Queens College, CUNY, Department of Computer Science

Object-Oriented Programming in C++

CSCI 211/611

Summer 2018

Instructor: Dr. Sateesh Mane

due date Friday, July 13, 2018, 11.59 pm

Homework: Vectors

- **Experience with other classes has demonstrated that in many cases the source of difficulty is not the mathematics or the programming.**
- **The source of difficulty is the English (understanding the text).**
- **If you do not understand the words in the lectures or homework, THEN ASK.**
- **If you do not understand the concepts in the lectures or homework, THEN ASK.**
- **Send me an email, explain what you do not understand.**
- **Do not just keep quiet and then produce nonsense in exams.**

- **Consult your lab instructor for assistance.**
- **You may also contact me directly, but I cannot promise a prompt response.**
- Please submit your inquiry via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`.
- Please submit one zip archive with all your files in it.
  1. The zip archive should have either of the names (CS211 or CS611):

     `StudentId_first_last_CS211_hw_vectors.zip`
     `StudentId_first_last_CS611_hw_vectors.zip`

  2. The archive should contain one "text file" named "hw_vectors.[txt/docx/pdf]" and one cpp file per question named "Q1.cpp" and "Q2.cpp" etc.
  3. Note that not all questions may require a cpp file.

General information

- You should include the following header files and namespace, to run the programs below.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>          // don't forget this!
#include <cmath>

using namespace std;
```

- You may require additional header files for individual questions.
- If you require additional header files to do your work, feel free to include them.
- **Include the list of all header files you use, in your solution for each question.**
- The questions below do not require complicated mathematical calculations.
- If for any reason you require help with mathematical calculations, **ask the lab instructor or the lecturer.**

Q1

- Write a program as follows and run it and print the outputs.
- Instantiate a string with your name.

```
string name("write your name here");
```

- Instantiate a vector `vchar` of type `char`.

```
vector<char> vchar;
```

- **Print the size of the vector `vchar.size()`.** It should be zero.
- **Print the capacity of the vector `vchar.capacity()`.** It should be zero.
- Write a loop as follows.
  1. **Call `push_back` to populate the vector one character at a time.**
  2. Then print the value of `vchar[i]`, also the size, capacity, front and back.
  3. Your code should look like this

```
for (int i = 0; i < name.size(); ++i) {
  vchar.push_back( name[i] );

  // vchar[i], vchar.size(), vchar.capacity(), vchar.front(), vchar.back()
}
```

- If you have done your work correctly, you should observe the following.
  1. The size grows by one at each step.
  2. The capacity grows automatically so that `capacity >= size` always.
  3. Therefore the memory allocated for a vector changes dynamically during program execution, to have enough space to hold all the elements.
  4. The value of `front()` does not change (the first element is always the same).
  5. The value of `back()` changes at each step. It is always the last element populated in the vector.

- **See next page(s).**

- Write a second loop to force all the characters to uppercase.
  1. The upper limit of the loop is **vchar.size()**, the size of the vector.
  2. Use the function `toupper` to convert the characters to uppercase.

```
for (int i = 0; i < vchar.size(); ++i) {
  vchar[i] = toupper( vchar[i] );
}
```

- Print the elements of the vector: `for (int i = 0; i < vchar.size(); ++i)`.
- **Print the values of `size`, `capacity`, `front` and `back`.**
- If you have done your work correctly, the values of `size()` and `capacity()` are the same as before, but `front()` and `back()` are now uppercase characters.
- **Call `pop_back`.**
- Print the elements of the vector again: `for (int i = 0; i < vchar.size(); ++i)`.
- **Print the values of `size`, `capacity`, `front` and `back`.**
- If you have done your work correctly, you should observe the following.
  1. The value of `size()` decreases by one.
  2. The value of `capacity()` does not change.
  3. The value of `front()` does not change.
  4. The value of `back()` changes to the second last character in your name.
- **Clear the vector `vchar.clear()`.**
- **Print the values of `size` and `capacity` only.**
- If you have done your work correctly, you should observe the following.
  1. The value of `size()` is zero.
  2. The value of `capacity()` does not change.

Q2

- In this question, we employ a vector of integers. Call the vector `v`.
- **Let $n = 10$ and instantiate a vector `v` with size n as follows.**

```
int n = 10;
vector<int> v(n);
```

- **Print the values of `size`, `capacity`, `front` and `back` of v.**
- If you have done your work correctly, you should observe the following.
  1. The size and capacity are both 10 (i.e. `size = capacity = n`).
  2. The values of `front()` and `back()` are both 0.
  3. The compiler initializes the values of all the elements to zero.
- **Set the value of `v[i]` to the digits of your student id, for $i = 0, \ldots, 7$.**

```
v[i] = (digit of student id)     // i = 0,...,7
```

- **Print the elements of the vector: `for (int i = 0; i < v.size(); ++i)`.**
- If you have done your work correctly, you should observe the following.
  1. The values of the first eight elements are the digits of your student id.
  2. The values of the last two elements are zero (initialized by the compiler).
- **Resize the vector.**

