d'une fonction si celui-ci est utilisé plusieurs fois dans un code (un programme) ou simplement pour une question de clarté (Imaginez un livre sans paragraphe!).

A l'identique de notre fonction principale `main ()`, une fonction s'écrit de la façon suivante :

```
<type de sortie> <nom de la fonction> (<paramètres d'appels>)
{
 Déclaration des variables internes à la fonction

 Corps de la fonction

 Retour
}
```

### 8.2.2 Void

Le type void signifie indéfini, il est utilisé :

- comme type de sortie pour les fonctions qui ne retournent aucun résultat.
- dans la déclaration de pointeurs sur des zones mémoires dont on ne connaît pas le type.

*Exemples :*

```
/* Pointeur sur une zone mémoire indéfinie */
void* m_ptr;

/* Fonction bête affichant un caractère */
void Affiche_car (char car)
{
 printf ("%c", car);
}
```

### 8.2.3 Variables globales et locales

Les variables déclarées dans les fonctions sont dites locales. Il existe aussi les variables dites globales qui sont déclarées en dehors de toute fonction y compris le main ().

Les variables globales sont modifiables et accessibles par toutes les fonctions sans avoir besoin de les passer en paramètre. Il est de ce fait extrêmement dangereux d'utiliser des variables globales.

Les variables locales ne sont modifiables et accessibles que dans la fonction où elles sont déclarées. Pour les modifier ou les utiliser par une autre fonction, il est nécessaire de les passer en paramètres.

*Exemple de variable locale :*

programme43.c

```
#include <stdio.h>

int carre (int val)
{
 int val_retour = 0; /* Déclaration variable locale */

 val_retour = val * val;
 return (val_retour);
}

int main ()
{
 int val_retour = 0; /* Déclaration variable locale */

 val_retour = carre (2);
 printf ("Le carré de 2 est : %d\n", val_retour);

 return (0);
}
```

val\_retour est dans les deux cas une variable locale. Bien qu'elles aient le même nom dans les fonctions main et carre, les deux variables n'ont rien à voir entre elles.

*Exemple de variable globale :*

programme44.c

```
#include <stdio.h>

int val = 0;
int val_retour = 0;

void carre ()
{
 val_retour = val * val;
}

int main ()
{
 val = 2;

 carre ();
 printf ("Le carré de 2 est %d\n", val_retour);

 return (0);
}
```

val est une variable globale.

On constate que le programme devient rapidement illisible...

## 8.2.4 Utilisation et modification de données dans les fonctions

L'appel d'une fonction peut s'effectuer à l'aide de paramètres.  
Ces paramètres peuvent être utilisés en les déclarant dans les parenthèses.

*Exemple :*

programme45.c

```
#include <stdio.h>

void Affiche_Nombre (int no)
{
 printf ("Le nombre est : %d\n",no);
}

int main ()
{
 Affiche_Nombre (12);

 return (0);
}
```

Ces paramètres peuvent être modifier à condition qu'ils soient passés par adresse c'est à dire que la fonction reçoive un pointeur.

Exemple :

programme46.c

```
#include <stdio.h>

void Avance_Position (int* x)
{
 *x = *x + 2;
}

int main ()
{
 int i=0;
 int x=0;

 printf ("Position de départ : %d\n", x);

 for (i = 0; i<5; i++)
 {
 Avance_Position (&x);
 printf ("Nouvelle position : %d\n", x);
 }

 return (0);
}
```

### 8.2.5 Piège!

Attention, les opérateurs unaires (avec une seule opérande) sur les pointeurs sont dangereux.

```
int* x;
*x++;
```

augmentera l'adresse de x (on va chez le voisin) et non la valeur, pour cela il faut écrire :

```
(*x)++;
```

Ceci est une faute courante, prenez garde ...

### 8.2.6 Exercice 3 :

Reprendre le programme du sapin de Noël du chapitre précédent et réaliser 2 fonctions : `Cone (int n)` & `Tronc (int n)` qui dessine respectivement le cône du sapin sur n lignes et le tronc en position n (n blancs avant le tronc).

Ecrire un programme qui, utilisant ces deux fonctions, dessine un sapin sur n lignes.

## Corrigés des exercices du chapitre 8

### Exercice 1

programme47.c

```
#include <stdio.h>

int main ()
{
 char car = 'C';
 char* ptr_car = NULL;

 printf ("Avant le caractère est : %c\n", car);
 ptr_car = &car;
 *ptr_car = 'K';
 printf ("Après le caractère est : %c\n", car);

 return (0);
}
```

### Exercice 2

programme48.c

```
#include <stdio.h>

int main ()
{
 int val = 10;
 int* ptr_val = NULL;

 printf ("Avant le nombre est : %d\n", val);
 ptr_val = &val;
 *ptr_val = 35;
 printf ("Après le nombre est : %d\n", val);

 return (0);
}
```

### Exercice 3

programme49.c

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

// Dessin du cône du sapin
void cone (int n)
{
 int i=0, j=0;

 textcolor(2);

 /* Initialisation des nombres aléatoires */
 srand (time (NULL));

 for (i=1; i<=n; i++)
 {
 for (j=0; j<n-i; j++)
 {
 printf(" ");
 }

 for (j=1; j<= (i*2-1); j++)
 {
 if (rand()%10==0)
 {
 textcolor(rand()%8);
 printf("O");
 }
 else
 {
 textcolor(2);
 printf("*");
 }
 }
 printf ("\n");
 }
}
```

```
void tronc (int n)
{
 int i=0, j=0;

 textcolor (3);
 for (j=1; j<=3; j++)
 {
 for (i=1; i<=n; i++)
 {
 printf (" ");
 }

 printf ("@@@\n");
 }
}

int main ()
{
 int nb_lig = 15;

 cone (nb_lig);
 tronc (nb_lig - 2);

 textreset ();

 getch ();
 return (0);
}
```

# Chapitre 9

## Tableaux & Chaînes de caractères

### 9.1 Tableaux

#### 9.1.1 Définition

Un tableau est un ensemble d'éléments consécutifs. Celui-ci peut être constitué de plusieurs lignes et colonnes. Nous n'utiliserons dans un premier temps que les tableaux à une seule ligne.

*Exemple de tableau de caractères :*

|         |   |   |   |   |   |   |   |   |
|---------|---|---|---|---|---|---|---|---|
| Case    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Contenu | A | B | C | D | E | F | G | H |

Les cases d'un tableau sont numérotées à partir de 0.

#### 9.1.2 Déclaration

Un tableau se déclare de la manière suivante :

```
<type> <nom du tableau> [<taille du tableau>];
```

*Exemples :*

```
char tab_char [10];
/* Déclaration d'un tableau de 10 caractères */
int tab_int [10];
/* Déclaration d'un tableau de 10 nombres. */
```

Un tableau à plusieurs lignes se déclare de la façon suivante :

```
<type> <nom du tableau> [<taille 1ère dimension 1>][<taille 2nde dimension>];
```



*Exemple :*

```
int table [5] [5];
/* représente un tableau d'entiers de 5 lignes * 5
 colonnes.*/
```

### 9.1.3 Utilisation

On accède à un tableau en l'appelant par son nom et son numéro de case.

*Exemple :*

```
int tab_int[10]; /* tableau de 10 cases (0 à 9) d'entiers */
int tab_char[10]; /* tableau de 10 cases (0 à 9) de caractères */
tab_char [3] = 'C'; /* Accès à la case 3 (la quatrième) de tab_char */
tab_int [6] = 10; /* Accès à la case 6 (la septième) de tab_int */
```

*Attention :* Le compilateur ne vérifie pas que vous utilisez le tableau dans ses limites. Il vous est donc possible d'écrire à l'extérieur de votre tableau donc chez le voisin. C'est l'un des bugs le plus courant de l'informatique.

## 9.2 Chaînes de caractères

Les chaînes de caractères sont des tableaux de caractères suivis d'un 0 binaire (ne pas confondre avec le caractère O, nous parlons ici du code ASCII) qui est considéré lui aussi comme un caractère. Une chaîne s'écrit donc : chaîne + 0.

*Exemple :* « Eric » s'écrit dans un tableau de 5 caractères de la façon suivante :

|            |    |     |     |    |   |
|------------|----|-----|-----|----|---|
| Caractère  | E  | r   | i   | c  |   |
| Code ASCII | 69 | 114 | 105 | 99 | 0 |
| Case       | 0  | 1   | 2   | 3  | 4 |

### 9.2.1 Déclaration d'une chaîne de caractères

Une chaîne de caractères se déclare sous la forme d'un tableau de caractères de longueur fixe. Attention, comme signalé auparavant, si vous dépassez la longueur de tableau, vous écrivez chez le copain.

*Exemple :*

```
char m_chaine [20];
permettra d'enregistrer des chaînes de 19 caractères maximum (20-1 pour le 0 de fin de chaîne).
```

D'autre part, il est possible de déclarer une chaîne de caractères sans en spécifier la longueur de départ de la façon suivante :

```
char chaine [] = "Eric";
```

C'est pratique à condition de ne pas la réutiliser (on ne connaît sa taille que par son contenu).

## 9.2.2 Écriture dans une chaîne de caractères

La première méthode de déclaration ne permettant pas d'initialiser la chaîne, on pourra lui affecter une valeur de la façon suivante :

```
sprintf (<variable de type chaîne de caractères>, "%s", "<valeur d'init.>");
```

*Exemple :*

```
sprintf (m_chaine, "%s", "Eric");
```

remplira `m_chaine` par `Eric0` (le 0 n'est là que pour vous rappeler que l'on a besoin du 0 pour terminer la chaîne).

## 9.2.3 Affichage d'une chaîne de caractères

Une chaîne de caractères s'affiche grâce à la commande `printf` et le format `%s`.

*Exemple :*

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

## 9.2.4 Longueur d'un chaîne

La longueur d'une chaîne de caractères s'obtient par la fonction `strlen`. Le 0 de fin de chaîne n'est pas compté dans cette longueur.

*Exemple :*

