École Centrale de Nantes

Algorithmique et Programmation

TP1
Initiation C++

Lorsque vous écrivez du code, sachez qu'une documentation abondante est présente en ligne, entre autres sur le serveur pédagogique.
Donc avant de poser une question aux chargés de TP : **LISEZ LA DOCUMENTATION C++ !**
Voici un des sites contenant cette documentation `http://www.cplusplus.com`.
La documentation de CodeBlocks est disponible sur `http://www.codeblocks.org`

FICHIERS À RENDRE DANS L'ARCHIVE :

- `ADeboguer.cpp`
- `mon1erprog.cpp`
- `lancerDes.cpp`
- `nombreCache.cpp`
- `rebug.cpp`
- `rapport_TP1.pdf`

1 Linux et terminal

Connectez-vous sous la machine virtuelle `VMlinux`. Ouvrez une fenêtre de commande (parfois aussi appelée Terminal, Console ou autre) avec l'icône *LXTerminal*.

Testez des commandes `linux` que vous trouverez dans les documents en ligne sur le serveur pédagogique https://spip.pedagogiev2.ec-nantes.fr/ et en particulier les commandes :	`ls`
	`pwd`
	`man gcc`

Pour quitter une page de manuel ouverte avec `man` :	`q`

La plupart des commandes sont mnémotechniques, par exemple la commande `cd` signifie change directory, la commande `ls` list, la commande `mkdir` make directory, etc.

2 Manuel rapide de CodeBlocks

- Ouvrez l'éditeur de code : CodeBlocks (vous pouvez utilisez un autre éditeur si vous le souhaitez)
- Pour créer un nouveau projet (*File→New...→Project...*) avec le modèle *Empty Project*, et indiquez bien le nom puis l'emplacement du projet (par exemple, votre dossier "algpr/tp1"),
- Pour ajouter un fichier vide à votre projet (*File→New...→Empty File*) avec un nom explicite, un emplacement identique au projet et l'extension `.cpp`, par exemple `helloworld.cpp`, ceci vous permet de bénéficier de la coloration syntaxique,
  Remarque : Avec l'extension `.cpp`, l'éditeur reconnait un programme `C++`.
- Pour utiliser le formatage automatique : *Plugins→Source code formatter ...*
- Pour lancer la compilation : *Build→Build*,

— Pour corriger les erreurs :
  — Les erreurs de compilation sont regroupées dans la fenêtre *Build messages* sous l'éditeur.
  — Une erreur est signalée avec le fichier, le numéro de la ligne et une indication de l'erreur.
  — Vous pouvez accéder directement à l'erreur en cliquant dessus.
— Pour lancer l'exécution de votre programme : *Build→Run*,
— Pour trouver des erreurs d'exécution, reprenez votre code ligne par ligne pour les détecter.

3 Analyse de code source – Débogage

— Créez un nouveau projet "debogage" dans le dossier "algpr/tp1"
— Téléchargez le fichier `ADeboguer.cpp`, disponible sur le serveur pédagogique, et déplacez-le dans le dossier du projet debogage "algpr/tp1/debogage" (avec le navigateur de fichier ou la commade `mv` du terminal),
— Ajoutez ce fichier au nouveau projet (*Click droit sur le nom du projet→Add Files...*),
— Compilez le fichier et corrigez toutes les erreurs. Il est conseillé de corriger les erreurs une par une, **en commençant par la première de la liste**, car les erreurs peuvent avoir des effets de bord et produire d'autre erreurs.
— Lorsque vous avez corrigé toutes les erreurs de compilation. Exécutez le programme produit, testez le avec plusieurs valeurs [-1, 0, 1, 2, 3] et corrigez au besoin des erreurs d'exécution.

