



## **HIGH LEVEL DESIGN DOCUMENT**

### **Automated Tool for C Source Code Optimization**

**UE18CS390A – Capstone Project Phase – 1**

*Submitted by:*

<b>Khushei Meghana Meda</b>	<b>PES1201800416</b>
<b>Sriram Subramanian</b>	<b>PES1201800655</b>
<b>Adithya Bennur</b>	<b>PES1201801891</b>
<b>Shashank Vijay</b>	<b>PES1201801828</b>

Under the guidance of

**Prof. N S Kumar**  
Visiting Professor  
PES University

**January - May 2021**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**FACULTY OF ENGINEERING**  
**PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

**TABLE OF CONTENTS**

1. Introduction	4
2. Design Considerations	4
3.1 Design Goals	4
3.2 Architecture Choices	4
3.3 Constraints, Assumptions and Dependencies	5
3. High Level System Design	6
4. Design Description	7
4.1 Class Diagram	7
5.2 Reusability Considerations	8
5. State Diagram	8
6. User Interface Diagrams	9
7. External Interfaces	14
8. Packaging Diagram	15
9. Help	15
10. Design Details	16
10.1 Novelty	16
10.2 Innovativeness	16
10.3 Interoperability	16
10.4 Performance	16
10.5 Reliability	16
10.6 Maintainability	16
10.7 Portability	16
10.8 Reusability	16



## **HIGH LEVEL DESIGN DOCUMENT**

Appendix A: Definitions, Acronyms and Abbreviations	16
Appendix B: References	16
Appendix C: Record of Change History	17
Appendix D: Traceability Matrix	17

## **1. Introduction**

The purpose of this document is to provide an overview of the tool we are developing for the automated optimization of C source code. With this tool, we aim to convert C source code into an optimized version by applying some common techniques, as well as implementing Jon Bentley's rules to optimize code. The high level design involves parsing the source code to perform optimizations and generate the output C code from the intermediate structures.

## **2. Design Considerations**

### **2.1. Design Goals**

- The existing tools for profiling code perform static and dynamic analyses but do not refactor the code to a better version. The latter we hope to achieve along with the former.
- Compilers perform some optimizations, especially using some intermediate representations of the source code, but do not display the high level optimized code to the user.
- Our tool will generate as the output, optimized C source code enabling a clear understanding of the optimizations applied and maybe a better insight into writing programs that increase performance.
- Our tool will also try to implement some optimizations that are not implemented by popular C compilers. Some of these optimizations are suggested by Jon Bentley.
- Our tool will also highlight performance metrics that will contrast the input source code and the generated output source code. The metrics we are considering are execution time and amount of memory used.

### **2.2. Architecture Choices**

The program execution environment during the development and testing of the tool has been the Python-lex-yacc (PLY) module, python version 3.6+ and Unix/Linux based systems for supporting tools such as clang-tidy, indent and kcache/grind. However, the requirements can easily be installed on other popular OS such as Windows and OSX.

Other than a minimal computer system setup (cpu, disk and memory) along with peripherals (mouse, monitor and keyboard), there are no explicit hardware requirements/constraints on the server.

The client would require internet access as well as a modern browser to access the online interface of the tool.

Pros-

- We are implementing the tool with a web based GUI with all the computational work at the server side, so the end user will not have to install or setup any software.
- All the architecture requirements on the server are quite easily achieved on modern systems.
- The use of open source software does not impose any financial constraints for both the client as well as the server.

Cons-

- The profiling metrics such as execution time will be dependent on the server side execution environment and there is no guarantee that the generated output code will perform equivalently on the client's system.
- Since a client server architecture is being followed , scalability problems may arise. However this could potentially be solved using cloud services with ease.