```
char ch [] = "toto" ;
printf ("La longueur de %s est : %d", ch, strlen (ch)) ;
```

Affichera : « La longueur de toto est 4 » à l'écran.

## 9.2.5 Exercices

En utilisant une boucle (`for`) :

- remplissez un tableau de 10 caractères avec les lettres de l'alphabet en commençant par A (code ASCII 65);
- faites afficher la chaîne de caractères ainsi obtenue (n'oubliez pas de rajouter le 0);
- faites afficher chaque caractère du tableau sous la forme « Caractère n° 0 : A ».

*Rappel :* Je peux écrire `Tab_car [i] = code_ascii`; où `code_ascii` est un entier représentant le code ASCII du caractère désigné.

*Aide :* Pour faire afficher un seul caractère, on utilisera la syntaxe suivante :

```
int pos /* Position dans le tableau */
printf ("Caractère n° %d : %c", pos, Tab_car [pos]);
```

La valeur 0 peut être assignée à un élément de tableau de la façon suivante :

```
Tab_car [la bonne position] = 0;
```

## 9.2.6 gets : saisie d'une chaîne de caractères

*Description* La fonction `gets` permet de saisir une chaîne de caractère validée par un retour chariot. Attention, bien que cette chaîne nécessite d'être validée par un retour chariot, celui-ci n'est pas enregistré dans le tableau de caractères.

*Exemple d'utilisation*

programme50.c

```
#include <stdio.h>
int main(void)
{
 char line[81];
 /* 81 : taille arbitraire supposée suffisante */
 /* Une ligne écran = 80 caractères + 1 case pour le 0 de fin de chaîne */
 printf("Saisissez une chaîne de caractère :\n");
 gets(line);
 /* La frappe de l'utilisateur sera enregistrée dans line,
 on suppose qu'il ne frappera pas plus de 80 caractères,
 sinon aïe aïe aïe */
 printf("\nLa chaîne de caractères saisie est : %s\n", line);
 printf("Notons qu'il n'y a qu'un retour chariot.\n");
 return (0);
}
```

*Exemple d'exécution :*

```
Saisissez une chaîne de caractère :
Bonjour !

La chaîne de caractère saisie était :
Bonjour !
```

Notons qu'il n'y a qu'un retour chariot (celui affiché par la fonction `printf`).

### 9.2.7 Passage d'une chaîne de caractères en paramètres

Il est bien agréable de pouvoir utiliser cette fonction mais son efficacité est très limitée. Nous allons donc créer notre propre fonction de saisie. Pour cela, il est nécessaire de passer en paramètre une chaîne de caractères. Ceci s'effectue de la façon suivante :

```
int ma_saisie (char* chaine)
{
 ...Faire ce qu'il faut...
 return (0);
}
void main ()
{
 char ma_chaine [30];
 ma_saisie (ma_chaine);
 return ();
}
```

On n'utilise pas ici le passage par adresse (&) car un tableau est en fait une adresse mémoire sur une suite d'octets. Par contre, on pourrait écrire :

```
ma_saisie (&chaine [0])
```

&chaine[0] représentant l'adresse mémoire de la première case mémoire du tableau. C'est un peu compliqué mais il suffit de se dire : « je passe un tableau donc un ensemble de cases mémoire, donc une adresse, donc je ne mets pas de & » ou bien « je transmets une case donc un élément d'un type comme un autre donc je mets le & pour obtenir son adresse ».

### 9.2.8 Bugs en chaînes

Reprenez l'exercice 9.2.5 et mettez 12 caractères dans votre tableau de 10 cases... Essayez avec 80 caractères.

On peut sans se tromper dire que le cas de dépassement de la taille d'un tableau est le bug le plus fréquent rencontré en programmation C alors prenez garde...

*Note :* Ce problème n'existe pas sous Turbo Pascal, mais on ne peut pas concilier sécurité et rapidité, ce qui fait que le C est de loin plus rapide que Turbo Pascal. Il suffit de s'imaginer qu'il est nécessaire d'effectuer un test à chaque assignation d'un élément du tableau !!!.

### 9.2.9 Exercices

Réaliser votre propre fonction de saisie de chaîne de caractères.

*Paramètres d'entrée :* chaîne de caractères, nombre de caractères maximum à saisir

*Algorithme :*

Initialiser car à une valeur différente de 10 (valeur du retour-chariot)

**tant que** ((car est différent d'un retour chariot (10)) **et** (nombre de caractères saisis < nombre de caractères maximum)) **faire**

    car = saisie\_car

    afficher car à l'écran

    ajouter car dans le tableau

    mettre à jour la positions dans le tableau

**fin tant que**

ajouter le 0 binaire de fin de chaîne.

*Aide :* Pour afficher un caractère à l'écran instantanément, sans envoyer de retour chariot, il faut utiliser `fflush(stdout)`, qui vide le buffer de la sortie standard :

– `printf("a\n")` ; affiche un a puis un retour chariot ;

– `printf("a")` ; affichera un a plus tard, lors de la réception d'un retour chariot ;

– `printf("a"); fflush(stdout)` affiche un a instantanément, sans retour chariot.

Ajouter à votre fonction de saisie de caractères la gestion de la touches `backspace` permettant de corriger la saisie, ainsi que la gestion du retour chariot final (qui doit être supprimé).

*Aide :* Lors de l'appui sur la touche `backspace`, c'est le le caractère 127 est renvoyé. Pour afficher un caractère `backspace`, on pourra faire :

```
printf("\b"); /* Retour du curseur vers la gauche */
fflush(stdout); /* force l'exécution de l'affichage immédiatement */
```

## 9.3 Correction des exercices

### 9.3.1 Boucle for

En utilisant une boucle (`for`), remplissez un tableau de 10 caractères avec les lettres de l'alphabet en commençant par A (code ASCII 65). Faites afficher la chaîne de caractères ainsi obtenue (n'oubliez pas de rajouter le 0). Faites afficher chaque caractère du tableau sous la forme "Caractère n° 0 : A".

programme51.c

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

int main ()
{
 /* 10 caractères
 + 0 de fin de chaîne */
 char tableau [11];
 int i=0; /* compteur */

 /* Remplissage du tableau
 avec les caractères */
 for (i=0; i<10; i++)
 tableau [i] = 65 + i;

 /* Ajout du 0 de fin de chaine */
 tableau [10] = 0;
```

```
/* Affichage de la chaîne */
printf ("Tableau : %s\n",tableau);

/* Saut d'une autre ligne */
printf ("\n");

/* Affichage de chacun des caractères */
for (i=0; i<10; i++)
{
 printf ("Caractère n°%d
 : %c\n",i,tableau [i]);
}

/* Attente de la saisie d'une touche */
getch ();
return (0);
}
```

### 9.3.2 Fonction de saisie (1ère partie)

Réaliser votre propre fonction de saisie de chaîne de caractère.

programme52.c

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

void saisie (char *chaine, int max_car)
{
 char car = ' ';
 int nb_car = 0;

 while ((car != 10) && (nb_car < max_car))
 {
 /* Saisie d'un nouveau caractère */
 car = getch();
 /* Ajout du caractère dans le tableau */
 chaine [nb_car] = car;
 printf ("%c", car);
 fflush(stdout);
 /* Mise à jour de la position dans le tableau */
 nb_car ++;
 }
 chaine [nb_car] = 0;
}

int main ()
{
 char chaine [11];
 char message [] = "Votre saisie : ";

 printf ("%s\n", message);

 /* 10 : Laisser de la place pour le 0 binaire de fin de chaîne */
 saisie (chaine,10);

 /* Affichage du résultat */
 printf ("Votre saisie : %s\n",chaine);
 /* Attente pour visualisation */
 getch ();
 return(0);
}
```

### 9.3.3 Fonction de saisie (2ère partie)

Réaliser votre propre fonction de saisie de chaîne de caractère, en gérant les pressions sur la touche backspace et le retour chariot final.

programme53.c

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

void saisie (char *chaine, int max_car)
{
 char car = ' ';
 int nb_car = 0;

 while ((car != 10) && (nb_car < max_car))
 {
 car = getch(); /* Saisie d'un caractère */
 /* Si la touche backspace a été pressée ... */
 if (car==127)
 {
 printf("\b");
 /* Mise à jour de la position dans le tableau */
 nb_car--;
 }
 else
 {
 /* Ajout du caractère dans le tableau */
 chaine [nb_car] = car;
 printf("%c", car);
 /* Mise à jour de la position dans le tableau */
 nb_car ++;
 }
 fflush(stdout);
 }
 if (chaine[nb_car-1]==10) nb_car--;
 else printf("\n");
 chaine [nb_car] = 0;
}

int main ()
{
 char chaine [11];
 char message [] = "Votre saisie : ";
 printf ("%s\n", message);
 /* 10 : Laisser de la place pour le 0 binaire de fin de chaîne */
 saisie (chaine,10);
 printf ("Votre saisie : %s\n",chaine); /* Affichage du résultat */
 getch (); /* Attente pour visualisation */
 return(0);
}
```



# Chapitre 10

## Fichiers et Structures

Dans ce chapitre qui ne comporte pour ainsi dire pas d'exercices, nous allons survoler des notions difficiles. Prenez le temps de bien comprendre les exemples et n'hésitez pas à poser des questions ;-).

Pour une étude plus prononcée des fichiers, il est nécessaire d'avoir une connaissance minimale du système Linux ou d'Unix.

Pour l'assimilation de la notion de structure, l'expérience sera le maître mot.

### 10.1 Bases sur les fichiers

Un fichier représente tout ce qui est enregistré sur votre disque dur ou presque, on va dire tout ce qui porte un nom. Il est possible de créer, de lire ou d'écrire dans des fichiers. A noter, que certains fichiers par contre peuvent être protégés en lecture, en écriture ou les deux.

Afin de ne rien endommager, créez à l'aide de votre éditeur de texte favori un fichier nommé `test.txt` dans lequel vous taperez un texte de louanges pour ce superbe cours (à votre libre inspiration).

Voici un petit exemple de la lecture du fichier `test.txt`. A noter que l'utilisation des fichiers nécessite la bibliothèque `unistd.h`.

