**National University of Computer & Emerging Sciences**

**Karachi Campus**



# Context Free Grammar Checker

**Semester Project Report – Theory of Automata (Spring 2025)**

**Section: 4C**

**Group Members:**

Maryiam Khan 23K-0585

Aiman Nadeem 23k-0876

Amna Rais 23k-0824

## Table of Contents

1. Introduction
2. Project Objectives
3. File Structure and Purpose
4. Methodology
5. Tools and Technologies Used
6. Team Contributions
7. Challenges and Risks
8. 8.Known Improvements and Future Work
9. Visual Demonstration of Project Workflow
10. Conclusion

### Introduction

This project explores core principles from Automata Theory by developing a Context-Free Grammar (CFG)-based Sentence Parser in C++. CFGs form the foundation for understanding syntactic structures in languages, making them essential for designing compilers, interpreters, and natural language processing systems.

Our program allows users to input sentences and checks their correctness against a predefined CFG. Although the project does not directly implement a Pushdown Automaton (PDA), it conceptually aligns with PDA behavior — CFGs are recognized by PDAs, and our parser simulates this behavior through recursive descent parsing, where the C++ function call stack acts as an implicit simulation of the PDA stack.

The parser reads grammar rules from a file and stores them using a Binary Search Tree (BST) for efficient rule lookup. User management is also included via a login/signup system, making the tool interactive and accessible.

This project serves as a practical implementation of how CFGs work in real applications, enhancing our understanding of syntax analysis and theoretical language models.

### Project Objectives

 Simulate Context-Free Grammar (CFG) parsing to check sentence validity.

 Demonstrate automata theory concepts, especially how recursive parsing reflects Pushdown Automata (PDA) behavior.

 Use a Binary Search Tree (BST) to store and access grammar rules efficiently.

 Provide user interaction features like login and signup using file handling.

 Promote modular programming by dividing functionality into multiple well-organized C++ files

### File Structure and Purpose

|  |  |
| --- | --- |
| **File Name** | **Purpose** |
| BST.cpp | Implements the **Binary Search Tree (BST)** data structure for managing student records and handling operations like add, delete, search, and traversal. |
| BST. h | Header file for the **BST** class, defining the structure and function prototypes for operations on student records stored in a binary search tree. |
| Context\_grammar.cpp/.h | Implements the **context-free grammar** (CFG) parsing logic, defining the grammar rules, tokens, and functions used to parse input sentences. |
| Queue.cpp/.h | Implements a **queue** structure for managing the parsing process, storing intermediate tokens and rules during the CFG parsing. |
| Menudriven.cpp/.h | Implements menu-driven functionality, including user interaction for signing up, logging in, and other user interface actions. |
| |  | | --- | | Main.cpp/.h |  |  | | --- | |  | | The entry point of the program that connects all the components, initializing the BST, managing user input, and controlling the flow of the parser program. |

**Methodology**

For this project, we will develop the entire system from scratch. The primary focus is on building a **context-free grammar (CFG)** parser and integrating it with a **Binary Search Tree (BST)** to handle student records. We will create a parser that can read, validate, and manipulate sentences (or inputs) based on predefined grammar rules, and store these inputs as structured records in a Binary Search Tree. This will allow us to efficiently manage the data and perform operations like insertions, searches, and deletions.

The implementation will involve the following steps:

1. **Building the CFG Parser**: We will design a parser that takes input strings and verifies them against a set of context-free grammar rules. The parser will ensure that inputs conform to the grammar before storing or processing them.
2. **Creating the Binary Search Tree**: A Binary Search Tree will be implemented to store and manage the student records, allowing efficient operations like insertion and searching based on keys.
3. **Integration**: The two components, the CFG parser and BST, will be integrated to handle real-time validation of input and efficient management of student data. This will ensure that only valid records are stored in the tree and can be accessed or modified efficiently.

The implementation will be done entirely in **C++** to provide full control over memory management and performance optimization.

### Tools and Technologies Used

* **C++**: The main programming language used for implementing both the **CFG parser** and the **Binary Search Tree**.
* **Windows API**: For handling console input/output operations (e.g., for interacting with users, clearing the screen, or displaying messages).
* **Custom Header Files**: We will create and include custom header files (BST. h, Context\_Free\_Grammar. h) for the functionality related to Binary Search Tree and Context-Free Grammar parsing.

#### **Team Contributions**

#### **Group Member 1:**

* **Designing the CFG Parser**: Implementing the logic for parsing and validating sentences based on context-free grammar rules.
* **Developing the Binary Search Tree (BST)**: Implementing the data structure for managing student records, including functions for insertion, deletion, and search.
* **File: context\_grammar.cpp and bst.cpp**: Responsible for the creation of the grammar validation logic and the tree structure that will manage student data.

