1 Pointeurs et Fonctions

1.1 Variables : pointeurs et valeurs

1.1.1 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 (c'est la boîte vue durant le niveau 1). Chaque type de variable utilise au moins 1 octet. Le type int varie selon les compilateurs, pour nous il utilise 2 octets c'est à dire 2 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, c'est ce que l'on appelle son adresse postale. Pour une case mémoire, on parlera d'adresse mémoire.
Cette adresse est unique pour chaque maison. Cependant, rien ne vous empêche de vous tromper intentionnellement ou non de maison. De la même façon, l'adresse mémoire d'une case est unique mais rien ne vous empêche de vous tromper intentionnellement ou non de case mémoire. (cf. Niveau 1)

Une adresse mémoire :

Exemple vie courante :

Mr Leneuf adresse : 99, grand rue

Exemple en C :

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

0x signifie adresse hexadécimale (ou 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 : %ld [%lx]",&car, &car);
printf ("Adresse de nbre : %ld [%lx]",&nbre, &nbre);
```

On ajoute "l" devant le d (%ld) et devant le x (%lx) afin de faire afficher un entier long.

1.1.2 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 C doit interpréter l'adresse mémoire lors de sa lecture. Lors de l'utilisation des chaînes de caractères nous en verrons des exemples.
Pour enregistrer une valeur dans une adresse mémoire on écrit :

***variable = <valeur>**

*variable signifie contenu de l'adresse mémoire.

Exemple:

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

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

        clrscr ();
        printf ("Avant, le caractère est : %c",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",car);  /* i.e. on a modifié car */

        getch ();
        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.

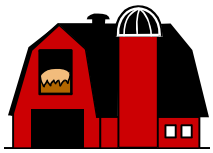
ptr_car = &car; Signifie que l'on m'a donné l'adresse postale de M. car.
*ptr_car = 'E'; Signifie que je rentre chez M. car et que j'y dépose le caractère E.
printf ("%c",car); Signifie que l'on va lire ce qu'il y a dans la maison de M. et qu'on le fait afficher.

1.1.3 Exercices d'application

Réaliser un programme équivalent qui change la valeur en K.
Réaliser un programme équivalent qui change une valeur numérique (int) de 10 à 35.

1.2 Les fonctions

1.2.1 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 notre code (dans notre 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 du programme

        Retour
}
```

1.2.2 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);
}
```

1.2.3 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

```
int carre (int val)
{
        int val_retour = 0;     /* Déclaration variable locale */
        val_retour = val * val;
        return (val_retour);
}
void main ()
{
        int val_retour = 0;     /* Déclaration variable locale */

        val_retour = carre (2);
        printf ("Le carré de 2 est : %d", val_retour);
        getch ();
}
```

val_retour est dans les deux cas une **variable locale**. Bien qu'elles aient le même nom dans la fonction main et carre, **les deux variables n'ont rien à voir entre elles**.

Exemple de variable globale

💣 Il est fortement déconseillé d'utiliser les variables globales. 💣

```
#include …

int     val = 0;        /* Déclaration variable globale */

int carre ()
{
        int     val_retour = 0;
        val_retour = val * val;
        return (val_retour);
}

void main ()
{
        val = 2;
        carre ();
        printf ("Le carre de 2 est %d", carre ());
}
```

val est une variable globale.

On constate que le programme devient rapidement illisible …

1.2.4 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

```
void Affiche_Coucou (int x, int y)
{
        gotoxy (x,y);
        printf ("coucou");
}

void main ()
{
        Affiche_coucou (10,12);
        getch ();
}
```

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

Exemple

```
void Avance_Position (int* x)
{
        *x = *x + random (6);
}

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

        printf ("Position actuelle : %d",x);

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

        getch ();
}
```

1.2.5 Piège !

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

En effet :

```
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 ...

1.2.6 Aide pour l'éditeur Turbo C 2.0

Le Turbo C permet de réaliser des copier coller dont vous aurez besoin dans les exercices suivants. Pour se faire, il faut sélectionner un bloc (un bloc est un morceau de texte de l'éditeur, il est marqué en surbrillance).

