Séance 1 - Révision algorithmique/Prise en main Codeblocks

Nous nous contentons dans un premier temps de faire quelques rappels algorithmiques. Les exercices sont à faire sur papier et les réponses seront données en langage algorithmique.

A FAIRE SUR PAPIER

Exercice 1 (Expressions booléennes) Donner la table de vérité des expressions suivantes :
- non(a ou b)
- (a et non(b)) ou (b et non(a))
- non(a) et non(b)
- non(a ou b) ou (a et b)

Exercice 2 (Nombres parfaits) Un nombre entier positif est dit *parfait* s'il est est égal à la somme de ses diviseurs propres + 1.
Exemples : $6 = 2 \text{ x } 3$ et $6 = 2 + 3 + 1$;
$28 = 2 \text{ x } 14 = 4 \text{ x } 7$ et $28 = 2 + 14 + 4 + 7 + 1$;
Écrivez un algorithme qui affiche tous les nombres parfaits $\leq$ à une valeur donnée.

Exercice 3 (Lecture de Code C) Pour les programmes suivants, dire si ils sont corrects ou non. Si vous détectez des erreurs, les corriger. Ensuite, donner le résultat affiché.

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

int main() {
        int a,b;
        a=(1+2*3+8)/5+7;
        b=3*4*5-4;
        printf("a=%d\n b=%d\n",a,b);
        return 0;
}
```

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

int main() {
        int a,b;
        a=2;
        b=a++;
        printf("a=%d\n b=%d\n",a,b);
        b+=a+2;
        a+=b--;
        printf("a=%d\n b=%d\n");
        return 0;
}
```

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

int main() {
        int a,b; float c;
        a=2; b=3; c=1.5;
        d=a+b+c;
        a=d;
        c+d=a-b;
        printf("a=%d ; b=%d \n",a,b);
        printf("c=%d ; d=%f \n",c,d);
        return 0;
}
```

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

int main() {
        float a,b,c;
        a=1.6;
        b=2*a+0.8;
        c=b/3;
        printf("a=%.2f \n",a);
        printf("b=%.2f \n",b);
        printf("c=%.2f \n",c);
        return 0;
}
```

Exercice 4 Soient les déclarations suivantes :

```c
int n=5, p=9;
int q;
float x;
```

Quelle est la valeur affectée aux différentes variables par chacune des instructions suivantes :

```c
q=n<p;
q=n==p;
q=p%n+p>n;
x=p/n;
x=(float)p/n;
x=(p+0.5)/n;
x=(int)(p+0.5)/n;
```

Exercice 5 Que calcule et affiche ce code ?

```
1  int i,s;
2  s=0;
3  for (i=0;i<10;s+=i)
4  {
5    i=i+1;
6  }
7  printf("La valeur de s est : %d", s);
```

A FAIRE SUR MACHINE

Consignes générales

Commencez par faire du ménage... Créez un répertoire dans votre dossier personnel réservé à vos exercices de programmation (que vous appellerez par exemple TPC++), dans lequel vous aurez un répertoire par TP (par exemple un répertoire TP1 pour le premier TP, etc..). Chaque programme (fichier source) devrait avoir un en-tête avec en commentaire des renseignements utiles (auteur, date, numéro et but de l'exercice,...)

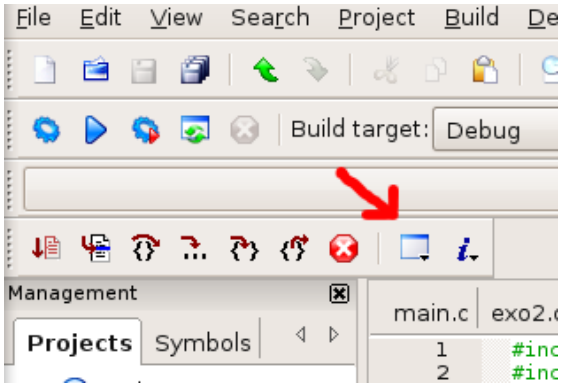
Découverte de Code::Blocks

Exercice 6 (Première utilisation) Lancez l'environnement de programmation `Code::Blocks`. Cet environnement, comme beaucoup d'autres, fonctionne par projets. Il faudra donc dorénavant créer un projet pour chaque exercice. Nous allons voir dans un premier temps la démarche à suivre pour créer un projet prêt à l'emploi (à utiliser donc lorsque vous avez tous les programmes à saisir vous-même).