Exemple :

programme54.c

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main ()
{
 int fd = 0;
 char ligne [81];
 int nb_car_lus = 0;

 /* Ouverture du fichier */
 fd = open ("test.txt", O_RDONLY);

 /* Test si fichier ouvert */
 if (fd < 0)
 {
 printf ("Impossible d'ouvrir le fichier\n");
 getchar ();
 return (-fd);
 }

 do
 {
 /* Lecture de 80 octets maximum */
 nb_car_lus = read (fd, ligne, 80);

 if (nb_car_lus > 0)
 {
 /* Fin de chaîne de caractère */
 ligne [nb_car_lus] = 0;

 /* Ecriture de ce qui a été lu */
 printf ("%s", ligne);
 }
 }
 while (nb_car_lus > 0);

 /* Fermeture du fichier */
 close (fd);

 /* Ecrire que c'est terminé */
 printf ("\n --- Fin ---\n");

 getchar ();
 return (0);
}
```

Analysons l'ensemble

```
/* Ouverture du fichier */
fd = open ("test.txt", O_RDONLY);
```

La fonction `open` permet d'ouvrir un fichier, le second paramètre est son mode d'ouverture. `O_RDONLY` signifie ouverture en lecture seule. La fonction `open` renvoie un nombre négatif si l'ouverture du fichier est un échec et un handle (poignée) sur le fichier dans le cas contraire (positif).

*What is an handle ?* Un lien sur quelque chose. On appelle ceci une poignée car comme avec un sac, vous le prenez par la poignée pour le porter.

```
/* Test si fichier ouvert */
if (fd < 0)
{
 printf ("Impossible d'ouvrir le fichier\n");
 getchar ();
 return (-fd);
}
```

La fonction `read` permet de lire des octets (ici transformés en caractères (`char`)). Elle prend pour paramètre le handle de fichier, la zone de mémoire dans laquelle on va recevoir les données et la taille maximale que l'on veut lire (attention au dépassement (cf. chapitres précédents)). Elle renvoie un nombre négatif lorsqu'elle ne peut plus lire d'octets dans le fichier, dans le cas contraire, elle retourne le nombre de caractères lus.

```
/* Lecture de 80 octets maximum */
nb_car_lus = read (fd, ligne, 80);

if (nb_car_lus > 0)
...

```

On affiche alors les données saisies en ajoutant un 0 de fin de chaîne de caractères qui explique `char` chaîne [81].

A ne pas oublier, il faut fermer le fichier après l'avoir ouvert !

```
/* Fermeture du fichier */
close (fd);

/* Ecrire que c'est terminé */
printf ("\n --- Fin ---\n");

getchar ();
return (0);
```

Très simple non !

## 10.2 Création d'un autre fichier

Reprenez l'exemple précédent et modifiez-le avec les éléments qui suivent :

programme55.c

```
int fd2;

/* Création d'une copie */
fd2 = open ("copie.txt", O_CREAT | O_RDWR, 0777);

/* Test si fichier bien créé */
if (fd2 < 0)
{
 printf ("Impossible de créer le nouveau fichier");
 getchar ();
 return (-fd2);
}

do
{
 ... / ...

 /* Ecriture dans notre copie de fichier */
 write (fd2, ligne, nb_car_lus);
}

/* Fermeture de ma copie de fichier */
close (fd2);
```

A vous de voir comment ajouter ces données dans le programme précédent.

Une petite modification sur la fonction `open`, ici nous créons un fichier, lors de la création ce fichier possède des droits qui sont donnés par `0777` (se référer à un cours de Linux pour le détail des droits). De plus, il nous faut pouvoir écrire dans ce fichier, nous ajoutons donc `O_RDWR` à l'option de création (`O_CREAT`) afin d'ouvrir le fichier en mode Lecture/Ecriture (Reading/Writing). `O_CREAT` signifie ouverture en lecture si le fichier existe ou création dans le cas contraire (s'il n'existe pas).

La fonction `write` permet d'écrire des octets sur le fichier. Les paramètres sont le handle de fichiers, les octets à écrire et la taille à écrire.

Mettez 80 au lieu de `nb_car_lus` au niveau du `write`. Regardez, avec un peu de chance, vous allez trouver des morceaux de texte non désirés, eh oui, personne ne vous a dit que votre chaîne de départ était vide !

## 10.3 Structures

Une structure peut s'identifier à une fiche de travail avec des zones à remplir. Ces structures sont très pratiques pour arranger les données.

### 10.3.1 Déclaration

2 choix s'offrent à nous.

*Solution 1 :*

```
struct
{
 /* Définition de la structure */
} nom de la variable qui aura comme forme cette structure;
```

*Solution 2 :*

```
typedef struct
{
 /* Définition de la structure */
} nom de la structure;

<nom de la structure> nom de la variable qui aura comme structure cette structure.
```

*Exemple :*

```
typedef struct
{
 char nom [40];
 char prenom [20];
 int age;
} elt_fiche_nom;

elt_fiche_nom une_fiche ;
elt_fiche_nom une_seconde_fiche ;
```

La macro `typedef` signifie définition d'un type, de ce fait là `elt_fiche_nom` se comporte comme un nouveau type de données (comme `int`, `char`, ...).

### 10.3.2 Utilisation

```
struct
{
 char nom [40];
 char prenom [20];
 int age;
} fiche_nom;

fiche_nom une_fiche;
fiche_nom une_seconde_fiche;

une_fiche.age = 20;
```

### 10.3.3 Taille d'une structure

La taille d'une structure ou d'un type se détermine par la fonction `sizeof`.

*Exemple :*

programme56.c

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

int main()
{
 typedef struct
 {
 char nom [40];
 char prenom [20];
 int age;
 } fiche;

 fiche ma_fiche;

 printf ("Taille de la structure : %d\n",sizeof (fiche));
 sprintf (ma_fiche.nom, "%s","BERTHOMIER");
 sprintf (ma_fiche.prenom, "%s","Eric");
 ma_fiche.age = 30;
 printf ("%s %s est âgé de %d",ma_fiche.prenom, ma_fiche.nom, ma_fiche.age);

 getchar ();
 return (0) ;
}
```

## 10.4 Fichiers et Structures

programme57.c

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

int main(int argc, char* argv[])
{
 typedef struct
 {
 char nom [40];
 char prenom [20];
 unsigned char age;
 } fiche;

 fiche ma_fiche;
 int h_fic ;
 int i;

 /* Création du du fichier -> le vider s'il existait */
 h_fic = open ("fiche.dat", O_CREAT | O_TRUNC | O_WRONLY, 0777);

 /* Test si fichier ouvert */
 if (h_fic < 0)
 {
 printf ("Impossible d'ouvrir le fichier");
 getchar ();
 return (-h_fic);
 }

 printf ("Taille de la structure : %d\n\n",sizeof (fiche));

 sprintf (ma_fiche.nom, "%s","BERTHOMIER");
 sprintf (ma_fiche.prenom, "%s","Eric");
 ma_fiche.age = 30;
 printf ("Fiche 1 : %s %s est âgé de %d\n",ma_fiche.prenom, ma_fiche.nom,
 ma_fiche.age);
 write (h_fic, &ma_fiche, sizeof (fiche));

 sprintf (ma_fiche.nom, "%s","P'TIT");
 sprintf (ma_fiche.prenom, "%s","Luc");
 ma_fiche.age = 48;
 printf ("Fiche 2 : %s %s est âgé de %d\n",ma_fiche.prenom, ma_fiche.nom,
 ma_fiche.age);
 write (h_fic, &ma_fiche, sizeof (fiche));
```

```

sprintf (ma_fiche.nom, "%s", "Tolkien");
sprintf (ma_fiche.prenom, "%s", "JRR");
ma_fiche.age = 89;
printf ("Fiche 3 : %s %s est âgé de %d\n", ma_fiche.prenom, ma_fiche.nom,
 ma_fiche.age);
write (h_fic, &ma_fiche, sizeof (fiche));

/* Fermeture du fichier */
close (h_fic);

/* ----- */
/* Lecture des éléments du fichier */
/* ----- */
/* Ouverture du fichier */
h_fic = open ("fiche.dat", O_RDONLY);

/* Test si fichier bien ouvert */
if (h_fic < 0)
{
 printf ("Impossible d'ouvrir le fichier");
 getchar ();
 return (-h_fic);
}

for (i=0; i<3; i++)
{
 /* Lecture de la fiche */
 read (h_fic, &ma_fiche, sizeof (fiche));

 /* Affichage des données lues */
 printf ("Fiche n° %d : %s %s est âgé de %d\n", i, ma_fiche.prenom,
 ma_fiche.nom, ma_fiche.age);
}

/* Fermeture du fichier */
close (h_fic);

getchar ();
return 0;
}

```

## 10.5 Fin et commencement

Vous disposez maintenant d'à peu près toutes les connaissances nécessaires pour l'élaboration de petits programmes.



## 10.6 Correction de l'exercice

L'intégralité du programme de référence pour la "Création d'un autre fichier" ne vous est pas donnée dans le cours, la voici donc :

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main ()
{
 int fd = 0, fd2 = 0, nb_car_lus = 0;
 char ligne [81];

 /* Ouverture du fichier */
 fd = open ("test.txt", O_RDONLY);

 if (fd < 0)
 {
 printf ("Impossible d'ouvrir le fichier\n");
 getchar ();
 return (-fd);
 }

 /* Création du fichier */
 fd2 = open ("copie.txt", O_CREAT | O_RDWR, 0777);

 if (fd2 < 0)
 {
 printf ("Impossible d'ouvrir le fichier\n");
 getchar ();
 return (-fd2);
 }
}
```

```

do
{
 /* Lecture de 80 octets maximum */
 nb_car_lus = read (fd, ligne, 80);

 if (nb_car_lus > 0)
 {
 /* Fin de chaîne de caractère */
 ligne [nb_car_lus] = 0;

 /* Ecriture de ce qui a été lu */
 printf ("%s", ligne);

 /* Ecriture dans le second fichier */
 write (fd2, ligne, nb_car_lus);
 }
}
while (nb_car_lus > 0);

/* Fermeture du fichier */
close (fd);

/* Fermeture du second fichier */
close (fd2);

/* Ecrire que c'est terminé */
printf ("\n --- Fin ---\n");

getchar ();
return (0);
}

```

# Chapitre 11

## Curses

Ce cours est dédié à un joyeux luron nommé "Bernard" et mon oeil !

