



SIMATS SCHOOL OF ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES
CHENNAI-602105



ERROR HANDLING AND RECOVERY STRATEGIES IN COMPILER DESIGN

A CAPSTONE PROJECT REPORT

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

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE ENGINEERING

Submitted By

K.MANI SAKETH (192211670)

TEAM MEMBERS

S.DANIEL (192211637)

K.MANI SAKETH (192211670)

T.ANOOP CHANDAR(192210244)

Under the Supervision of

DR.S.SANKAR

DECLARATION

We, **K.MANI SAKETH, S.DANIEL, ANOOP CHANDRAN** students of **‘Bachelor of Engineering in Information Technology**, 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 **Error handling and recovery strategies in compiler design** 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.

K.MANI SAKETH(192211670)

S.DANIEL (192211637)

ANOOP CHANDRAN(192210244)

CERTIFICATE

This is to certify that the project entitled **“Error handling and recovery strategies in compiler design ”** submitted by **K.MANISAKETH, S.DANIEL, ANOOP CHANDRAN** has been carried out under our supervision. The project has been submitted as per the requirements in the current semester of B. Tech Information Technology.

Teacher-in-charge

Dr.S.SANKAR

S.NO	TOPICS
1	Abstract
2	Introduction
3	Problem Statement
4	Proposed Design <ul style="list-style-type: none">1. Requirement Gathering and Analysis2. Tool selection criteria3. Scanning and Testing Methodologies
5.	Functionality <ul style="list-style-type: none">1.User Authentication and Role Based Access Control.2.Tool Inventory and Management3.Security and Compliance Control
6	UI Design <ul style="list-style-type: none">1. Layout Design2. Feasible Elements Used3. Elements Positioning and Functionality
7	RESULT <ul style="list-style-type: none">1.CODE
8.	CONCLUSION

ABSTRACT

Error handling and recovery strategies are fundamental aspects of compiler design projects, essential for ensuring the reliability and resilience of the compilation process. This project abstract explores the importance and implementation of these strategies, which encompass various techniques to detect, report, and recover from errors encountered during compilation. From lexical analysis to semantic validation, compilers utilize a range of mechanisms to identify errors in the source code, providing informative feedback to users. Additionally, error recovery strategies such as panic mode recovery and automatic correction allow compilers to gracefully handle errors and continue compilation whenever possible, minimizing disruptions. The project aims to design, implement, and evaluate these error handling mechanisms, aiming to enhance the usability and robustness of the compiler. Through comprehensive testing and validation, the project seeks to ensure the effectiveness and reliability of the error handling and recovery strategies, ultimately contributing to the development of a more efficient and dependable compiler system.

INTRODUCTION

Compiler design must have error management and recovery in order for the compiler to be able to handle these interruptions efficiently. Errors encountered during compilation can be broadly categorized into three types:

1. **Lexical errors:** These happen when the compiler comes across characters or sequences that don't match any token specified in the language grammar during the lexical analysis stage. Illegal identifiers or unidentified symbols are two examples.
2. **Syntax errors:** These happen when a sequence of tokens deviates from the programming language's grammatical norms during the syntax analysis stage. Missing semicolons, mismatched parentheses, and improper statement structures are examples of common syntax problems.
3. **Semantic Errors:** These happen when the compiler finds problems with the logic or meaning of the code during the semantic analysis stage. Type inconsistencies, unreported variables, and scope violations are a few examples.

It is necessary to properly construct error handling procedures to handle every one of these types of errors. The compiler may continue scanning after ignoring unknown characters if it detects lexical mistakes. The compiler can

recover and continue parsing when there are syntax problems by using strategies like error productions and synchronized parsing. When a semantic error occurs, the compiler can assist developers fix their programs' logic by giving them thorough warnings that explain the type of issue and where it occurred.

PROBLEM STATEMENT

Effective error handling and recovery strategies are vital in compiler design projects. Challenges include detecting errors at various compilation stages, providing informative feedback to users, implementing recovery mechanisms, ensuring usability, and enhancing robustness. Addressing these challenges requires a deep understanding of compiler design principles and proficiency in software engineering. By tackling error handling effectively, compiler designers can enhance the reliability, efficiency, and usability of their systems, empowering software developers to build more robust applications.

