Université Pierre et Marie Curie — **Master M1 "Mathématiques et Applications"**
Période 2 - 2011/2012 — **Programmation en C/C++**

TD 5 : Premiers pas en C++, utilisation de classes

Le compilateur libre de la *GNU Compiler Collection* à utiliser en C++est `g++`. Son fonctionnement est similaire à `gcc`.

Opérateurs de flux, classe `string`

Les opérateurs de flux `<<` (injection) et `>>` (extraction) s'appliquent entre un *flux* et une variable d'un type pour lequel sont définis ces opérateurs. On peut utiliser ces opérateurs pour tous les types standards du C++.

1. *Réécrire en C++ l'exemple vu en cours qui affiche* `"Bonjour"`.
Modifier ce programme pour afficher la constante π.

L'objet `cout` que vous avez utilisé est une variable de type `ostream` (flux de sortie vers le terminal, la console). Pour écrire dans un fichier, on doit utiliser un flux de sortie vers un fichier, c'est à dire une variable de type `ofstream` définie dans le header `fstream`. Pour écrire dans un fichier, on doit donc :
– initialiser une variable de type `ofstream` par exemple : `ofstream fichier("coucou.txt");` (c'est l'équivalent de `fopen` en C),
– écrire dedans, avec l'opérateur `<<` par exemple : `fichier << "Bonjour";`
– fermer le flux en utilisant la méthode `close()` i.e. `fichier.close()` (c'est l'équivalent de `fclose` en C).
Pour la lecture sur la ligne de commande on utilise l'objet `cin` de la classe `istream`. Pour la lecture à partir d'un fichier on utilise un objet de la classe `ifstream`.

2. *Modifier le programme précédent pour écrire* `"Bonjour"` *dans le fichier* `"coucou.txt"`.

La classe `string` permet de manipuler aisément les chaînes de caractères. Les opérateurs de comparaison sont définis ainsi que l'opérateur `+` qui permet de concaténer 2 chaînes. L'opérateur d'accès `[]` qui prend un entier `i` permet d'accéder au `i`-ème caractère. De plus de nombreuses méthodes sont définies comme : `size`, `length`, `push_back` (ajoute un caractère à la fin), `insert`, `erase`, `replace`, `swap`, `find`, `rfind` etc.

3. *Ecrire un programme qui demande un entier* `n` *à l'utilisateur puis qui demande un nom de fichier et qui le stocke dans un objet* `nom` *de la classe* `string`.
On veut maintenant écrire l'entier `n` *dans le fichier* `nom`. *Attention, le constructeur de la classe* `ofstream` *prend pour argument une chaîne de caractères de type* `char *`. *Pour récupérer un tableau de caractère* `char *` *à partir de l'objet* `nom` *de la calsse* `string` *on utilise la méthode* `data()` *ie* `nom.data()`.

Nous allons maintenant utiliser la méthode `find_first_of` définie de 3 façons différentes [1] :

```
int find_first_of(const string & str, int pos = 0) const;
int find_first_of(const char * s, int pos = 0) const;
int find_first_of(char c, int pos = 0) const;
```

Le `const` qui apparaît dans ces prototypes sera définie plus tard dans le cours. Cette fonction recherche dans l'objet courant un des caractères de `str`, `s` ou `c` et renvoie la position de la première occurence. Si le paramère `pos` est spécifié, la recherche se fait à partir de l'indice `pos`. S'il n'y a pas d'occurence, la méthode renvoie la valeur `string::npos`.

4. *Ecrire une fonction qui prend une référence sur une variable* `string` *et qui remplace toutes les voyelles de cette chaîne de caractères par le caractère* `*`.

