

Assignment no. 5

Title: Applications of Linear Lists, Stacks and Queue

All four problems are mandatory for this assignment.

- Part 1 : Nested linked lists
 1. Objective: The objective of this lab assignment is to implement a library catalog management system using nested linked lists in the C programming language.
 2. Define the Book Structure: Define a structure named "Book" with the following members:
 - a. "title" (string) to store the title of the book.
 - b. "author" (string) to store the name of the author.
 - c. "publicationYear" (integer) to store the year of publication.
 3. Define the Genre Structure: Define a structure named "Genre" with the following members:
 - a. "genreName" (string) to store the name of the genre.
 - b. "books" (linked list) to store the list of books belonging to the genre.
 4. Implement the Nested Linked List:
 - a. Create a linked list data structure for the "Book" structure to represent the list of books under each genre.
 - b. Each node of the "Book" linked list should contain a "Book" structure and a pointer to the next node.
 - c. Create a linked list data structure for the "Genre" structure to represent the list of genres in the library catalog.
 - d. Each node of the "Genre" linked list should contain a "Genre" structure and a pointer to the next genre.
 5. Basic Operations: Implement functions for the following basic linked list operations for the "Book" linked list:
 - a. "createBookNode" - creates a new book node and returns its address.
 - b. "insertBookAtFront" - inserts a new book at the front of the book linked list under a specific genre.
 - c. "deleteBookFromFront" - deletes the first book from the book linked list under a specific genre.
 - d. "displayBooks" - prints the details of all books under a specific genre.
 6. Implement functions for the following basic linked list operations for the "Genre" linked list:
 - a. "createGenreNode" - creates a new genre node and returns its address.
 - b. "insertGenreAtFront" - inserts a new genre at the front of the genre linked list.
 - c. "deleteGenreFromFront" - deletes the first genre from the genre linked list.
 - d. "displayGenres" - prints the names of all genres in the library catalog.
 7. Library Catalog Management System:
 - a. Provide a user interface for the library catalog management system, allowing users to:
 - Add new genres to the library catalog.
 - Add new books under a specific genre.
 - Delete a genre from the library catalog (and all books under it).
 - Delete a book from a specific genre.
 - Display all genres and books in the library catalog.
 - b. Search and Update: Implement functions for the following additional operations:
 - "searchBookByTitle" - searches for a book by title and returns its details (genre, author, publication year).
 - "updateBookDetails" - allows the user to update the details (author, publication year) of a book by its title.
 8. Testing and Demonstration:
 - a. Test your library catalog management system by adding, deleting, and updating genres and books.
 - b. Demonstrate the functionality of the nested linked list by displaying all genres and books in the library catalog.

9. Note: Ensure that you handle memory allocation and deallocation correctly to avoid memory leaks in the nested linked lists. Encourage students to test their implementation thoroughly and handle edge cases appropriately.

- Part 2 : Polynomial using linked lists
 1. Define the Polynomial Structure:
 - a. Define a structure named "Term" with the following members:
 - "coeff" (integer) to store the coefficient of the term.
 - "exp" (integer) to store the exponent of the term.
 - "next" (pointer to the next Term) to create the linked list.
 2. Polynomial Creation:
 - a. Implement a function named "createTerm" to create a new Term node and return its address.
 - b. Implement a function named "insertTerm" to insert a new term (with given coefficient and exponent) into the polynomial linked list.
 - c. Prompt the user to input the number of terms in the polynomial and their coefficients and exponents. Create the linked list accordingly.
 3. Polynomial Display: Implement a function named "displayPolynomial" to display the polynomial expression in a readable format (e.g., $3x^2 + 2x + 5$).
 4. Polynomial Addition:
 - a. Implement a function named "addPolynomials" to add two polynomial linked lists and store the result in a new linked list.
 - b. Display the original polynomials and the result polynomial.
 5. Polynomial Multiplication:
 - a. Implement a function named "multiplyPolynomials" to multiply two polynomial linked lists and store the result in a new linked list.
 - b. Display the original polynomials and the result of polynomial multiplication.
 6. Polynomial Evaluation:
 - a. Implement a function named "evaluatePolynomial" to evaluate the polynomial for a given value of x.
 - b. Prompt the user to input the value of x, and then calculate and display the result of polynomial evaluation.
 7. Testing and Demonstration:
 - a. Test each operation (creation, addition, evaluation, and display) with various polynomial expressions.
 - b. Demonstrate the functionality of the program by showing the results of different test cases.
 8. *Optional Challenge*: Polynomial Multiplication by Scalar:
 - a. Implement a function named "multiplyPolynomialByScalar" to multiply a polynomial linked list by a scalar (constant) value.
 - b. Display the original polynomial and the result of polynomial multiplication by scalar.
 9. *Optional Challenge*: Polynomial Differentiation:
 - a. Implement a function named "differentiatePolynomial" to calculate the derivative of the polynomial and store it in a new linked list.
 - b. Display the original polynomial and its derivative.
 10. *Optional Challenge*: Polynomial Integration:
 - a. Implement a function named "integratePolynomial" to calculate the indefinite integral of the polynomial and store it in a new linked list.
 - b. Display the original polynomial and its integral.

