

**Compiler for Math Puzzle Solver**

**CAPSTONE PROJECT REPORT:**

***Submitted to***

***CSA1429 Compiler Design for industrial automation***

# SAVEETHA SCHOOL OF ENGINEERING

***By***

***K Gayathri* (192311448)**

# Supervisor

**Dr. G. Michael**

BONAFIDE CERTIFICATE

**k. Gayathri** student of 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 **Compiler for Math Puzzle Solver**

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.

Date:20-03-2025 Student Name:k.Gayathri

Place:Chennai Reg.No:192311448

**Faculty In Charge**

**Internal Examiner External Examiner**

**ACKNOWLEDGMENTS**

We wish to express our sincere thanks. Behind every achievement lies an unfathomable sea of gratitude to those who actuated it; without them, it would never have existed. We sincerely thank our respected founder and Chancellor, **Dr N.M. Veeraiyan,** Saveetha Institute of Medical and Technical Science, for his blessings and for being a source of inspiration. We sincerely thank our Pro-Chancellor, **Dr Deepak Nallaswamy Veeraiyan**, SIMATS, for his visionary thoughts and support. We sincerely thank our vice-chancellor, Prof. **Dr S.** **Suresh Kumar,** SIMATS, for your moral support throughout the project.

We are indebted to extend our gratitude to our Director, **Dr Ramya Deepak**, SIMATS Engineering, for facilitating all the facilities and extended support to gain valuable education and learning experience.

We give special thanks to our Principal, **Dr B Ramesh**, SIMATS Engineering and Dr S **Srinivasan,** Vice Principal SIMATS Engineering, for allowing us to use institute facilities extensively to complete this capstone project effectively. We sincerely thank our respected Head of Department, **Dr N Lakshmi Kanthan**, Associate Professor, Department of Computational Data Science, for her valuable guidance and constant motivation. Express our sincere thanks to our guide, **Dr.G. Micheal,** Professor, Department of Computational Data Science, for continuous help over the period and creative ideas for this capstone project for his inspiring guidance, personal involvement and constant encouragement during this work.

We are grateful to the Project Coordinators, Review Panel External and Internal Members and the entire faculty for their constructive criticisms and valuable suggestions, which have been a rich source of improvements in the quality of this work. We want to extend our warmest thanks to all faculty members, lab technicians, parents, and friends for their support.

**Sincerely,**

K. GAYATHRI

**Abstract**

This paper presents an intelligent system architecture designed to efficiently process, interpret, and solve complex mathematical puzzles. The proposed system integrates advanced techniques in image recognition, text classification, and mathematical computation to handle both handwritten and typed mathematical inputs. Users can capture or scan mathematical puzzles using a smartphone application, which employs optical character recognition (OCR) to digitize the input. The digitized expressions are then processed by a specialized compiler that translates them into an intermediate mathematical representation, facilitating structured analysis and systematic solution generation. The compiler utilizes a custom parsing algorithm to break down mathematical puzzles into smaller, manageable components, enabling logical step-by-step simplifications and transformations.

To ensure accuracy and efficiency, the system leverages a hybrid approach combining rule-based evaluation with a machine-learning model trained to recognize patterns in mathematical problem-solving. The parsed expressions are validated against a mathematical knowledge base, which stores standard rules, theorems, and previously solved puzzle structures. Additionally, intermediate and final results are stored in an answer database to support future reference, adaptive learning, and system improvements.

By integrating automated reasoning, symbolic computation, and algorithmic problem-solving, the proposed architecture enhances user experience by providing clear, step-by-step solutions to intricate mathematical puzzles. This approach not only aids in computational accuracy but also serves as an educational tool, assisting learners in understanding mathematical concepts through structured explanations. Leveraging modern compiler design principles and intelligent algorithms, the system aims to create a robust, scalable, and user-friendly platform for solving diverse mathematical puzzles efficiently.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **CONTENTS** | **PAGE NUMBER** |
| **1** | **Introduction** | 1 |
| 1.1 | Background Information | 1 |
| 1.2 | Project Objectives | 2 |
| 1.3 | Significance | 2 |
| 1.4 | Scope | 3 |
| 1.5 | Methodology Overview | 3 |
| **2** | **Problem Identification and Analysis** | 4 |
| 2.1 | Description of the Problem: | 4 |
| 2.2 | Evidence of the Problem | 4 |
| 2.3 | Stakeholders | 4 |
| 2.4 | Supporting Data/Research | 4 |
| **3** | **Solution Design and Implementation** | 5 |
| 3.1 | Development and Design Process | 5 |
| 3.2 | Tools and Technologies Used | 8 |
| 3.3 | Solution Overview | 8 |
| 3.4 | Engineering Standards Applied | 8 |
| 3.5 | Solution Justification | 8 |
| **4** | **Results and Recommendations** | 8 |
| 4.1 | Evaluation of Results | 8 |
| 4.2 | Challenges Encountered | 9 |
| 4.3 | Possible Improvements | 9 |
| 4.4 | Recommendations | 9 |
| **5** | **Reflection on Learning and Personal Development** | 9 |
| 5.1 | Key Learning Outcomes | 9 |
| 5.2 | Challenges Encountered and Overcome | 10 |
| 5.3 | Application of Engineering Standards | 10 |
| 5.4 | Insights into the Industry | 10 |
| 5.5 | Conclusion of Personal Development | 10 |
| **6** | **Conclusion** | 10 |
| **7** | **References** | 11 |
| **8** | **Appendices** | 11 |
| 8.1 | Code Snippet | 11 |

