Homework 4

All functions should be in a main.cpp file. Only include: **`<iostream>, <string>, <stdexcept>, <vector>, "Stack.h", and "Queue.h".`** All code should be uploaded to HW/Homework04 in your GitHub class repository.

Task 1: (2 points)

Define a function named **validParentheses()** that takes a constant string reference parameter. The function should return true if the parentheses in the string are properly nested, and false otherwise. You can assume the string parameter only contains the characters **`'(', ')', '{', '}',`** and **`'[', ']'`**.

For example:

```
validParentheses("{[()]}") => true.
validParentheses("{[(])}") => false.
validParentheses("}") => false."
```

Steps:

1. Create an empty dynamic stack to hold opening parentheses.
2. Iterate through each character in the input string.
3. If the character is an opening parenthesis '(', '{', '[' , push it onto the stack.
4. If the character is a closing parenthesis ')', '}', ']':
   a. If the stack is empty, return false (no matching opening parenthesis).
   b. Pop the top element from the stack and check if it matches the closing parenthesis. If it doesn't match, return false.
5. After processing all characters, if the stack is empty, return true (all parentheses matched). Otherwise, return false (some opening parentheses were not matched).

Task 2: (1 point)

Define a function named **reverseString()** that takes a constant string reference parameter. The function should return a new string that is the reverse of the input string.

For example:

```
reverseString("Hello, World!") => "!dlroW ,olleH"
```

Steps:

1. Create an empty static stack with a capacity equal to the length of the input string.
2. Iterate through each character in the input string and push it onto the stack.
3. Create an empty string to hold the reversed result.
4. While the stack is not empty, pop characters from the stack and append them to the result string.
5. Return the reversed string.

Task 3: (2 points)

Define a function named **printBinaries()** that takes an integer parameter *n*. The function should print the unsigned binary numbers from 1 to *n*, each on a new line.

For example:

```
printBinaries(5)=>: 1
                    10
                    11
                    100
                    101
```

Steps:

1. Create an empty dynamic queue to hold binary numbers as strings.
2. Enqueue the first binary number "1" into the queue.
3. While the size of the queue is less than n:
   a. Dequeue the front element from the queue and print it.
   b. Generate the next two binary numbers by appending "0" and "1" respectively to the dequeued number.
   c. Enqueue these two new binary numbers back into the queue.
4. After the loop, dequeue and print the last binary number in the queue.

Task 4: (2 points)

Define a function named **hotPotato()** that takes a vector of strings called names and an integer called *n* as parameters. The function should simulate the Hot Potato game and print the name of the winner. Every *n-th* person is eliminated until only one person remains.

For example:

```
hotPotato({"Alice", "Bob", "Charlie", "Diana", "Eve"}, 3)  =>
Charlie is eliminated.
Alice is eliminated.
Eve is eliminated.
Bob is eliminated.
Diana is the winner.
```

Steps:

1. Create an empty static queue with a capacity equal to the number of names.
2. Enqueue all names into the queue.
3. While there is more than one name in the queue:
    a. For n - 1 times, dequeue the front name and enqueue it back to the rear of the queue (simulating passing the potato).
    b. Dequeue the front name (the one holding the potato) and print that this person is eliminated.
4. After the loop, dequeue and print the last remaining name as the winner.

Task 5: (2 points)

Create a header file named **Queue2Stacks.h**, with a header guard, and namespace dshw. Include "Stack.h" and <stdexcept>. Define a generic class named **Queue2Stacks** that implements a queue using two stacks. The class should support the following operations:

- **void enqueue(const T& value)**: Adds an element to the end of the queue.
- **T dequeue()**: Removes and returns the element at the front of the queue. Throws an exception if the queue is empty.
- **bool isEmpty() const**: Returns true if the queue is empty, false otherwise.
- **size_t size() const**: Returns the number of elements in the queue.
- **const T& peek()**: Returns the element at the front of the queue without removing it. Throws an exception if the queue is empty.

Steps:

1. Use two dynamic stacks to manage the elements of the queue.
2. For enqueue() operation, push the new element onto stack1.
3. For dequeue() operation, if stack2 is empty, pop all elements from stack1 and push them onto stack2.If stack2 is empty, throw an error. Otherwise, pop from stack2 and return the value.
4. Implement isEmpty() and size() methods based on the sizes of both stacks.
5. Implement peek() method to return the top element of stack2, or transfer elements from stack1 to stack2 if needed.

Task 6: (1 point)

In the main function of main.cpp:

1. Test the validParentheses() function with various strings and print the results.
2. Test the reverseString() function with different strings and print the reversed results.
3. Test the printBinaryNumbers() function with a positive integer n to print binary numbers from 1 to *n*.
4. Test the hotPotato() function with a list of names and a number to simulate the game and print the winner.
5. Create an instance of Queue2Stacks, perform a series of enqueue() and dequeue() operations, and print the results to verify its functionality.