



École Nationale Supérieure d'Ingénieurs de Caen

6, Boulevard Maréchal Juin
F-14050 Caen Cedex, FRANCE

TP n^o1

Niveau	3 ^{ème} année
Parcours	Électronique et Physique Appliquée
Unité d'enseignement	2E1AC2 - Architectures pour le calcul [Parallélisme]
Création	9 Novembre 2020
Responsables	Emmanuel Cagniot emmanuel.cagniot@ensicaen.fr Hugo Descoubes Hugo.Descoubes@ensicaen.fr

1 Exercice

Ce TP aborde la notion du template programming C++ en tant qu'outil d'optimisation mono-cœur.

L'archive `tp1.tar.gz` contient un squelette incomplet de l'application à réaliser. Sa structure est la suivante :

`src/include/` : contient les définitions en ligne de tous les templates demandés dans cet exercice ;
`src/testMathematics.cpp` : programme de test de tous les templates définis dans le sous-répertoire ci-dessus ;
`CMakeLists.txt` : script permettant de générer le makefile de l'application via l'utilitaire `cmake` ;
`Lisezmoi.txt` : fichier texte décrivant la procédure de génération du makefile et celle de la configuration du fichier `CMakeCache.txt` produit par `cmake`.

1.1 Question

Complétez le fichier en-tête `Pow.hpp` contenant la définition du *template* récursif `Pow`. Ce dernier permet de calculer récursivement la $n^{\text{ième}}$ puissance de l'entier e comme $e^n = e \times e^{n-1}$. Dé-commentez ensuite la zone correspondante dans le fichier source `testMathematics.cpp` afin de le tester.

Considérons deux entiers positifs a et b et r le reste de la division de a par b . Nous pouvons alors écrire :

$$a = b \times q + r, \quad r < b. \quad (1.1)$$

Tout diviseur de a et b divise également $r = a - b \times q$. Réciproquement, tout diviseur de b et r divise également $a = b \times q + r$. Par conséquent, calculer le pgcd (plus grand commun diviseur) de a et de b

revient à calculer celui de b et de r , ce qui induit un algorithme itératif (appelé algorithme d'EUCLIDE) de calcul dans lequel les restes successifs constituent une suite strictement décroissante de limite nulle. Le dernier reste non nul de cette suite représente le pgcd de a et de b .

Illustrons le fonctionnement de cet algorithme sur l'exemple suivant dans lequel $a = 42$ et $b = 72$. Nous avons :

itération	a	b	q	r
1	42	72	0	42
2	72	42	1	30
3	42	30	1	12
4	30	12	2	6
5	12	6	2	0

L'algorithme converge en cinq itérations. Le pgcd est le dernier reste donc nul de la suite, soit $r = 6$. Par conséquent, nous avons :

$$\frac{42}{72} = \frac{7}{12}.$$

1.2 Question

Complétez le fichier en-tête `Pgcd.hpp` contenant la définition du *template* `Pgcd`. Ce dernier instancie un `template PgcdImpl` qui implémente une version récursive de l'algorithme d'EUCLIDE. Dé-commentez ensuite la zone correspondante dans le fichier source `testMathematics.cpp` afin de le tester.

1.3 Question

Complétez le fichier en-tête `FracSum.hpp` contenant la définition du *template* `FracSum`. Ce dernier permet de calculer la somme de deux fractions puis de la réduire via un calcul de pgcd. Le signe de la somme est systématiquement affecté au numérateur. Dé-commentez ensuite la zone correspondante dans le fichier source `testMathematics.cpp` afin de le tester.

Le développement limité d'ordre n de la fonction exponentielle est défini par :

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \mathcal{O}(x^n). \quad (1.2)$$

1.4 Question

Complétez le fichier en-tête `Exp.hpp` contenant la définition du *template* `Exp`. Ce dernier permet de calculer, sous forme d'une fraction réduite, le développement limité de la fonction exponentielle à l'ordre n pour un argument x de type entier. Dé-commentez ensuite la zone correspondante dans le fichier source `testMathematics.cpp` afin de le tester.

Une opération couramment rencontrée dans les codes scientifiques est le produit scalaire de deux vecteurs \vec{a} et \vec{b} de taille n , opération définie par :

$$\vec{a} \cdot \vec{b} = a_0 \times b_0 + a_1 \times b_1 + \cdots + a_{n-1} \times b_{n-1}. \quad (1.3)$$

Lorsqu'un compilateur rencontre ce type de boucle (typiquement de type `for`), il ne met en œuvre la technique du déroulage que si la taille n est suffisamment significative. Dans le cas contraire, la boucle n'est simplement pas optimisée et l'*overhead* qu'elle induit dépasse alors largement le coût du produit scalaire lui-même. La seule solution est alors d'évaluer directement le produit scalaire sous la forme 1.3. Cependant, cette écriture délicate revient au programmeur d'où un gros risque d'erreur sur les composantes de nos vecteurs.

Le *template programming* permet de résoudre efficacement ce problème en transformant automatiquement le produit scalaire de deux vecteurs en une expression et ce, quelle que soit leur taille. L'idée est ici

d'exploiter une propriété des méthodes définies en ligne (*inline*) : leur appel dans le code est directement remplacé par leurs instructions. Par conséquent, il suffit d'écrire une méthode récursive dont les appels vont progressivement construire l'expression 1.3. Cette technique est mise en œuvre dans de nombreuses bibliothèques scientifiques C++ telles que POOMA, BLITZ++ ou bien encore MTL.

1.5 Question

Complétez le fichier en-tête `DotProduct.hpp` contenant la définition du *template* `DotProduct`. La méthode de classe (mot-clé `static`) définie en ligne `result` permet de calculer récursivement le produit scalaire des deux vecteurs fournis en argument, leur taille étant l'un des composants du patron de la classe. Dé-commentez ensuite la zone correspondante dans le fichier source `testMathematics.cpp` afin de le tester.