**SIMATS SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**CHENNAI-602105**

**GRAMMER AUTO CORRECTION**

**A CAPSTONE PROJECT REPORT**

*Submitted in the partial fulfillment for the award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE OF ENGINEERING**

**Submitted by**

**KAVYANJALI.V (192210218)**

**LOHITHA.K (192210219)**

**BHUMIKA.K (192210344)**

**GEETHA MADHURI.D (192210239)**

**Under the Supervision of**

**Dr. K.V.KANIMOZHI**

**JULY 2024**

**DECLARATION**

We, **KAVYANJALI.V, LOHITHA.K, BHUMIKA.K, GEETHA MADHURI.D** students of **‘Bachelor of Engineering’**, Department of Computer Science and Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work entitled **Phases of Compiler** is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics.

(Kavyanjali. V 192210218)

(Lohitha. K 192210219)

(Bhumika. K 192210344)

(Geetha Madhuri. D 192210239)

Date:

Place:

**CERTIFICATE**

This is to certify that the project entitled **“Grammer Auto Correction”** submitted by **Kavyanjali. V , Lohitha. K , Bhumika. K , Geetha Madhuri. D** has been carried out under our supervision. The project has been submitted as per the requirements in the current semester of B. Tech Computer Science.

Teacher-in-charge

Dr. K.V. KANIMOZHI

**Table of Contents**

|  |  |
| --- | --- |
| **S.NO** | **TOPICS** |
| 1 | **Abstract** |
| 2 | **Introduction** |
| 3 | **Problem Statement** |
| 4 | **Proposed Design**   1. Requirement Gathering and Analysis 2. Tool selection criteria 3. Scanning and Testing Methodologies |
| 5. | **Functionality**   1. User Authentication and Role Based Access Control. 2. Tool Inventory and Management 3. Security and Compliance Control |
| 6 | **UI Design**   1. Layout Design 2. Feasible Elements Used 3. Elements Positioning and Functionality |
| 7 | **Conclusion** |

**ABSTRACT**

The aim of this capstone project is to design and implement a Grammar Auto corrector system that automatically identifies and corrects grammatical errors in written text, focusing on input texts of varying lengths. This project outlines the significance of each phase in the development of the Grammar Auto corrector, demonstrating the workflow from data preprocessing to error correction and output generation. The abstract provides an overview of the essential components of the system, emphasizing their roles in transforming erroneous text into grammatically correct sentences while ensuring accuracy and efficiency. Additionally, the project highlights the challenges faced during the design and implementation process, providing insights into how these challenges were addressed to achieve an optimal solution.

**Introduction**

In the field of Natural Language Processing (NLP), a Grammar Auto corrector is a valuable tool that assists users in producing grammatically correct written content. The development of such a system involves multiple complex phases, each playing a critical role in ensuring the final output is accurate and free from grammatical errors. This project aims to provide a detailed understanding of the various stages involved in creating a Grammar Auto corrector, particularly focusing on handling texts with a range of grammatical errors. The goal is to demonstrate how each phase contributes to the overall functionality of the system and to highlight the importance of a well-structured development process. Through this project, we aim to offer a comprehensive overview of the design and implementation of a Grammar Auto corrector, shedding light on both theoretical concepts and practical applications.

**Problem Statement**

Designing a Grammar Auto corrector that efficiently identifies and corrects grammatical errors in written text presents unique challenges. These challenges include ensuring accuracy in identifying grammatical errors, syntactic correctness, semantic validation, and providing appropriate corrections. The objective is to develop a system that not only handles diverse grammatical errors effectively but also highlights the importance of each phase in the overall correction process.

**Proposed Design**

1. **Stakeholder Interviews and Surveys:**

* Understand the requirements of the Grammar Auto corrector and the significance of handling various grammatical errors.
* Gather information about the expected output and performance criteria.

1. **Tool Selection Criteria:**

* Identify the necessary tools and technologies required for developing the Grammar Auto corrector.
* Evaluate different NLP tools and select the most appropriate ones.

1. **Scanning and Testing Methodology:**

* Define the test scenarios and environments for evaluating the system's performance.
* Ensure comprehensive testing to cover various types of grammatical errors.

**Functionality:**

**User Authentication and Role-Based Access Control:**

* Implement user authentication measures to manage access to the system.
* Define roles and permissions to control access based on user responsibilities and authorization levels.

**Tool Inventory and Management:**

* Implement user authentication measures to manage access to the system.
* Define roles and permissions to control access based on user responsibilities and authorization levels.