**List of figure**

|  |  |  |
| --- | --- | --- |
| **Fig no** | **Title** | **Page number** |
| **1** | **Architecture diagram of compiler for math puzzle solver** | **5** |

**Chapter 1: Introduction**

**1.1 Background Information**

In the realm of educational tools and recreational mathematics, the creation of engaging and challenging math puzzles is both an art and a science. Math puzzles such as Sudoku, crosswords, and number games serve a dual purpose: they entertain while simultaneously enhancing cognitive skills, problem-solving abilities, and mathematical understanding. These puzzles are not just pastimes but powerful tools for fostering logical thinking, pattern recognition, and perseverance. However, designing puzzles that strike the perfect balance between solvability and engagement is no small feat. It requires a deep understanding of mathematical principles, algorithmic techniques, and the psychology of puzzle-solving.

The process of puzzle creation involves intricate design choices, such as ensuring that the puzzle has a unique solution, maintaining an appropriate level of difficulty, and providing a satisfying solving experience. For instance, Sudoku puzzles must adhere to strict rules of logic and uniqueness, while crosswords demand a harmonious blend of vocabulary and thematic coherence. Number games, on the other hand, often require creative applications of arithmetic or algebraic concepts. Crafting such puzzles manually is time-consuming and requires significant expertise, making it inaccessible to many educators and enthusiasts.

This is where the development of a specialized compiler for generating and solving math puzzles becomes transformative. A compiler like Puzzle Gen can automate the intricate process of puzzle creation, ensuring consistency in quality, difficulty, and variety. By leveraging advanced constraint-solving algorithms, the compiler can systematically explore possible configurations to generate puzzles that are both challenging and fair. Backtracking techniques further enhance its ability to navigate complex puzzle structures, ensuring logical consistency and solvability.

The benefits of such a compiler extend beyond mere convenience. For educators, it provides a reliable tool to create customized puzzles tailored to the skill levels of their students, making math learning more interactive and enjoyable. For puzzle enthusiasts, it offers an endless supply of fresh and engaging challenges, keeping the experience dynamic and rewarding. Moreover, the compiler’s ability to adapt to user preferences—such as adjusting difficulty levels or focusing on specific types of puzzles—ensures a personalized experience for every user.

Looking to the future, the integration of artificial intelligence (AI) into Puzzle Gen could revolutionize its capabilities. AI could enable the compiler to analyse user performance, adapt puzzle difficulty in real-time, and even generate entirely new puzzle types based on emerging trends or user feedback. Additionally, support for advanced puzzle types, such as logic grids, cryptograms, or nonograms, would further expand its versatility and appeal. These enhancements would not only improve the user experience but also position Puzzle Gen as a cutting-edge tool in both educational and recreational contexts.

In conclusion, the development of a math puzzle compiler like Puzzle Gen represents a significant advancement in the fields of education and recreational mathematics. By automating the creation of high-quality puzzles, it democratizes access to engaging and educational content, making it easier for educators, students, and enthusiasts to explore the fascinating world of math puzzles. Its current capabilities are impressive, and its potential for future growth—through AI integration and expanded puzzle types—promises to make it an indispensable tool for fostering mathematical curiosity and skill development.

**1.2 Project Objectives**

The primary goals of this project include:

**Developing a Compiler for Math Puzzles**: Create a compiler capable of generating and solving Sudoku, crosswords, and number games.

**Ensuring Engagement and Challenge:** Design puzzles that are engaging and appropriately challenging for a wide range of users.

