Assignment 4 - Vectors, Menus, & Creatures!

Due Wednesday July 20, 2011 by 11:59pm
You may not work on assignments during your lab time.

- Submit all the deliverables to the Course Management System: https://courses.cs.sfu.ca/
- The last possible day to submit is Friday July 22nd.
- This assignment is to be done individually. Do not show another student your code, do not copy code found online, and do not post questions about the assignment online. Please direct all questions to the instructor or TA:
  - For CMPT125, email cmpt-125-d2-help@sfu.ca;
  - For CMPT128, email cmpt-128-help@sfu.ca.
- See the marking guide for details on how each part will be marked.

General Requirements

All programs you submit must have good style:

- Use named constants appropriately: do not use magic numbers.
- Indent and format your code correctly.
- Correctly comment your code. Comment all functions, classes, and blocks of code.
- Conform to all C++ naming conventions discussed in class.

1. Test Score Calculator

Write the program `vectorStats.cpp` which reads in a sequence of test scores, stores them in a vector, and then does some simple processing on the values.

- First ask the user the total number of marks on the exam.
- Next, repeatedly ask the user for test scores. Stop when the user enters -1.
  - Store each score (but not the -1) in a vector.
  - Test scores may be fractional (such as 10.5).
  - You do not need to validate the input in any way.
- Finally, display the following results:
  - Display all the values which the user entered as a comma separated list. Note that there should not be a trailing comma. For example:
    ```
    38, 12, 37.5, 0, 20, 39.1, 45
    ```
  - Display the maximum value entered.
  - Display the number of passing grades. A passing grade for this program is defined to be 50% or higher (i.e., half of the maximum test points or better).
- Your solution must include functions for at least the following three parts. You can use more functions if you would like.
  - Finding the maximum value in the vector.
  - Counting the number of passing scores in the vector.
  - Displaying the comma separated list of values in the vector.
- See the sample outputs on the following page.
  - Be sure to handle the case of no test scores!
- Remember to ensure your program has good style before submitting your code.

Sample Outputs: User input bold and underlined.

```
Test Summary Calculator
-=-=-=-=-=-=-=-=-=-=-=-
What was the test total? 40
Enter the next number (-1 to end): 38
Enter the next number (-1 to end): 12
Enter the next number (-1 to end): 37.5
Enter the next number (-1 to end): 0
Enter the next number (-1 to end): 20
Enter the next number (-1 to end): 39.1
Enter the next number (-1 to end): 45
Enter the next number (-1 to end): -1

Results:
***********
Test scores:
38, 12, 37.5, 0, 20, 39.1, 45

Maximum: 45
Number of passing grades: 5
```

```
Test Summary Calculator
-=-=-=-=-=-=-=-=-=-=-=-
What was the test total? 10
Enter the next number (-1 to end): -1

Results:
***********
Test scores:
None.

Maximum: 0
Number of passing grades: 0
```

2. Creature Class

Complete the code found online in `creatureTest.cpp`. You will need to implement a class called `Creature`. We will extend this class in the next assignment for the RPG game.

At the top of the `creatureTest.cpp` file, implementing the class shown in the UML diagram to the right.

Suggestion: Work through the test functions one by one: uncomment the test code, implement the features of the class as required, then test the program to ensure what you have written works correctly so far.

Creature
- name : string - hitPoints : int
+ initialize(newName : string) : void + setName(newName : string) : void + getName( ) : string + getHitPoints( ) : int + getHitPointsMax( ) : int + takeDamage (damage : int) : void + isDead( ) : bool + healAllDamage( ) : void + displayInfo( ) : void + displayInfoShort( ) : void

Description of Creature class:

- A `Creature` is either the player, or some creature (like a dog or a dragon) in the game.
- Each `Creature` has a name, and some health (called hit-points).
- Each `Creature` has a maximum possible health (here just 25).
- A `Creature` should start with full health (25 in this case), and then lose health each time it takes damage. When its health hits 0 or less (say by taking 5, then 10, then 8, then 7 damage), it is considered dead. However, it can be fully healed (even if dead) using `healAllDamage()`.
- A `Creature` object can display its information to the screen using two display functions.

Member Function Explanation

- Call `initialize()` when the object is created. It sets name and initialize it to full health.
- `setName()`, `getName()`, `getHitPoints()` are setters and getters for the member variables.
- `getHitPointsMax()` always returns 25 (for the moment).
- `takeDamage()` subtracts `damage` (the parameter) from the Creature's current health (hit points). If the creature's health should go below 0, set it to 0.
- `isDead()` returns `true` when the creature is dead (see above).
- `displayInfo()` displays a full summary of the character's information on multiple lines; `displayInfoShort()` does a display on only one line.
- See the output and the sample test code for what each function should do.
- Notes:
  - You may (if you like) implement simple setter and getter methods as inline functions.
  - All other member functions of the class must be implemented as non-inline (i.e., outside the class definition and using the scope resolution operator ::).

- Comment each function where it is implemented: inline functions have their comments in the class definition, but non-inline functions have their comment with the implementation of the function.

