Week 9 Assignment

MATH 5061: Fundamentals of Computer Programming for Scientists and Engineers

Due: November 21st, 2016

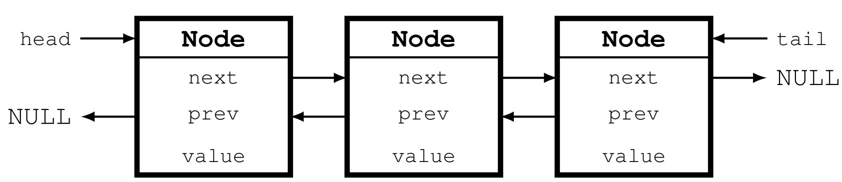
Instructions

Please submit your entire program as .cpp and .h files and include the compiled binary program or output file. Create folders to separate the examples. Send them as a .tar.gz or ZIP file via email to math5061@temple.edu. Make sure to use the subject line (without quotes) "MATH5061:Assignment 09 :ACCESSID" Where ACCESSID is your AccessNet ID, for example tue86537.

1 Double-Linked List

In class we talked about a single-linked list. For this exercise you will now write code necessary for a double-linked list. As a reminder, a linked list is a data structure which manages a collection of nodes. Each node is connected to other nodes through pointers.

In a double-linked list each node contains a pointer to the previous and next element of the sequence. The list itself only stores the head and tail pointer of the sequence, which point to the first and last node.



The goal of this exercise is to implement functions which manipulate the linked list. Follow these steps:

1. Download `double_linked_list.h` from the website (Listing 1)
2. Create a new file `double_linked_list.cpp` and write method code which implements the behavior described in Section 1.1.
3. Finally add a file `main.cpp` to test your new class. See Section 1.2 for details.

Listing 1: double_linked_list.h

```
struct Node {
    Node * next;
    Node * prev;
    int value;
};

class DoubleLinkedList {
    Node * head;
    Node * tail;
public:
    DoubleLinkedList();
    ~DoubleLinkedList();

    void push_front(int value);
    void push_back(int value);

    int pop_front();
    int pop_back();

    int get(int index);

    bool remove(int index); // bonus

    bool insert(int index, int value); // bonus

    int size();

    void print();
};
```

1.1 Member Methods

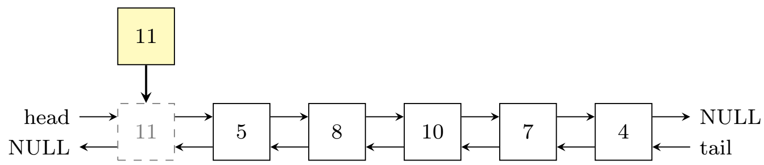
In the following all methods of the `DoubleLinkedList` class are described. Implement them according to this specification. Note some functions are optional and can be implemented for bonus points.

1.1.1 Constructor: `DoubleLinkedList()`

Initialize empty sequence. All pointers point to nothing.

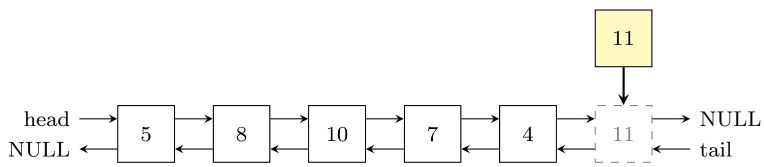
1.1.2 `push_front(int value)`

Add a new value to the sequence at the beginning (head).



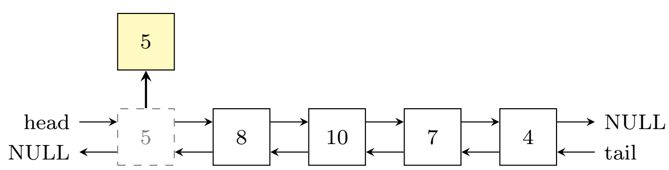
1.1.3 `push_back(int value)`

Add a new value to the sequence at the end (tail).



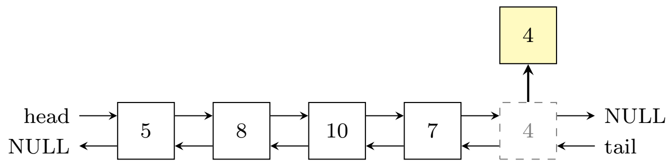
1.1.4 `pop_front()`

Remove the first element (head) of the sequence and return its value. Return -1 if sequence is empty.



1.1.5 `pop_back()`

Remove the last element (tail) of the sequence and return its value. Return -1 if sequence is empty.



1.1.6 `size()`

Return length of the linked list. Do not store this information explicitly. Compute it by counting the nodes from `head` to `tail`.

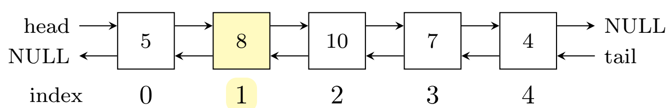
1.1.7 `print()`

Print all values of the entire list on the console, separated by commas and surrounded by brackets.

```
[11, 22, 33, 44, 55, 66]
```

1.1.8 `get(int index)`

Return value of element with a given index. Elements are numbered from 0 to `size()-1` starting from `head`. If the index is out of range, return -1.