**Security and Compliance Controls:**

* Implement strong security measures such as encryption, access controls, and audit trails to protect sensitive data.

**Architectural Design:**

**Presentation Layer:**

* A web-based user interface (UI) for interacting with the Grammar auto corrector.
* Use role-based access control (RBAC) to manage user authentication and permissions.

**Application Layer:**

* The business logic layer is responsible for processing user requests and orchestrating system functionality.
* The error correction module identifies and corrects grammatical errors in the text

**Monitoring and Management Layer:**

* Tools for real-time performance monitoring, log analysis, and system health checks.
* Platforms for storing and analyzing system logs in a centralized and aggregated manner.

**UI Design:**

**Dashboard**

* Provides a summary of the system's current state, including the number of corrections performed, recent correction results, and system status indicators.

**User Management**

* Allows administrators to manage user accounts, roles, and permissions.
* Users are assigned roles with predefined permissions to control access to various features and functionalities.

**Help and Support**

* Links to user manuals, tutorials, and documentation materials for understanding how to use the system efficiently.
* Contact details for technical help, FAQs, and community forums for asking questions and sharing best practices.

**Feasible Elements Used**

**Dashboard**

* Tiles/cards that show summary information about the system, such as the number of corrections performed, errors identified, and system condition.

**User Management**

* Table of user accounts, with options for modifying, deleting, and establishing new accounts.
* Users can be assigned roles (such as administrator or user) with related rights using a dropdown menu or checkboxes

**Help and Support**

* Positioned on the dashboard to provide real-time monitoring of the system.
* Widgets offer live statistics such as active corrections, identified errors, and system resource utilization.

**Element Positioning and Functionality**

**Real-time Monitoring**

* Positioned on the dashboard to provide real-time monitoring of the system.
* Widgets offer live statistics such as active corrections, identified errors, and system resource utilization.

**Collaboration Features**

* Located within correction results or reports.
* Allowing users to post comments, annotations, or notes on specific correction outcomes enables team members to collaborate and share knowledge more effectively.

**Trend Analysis**

* positioned in the reporting and analysis area.
* The functionality offers interactive charts or graphs for visualizing patterns in correction results over time, such as the frequency of identified errors or changes in correction accuracy.

**Conclusion:**

The Grammar Auto corrector system is designed with a comprehensive set of features and functionalities to facilitate efficient grammar correction, analysis, and management. By strategically arranging UI elements and implementing straightforward functionality, the system enables users to identify and correct grammatical errors, create analysis criteria, examine results, and communicate effectively. The dashboard provides an instant overview of system state and recent activity, while real-time monitoring, interactive analysis tools, and trend analysis allow users to gain a deeper understanding of correction performance and errors. This project not only highlights the critical phases of grammar correction but also demonstrates the practical implementation of these phases in a user-friendly and robust system.

**PROGRAM**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX\_RULES 100

#define MAX\_LENGTH 100

int isNonTerminal(char ch);

int isValidTerminal(char ch);

int isValidRightHandSide(char \*rhs);

void checkGrammar(char rules[][MAX\_LENGTH], int numRules);

int main() {

int numRules;

char rules[MAX\_RULES][MAX\_LENGTH];

char originalRules[MAX\_RULES][MAX\_LENGTH];

printf("Enter the number of grammar rules: ");

scanf("%d", &numRules);

getchar();

printf("Enter each rule in the format 'A -> aB' (one per line):\n");

for (int i = 0; i < numRules; i++) {

fgets(rules[i], MAX\_LENGTH, stdin);

rules[i][strcspn(rules[i], "\n")] = '\0';

strcpy(originalRules[i], rules[i]);

}

checkGrammar(rules, numRules);

return 0;

}

int isNonTerminal(char ch) {

return isupper(ch);

}

int isValidTerminal(char ch) {

return islower(ch) || ispunct(ch);

}

int isValidRightHandSide(char \*rhs) {

for (int j = 0; j < strlen(rhs); j++) {

if (!isNonTerminal(rhs[j]) && !isValidTerminal(rhs[j])) {

return 0;

}

}

return 1;

}

void checkGrammar(char rules[][MAX\_LENGTH], int numRules) {

for (int i = 0; i < numRules; i++) {

char left[MAX\_LENGTH], right[MAX\_LENGTH];

char \*token;

char fullRule[MAX\_LENGTH];

strcpy(fullRule, rules[i]);

token = strtok(rules[i], " ->");

if (token == NULL) {

printf("\n%s IS INVALID GRAMMAR\n", fullRule);

continue;