PROPOSED DESIGN

Requirement Gathering and Analysis

- Identify stakeholders involved in the project, including developers, users, project managers, and quality assurance teams.
- Gather user stories and use cases to understand typical error scenarios during compilation and the expected behavior of the compiler.
- Classify errors into categories based on severity and impact on the compilation process, distinguishing between critical and non-critical errors
- Define the information users need when an error occurs, including error messages, error location, and suggested resolution. Design clear and consistent error message formats.
- Analyze compilation stages where errors can occur and determine efficient and accurate techniques for error detection
- Explore different error recovery strategies such as panic mode recovery, automatic correction, and error-tolerant parsing. Evaluate effectiveness and complexity trade-offs

TOOL SELECTION CRITERIA

- Ensure compatibility with the target platform, programming languages, and development environment.
- Evaluate tools based on their support for error detection, reporting, and recovery mechanisms relevant to compiler design
- Choose tools that seamlessly integrate with existing development workflows and toolchains
- Select tools capable of scaling to accommodate larger codebases and growing project requirements.
- Prioritize tools with active user communities and robust support resources for learning and troubleshooting.
- Choose tools with a track record of reliability and stability to minimize disruptions during development.

SCANNING AND TESTING METHODOLOGIES

- Supplement automated testing with manual testing techniques to validate user interfaces, error messages, and user interactions.
- Implement automated testing frameworks and tools to streamline the testing process and improve productivity.
- Develop comprehensive test suites covering a wide range of error scenarios, including syntactic errors, semantic errors, and error recovery situations.
- Generate test cases based on input partitions, equivalence classes, and boundary values to ensure thorough coverage of error scenarios
- Conduct syntactic analysis to parse the code and identify syntax errors based on the grammar rules of the programming language.
- Employ dynamic analysis techniques to detect errors by executing the compiled code and observing its behavior.

FUNCTIONALITY

User Authentication and Role Based Access Control

- User authentication ensures that individuals attempting to access the system are who they claim to be, typically through methods like passwords, biometrics, or security tokens.
- RBAC controls access to system resources based on predefined roles assigned to users. Each role carries a set of permissions dictating what actions users with that role can perform
- Authentication methods include traditional username/password, multi-factor authentication (MFA), and single sign-on (SSO), providing varying levels of security and user convenience
- Robust encryption, secure storage of credentials, and adherence to best security practices are essential to prevent unauthorized access and protect user data.
- RBAC consists of roles, permissions, and access control lists (ACLs). Roles define sets of permissions, while ACLs associate roles with specific resources and actions.
- Integration with centralized user management systems, fine-grained access control, secure session management, and logging/auditing are crucial implementation aspects to ensure effective user authentication and RBAC

Tool Inventory and Management

- Identify all software tools, hardware devices, and services used in the organization's workflows, including development, testing, and deployment tools
- Maintain comprehensive documentation for each tool, including its purpose, version, license details, installation instructions, and configuration settings.
- Establish a centralized inventory tracking system to monitor the usage, availability, and status of all tools within the organization.
- Implement version control for software tools to track changes, updates, and modifications over time, ensuring consistency and reliability.
- Manage software licenses effectively, ensuring compliance with legal requirements and optimizing license usage to minimize costs

Security and Compliance Control

- Identify relevant regulations and standards applicable to the organization's operations.
- Conduct risk assessments to identify security threats and compliance gaps.
- Develop and implement comprehensive security policies and procedures.
- Implement access controls and data protection measures to restrict unauthorized access and protect sensitive data.
- Establish incident response procedures and security monitoring mechanisms to detect and respond to security incidents promptly.