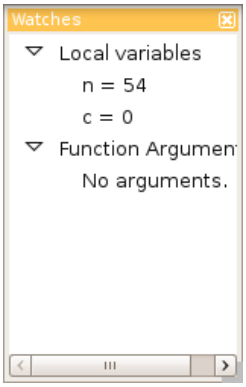
Cliquez sur **File** puis **New** et enfin **Project**. Dans la fenêtre qui s'ouvre choisissez **Console Application**, et laissez vous guidez. Pour le choix du langage, choisissez *C*. Donnez un nom à votre projet (par exemple **exo1**), vérifiez son emplacement (dans notre exemple il doit être dansle dossier **TPC++/TP1** et validez. Laissez les options par défaut dans les fenêtres suivantes. Un sous-répertoire de même nom que le projet est automatiquement créé. Cliquez enfin sur **Finish**.
Observez attentivement l'environnement Code::Blocks. Sur la gauche, vous avez la fenêtre **management**, où vous pouvez gérer vos différents projets en cours. Pour l'instant, vous n'avez qu'un seul projet, exo1. Dans les sources de ce projet, il n'y a pour l'instant qu'un fichier généré automatiquement **main.c**. Double-cliquez dessus. Le fichier s'ouvre dans la fenêtre principale.
Vous allez compiler le projet en cliquant sur l'icône en forme d'engrenage jaune en haut, ou en sélectionnant **Build** dans le menu **Build**, ou encore en tapant **Ctrl-F9**. Dans la fenêtre du bas, vous pouvez observer les messages donnés par le compilateur.
Exécutez maintenant le projet en cliquant sur l'icône suivant en forme de triangle vert, ou en sélectionnant **Run** dans le menu **Build**, ou encore en tapant **Ctrl-F10**.
Remarque : on peut compiler puis exécuter avec une seule commande (**Build and run**, ou F9, ou la troisième icône avec l'engrenage et le triangle).
Une nouvelle fenêtre (une console) s'est ouverte dans laquelle s'est exécuté votre premier programme. Il a simplement affiché Hello world !. Un message vous invite à appuyer sur une touche pour terminer l'exécution et revenir dans Code::Blocks.
Modifiez le programme pour qu'il affiche non plus "Hello world" mais "1+1=" suivi du résultat de l'opération, calculé par le programme. Recompilez et exécutez à nouveau.
Remarque importante : Afin d'éviter d'éventuels désagréments lors d'un crash de la machine ou du programme, habituez vous à sauvegarder très souvent votre travail. Pour cela allez dans **File ¿ Save file**, ou bien (plus simple) tapez **Ctrl-S**. Notez que le programme sauvegarde vos fichiers à chaque compilation. Lorsque le fichier que vous avez au centre a été modifié mais pas sauvegardé, une astérisque apparaît devant son nom en haut.
Avant de passer à l'exercice suivant, faites un clic droit sur le nom de votre projet et sélectionnez Clean. Cela effacera les fichiers exécutables et objet et libérera ainsi de l'espace disque.
Enfin, vous pouvez fermer le projet en cliquant droit sur son nom et en sélectionnant **Close project**. Cela n'est pas obligatoire mais avoir plusieurs projets ouverts simultanément peut être troublant.