}

strcpy(left, token);

token = strtok(NULL, " ->");

if (token == NULL) {

printf("\n%s IS INVALID GRAMMAR\n", fullRule);

continue;

}

strcpy(right, token);

if (strlen(left) != 1 || !isNonTerminal(left[0])) {

printf("\n%s IS INVALID GRAMMAR\n", fullRule);

printf("CORRECTED GRAMMAR: A -> %s\n", right);

continue;

}

if (isValidRightHandSide(right)) {

printf("\n%s IS A VALID GRAMMAR\n", fullRule);

} else {

printf("%s IS INVALID GRAMMAR\n\n", fullRule);

printf("CORRECTED GRAMMAR: %s -> ", left);

for (int j = 0; j < strlen(right); j++) {

if (isNonTerminal(right[j]) || isValidTerminal(right[j])) {

printf("%c", right[j]);

}

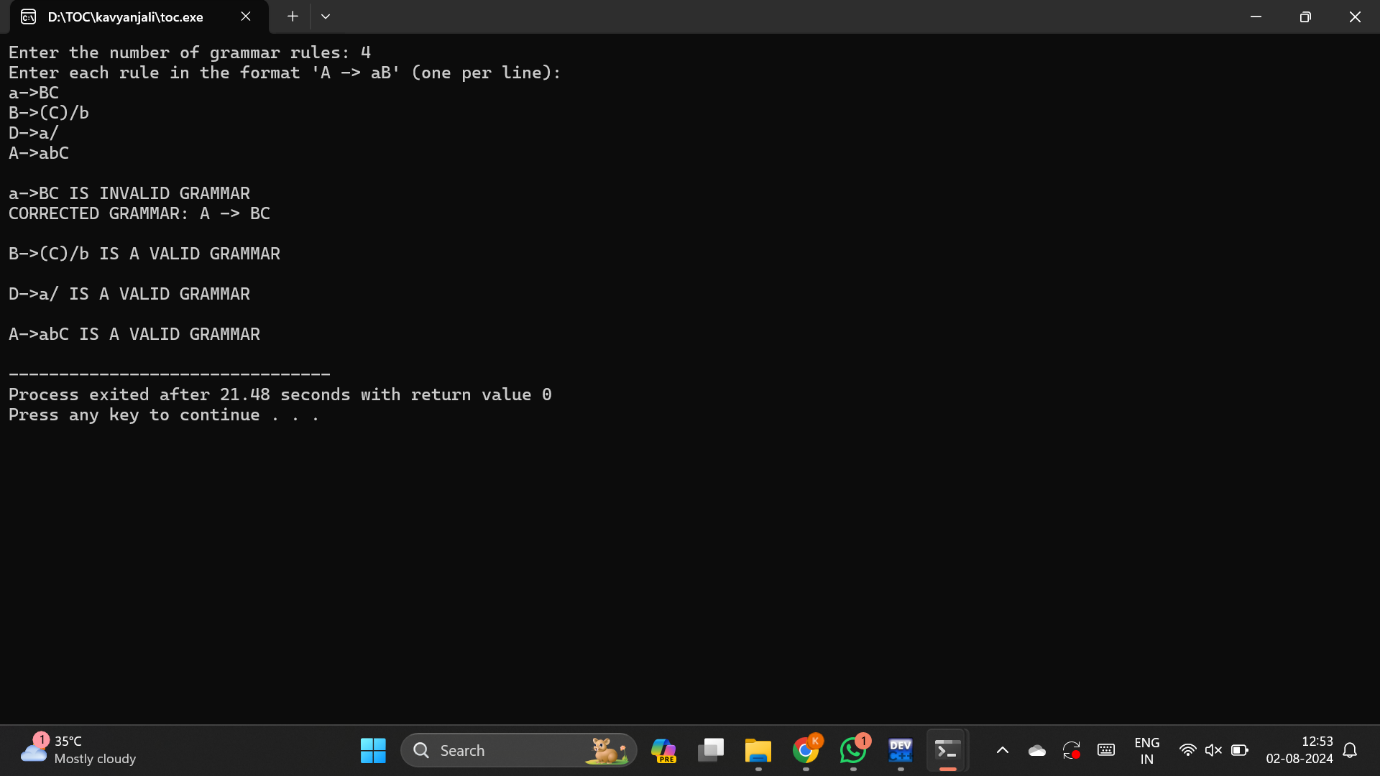
}

printf("\n");

}

}

}

**OUTPUT**