Ce cours a été réalisé grâce

- au livre *Programming with Curses* de John Strang Editions O'Reilly & Associates, Inc.
- au cours "The Linux Programmer's Guide" de Sven Godt, Sven van der Meer, Scott Burtkett, Matt Welsh.

### 11.1 Concepts de base

Curses est une bibliothèque qui offre des fonctionnalités de manipulation de l'écran. Il faut savoir que chaque console possède des caractéristiques qui lui sont propres (taille d'écran, vitesse d'affichage ...), de ce fait il est très difficile de réaliser des applications "portables" (qui vont d'un système à un autre). Curses nous fait le travail en nous proposant un large éventail de fonctions qui nous permettent de gérer l'écran. Pour plus de détails sur ces fonctions, une documentation très fournie et très complète est disponible sur le Web.

Le principe de fonctionnement de Curses est un tableau de nLignes et nColonnes contenant d'une part tous les caractères de fenêtre de traitement mais aussi toutes les caractéristiques de ces caractères. Tous ses renseignements sont enregistrés dans une structure nommée WINDOW. Afin d'optimiser au maximum les programmes tous les affichages sont bufferisés. Un changement ne sera actif que lorsque la commande `refresh ()` est appelée. En attendant que cette commande soit appelée, les changements sont opérés dans l'image mémoire de la fenêtre (WINDOW). On obtient donc le principe de fonctionnement suivant :

```
Initialisation de la fenêtre {\tt initscr ()}

Modification de la fenêtre
Affichage des modifications {\tt refresh ()}

Restauration de la fenêtre {\tt endwin ()}
```

`initscr ()` initialise Curses, il renseigne le type de terminal et les caractéristiques qui lui sont associées.

`endwin ()` restaure le terminal dans son état initial. Ceci permet de mettre fin à toutes les modifications opérées par Curses lors de son fonctionnement.

`refresh ()` affiche les modifications opérées dans l'écran. Cette commande est facultative lorsqu'une demande de saisie est effectuée (`scanw` par exemple).

Tout ceci est bien beau mais pour l'instant, nous n'avons rien vu et surtout rien fait. Avant de commencer à pianoter sur son joli clavier, il est nécessaire de savoir comment inclure la bibliothèque curses.

1. Ajoutez `#include <curses.h>`
2. Lors de la compilation, ajoutez `-lcurses`

Je ne détaille pas plus, reportez vous aux cours précédents si vous avez un trou de mémoire &-). Bon maintenant passons en revue quelques fonctions

## 11.2 Quelques fonctions d'affichage de Curses

### 11.2.1 Localisation du curseur

`move (y, x)` permet de positionner le curseur sur l'écran. Bien entendu, ce positionnement ne sera effectif qu'après l'appel de la fonction `refresh`. Le coin haut gauche de la fenêtre possède les coordonnées (0,0).

### 11.2.2 Affichage de caractères

`addch (char ch)` affiche un caractère à la position courante du curseur. Certains caractères spéciaux donnent les résultats suivants :

|                 |                |                                                                                  |
|-----------------|----------------|----------------------------------------------------------------------------------|
| <code>\t</code> | Tabulation     | 8 caractères blancs                                                              |
| <code>\r</code> | Retour-Chariot | Déplace le curseur sur la prochaine ligne                                        |
| <code>\b</code> | Backspace      | Écrit un blanc en arrière                                                        |
| <code>\n</code> |                | Efface le reste de la ligne et déplace le curseur au début de la prochaine ligne |

Beaucoup d'autres caractères permettent aussi l'affichage de caractères spéciaux semi-graphique, nous en verrons un peu plus tard.

### 11.2.3 Effacement de la fenêtre

`clear ()` efface la fenêtre en la remplissant de blancs.

## 11.2.4 Affichage d'une chaîne de caractères

`addstr (char* ch)` affiche une chaîne de caractères. Il est important de faire attention à ce que la chaîne ne dépasse pas la fin de la ligne, enfin, à vous de voir et de tester ...

## 11.2.5 Affichage formaté d'une chaîne de caractères

`printw (...)` est l'adaptation de `printf` pour l'affichage formaté de chaîne de caractères.

*Exemple :*

```
printw ("%s", chaine);
```

## 11.2.6 Boîtes

Tout est plus beau dans une boîte, boîte de dialogue, boîte de saisie ..., pour se faire `Curses` nous offre la fonction `box (WINDOW* win, char vert, char horz)`.

Dans le cadre de ce cours, nous n'utiliserons pas les possibilités de fenêtres multiples, de ce fait, `win` sera toujours `stdscr` traduisant "standard screen". `vert` définit le caractère qui servira de bord vertical, `horz` définit le caractère qui servira de bord horizontal.

*Exemple :*

```
!=====!
!box (stdscr, '|', '=');!
!=====!
```

## 11.2.7 Les anciennes fonctions d'affichage

Les anciennes fonctions d'affichage sont tout simplement proscrites. Cette recommandation s'applique aussi aux fonctions de saisie que nous verrons par la suite.

## 11.3 1<sup>re</sup> utilisation

Voici un programme exemple qui résume ce que l'on vient de voir ...

programme58.c

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

int main ()
{
 initscr (); // Initialisation de Curses

 box (stdscr, '|', '-'); // Dessin d'une boîte

 move (3,2); // Positionnement en 3,2

 addstr ("Quel superbe bibliothèque !"); // Affichage d'un message

 refresh (); // Application des changements

 getch (); // Attente, pour que l'on puisse voir !

 endwin (); // Fin de Curses

 return (0);
}
```

*Compilation :*

```
gcc -c -Wall programme58.c
gcc -o programme58 programme58.o -lcurses
```

## 11.4 Quelques fonctions de saisie de Curses

### 11.4.1 Saisie d'un caractère

`getch` permet de saisir un caractère.

programme59.c

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

int main ()
{
 char car;

 initscr (); // Initialisation de Curses

 box (stdscr, '|', '-'); // Dessin d'une boîte

 move (3,2); // Positionnement en 3,2

 addstr ("Votre caractère : "); // Demande la saisie d'un caractère

 car = getch (); // Saisie du caractère

 move (4,2); // Positionnement en 4,2

 printw ("Caractère : %c", car); // Affiche le caractère saisi

 refresh (); // Affichage des modifications

 getch (); // Attente pour voir le résultat

 endwin (); // Fin de Curses

 return (0);
}
```

## 11.4.2 Saisie d'une chaîne de caractère

`getstr (char* str)` permet de saisir une chaîne de caractères. Au vue que l'on ne connaît jamais la taille de ce qui va être saisi par l'utilisateur, cette fonction demeure dangereuse (risque de dépassement de tableau).

programme60.c

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

int main ()
{
 char chaine [256];

 initscr (); // Initialisation de Curses

 box (stdscr, '|', '-'); // Dessin d'une boîte

 move (3,2); // Positionnement en 3,2

 addstr ("Votre chaîne de caractère : "); // Demande la saisie d'un caractère

 getstr (chaine); // Saisie de la chaîne

 move (4,2); // Positionnement en 4,2

 printw ("Chaine : %s", chaine); // Affiche le caractère saisi

 refresh (); // Affichage des modifications

 getch (); // Attente pour voir le résultat

 endwin (); // Fin de Curses

 return (0);
}
```



### 11.4.3 Saisie formatée

`scanw (...)` est l'équivalent de `scanf` pour Curses.

programme61.c

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

int main ()
{
 int nbre = 0;

 initscr (); // Initialisation de Curses

 box (stdscr, '|', '-'); // Dessin d'une boîte

 move (3,2); // Positionnement en 3,2

 addstr ("Votre nombre : "); // Demande la saisie d'un caractère

 scanw ("%d",&nbre); // Saisie du nombre

 move (4,2); // Positionnement en 4,2

 printw ("Nombre : %d", nbre); // Affiche le nombre saisi

 refresh (); // Affichage des modifications

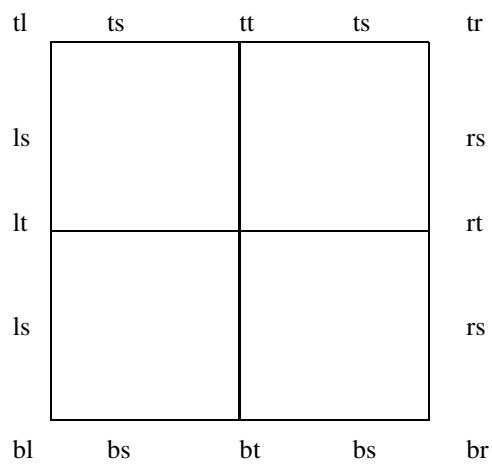
 getch (); // Attente pour voir le résultat

 endwin (); // Fin de Curses

 return (0);
}
```

## 11.5 Affichage de caractères spéciaux

Voici quelques caractères spéciaux que permet d'afficher curses.



| Caractère | Position     | Valeur       |
|-----------|--------------|--------------|
| tl        | top left     | ACS_ULCORNER |
| ts        | top side     | ACS_HLINE    |
| tr        | top right    | ACS_URCORNER |
| ls        | left side    | ACS_VLINE    |
| rs        | right side   | ACS_VLINE    |
| bl        | bottom left  | ACS_LLCORNER |
| bs        | bottom side  | ACS_HLINE    |
| br        | bottom right | ACS_LRCORNER |
| rt        | right tee    | ACS_RTEE     |
| lt        | left tee     | ACS_LTEE     |
| tt        | top tee      | ACS_TTEE     |
| bt        | bottom tee   | ACS_BTEE     |

