Travaux Pratiques de programmation n°3

Cours de programmation impérative

—Licence MPI L1 S2 - Info 121—

▸ **Exercice 1.** **Mise en place pour Premier Langage**

La semaine prochaine, vous aurez une interrogation sur Premier Langage. Vous devez **impérativement vérifier par vous même** que vous pouvez y accéder. La mise en place des comptes demande un délai, nous **ne pourrons pas vous donner l'accès** le jour même de l'interrogation.

- Pour ceci, vous devez :
  — Aller sur `https://ecampus.paris-saclay.fr/`
  — Vous connecter en cliquant sur «compte établissement» puis «Paris-Sud University».
  — Une fois connecté, vérifier que vous êtes bien inscrit au cours «Info 121 MPI» (dans ce cas il apparaît sur votre page d'accueil).

- Si vous n'arrivez pas à vous connecter :
  — Cliquez sur : «Activez votre accès à eCampus via votre compte établissement en cliquant ici (création d'un compte dit "mutualisé")»
  — Sélectionnez «Paris-Sud University».
  — Activez le service «Moodle»
  — Et finalement «Valider vos choix».
  L'accès n'est pas immédiat... Vous devez IMPÉRATIVEMENT vérifier le lendemain que cela fonctionne bien et que vous avez bien accès au cours.

- Si vous arrivez à vous connecter, mais que vous n'êtes pas inscrit au cours, envoyez au plus tard lundi 25 février à `hugo.mlodecki@u-psud.fr` un courrier électronique avec comme sujet «`[info-121 ecampus]`» demandant l'inscription.

Les exercices d'entraînement devraient être disponibles à partir de vendredi 22.

ATTENTION ! La procédure d'inscription demande un délai. Si vous êtes en retard, vous prenez le risque de ne pas pouvoir vous connecter ce qui entraînera automatiquement la note de 0, sans rattrapage possible.

Fonctions et procédures

Cette séance de travaux pratiques est dédiée à l'écriture et l'utilisation de fonctions simples.
Voici quelques exemples de fonctions et procédures de la bibliothèque standard de `C++` :

prototype de la fonction	fichier	description
`int abs(int j)`	`cstdlib`	valeur absolue entière
`float fabs(float x)`	`cmath`	valeur absolue réelle
`float round(float x)`	`cmath`	arrondi à l'entier le plus proche
`float trunc(float x)`	`cmath`	arrondi à l'entier inférieur
`float pow(float x, float y)`	`cmath`	puissance réelle
`float sqrt(float x)`	`cmath`	racine carrée réelle
`float exp(float x)`	`cmath`	exponentielle réelle
`float log(float x)`	`cmath`	logarithme réel
`void exit(int e)`	`cstdlib`	quitte le programme

Pour utiliser une fonction, il faut inclure le fichier de déclaration correspondant (par exemple `#include <cmath>`).
Le `C++` ne fait pas la différence entre une fonction et une procédure : une procédure est juste une fonction qui ne retourne rien (c'est-à-dire `void`).
Voici comment on peut écrire la fonction valeur absolue :

```
float absolue(float x) {
   if (x >= 0.) return x;
   else         return -x;
}
```

▸ Exercice 2. Fonction factorielle et coefficients du binôme de Newton

La fonction pour calculer la factorielle d'un entier est donnée dans le fichier `binome.cpp`.

1. Pour tester la fonction `factoriel`, on utilise la fonction `testFactoriel`. Ajouter dans cette fonction quelques tests en dehors de la convention factoriel$(0) = 1$.
2. On appelle coefficient du binôme de Newton (ou coefficient binomial) $\binom{n}{p}$ le nombre de parties à p éléments dans un ensemble à n éléments. Par exemple :

$$\binom{0}{0} = 1, \quad \binom{3}{2} = 3, \quad \binom{4}{2} = 6$$

Le coefficient binômial $\binom{n}{p}$ peut être calculé par :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}$$

En utilisant la fonction factorielle écrite à la question précédente, compléter la fonction `binome` dans le fichier `binome.cpp`, et la tester par la fonction `testBinome`.

Dans la suite, on demande d'écrire soi-même les fonctions dont on a besoin. De plus, on commentera et testera, dans la mesure du possible, toutes les fonctions à l'aide de la macro `ASSERT` ci-dessous :

```
#define ASSERT(test) if (!(test)) cout << "Test failed in file " << __FILE__ \
                                       << " line " << __LINE__ << ": " #test << endl
```

▸ **Exercice 3. Ecriture et test d'une fonction simple.** Vous allez maintenant faire la démarche complète d'ecrire une fonction vous-même puis de l'appeler. Le but est de maîtriser la syntaxe de base de définition et d'appel d'une fonction. Vous pouvez consulter votre cours ou bien repartir de l'exemple de la factorielle.

1. Rajoutez dans le même fichier le code d'une fonction `square` qui prend en paramètre un nombre entier et calcule son carré.
2. Modifiez votre programme `main` pour tester cette fonction en l'appellant avec différentes valeurs. Faites notamment deux appels imbriqués, pour vérifier que $(3^2)^2 = 81$.

