esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

TD n°3

INF1031

Fonctions

Arnaud Bannier
Nicolas Bodin
Aurélien Texier

1. Les conditionnelles

Le code suivant est sale, vous allez devoir nettoyer tout ça. Dans un premier temps, réécrivez-le en conservant la structure `if ... else`.

```
1  if ( a == 0 )
2    printf( "Ca vaut 0\n" );
3  if ( a == 1 )
4    printf( "Ca vaut 1\n" );
5  if ( a == 2 )
6    printf( "%d\n", a );
7  if ( a == 3 )
8    printf( "%d\n", a );
9  if ( a == 5 )
10   printf( "%d\n", a );
11 if ( a == 4 )
12   printf( "%d\n", a );
13 if ( (a!=0) && (a!=1) && (a!=2) && (a!=3) && (a!=4) && (a!=5) )
14   printf( "Ca vaut un truc\n" );
```

Réécrivez-le maintenant à l'aide de la structure `switch ... case`.

2. Un peu de code

Avant de commencer, récupérez l'exemple du cours concernant le *Hello World*, compilez-le puis exécutez-le. Ce code fournit la base pour le reste de l'exercice.

— Écrivez un programme qui vous affiche la surface d'un cercle, en utilisant un diamètre fixe dans votre programme.
— Maintenant rajoutez le calcul de la surface d'un rectangle de la longueur de votre choix et de la même largeur que le périmètre de votre cercle.
— Puis finissez votre programme en calculant la surface totale du cylindre, décrit par le calcul des éléments précédents.
— Modifiez votre programme pour que l'utilisateur rentre lui même les différentes valeurs de longueur et de diamètre, puis affichez le résultat final.

3. Compréhension de code

Sans compiler, ou exécuter ce code, qu'affiche le programme suivant ?

```c
#include <stdio.h>
int main()
{
    int res,n;
    for(res=1,n=0;n<8;n=n+1)
    {
        res=2*res;
        printf("res=%d\n",res);
    }
    return 0;
}
```

Pour vous aider, vous pouvez compléter le tableau suivant :

n	
res	

Compilez ce code et exécutez le afin de vérifier votre travail.

4. Erreur de code

Le code suivant comporte quelques erreurs et/ou avertissements. Avant de le compiler, trouvez et corrigez ces erreurs, puis vérifiez votre correction en compilant ce code.

```c
int main(
{
    int i;j;
    scanf("%d %d",i,j);
    for(i=0,j=1,i>3;i=i+1,j=j+2)
        printf("i=%d, j=%d\n,i,j)
    if (i=j)
        i=2*j
    else
        printf("%d\n",j);
    printf("\nAu revoir\n");
    return(0);
}
```

5. Du code, du code, du code !

Ecrivez une boucle partant de 0 et allant jusqu'à 100 inclus par une incrémentation de 1. Affichez la valeur de votre variable juste avant la fin du programmme.

Ecrivez une boucle qui demander à l'utilisateur de rentrer une valeur comprise entre 0 et 100 tant que cette valeur n'est pas comprise dans cet intervalle.

Rajoutez au code précédent une variable entière, nommée tentatives.

Affichez le nombre de tentatives en sortie de boucle.

Ajoutez une variable entière du nom de votre choix et fixez-la arbitrairement à une valeur comprise entre 0 et 100.

Modifiez votre boucle afin de vérifier si la valeur de l'utilisateur est égale à votre variable. Si c'est le cas sortez de votre boucle, autrement continuez.

Modifiez votre code pour que l'ordinateur choisisse une valeur aléatoire comprise entre 0 et 100 et laissez l'utilisateur tenter de la trouver. Pour cette étape, il est conseillé de regarder la partie suivante.

6. Trucs & astuces !

Génération d'aléa

On souhaite souvent dans le programme que le hasard intervienne. Simuler le hasard est quelque chose de très compliqué. Pire, il est parfois capital que l'on puisse refaire exactement la même séquence de nombre aléatoire pour comparer l'efficacité de plusieurs algorithmes par exemple. On parle alors de séquences pseudo-aléatoires.