```
v.resize(13, -1);          // resize and initialize
```

- **Print the elements of the vector: `for (int i = 0; i < v.size(); ++i)`.**
- If you have done your work correctly, you should observe the following.
  1. The values of the first eight elements are the digits of your student id.
  2. The values of the next two elements are zero (initialized by the compiler).
  3. The values of the last three elements are -1.
  4. This is because we initialized their values to -1 in the `resize` statement.
- **Print the values of `size`, `capacity`, `front` and `back` of v.**
- If you have done your work correctly, you should observe the following.
  1. The value of `size()` is 13.
  2. The value of `capacity()` is 20. *This value may depend on your compiler.*

3. The compiler (actually the run-time system) automatically allocated more memory, so that `capacity >= size`.
4. Therefore the memory allocated for a vector can be changed dynamically during program execution.

- Next declare another vector w.

```
vector<int> w;
```

- **Print the values of `size` and `capacity` of w.**
- As you should know by now, the size and capacity of w are both zero.
- **Next set $w = v$. We can copy one vector to another.**

```
w = v;
```

- **Print the elements of w: `for (int i = 0; i < w.size(); ++i)`.**
- **Print the values of `size`, `capacity`, `front` and `back` of w.**
- If you have done your work correctly, you should observe the following.

1. The values of the elements of w are equal to the values of the elements of v.
2. **The size and capacity of w are equal** (to 13, in this example).
3. *The programmers who invented the vector class had enough sense to copy only the elements up to the size of v.*

- **Next declare another vector s, but of type `string`.**

```
vector<string> s;                    // type "string"
```

- Next set $s = v$. **This will genetate a compiler error.**

```
s = v;                    // COMPILER ERROR
```

- *We cannot mix and match vectors of incompatible types. We cannot copy* `int` *to* `string`.

Q3

- **We can create a vector of vectors.** *Why not?*
- **Declare** *vv* **as a vector of vectors of type** `int`**.**
- Also declare three vectors `vec1, vec2, vec3` of type `int`.

```
vector<vector<int>> vv;              // vector of vectors
vector<int> vec1, vec2, vec3;
```

- **Populate** `vec1, vec2, vec3` **to hold the following data:**
  1. `vec1` has length 1 and holds the value $\{1\}$.
  2. `vec2` has length 2 and holds the values $\{2, 3\}$.
  3. `vec3` has length 4 and holds the values $\{4, 5, 6, 7\}$.
- **Populate** `vv` **to hold the following data:**
  1. `vv[0] = vec1`.
  2. `vv[1] = vec2`.
  3. `vv[2] = vec3`.
- **Run a nested loop to print the data values.** Do it as follows.
  1. Run an outer loop for $i = 0, \ldots,$ `vv.size()` $-$ `1`. This is obvious.
  2. **Reference to vector:** declare a temporary reference variable inside the loop.

     ```
     vector<int> &tmp = vv[i];
     ```

  3. Run an inner loop and print the values of `i, j, tmp[j], vv[i][j]` for $j = 0, \ldots,$ `tmp.size()` $-$ `1`.
  4. **Note the double subscript** `vv[i][j]`**.**
  5. A vector of vectors is effectively a two-dimensional array.
- Your overall code should look like the following.

```
for (int i = 0; i < vv.size(); ++i) {          // outer loop
   vector<int> &tmp = vv[i];                   // reference to vector
  for (int j = 0; j < tmp.size(); ++j) {       // inner loop
        //print i, j, tmp[j], vv[i][j]
close all the loops
```

- **Lesson to learn:** It is perfectly possible to have all of the following:
  1. Vector of vectors.
  2. Reference to vector.
  3. Double subscript (two-dimensional index).

Q4

- **Write a function `concat_vec` to input two vectors and return a vector with the "concatenated" elements of the two vectors.**

```
vector<string> concat_vec(const vector<string> &vs1, const vector<string> &vs2);
```

- Example:
  1. `vs1` has data { `"a", "b"` }.
  2. `vs2` has data { `"x", "y", "z"` }.
  3. Output vector has data { `"a", "b", "x", "y", "z"` }.
- Note the following:
  1. The inputs arguments are **const references to vectors.**
  2. Therefore the function cannot change them internally.
  3. **The function return value is a vector.**
  4. It is perfectly possible for the return type of a function to be a vector. *Why not?*
- **Your function must work even if one or more inputs are empty.**
- **Your function must work even if both references are bound to the same object in the calling application.**

```
vector<string> v, w;
// (populate v)
w = concat_vec(v,v);     // both input references bound to same vector
```

- Your function will be tested with the following code.

```
vector<string> v;
// (populate v)
v = concat_vec(v,v);     // all three are v
// (print result)
v = concat_vec(v,v);     // call it twice!
// (print result)
```