

Module Base de la programmation *C++*

Travaux Pratiques 3

base du langage *C++* : vecteurs et fonctions (révision)

Exercice 3.1 *Base des vecteurs*

L'objectif de cet exercice est de se familiariser avec la structure de données des vecteurs.

Notions de cours :

En C++ les vecteurs sont des structures de données très utilisées qui permettent de stocker des éléments d'un type unique. L'usage typique est le suivant :

```
#include <vector>
...
// Déclaration d'un vecteur contiendra des éléments de type entier.
std::vector<int> v;
// Ajout d'élément :
v.push_back(42);
v.push_back(33);
// Accès au dernier élément:
std::cout << v.back() << std::endl;
// affichera 33
// Accès à un élément du vecteur:
std::cout << v[1] << std::endl;
// affichera 33
// Accès à la taille du vecteur:
std::cout << v.size() << std::endl;
```

Le type mis entre les les symboles < et > correspond au type que pourra stocker le vecteur. Dans cet exemple le vecteur peut contenir uniquement des entiers.

1. Testez l'exemple ci-dessus dans un fichier C++ en rajoutant des éléments de types entier.
2. Pouvez vous rajoutez des éléments de type string?
3. Reprendre l'exemple en ajoutant des éléments de type string.
4. Ecrire du code de façon à afficher automatiquement tous les éléments des vecteurs. Il est demandé d'utiliser à la fois le parcours classique basé sur les indices et le parcours basé sur la collection d'objets qui est autorisé depuis C++11. **Indication** : la syntaxe sur les boucles est la suivante :

```
for(auto e : v){}
```

A titre tout à fait indicatif, voilà ce que vous pouvez obtenir :

```
Exo 1 TP3      C++
-----
-----Question 1-----
dernier element (access avec back): 33
acces par indice (0): 42
taille du vecteur v: 2
-----
-----Question 3-----
dernier element (access avec back): Bonjour
acces par indice (0): Bonsoir
taille du vecteur v: 2
-----
-----Question 4-----
Parcours du vecteur par indices:
element [0]: 42
element [1]: 33
Parcours vecteur sans indice sur la collection :
element: Bonsoir
element: Bonjour
```

Exercice 3.2 Fonctions et vecteurs

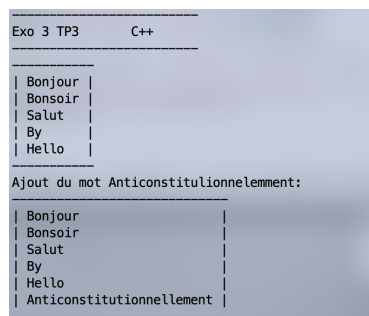
Dans cet exercice, nous allons revoir l'utilisation des fonctions en association avec les vecteurs.

1. Dans la fonction **main** de votre programme, construisez un vecteur de type **string** qui sera initialisé au moment de la déclaration par un ensemble de 5 chaînes de caractères de votre choix.
Indication : depuis C++11, il est possible d'affecter directement un vecteur en utilisant les symboles `{}` contenant les éléments séparés par des virgules.
2. Avant votre fonction **main**, ajoutez une fonction **afficheVecteur** qui prendra en paramètre un vecteur de type **string** et qui parcourra ce vecteur et affichera le contenu de chaque élément en indiquant l'indice et le contenu du vecteur.
3. Ajoutez un paramètre de type booléen (*bool*), nommé **affIndice** et qui aura pour rôle d'afficher ou non l'indice de l'élément considéré.
4. Testez la fonction précédente en l'appelant avec les différentes valeurs du paramètre d'affichage.
5. Dans la fonction, donner une valeur par défaut pour le paramètre **affIndice** et testez son bon fonctionnement.

Exercice 3.3 Fonctions et vecteurs

Dans un nouveau fichier recopiez le code de l'exercice précédant qui servira de base de cet exercice.

1. Afin d'améliorer l'affichage du vecteur, ajoutez une fonction qui calcule la largeur en nombre de caractères de la plus grande chaîne de caractères. Cette fonction devra renvoyer un entier.
2. Testez cette fonction avec votre vecteur.
3. En utilisant cette fonction, améliorez la version de l'affichage qui n'affiche pas les indices, de façon à obtenir un cadre autour des noms quelque soit les vecteurs utilisés. La séquence de test de votre fonction devra ressembler à l'image ci-dessous.



```
Exo 3 TP3      C++

| Bonjour |
| Bonsoir |
| Salut   |
| By      |
| Hello   |

Ajout du mot Anticonstitutionnellement:

| Bonjour      |
| Bonsoir      |
| Salut        |
| By           |
| Hello        |
| Anticonstitutionnellement |
```

Exercice 3.4 Passage en paramètre des vecteurs

Notions de cours :

En C++, le passage en paramètre des éléments se fait par défaut **par copie**. Il est probable que vous ayez utilisé ce type de passage dans les déclarations précédentes.