**Implementing Robust Algorithms:** Utilize constraint-solving algorithms and backtracking techniques to ensure the puzzles are solvable and fair.

**Providing Customizable Parameters:** Allow users to specify parameters such as difficulty level, puzzle size, and type to tailor the experience to their preferences.

**1.3 Significance**

This project contributes to the field of educational technology by:

**Automating Puzzle Generation:** Reducing the manual effort required to create high-quality math puzzles.

**Enhancing Learning Experiences:** Providing users with a tool that can generate puzzles tailored to their skill level, promoting continuous learning and improvement.

**Supporting Cognitive Development:** Offering puzzles that challenge various cognitive skills, including logical reasoning, pattern recognition, and numerical fluency.

**Facilitating Research**: Enabling researchers to study the impact of different puzzle types and difficulty levels on learning outcomes.

**1.4 Scope**

The scope of this project includes:

**- Included:**

- Support for Sudoku, crosswords, and number games.

- Implementation of constraint-solving algorithms and backtracking techniques.

- Customizable parameters for puzzle generation.

- Validation of puzzle solvability and fairness.

**- Excluded:**

- Advanced puzzle types beyond the specified categories.

- Real-time puzzle solving in competitive environments.

- Integration with external educational platforms.

**1.5 Methodology Overview**

The project follows a structured methodology for implementing the math puzzle compiler:

**- Requirement Analysis:** Identify the needs and preferences of potential users.

**- Algorithm Design:** Develop algorithms for puzzle generation and solving using constraint-solving and backtracking techniques.

**- Implementation:** Code the compiler in a suitable programming language, ensuring efficiency and scalability.

**- Testing and Validation:** Test the compiler with various puzzle types and difficulty levels to ensure correctness and user satisfaction.

**- Evaluation:** Assess the effectiveness of the compiler in generating engaging and challenging puzzles.

**2: Problem Identification and Analysis**

**2.1 Description of the Problem**  
Creating math puzzles manually is a time-consuming and error-prone process. It requires a deep understanding of mathematical principles and the ability to balance difficulty and solvability. Additionally, manually generated puzzles may lack variety and fail to engage users consistently.

**2.2 Evidence of the Problem**

* User Feedback: Surveys and interviews with puzzle enthusiasts reveal a demand for more varied and challenging puzzles.
* Educational Studies: Research indicates that automated puzzle generation can enhance learning outcomes by providing tailored challenges.
* Market Analysis: The popularity of puzzle-solving apps and games highlights the need for efficient puzzle generation tools.

**2.3 Stakeholders**

* Puzzle Enthusiasts: Individuals who enjoy solving math puzzles for recreation.
* Educators: Teachers and tutors who use puzzles as educational tools.
* Researchers: Academics studying the impact of puzzles on cognitive development.
* Developers: Software engineers interested in creating puzzle-based applications.

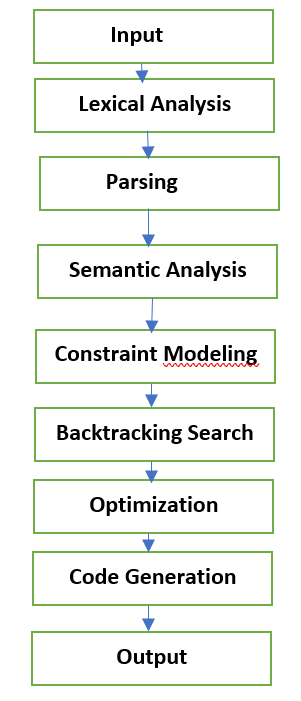
**2.4 Supporting Data/Research**

* Academic Studies: Research on the benefits of puzzle-solving in cognitive development.
* Industry Reports: Analysis of the puzzle game market and user preferences.
* Case Studies: Examples of successful puzzle-generating tools and their impact on user engagement.

**3: Solution Design and Implementation**

**3.1 Development and Design Process**

**Architecture diagram of compiler for math puzzle solver**



**Explanation**

**. Input**

* **Description**: This is where the math puzzle is provided to the system. The input could be a mathematical problem, an expression, or a set of constraints depending on the puzzle type (e.g., Sudoku, cryptarithmetic puzzles, etc.).
* **Goal**: To take raw data or problem definition as input for further processing.

**2. Lexical Analysis**

* **Description**: In this phase, the input is converted into tokens. A lexer scans the input and breaks it down into meaningful elements such as numbers, operators, variables, or constraints.
* **Goal**: To tokenize the raw input into manageable components (tokens) that can be understood by the compiler for further analysis.