- Part 3 : Title: Postfix Expression Evaluation using Stack
 1. Objective: The objective of this lab assignment is to implement the evaluation of postfix expressions using a stack data structure in the C programming language.
 2. Define the Stack Structure:
 - a. Define a structure named "StackNode" with the following members:
 - "data" (integer) to store the value of the node.
 - "next" (pointer to the next StackNode) to create the stack.
 3. Stack Operations:
 - a. Implement stack operations as necessary, e.g., push, pop, peek etc.
 4. Postfix Expression Evaluation:
 - a. Implement a function named "evaluatePostfixExpression" to evaluate a given postfix expression and return the result.
 - b. Prompt the user to input a postfix expression containing numbers and operators (+, -, *, /).
 - c. Implement the algorithm to iterate through the expression, push operands onto the stack, and perform operations when encountering operators.
 5. Testing and Demonstration:
 - a. Test the postfix expression evaluation with various postfix expressions.
 - b. Demonstrate the functionality of the program by showing the results of different test cases.
 6. Optional Challenge: Handling Expressions with Parentheses:
 - a. Extend the program to handle postfix expressions with parentheses.
 - b. Implement the algorithm to evaluate expressions with parentheses using appropriate stack operations.
 7. Extend to Infix to Postfix Conversion:
 - a. Implement a function named "convertInfixToPostfix" to convert an infix expression to a postfix expression.
 - b. Prompt the user to input an infix expression, and then use the stack to convert it to postfix.
 - c. Implement the Shunting Yard algorithm or a similar approach to achieve the conversion.
 8. Optional Challenge: Evaluate Infix Expressions:
 - a. Implement a function named "evaluateInfixExpression" to evaluate infix expressions using postfix conversion and evaluation.
 - b. Combine the "convertInfixToPostfix" and "evaluatePostfixExpression" functions to achieve this.

- Part 4 : Hot Potato Game Simulation using Queues
 1. Objective: The objective of this lab assignment is to implement and simulate the "Hot Potato" game using a queue data structure in the C programming language.
 2. Define the Queue Structure: Define a structure named "QueueNode" with the following members:
 - a. "name" (string) to store the name of the player.
 - b. "next" (pointer to the next QueueNode) to create the queue.
 3. Queue Operations:
 - a. Implement a function named "createQueue" to create an empty queue and return its address.
 - b. Implement a function named "isEmpty" to check if the queue is empty.
 - c. Implement a function named "enqueuePlayer" to add a new player (name) to the queue.
 - d. Implement a function named "dequeuePlayer" to remove and return the front player from the queue.
 4. Hot Potato Game Simulation:
 - a. Implement a function named "playHotPotato" that simulates the "Hot Potato" game.
 - b. Prompt the user to input the names of players and the pass count (number of passes before removing a player).
 - c. Use the queue operations to simulate passing the potato and removing players in a circular manner.
 - d. Display the final player who gets eliminated from the game.
 5. Testing and Demonstration:
 - a. Test the hot potato game simulation with various inputs to ensure correctness.
 - b. Demonstrate the functionality of the program by showing the results of different test cases.
 6. Optional Challenge: Visualize the Game:
 - a. If desired, you can create a simple text-based visualization of the game progress.
 - b. Display the current player holding the potato and the queue of remaining players after each pass.
 7. Optional Challenge: Implement a Variable Pass Count:
 - a. Extend the program to allow for a variable pass count for each player (determined randomly or input by the user).
 - b. Simulate the game with the variable pass count and display the final eliminated player.
 8. Optional Challenge: Extend to Elimination Rules:
 - a. Implement more complex elimination rules (e.g., players with certain characteristics are eliminated after certain passes).
 - b. Simulate the game with the new elimination rules and display the final eliminated player.