Pour que votre fonction considère le paramètre passé comme une référence, il faut juste rajouter le symbole **&** devant la déclaration du paramètre dans la définition de la fonction. Par exemple, si vous testez le code suivant, la variable **x** sera bien modifiée par le programme :

```
void
testParam(int &i){
    i = 42;
}

int main() {
    int x = 3;
    std::cout << "valeur_de_x_avant:_" << x;
    testParam(x);
    std::cout << "valeur_de_x_après:_" << x;
    return 0;
}
```

1. Testez le programme de la notion de cours précédent, et vérifiez que votre variable `x` a bien été modifiée.
2. Supprimez le symbole `&` devant le nom du paramètre et mettez en évidence le passage par copie de votre variable.
3. Reprendre les questions précédente de façon à tester le passage par référence et par copie d'un vecteur.
4. Pour mettre en avant la différence de temps d'exécution perdue à cause de la copie, utilisez vos deux fonctions de la question précédente avec en paramètre un vecteur contenant 100 000 entiers. Utilisez la librairie `std::chrono`. Cette dernière s'utilise en appelant `auto start = std::chrono::high_resolution_clock::now();` et `auto stop = std::chrono::high_resolution_clock::now();` puis en calculant la différence avec `start-stop.count()`.
5. Affichez les temps obtenus à travers le passage par référence et par copie. Vous pourrez obtenir une différence comparable à l'image ci-dessous.

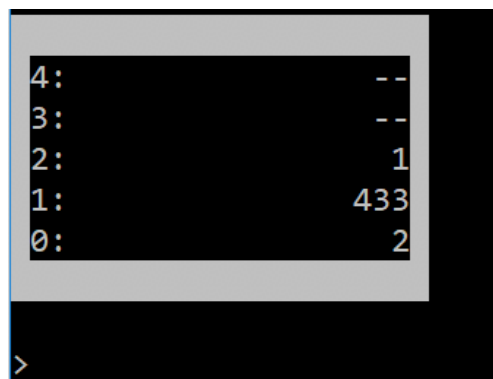
```
Temps passage par ref:791 nanosec
Temps passage par copie:2789459 nanosec
```

Exercice 3.5 *Calculatrice*

Le type de calculatrice à notation polonaise inversée fut exploité entre autre par HP (avec par exemple la calculatrice HP 48GX). Elle permet d'éviter l'utilisation de parenthèses dans les différents calculs. Basée sur l'exploitation d'une pile de type LIFO, son utilisation diffère d'une calculatrice normale puisque le calcul d'une expression nécessite d'abord le placement dans la pile puis l'application de l'opérateur considéré. Par exemple, pour calculer l'expression $(3 + 4)/5$, il faut d'abord placer 3 et 4 dans la pile, appeler l'opérateur `+` qui fera l'addition (et remplacera dans la pile les deux opérandes par le résultat) et enfin rajouter 5 dans la pile et appliquer l'opérateur `/`.

1. Décrivez les étapes pour calculer les expressions suivantes :
 - $(23 - 10)/3$
 - $((223 * 2) + 32)/(78 * 9)$
 - $((42 - x) * 23)/(30 * y)$
2. Comme pour les TP précédents, créez un nouveau projet vierge de type console C++.

La réalisation de la calculatrice peut s'effectuer de manière assez simple avec l'utilisation d'une simple pile (que l'on implémentera par un simple vecteur). Une première étape pour la réalisation de l'application sera d'afficher l'écran représentant la pile dans la fenêtre du terminal comme sur l'image suivante :



```
4:      --
3:      --
2:      1
1:     433
0:      2
>
```

3. Pour réaliser cet affichage, on vous propose d'utiliser la fonction suivante :

```

void
displayStack(const vector<double> &vect) {
    string border = "\\033[1;47m \\033[0m";
    string line = "\\033[1;47m                                \\033[0m";
    cout << line << endl;
    for (int i = 0; i < 5; i++){
        cout << border << 5-i-1 << ":";
        cout.width(18);
        if ( (i < 5) && ((vect.size()-5+i) >=0)) {
            cout << vect[vect.size() - 5 + i] << border<< endl;
        }
        else {
            cout << "--" << border << endl;
        }
    }
    cout << line << endl;
}

```

4. Recopiez rigoureusement le code ci-dessus.
5. Dans la fonction `main`, rajoutez un objet nommé `maPile` de type `vector` représentant une pile contenant trois éléments et testez cette fonction. Normalement, le code contient plusieurs erreurs et vous devez obtenir l'affichage suivant avec une erreur.

```

4:
Sortie de C:\Users\admin\Desktop\TPModuleCPP\TP2exercice1\Debug\exit.exe (processus 8796) avec le code 3.
Pour fermer automatiquement la console quand le débogage s'arrête, activez Outils->Options->Débogage->Fermer automatique
ment la console à l'arrêt du débogage.
Appuyez sur une touche pour fermer cette fenêtre.

```

Attention : oubliez pas de rajouter `#include <vector>` afin de pouvoir utiliser les vecteurs.

6. Utilisez des *breakpoints* afin de trouver l'erreur et la corriger.
7. Gérer l'empilement des valeurs dans la pile. Par exemple après le lancement de l'application vous devez pouvoir obtenir les affichages suivants : en ajoutant successivement les valeurs suivantes : 33, 12, 49 :

Indication : pour convertir un `string` en double vous pouvez utiliser la fonction `stod` qui est accessible si vous rajoutez `#include <string>`.

8. Écrire une fonction qui renvoie `true` si son paramètre est un opérateur et qui `false` sinon. On considérera les opérateurs suivants ("`+`", "`-`", "`/`", "`*`").
9. Complétez la fonction suivante qui sera à appeler pour appliquer un opérateur.

```

bool
applyOperator(vector<double>& stack, string op)
{
    // question 9:
}

```

Pour cela vous utiliserez les expressions suivantes :

- `maPile.back()` : permet de récupérer la dernière valeur entrée par l'utilisateur.

- `maPile.pop()` : permet de supprimer la dernière valeur entrée par l'utilisateur.
10. En revenant dans la fonction `main`, appeler la fonction précédente lorsque l'utilisateur entre un opérateur.