Exercice 7 Parfois (voire assez souvent en TP) vous aurez à recopier un fichier déjà saisi pour commencer un projet. Dans ce cas vous n'avez pas besoin du fichier main.c, il constitue même une gêne. La création du projet va donc varier légèrement.
Créez un nouveau projet et choisissez **Empty Project**. Ensuite, nommez le exo2 comme précédemment, et continuez jusqu'à **Finish**. Comme auparavant, un nouveau répertoire a été créé.
Copiez le fichier exo2.c (disponible à l'adresse http ://www.lirmm.fr/~watrigant/files/cplusplus/tp1/exo2.c) dans ce nouveau répertoire (dans mon exemple **TPC++/TP1/TP1/exo2/**). Ajoutez ensuite le fichier au projet. Pour cela faites un clic droit sur le nom du projet et sélectionnez **Add files**, puis le fichier copié et **Select All** pour qu'il soit bien ajouté à l'exécutable.
Nous allons commencer par rendre le programme lisible, c'est à dire :
– une seule instruction par ligne
– en respectant l'indentation (on décale à droite quand on ouvre un bloc, et à gauche à sa fermeture)
Vous pouvez le faire soit manuellement, soit en sélectionnant **Source code formatter** dans le menu **Plugins**.
Compilez. Que constatez-vous ? Corrigez les erreurs une par une en interprétant les messages d'erreur obtenus, sauvegardez et recompilez, jusqu'à disparition des erreurs.

Exercice 8 (Débuggueur) Créez un nouveau projet et choisissez **Empty Project**. Ensuite, nommez le exo3 comme précédemment, et continuez jusqu'à **Finish**. Comme auparavant, un nouveau répertoire a été créé.
Copiez le fichier exo3.c (disponible sur l'ENT dans le répertoire Fichiers_TP) dans ce nouveau répertoire (dans mon exemple à l'adresse http ://www.lirmm.fr/~watrigant/files/cplusplus/tp1/exo3.c). Ajoutez ensuite le fichier au projet. Pour cela faites un clic droit sur le nom du projet et sélectionnez **Add files**, puis le fichier copié et **Select All** pour qu'il soit bien ajouté à l'exécutable.
Ce programme, un peu plus compliqué, calcule le nombre de terme de la suite de Syracuse nécéssaire pour arriver à 1 en partant de 54. Compilez et exécutez le. A l'exécution, vous voyez qu'il a fallu 112 étapes.
Essayez avec comme valeur de départ 97, 703, 871.
Malheureusement, nous ne voyons pas les différents calculs qui ont été effectués pour arriver au résultat. Nous allons utiliser le **débuggueur** pour observer le comportement du programme et l'évolution des variables au cours du temps.
Dans l'écran principal, cliquez juste à coté du 10. cela doit vous faire apparaître un gros point rouge. C'est un point d'arrêt. cela signifie qu'on va pouvoir dire au programme de s'arrêter à cet endroits précis lors de l'exécution.
Compilez le programme. Nous allons l'exécuter en mode debug. Pour cela, dans le menu **Debug**, cliquez sur **Start** ou bien appuyer sur F8. Une fenêtre s'ouvre, mais le programme est stoppé. Revenez sur Code::Blocks sans fermer cette fenêtre.
Cliquez sur l'icône **debugging windows** comme indiqué sur l'image suivante :



Dans la fenêtre en sur-impression qui s'ffiche, choisissez **watches**. Une fenêtre comme la suivante apparaît :



Cette fenêtre affiche l'état des différentes variables (également les paramêtres d'appel de fonctions, comme nous le verrons plus tard). Dans l'état actuel du programme, n vaut 54 et c vaut 0. Nous allons demander au programme de s'exécuter ligne par ligne en cliquant sur l'icône suivante :

Observez l'évolution des variables en répétant l'opération un grand nombre de fois. Vous pouvez également cliquer sur l'icône la plus à gauche de la précédente pour demander de continuer l'exécution jusqu'au prochain point d'arrêt. Quand vous en aurez assez, cliquez sur **stop debugger** ou bien sur sur la croix blanche sur fond rouge dans les outils de débuggage.
Essayer de modifier le programme pour afficher à chaque étape de la boucle l'étape ou en est le programme ainsi que la valeur de n atteinte. Faites tourner votre programme avec les différentes valeurs proposées au dessus.

