MOOC Init. Prog. C++
Exercices supplémentaires facultatifs semaine 4

Fonctions simples (niveau 1)

Écrivez les fonctions suivantes, et testez-les en les appelant dans la fonction `main` de votre programme sur des exemples:

1. Écrivez une fonction `min2` qui reçoit deux arguments de type `double` et retourne le plus petit d'entre eux. Le type de retour devra donc être `double`.
2. Écrivez une fonction `min3` qui prend trois arguments de type `double` et retourne le plus petit d'entre eux. Comment utiliser la fonction `min2` du point précédent pour écrire le corps de `min3` en une ligne ?

Sapin (niveau 2)

On vous donne les deux fonctions suivantes:

```
void etoiles(int nb_etoiles)
{
  for(int i(0); i < nb_etoiles; ++i) {
    cout << '*';
  }
}

void espaces(int nb_espaces)
{
  for(int i(0); i < nb_espaces; ++i) {
    cout << ' ';
  }
}
```

1. Utilisez ces fonctions pour écrire une fonction qui affiche un triangle d'étoiles, et qui prend en paramètre le nombre de lignes du triangle:

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

2. Utilisez cette fonction pour afficher un sapin:

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

   Vous devrez modifier un peu la fonction écrite au point 1. afin que votre sapin ressemble à celui dessiné ci-dessus.
3. Le sapin précédent n'est pas très joli. Il est bien trop allongé et ne ressemble pas beaucoup à un vrai sapin. Adaptez votre code pour qu'il affiche le graphique ci-dessous:

```
   *
  ***
 *****
  ***
 *****
*******
 *****
*******
*********
   |||
```

Calcul approché d'une intégrale (niveau 2)

Cet exercice correspond à l'exercice n°15 (pages 33 et 214)
de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

On peut montrer que pour une fonction suffisamment régulière (disons C-infinie), on a la majoration suivante :

$$\left| \int_a^b f(u)\,\mathrm{d}u - \frac{b-a}{840}\left[41f(a) + 216f(\frac{5a+b}{6}) + 27f(\frac{2a+b}{3}) + 272f(\frac{a+b}{2}) + 27f(\frac{a+2b}{3}) + 216f(\frac{a+5b}{6}) + 41f(b)\right]\right| \leq M_8 \cdot (b-a)^9 \frac{541}{315 \cdot (9!) \cdot 2^9}$$

où M_8 est un majorant de la dérivée huitième de f sur le segment [a,b].

Ecrivez un programme calculant la valeur approchée d'une intégrale à l'aide de cette formule, c'est-à-dire par

$$\frac{b-a}{840}\left[41f(a) + 216f(\frac{5a+b}{6}) + 27f(\frac{2a+b}{3}) + 272f(\frac{a+b}{2}) + 27f(\frac{a+2b}{3}) + 216f(\frac{a+5b}{6}) + 41f(b)\right]$$

Pour cela écrivez 3 fonctions :

1. Une fonction `f` de votre choix, qui corresponde à la fonction dont vous souhaitez calculer l'intégrale.
   (Essayez plusieurs cas avec x^2, x^3, ..., sin(x), 1/x, etc. Il faudra bien sur recompiler le programme à chaque fois.)
   (Pour utiliser les fonctions mathématiques, n'oubliez pas d'ajouter « `#include <cmath>` » en début de programme.)
2. Une fonction `integre` qui, à partir de deux arguments correspondant à *a* et *b*, calcule la somme ci-dessus pour la fonction `f` ;
3. Une fonction qui demande à l'utilisateur d'entrer un nombre réel. S'en servir pour demander les bornes *a* et *b* de l'intégrale.

Utilisez ces fonctions dans le `main()` pour réaliser votre programme.

Note : Vous pourrez trouver ici une démonstration de la formule [lien externe] donnée plus haut (Attention ! Dans la page en question $h = (b-a)/n$, ici $(b-a)/6$: 6*140 = 840).

Fonctions logiques (niveau 3)

On vous donne la fonction non_et :

```
bool non_et(bool A, bool B)
{
  return not(A and B);
}
```

qui renvoie la valeur de NON(A ET B). Comment écrire le corps des fonctions correspondant aux 3 opérateurs logiques NON, ET et OU:

```
bool non(bool A)

bool et(bool A, bool B)

bool ou(bool A, bool B)
```

en utilisant UNIQUEMENT la fonction `non_et`, sans utiliser de `if`, ni d'opérateurs logiques `or`, `and`, `not`, `||`, `&&`, ou `!`.

Commencez par écrire la fonction `non` à l'aide de `non_et`. Vous pouvez alors utiliser la fonction `non` en plus de `non_et` pour écrire la fonction `et`. La fonction `ou` peut s'écrire en utilisant les fonctions `non` et `et`.