Exemple :

programme62.c

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

int main ()
{
 initscr (); // Initialisation de Curses

 box (stdscr, ACS_VLINE, ACS_HLINE); // Dessin d'une boîte

 refresh (); // Affichage des modifications

 getch (); // Attente pour voir le résultat

 endwin (); // Fin de Curses

 return (0);
}
```

## 11.6 Les couleurs

Les couleurs se définissent par paires, une couleur de fond et une couleur de premier plan. Le nombre de paires de couleurs maximal est `COLOR_PAIRS - 1`, la paire numéro 0 étant réservée pour le blanc sur fond noir).

### 11.6.1 Dénomination des couleurs :

| Couleur | Définition                 |
|---------|----------------------------|
| Noir    | <code>COLOR_BLACK</code>   |
| Rouge   | <code>COLOR_RED</code>     |
| Vert    | <code>COLOR_GREEN</code>   |
| Jaune   | <code>COLOR_YELLOW</code>  |
| Bleu    | <code>COLOR_BLUE</code>    |
| Magenta | <code>COLOR_MAGENTA</code> |
| Cyan    | <code>COLOR_CYAN</code>    |
| Blanc   | <code>COLOR_WHITE</code>   |

### 11.6.2 Fonctions liées au couleurs :

- `has_colors ()` renvoie `TRUE` si le terminal supporte les couleurs.
- `can_change_color ()` renvoie `TRUE` si le terminal permet de redéfinir les couleurs.
- `start_color ()` initialise les couleurs. Cette fonction doit être appelée avant l'utilisation de couleurs.
- `init_pair (pair, fg, bg)` permet de définir la paire de couleur `pair` sauf 0.

- `wattrset (win, COLOR_PAIR (pair))` permet d'utiliser la paire de couleur `pair` comme couleur d'écriture. `win` est la fenêtre concernée, dans notre cas, ce sera toujours la fenêtre standard i.e. `stdscr`.

programme63.c

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

int main ()
{
 initscr (); // Initialisation de Curses

 start_color ();

 box (stdscr, ACS_VLINE, ACS_HLINE); // Dessin d'une boîte

 move (3,3);

 addstr ("Coucou c'est moi");

 move (4,3);

 printw ("Nombre de paires : %d", COLOR_PAIRS);

 init_pair (1, COLOR_BLUE, COLOR_WHITE);

 wattrset (stdscr, COLOR_PAIR(1));

 move (5,3);

 addstr ("Coucou c'est moi");

 refresh (); // Affichage des modifications

 getch (); // Attente pour voir le résultat

 endwin (); // Fin de Curses

 return (0);
}
```

## Chapitre 12

# Le Jeu des Allumettes

Adaptée du livre de Jacques Deconchat : "102 Programmes Pour MO6, TO8 et TO9+"

### 12.1 Enoncé de l'exercice

Un certain nombre d'allumettes est disposé entre les deux partis, l'ordinateur et vous. Le but du jeu est de ne pas retirer la dernière allumette. Pour se faire, une prise maximale est désignée par le joueur.

En début de partie, on demande :

1. Le nombre d'allumettes disposées entre les deux joueur (de 10 à 60).
2. Le nombre maximal d'allumettes que l'on peut retirer.
3. Qui commence (0 pour le joueur et 1 pour l'ordinateur).

Puis, tour à tour, chaque parti donne le nombre d'allumettes qu'il prend.

La partie se termine lorsqu'il n'y a plus d'allumettes sur la table. La personne ayant tirée la dernière allumette est le perdant, l'autre le vainqueur.

### 12.2 Ce qu'il faut faire

Lisez l'intégralité de l'énoncé !-).

Dans un premier temps, essayer de réaliser l'exercice sans faire d'affichage graphique, i.e. sans utiliser Curses.

Dans un second temps, adaptez votre programme.

Pour se faire, écrivez une fonction qui dessine une allumette et une qui, utilisant le dessin d'une allumette, dessine le jeu dans son intégralité (toutes les allumettes).

## 12.3 Aide

L'ordinateur répond grâce à la fonction décrite ci-dessous :

```
int jeu_ordi (int nb_allum, int prise_max)
{
 int prise = 0;
 int s = 0;
 float t = 0;

 s = prise_max + 1;
 t = ((float) (nb_allum - s)) / (prise_max + 1);

 while (t != floor (t))
 {
 s--;
 t = ((float) (nb_allum-s)) / (prise_max + 1);
 }

 prise = s - 1;
 if (prise == 0)
 prise = 1;

 return (prise);
}
```

La fonction `floor` donne l'arrondi inférieur d'un nombre.

`Nb_allum` est le nombre d'allumettes sur la table (au moment du coup à jouer).

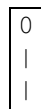
`Prise_max` est le nombre maximum d'allumettes autorisé lors d'une prise.

La fonction retourne en sortie le nombre d'allumettes prises par l'ordinateur.

*Exemple :*

```
prise = jeu_ordi (10,3) ;
```

Une allumette sera dessinée de la façon suivante :



- La fonction `floor` nécessite la bibliothèque `<math.h>`

- L'utilisation de la bibliothèque `<math.h>` oblige à linker avec `-lm: gcc -o monprog monprog.o -lm`

## 12.4 Corrections de l'exercice sans Curses

Ces programmes ne sont pas optimisés ni très beaux. Ils se veulent simplement clairs pour les débutants que vous êtes.

programme64.c

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

int jeu_ordi (int nb_allum, int prise_max)
{
 int prise = 0;
 int s = 0;
 float t = 0;

 s = prise_max + 1;
 t = ((float) (nb_allum - s)) / (prise_max + 1);

 while (t != floor (t))
 {
 s--;
 t = ((float) (nb_allum-s)) / (prise_max + 1);
 }

 prise = s - 1;

 if (prise == 0)
 prise = 1;

 return (prise);
}

int main ()
{
 int nb_max_d=0; /*nombre d'allumettes maxi au départ*/
 int nb_allu_max=0; /*nombre d'allumettes maxi que l'on peut tirer au maxi*/
 int qui=0; /*qui joue? 0=Nous --- 1=PC*/
 int prise=0; /*nombre d'allumettes prises par le joueur*/
 int nb_allu_rest=0; /*nombre d'allumettes restantes*/

 printf ("Nombre d'allumettes disposées entre les deux joueurs (entre 10 et 60) :
 scanf ("%d",&nb_max_d);
```

```

do
{
 printf ("\nNombre maximal d'allumettes que l'on peut retirer : ");
 scanf ("%d",&nb_allu_max);

 if (nb_allu_max >= nb_max_d)
 printf ("Erreur !");
}
while (nb_allu_max >= nb_max_d);
/* On répète la demande de prise tant que le nombre donné n'est pas correct */

do
{
 printf ("\nQuel joueur commence? 0--> Joueur ; 1--> Ordinateur : ");
 scanf ("%d",&qui);

 if ((qui != 0) && (qui != 1))
 printf ("\nErreur");
}
while ((qui != 0) && (qui != 1));

nb_allu_rest = nb_max_d;

do
{
 printf ("\nNombre d'allumettes restantes : %d",nb_allu_rest);

 if (qui==0)
 {
 do
 {
 printf ("\nCombien d'allumettes souhaitez-vous piocher ? ");
 scanf ("%d",&prise);

 if ((prise > nb_allu_rest) || (prise > nb_allu_max))
 {
 printf ("Erreur !\n");
 }
 }
 while ((prise > nb_allu_rest) || (prise > nb_allu_max));
/* On répète la demande de prise tant que le nombre donné n'est pas correct */
 }
 else
 {
 prise = jeu_ordi (nb_allu_rest , nb_allu_max);
 printf ("\nPrise de l'ordi : %d\n",prise);
 }

 qui=!qui;
 nb_allu_rest= nb_allu_rest - prise;
}
while (nb_allu_rest >0);

```



```
if (qui == 0) /* C'est à nous de jouer */
 printf ("\nVous avez gagné!\n");
else
 printf ("\nVous avez perdu!\n");

return (0);
}
```

## 12.5 Corrections de l'exercice avec Curses

programme65.c

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

int jeu_ordi (int nb_allum, int prise_max)
{
 int prise = 0;
 int s = 0;
 float t = 0;

 s = prise_max + 1;
 t = ((float) (nb_allum - s)) / (prise_max + 1);

 while (t != floor (t))
 {
 s--;
 t = ((float) (nb_allum-s)) / (prise_max + 1);
 }

 prise = s - 1;
 if (prise == 0)
 prise = 1;

 return (prise);
}

void dessine_allumette (int emplacement)
{
 move (15, emplacement);
 wattrset (stdscr, COLOR_PAIR (1));
 addch ('*');

 // On repasse en couleur standard
 wattrset (stdscr, COLOR_PAIR (0));

 move (16, emplacement);
 addch (ACS_VLINE);

 move (17, emplacement);
 addch (ACS_VLINE);
}
```

```

void dessine_jeu (int nb_allumettes)
{
 int allu = 0;

 for (allu = 0; allu < nb_allumettes; allu ++)
 {
 dessine_allumette (5+allu);
 }

 refresh ();
}

int main ()
{
 int nb_max_d=0; /*nbre d'allumettes maxi au départ*/
 int nb_allu_max=0; /*nbre d'allumettes maxi que l'on peut tirer
au maxi*/
 int qui=0; /*qui joue? 0=Nous --- 1=PC*/
 int prise=0; /*nbre d'allumettes prises par le joueur*/
 int nb_allu_rest=0; /*nbre d'allumettes restantes*/

 initscr ();
 start_color ();

 // Définition de la couleur rouge sur fond noir * de l'allumette
 init_pair (1, COLOR_RED, COLOR_BLACK);

 box (stdscr, ACS_VLINE, ACS_HLINE);

 do
 {
 move (1,1);
 addstr ("Nombre d'allumettes disposées entre les deux joueurs (entre 10 et 60)");
 scanw ("%d",&nb_max_d);

 if ((nb_max_d > 60) || (nb_max_d < 10))
 {
 move (2,1);
 addstr ("Erreur !");
 refresh ();
 }
 }
 while ((nb_max_d < 10) || (nb_max_d > 60));
 /* On répète la demande du nombre d'allumettes */
 /* tant que le nombre donné n'est pas correct */
}

```

```

do
{
 move (2,1);
 addstr ("Nombre maximal d'allumettes que l'on peut retirer : ");
 scanw ("%d",&nb_allu_max);

 if (nb_allu_max >= nb_max_d)
 {
 move (3,1);
 addstr ("Erreur !");
 refresh ();
 }
}
while (nb_allu_max >= nb_max_d);
/* On répète la demande de prise */
/* tant que le nombre donné n'est pas correct */

do
{
 move (3,1);
 addstr ("Quel joueur commence? 0--> Joueur ; 1--> Ordinateur : ");
 scanw ("%d",&qui);

 if ((qui != 0) && (qui != 1))
 {
 move (4,1);
 addstr ("Erreur");
 refresh ();
 }
}
while ((qui != 0) && (qui != 1));

nb_allu_rest = nb_max_d;

do
{
 // Ce n'est pas beau mais c'est efficace
 clear ();
 box (stdscr, ACS_VLINE, ACS_HLINE);
 dessine_jeu (nb_allu_rest);
}

```

```

 if (qui==0)
 {
 do
 {
 move (2,2);
 addstr ("Combien d'allumettes souhaitez-vous piocher ? ");
 scanw ("%d",&prise);

 if ((prise > nb_allu_rest) || (prise > nb_allu_max))
 {
 move (3,2);
 addstr ("Erreur !");
 refresh ();
 }
 }
 while ((prise > nb_allu_rest) || (prise > nb_allu_max));
/* On répète la demande de prise */
/* tant que le nombre donné n'est pas correct */
 }
 else
 prise = jeu_ordi (nb_allu_rest , nb_allu_max);

 qui=!qui;

 nb_allu_rest= nb_allu_rest - prise;
}
while (nb_allu_rest >0);

move (5,5);

if (qui == 0)
 addstr ("Vous avez gagné !");
else
 addstr ("Vous avez perdu !");

refresh ();

getchar ();

endwin ();

return (0);
}

```

# Chapitre 13

## Le Jeu de la Vie

Le présent exercice consiste à implémenter une version sur damier torique du *jeu de la vie* de John Conway, en C, et en utilisant la librairie *ncurses* pour l’affichage.

