CSE 232 Spring 2017

Programming Project #11

Assignment Overview

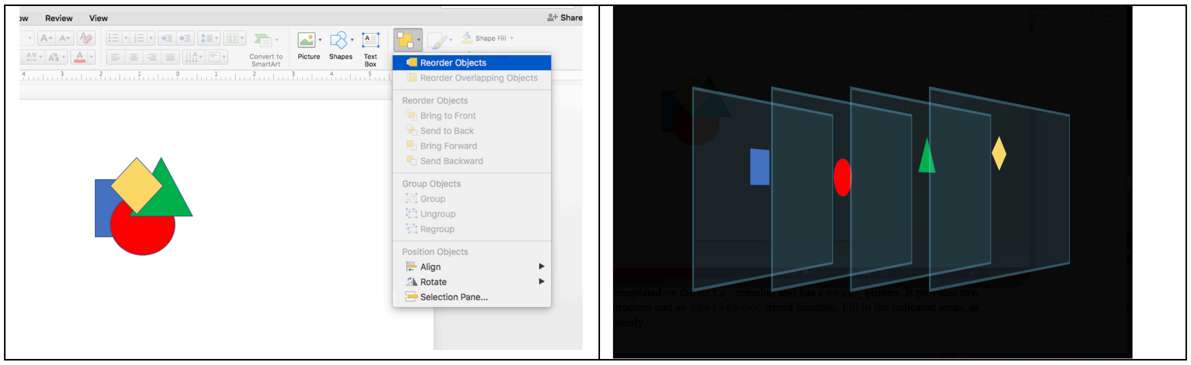
Last one. You are going to build a `Painters Algorithm` data structure for maintaining the order, and manipulating the order, of drawing elements on a screen.

This assignment is worth 60 points (6.0% of the course grade) and must be completed and turned in before 11:59PM on Monday, April 23rd .

Background

You can look up more about the Painters Algorithm (https://en.wikipedia.org/wiki/Painter%27s_algorithm). The basic idea is that the elements to be drawn or ordered, such that elements in the "back" are drawn first, each element drawn in order until the "front" element is drawn.

You have probably done this, perhaps without knowing it, something like PowerPoint. Any element that is drawn can be moved "forward" or "backward" in the ordered list, such that things in the back are "covered" by things in the front.



You can see that I have drawn four elements: blue square, red circle, green triangle, yellow diamond in that order, back to front. They occupy different layers, but I can move an element forward or backward to change the overall painting order. PowerPoint offers the ability to visualize the layers with "Reorder objects" so you can change their relative position in the ordered list (give it a try).

Specification

You are going to implement two templated classes which together allows us to create a singly linked list:

- an Element<T> class. This is the "payload" class
- a PAL<T> class. This is the singly linked list of Elements that, taken in order, constitute the Painter's ALgorithm (hence PAL)

I'm giving you a `proj11-skeleton.h`, which has the declarations but no definitions. Start your `proj11.h` from this and fill it in with what is required.

Element class
Has three elements:

- `name_` : (string) name of the object
- `color_` : color of the object. This is templated. Could be a string or an long
- `next_` : Element pointer to the next Element

PAL class
This class maintains only two pointers:

- `back_` : Element pointer to the first element in the list. This is the first thing drawn in the painter's algorithm, painting back to front
- `front_` : Element pointer to the last element in the list. This is the last thing drawn in the painter's algorithm, painting back to front.

Parts that are declared but not defined. You cannot change the declarations, but you must provide the definitions.

- **void print_list(ostream& out) (PRIVATE)**. Print the list. Used by `operator<<`. See output for format
- **default constructor**
- **2 arg constructor (string name, T color)**. Create a node, both `back_` and `front_` point to this single node
- **1 arg constructor (Element<T> n)**. Both `back_` and `front_` point to this single node.
- **copy constructor**
- **destructor**
- **operator=**
- **void add(Element<T> &n)**: adds the node to be the back of the list (first thing drawn, what `back_` points to)
- **void add(string name, T d)**: create node, add node to the back of the list (first thing drawn, what `back_` points to)
- **pair<Element<T>*, Element<T>* > find(Element<T> &n)**. Searches list for the first node whose `name_` is equal to `n.name_`. Returns a pair where:
    - o first element is pointer to the found list node (could be a `nullptr` if not found)
    - o second element is pointer to the found list node's predecessor (just before it in the list, could be `nullptr` if no predecessor or if node was not found).
- **pair<Element<T>*, Element<T>* > find(Element<T> &n)**. Similar to above, same return rules but takes the `name_` being sought as the argument.
- **void move_forward1(Element<T> &n)**. Move the node one forward (towards the front).
    - o node must be in this list, else do nothing (uses `find`)
    - o if node is already at the front, do nothing.
- **void move_to_front(Element<T> &n)**. Move node all the way to the front of the list (last thing drawn).

    - node must be in this list, else do nothing (uses `find`)
    - if node is already at the front, do nothing.
- **void move_back1(Element<T> &n)**. Move the node one back (towards the back).
    - node must be in this list, else do nothing (uses `find`)
    - if node is already at the back, do nothing.
- **void move_to_front(Element<T> &n)**. Move node all the way to the back of the list (first thing drawn).
    - node must be in this list, else do nothing (uses `find`)
    - if node is already at the back, do nothing.

Anything else you need!!!
You are defining your class, so any other functions/members you might need, feel free to add them. The test will only involve using the main provided.

Important things to Note
Deliverables

`proj11.h` your source code solution along with the main program unchanged!

1. Please be sure to use the specified file names
2. Save a copy of your file in your CSE account disk space (H drive on CSE computers).
3. You will electronically submit a copy of the file using the "handin" program: http://www.cse.msu.edu/handin/webclient

Assignment Notes
1. Edge cases are everything here. You need to be careful of `nullptr` and it pops up everywhere. List your edge cases, especially for the moves:
    a. what if it is first in the list
    b. what if it is last in the list
    c. what if the node doesn't exist
    d. etc.
2. You just have to suck it up and use the debugger when you get a Seg Fault, and you will get Seg Faults.
    a. make sure you compile with –g
    b. start with `gdb a.out` (or whatever you named the executable)
    c. then `run < inputX.txt`
    d. use `bt` to see the trace
    e. us the `up` and `down` commands to get to a source line you recognize and get out of all the library gibberish
    f. you can then `print` variables listed in the source line (who has the `nullptr`).
    g. you can set a breakpoint by saying `break proj11.h:lineno` to set a break and step through.
3. You know and I know that the code I gave for singly linked lists would be useful here.
    a. go ahead and use it but you should write a comment saying you did so.

   b. using code from other places is a time-honored tradition in programming but not giving attribution is just nasty wrong. If you borrow code, say so.

4. If you are not drawing pictures with arrows and nodes, then you are not in the game. I ***absolutely*** drew pictures to make sure I did what was required (well, to figure out what was required). Draw pictures, simulate the action. Let's be better programmers, let's thing before we write code!!!