Lecture review

Linked list

A *linked list* is a data structure that consists of a sequence of elements, where each element points to the next element in the sequence. The first element is called the *head* of the list, and the last element points to `null`. A linked list can be singly linked or doubly linked. In a singly linked list, each element points to the next element, whereas in a doubly linked list, each element points to both the next and the previous elements.
A *circular linked list* is a linked list where the last element points to the first element. Circular linked lists are useful for applications where the list is traversed in a circular manner.

Resizing array

A *resizing array* is an array that automatically resizes itself when the number of elements exceeds its capacity. When the array is full, a new array of double the size is created, and all elements are copied to the new array. This operation takes n steps, where n is the number of elements in the array. Resizing arrays are useful when the number of elements is not known in advance.

Stack and Queue

Stack and *queue* are two important ADTs. A stack stores items in a last-in, first-out (LIFO) manner, whereas a queue stores items in a first-in, first-out (FIFO) manner. Following table summarizes the operations of a stack and a queue.

Stack	Queue	Description
`push(x)`	`enqueue(x)`	Insert an item x
`pop()`	`dequeue()`	Remove and return an item (most recently inserted item for stack and least recently inserted one for queue)
`empty()`	`empty()`	Return true if the stack/queue is empty
`size()`	`size()`	Return the number of items in the stack/queue

We have seen two implementations of stacks and queues in the lecture: using linked list and using resizable array. Note that from the perspective of the user, the two implementations of stack (respectively queue) are indistinguishable as both provide the identical public interface.
The following files are provided for your reference:

- `linkedstackofstrings.hpp` implements a stack using linked list.
- `resizingarraystackofstrings.hpp` implements a stack using resizable array.

Infix to Postfix conversion

Infix notation is the common arithmetic notation in which operators are written between the operands. For example, $2+3$ is an *infix* expression. In *postfix* notation, the operators are written after their operands. For example, the postfix equivalent of $2+3$ is $23+$.
To convert an infix expression to postfix, we use the following algorithm:

- Initialize an empty stack and an empty string to store the postfix expression.
- For each token in the infix expression:
  - If the token is an operand, append it to the postfix expression.
  - If the token is an operator, pop all operators from the stack that have higher or equal precedence than the current operator, and append them to the postfix expression. Then push the current operator to the stack.
  - If the token is an opening parenthesis, push it to the stack.
  - If the token is a closing parenthesis, pop all operators from the stack until an opening parenthesis is encountered. Append all the popped operators to the postfix expression.
- Pop all remaining operators from the stack and append them to the postfix expression.
- The postfix expression is the desired output.

Postfix evaluation

To evaluate a postfix expression, we use the following algorithm:

- Initialize an empty stack to store the operands.
- For each token in the postfix expression:
  - If the token is an operand, push it to the stack.
  - If the token is an operator, pop the top two operands from the stack, apply the operator, and push the result back to the stack.
- The result of the expression is the only item left in the stack.
- Pop the result from the stack and return it.

Lab exercises

Exercise 1

Write a `Stack` client that reads a string of parentheses, square brackets, and curly braces from standard input and uses a stack to determine whether they are properly balanced. For example, your program should print true for `[()]{}{[()()]()}` and false for `[()]`.

Exercise 2

Write a function `InfixToPostfix()` that converts an *infix* arithmetic expression given as `string` and returns the equivalent expression as `string` in *postfix*.

For simplicity, we assume that the input expression contains paranthesis, only the following operators: addition(`+`), subtraction(`-`), mutiply(`*`), divide(`/`), and exponentiation(`^`), and the operands are single-digit positive integers.

- *Example 1:*
  Input: `( 2 + ( ( 3 + 4 )* ( 5 * 6 )))`
  Output: `2 3 4 + 5 6 * * +`

- *Example 2:*
  Input: `( ( ( 5 + ( 7 * ( 1 + 1 )))* 3 )+ ( 2 * ( 1 + 1 )))`
  Output: `5 7 1 1 + * + 3 * 2 1 1 + * +`

Exercise 3

Write a function `EvaluatePostfix()` that that takes a postfix expression as string, evaluates it, and returns the value.

For simplicity, we assume that the input expression contains only the following operators: addition(`+`), subtraction(`-`), mutiply(`*`), divide(`/`), and exponentiation(`^`), and the operands are single-digit positive integers.

- *Example 1:*
  Input: `1 2 3 4 5 * + 6 * * +`
  Output: `277`
- *Example 2:*
  Input: `7 16 16 16 * * * 5 16 16 * * 3 16 * 1 + + +`
  Output: `30001`
- *Example 3:*
  Input: `7 16 * 5 + 16 * 3 + 16 * 1 +`
  Output: `30001`

Exercise 4

Josephus problem. In the *Josephus problem* from antiquity, n people are in dire straits and agree to the following strategy to reduce the population. They arrange themselves in a circle (at positions numbered

from 0 to $n-1$) and proceed around the circle, eliminating every m-th person until only one person is left. Legend has it that Josephus figured out where to sit to avoid being eliminated.

Using circular linked list, write a program that takes two integer inputs `m` and `n` and prints the order in which people are eliminated (and thus would show Josephus where to sit in the circle).

- *Example 1:*
  Input: $n = 5$ and $m = 2$
  Output: $1, 3, 0, 4, 2$
- *Example 2:*
  Input: $n = 7$ and $m = 3$
  Output: $2, 5, 1, 6, 4, 0, 3$

Solution: Use a circular linked list to store the positions of the people. Do $m-1$ times: Find the m-th person in the list, and remove it.