ARCHITECTURAL DESIGN

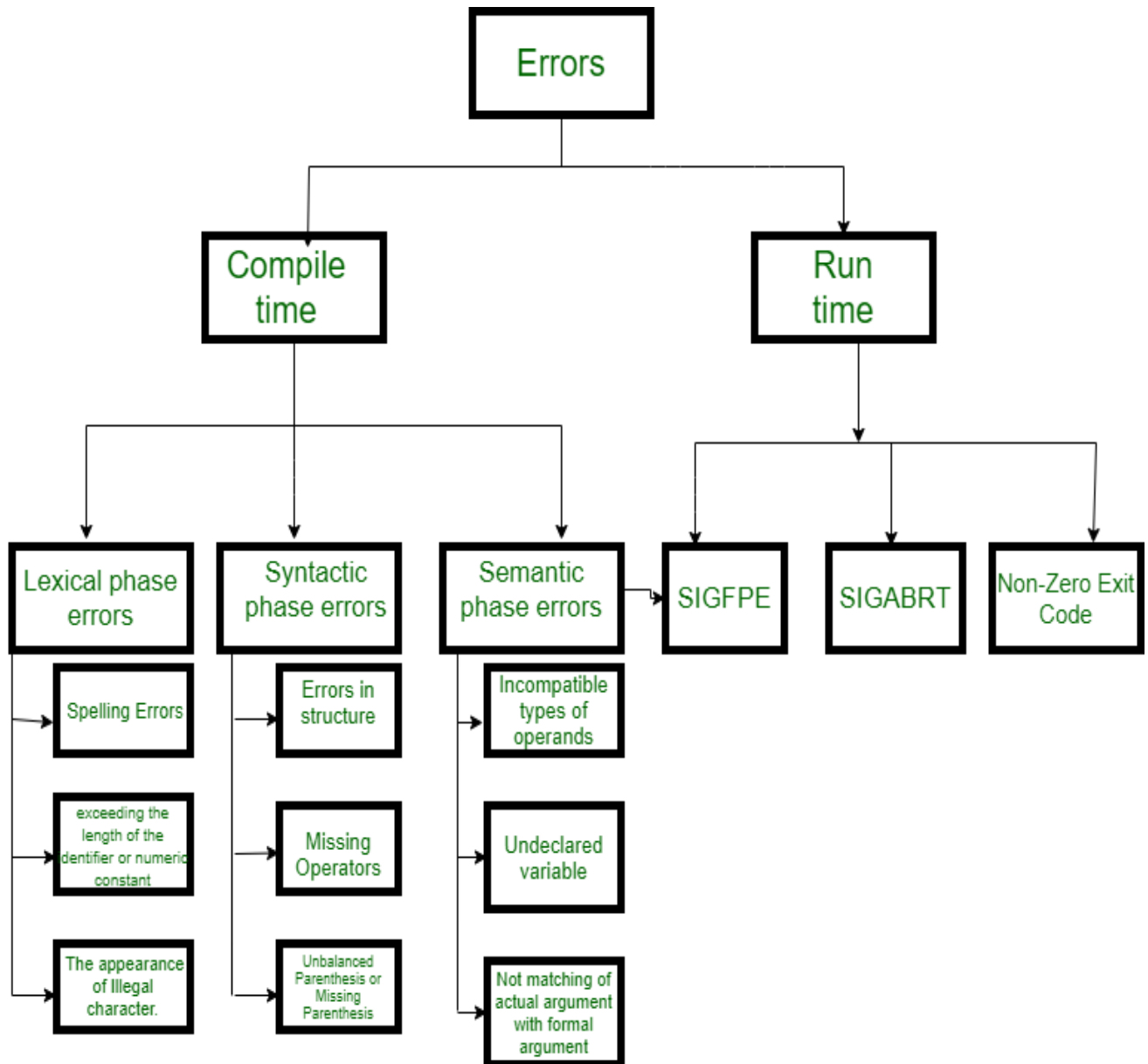
Presentation Layer:

- The presentation layer, often referred to as the user interface (UI) layer, is a component of software architecture responsible for presenting information to users and facilitating interaction with the underlying system.
- It encompasses graphical user interfaces (GUIs), web interfaces, and other forms of interaction, focusing on delivering a visually appealing, intuitive, and user-friendly experience.
- Through elements like forms, buttons, menus, and widgets, the presentation layer interprets data from the application and formats it into a comprehensible interface, fostering efficient communication between users and the software.

.Application Layer:

- The application layer serves as the core functionality of software, executing business logic, data manipulation, integration, and transaction management.
- It ensures security, handles errors, manages concurrency, and offers availability.
- Customization to meet specific business requirements, it plays a pivotal role in the overall software architecture, facilitating data processing and user interaction.
- The monitoring and management layer oversees system health and performance, generating alerts, managing configurations, optimizing resources, and ensuring security and compliance.

- It automates routine tasks, facilitates troubleshooting, and supports business continuity through backup and recovery mechanisms, ensuring the smooth operation of the software system.



