Zombie Conga Party!

Description:

You are going to create a silly Zombie Conga Line using your Linked List.

Each node is going to store a Zombie in it. Zombies can be Red, Yellow, Green, Blue, Magenta and Cyan.

Every turn you will randomly generate a Zombie object and an action. You will then perform that action using the Zombie object as the parameter for that action.

Linked List

You will create a Doubly-LinkedList Class and a Node Class.

The LinkedList should contain the following methods in its public interface:
NOTE: T stands for the template data type, so T data means the variable of type T (whatever T is)

- Constructor
- Destructor
- AddToFront(T data):void – create a node containing T data and add it to the front of the list
- AddToEnd(T data):void – create a node containing T data and add it to the end of the list
- AddAtIndex(T data, int index):void – create a node containing T data and add it to the list at index. The new node containing the data will be the #index node in the list. *Throws std::out_of_range* exception if index is less than zero or greater than the size of the list
- AddBefore(Node<T>*, T data):void – create a node containing T data and add it before a particular node
- AddAfter(Node<T>*, T data):void – create a node containing T data and add it after a particular node
- RemoveFromFront():T – Delete first item and return its contents
- RemoveFromEnd():T – Delete last item and return its contents
- RemoveTheFirst(T data):void – find first instance of T data and remove it
- RemoveAllOf(T data):void – find each instance of T data and remove it
- RemoveBefore(Node<T>*):T – delete the node before a particular node, return its contents
- RemoveAfter(Node<T>*):T – delete the node after a particular node, return its contents
- ElementExists(T data):bool – Returns a T/F if element exists in list
- Find(T data):Node<T>* – Look for data in the list, return a pointer to its node
- IndexOf(T data):int – returns an index of the item in the list (zero-based), returns -1 if the item is not in the list.
- RetrieveFront:T – returns the data contained in the first node, *does not delete it*
- RetrieveEnd:T – returns the data contained in the last node, *does not delete it*
- Retrieve(int index):T – returns the data contained in node # index, *does not delete* it. *Throws std::out_of_range* exception if index is less than zero or greater than the size of the list
- PrintList:void – Loop through each node and print the contents of the Node
- Empty:void – Empty out the list, delete everything
- Length:int – How many elements are in the list

Linked List contd

More methods private or public should be created as needed to facilitate the functionality of the Interface methods.

If you feel your list needs more functionality, feel free to create it.

Node Class

- Properties
  - -data:T
  - -next:Node<T>*
  - -previous:Node<T>*
- Methods
  - +Node()
  - +Node(T)
  - +Node(T, Node<T>*)
  - +getData():T
  - +setData(T):void
  - +getNext():Node<T>*
  - +setNext(Node<T>*):void
  - +getPrevious():Node<T>*
  - +setPrevious(Node<T>*):void
- Friend (optional)
  - LinkedList class

The node class should be fairly rudimentary. Ideally, it should be templated so you can store anything in it.

Zombie Class

- Properties
  - -type:char
- Methods
  - +Zombie()
  - +Zombie(char)
  - +getType():char
  - +operator==:bool
- Friend
  - ostream& operator<<(ostream&, const Zombie&)

The Zombie class is very simple, don't give it any more work to do.

Conga (class?)

You can create a class for your Conga or you can create a set of functions.

Spelling it out for you:

You should create the following classes:

1. LinkedList
2. Node
3. Zombie

Your Nodes should store Zombies.

Your LinkedList is made of Nodes.

Actions

If you chose to make a Conga class, then these actions should be methods of your Conga class. Methods don't need the list passed in since the linked list should be a property of the Conga class.

If you are not making a Conga class, then these actions should be functions.

- Engine!
  - This zombie becomes the first Zombie in the conga line
  - Suggested function:
    - void engine_action(LinkedList<Zombie>* list, Zombie randomZomb)
- Caboose!
  - This zombie becomes the last zombie in the conga line
  - Suggested function:
    - void caboose_action(LinkedList<Zombie>* list, Zombie randomZomb)
- Jump in the Line!
  - This zombie joins the conga line at position X where X <= length of the linked list
  - Suggested function:
    - void jump_in_action(LinkedList<Zombie>* list, Zombie randomZomb)
- Everyone Out!
  - Remove all matching zombies from the linked list
  - Suggested function:
    - void everyone_out_action(LinkedList<Zombie>* list, Zombie randomZomb)
- You Out!
  - Remove the first matching zombie from the linked list
  - Suggested function:
    - void you_out_action(LinkedList<Zombie>* list, Zombie randomZomb)
- Brains!
  - Generate two more matching Zombies and add one to the front (engine_action), one to the end (caboose_action) and one to the middle (round down).
  - Suggested function:
    - void brains_action(LinkedList<Zombie>* list, Zombie randomZomb)
- Rainbow Brains!
  - Perform an engine_action on the zombie that was generated
  - Add one of each zombie color to the end via caboose_action in this order: Red, Yellow, Green, Blue, Cyan, Magenta
  - Suggested function:
    - void rainbow_action(LinkedList<Zombie>* list, Zombie randomZomb)
- Making new Friends!
  - Find the first Zombie of this color in line.
    - Do a coin flip - if rand() % 2 == 0 then insert before, else insert after.
  - If no Zombie of that color exists, then perform caboose_action on the zombie
  - Suggested function:
    - void friends_action(LinkedList<Zombie>* list, Zombie randomZomb)

