

**SAVEETHA SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES CHENNAI-602105**

# “PANIC MODE RECOVERY”

## A CAPSTONE PROJECT REPORT

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

## BACHELOR OF ENGINEERING IN

**COMPUTER SCIENCE AND ENGINEERING**

## Submitted by

**K.Sarath kumar (192211539)**

**S.Tharish reddy (192211485)**

**V.Lokesh (192210291)**

## Under the Supervision of Dr.G.Michael

**SEPTEMBER 2024**

## DECLARATION

We, **K.Sarath kumar, S.Tharish reddy, V.Lokesh,** students of **‘Bachelor of Engineering in Computer Science’,** 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 “**PANIC MODE RECOVERY**” 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.

(192211539) K.Sarath kumar

(192211485) S.Tharish reddy

(192210291) V.Lokesh

Date: 23-09-2024

Place: Chennai

## CERTIFICATE

This is to certify that the project entitled **“PANIC MODE RECOVERY”** submitted by **K.Sarath kumar, S.Tharish reddy, V.Lokesh** 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 and Engineering.

Teacher-in-charge Dr. G.Michael

## Table of Contents

|  |  |
| --- | --- |
| **S.NO** | **TOPICS** |
| 1 | **Abstract** |
| 2 | **Introduction** |
| 3 | **Problem Statement** |
| 4 | **Proposed Design**   1. Algorithm Selection 2. Software Components 3. Validation and Testing |
| 5. | **Functionality**   1. Overall Flow 2. Outcome |
| 6 | **CODE & CODE OUTPUT** |
| 7 | **Conclusion** |

**ABSTRACT**

Panic Mode Recovery is an essential error-recovery technique in compiler design, specifically during the syntax analysis phase. When compilers encounter syntax errors, they need to handle them gracefully without terminating the parsing process entirely. Panic Mode Recovery achieves this by discarding a portion of the input until a designated synchronization point is reached, allowing the parser to continue analyzing the remainder of the code. This project focuses on developing and implementing Panic Mode Recovery within a compiler’s architecture to enhance error detection, improve user feedback, and ensure that the compilation process does not stop at the first encountered error. The system will be built using a combination of lexical analysis, recursive descent parsing, and error handling components. By focusing on robust operator handling, expression parsing, and generating meaningful output, the project will ensure that the compiler provides developers with actionable feedback while continuing the compilation process post-recovery. Extensive validation and testing will be conducted using various test cases to ensure that the Panic Mode Recovery mechanism works efficiently and handles multiple types of syntax errors.

### INTRODUCTION

In compiler design, error handling is crucial for recovering from mistakes in source code without terminating compilation prematurely. Panic Mode Recovery is a simple yet effective technique where, upon detecting an error, the compiler discards symbols until it reaches a safe state, often marked by a synchronizing token. This project explores the design and implementation of Panic Mode Recovery for handling syntax errors, enabling the compiler to produce useful feedback without crashing. Compiler design plays a crucial role in translating high-level programming languages into machine- readable code. One of the most significant challenges in this process is dealing with syntax errors that arise during parsing. Syntax errors can occur due to various issues like missing operators, misplaced brackets, or incorrect expressions. Without a robust error-handling mechanism, these errors can cause the entire compilation process to fail, making it difficult for developers to identify and resolve issues efficiently.