5. *Ecrire un programme qui demande une phrase à l'utilisateur et qui écrit cette même phrase avec des* `'*'` *à la place des voyelles.*

1. en fait il existe 4 définitions mais on simplifie un peu

Utilisation de la classe `complex`

La classe `complex` est définie dans le header `complex.hpp`. C'est une classe générique qui doit être instanciée avec un type spécifié. Par exemple, en déclarant une variable `c` de type `complex<double>` on indique que `c` est un objet de la classe `complex` instanciée avec le type `double`. Dans ce cas, le type utilisé pour coder la partie réelle et la partie imaginaire est `double`.
La classe `complex<double>` possède deux méthodes : `real` et `imag` qui renvoient respectivement la partie réelle et imaginaire (de type `double`), des constructeurs, des opérateurs arithmétiques, d'afectation et de comparaison. Les opérateurs de flus sont aussi définis pour cette classe. De plus les fonctions globales suivantes prennent pour argument une référence vers un `comple<double>` constant :

- `real`, `imag`, `abs`, `arg`, `norm`, `conj`, `polar`
- `exp`, `log`, `log10`, `pow`, `sqrt`
- les fonctions trigonométriques

Le constructeur de la classe prend 2 arguments : la partie réelle puis la partie imaginaire, et ces 2 arguments ont pour valeur par défaut la valeur 0.

6. *Dans un programme de test, utiliser le* namespace *standard et redéfinir le type* `complex<double>` *en* `Complex`. *Ecrire une fonction* `main` *qui contient les instructions suivantes :*

```
Complex a;
Complex b = 1.1;
Complex c(2.2);
Complex d = Complex(2, 3.1);
Complex e(4, 5);
cout << a << b << c << d << e << endl;
```

Modifier ces instructions et bien comprendre les différents appels du constructeur.

On propose de tester cette classe en construisant une fractale : un ensemble de Julia. Étant donnés deux nombres complexes, c et z_0, on définit la suite $(z_n)_{n\geq 0}$ par la relation récurrente :

$$z_{n+1} = z_n^2 + c$$

Pour une valeur donnée c, l'ensemble de Julia est formé de toutes les valeurs initiales z_0 pour lesquelles, la suite est bornée.

7. *Dans un programme de test, définir une fonction de prototype*

```
int iter(const Complex & z0, const Complex & c, double rayon, int iter_max);
```

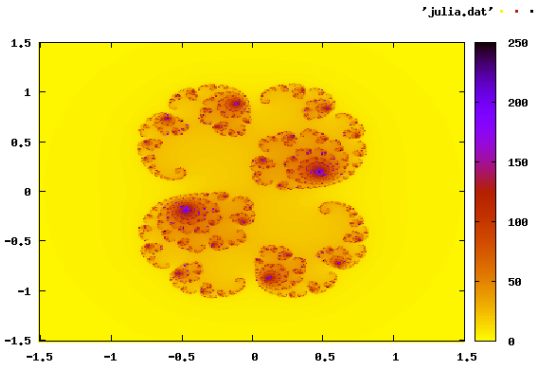
qui renvoie le premier indice `0 <= i <= iter_max` *tel que* $|z_i| >$`rayon` *ou bien* `iter_max`.

8. *Ecrire un programme qui initialise le complexe c à la valeur $0.285 + 0.01i$, qui ouvre le flux de fichier* `julia.dat` *et qui execute le pseudo-code suivant :*

```
pour x = -1.5, -1.49, -1.48,...,1.49, 1.5
    pour y = -1.5, -1.49, -1.48,...,1.49, 1.5
        z = x + yi
        k = iter(z, c, 40, 250)
        ecriture de z.real() et z.imag() et k dans "julia.dat" puis saut de ligne
    saut de ligne dans "julia.dat"
fermeture de "julia.dat"
```

Le fichier `julia.dat` contient donc 3 colonnes, dont les 2 premières sont les coordonnées du point z_0 et la troisième est le nombre d'itérations vérifiant $|z_i| \leq 40$. Pour visualiser ce fichier, vous pouvez utiliser le logiciel `gnuplot`. Une fois lancé, voici les instructions à taper dans `gnuplot` :

```
set pm3d map
set palette negative
splot 'julia.dat' w p ps 0.2 pt 5 palette
```



9. *Refaire la question précédente avec le nombre $c = (\phi - 2) + (\phi - 1)i$ avec $\phi = \frac{1+\sqrt{5}}{2}$.*
Adapter votre programme pour demander un nombre complexe c à l'utilisateur (utiliser l'opérateur `>>`*).*