

TP 5

1 Fonctions template

Considérons la procédure qui échange les valeurs de 2 variables. Le corps de cette procédure est identique qu'il s'agisse d'entiers, de réels ou de caractères (seul le prototype diffère!). Dans ce cas, le C++ offre la possibilité de définir un unique patron de fonction (template) qui pourra ensuite être instancié avec des entiers, des réels, des caractères ... (ou tout autre type compatible avec le code générique). Le mécanisme d'instanciation correspond à la génération automatique de code par le compilateur.

- Ecrivez la procédure template d'échange `mySwap` de 2 éléments.

On s'intéresse à présent à la famille des fonctions qui permettent d'identifier le minimum de deux éléments de même type.

- Ecrivez une fonction template `myMin` qui permet de retourner le minimum de deux éléments de même type.
- Modifiez le template `myMin` de manière à ce qu'il travaille sur des références sur constantes. Ainsi les éléments dont on veut connaître le minimum ne seront pas copiés.
- Spécialisez la fonction `myMin` de manière à étendre sa validité au cas de 2 chaînes de caractères (comparaison par ordre lexicographique).
- Votre code ne pourra sans doute pas fonctionner sur des chaînes de caractères stockées dans des tableaux de tailles différentes. Pour cela, vous pourrez recourir à une surcharge de votre template agissant sur des paramètres de type "référence sur la valeur de l'adresse du premier élément d'un tableau statique" (*const char(&a)[I]* où *a* est une référence sur un tableau statique de taille *I*)).

Exemple de programme utilisateur :

```
int main()
{
    std::cout << min(5,6) <<std::endl;
    std::cout << min(6,5) <<std::endl;
    std::cout << min("lili","lala") <<std::endl;
    std::cout << min("li","lala") <<std::endl; // 2 arguments de types différents
    const char * cc="mumu";
    const char * dd="ma";
    std::cout << min(cc,dd) <<std::endl;
    char ee[5]="toto";
    char ff[5]="ta"; //tableau de même taille que le précédent
    std::cout << min(ee,ff) <<std::endl;
    std::cout << min("zut",ff) <<std::endl;
    return 0;
}
```

2 Type abstrait générique : Tableau

Écrivez une classe `Tableau` (dynamique) qui soit générique sur le type `T` de ses éléments et sur la taille `AGRANDISSEMENT` des agrandissements potentiels de sa capacité : lorsque le tableau

est plein, on agrandit sa capacité de **AGRANDISSEMENT** cases). Parmi les fonctions membres et/ou amies de cette classe, on définira :

- le ou les constructeurs utiles (avec une possibilité de construction à partir d'un tableau classique, au sens C-ANSI, ie. une séquence d'éléments contigus en mémoire dont on donne l'adresse) ;
- la fonction membre **ajoute** ajoutant un élément à la fin du tableau (en l'agrandissant si nécessaire de **AGRANDISSEMENT** cases, de manière transparente à l'utilisateur) ;
- l'opérateur `[]` pour récupérer une référence sur le $i^{\text{ème}}$ élément du Tableau courant ;
- l'opérateur d'affectation et le constructeur par copie (qui sont nécessaires puisqu'une copie profonde de la structure est requise) ;
- un destructeur.

Pour cela, la classe **Tableau** stockera le nombre d'éléments effectivement insérés dans le tableau (**taille**) et la taille réservée (**capacité** actuelle). Pour que votre code soit factorisé, faites une fonction membre privée se chargeant de l'agrandissement automatique de la capacité du tableau.

Comme d'habitude, testez chaque méthode après son implémentation.

S'il vous reste du temps, ajoutez les fonctionnalités suivantes :

- des opérations d'entrées/sorties ;
- munir la classe template **Tableau** d'un compteur permettant de connaître le nombre d'instances de ce type de tableau.