Note : Ctrl Lettre Lettre s'opère en appuyant sur la touche Ctrl et sans lâcher cette touche en appuyant successivement sur les deux lettres.

Pour sélectionner un bloc :

1. Utilisez la combinaison de touches Ctrl K B, pour marquer le début du bloc.
2. Utilisez la combinaison de touches Ctrl K K pour marquer la fin du bloc.
3. Votre zone est maintenant en surbrillance.

Pour désélectionner un bloc :
Utilisez la combinaison de touches Ctrl K K au début du bloc en surbrillance.

Positionnez vous à l'endroit où vous désirez faire la copie et faîtes Ctrl K C
Pour déplacer faîtes Ctrl K V

1.2.7 Exercices

Réalisez une fonction Avance_cheval qui :

- ≅ prendra pour paramètres la position x, la position y, la couleur et le caractère représentant le cheval.
- ≅ avancera le cheval en l'effaçant, le dessinant et mettant à jour sa position.

Augmenter le nombre de chevaux à 5.

Correction de l'exercice du chapitre 1

En **gras** apparaissent les principaux changements opérés depuis le programme du niveau 1 chapitre 10.

! *Exercices 1.1.3*

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

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

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

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

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

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

      clrscr ();
      printf ("Avant, la valeur est : %d",val);
      ptr_val = &val;
      *ptr_val = 35;
      printf ("\nAprès la valeur est : %d",val);

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

! *Fonction Avance_cheval*

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

void Avance_cheval (int *pos_x, int pos_y, int coul, char cheval)
{
      /* Effacement du cheval (position précédente) */
      gotoxy (*pos_x,pos_y);
      printf (" ");

      /* Affichage du cheval */
      textcolor (coul);
```

```
      *pos_x += random (6) + 1;
      gotoxy (*pos_x,pos_y);
      cprintf ("%c",cheval);
}

int main ()
{
      int   i;
      int   x1=0,x2=0,x3=0;
      time_t      t;
      int   pari;
      int   premier;
      int   sortie;
      int   coul1,coul2,coul3;
      char car;

      /* Pari sur un cheval */
      clrscr ();
      gotoxy (1,1);
      printf ("Sur quel cheval voulez vous parier (1,2 ou 3) ?");

      /* Choix d'un cheval */
      do
      {
            sortie = 1;

            car = getch ();

            switch (car)
            {
            case '1':
                  pari = 1;
                  coul1 = 2;
                  coul2 = 1;
                  coul3 = 1;
                  break;

            case '2':
                  pari = 2;
                  coul1 = 1;
                  coul2 = 2;
                  coul3 = 1;
                  break;

            case '3':
                  pari = 3;
                  coul1 = 1;
                  coul2 = 1;
                  coul3 = 2;
                  break;

            default:
                  sortie = 0;
                  printf ("%c",0x7);
                  break;
            }
      } while (!sortie);

      /* Efface l',cran */
      clrscr ();
```

```
	/* Dessin de la piste */
	for (i=1; i<=80; i++)
	{
		gotoxy (i,10);
		printf ("-");

		gotoxy (i,14);
		printf ("-");
	}

	/* Initialisation des variables aléatoire */
	randomize ();

	/* Dessin des chevaux */
	do
	{
		Avance_cheval (&x1, 11, coul1, '1');
		Avance_cheval (&x2, 12, coul2, '2');
		Avance_cheval (&x3, 13, coul3, '3');

		/* Attente */
		for (i=0; i<5000; i++)
			time (&t);
	}
	while ((x1<74) && (x2<74) && (x3<74));

	if ((x1>x2) && (x1>x3))
		premier = 1;
	else
	{
		if ((x2>x1) && (x2>x3))
			premier = 2;
		else
		{
			if ((x3>x1) && (x3>x2))
				premier = 3;
			else
				premier = 0 ;
		}
	}

	gotoxy (1,14);
	if (premier == pari)
		printf ("\nBravo vous avez gagné.");
	else
	{
		if (premier == 0)
			printf ("\nDésolé il y a égalité");
		else
			printf   ("\nDésolé   vous   avez   perdu.\nCheval   n°%d
		vainqueur",premier);
	}

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

Nous pouvons remarquer que le programme devient plus propre, plus clair.