4 Modification de code source

Vous trouverez ci-dessous un algorithme pour calculer n termes de la suite de Fibonacci. Le programme est déjà écrit, dans le fichier `mon1erprog.cpp`, disponible sur le serveur pédagogique.

problème : Le programme mon1erprog permet de calculer n termes de la suite de Fibonacci (suites définies par récurrence $u_{n+1} = u_n + u_{n-1}$) obtenue à partir de $u_0 = 1$ et $u_1 = r$ avec $r = (1-\sqrt{5})/2$. La valeur r est telle que $r \in]-1, 0[$ et $r^2 = r + 1$. On peut montrer (par récurrence) que pour tout n, on a $u_n = r^n$, d'où la convergence théorique de la suite (u_n) vers 0. Le but du programme est de mettre en évidence l'effet de la propagation des erreurs d'arrondi dans le calcul par récurrence de la suite (u_n).

```
constante
    réel r

algorithme
variables
    entier n
    réels uprec, ucour, usuiv / pour u_{i-2}, u_{i-1}, u_i/
début
    / initialisations /
    n ← lire( ) /lecture de n au clavier/
    uprec ← 1
    ucour ← r
    / calcul de u_2, ..., u_n /
    pour i ← 2 à n faire
        usuiv ← ucour + uprec
        uprec ← ucour
        ucour ← usuiv
        / affichage de u_i calculé par récurrence, et de u_i calculé directement par sa valeur r^i /
        écrire(i,ucour,r^i)
```

fin pour
fin

4.1 Compilation et Modification du programme

— Créez un nouveau projet et ajouter ce fichier au nouveau projet,
— Compilez et lancez l'exécution,
— **Faites les modifications indiquées en commentaire dans le code source,**
— Sauvegardez et lancez la compilation,
— Déboguez au besoin (modification et compilation),

4.2 Analyse des résultats

- jeux d'essais
Exécutez le programme pour plusieurs valeurs de n. Retenez une valeur de n pour laquelle l'observation des résultats est intéressante.
- analyse des résultats
Le programme calcule bien des valeurs de r^n qui s'approchent de 0. En revanche, les valeurs de u_n calculées par récurrence divergent assez rapidement. Cela vient de la propagation de l'erreur d'arrondi ε faite par l'ordinateur dans le calcul de r. En effet, si on note $\tilde{u}_n$ les valeurs calculées, on obtient : $\tilde{u}_1 = u_1 + \varepsilon$, $\tilde{u}_2 = u_2 + 2\varepsilon$, $\tilde{u}_3 = u_3 + 3\varepsilon$, $\tilde{u}_4 = u_4 + 5\varepsilon$, $\tilde{u}_5 = u_5 + 8\varepsilon$, $\tilde{u}_6 = u_6 + 13\varepsilon$, ... $\tilde{u}_n = u_n + a_n\varepsilon$,... avec a_n qui tend vers l'infini.
— Commentez les résultats obtenus dans votre rapport en PDF.

5 Traduction d'algorithme

Traduisez l'algorithme donné dans le paragraphe 5.1 dans un fichier `lancerDes.cpp`. **Commencez le programme par un en-tête de présentation contenant vos noms, la date, le nom du fichier et une brève description de son contenu.** Compilez, testez ce programme et donnez vos commentaires dans le rapport.

5.1 Algorithme de l'exercice du calcul de fréquences

problème : Pour une suite de lancers de dés, calculer les fréquences d'obtention de chaque valeur. La suite se terminera par une valeur autre que 1 à 6.

algorithme
variables
    entiers nbLancers, nb1, nb2, nb3, nb4, nb5, nb6 /compteurs/
debut
    initialisation à 0 des compteurs nbLancers et nb1 à nb6
    valeur ← lire( ) /1ere valeur de la suite de lancers/
    tant que (valeur > 0) et (valeur < 7) faire
        /traiter la valeur lue/
        incrémenter le compteur nb1,... ou nb6 correspondant à la valeur du dé (structure choix selon)
        nbLancers ← nbLancers+1
        /lire la valeur suivante/
        valeur ← lire( )
    fin tant que
    si (nbLancers ≠ 0) alors

```
        /calcul des fréquences/
        f1 ← nb1/nbLancers ; ... ; f6 ← nb6/nbLancers
        écrire(f1,f2,f3,f4,f5,f6)
    sinon
        écrire("aucune valeur entre 1 et 6 !")
    fin si
fin
```