### 13.1 Historique

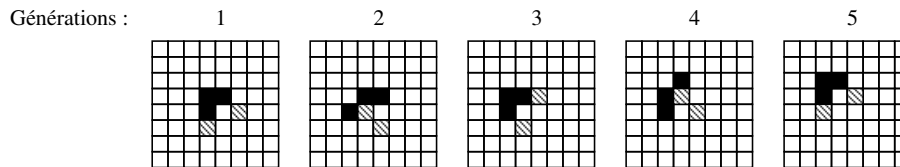
John Conway était un mathématicien de l’Université de Cambridge. Très prolifique en matière de jeux mathématiques, il décrivit en 1970 le *jeu de la vie*, visant à modéliser d’une façon simple l’évolution d’organismes vivants. Le jeu de la vie est une application particulière d’*automates cellulaires*.

### 13.2 Règles du jeu

Le *jeu de la vie* évolue normalement sur un damier infini. Chaque case est occupée par une cellule qui peut être vivante ou morte. À chaque génération, chaque cellule peut naître, mourir, ou rester dans son état. Les règles qui permettent de passer d’une génération à l’autre sont précises et ont été choisies avec soin pour que l’évolution des organismes soit intéressante et imprévisible. En premier lieu, notons que sur un damier infini, chaque case a exactement 8 voisins. Les règles données par J. Conway sont les suivantes :

- Une cellule ayant exactement 2 ou 3 voisins vivants survit à la génération suivante.
  - Une cellule ayant au moins 4 cellules voisines vivantes meurt d’étouffement à la génération suivante.
  - Une cellule ayant au plus une cellule voisine vivante meurt d’isolement à la génération suivante.
  - Sur une case vide ayant exactement 3 voisins vivants, une cellule naîtra à la génération suivante.
- Notons que c’est l’ensemble de la génération *actuelle* qui doit être pris en compte pour l’établissement de l’état des cellules à la génération *suivante*.

Voici un exemple de figure sur un petit damier. Les cellules qui devront mourir à la génération suivante sont grisées :

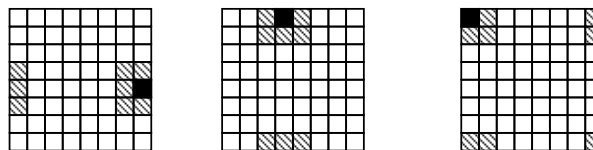


### 13.3 Damier torique

Représenter le damier par un tableau est la solution la plus simple. Mais on ne peut pas représenter un tableau infini. Plusieurs choix sont alors possibles :

- considérer l'absence de voisin (au bord du tableau) comme une cellule morte ;
- considérer l'absence de voisin (au bord du tableau) comme une cellule vivante ;
- jouer sur un damier torique.

La troisième solution, respecte mieux le jeu initial. On doit considérer que les bords droit et gauche du tableau sont reliés entre eux, ainsi que les bords supérieurs et inférieurs. Le voisinage des points est donc le suivant :



### 13.4 Implémentation du jeu

Pour réaliser l'affichage de notre *jeu de la vie*, nous utiliserons la librairie `ncurses`.

#### 13.4.1 Rappels utiles sur `ncurses`

On initialise la librairie proprement avec la fonction `initscr()` ;.

On la prépare à l'utilisation des couleurs, et on définit les couleurs ainsi :

```

if (has_colors())
{
 start_color();
 init_pair(1, COLOR_WHITE, COLOR_BLUE);
 init_pair(3, COLOR_RED, COLOR_BLACK);
}

```

Le bout de code précédent vérifie que le terminal supporte l'utilisation de couleurs, initialise leur utilisation et définit les couleurs 1 et 3 comme étant du blanc sur bleu et rouge sur noir.

On utilise les couleurs et on affiche un caractère à un endroit donné ainsi :

```
/* Sélection du blanc sur fond bleu */
wattrset(stdscr, COLOR_PAIR(1));
/* déplacement de la position courant ligne 10, colonne 14 */
move(10,14);
/* Affichage d'un espace (donc un carré bleu avec la déf. précédente) */
addch(' ');
```

Les commandes d'affichages terminées, un appel à la fonction `refresh()` ; permet d'obtenir l'affichage effectif.

Enfin, on quitte proprement la librairie par un appel à la fonction `endwin()`

### 13.4.2 Algorithme du jeu de la vie

On pourra reprendre l'algorithme suivant, qui décrit le programme globalement :

```
// Initialise la librairie
procedure init_curses()
// Termine proprement la librairie
procedure fin_curses()
// Initialise le damier avec des cellules vivantes prises au hasard
procedure damier_hasard()
// Renvoie le nombre de voisins de la cellule (x,y) du damier (utilisé
// par la procédure prochaine_generation)
fonction nombre_voisins(x,y)
// met à jour le damier pour la génération suivante
procedure prochaine_generation()
// Affiche le damier
procedure affiche(monde)

procedure principale :
 init_curses()
 damier_hasard()
 répéter :
 affiche_damier()
 prochaine_generation()
 saisir un caractère au clavier
 jusqu'à ce que le caractère saisi soit 'q'
 fin_curses
```



### 13.4.3 Structures de données utilisées

On pourra représenter le damier par un tableau de char. Un damier de taille 30 sera donc déclaré ainsi :

```
char monde[30][30];
```

On pourra choisir de mettre la valeur 0 dans une case si la cellule est « morte » et la valeur 1 dans le cas contraire.

Même si il est généralement conseillé d'éviter de déclarer des variables globales, on pourra dans cet exemple simple déclarer un tableau `monde` global.

### 13.4.4 Difficultés

Si vous vous sentez suffisamment en forme pour écrire le programme, ne vous en privez pas. Vous pourrez regarder cette section uniquement en cas de problème.

Les principales difficultés sont :

- le comptage du nombre de voisins sur damier torique ;
- le calcul de la génération suivante ;

La première difficulté peut être traitée ainsi :

```
// Renvoie le nombre de voisins vivants de la cellule (x,y)
// monde est le damier
// TAILLEX et TAILLEY les dimensions du damier
int nombre_voisins(int x,int y)
{
 int ofx,ofy;
 int x0,y0;
 int n=0;
 for (ofx=-1;ofx<=1;ofx++)
 {
 for (ofy=-1;ofy<=1;ofy++)
 {
 x0=x+ofx%TAILLEX;
 y0=y+ofy%TAILLEY;
 while (x0<0) x0+=TAILLEX;
 while (y0<0) y0+=TAILLEY;
 n+=monde[x0][y0];
 }
 }
 n-=monde[x][y];
 return n;
}
```

*Remarque :* En C l'opérateur `%` correspond à l'opération *modulo*. Ainsi, `a%b` est le reste de la division entière de `a` par `b`.

Pour calculer la génération suivante, on doit utiliser un tableau auxiliaire :

```

// Met à jour le damier avec la génération suivante
// monde est le damier
// TAILLEX et TAILLEY les dimensions du damier
void generation_suivante(void)
{
 int x,y;
 char monde_aux[TAILLEX][TAILLEY];
 // Calcul de la nouvelle génération dans le tableau auxiliaire
 for(x=0;x<TAILLEX;x++)
 {
 for(y=0;y<TAILLEY;y++)
 {
 switch(nombre_voisins(x,y))
 {
 case 2: // la cellule reste dans le même état
 monde_aux[x][y]=monde[x][y];
 break;
 case 3: // la cellule naît
 monde_aux[x][y]=(char)1;
 break;
 default: // la cellule meurt dans les autres cas
 monde_aux[x][y]=(char)0;
 break;
 }
 }
 }
 // Recopie du tableau auxiliaire dans le tableau courant
 for(x=0;x<TAILLEX;x++)
 {
 for(y=0;y<TAILLEY;y++)
 {
 monde[x][y]=monde_aux[x][y];
 }
 }
}

```

## 13.5 Possibilités d'amélioration

Si vous finissez rapidement, vous pouvez améliorer le programme.

### 13.5.1 Contrôle interactif

L'appel à la fonction `nodelay(stdscr, TRUE)` de `ncurses` permet d'utiliser la fonction `getch()` de façon non-bloquante. Un appel à la fonction renverra `ERR` si aucune touche n'a été tapée et le caractère entré dans le cas contraire.

En combinant ces deux fonctions, vous pouvez permettre à l'utilisateur de voir évoluer le jeu de la vie, sans qu'il ait à frapper une touche, tout en lui laissant les possibilités suivantes (par exemple) :

- taper 'q' pour sortir ;
- taper 'r' pour réinitialiser le jeu ;
- taper 'w' ou 'x' pour accélérer ou ralentir son déroulement.

### 13.5.2 Affichage des coordonnées

Vous pouvez afficher les coordonnées des cases sur les bords du damier.

