REPUBLIC OF THE PHILIPPINES

BICOL UNIVERSITY

BICOL UNIVERSITY POLANGUI

Name: Faith Ann Sañado

Susaine Rico

Rechelle Borbe

Professor: Ms. Khristine Botin

Course and year: BSIS 2A

PROJECT/FINAL EXAMINATION

Project Overview

The Mad Libs game is a word game where players fill in blanks with different types of words (nouns, verbs, adjectives, etc.) to create a funny or nonsensical story. This program automates the process. The user provides three words: a noun, a verb, and an adverb. These words are stored in a BST, which allows for efficient searching and deletion. The program then substitutes these words into a pre-written story template, generating a customized Mad Libs story. Finally, it demonstrates the BST's functionality by performing inorder, preorder, and postorder traversals before and after deleting a word type.

How to run the code

Welcome to the Mad Libs game! This fun game allows you to create a silly story using your own words. Here's how to play

Objective

The goal is to fill in the blanks of a story with specific types of words to create a funny or interesting narrative.

How to Play

1. Input Words

- You will be prompted to enter three words. Each word must be of a specific type
- Noun: A person, place, or thing (e.g., "dog," "city," "Rechelle")
- Verb: An action word (e.g., "run," "cry," "jump")
- Adverb: A word that describes how an action is performed (e.g., "quickly," "sadly," "disappointedly")

2. Story Creation

- Once you have entered your words, the game will generate a story using the words you provided. The story will contain placeholders (like [noun], [verb], and [adverb]) that will be replaced by your chosen words.

3. Word Deletion

- After the story is created, you will have the option to delete one of the words you entered. You can choose to delete a noun, verb, or adverb.
- After deleting a word, the game will show you the updated story and the new traversals of the word list.

4. Traversals

- The game will display the words in three different orders:
- Inorder Traversal- Words are shown in alphabetical order.
- Preorder Traversal- Words are shown starting from the root down to the leftmost node.
- Postorder Traversal- Words are shown after visiting all the child nodes.

Example

- If you enter "Rechelle" (noun), "cry" (verb), and "disappointedly" (adverb), the generated story might look like:
- "My friend, Rechelle, came over my room while I was working. Suddenly, she spilled her coffee on my papers, and I cry. My deadline is now completely and

disappointedly missed."

Enjoy the Game .Feel free to be creative with your word choices to make the story as funny or silly as you like!

Sample output

Compile Result

```
Hello! Welcome to my Mad Libs game!
We're going to create a story based on your input.
Prepare your 3 words:
NOTE:first word will be a name of person
1) Enter a word (noun): Rechelle
2) Enter a word (verb): cry
3) Enter a word (adverb): disapointedly

Here's your Mad Libs story:
My friend, Rechelle, came over my room while I was workin
g. Suddenly, she spilled her coffee to my papers, and I c
ry. My deadline is now completely and disapointedly misse
d

Inorder Traversal:
 adverb: disapointedly
 noun: Rechelle
 verb: cry


Preorder Traversal:
 noun: Rechelle
 adverb: disapointedly
 verb: cry


Postorder Traversal:
 adverb: disapointedly
 verb: cry
 noun: Rechelle


Enter a word type to delete: verb
Word deleted.
New Inorder Traversal:
adverb: disapointedly
 noun: Rechelle

New Preorder Traversal:
noun: Rechelle
 adverb: disapointedly

New Postorder Traversal:
adverb: disapointedly
 noun: Rechelle
```

Description of each functionality

1. Header inclusion

- #include <iostream> : Provides input/output functionalities (like cout and cin).
- #include <string> : Enables the use of strings.
- #include <vector> : (While not directly used in this code, it's a common header for dynamic arrays, which might be relevant in future expansions of the project).

2. Node Structure: (As before) A basic building block for the binary search tree. It holds the word type ("noun," "verb," "adverb"), the word itself, and pointers to its children nodes in the tree.

3. BST Class: (As before) The core data structure for managing the words. It uses a binary search tree for efficient storage and retrieval based on word type.

4. BST Class Methods (insert, preorderTraversal, inorderTraversal, postorderTraversal, search, findMin, deleteNode): (As before). These methods handle the internal workings of the BST, including insertion, deletion, and different traversal orders (preorder, inorder, postorder).

5. BST Class Public Interface Methods: These are the functions that are accessible from outside the BST class:

- insert(string wordType, string word) : Adds a new word-type pair to the BST.
- preorderTraversal() : Prints the tree's contents using a preorder traversal.
- inorderTraversal() : Prints the tree's contents using an inorder traversal (alphabetical order by word type).
- postorderTraversal() : Prints the tree's contents using a postorder traversal.

- getWord(string wordType) : Retrieves the word associated with a given word type.
- deleteWord(string wordType) : Removes a word from the BST based on its type.

6. **main() Function: Program Execution Flow:**
   - Initialization: A BST object (madLibs) is created.
   - User Input: The program prompts the user to enter three words: a noun, a verb, and an adverb. These words are stored in the words array.
   - BST Insertion: Each word and its type are inserted into the madLibs BST using the insert() method.
   - Story Template: A finalStory string contains the Mad Libs template with placeholders ([noun] , [verb] , [adverb]).
   - Story Generation: The program iterates through the wordTypes array. For each type, it finds the corresponding placeholder in finalStory and replaces it with the word retrieved from the BST using getWord() .
   - Story Output: The completed Mad Libs story is printed to the console.
   - Tree Traversals: The program demonstrates the different tree traversal methods (inorderTraversal , preorderTraversal , postorderTraversal) to show how the words are stored in the tree.
   - Word Deletion: The user is prompted to enter a word type to delete. The deleteWord() method removes the corresponding node from the BST. The traversals are then shown again to reflect the change.
   - Program Termination: The main() function returns 0, indicating successful execution.

7. **Use of getline**: The getline(cin, words[i]); statement correctly reads the entire line of user input, handling spaces within the words, which would be a problem if cin >> words[i] were used instead.

10. **String Manipulation:** The find() and replace() methods are used effectively to insert the user's words into the Mad Libs story template.

Example Walkthrough of Traversals

Inorder Traversal (Left → Root → Right)

Given a BST that stores words as “noun”, “verb”, and “adverb”, inorder traversal will visit the nodes in alphabetical order of their word types. For example, if the tree stores “noun” first, “verb” second, and “adverb” third, the output will list “adverb”, “noun”, “verb” in that order. In the sample output, we could see that the ordering goes like this:

adverb: disappointedly

noun: Rechelle

verb: cry

Inorder traversal visits the left subtree, then the node, and then the right subtree. Because of how the words were inserted based on the lexicographical order of the wordTypes , the “adverb” ends up being visited first in an inorder traversal.

Preorder Traversal (Root → Left → Right)

The traversal here visit the root, which is the noun, it then visits the left subtree. Visit adverb (no children), viisit the right subtree: visit verb (no children).

noun: Rechelle

adverb: disappointedly

verb: cry

Postorder Traversal (Left → Right → Root)

The traversal here visit the left subtree: visit adverb (no children). Visit the right subtree: Visit verb (no children). Last, Visit the root: noun.

adverb: disappointedly

verb: cry

noun: Rechelle

Again, the outputs depend on how the tree is structured due to insertion order: The root

node is "noun" because it was inserted first. "adverb" becomes the left child of "noun" because "adverb" < "noun". "verb" becomes the right child of "noun" because "verb" > "noun". This structure dictates how each traversal method visits nodes.

Tree Structure