C++ Lab4 (3 classes): Vectors

General comments

- You have to test your program ***as you write it***: do not go on to the next question as long as you have not written (and run!) a set of test cases for the question you are working on.
- Even if it is not explicitly stated, it is necessary to ***immediately*** display a vector each time it is created or modified, so that its contents can be verified.
- A list of useful programming terminology is given at the end of this lab.

Exercise 1: basic operations on vectors

1. Copy from Moodle the program **football.cpp** into your working directory.
2. Extend this program to display the name of the player that is stored in the first element of the vector ***team***. Do so in 2 different ways: using its index and the function **front**. Same question for the last element (using its index and the function **back**).
3. Display the full list of players contained in the vector ***team***. **NB: this loop will be used (i.e. copied) for each question in order to check that the content of the vector is correct.**
4. Display the same list, but this time starting from the last element.

   Note: always add a message indicating what is being displayed. This will make it easier to verify the output of your program. For instance:

```
cout << "First member in the team: ";
… // your code
cout << "Last member in the team: ";
… // your code
cout << " Members of the team: ";
… // your code
cout << " Members of the team in reverse order: ";
… // your code
```

5. **Adding an element**: Extend the program to be able to **add** new players to the vector. Ask the user for the number of players to be added. New players will be added ***at the end of the vector***. Display the vector content.
6. **Searching for an element:** Extend the program to be able to **test** if a given player is stored in the vector. Ask the user for the name to be searched. The program will display "Unknown player" or "player stored in element XX of the vector" depending on the result (with XX being the index). If the player is found, then display the index and the value stored at that index to verify. What loop is necessary? See your algorithmic class notes. Use a Boolean **found** to determine if the player is stored in the vector. The player is initially not found, so initialize the Boolean to false. Then use this Boolean for the loop test, and update it inside the body of the loop.
7. **Replacing an element**: If the player was found, extend your program to be able to replace her by another player (check by displaying the contents of the vector using the loop written for question 3).
8. **Normalization**: to facilitate searches, it is preferable to have strings that are stored in a format that is normalized. In this exercise we decide that player names will be stored in upper case. Modify your program to add the statements that are necessary to convert the player name to upper case everywhere that it is needed.

Reminder:

In C++ a string can be manipulated as a vector of characters with the [] operator. For instance:

```
string name="toto";
name[0]; // first character of the string, here 't'
name[name.length()-1]; // last character of the string, here 'o').
```

We will use the function **`toupper`** to return an upper-case character (requires including the file **`cctype`**). For instance, if **`name`** is a string, then **`name[2]=toupper(name[2])`** replaces the 3rd character of the **`name`** string with that same character converted to upper case.

➔ Upload your program, questions 1-8, including test cases.

Exercise 2: vectors with real values

Note: starting at question 2 of this exercise, tests will be designed with different vectors that are initialized at their declaration.

1. Using algorithmic class notes: write a program **InputVectorReal.cpp** which inputs a sequence of real numbers and stores them in an vector named **myVector**. Data input has to be interrupted when the user inputs the value **STOP** (a constant, -999.9 for instance). This STOP value is a flag and should therefore **NOT** be stored in the vector. Display the contents of the vector once the input is finished.

   Don't forget the set of test cases, as comments in the program.

2. Comment out the code written for question 1 (but do not delete it!)

   Write a program that declares a vector **myVector** of real numbers. Initialize the vector at declaration with the values {10,15,9]. Compute and display the average of the numbers stored in **myVector**. Test your code with several vectors to constitute a set of test cases. Think of all possible cases.

3. Extend your program to display the ***index*** and the ***value*** of the largest number stored in the vector.

   Hint: using previous exercises, start by computing the ***value of the max***. Test your program. Then figure out what needs to be changed to find the ***index*** rather than the value of the max. (Do NOT store the max value in a variable. Remember that you can always access a value given its index). Use a variable **iMax** to store this index instead of a variable to store the value. Test your code with several vectors. Think of all possible cases.

   Finally, add statements to also determine the index of the minimum. Test by displaying the largest and smallest value of the vector.

4. We wish to separate into two distinct vectors, **vectorPos** and **vectorNeg**, the positive and negative elements of of **myVector**. Display the contents of these vectors to check them.

   For instance: myVector: 1.2 -2.0 13.4 14.5 -13.6
   vectorNeg: -2.0 -13.6
   vectorPos: 1.2 13.4 14.5

5. Extend the program to delete all null values from **myVector**. Note: it is not necessary to maintain the order of the elements. You can therefore replace each null value by the last value of the vector, then remove the last value. Do not forget to display the vector before and after…

   For instance: myVector: 1.0, 0, -2.2, 13.6, 0, 0, 14.5, -13.8

   After removing null values: : 1.0, -2.2, 13.6, 14.5, -13. 8

6. **Indirection**. Go back to question 4, but this time, store in two additional vectors (**indNeg** and **indPos)**, not the values but the **indices** of the negative and positive elements. Then use these vectors to display all negative values, followed by all positive values, of **myVector**.

   For instance: myVector: 1.2 -2.0 13.4 14.5 -13.6
   indNeg: 1 4
   indPos: 0 2 3
   You will then display:
   Negative values: -2.0 -13.6
   Positive values: 1.2 13.4 14.5

   This is called indirection, because you are using a vector of indices to "indirectly" display the values stored in a vector.

➔ Upload your program, questions 1-8, including test cases.

Programming Vocabulary Lab4

- vector — *vecteur*
  - An integer vector is a vector containing integers.
- index (plural : indices) — *indice*
  - A number identifying a specific element in a vector.
  - Vectors in C++ are zero-indexed, so the first element has an index of 0.
- vector element — *case d'un vecteur*
  - If myVector[0]==9; is true, then the first element of myVector contains the value 9.
- to store — *enregistrer*
  - The following statement stores the value 7 in the 3rd element of the vector myVector: myVector[2]=7;
- vector length or size — *taille d'un vecteur*
  - The number of elements stored in a vector.
- out of bound access — *accès hors limite*
  - An attempt to assign to, or acquire from, a memory location outside of the allocated vector memory.

If the vector capacity of myVector is 3, then myVector[10] constitutes an out of bound access, and may lead to a program crash