#### **Group Member 2:**

* **UI Design and Console Interaction**: Designing and implementing the user interface for interacting with the system via the console. This includes displaying menus, accepting inputs, and showing results to users.
* **Debugging**: Responsible for debugging and ensuring that the input/output flow is smooth and that both the parser and the BST work efficiently.
* **File: main.cpp and menudriven.cpp**: Working on the user-facing aspects of the program and integrating the logic with the rest of the system.

#### **Group Member 3:**

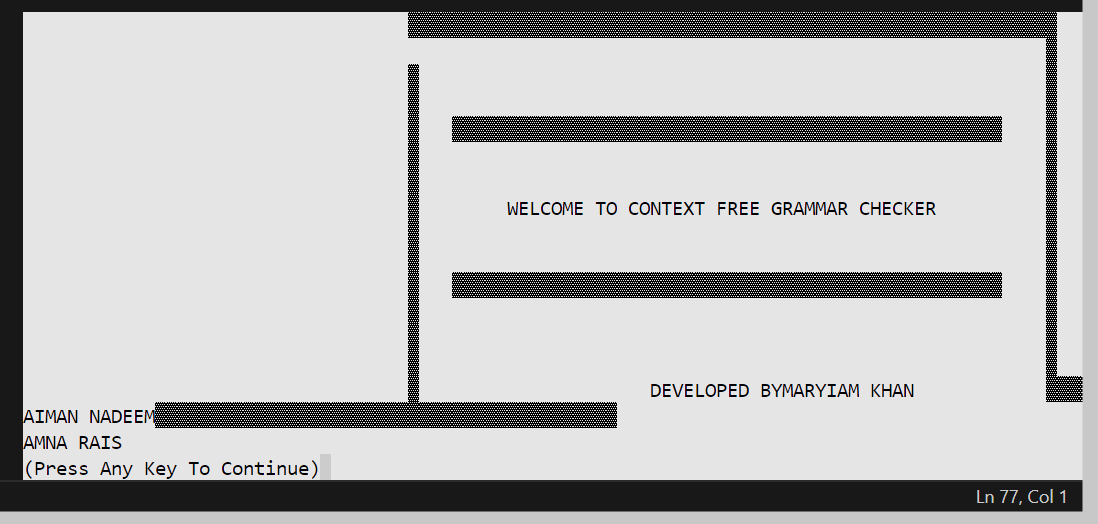
* **Testing and Documentation**: Writing test cases to verify the correctness of the parser and tree operations, as well as ensuring that the overall program functions as intended.
* **Final Documentation**: Writing and formatting the final report, including the methodology, objectives, and the overall working of the system.
* **File: context\_ grammar. h, BST. h, and documentation**: Ensuring that all components are well-documented and testing is done on both the grammar validation and the tree operations.