Recherche dichotomique (fonctions récursives, niveau 2)

Cet exercice correspond à l'exercice n°34 (pages 83 et 256)
de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Pour illustrer l'utilisation de la récursivité dans la recherche d'un élément dans une *liste ordonnée*, nous allons jouer à un petit jeu :

Pensez très fort à un nombre entre 0 et 100...
voilà !
L'ordinateur doit maintenant le trouver...

Principe

Pour ce faire, il pourrait énumérer tous les nombres les uns après les autres jusqu'à avoir la bonne réponse.
Mais, vous en conviendrez aisément, ce n'est pas une méthode très efficace...

Une autre méthode (plus efficace) est la méthode dite « par dichotomie » : on réduit le champ de recherche à chaque étape et on recommence la recherche sur le nouvel intervalle.

On commence avec la zone complète, puis on choisit un *pivot* (point central de la zone de recherche). En fonction de la comparaison de ce pivot avec l'élément recherché, plusieurs cas de figures peuvent se présenter:

- on a trouvé le bon élément -> il suffit de le retourner
- le pivot est plus grand que l'élément recherché -> on recommence la recherche sur le domaine délimité par le pivot (à gauche)
- le pivot est plus petit que l'élément recherché -> on recommence la recherche sur le domaine débutant par le pivot (à droite)

Mise en pratique

Dans le fichier `dichotomie.cc`

1. Prototypez et définissez la fonction :

   ```
   unsigned int cherche(unsigned int borneInf, unsigned
   int borneSup);
   ```

   de sorte qu'elle effectue la recherche dans l'intervalle [`borneInf`, `borneSup`]

   Cette fonction devra choisir un pivot au milieu de l'intervalle (ou s'arrêter, avec un message d'erreur, si l'intervalle est vide), proposer le pivot à l'utilisateur, puis en fonction de sa réponse (<, > ou =) s'appeler elle-même sur le nouvel intervalle ainsi déterminé, ou arrêter la recherche.

   La fonction retourne la valeur trouvée (ou la borne inférieure en cas d'erreur).

2. Dans la fonction `main`, demandez à l'utilisateur de choisir (dans sa tête) un nombre entre 1 et 100 (définissez deux constantes `MIN` et `MAX`).
   Cherchez la solution à l'aide le la fonction `cherche` puis affichez le résultat trouvé.

Exemple de déroulement

```
Pensez à un nombre entre 1 et 100.

Le nombre est il <, > ou = à 50 ? <
Le nombre est il <, > ou = à 25 ? >
Le nombre est il <, > ou = à 37 ? <
Le nombre est il <, > ou = à 31 ? >
Le nombre est il <, > ou = à 34 ? <
Le nombre est il <, > ou = à 32 ? >
Le nombre est il <, > ou = à 33 ? =

Votre nombre était 33.
```

Recherche approchée de racine (niveau 2)

Cet exercice correspond à l'exercice n°42 (pages 96 et 281)
de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

On souhaite écrire un programme calculant, de façon approchée, une solution à une équation de type $f(x) = 0$.

La méthode à mettre en œuvre consiste à déterminer, à partir d'une approximation x_n de la solution, une meilleure solution x_{n+1}, et itérer ainsi jusqu'à ce que la valeur absolue de la différence entre x_n et x_{n+1} soit « suffisamment petite ».

Le calcul de x_{n+1} à partir de x_n se fait à l'aide de la formule suivante (« méthode de Newton ») :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

où f' est la dérivée de f.

Écrire un programme `reseq.cc` contenant la définition d'une fonction f (à choix) de prototype « double f(double); ».

Le programme demandera un point de départ à l'utilisateur, itérera la recherche d'une solution à partir de ce point, et affichera le résultat trouvé.

Note : Il n'est pas nécessaire d'utiliser un tableau (de taille fixe ou dynamique) pour résoudre ce problème.

Indication : Pour obtenir la valeur de la dérivée d'une fonction en un point, on s'inspire simplement de sa définition :

$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon},$$

dont on calcule une approximation pour une valeur ε donnée (par exemple 10−6) : définir la constante globale epsilon de valeur « 1e-6 », puis la fonction « double df(double x) » calculant la valeur approchée de la dérivée de f() au point x par :

```
(f(x+epsilon)-f(x))/epsilon
```

Exemple (détaillé) de déroulement

La fonction utilisée dans cet exemple est $f(x) = (x - 1.0)(x - 1.5)(x - 2.0)$:

```
Point de depart ? 1.55
au point 1.55 :
f(x) = -0.012375
f'(x) = -0.2425
nouveau point = 1.49897
```

```
au point 1.49897 :
f(x) = 0.000257739
f'(x) = -0.249997
nouveau point = 1.5
au point 1.5 :
f(x) = -2.18843e-09
f'(x) = -0.25
nouveau point = 1.5
Solution : 1.5
```