### 13.5.3 Chargement de formes prédéfinies

Malgré son apparente simplicité, le jeu de la vie recèle de nombreuses surprises. En partant d'un tableau aléatoire, on peut difficilement les observer même si l'apparition d'un *planeur* (voir l'exemple du début) ou de formes périodiques (à la fin) ne sont pas rares.

Certaines personnes ont répertoriées des centaines des formes aux propriétés particulières :

- générateurs de planeurs (les lanceurs),
- portes logiques,
- formes périodiques de grande période,
- formes stables...

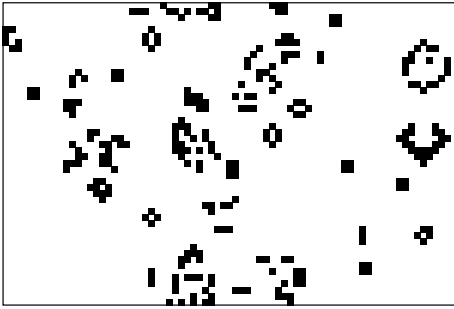
La possibilité d'initialiser le damier avec une forme particulière est donc intéressante pour pouvoir observer ces phénomènes. On pourra donc permettre à l'utilisateur de lire l'état des cases dans un fichier, afin qu'il choisisse l'état initial du damier.

## 13.6 Exemples obtenus

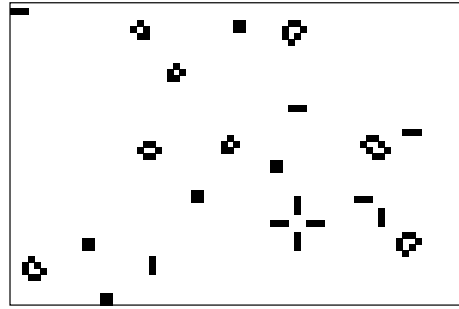
On peut voir sur les figures suivantes des configurations obtenues :



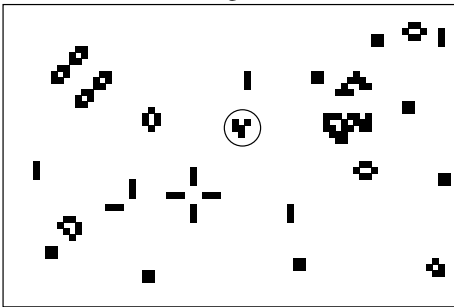
Configuration aléatoire de départ...



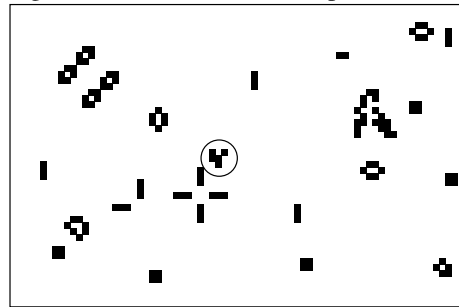
Formations d'amalgames de cellules...



Configuration finale de motifs de période 1 ou 2...



Apparition d'un *planeur*...



...qui se balade.

## 13.7 Exemple de corrigé

Le programme suivant affiche le damier, les coordonnées des cases, et permet à l'utilisateur d'utiliser les touches r, q, w et x pour réinitialiser le damier, quitter, accélérer ou ralentir le jeu.

programme66.c

```
#include <stdio.h>
#include <stdlib.h>
#include <ncurses.h>
#define TAILLEX 75
#define TAILLEY 50
char monde[TAILLEX][TAILLEY];
int usecolors=0;
/* Procédure de temporisation... */
void temporise (int t)
{
 int j,i;
 for (j=0;j<t;j++)
 for(i=0;i<32000;i++)
 { }
}
/* procédure vidant le buffer clavier */
void vide_buffer(void)
{
 while (getch()!=ERR);
}
/* Initialisation de ncurses et
définition des couleurs */
void init_curses(void)
{
 initscr();
 if (has_colors())
 {
 start_color();
 init_pair(1,COLOR_WHITE,COLOR_BLUE);
 init_pair(2,COLOR_WHITE,COLOR_GREEN);
 init_pair(3,COLOR_RED,COLOR_BLACK);
 usecolors=1;
 }
}
/* Terminaison propre de ncurses */
void fin_curses(void)
{
 endwin();
}

/* Affichage du damier */
void affiche(void)
{
 int i,j,k;
 char str[5];
 for (j=0;j<TAILLEY;j++)
 {
 for (i=0;i<TAILLEX;i++)
 {
 move(j,i);
 if (monde[i][j])
 {
 if (usecolors)
 {
 wattrset(stdscr,COLOR_PAIR(1));
 addch(' ');
 wattrset(stdscr,COLOR_PAIR(0));
 }
 else
 {
 addch('@');
 }
 }
 else
 {
 addch(' ');
 }
 }
 }
 /* Affichage des coordonnées
sur le bord droit */
 if (usecolors)
 wattrset(stdscr,COLOR_PAIR(3));
 for (j=0;j<TAILLEY;j++)
 {
 move(j,TAILLEX+1);
 printw("%03d",j%100);
 }
 /* Affichage des coordonnées
sur le bord inférieur */
 for (i=0;i<TAILLEX;i++)
 {
 sprintf(str,"%d",i);
 for (k=0;k<strlen(str);k++)
 {
 move(TAILLEY+k+1,i);
 addch(str[strlen(str)-1-k]);
 }
 }
 if (usecolors)
 wattrset(stdscr,COLOR_PAIR(0));
 refresh();
}
```

```

// Renvoie le nombre de voisins
// vivants de la cellule (x,y)
// monde est le damier
// TAILLEX et TAILLEY les dimensions
// du damier
int nombre_voisins(int x,int y)
{
 int ofx,ofy;
 int x0,y0;
 int n=0;
 for (ofx=-1;ofx<=1;ofx++)
 {
 for (ofy=-1;ofy<=1;ofy++)
 {
 x0=x+ofx*TAILLEX;
 y0=y+ofy*TAILLEY;
 while (x0<0) x0+=TAILLEX;
 while (y0<0) y0+=TAILLEY;
 n+=monde[x0][y0];
 }
 }
 n-=monde[x][y];
 return n;
}
/* Rempli au hasard le damier à
perc % de cellules vides */
void damier_hasard(int perc)
{
 int i,j;
 perc=perc%101;
 for (j=0;j<TAILLEY;j++)
 for (i=0;i<TAILLEX;i++)
 {
 if (rand()%101 > perc)
 monde[i][j]=1;
 else
 monde[i][j]=0;
 }
}

/* Calcule la génération suivante */
void generation_suivante(void)
{
 int x,y;
 char monde_aux[TAILLEX][TAILLEY];
 // Calcul de la nouvelle génération
 // dans le tableau auxiliaire
 for (x=0;x<TAILLEX;x++)
 {
 for (y=0;y<TAILLEY;y++)
 {
 switch(nombre_voisins(x,y))
 {
 // la cellule reste
 // dans le même état
 case 2:
 monde_aux[x][y]=monde[x][y];
 break;
 // la cellule naît
 case 3:
 monde_aux[x][y]=(char)1;
 break;
 // la cellule meurt
 // dans les autres cas
 default:
 monde_aux[x][y]=(char)0;
 break;
 }
 }
 }
 // Recopie du tableau auxiliaire
 // dans le tableau courant
 for (x=0;x<TAILLEX;x++)
 {
 for (y=0;y<TAILLEY;y++)
 {
 monde[x][y]=monde_aux[x][y];
 }
 }
}

```

```

int main(void)
{
 int temp=500;
 char c=' ';
 init_curses();
 noecho();
 nodelay(stdscr,TRUE);
 damier_hasard(50);
 move(54,0);
 addstr("q-Quit r-Redraw \
 w-Fast x-Slow");
 refresh();
 while(c!='q')
 {
 affiche();
 generation_suivante();
 c=getch();
 vide_buffer();
 switch(c)
 {
 case 'q':
 break;
 case 'r':
 damier_hasard(50);
 break;
 case 'w':
 temp-=10;
 if (temp<0) temp=0;
 break;
 case 'x':
 temp+=10;
 if (temp>10000) temp=10000;
 break;
 }
 temporise(temp);
 }
 fin_curses();
 return (0);
}

