Licence 1 - section B
TP 2 d'éléments d'informatique

Catherine RECANATI – Département d'Informatique – Institut Galilée

Semaine du 14 au 18 novembre 2016

L'objectif de ce TP est de vous familiariser avec les notions d'itération et de boucles imbriquées, ainsi qu'avec les expressions booléennes et leur utilisation dans une structure de contrôle "while".

Nous allons mettre tous les programmes écrits dans ce TP dans le répertoire TP2.

1. Depuis votre répertoire principal (base de votre arborescence de fichiers), créez le répertoire TP2 en tapant la commande `mkdir TP2` dans une fenêtre de terminal.
2. Allez dans le répertoire TP2 grâce à la commande `cd` (= *change directory*) en tapant : `cd TP2`.
3. Lancez l'éditeur de texte `gedit` pour créer un nouveau fichier source appelé `exo1.c`, en tapant la commande `gedit exo1.c &`. Le caractère & en fin de commande permet de lancer l'exécution de la commande en arrière plan (et ainsi garder la main dans la fenêtre du terminal, sans avoir à attendre la fin de la commande, ici donc sans attendre la fin de l'éditeur `gedit`) . L'intérêt ici, sera de pouvoir lancer la compilation du programme et son exécution sans avoir à quitter l'éditeur `gedit`.
4. ...ou bien copier simplement le fichier `bonjour.c` de votre répertoire TP1 dans TP2 en tapant :
   ```
   cp ../TP1/bonjour.c exo1.c
   gedit exo1.c &
   ```
   Vous pouvez-aussi ouvrir `bonjour.c` et utiliser la fonction *Enregistrer sous...* de votre éditeur mais attention à bien enregistrer la nouvelle copie dans le bon répertoire (c'est-à-dire celui, ici TP2, d'où vous lancez la compilation et l'exécution du programme compilé).

Vous pouvez utiliser à tout moment la commande ls (list directory) pour voir la liste des fichiers d'un répertoire.
Comme indiqué dans le premier TP, les trois étapes suivantes sont à répéter autant de fois que nécessaire pour la mise au point de chaque programme (apprenez à utiliser les raccourcis clavier).

1. Après avoir fini d'écrire votre programme, enregistrez le.
2. Créez un programme exécutable à partir de votre fichier source :
   ```
   gcc -Wall exo1.c -o exo1.exe
   ```
3. Quand l'étape précédente est franchie avec succès (il faut lire attentivement les messages affichés pour corriger les erreurs), exécutez le programme pour vérifier qu'il fonctionne en tapant `exo1.exe` (ou `./exo1.exe`).

Si vous n'avez pas d'inspiration vous pouvez appeler vos fichiers exo1.c ou exo1.1.c, mais le mieux serait que vous donniez un nom évocateur du contenu à vos programmes (dans cette feuille par exemple, rectangleEtoile.c, demiCarre.c, nombreSecret.c, etc.).

1 Affichage de figures géométriques

Les exercices suivants utilisent le caractère $\star$ (étoile) pour dessiner des figures géométriques simples, appelées figures d'étoiles.

1.1 Affichage d'un rectangle d'étoiles

Exercice 1.1 Écrire un programme qui, étant données deux variables entières `longueur` et `largeur`, initialisées à des valeurs strictement positives quelconques, affiche un rectangle d'étoiles ayant pour longueur `longueur` étoiles et largeur `largeur` étoiles. Deux exemples d'exécution, avec deux initialisations différentes, sont les suivants :

```
Affichage d'un rectangle d'etoiles de longueur 10 et largeur 5.
**********
**********
```

```
**********
**********
**********

Affichage d'un rectangle d'etoiles de longueur 6 et largeur 3.
******
******
******
```

Correction 1.1 *Les algos sont à écrire d'abord en français (les extraire du code).*