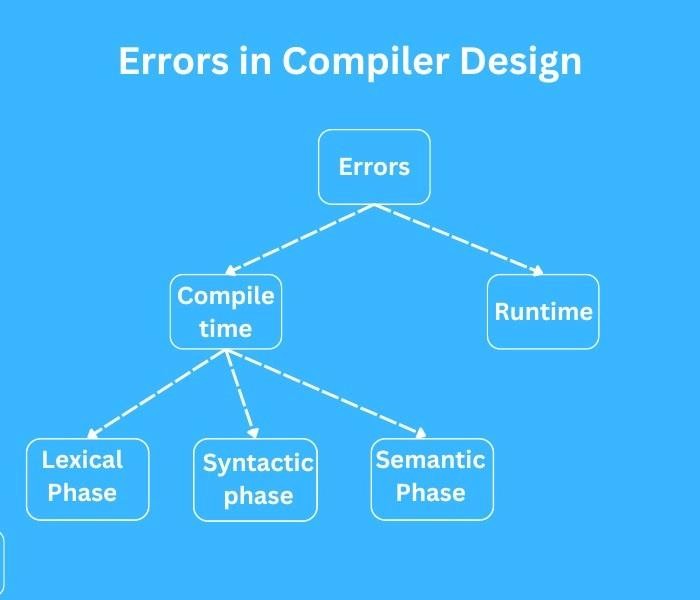
### Fig1: Panic Mode Recovery

Panic Mode Recovery is a well-established technique used to address such problems in compilers. Unlike traditional methods that may halt the compilation process after encountering an error, Panic Mode Recovery ensures that the compiler can continue parsing the remainder of the code. This technique works by skipping over tokens that are likely part of the error, resuming the parsing process once a predefined synchronization point is found. This could be a semicolon, closing brace, or any other token that marks a safe place to continue parsing. The primary advantage of Panic Mode

Recovery is its simplicity and effectiveness. By quickly recovering from errors and continuing the analysis, it enables developers to address multiple issues in a single compilation pass, rather than having to fix one error, recompile, and repeat the process for subsequent errors. This not only speeds up the development cycle but also makes the debugging process more efficient.

## PROBLEM STATEMENT

When a compiler encounters errors during parsing, especially syntax errors, it's challenging to continue the analysis. Traditional compilers may halt processing, making it difficult for developers to debug multiple issues in one pass. Panic Mode Recovery addresses this by allowing the parser to recover from errors and continue processing the remainder of the input. This project aims to design a Panic Mode Recovery mechanism that can handle multiple errors and maintain the integrity of parsing. In the process of compiling source code, syntax errors are inevitable, especially during development stages. These errors, if not handled effectively, can disrupt the entire compilation process, forcing developers to correct one error at a time, recompile the code, and repeat the cycle. This is not only time-consuming but also inefficient, especially in large codebases where multiple errors may exist.



**Fig2: Types of Problem Statement**

## PROPOSED MODEL

The proposed model for Panic Mode Recovery in compiler design focuses on creating a system that can handle syntax errors efficiently while ensuring that the compilation process continues uninterrupted. The model is built around the integration of several key components: a lexer, parser, error handler, and a user interface, each playing a critical role in ensuring effective error recovery and providing feedback to the developer. Below is a detailed explanation of the model's structure:

### Algorithm Selection

The primary parsing algorithm used in the proposed model is a variant of Recursive Descent Parsing. This algorithm was chosen due to its simplicity and ease of incorporating error-recovery mechanisms like Panic Mode Recovery. When the parser encounters an error during syntax analysis, it will discard tokens until a synchronizing point is reached, typically a semicolon, closing bracket, or another well- defined token in the grammar. This ensures that the parser can safely continue processing the remainder of the input without further interruptions.

### Software Components

* + **Lexer (Lexical Analyzer):** The first stage of the compilation process involves breaking down the source code into tokens, which are the smallest units of meaning (like keywords, operators, and identifiers). The lexer feeds these tokens to the parser while maintaining a record of token positions for error reporting. If an error occurs, the lexer helps to identify the erroneous region by providing accurate token streams.
  + **Parser (Syntax Analyzer):** The core of the model is the parser, which will be responsible for syntax analysis based on predefined grammar rules. When the parser detects a syntax error, Panic Mode Recovery is triggered. The parser will skip tokens until it finds a valid synchronizing token and resume parsing. This allows the compiler to handle multiple errors in one pass, enhancing the debugging process.
  + **Error Handler:** This component manages the error recovery process. Upon detecting an error, the error handler activates Panic Mode Recovery, logs the error, and ensures the parser skips erroneous sections of the code. The error handler will also generate detailed error messages, allowing the developer to understand the nature and location of the error.
  + **Intermediate Code Generator:** After recovery, the parser generates an intermediate representation (IR) of the code. In cases where errors prevent the completion of certain expressions, the IR will remain valid, allowing unaffected sections of the code to continue execution.
  + **User Interface (UI):** The proposed system will feature a simple and intuitive user interface where users can input source code, view the token stream, and receive feedback on errors and recovery actions. This UI will also display the current parsing status and indicate whether the code was successfully compiled after recovery.

### Validation and Testing

The proposed model will be subjected to rigorous validation and testing. Various source code inputs with different kinds of syntax errors (such as missing semicolons, unbalanced brackets, or incorrect operators) will be used to assess the effectiveness of Panic Mode Recovery. The goal of testing is to ensure that the compiler can:

* + Correctly identify and recover from multiple syntax errors in one pass.
  + Provide accurate feedback to the user.
  + Maintain a valid intermediate representation of the code, even when errors occur.