▸ **Exercice 4. (Racine carrée et n-ième)** Cet exercice est obligatoire, ceux qui ne l'ont pas fini en TP devront envoyer le corrigé par email au professeur du TP. On reprend l'exercice 2 du TP précédent. Sur les nombres à virgule (`float`), l'opérateur `==` n'est pas très utile à cause des erreurs d'arrondis. Pour résoudre ce problème, quand on veut comparer deux nombres à virgule, on teste si la valeur absolue de la différence est négligeable devant les deux nombres :

$$|x - y| \leq \epsilon|x| \qquad \text{et} \qquad |x - y| \leq \epsilon|y| \tag{1}$$

où ϵ est un très petit nombre.

1. Définir une constante `epsilon` égale à 10^{-6} (`1e-6` en `C++`) ;
2. Écrire une fonction `presqueEgal` qui prend deux nombres x et y et qui teste s'ils vérifient la condition ci-dessus, c'est à dire s'ils sont égaux avec une précision de ϵ.
3. Écrire une fonction `testpresqueEgal` que vérifie par des `ASSERT` que `presqueEgal(1, 1+epsilon/2)`, `presqueEgal(1, 1)`, `presqueEgal(1+1, 2)`, `presqueEgal(0, 0)` retournent bien vrai et que `presqueEgal(1, 1+2*epsilon)`, `presqueEgal(0, 1)` retournent bien faux.

On montre en mathématique que étant donné un réel positif a la suite

$$u_0 := a, \qquad u_{n+1} := \frac{u_n + a/u_n}{2} \tag{2}$$

converge vers $\sqrt{a}$.

4. Écrire une fonction qui prend en argument un réel a et calcule sa racine carrée en utilisant la suite définie ci-dessus. Par définition, la racine carrée est la solution positive x de l'équation $x^2 = a$. On utilisera ce test et la fonction `presqueEgal` définie plus haut pour arrêter le calcul au bon moment. Si vous en avez besoin, vous pouvez adapter la fonction `square` précédente pour qu'elle travaille aussi avec les réels (type `float`).
5. Tester cette fonction en vérifiant entre autre que $\sqrt{0} = 0$, $\sqrt{1} = 1$, $\sqrt{4} = 2$ et $\sqrt{2} \approx 1.4142135$
6. Écrire un programme qui demande un nombre positif à l'utilisateur et qui affiche sa racine carrée.

Pour calculer la racine n-ième d'un nombre, on procède de la même manière que pour la racine carrée en utilisant la suite

$$u_0 := a, \qquad u_{k+1} = \frac{1}{n}\left((n-1)u_k + \frac{a}{u_k^{n-1}}\right)$$

qui converge vers $\sqrt[n]{a}$ si $a > 0$.

7. Écrire une fonction qui calcule la racine n-ième d'un réel a. Par définition, la racine n-ième est la solution positive x de l'équation $x^n = a$. On utilisera ce test et la fonction `presqueEgal` définie plus haut pour arrêter le calcul au bon moment. On reprendra la fonction puissance du TP précédent.
8. Tester la fonction sachant que $\sqrt[5]{2} \approx 1.1486983$.

▸ Exercice 5. Exponentielle

La fonction exponentielle est définie par $\exp(a) := \sum_{i=0}^{\infty} a^i/i!$.

1. En réutilisant la fonction `presqueEgal` écrire une fonction `exponentielle` qui calcule l'exponentielle d'un nombre a. On utilisera une boucle et un accumulateur pour calculer les sommes $\sum_{i=0}^{N} a^i/i!$. On stoppe la boucle dès que deux sommes calculées consécutivement sont «presque égales».

Cette méthode n'est pas très efficace car, en utilisant les fonctions `factorielle` et `puissance`, on recalcule plusieurs fois les mêmes produits. Pour aller plus vite, on peut, dans la même boucle, accumuler la factorielle, la puissance et la somme.

2. Écrire une fonction `exponentielle2` qui fait le calcul plus rapidement en utilisant les trois accumulateurs dans la même boucle. On gardera la même condition d'arrêt de la boucle.

▸ Exercice 6. Logarithme

Pour calculer le logarithme d'un nombre positif a, on peut utiliser de la même manière que pour la racine le fait que la suite

$$u_0 := a, \qquad u_{n+1} := u_n + \frac{a}{\exp(u_n)} - 1$$

converge vers $\ln(a)$.

1. Écrire une fonction `logarithme` qui calcule le logarithme d'un nombre positif. Par définition, le logarithme de a est la solution x de l'équation $\exp(x) = a$. On utilisera ce test et la fonction `presqueEgal` définie plus haut pour vérifier que l'on a bien le résultat.
2. Écrire un programme qui vérifie que pour un nombre x, on a bien $\exp(\ln(x)) = x$.
3. Écrire un programme qui calcule $\sqrt{x}$ par la formule

$$\sqrt{x} = x^{\frac{1}{2}} = \exp\left(\frac{\ln(x)}{2}\right).$$

   Vérifier que l'on obtient bien le même résultat qu'avec la fonction de l'exercice précédent.
4. Même question pour $\sqrt[n]{x}$, avec la formule

$$\sqrt[n]{x} = x^{\frac{1}{n}} = \exp\left(\frac{\ln(x)}{n}\right).$$