Attention ! L'algorithme précédent est peut-être ambigu ou erroné, c'est à vous de l'interpréter pour faire un programme efficace.
Remarque : La division d'un entier par un entier est une division entière. L'utilisation du transtypage explicite vous sera peut-être utile pour transformer ponctuellement le type d'une variable. Vous pouvez chercher `C-style cast` dans la documentation.

5.2 Évolution de l'algorithme

Faites évoluer votre programme pour que l'on puisse utiliser un dé à 100 faces (D100). Écrivez l'algorithme de votre programme dans le rapport et modifiez le code de lancerDes.cpp.

6 Écriture d'algorithme et de programme

6.1 Jeu du nombre caché

Un nombre entier est initialisé au hasard et l'utilisateur doit proposer des valeurs pour le trouver. Tant qu'il ne trouve pas la valeur du nombre caché, le programme affiche si celle-ci est plus grande ou plus petite que la valeur proposée.

- Écrire l'algorithme d'un programme qui simule ce jeu et dit en combien de propositions le nombre a été trouvé,
- Traduire en C++ dans un fichier `nombreCache.cpp`,
- Écrire l'algorithme et l'analyse des résultats dans votre rapport.

Pour générer des nombres aléatoires, utilisez la fonction `rand` de la bibliothèque **cstdlib**. Cette fonction renvoie un entier entre 0 et `RAND_MAX` (constante qui vaut 2 147 483 647). Au préalable, initialisez la série de nombres utilisés avec la fonction `srand` de la bibliothèque **cstdlib** et la fonction `time` de la bibliothèque **ctime** : `srand(time(NULL))`.

Pour aller plus loin : Consulter la documentation de la librairie **random**.

7 Rebug party

Dans les exercices précédent, vous avez débogué un programme, tâche usuelle de tout programmeur. Les informations relatives aux bogues apparaissent dans l'onglet *Build Messages* sous la forme de trois colonnes :
`fichier | numéro de ligne | message`
Les messages peuvent être de type `error`, `warning` ou `note`.

- `error` : erreur à corriger pour réussir la compilation,
- `warning` : avertissement dont est préférable de tenir compte, mais sans que ce soit obligatoire,
- `note` : information qui suit une erreur ou un avertissement pour préciser le contexte ou suggérer une correction.

Toutefois, afin de bien comprendre comment de tels bogues pouvaient survenir, vous allez maintenant ajouter des bogues dans le programme `rebug.cpp`, afin de produire le résultat de compilation suivant, dans l'ordre, et avec les numéros de lignes :

```
||=== Build: Debug in TP1 (compiler: GNU GCC Compiler) ===|
/path/rebug_bug.cpp|15|error: expected ';' before 'const'|
/path/rebug_bug.cpp||In function 'void lecture(float*)':|
/path/rebug_bug.cpp|38|error: return-statement with a value, in function returning 'void'
                [-fpermissive]|
/path/rebug_bug.cpp||In function 'float moyenne(float*, int)':|
/path/rebug_bug.cpp|47|error: 'nbe' was not declared in this scope|
/path/rebug_bug.cpp|47|error: expected ';' before 'i'|
/path/rebug_bug.cpp||In function 'int main()':|
/path/rebug_bug.cpp|64|error: too many arguments to function 'void lecture(float*)'|
/path/rebug_bug.cpp|19|note: declared here|
/path/rebug_bug.cpp|71|error: expected ';' before '}' token|
/path/rebug_bug.cpp|76|error: 'EXIT_SUCCESS' was not declared in this scope|
/path/rebug_bug.cpp|77|error: expected '}' at end of input|
||=== Build failed: 8 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===|
```