*Vous pouvez dans un premier temps supprimer la boucle la plus imbriquée en leur demandant d'afficher un rectangle de longueur exactement "**********".*

```
/* declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* pour EXIT_SUCCESS */
#include <stdio.h> /* pour printf */

/* declaration de constantes et types utilisateurs */

/* declaration de fonctions utilisateurs */

/* fonction principale */
int main()
{
    /* declaration et initialisation de variables locales */
    int largeur = 3; /* largeur du rectangle en nb d'etoiles */
    int longueur = 6; /* longueur du rectangle en nb d'etoiles */
    int i; /* var. de boucle */
    int j; /* var. de boucle */

    printf("Affichage d'un rectangle d'etoiles de longueur %d et largeur %d.\n",
                        longueur,largeur);

    for( i = 0; i < largeur; i = i + 1) /* pour chaque ligne d'étoiles */
    {
        /* affiche longueur etoiles */
        for(j = 0; j < longueur; j = j + 1) /* chaque colonne d'etoiles */
        {     /* affiche une etoile */
            printf("*");
        }
        /* en sortie du for,  j >= longueur */

        /* pour passer a la ligne suivante */
        printf("\n");
    }
    /* en sortie du for,  i >= largeur */

    return EXIT_SUCCESS;
}

/* definitions des fonctions utilisateurs */
```

1.2 Affichage d'un demi-carré d'étoiles

Exercice 1.2 Écrire un programme qui affiche, étant donnée la variable `cote`, initialisée à une valeur entière quelconque, un demi-carré d'étoiles (triangle rectangle isocèle) ayant pour longueur de côté `cote` étoiles. Deux exemples d'exécution (avec deux initialisations différentes) sont les suivants :

```
Affichage d'un demi-carre d'etoiles de cote 6.
*
**
```

```
***
****
*****
******

Affichage d'un demi-carre d'etoiles de cote 2.
*
**
```

Correction 1.2 *Les algos sont à faire d'abord en français (les extraire du code).*

```
/* declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf */

/* declaration constantes et types utilisateurs */

/* declaration de fonctions utilisateurs */

/* fonction principale */
int main()
{
    /* declaration et initialisation variables */
    int cote = 2; /* cote du demi-carré en nb d'etoiles */
    int i; /* var. de boucle */
    int j; /* var. de boucle */

    printf("Affichage d'un demi-carre d'etoiles de cote %d.\n",cote);

    for(i = 1;i <= cote;i = i + 1) /* chaque numero de ligne d'étoiles */
    {
        /* affiche autant d'etoiles que le numero de ligne */
        for(j = 0;j < i;j = j + 1) /* chaque colonne d'etoiles */
        {
            /* affiche une etoile */
            printf("*");
        }
        /* j >= i */

        /* passe a la ligne suivante */
        printf("\n");
    }
    /* i > cote */

    return EXIT_SUCCESS;
}

/* definitions des fonctions utilisateurs */
```

1.3 Affichage d'un demi-carré droit d'étoiles

Exercice 1.3 Écrire un programme qui affiche un demi-carré droit d'étoiles de côté spécifié par l'utilisateur. Exemple d'exécution :

```
Entrer la taille du demi-carré :
5
Affichage d'un demi-carre droit d'etoiles de cote 5.
    *
   **
  ***
 ****
*****
```

Correction 1.3
```
/* declaration de fonctionnalites supplementaires */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */

/* declaration constantes et types utilisateurs */

/* declaration de fonctions utilisateurs */

/* fonction principale */
int main()
{
    int cote; /* cote du demi-carré droit en nb d'etoiles a saisir par l'utilisateur*/
    int i; /* var. de boucle */
    int j; /* var. de boucle */

    /* saisie cote */
    printf("Entrer la taille du demi-carré :\n");
    scanf("%d",&cote);

    /* affichage du demi-carre droit */
    printf("Affichage d'un demi-carre droit d'etoiles de cote %d.\n",cote);

    for(i = 1;i <= cote;i = i + 1) /* chaque numero de ligne d'étoiles */
    {
        /* affiche les blancs */
        for(j = 0;j < cote - i;j = j + 1) /* chaque colonne de blancs */
        {
            /* affiche un blanc */
            printf(" ");
        }
        /* j >= cote - i */

        /* affiche autant d'etoiles que le numero de ligne */
        for(j = 0;j < i;j = j + 1) /* chaque colonne d'etoiles */
        {
            /* affiche une etoile */
            printf("*");
        }
        /* j >= i */

        /* passe a la ligne suivante */
        printf("\n");
    }
    /* i > cote */

    return EXIT_SUCCESS;
}

/* definition de fonctions utilisateurs */
```