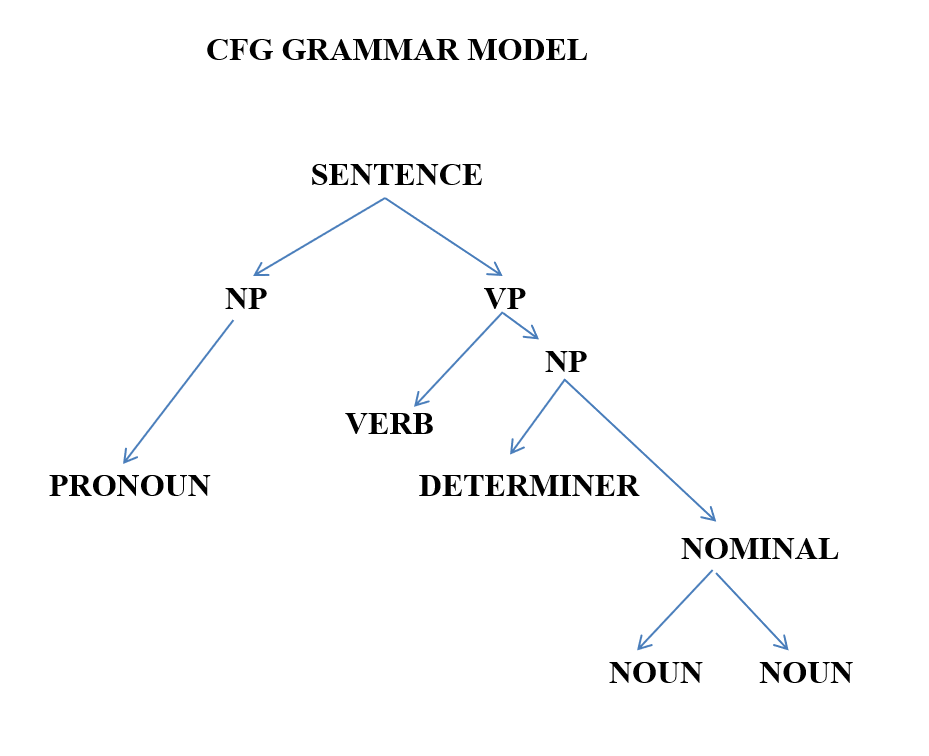
**Challenges and Risks**

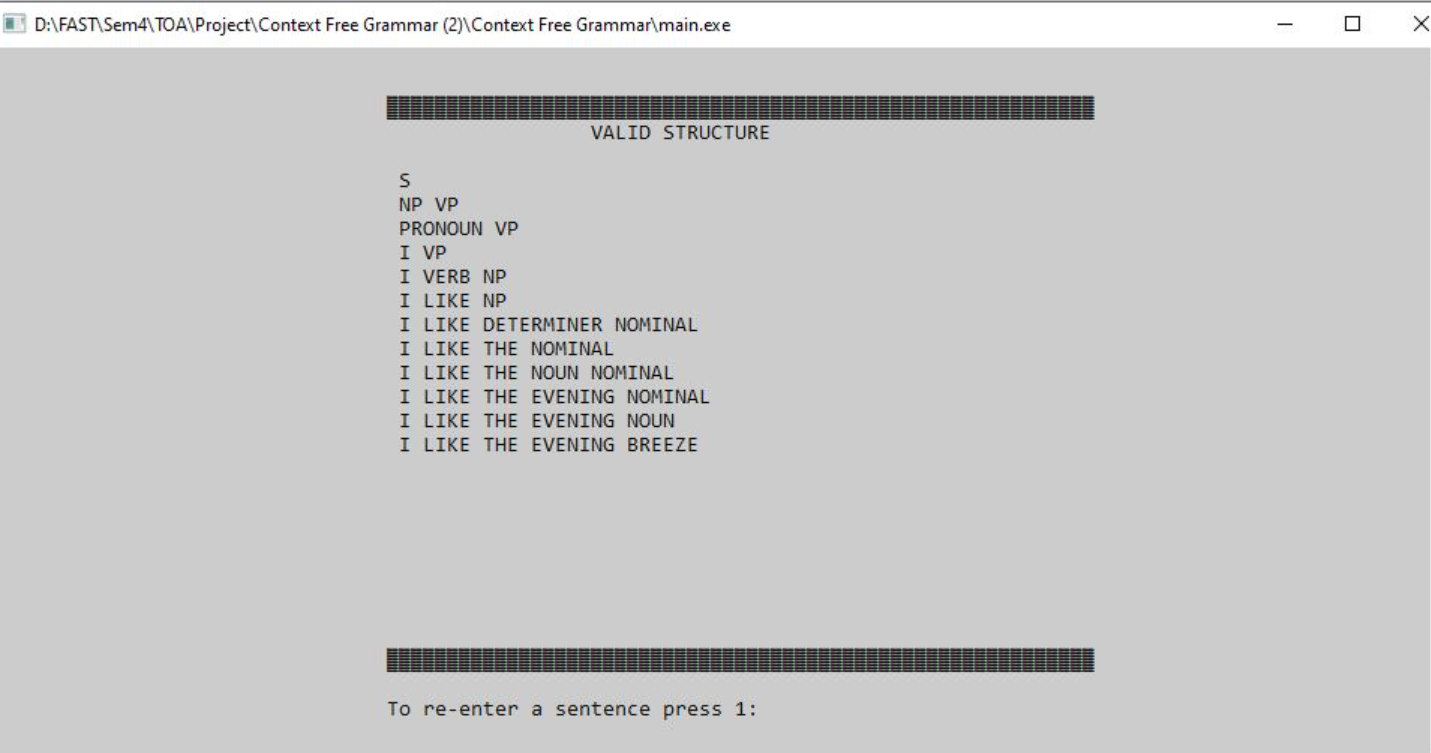
The project faces several challenges, such as managing complex algorithms for parsing context-free grammar (CFG) and maintaining efficient operations on data structures like binary search trees (BST). Issues like state explosion in CFG parsing and memory management are also potential risks, especially with large datasets. Furthermore, concurrency issues may arise if multiple threads are involved, and ensuring thread safety becomes crucial. The project could also face difficulties in designing a simple yet effective user interface, as well as ensuring smooth integration of all components. To mitigate these risks, we plan to use efficient algorithms, proper memory management techniques, and synchronization methods for multithreading. We'll also break the project into smaller modules for easier testing and debugging to ensure smooth execution.

### Visual Demonstration of Project Workflow







### Known Improvements and Future Work

 Enhance the grammar parser to support more complex grammars.

 Optimize performance for handling larger datasets.

 Implement advanced error handling for edge cases.

 Add a user-friendly graphical interface.

 Expand functionality to support more diverse input formats.

### Conclusion

This project plays a significant role in understanding automata theory by providing practical experience with concepts like context-free grammars, parsing, and tree structures. By implementing a grammar parser and utilizing tree-based data structures like Binary Search Trees (BST), it bridges the gap between theoretical concepts and real-world application. The project contributes to a deeper understanding of how automata can be used to process languages and solve practical problems. Future improvements could include supporting more complex grammars, enhancing performance, and adding user-friendly features to further extend its functionality and practical applications.