3rd Recitation

Lists, stacks and queues

- Download the file `aed2324_p03.zip` from the class page on Moodle and unzip it (it contains a `docs` folder, a `lib` folder, and the `Tests` folder with the files `cell.h`, `cell.cpp`, `funListStackQueueProblem.h`, `funListStackQueueProblem.cpp`, `game.cpp`, `game.h`, `kid.cpp`, `kid.h`, `stackExt.h` and `tests.cpp`, and the files `CMakeLists.txt` and `main.cpp`).
- On CLion, open a project by selecting the folder containing the files from the previous point.

1. Implement the function:

```
list<int> FunListStackQueueProblem::removeHigher(list<int> &values, int x)
```

This function removes elements greater than x (exclusive) from the values list, and returns a new list with the elements that were removed, according to the order in which they were removed. The elements that remain in the values list must retain their relative position.

Expected time complexity: $O(n)$

Execution example:
input: *values* = {7, 8, 12, 5, 2, 3, 5, 6}
x = 5
output: *result* = {7, 8, 12, 6}
values = {5, 2, 3, 5}

2. Implement the member function:

```
list<pair<int,int>> FunListStackQueueProblem::overlappingIntervals(list<pair<int,int>> values)
```

Given a collection of intervals (list `values`), your task is to join all the overlapping intervals. An interval is identified by a pair (`pair<int,int>`). Thus, the interval starting at 3 and ending at 7 is identified by the pair `<3,7>`. Consider that the start of the interval is always no greater than the end of the interval.
Tip: Start by sorting the values list.

Expected time complexity: $O(n \times \log n)$

Execution example:
input: *values* = {<1,3>, <6,8>, <2,4>, <9,10>}
output: *result* = {<1,4>, <6,8>, <9,10>}

input: *values* = {<6,8>, <1,9>, <2,4>, <4,7>}
output: *result* = {<1,9>}

3. The `StackExt` class implements a stack structure with a new method: findMin.

```
class StackExt {
   //...
public:
   StackExt() {};
   bool empty() const;          // checks if the stack is empty
   T& top();                    // returns the element at the top of the stack
   void pop();                  // removes element
   void push(const T& val);     // inserts element
   T& findMin();                // returns the value of the smallest element
};
```

In addition to the methods already known from the `stack` class, the `StackExt` class includes the `findMin` method. This method returns the value of the smallest element in the stack.

- Must implement all methods of the `StackExt` class in:
  Expected time complexity: O(1), for all methods

  Tip: Use the `stack` class from the STL library. Use two stacks: one to store all the values, and another to store the minimum values as they arise.
- Document the implemented code using Doxygen** (see note at the end of the sheet).

4. Implement the function:

```
vector<string> FunListStackQueueProblem::binaryNumbers(int n)
```

Given a number `n`, the task is to generate all the binary numbers with decimal values from `1` to `n`.
Tip: Use a queue to help generate/build the binary numbers.

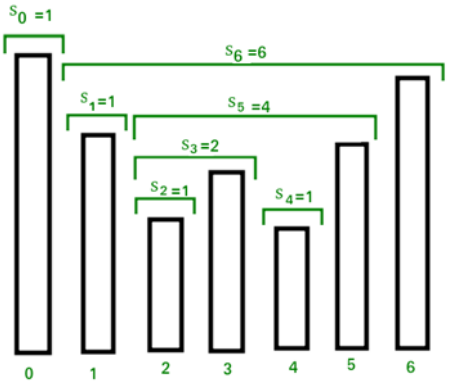
Expected time complexity: $O(n)$

Execution example:
input: $n = 5$
output: *result* = `{"1", "10", "11", "100", "101"}`

5. Implement the function:

```
vector<int> FunListStackQueueProblem::calculateSpan(vector<int> prices)
```

A stock's price span can be defined as the maximum number of consecutive days before the current day when the stock's price was equal to or lower than the current price.
Given a vector of prices for a particular stock on a set of consecutive days, the function calculates the price span of that stock on those days, returning this information in a vector.



Tip: Note that $span_i$ can be easily calculated if we know the closest day before i, such that the price of that day is higher than the price of day i. Use a stack in this step.

Expected time complexity: $O(n)$

Example execution:
input: *prices* = `{100, 80, 60, 70, 60, 75, 85}`
output: *result* = `{1, 1, 1, 2, 1, 4, 6}`

More Exercises

6. "Pim Pam Pum, each ball kills one chicken and the turkey gets rid of one yourself". Remember this children's game, whose rules are as follows:

- The first child says the sentence, and at each word points to each of the children in the game (starting with themselves). When they reach the end of the list of children, they go back to the beginning, i.e. to themselves.
- When the last word of the sentence is said, the child who is being pointed to gets out and leaves the game. Counting starts again with the next child on the list.
- The remaining child loses the game.

Use a list (we'll use the STL's `list` class) to implement this game. The elements of the list are objects of the `Kid` class (see file `kid.h`).

```
class Kid {
   string name;
   int age;
   char sex;
public:
   Kid();
   Kid(string nm, unsigned a, char s);
   Kid(const Kid& k1);
   //...
};
```

The `Game` class is also defined:

```
class Game {
   list<Kid> kids;
public:
   Game();
   Game(list<Kid>& l2);
   static unsigned numberOfWords(string phrase);
   Kid loseGame(string phrase);
   list<Kid> rearrange();
   list<Kid> shuffle() const;
   //...
};
```

6.1. Implement the member-function:

```
Kid Game::loseGame(string phrase)
```

This function plays the game according to the stated rules when the phrase used is a phrase and returns the child who loses the game. Use the `numberOfWords` function member (already supplied) which determines the number of words in a phrase given in a parameter:

```
unsigned int Game::numberOfWords(string phrase)
```

