CS302

Assignment 1: Connect Four

Description

Ah yes, connect four, the game of champions. The game of connect four is a turn based game where you have an $n \times n$ board, and a player places a chip at the top of the column and lets go of the chip and slides on top of other chips on that column. Each player has either red chips or blue chips, and players alternate between each other. This continues until a player wins (when there is 4 chips of the same color adjacent to each other either horizontally, vertically, or diagonally). You can play the game yourself here https://www.cbc.ca/kids/games/play/connect-4. Now to make things complicated, we will use an array of circular linked lists to represent each column of the board and you will use iterators to traverse the board. The linked list and iterator class can be seen below:

```
template <typename T>
class LL
{
  //contents of each node
  struct node
  {
    T data;
    node* prev;
    node* next;
  };
  //iterator class to allow access of each node in main
public:
  class Iterator
  {
    public:
      friend class LL;
      Iterator();
      Iterator(node*);
      T operator*() const;
      Iterator operator++(int);
      Iterator operator++();
      Iterator operator--(int);
```

```
      Iterator operator--();
      bool operator==(const Iterator&) const;
      bool operator!=(const Iterator&) const;

    private:
      node* current;
  };

  LL();
  LL(const LL<T>&);
  const LL<T>& operator=(const LL<T>&);
  ~LL();
  void headInsert(const T&);
  void tailInsert(const T&);
  void headRemove();
  bool isEmpty() const;
  std::size_t size() const;
  Iterator begin() const;
  Iterator end() const;

private:
  node * portal;
};
```

Each member of the `Iterator` class performs/contains the following

- `Node * current` - a pointer that contains the address of the node that the Iterator object points to
- `LL<T>::Iterator::Iterator()` - default constructor that sets `current` with `NULL` or `nullptr`
- `LL<T>::Iterator::Iterator(Node* ptr)` - constructor that sets `current = ptr`
- `T LL<T>::Iterator::operator*() const` - overloads the dereference operator, just returns the data field of the node the Iterator object is pointing to
- `typename LL<T>::Iterator LL<T>::Iterator::operator++(int)` - postfix ++ operator that moves the Iterator object one node over to the right (sets `this->current = this->current->next;`) and returns an Iterator object that has its current set to `this->current`
- `typename LL<T>::Iterator LL<T>::Iterator::operator++()` - prefix ++ operator that moves the Iterator object one node over to the right (sets `this->current = this->current->next`) and then returns an iterator object whose current pointed to the previous node, returns an iterator whose current is set to `this->current->prev`
- `typename LL<T>::Iterator LL<T>::Iterator::operator--(int)` - postfix – operator that moves the Iterator object one node over to the left (sets `this->current = this->current->prev`) and returns an Iterator object that has its current set to `this->current`
- `typename LL<T>::Iterator LL<T>::Iterator::operator--()` - prefix – operator that moves the Iterator object one node over to the left (sets `this->current = this->current->prev`) and then returns an iterator object whose current pointed to the previous node, returns an iterator whose current is set to `this->current->next`
- `bool LL<T>::Iterator::operator==(const Iterator& rhs) const` - comparison operator, compares if the `*this` Iterator and the `rhs` Iterator point to the same node, if they do return `true` else return `false`
- `bool LL<T>::Iterator::operator!=(const Iterator& rhs) const` - comparison operator, compares if the `*this` Iterator and the `rhs` Iterator point to a different node, if they point to different

nodes return `true` else return `false`

Each member of the `LL` class performs/contains the following

- `struct node` - a struct object that contains an element in the list and a pointer for its left and right neighbor within the list
- `node * portal` - a pointer that points to the dummy node, `portal->prev` points to the tail node and `portal->next` points to the head node
- `LL<T>::LL()` - default constructor, allocates a node to `portal` and sets `portal->prev = portal->next = portal;`
- `LL<T>::LL(const LL<T>& copy)` - deep copy constructor, deep copies the `copy` object into the `*this` object
- `const LL<T>& LL<T>::operator=(const LL<T>& rhs)` - deep copy assignment operator, deep copies the `rhs` object into the `*this` object, make sure you deallocate the `*this` object first before performing the deep copy, also check for self assignment, then `return *this` at the end
- `LL<T>::~LL()` - destructor, deallocates the entire linked list and then deallocates the dummy node
- `void LL<T>::headInsert(const T& item)` - insert a new node to the front of the linked list and this node's data field must contain the contents in the item parameter
- `void LL<T>::tailInsert(const T& item)` - insert a new node to the back of the linked list and this node's data field must contain the contents in the item parameter
- `typename LL<T>::Iterator LL<T>::begin() const` - returns an `Iterator` object whose current field contains `portal->next` (address of head node)
- `typename LL<T>::Iterator LL<T>::end() const` - returns an `Iterator` object whose current field contains `portal` (the address of the dummy node)

Contents Of Main

You will implement a connect four game on a 7 by 7 board, so you will need to declare the following

```
LL<bool> board[7]; //use a constant instead of literal 7
```

You have an array of 7 linked lists, think of each as a column of the board. Each node contains a boolean, that denotes a red chip or blue chip `true`/`false` respectively or vice versa. Suppose you want to insert a blue chip into column 0, you can write the following

```
board[0].tailInsert(false);
```