```

## **Annexe A**

### **Code Ascii**



| Ctl | Dec | Hex | Char | Code | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char           |
|-----|-----|-----|------|------|-----|-----|------|-----|-----|------|-----|-----|----------------|
| ^@  | 0   | 00  |      | NUL  | 32  | 20  | sp   | 64  | 40  | Q    | 96  | 60  | '              |
| ^A  | 1   | 01  | ␣    | SOH  | 33  | 21  | !    | 65  | 41  | A    | 97  | 61  | a              |
| ^B  | 2   | 02  | ␣    | SIX  | 34  | 22  | "    | 66  | 42  | B    | 98  | 62  | b              |
| ^C  | 3   | 03  | ␣    | EIX  | 35  | 23  | #    | 67  | 43  | C    | 99  | 63  | c              |
| ^D  | 4   | 04  | ␣    | EOI  | 36  | 24  | \$   | 68  | 44  | D    | 100 | 64  | d              |
| ^E  | 5   | 05  | ␣    | ENQ  | 37  | 25  | %    | 69  | 45  | E    | 101 | 65  | e              |
| ^F  | 6   | 06  | ␣    | ACK  | 38  | 26  | &    | 70  | 46  | F    | 102 | 66  | f              |
| ^G  | 7   | 07  | ␣    | BEL  | 39  | 27  | '    | 71  | 47  | G    | 103 | 67  | g              |
| ^H  | 8   | 08  | ␣    | BS   | 40  | 28  | (    | 72  | 48  | H    | 104 | 68  | h              |
| ^I  | 9   | 09  | ␣    | HI   | 41  | 29  | )    | 73  | 49  | I    | 105 | 69  | i              |
| ^J  | 10  | 0A  | ␣    | LF   | 42  | 2A  | *    | 74  | 4A  | J    | 106 | 6A  | j              |
| ^K  | 11  | 0B  | ␣    | VI   | 43  | 2B  | +    | 75  | 4B  | K    | 107 | 6B  | k              |
| ^L  | 12  | 0C  | ␣    | FF   | 44  | 2C  | ,    | 76  | 4C  | L    | 108 | 6C  | l              |
| ^M  | 13  | 0D  | ␣    | CR   | 45  | 2D  | -    | 77  | 4D  | M    | 109 | 6D  | m              |
| ^N  | 14  | 0E  | ␣    | SD   | 46  | 2E  | .    | 78  | 4E  | N    | 110 | 6E  | n              |
| ^O  | 15  | 0F  | ␣    | SI   | 47  | 2F  | /    | 79  | 4F  | O    | 111 | 6F  | o              |
| ^P  | 16  | 10  | ␣    | SLE  | 48  | 30  | 0    | 80  | 50  | P    | 112 | 70  | p              |
| ^Q  | 17  | 11  | ␣    | CS1  | 49  | 31  | 1    | 81  | 51  | Q    | 113 | 71  | q              |
| ^R  | 18  | 12  | ␣    | DC2  | 50  | 32  | 2    | 82  | 52  | R    | 114 | 72  | r              |
| ^S  | 19  | 13  | ␣    | DC3  | 51  | 33  | 3    | 83  | 53  | S    | 115 | 73  | s              |
| ^T  | 20  | 14  | ␣    | DC4  | 52  | 34  | 4    | 84  | 54  | T    | 116 | 74  | t              |
| ^U  | 21  | 15  | ␣    | NAK  | 53  | 35  | 5    | 85  | 55  | U    | 117 | 75  | u              |
| ^V  | 22  | 16  | ␣    | SYN  | 54  | 36  | 6    | 86  | 56  | V    | 118 | 76  | v              |
| ^W  | 23  | 17  | ␣    | EIB  | 55  | 37  | 7    | 87  | 57  | W    | 119 | 77  | w              |
| ^X  | 24  | 18  | ␣    | CAN  | 56  | 38  | 8    | 88  | 58  | X    | 120 | 78  | x              |
| ^Y  | 25  | 19  | ␣    | EM   | 57  | 39  | 9    | 89  | 59  | Y    | 121 | 79  | y              |
| ^Z  | 26  | 1A  | ␣    | SIB  | 58  | 3A  | :    | 90  | 5A  | Z    | 122 | 7A  | z              |
| ^[  | 27  | 1B  | ␣    | ESC  | 59  | 3B  | ;    | 91  | 5B  | [    | 123 | 7B  | {              |
| ^\  | 28  | 1C  | ␣    | FS   | 60  | 3C  | <    | 92  | 5C  | \    | 124 | 7C  |                |
| ]   | 29  | 1D  | ␣    | G-S  | 61  | 3D  | =    | 93  | 5D  | ]    | 125 | 7D  | }              |
| ^_  | 30  | 1E  | ␣    | RS   | 62  | 3E  | >    | 94  | 5E  | ^    | 126 | 7E  | ~              |
| ^`  | 31  | 1F  | ␣    | US   | 63  | 3F  | ?    | 95  | 5F  | _    | 127 | 7F  | Δ <sup>†</sup> |

† ASCII code 127 has the code DEL. Under MS-DCS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTLK+EKSP key.

| Dec | Hex | Char |
|-----|-----|------|
| 128 | 80  | Ç    |
| 129 | 81  | ü    |
| 130 | 82  | é    |
| 131 | 83  | â    |
| 132 | 84  | ä    |
| 133 | 85  | à    |
| 134 | 86  | ã    |
| 135 | 87  | ç    |
| 136 | 88  | ê    |
| 137 | 89  | ë    |
| 138 | 8A  | è    |
| 139 | 8B  | ÿ    |
| 140 | 8C  | î    |
| 141 | 8D  | ì    |
| 142 | 8E  | ï    |
| 143 | 8F  | ä    |
| 144 | 90  | é    |
| 145 | 91  | æ    |
| 146 | 92  | ø    |
| 147 | 93  | ô    |
| 148 | 94  | ö    |
| 149 | 95  | ò    |
| 150 | 96  | ù    |
| 151 | 97  | û    |
| 152 | 98  | ü    |
| 153 | 99  | ö    |
| 154 | 9A  | ü    |
| 155 | 9B  | ç    |
| 156 | 9C  | £    |
| 157 | 9D  | ¥    |
| 158 | 9E  | ℞    |
| 159 | 9F  | ƒ    |

| Dec | Hex | Char |
|-----|-----|------|
| 160 | A0  | á    |
| 161 | A1  | í    |
| 162 | A2  | ó    |
| 163 | A3  | ú    |
| 164 | A4  | ñ    |
| 165 | A5  | Ñ    |
| 166 | A6  | º    |
| 167 | A7  | º    |
| 168 | A8  | ¿    |
| 169 | A9  | ¡    |
| 170 | AA  | ¡    |
| 171 | AB  | ½    |
| 172 | AC  | ¼    |
| 173 | AD  | ¿    |
| 174 | AE  | «    |
| 175 | AF  | »    |
| 176 | B0  | ⋮    |
| 177 | B1  | ⋮    |
| 178 | B2  | ⋮    |
| 179 | B3  | ⋮    |
| 180 | B4  | ⋮    |
| 181 | B5  | ⋮    |
| 182 | B6  | ⋮    |
| 183 | B7  | ⋮    |
| 184 | B8  | ⋮    |
| 185 | B9  | ⋮    |
| 186 | BA  | ⋮    |
| 187 | BB  | ⋮    |
| 188 | BC  | ⋮    |
| 189 | BD  | ⋮    |
| 190 | BE  | ⋮    |
| 191 | BF  | ⋮    |

| Dec | Hex | Char |
|-----|-----|------|
| 192 | C0  | Ł    |
| 193 | C1  | ł    |
| 194 | C2  | ł    |
| 195 | C3  | ł    |
| 196 | C4  | ł    |
| 197 | C5  | ł    |
| 198 | C6  | ł    |
| 199 | C7  | ł    |
| 200 | C8  | ł    |
| 201 | C9  | ł    |
| 202 | CA  | ł    |
| 203 | CB  | ł    |
| 204 | CC  | ł    |
| 205 | CD  | ł    |
| 206 | CE  | ł    |
| 207 | CF  | ł    |
| 208 | D0  | ł    |
| 209 | D1  | ł    |
| 210 | D2  | ł    |
| 211 | D3  | ł    |
| 212 | D4  | ł    |
| 213 | D5  | ł    |
| 214 | D6  | ł    |
| 215 | D7  | ł    |
| 216 | D8  | ł    |
| 217 | D9  | ł    |
| 218 | DA  | ł    |
| 219 | DB  | ł    |
| 220 | DC  | ł    |
| 221 | DD  | ł    |
| 222 | DE  | ł    |
| 223 | DF  | ł    |

| Dec | Hex | Char |
|-----|-----|------|
| 224 | E0  | α    |
| 225 | E1  | β    |
| 226 | E2  | γ    |
| 227 | E3  | π    |
| 228 | E4  | σ    |
| 229 | E5  | τ    |
| 230 | E6  | μ    |
| 231 | E7  | γ    |
| 232 | E8  | ϕ    |
| 233 | E9  | θ    |
| 234 | EA  | ϖ    |
| 235 | EB  | δ    |
| 236 | EC  | ε    |
| 237 | ED  | ϑ    |
| 238 | EE  | €    |
| 239 | EF  | ∏    |
| 240 | F0  | ≡    |
| 241 | F1  | +    |
| 242 | F2  | ≥    |
| 243 | F3  | ≤    |
| 244 | F4  | ↑    |
| 245 | F5  | ↓    |
| 246 | F6  | ÷    |
| 247 | F7  | ≈    |
| 248 | F8  | •    |
| 249 | F9  | •    |
| 250 | FA  | •    |
| 251 | FB  | √    |
| 252 | FC  | ∥    |
| 253 | FD  | z    |
| 254 | FE  | ■    |
| 255 | FF  | ■    |

## Annexe B

# Bibliothèque Asciiart

### B.1 Introduction

Afin de pouvoir réaliser des exercices plus amusants et colorés, nous avons regroupé un ensemble de fonctions dans une bibliothèque, pompeusement appelée ASCIIART.

Cette bibliothèque permet pour l'instant de modifier les couleurs du terminal et de saisir des caractères à la volée, avec ou sans echo. Il ne tient qu'à vous de l'améliorer.

Sachez cependant qu'il existe de *très bonnes* bibliothèques faisant entre autres tout ceci, comme le célèbre `ncurses`, dont l'utilisation complète dépasse l'objet de ce cours.

### B.2 Utilisation de la bibliothèque

Les fonctions sont déclarées dans le fichier `asciiart.h`. Vous devez donc l'inclure dans vos sources (`#include "asciiart.h"`). La fonction `test_asciiart()` permet de tester les différentes fonctions de la bibliothèque. Celle-ci est fournie au format objet (`.o`), mais vous pouvez bien sûr regarder la source.

Pour compiler l'exemple suivant :

```
test_bibli.c
```

```
#include "asciiart.h"
int main(void)
{
 test_asciiart();
 return(0);
}
```

vous devez entrer :

```
gcc -c test_bibli.c
```

puis :

```
gcc -o test_bibli test_bibli.o asciiart.o
```

et enfin exécuter :

```
./test_bibli
```

### B.3 Fichier d'en-tête

À l'heure actuelle, le fichier d'en-tête est le suivant :

asciiart.h

```
#ifndef ASCII_ART_H
#define ASCII_ART_H

/* Cet ensemble de fonctions permet d'utiliser la saisie de caractères
à la console en mode non bufferisé (ici appelé mode immédiat)
Il permet d'accéder à différents paramètres du terminal comme la couleur
*/

// Saisit un caractère au vol. Sans echo
char getch(void);
// Saisit un caractère au vol. Avec echo
char getche(void);
// procédure permettant de tester les différentes fonctionnalités
void test_asciiart(void);
// modifie la couleur d'affichage
void textcolor(int i); // 0<=i<=7
// passage (1) ou sortie (0) mode clignotement (expérimental...)
void textblink(int val);
// réinitialisation de la console
void textreset(void);
// Exemple d'utilisation :
//# include "asciiart.h"
//int main(void)
//{
//test_asciiart();
//return 0;
//}
#endif
```