Notes

These actions are external to the Linked List. They are accomplished by calling Linked List methods.

These actions are not part of your Zombie class. Zombies are very simple classes and have no responsibility for Conga actions.

If you allocate your LinkedList on the stack, then you should pass it by reference into the action functions rather than pointer or value.

Note on Random calls:

Don't call a random number unless you actually need to!! If you call too many randoms, your output will not match the Gradescope!

- The one that people mess up the most is the Making New Friends action. Don't flip the coin until you know you need to!!

Setting up the List:

Set up the initial Conga Line by running these actions:

1. Run a Rainbow Brains! Action
2. Run 3 Brains actions

User Interface:

Command line argument:

Add a command line option -s <integer> to allow the user to provide a seed for the random number generator. This seed value should be given to srand(<int>).

If no -s option is given, then seed the random number generator with time: srand(time(0));

Console input:

- Ask the user how many rounds they want to run.
- Run the conga party for that many rounds
  - Start with round 0
  - Every time round % 5 == 0 – delete the first and last zombie in the linked list
  - Choose your action with rand() % 8
    - Keep the actions in the order show previously
  - Choose your zombie with rand() % 6
    - Use the order: Red, Yellow, Green, Blue, Cyan, Magenta
  - Run that action with your chosen zombie
- If the conga line ever empties completely due to an action tell the user that the Party is Over.
- Once the number of rounds has finished. Ask the user if they want to continue the party or end.
- If they choose to continue ask them for a new number of rounds to run.

Output:

Each round you’ll output the entire Linked List. You can represent each zombie as a single character corresponding to their color (R, Y, G, B, M, C).

You’ll show the zombie generated and the action generated.

Then you’ll show the outcome of the action.

The output should follow this pattern (see sample output for example):

```
Round: <round number>
Size: <list size> :: [<R|Y|G|B|M|C>]= …
New Zombie: [<R|Y|G|B|M|C>] -- Action: [<Engine!|Caboose!|Jump In!|Everyone Out!|You
Out!|Brains!|Rainbow!|New Friends!>]
The conga line is now:
Size: <list size> :: [<R|Y|G|B|M|C>]= …
**************************************************
```

Hints:

- Start by building an Integer version of the doubly linked list. It will get you points if you can’t get other things to work and it will be easier to transition to the template after getting all the functionality of the data structure solid.
- Build robust constructors and use them! Your Node and your Zombie will benefit highly from good constructors.
- Do yourself a favor and overload the == operator in your Zombie. It will make life easier!
- Overload the cout for your Zombie. It’ll make your printList method easier.
- You might consider making the Conga its own class so you can make each of the actions a method in the conga class.

Extra Credit:

+2 – Color your Zombies in the output.

If you use termcolor (https://github.com/ikalnytskyi/termcolor) you must do std::cout << termcolor::colorize at the beginning of your program for the color to display correctly on grade scope, otherwise the graders won't see it and you won’t get credit.

+3 – Zombie Fight!

Add a command line argument -EC to run this extra credit.

If the program is run with -EC, create a Zombie Conga, but don’t prompt the user for how many rounds they want to run. Just arbitrarily run 1000 rounds for them.

After that … Create an array with the counts for each Zombie type. (I suggest writing a function to do this).

Then pick 2 random indexes to send zombies to fight each other. “Roll a d6” (rand() % 6) for each zombie and apply any modifier. This table shows who has an advantage when two colors face. If a color has an advantage multiply its roll by 1.5.

Whoever has the highest roll wins the fight, reduce the count of the losing Zombie color by 1. Continue until one color remains and announce the winner! The same color should never fight itself.

	R	Y	G	B	C	M
R	X	🟥			🟥	
Y		X	🟨			🟨
G	🟩		X	🟩		
B		🟦		X	🟦	
C			■		X	■
M	🟪			🟪		X

NOTE – If you didn’t do the Zombie Conga you can still do this extra credit! When you program is run with the -EC … create the array and fill each index with ((rand() % 100) + 50) and then …