! *5 chevaux*

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

```
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

void Avance_cheval (int *pos_x, int pos_y, int coul, char cheval)
{
      /* Effacement du cheval (position pr,c,dente) */
      gotoxy (*pos_x,pos_y);
      printf (" ");

      /* Affichage du cheval */
      textcolor (coul);
      *pos_x += random (6) + 1;
      gotoxy (*pos_x,pos_y);
      cprintf ("%c",cheval);
}

int main ()
{
      int   i;
      int   x1=0,x2=0,x3=0,x4=0,x5=0;
      time_t      t;
      int   pari;
      int   premier;
      int   sortie;
      int   coul1,coul2,coul3,coul4,coul5;
      char car;

      /* Pari sur un cheval */
      clrscr ();
      gotoxy (1,1);
      printf ("Sur quel cheval voulez vous parier (1,2,3,4 ou 5) ?");

      /* Choix d'un cheval */
      do
      {
            sortie = 1;

            car = getch ();

            switch (car)
            {
            case '1':
                  pari = 1;
                  coul1 = 2;
                  coul2 = 1;
                  coul3 = 1;
                  coul4 = 1;
                  coul5 = 1;
                  break;

            case '2':
                  pari = 2;
                  coul1 = 1;
                  coul2 = 2;
                  coul3 = 1;
                  coul4 = 1;
                  coul5 = 1;
                  break;
```

```
      case '3':
            pari = 3;
            coul1 = 1;
            coul2 = 1;
            coul3 = 2;
            coul4 = 1;
            coul5 = 1;
            break;

      case '4':
            pari = 4;
            coul1 = 1;
            coul2 = 1;
            coul3 = 1;
            coul4 = 2;
            coul5 = 1;
            break;

      case '5':
            pari = 5;
            coul1 = 1;
            coul2 = 1;
            coul3 = 1;
            coul4 = 1;
            coul5 = 2;
            break;

      default:
            sortie = 0;
            printf ("%c",0x7);
            break;
      }
} while (!sortie);

/* Efface l',cran */
clrscr ();

/* Dessin de la piste */
for (i=1; i<=80; i++)
{
      gotoxy (i,10);
      printf ("-");

      gotoxy (i,16);
      printf ("-");
}

/* Initialisation des variables aléatoire */
randomize ();

/* Dessin des chevaux */
do
{
      Avance_cheval (&x1, 11, coul1, '1');
      Avance_cheval (&x2, 12, coul2, '2');
      Avance_cheval (&x3, 13, coul3, '3');
      Avance_cheval (&x4, 14, coul4, '4');
      Avance_cheval (&x5, 15, coul5, '5');

      /* Attente */
      for (i=0; i<5000; i++)
```

```
                time (&t);
        }
        while ((x1<74) && (x2<74) && (x3<74) && (x4<74) && (x5<74));

        if ((x1>x2) && (x1>x3) && (x1>x4) && (x1>x5))
                premier = 1;
        else
        {
                if ((x2>x1) && (x2>x3) && (x2>x4) && (x2>x5))
                        premier = 2;
                else
                {
                        if ((x3>x1) && (x3>x2) && (x3>x4) && (x3>x5))
                                premier = 3;
                        else
                        {
                                if ((x4>x1) && (x4>x2) && (x4>x3) && (x4>x5))
                                        premier = 4;
                                else
                                {
                                        if ((x5>x1) && (x5>x2) && (x5>x3) && (x5>x4))
                                                premier = 5;
                                        else
                                                premier = 0;
                                }
                        }
                }
        }

        gotoxy (1,17);
        if (premier == pari)
                printf ("\nBravo vous avez gagné.");
        else
        {
                if (premier == 0)
                        printf ("\nDésolé il y a égalité entre deux chevaux.");
                else
                        printf   ("\nDésolé   vous   avez   perdu.\nCheval   n°%d
vainqueur",premier);
        }

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

Les test opérés à la fin du programme ne sont pas très réjouissants mais nous sommes là pour apprendre le C et non les méthodes d'optimisation.