2 Un nombre secret

Nous voulons programmer le jeu d'un nombre à découvrir. Le joueur doit deviner un nombre secret choisi par l'ordinateur entre 0 et `NB_MAX` (une constante du programme). S'il propose un nombre trop grand, l'ordinateur lui répond "Plus petit", s'il propose trop petit, l'ordinateur lui répond "Plus grand". Dans ces deux cas, il est invité à proposer un autre nombre. Le jeu s'arrête quand il devine le nombre exact. Un exemple d'exécution de ce jeu pourrait être :

```
Votre choix ?
```

```
8
Plus petit.
Votre choix ?
4
Plus petit.
Votre choix ?
2
Vous avez trouvé le nombre secret.
```

Exercice 2.1 Proposez un algorithme en français pour le jeu (en considérant que le nombre à deviner est initialisé directement dans le programme), traduisez-le en langage C et exécutez-le. Pourquoi une boucle `while` convient bien ici ?

Exercice 2.2 Modifier le programme précédent pour que le nombre à deviner soit initialisé aléatoirement.

Indications : l'ordinateur doit choisir le nombre secret *au hasard*. La librairie C standard propose des fonctions (déclarées dans `<stdlib.h>`) qui renvoient des nombres pseudo-aléatoires [1]. On va ici utiliser la fonction :
`int rand();`
qui renvoie un nombre entier compris entre 0 et la constante `RAND_MAX` incluse (`RAND_MAX` égale 2147483647). Le nom `rand` vient de *random* qui signifie aléatoire en anglais. Pour obtenir un nombre entre 0 et `NB_MAX` inclus, (avec `NB_MAX` $<$ `RAND_MAX`), il suffit de considérer le reste de la division entière d'un appel à `rand()` par `(NB_MAX + 1)`, ce qui s'écrira avec l'opérateur modulo noté % en C : `rand() % (NB_MAX + 1)`.

Un exemple de programme illustrant l'utilisation de `rand` pour engendrer un nombre pseudo-aléatoire est le suivant :

```
#include <stdlib.h> /* pour EXIT_SUCCESS, rand, srand */
#include <time.h> /* pour time */

#define NB_MAX 15 /* le nombre secret sera entre 0 et NB_MAX inclus */

int main()
{
    int nombre_secret; /* nombre secret à deviner */

    /* initialisation du générateur de nombres pseudo-aléatoires */
    srand(time(NULL)); /* à ne faire qu'une fois */

    /* tirage du nombre secret */
    nombre_secret = rand() % (NB_MAX + 1); /* entre 0 et NB_MAX inclus */

    /* manche joueur ... */

    return EXIT_SUCCESS;
}
```

Correction 2.1 *Une boucle while convient bien ici car on ne connait pas par avance le nombre de tour de boucle qui seront nécessaires pour deviner le nombre.*

Correction 2.2
```
#include <stdlib.h> /* EXIT_SUCCESS, rand, srand */
#include <stdio.h> /* printf, scanf */
#include <time.h> /* time */

#define TRUE 1
#define FALSE 0

#define NB_MAX 15 /* nombre secret entre 0 et NB_MAX inclus */

/* declaration de fonctions utilisateurs */

int main()
```