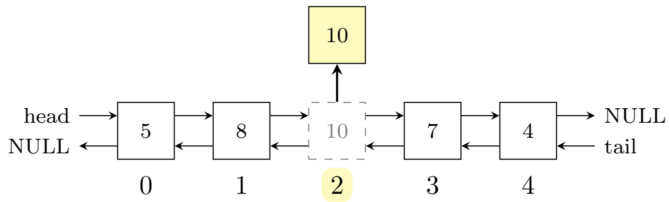


1.1.9 Destructor: `~DoubleLinkedList()`

Should delete all nodes from the sequence. You could reuse any of the `pop_front`, `pop_back` or `remove` methods to implement this.

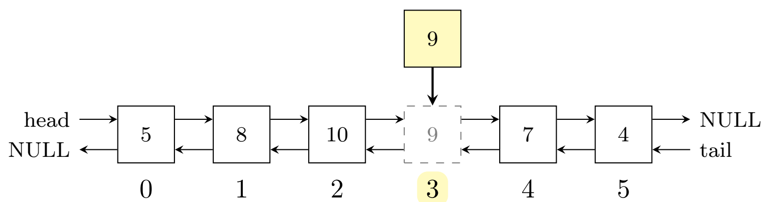
1.1.10 (Optional Bonus) `remove(int index)`

Remove element at a given index. You first have to find the node with that index, adjust the pointers of its neighbors stored in `prev` and `next`, and finally delete the node. Note: removing at index 0 (head) does the same as `pop_front()`. Removing at index `size()-1` (tail) does the same as `pop_back()`. Return **`true`** if deletion was successful. **`false`** if index is out of range.



1.1.11 (Optional Bonus) `insert(int index, int value)`

Insert element at a given index. Push any existing element to the right (make it next of new node). Inserting at index 0 does the same as `push_front()`. Inserted at index `size()` is the same as `push_back()`. Return **`true`** if insertion was successful and index is in valid range, otherwise return **`false`**.



1.2 Testing

Write a small test program which creates an object of `DoubleLinkedList` on the stack and showcases all of its functions. Use the `print` function to output what happens after each change. All functions must be used at least once in this test program. It should compile and run as follows:

```
g++ -o test main.cpp double_linked_list.cpp
./test
```

Listing 2: Example output

```
Staring with an empty list
[]

push_front(3)
[3]

push_front(4)
[4, 3]

push_back(5)
[4, 3, 5]

push_back(7)
[4, 3, 5, 7]

push_back(9)
[4, 3, 5, 7, 9]

get(2) = 5

insert(1, 6)
[4, 6, 3, 5, 7, 9]

remove(3)
[4, 6, 3, 7, 9]

pop_front()
[6, 3, 7, 9]

pop_back()
[6, 3, 7]

final size is: 3
```

2 Polymorphism: Writing a custom `Shape` class

In this exercise you will extend an existing program. The current version draws a random selection of shapes into an image. Finally, this image is saved as a bitmap file in the current directory. The only supported shape so far is a circle.

1. Download `shapes.tar.gz` from the website and extract its contents.
2. To compile & run this program execute the following commands:

```
g++ -o shapes main.cpp image.cpp shape.cpp
./shapes
```

3. Study the included files and extend `shape.h` and `shape.cpp` with a `Rectangle` class similar to the `Circle` class. A rectangle should be defined by its top-left corner, its width and height. It should also have a color, which comes from its base class `Shape`. When the `draw()` function is called, all pixels inside the rectangle should be filled with this color.
4. The `main` function in `main.cpp` creates random selection of shapes and draws them on an image. Create a utility function `create_random_rectangle()` which returns a new instance of a rectangle at a random location, width, height and color. It is very similar to `create_random_circle()`. Use this utility function inside `main` so that rectangles are created as well.
5. Save the resulting image as bitmap and attach it to your submission. The final image created by the program should look similar to this:

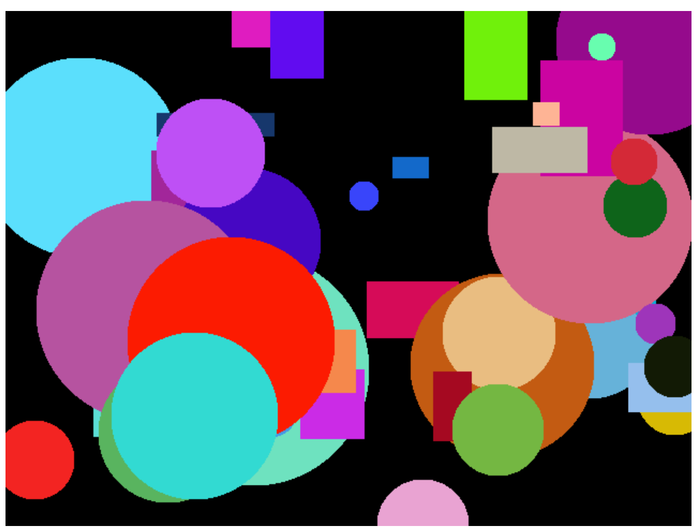


Figure 1: Example output of the final program