Expected time complexity: $O(n^2)$

Execution example:
input: *phrase* = `"Pim Pam Pum Pim"`
kids = `{"Rui", "Ana", "Rita", "Joao", "Marta", "Vasco"}`
// kids is a list of `Kid` *objects, here represented only by the child's name, for simplification*
output: *result* = `"Marta"`
explanation:
in the 1st round, the 4th element comes out (phrase with 4 words), which is "`Joao`"
in the 2nd round, the count starts next, on "`Marta`". "`Ana`" comes out
in the 3rd round, the count starts next, on "`Rita`". "`Rui`" is out
in the 4th round, the count starts next at "`Rita`". "`Rita`" out
in the 5th round, the count starts next at "`Marta`". "`Vasco`" out
that leaves "`Marta`", who loses the game

6.2. Implement the member-function:

```
list<Kid> Game::rearrange()
```

This function changes the arrangement (and possibly the number) of children in the game, distributing the girls and boys along the list. The children are arranged by repeating the same pattern relative to the child's sex. Thus, if there are **`n`** girls and **`m`** boys: i) if **`n<m`**, the pattern will consist of **`1`** girl and **`m/n`** boys; ii) if **`n>=m`**, the pattern will consist of **`n/m`** girls and **`1`**

boy. The relative arrangement of the boys to each other and the relative arrangement of the girls to each other must be maintained. The first child on the list is female. The remaining children are stored in a queue that is returned by the function.

Tip: use two auxiliary data structures, one to place the girls and one to place the boys.

Expected time complexity: $O(n)$

Execution example:
input: *kids* = `{"Rui", "Ana", "Maria", "Joao", "Vasco", "Luis"}`
// kids is a list of `Kid` *objects, here represented only by the child's name, for simplification*
output: *result* = `{}` ;
kids = `{"Ana", "Rui", "Joao", "Maria", "Vasco", "Luis"}`
explanation: the list `{"Rui", "Ana", "Maria", "Joao", "Vasco", "Luis"}` gives rise to the pattern `<girl, boy, boy>`, leaving the final list equal to `{"Ana", "Rui", "Joao", "Maria", "Vasco", "Luis"}`, no children left.

6.3. Implement the member-function:

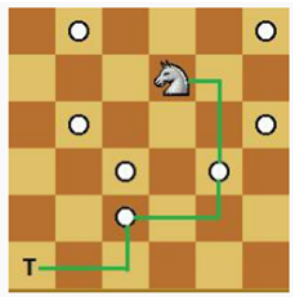
```
list<Kid> Game::shuffle() const
```

This function creates a new list where the children in the game are placed in a randomly determined position (use the `rand()` function to generate random numbers).

7. Implement the function:

```
int FunListStackQueueProblem::knightJumps(int initialPosx, int initialPosy,
                                          int targetPosx, int targetPosy, int n)
```

Given a square chessboard of size `n×n`, the position of the horse (`initialPosx`, `initialPosy`) and the target position (`targetPosx`, `targetPosy`), the function must determine the minimum number of jumps the horse makes to reach the target position.



in the picture:
- the circles illustrate the next position the horse can reach with one jump.
- the horse reaches the target position with 3 jumps: `(4,5) -> (5,3) -> (3,2) -> (1,1)`

Tip: Use a queue to keep track of the possible positions the horse can reach. Build the queue so that at the front of the queue are the positions that can be reached in the fewest jumps.

Expected time complexity: $O(n^2)$

Example execution:
input: *initialPosx* = 4, *initialPosy* = 5, *targetPosx* = 1, *targetPosy* = 1, *n* = 6
output: *result* = 3

****Doxygen Notes**

Doxygen generates documentation from source code. Steps to follow (see also moodle and Doxygen page):

1. Install Doxygen.
2. Include the reference to Doxygen in the project, placing this information in `CMakeLists.txt`.
   The following text is an example. Note the indication (you should change it to what you want) of:
   - Doxyfile configuration file in "`CMAKE_CURRENT_SOURCE_DIR}/docs/Doxyfile`"

```
# Doxygen Build
find_package(Doxygen)
if(DOXYGEN_FOUND)
        set(BUILD_DOC_DIR "${CMAKE_SOURCE_DIR}/docs/output")
        if(NOT EXISTS "${BUILD_DOC_DIR}")
                file(MAKE_DIRECTORY "${BUILD_DOC_DIR}")
        endif()

        set(DOXYGEN_IN "${CMAKE_CURRENT_SOURCE_DIR}/docs/Doxyfile")
        set(DOXYGEN_OUT "${CMAKE_CURRENT_BINARY_DIR}/Doxyfile")
        configure_file("${DOXYGEN_IN}" "${DOXYGEN_OUT}" @ONLY)

        message("Doxygen build started")
        add_custom_target(Doxygen ALL
                        COMMAND "${DOXYGEN_EXECUTABLE}" "${DOXYGEN_OUT}"
                        WORKING_DIRECTORY "${CMAKE_CURRENT_BINARY_DIR}"
                        COMMENT "Generating API documentation with Doxygen"
                        VERBATIM)
else(DOXYGEN_FOUND)
        message("Doxygen needs to be installed to generate the documentation.")
endif(DOXYGEN_FOUND)
```

3. The *Doxyfile* configuration file should be placed in the folder indicated in `CMakeLists.txt`. An example of this file is on the UC moodle page and has already been included in the set of files provided for this class (`aed2324_p03.zip`).
   Note the indication (you should change it to what you want) of where the source code is:
   - The `Doxyfile` file indicates in "`INPUT =`" the place where the source code is located
4. Document the C++ source code; the list of commands can be found here.