UI DESIGN:

Dashboard:

- Utilize charts, graphs, and tables to visually represent key metrics and trends for easy comprehension.
- Allow users to customize the dashboard layout, choose specific metrics, and adjust visualizations according to their preferences.
- Incorporate interactive elements such as filters, drill-downs, and tooltips to enable users to explore data and gain deeper insights.
- Provide real-time or near-real-time updates to ensure that users have access to the latest information and insights.
- Design the dashboard to be responsive across different devices and screen sizes, ensuring optimal viewing and interaction experiences.
- Maintain consistency in design elements, color schemes, and typography to enhance usability and user experience.

User Management:

- User management involves verifying user identities, registering accounts, assigning roles and permissions, managing profiles, controlling access, and facilitating onboarding/offboarding.
- It ensures secure and efficient user interactions with the system while maintaining data integrity and confidentiality.

Help and Support:

- Links to user manuals, tutorials, and documentation materials for understanding how to utilize the assessment framework efficiently.
- Contact details for technical help, FAQs, and community forums for asking questions and sharing best practices.

Feasible Element Used:

Dashboard:

- A dashboard is a visual representation of key metrics, trends, and data, offering users an at-a-glance view of critical information.
- It provides a centralized hub for monitoring and analyzing performance, enabling quick decision-making and action.
- Dashboards typically incorporate data visualization techniques such as charts, graphs, and tables to present information in a clear and concise manner, facilitating insights and understanding

User Management:

- User management involves overseeing user activities within a system, including authentication, registration, role assignment, access control, and profile management.
- It ensures proper handling of user accounts, permissions, and data security, facilitating smooth interactions and maintaining system integrity.

Help and Support:

- Help and support services offer assistance to users encountering difficulties or seeking guidance with software products or services.
- They encompass resources such as FAQs, knowledge bases, tutorials, and customer service channels like chat support, email, and phone assistance.
- These services aim to address user queries, troubleshoot problems, and ensure a positive user experience.

Element Positioning and Functionality:

Real-time Monitoring:

- Real-time monitoring provides immediate insights into system performance, positioned prominently on a dashboard or dedicated page.
- It continuously tracks live data, triggers alerts for anomalies, and offers customization and interactivity for users to explore specific metrics.

- Integrated with other system components, it supports informed decision-making and troubleshooting.

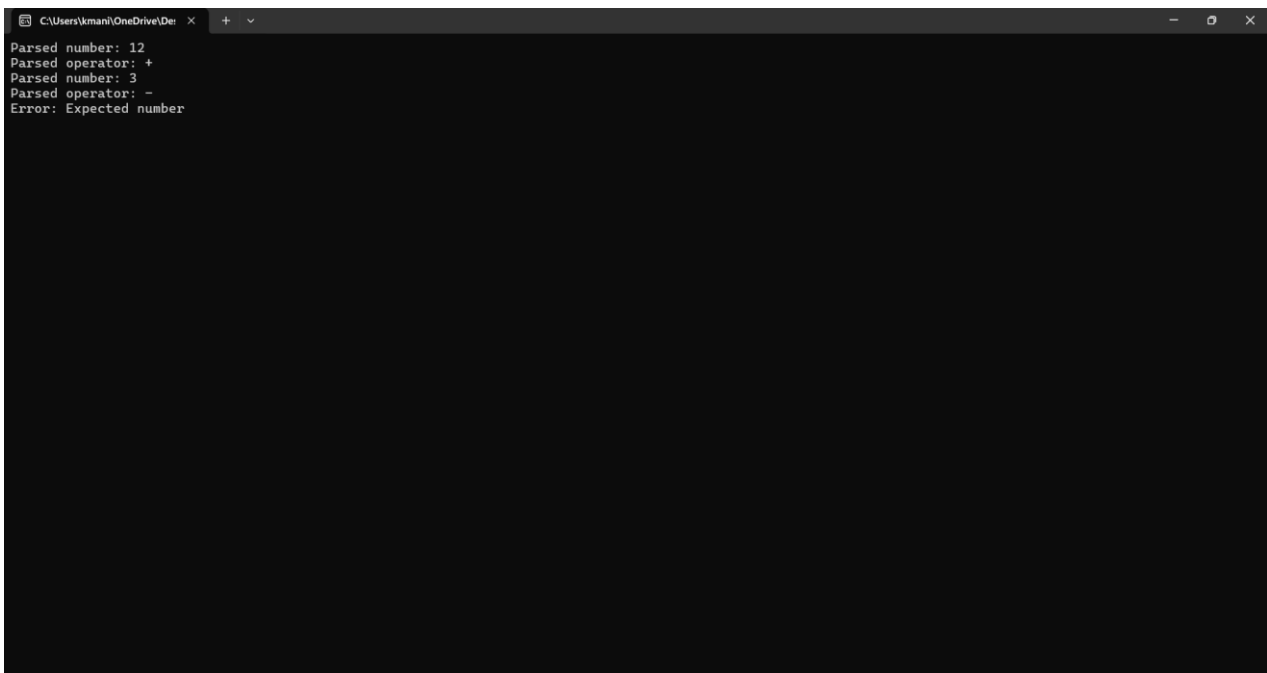
Collaboration Features:

- Collaboration features facilitate teamwork and communication among users within a software platform.
- They include tools such as real-time document editing, commenting, file sharing, task assignment, version control, and communication channels like chat, video conferencing, and discussion forums.
- These features promote collaboration, streamline workflows, and enhance productivity by enabling seamless interaction and coordination among team members.

Trend Analysis:

- Trend analysis involves examining historical data over time to identify patterns, trends, and correlations.
- It helps in forecasting future outcomes, making informed decisions, and detecting anomalies or changes in data behavior.
- Trend analysis is commonly used in various fields such as finance, marketing, and operations to understand market trends, customer behavior, and business performance, enabling organizations to adapt strategies and capitalize on opportunities.

RESULT:



```
C:\Users\kmanil\OneDrive\Desktop
Parsed number: 12
Parsed operator: +
Parsed number: 3
Parsed operator: -
Error: Expected number
```

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
```

```
typedef enum {
    TOKEN_PLUS,
    TOKEN_MINUS,
    TOKEN_NUMBER,
    TOKEN_END,
    TOKEN_ERROR
} TokenType;
```

```
typedef struct {
    TokenType type;
    int value;
} Token;
```

```
Token getNextToken(const char **input) {
    Token token;
    while (isspace(**input)) (*input)++;
    if (**input == '\0') {
        token.type = TOKEN_END;
    } else if (**input == '+') {
        token.type = TOKEN_PLUS;
        (*input)++;
    } else if (**input == '-') {
        token.type = TOKEN_MINUS;
        (*input)++;
    } else if (isdigit(**input)) {
        token.type = TOKEN_NUMBER;
        token.value = strtol(*input, (char **)input, 10);
    } else {
        token.type = TOKEN_ERROR;
    }
    return token;
}
```

```
void parseExpression(const char **input);
```

```
void parseTerm(const char **input) {
    Token token = getNextToken(input);
    if (token.type == TOKEN_NUMBER) {
        printf("Parsed number: %d\n", token.value);
    } else {
        printf("Error: Expected number\n");
        // Panic mode recovery: skip to next plus or minus sign
    }
}
```

```

        while (token.type != TOKEN_PLUS && token.type != TOKEN_MINUS
&& token.type != TOKEN_END) {
            token = getNextToken(input);
        }
    }
}

void parseExpression(const char **input) {
    parseTerm(input);
    Token token = getNextToken(input);
    while (token.type == TOKEN_PLUS || token.type == TOKEN_MINUS) {
        printf("Parsed operator: %c\n", (token.type == TOKEN_PLUS) ? '+' : '-');
        parseTerm(input);
        token = getNextToken(input);
    }
    if (token.type != TOKEN_END) {
        printf("Error: Unexpected token\n");
    }
}

int main() {
    const char *input = "12 + 3 - abc + 45";
    parseExpression(&input);
    return 0;
}

```

Conclusion

In conclusion, the elements discussed, including real-time monitoring, collaboration features, trend analysis, and others, are essential components of modern software systems. They empower users to make informed decisions, enhance productivity, and drive business growth. By incorporating these elements into software design and development processes, organizations can create robust and user-centric solutions that meet the evolving needs of users and markets. Embracing innovation and leveraging technology trends will be key to staying competitive and achieving success in today's dynamic business landscape.