And if you want to stack a red chip on top of the blue chip on column 0, you can write

```
board[0].tailInsert(true);
```

And if you want to traverse all the chips in row x, you can write the code in a few ways

```
LL<bool>::Iterator it = board[x].begin();

while (it != board[x].end())
{
  *it; //retrieves true/false value of node
  it++;
}
```

You can also you a for loop

```
for (LL<bool>::Iterator it = board[x].begin(); it != board[x].end(); it++)
{
  *it; //retrieves true/false value of node
}
```

Also, since code grade and most modern Linux Ditros use C++ 11, you can use the keyword `auto` for the Iterator type, so

```
auto it = board[x].begin();

while (it != board[x].end())
{
  *it; //retrieves true/false value of node
  it++;
}
```

And likewise

```
for (auto it = board[x].begin(); it != board[x].end(); it++)
{
  *it; //retrieves true/false value of node
}
```

In main, you prompt the user for a column A through G, and you need to translate the letter to a column index using `tolower(ch) - 'a'`, if the column is out of bounds or if the column is maxed out (contains 7 chips), you need to reprompt, once a chip is placed, you check the board if a winner can be determined or a tie, if not alternate to the next player (from blue to red or red to blue) and continue until a winner can be determined or tie.

Check vertically (4 consecutive matching chips in a column will be the easier check), the diagonal check will be challenging, but the horizontal check will be the toughest check.

Specifications

- Must use `LL<bool>` and `LL<bool>::Iterator` objects to manipulate and traverse the list
- Do not add extra nodes to make the win logic easier, the only nodes that can exist in the linked lists are based on user selection
- Do not create a 2D array of any additional arrays, you can have an array of linked lists only
- Do not convert your linked lists in arrays (that would make the assignment too easy)
- Have your code well documented
- Make sure your code is memory leak free

Sample Run

```
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
```

```
-  -  -  -  -  -  -

A  B  C  D  E  F  G

Red Player Select a row: B

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  R  -  -  -  -  -

A  B  C  D  E  F  G

Blue Player Select a row: C

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  R  B  -  -  -  -

A  B  C  D  E  F  G

Red Player Select a row: B

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  R  -  -  -  -  -
-  R  B  -  -  -  -

A  B  C  D  E  F  G

Blue Player Select a row: B

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  -
-  R  -  -  -  -  -
-  R  B  -  -  -  -

A  B  C  D  E  F  G

Red Player Select a row: E
```

```
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  -
-  R  -  -  -  -  -
-  R  B  -  R  -  -

A  B  C  D  E  F  G

Blue Player Select a row: B

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  -
-  R  -  -  -  -  -
-  R  B  -  R  -  -

A  B  C  D  E  F  G

Red Player Select a row: D

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  -
-  R  -  -  -  -  -
-  R  B  R  R  -  -

A  B  C  D  E  F  G

Blue Player Select a row: B

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  -
-  R  -  -  -  -  -
-  R  B  R  R  -  -

A  B  C  D  E  F  G

Red Player Select a row: R
Inavlid Column
Red Player Select a row: G

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  -
```

```
-  B  -  -  -  -  -
-  R  -  -  -  -  -
-  R  B  R  R  -  R

A  B  C  D  E  F  G

Blue Player Select a row: G

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  -
-  R  -  -  -  -  B
-  R  B  R  R  -  R

A  B  C  D  E  F  G

Red Player Select a row: G

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  R
-  R  -  -  -  -  B
-  R  B  R  R  -  R

A  B  C  D  E  F  G

Blue Player Select a row: G

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  -
-  B  -  -  -  -  B
-  B  -  -  -  -  R
-  R  -  -  -  -  B
-  R  B  R  R  -  R

A  B  C  D  E  F  G

Red Player Select a row: g

-  -  -  -  -  -  -
-  -  -  -  -  -  -
-  B  -  -  -  -  R
-  B  -  -  -  -  B
-  B  -  -  -  -  R
-  R  -  -  -  -  B
-  R  B  R  R  -  R

A  B  C  D  E  F  G
```

```
Blue Player Select a row: g

-  -  -  -  -  -  -
-  -  -  -  -  -  B
-  B  -  -  -  -  R
-  B  -  -  -  -  B
-  B  -  -  -  -  R
-  R  -  -  -  -  B
-  R  B  R  R  -  R

A  B  C  D  E  F  G

Red Player Select a row: g

-  -  -  -  -  -  R
-  -  -  -  -  -  B
-  B  -  -  -  -  R
-  B  -  -  -  -  B
-  B  -  -  -  -  R
-  R  -  -  -  -  B
-  R  B  R  R  -  R

A  B  C  D  E  F  G

Blue Player Select a row: g
Invalid Move
Blue Player Select a row: b

-  -  -  -  -  -  R
-  B  -  -  -  -  B
-  B  -  -  -  -  R
-  B  -  -  -  -  B
-  B  -  -  -  -  R
-  R  -  -  -  -  B
-  R  B  R  R  -  R

A  B  C  D  E  F  G

Blue wins!
```

Submission

Submit the source files to code grade by the deadline

References

- Supplemental Video https://youtu.be/e-fnvD6CqXA
- Link to the top image can be found at https://magicspecialevents.com/event-rentals/rental-item/giant-connect-four-4-game