La fonction C de base dont vous avez besoin pour créer du hasard est la fonction `rand()` qui, à chaque appel, va renvoyer un nombre pseudo-aléatoire compris entre 0 et `RAND_MAX` (`man` vous dit qu'il faut inclure `stdlib.h`). Cette fonction part d'une situation de départ, appelée graine dans la littérature, pour générer la séquence. Dit autrement, à partir d'une même graine, vous verrez apparaître toujours la même séquence de nombres aléatoires. Le programme suivant par exemple génère 10 nombres aléatoirement entre 0 et `RAND_MAX`.

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

int main()
{
    int i;
    for(i=0;i<10;i=i+1)
        printf("%ld\n",rand());
    return(0);
}
```

Si vous souhaitez maintenant que le tirage soit compris entre 0 et 100 par exemple, la ligne contenant le `printf` est remplacée par :

```
printf("%d\n", (int)(100.*rand()/RAND_MAX));
```

Commentons un peu le code précédent : le code `rand()/RAND_MAX` permet de ramener les nombres tirés aléatoirement entre 0 et 1. Il suffit alors de multiplier le résultat précédent par 100 pour que le tirage soit alors compris entre 0 et 100. Mais à cette étape, le résultat est de type réel et on souhaiterait l'avoir en entier. Pour forcer la transformation de ce résultat en type entier, il faut mettre devant l'expression `(int)`. Cette opération s'appelle dans la littérature une conversion de type (cast en anglais) explicite.

Écrivez maintenant le programme précédent et exécutez-le plusieurs fois. Vous devez observez que vous obtenez toujours la même séquence de nombres, ce qui pour un tirage aléatoire n'est pas très satisfaisant. La raison en est que vous partez toujours de la même graine (i.e. situation de départ) lors de l'appel de rand.

Pour changer cela, il vous faut utiliser la fonction `srand` qui permet de fournir une nouvelle graine à la fonction `rand`. D'après `man`, le prototype de cette fonction est `void srand (unsigned int seed)`. La grande question est alors comment générer le nombre `seed` (graine en anglais) pour qu'il soit différent à chaque lancement du programme ?

Pour cela, on utilise classiquement l'horloge de la machine qui change en permanence via la fonction `time`. Cette fonction donne le nombre de secondes écoulées depuis le 1er janvier 1970 minuit GMT. Ainsi, pour peu qu'il s'écoule plus d'une seconde entre 2 exécutions successives, vous avez 2 graines différentes pour `srand` qui génèrera donc 2 situations de départ différentes pour `rand` et au final, vous obtiendrez 2 séquences aléatoires qui sembleront totalement décorrélées.

Pour fixer les idées, il suffit d'ajouter au code précédent l'instruction `srand(time(NULL))` en début de `main` pour obtenir à chaque lancement de l'exécutable une séquence de 10 nombres aléatoires différente de celle obtenue à l'exécution précédente.

scanf et les « retour charriot »

Copiez et exécutez le code suivant :

```
int main()
{
    int i, res=0;
    while (res < 10)
    {
        i = scanf("%d", &res);
        printf ("i: %d\t-- %d --\n", i, res);
    }
    return 0;
}
```

Que se passe-t-il si vous utilisez des caractères alphabétiques ? Que proposez-vous pour contrer cela ? Sous Windows il est possible d'utiliser `fflush(stdin)`, malheureusement cela ne passe pas sous Linux sans effectuer une manipulation spéciale que vous ne verrez pas dans ce td. C'est pourquoi nous vous recommandons d'utiliser le code suivant :

```c
#include <stdio.h>

void clean_stdin(void)
{
    int c;
    do
    {
        c = getchar();
    } while (c != '\n' && c != EOF);
}

int main()
{
    int i, res=0;
    while (res < 10)
    {
        i = scanf("%d", &res);
        if (i<=0)
            clean_stdin();
        printf ("i: %d\t-- %d --\n", i, res);
    }
    return 0;
}
```

Le détail complet de ce code vous sera fourni dans un prochain TD (après avoir vu les fonctions). Mais vous pouvez d'ores et déjà l'utiliser dans vos codes.