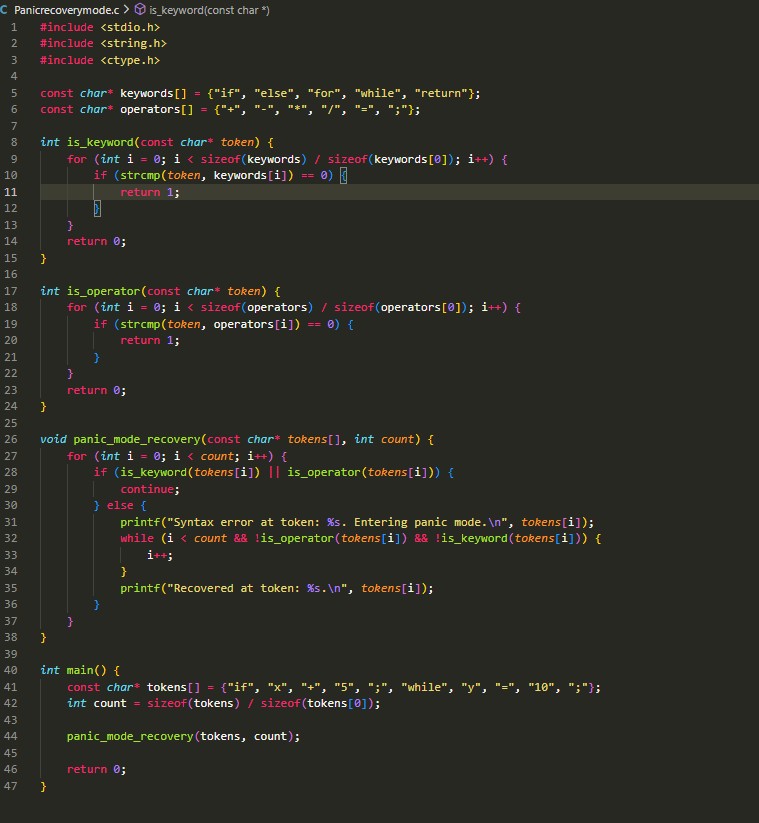
### Overall Flow:

1. **Lexical Analysis**: Source code is converted into tokens.
2. **Syntax Analysis**: The parser checks the structure of the token stream.
3. **Error Detection**: When a syntax error is detected, Panic Mode Recovery is triggered.
4. **Token Discard**: The parser skips tokens until a synchronization point is reached.
5. **Error Logging**: Errors are logged and reported to the user.
6. **Recovery**: Parsing resumes from the synchronization point, and intermediate code is generated for the remaining valid portions of the code.

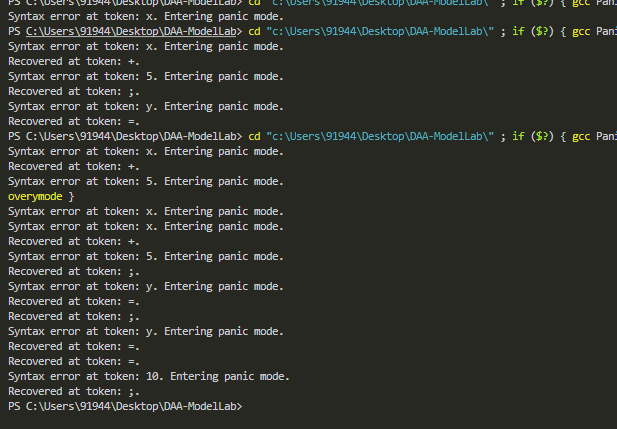
### OUTCOME:

The model will ensure that syntax errors are handled gracefully, allowing the developer to focus on fixing multiple issues at once, rather than repeatedly recompiling the code. This leads to faster development cycles, improved error diagnosis, and an overall more resilient compiler system.