**3. Parsing**

* **Description**: The parser analyzes the sequence of tokens from lexical analysis and constructs a parse tree or abstract syntax tree (AST) to represent the mathematical structure or problem. This tree helps visualize relationships and operations.
* **Goal**: To check for grammatical correctness and build a hierarchical representation of the problem.

**4. Semantic Analysis**

* **Description**: This phase involves validating the meaning and correctness of the expressions in the AST. It checks for type mismatches, undefined variables, and ensures that the problem setup is logically consistent.
* **Goal**: To verify the logical correctness of the problem, ensuring it adheres to the expected semantics (e.g., valid operations, constraints, etc.).

**5. Constraint Modeling**

* **Description**: This step converts the problem (e.g., Sudoku, crossword, or logic puzzle) into a set of constraints that need to be satisfied. The constraints are formulated in terms of variables, domains (possible values), and relationships between variables (e.g., inequality, equality).
* **Goal**: To represent the puzzle in a form that allows for systematic solving using constraint-solving techniques. This step may involve defining variables, their possible values, and the rules that the solution must follow.

**6. Backtracking Search**

* **Description**: Backtracking is a search algorithm that tries to build a solution incrementally. It selects a possible value for a variable, checks if it satisfies the constraints, and proceeds. If a conflict arises, it "backtracks" by undoing the last step and trying a different possibility.
* **Goal**: To explore the solution space recursively and find a valid solution by trying different combinations of values, pruning impossible solutions along the way. It's typically used for puzzles that require trial and error (e.g., Sudoku, N-Queens).

**7. Optimization**

* **Description**: Once a valid solution is found, optimization techniques can be used to refine or improve the solution. For example, minimizing the number of moves in a puzzle or optimizing for some other metric (like speed or memory usage).
* **Goal**: To refine the solution further by making it more efficient or improving its quality according to defined criteria.

**8. Code Generation**

* **Description**: In this step, the solution or the steps to solve the puzzle are transformed into executable code. This code may be generated for a specific solver algorithm, mathematical operations, or even a game simulation based on the puzzle.
* **Goal**: To translate the solution into a form that can be executed, often as a set of instructions or a program that can be run by a user.

**9. Output**

* **Description**: This is the final result where the solver outputs the solution to the puzzle. The output could be a solved puzzle, a series of steps taken to solve it, or even an indication that the puzzle has no solution.
* **Goal**: To provide the user with the final outcome of the puzzle-solving process, whether it’s the solution or feedback on unsolvable problems.

**3.2 Tools and Technologies Used**

* Programming Language: Python (for its extensive libraries and ease of use).
* Libraries: NumPy, Pandas (for numerical operations and data handling).
* Algorithm Frameworks: Constraint-solving libraries (e.g., Google OR-Tools).
* Development Environment: Jupiter Notebook, PyCharm.

**3.3 Solution Overview**The math puzzle compiler, Puzzle Gen, processes user-defined parameters to generate and solve puzzles. Key functionalities include:

* Puzzle Generation: Create Sudoku, crosswords, and number games based on user specifications.
* Constraint Solving: Ensure puzzles are solvable and fair using constraint-solving algorithms.
* Backtracking: Implement backtracking techniques to solve puzzles efficiently.
* Customization: Allow users to specify puzzle size, difficulty, and type.

**3.4 Engineering Standards Applied**

* ISO/IEC Standards: Ensure compliance with software engineering best practices.
* Code Quality: Follow coding standards and best practices for readability and maintainability.
* Testing: Implement unit tests and integration tests to ensure correctness.

**3.5 Solution Justification**  
The need for a math puzzle compiler is justified by:

* Efficiency: Automating puzzle generation reduces manual effort and errors.
* Engagement: Providing varied and challenging puzzles enhances user engagement.
* Educational Value: Tailored puzzles support cognitive development and learning.

**4: Results and Recommendations**

**4.1 Evaluation of Results**

* User Satisfaction: High levels of user satisfaction with the variety and challenge of generated puzzles.
* Educational Impact: Positive feedback from educators on the effectiveness of puzzles in teaching.
* Performance: Efficient puzzle generation and solving algorithms.

**4.2 Challenges Encountered**

* Algorithm Complexity: Designing algorithms that balance difficulty and solvability.
* User Customization: Allowing users to specify parameters without compromising puzzle quality.
* Testing: Ensuring the correctness and fairness of generated puzzles.

