Lab 7: Morse Code Generator

Topics

1. Inheritance
2. Dynamic Binding

Prelab Assignments

- Review the lectures, readings, examples and assignments on inheritance and dynamic binding.
- Read about Morse Code. You will need a conversion chart for this lab.

Interesting fact - The current Mars Science Laboratory has Morse code inscribed on its wheel! Scientists and Engineers on the ground use this pattern to track the path followed by the rover on Mars. For more information - http://www.nasa.gov/mission_pages/msl/news/msl20120829f.html.

Tasks:

Write a *Message* class that should contain:

1. **Constructors** – an empty constructor and a parametric constructor.
   - The empty constructor should prompt user for the message.
   - The parametric constructor should take in the message when the object is declared/allocated and initialized.
2. **Destructor** – To free any memory that may have been allocated (depends on your implementation).
3. **Method: print** – This will print the message to the screen.
4. Appropriate data member(s) for the message. You may need other methods.

Write a *MorseCodeMessage* class that should extend the *Message* class mentioned above:

1. **Translate** – method to translate your input message into Morse code (called during initialization).
2. Redefines the **print** method to print the message in both Morse code and text to the screen on two separate lines.
3. You may need to define additional data members and methods.

Write a *MessageStack* class – A generic stack class.

1. **Constructor** – a parametric constructor. It can take one *Message* or *MorseCodeMessage* object.
2. **Destructor** – To de-allocate memory, if needed.
3. **Methods**:
   - **Push** – push a *Message* or *MorseCodeMessage* object into the stack.
   - **Pop** – pop an object from the stack. You should follow LIFO (Last In First Out).
   - **Print stack** – to print all messages on the stack to the screen (beginning from the top of the stack). You should employ dynamic binding to call the appropriate print function depending on the message object type.

Note: There are several ways to create your stack. You can use arrays or vectors (Hint: what type should that array/vector be to be able to use dynamic binding?). You can use a linked list (but you don't *need* to). That might be a good way of implementing your stack, especially if you have previous experience with linked lists.

Testing Details:

You need to create only **one** stack object of type *MessageStack*. As mentioned above, the stack can handle objects from both the *Message* and the *MorseCodeMessage* classes.

Create a menu with the following options:

1. **Enter Text** – Read text from the user. After the user enters the text, a follow up option should be given: translate or not. Depending on what the user selects, an appropriate object should be used. Then, it should be pushed onto the stack.
2. **Print Stack** – Print all the existing elements in the stack. If the stack is empty, just print "*Stack is Empty*."
3. **Pop** – Pop the latest message element from the stack. If the stack is empty, print a message like "*Cannot pop! Stack is Empty*."
4. **Exit** – Quit your program

Run various cases, using several messages. Make sure to test all your methods.

Deliverables:

1. Lab Report:

   The report should follow the lab report format posted on Canvas.

   - Results: show a few test cases. Include screenshots showing a few messages displayed.
   - Discussion: Discuss your results. Describe any problems and interesting things that you may have encountered while coding the lab.
   - Append your source code with proper comments and indentation to the end of your report.

Grading:

Report:	30	
Code:	70	**(Well organized, properly commented and indented)**