1. http ://fr.wikipedia.org/wiki/Générateur_de_nombres_pseudo-aléatoires

```
{
    int nombre_secret; /* nombre secret à deviner */
    int choix; /* choix de l'utilisateur pour le nombre secret */
    int pas_trouve = TRUE; /* TRUE si pas trouvé nombre secret */

    /* initialisation du générateur de nombres pseudo-aléatoires */
    srand(time(NULL)); /* à ne faire qu'une fois */

    /* tirage du nombre secret */
    nombre_secret = rand() % (NB_MAX + 1); /* entre 0 et NB_MAX inclus */

    /* manche joueur */
    while(pas_trouve) /* pas trouvé nombre secret */
    {
        /* demande nombre à l'utilisateur */
        printf("Votre choix (nombre entre 0 et %d) ?\n",NB_MAX);
        scanf("%d",&choix);

        if(choix == nombre_secret) /* trouvé */
        {
            pas_trouve = FALSE;
        }
        else /* pas trouvé */
        {
            /* donne indice */
            if(choix > nombre_secret)
            {
                printf("Plus petit.\n");
            }
            else
            {
                printf("Plus grand.\n");
            }
        }
    }
    /* trouvé nombre secret */

    printf("Vous avez trouvé le nombre secret.\n");

    return EXIT_SUCCESS;
}

/* Definition de fonctions utilisateurs */
```

3 Évaluation d'expressions booléennes

Une table de vérité donne la valeur d'une ou de plusieurs expressions booléennes, construites à partir de variables et d'opérateurs (booléens), en fonction des valeurs de ses variables. Un exemple de table de vérité est celle qui donne pour l'expression *a OU b* ses valeurs. Elle est constituée du tableau ci-dessous (avec F : faux et V : vrai) :

a	b	a OU b
F	F	F
F	V	V
V	F	V
V	V	V

Exercice 3.1 Écrire un programme qui demande à l'utilisateur les valeurs de deux variables booléennes `a` et `b` et qui affiche à l'écran la ligne correspondante de la table de vérité de l'ensemble des expressions : a ET b, a OU b, NON a, NON b, (NON a) ET b. Cette ligne à afficher (qui aura 7 "colonnes") est donc déterminée par la valeur des deux variables a et b. Pour une indentation correcte des colonnes, vous pouvez utiliser dans printf le meta-caractère `'\t'`

qui affiche une tabulation.

Voici un exemple d'exécution du programme :

```
entrez deux valeurs booleennes : 1 0
a       b       a ET b  a OU b  NON a   NON b   (NON a) ET b
1       0       0       1       0       1       0
```

Correction 3.1
```
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h> /* printf, scanf */

/* declaration constantes et types utilisateurs */

/* declaration de fonctions utilisateurs */

int main()
{
    int a; /* booléen */
    int b; /* booléen */

    printf("Entrez deux valeurs booléennes : ");
    scanf("%d",&a);
    scanf("%d",&b);

    printf("a\tb\ta ET b\ta OU b\tNON a\tNON b\t(NON a) ET b\n");
    printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n",a,b,a && b,a || b,!a,!b,!a && b);

    return EXIT_SUCCESS;
}

/* Definition de fonctions utilisateurs */
```

Exercice 3.2 Ajouter une colonne supplémentaire, celle de NON (a ET b) et comparez les deux dernières colonnes en exécutant le programme pour différentes valeurs. Les résultats vous surprennent-ils ?

Correction 3.2 *On constate effectivement que les expressions NON (a ET b) et (NON a) ET b ne sont pas équivalentes. Il est donc important de bien mettre les parenthèses pour savoir de quoi on parle. Sans parenthèses, i.e. pour l'expression NON a ET b, c'est la priorité des opérateurs qui détermine l'ordre du calcul, et l'opérateur NON étant prioritaire sur l'opérateur ET, c'est l'expression (NON a) ET b qui est calculée.*
On retiendra qu'il n'est pas nécessaire de connaitre l'ordre de priorité des opérateurs si on met explicitement des parenthèses, et que c'est aussi la meilleure façon de faire, pour que le code source puisse être compris par d'autres.