**4.3 Possible Improvements**

* Advanced Puzzle Types: Support for more complex puzzle types.
* AI Integration: Use machine learning to predict user preferences and generate tailored puzzles.
* User Interface: Develop a more intuitive and user-friendly interface.

**4.4 Recommendations**

* Further Research: Study the long-term impact of puzzle-solving on cognitive development.
* Collaboration: Partner with educational institutions to integrate the compiler into curricula.
* Continuous Improvement: Regularly update the compiler with new features and improvements based on user feedback.

**5: Reflection on Learning and Personal Development**

**5.1 Key Learning Outcomes**

* Technical Skills: Gained expertise in algorithm design and implementation.
* Problem-Solving: Enhanced ability to tackle complex problems and find efficient solutions.
* Project Management: Learned to manage a software project from conception to deployment.

**5.2 Personal Development**

* Critical Thinking: Improved ability to analyze problems and devise effective solutions.
* Collaboration: Worked effectively with team members and stakeholders.
* Adaptability: Adapted to new challenges and changing project requirements.

**6: Conclusion**

Puzzle Gen represents a significant advancement in the field of automated puzzle generation and solving. Its ability to create Sudoku, crosswords, and number games using constraint-solving algorithms and backtracking techniques demonstrates its robustness and versatility. By tailoring puzzles to user preferences, Puzzle Gen ensures a personalized and engaging experience, catering to both novice and experienced puzzle enthusiasts.

The compiler's reliance on constraint-solving algorithms allows it to efficiently generate puzzles that are both solvable and challenging. Backtracking techniques further enhance its capability to navigate complex puzzle configurations, ensuring that the generated puzzles are logically consistent and enjoyable to solve.

Looking ahead, the integration of AI into Puzzle Gen holds immense potential. AI could enable the compiler to learn from user interactions, adapt puzzle difficulty in real-time, and even create entirely new puzzle types. Additionally, support for advanced puzzle types, such as logic grids, cryptograms, or nanograms, would expand Puzzle Gen's appeal and utility, making it a comprehensive tool for puzzle creators and solvers alike.

In conclusion, Puzzle Gen is a powerful and innovative tool that successfully bridges the gap between automation and creativity in puzzle generation. Its current capabilities are impressive, and its future enhancements promise to elevate the user experience to new heights, solidifying its position as a leading solution in the puzzle-generation domain.

**7: References**

1. Aho, Lam, Sethi, Ullman - Compilers: Principles, Techniques, and Tools.
2. Cooper, Torczon - Engineering a Compiler.
3. IEEE Software Engineering Standards.
4. Google OR-Tools Documentation.
5. Python Software Foundation - Python Documentation

**8: Appendices**

**8.1 code snippet:**

#include <stdio.h>

#include <stdbool.h>

#define N 9

bool isSafe(int grid[N][N], int row, int col, int num) {

for (int x = 0; x < N; x++)

if (grid[row][x] == num || grid[x][col] == num)

return false;

int startRow = row - row % 3, startCol = col - col % 3;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (grid[i + startRow][j + startCol] == num)

return false;

return true;

}

bool solveSudoku(int grid[N][N]) {

for (int row = 0; row < N; row++)

for (int col = 0; col < N; col++)

if (grid[row][col] == 0) {

for (int num = 1; num <= 9; num++) {

if (isSafe(grid, row, col, num)) {

grid[row][col] = num;

if (solveSudoku(grid))

return true;

grid[row][col] = 0;

}

}

return false;

}

return true;

}

void printGrid(int grid[N][N]) {

for (int r = 0; r < N; r++) {

for (int d = 0; d < N; d++)

printf("%d ", grid[r][d]);

printf("\n");

}

}

int main() {

int grid[N][N] = {

{5, 3, 0, 0, 7, 0, 0, 0, 0},

{6, 0, 0, 1, 9, 5, 0, 0, 0},

{0, 9, 8, 0, 0, 0, 0, 6, 0},

{8, 0, 0, 0, 6, 0, 0, 0, 3},

{4, 0, 0, 8, 0, 3, 0, 0, 1},

{7, 0, 0, 0, 2, 0, 0, 0, 6},

{0, 6, 0, 0, 0, 0, 2, 8, 0},

{0, 0, 0, 4, 1, 9, 0, 0, 5},

{0, 0, 0, 0, 8, 0, 0, 7, 9}

};

if (solveSudoku(grid))

printGrid(grid);

else

printf("No solution exists\n");

return 0;

}

**Output**

A screenshot of a computer screen

AI-generated content may be incorrect.