Test Code Output:

```
******************************
Testing Creature Name.
******************************
Ant name: Ant
Gopher name: Killer Gopher
New ant name: Mutant Ant


******************************
Testing Creature Health and Damage.
******************************
Max health: 25
Full health: 25
5 damage: 20
Healed: 25
Is dead?: 0
0 damage: 0
Is dead?: 1
10 damage: 0
Hero full health: 25


******************************
Testing Creature initialization.
******************************
Name:   Evil Villain
HP:     25
Max HP: 25
Name:   Awesome Hero
HP:     25
Max HP: 25


******************************
Testing display functions.
******************************
- - - - - - Long Display - - - - - - - -
Character info for Evil Villain:
Health      17 / 25

- - - - - - Short Display - - - - - - - -
Evil Villain has health [17 / 25]

- - - - - - Long Display - - - - - - - -
Character info for Awesome Hero:
Health      20 / 25

- - - - - - Short Display - - - - - - - -
Awesome Hero has health [20 / 25]
```

3. User Interface

Complete the code found online in `menuTest.cpp`. You will need to implement two prompt functions and the class called `Menu`. You will use this functionality in the next assignment.

3.1 Prompt Functions

- At the top very bottom of the `menuTest.cpp` file, implement the functions for the following two prototypes:
  ```
  string promptForText(string prompt);
  bool promptForYesNo(string prompt);
  ```
- For the function which prompts for text, return an entire line of text entered by the user.
  - The string parameter is what the user is to be prompted with.
  - Hint: Do not put a `cin.ignore()` in this function. A better way is to put a `cin.ignore()` after every `cin` statement which leaves behind an extra `'\n'`. This will be mentioned for the other functions.
- For the function which prompts for a yes or no, ensure the user input is valid.
  - Return `true` if the user enters either "Y" or "Yes";
    return `false` if the user enters either "N" or "No".
  - Be case insensitive ("No" is treated the same as "no", "NO", and "nO"). Hint: use the function `toupper()` to convert a single character to upper case:
    ```
    char nowUppreCase = toupper('a');
    ```
  - If the user does not enter yes/no/y/n correctly, display an error message and repeat the question.
  - After the `cin` statement, add a `cin.ignore()` statement to clear the extra `'\n'`.
- As you implement each prompt function, un-comment the appropriate provided test code in the file. This will help you test the code.
- Sample output for just the prompt functions:

```
******************************
Testing Prompt for String.
******************************
What is your favourite colour? Deep blue or purple

You said, "Deep blue or purple".


******************************
Testing Prompt for Yes/No.
******************************
Would you like a mint? [yes/no]: K
Please enter one of: Yes, Y, No, or N.
Would you like a mint? [yes/no]: yoh!
Please enter one of: Yes, Y, No, or N.
Would you like a mint? [yes/no]: yEs

Have a nice mint!
Type 'Hello':  Hello
You said, "Hello".
```

3.2 Menu

Create a `Menu` class, as shown in the UML diagram below, to display a menu.

Menu
- menuTitle : string - menuOptions : vector<string> - menuKeys : vector<char>
+ setTitle(newTitle : string) : void + addOption(description : string, key : char) : void + showMenu(isVertical : bool) : char

At the top of the `menuTest.cpp` file, there is a place to implement this class.

- The private member variables store:
  - The menu's title/prompt.
  - A vector of all the menu option descriptions (example element: "Run away").
  - A vector of all the menu option keys (example element: 'R').
  - The vectors will be used to display the menu and validate user input.
- The title is set using the setter member functions.
- `addOption()` adds a new option to the menu. Its two parameters should be added to the end of the respective member-variable vectors (`menuOptions` and `menuKeys`).
- `showMenu()` displays the menu to the screen and lets the user make a selection.
  - It returns the key (a `char`) the user selected.
  - If the user makes an invalid selection (the key is not in `menuKeys`), then it displays an error message and re-displays the menu.
  - It is case-insensitive: it does matter if `addOptions()` is given the key `'a'` vs `'A'`; nor does it matter if the user presses `'a'` or `'A'`; either way it should work.
  - Add a `cin.ignore()` after any `cin` statements to discard the `'\n'`.
  - If `showMenu()`'s parameter is `true`, the menu is displayed vertically, such as:

    ```
    (F) First stuff
    (S) Second thing
    :
    ```

  - If the parameter is `false`, the menu is displayed horizontally, such as:

    ```
    (F) First stuff, (S) Second thing:
    ```

- Notes:
  - You may (if you choose), implement any simple setter/getter member functions as inline functions. (Note there is only one!). Other member functions must be non-inline.
  - Put your function comments with the implementation of the functions.

```
******************************
Testing Horizontal Menu.
******************************

You are face-to-face with a big dragon!
(R) Run, (H) Hide, (W) Throw rocks, (T) Taunt: a
I'm sorry, you can't do that. Enter a character shown in the brackets.

You are face-to-face with a big dragon!
(R) Run, (H) Hide, (W) Throw rocks, (T) Taunt: r

You run as fast as you can!


******************************
Testing Vertical Menu.
******************************

You meet a friend and you wish to greet her. What language do you choose?
(E) English
(F) French
(C) C++
(K) Klingon
(V) Elvish
: x
I'm sorry, you can't do that. Enter a character shown in the brackets.

You meet a friend and you wish to greet her. What language do you choose?
(E) English
(F) French
(C) C++
(K) Klingon
(V) Elvish
: K

You say, "nuqneH!"
```

4. Deliverables

Submit the items listed below to the Course Management System: https://courses.cs.sfu.ca/

1. `vectorStats.cpp`
2. `creatureTest.cpp`
3. `menuTest.cpp`

Each files must begin with a comment of :

```
// <Name>, <user ID>, student number
```

Please remember that all submissions will automatically be compared for unexplainable similarities. We will also be comparing assignments against what we can find on the internet. Please review the notes from lecture on the expectations for academic honesty.