Exercice 9 Créez un nouveau projet de type console. Nommez le exo4. Récupérez le fichier **lecture.h** (disponible à l'adresse http ://www.lirmm.fr/~watrigant/files/cplusplus/lecture.h).
et copiez le dans le répertoire de votre exercice 4. Avec le bouton droit de la souris, ajouter le fichier (Add files). Grâce à ce fichier, vous pouvez disposer maintenant de fonctions permettant de demander à l'utilisateur une valeur entière ou réelle. Pour cela, il faut ajouter au début de votre fichier main.c la ligne suivante : `#include "lecture.h"` Modifiez le programme principal en ajoutant deux variables entières `(int) a et b`. Vous pourrez leur affecter une valeur avec les instructions suivantes :

```
a=lire_entier();
b=lire_entier();
```

Faites en sorte que votre programme en calcule la somme et le produit puis affiche ces différentes valeurs. Maintenant, reprenez l'exercice 3 et modifiez votre programme pour que l'utilisateur rentre lui-même une valeur lors de l'exécution.

Exercice 10 En utilisant le fichier **lecture.h**, écrivez un programme qui demande de saisir 2 entiers au clavier et qui les stocke dans 2 variables (a et b par exemple). Affichez à l'écran l'adresse des variables a et b ainsi que leur valeur. Ensuite, échangez les valeurs de a et de b et refaite le même affichage (adresse et valeur). Qu'observez vous ?

Exercice 11 En utilisant une boucle `while`,

1. écrire un programme qui lit des entiers positifs. Le programme s'arrête dès qu'un négatif est saisi. Il affiche alors le nombre d'entiers positifs qui ont été saisis.
2. modifier le programme précédent pour calculer la moyenne des valeurs saisies.

Exercice 12 Ecrire une **fonction** `FahrenheitToCelcius` qui prend en entrée un entier représentant une température en degré Fahrenheit et retourne sa conversion en degré Celcius (en float). On rappelle la formule.

$$C = \frac{5}{9}(F - 32)$$

Testez votre fonction avec des températures saisies par l'utilisateur, et ecrivez ensuite le une fonction qui fait l'opération inverse, et testez la également.
Il se peut que votre programme ne calcule pas le bon résultat (toujours 0). Si tel est le cas, pensez à vérifier l'expression représentant votre calcul en gardant en tête qu'un entier est un entier et que le calcul avec des entiers renvoie forcément un entier...

Exercice 13 Ecrire une fonction `isPerfect` qui prend en entrée un entier, et affiche si cet entier est un nombre parfait ou non. Cette fonction ne retourne donc rien.

Exercice 14 Pour calculer la racine carrée d'un nombre réel a, il est possible d'utiliser l'algorithme de Héron d'Alexandrie. Cette méthode est la suivante. Supposons que l'on cherche la racine x de a ($x^2 = a$). Nous avons donc $2x^2 = x^2 + a$, on en déduit $2x = x + \frac{a}{x}$. Et finalement, on trouve que x est solution de $x = \frac{x}{2} + \frac{a}{2x}$. Nous pouvons donc construire la suite suivante :

$$x_0 = a$$
$$x_{i+1} = \frac{x_i}{2} + \frac{a}{2x_i}$$

Écrire une fonction qui prend en entrée un entier a et un entier n, et qui calcule et affiche les termes de la suite jusqu'a ce que la valeur absolue de la différence entre deux termes consécutifs soit plus petite que $\frac{1}{n}$. La fonction renvoie la valeur de la racine carrée ainsi calculée. Dans le main, comparez le résultat donné avec la valeur renvoyée par la fonction `sqrt` disponible dans `math.h`.