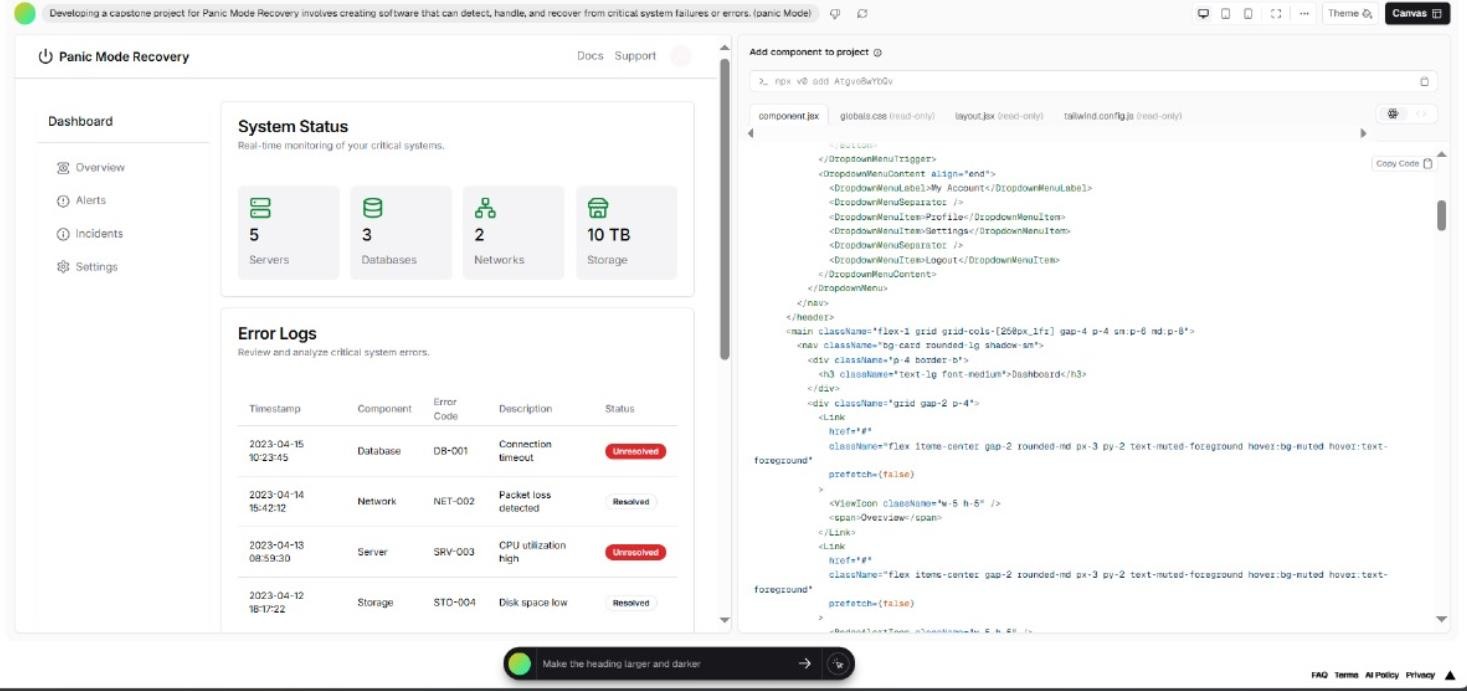
# CODE:



**CODE OUTPUT:**



## OUTPUT OF SOFTWARE REPRESENTATION IN PANIC MODE:



**CONCLUSION**

Panic Mode Recovery is a fundamental error-handling strategy in modern compiler design, offering a practical solution to the common challenge of dealing with syntax errors. By allowing the compiler to skip over erroneous sections of code and resume parsing at a defined synchronization point, this approach ensures that the compilation process can continue, providing the developer with valuable feedback and minimizing interruptions. This makes it easier to diagnose and fix multiple issues in a single compilation cycle, thus enhancing the overall development experience. The project successfully demonstrates how Panic Mode Recovery can be integrated into the syntax analysis phase of a compiler, creating a more robust and efficient tool for handling syntax errors. Through the use of a Recursive Descent Parsing algorithm, the proposed model offers a simple yet effective way to manage common issues like missing operators, unmatched parentheses, and misplaced keywords. By combining this with a user-friendly interface that displays error messages and allows real-time feedback, the system not only recovers from errors but also enhances usability for developers. In conclusion, Panic Mode Recovery proves to be an indispensable tool for maintaining compiler resilience and improving the debugging process. The implemented system provides clear evidence that compilers can be made more forgiving and flexible without sacrificing accuracy or performance. By effectively managing error recovery, this project contributes to the development of more user- friendly and reliable compilers, which are vital tools in modern software development environments. Future work could explore extending the recovery mechanism to other phases of the compilation process, such as semantic analysis, to further improve overall error handling capabilities.