### **2.3. Constraints, Assumptions and Dependencies**

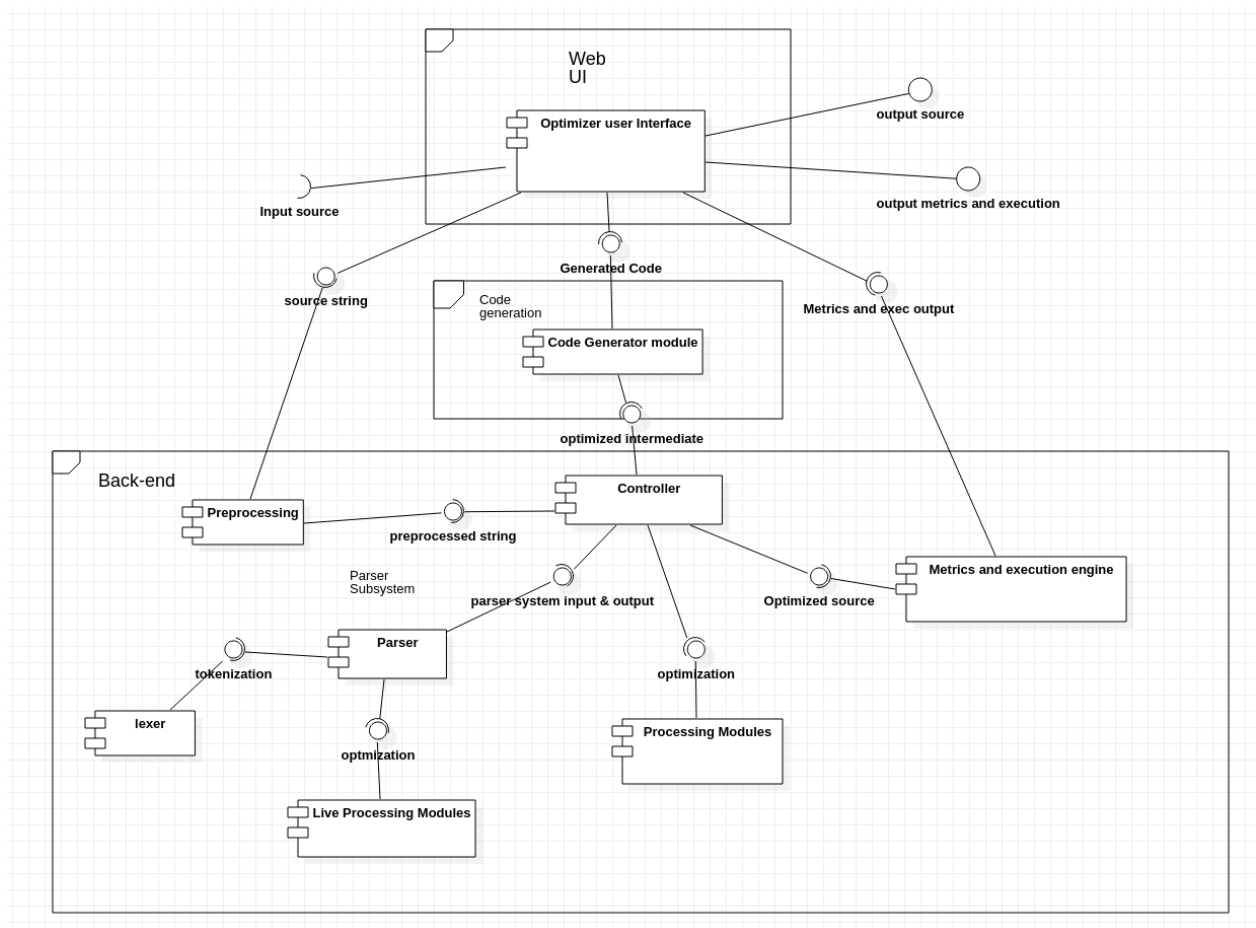
- The GUI must be accessible irrespective of browser used by the client.
- The GUI must provide interfaces to select required optimizations , upload input files, download output files as well as a way to view the outputs and metrics of performance.
- The tool must output generated programs in second scale latencies and it must not increase the load on compilers.
- The user is required to input syntactically and semantically correct programs for the tool to output a logically equivalent and optimized version of the code. Otherwise the output program may not compile or the underlying logic may be undefined.
- Some of the optimizations being implemented are suggested programming practices that are not implemented by popular C compilers. Fully automating such optimizations is not possible with a rule-based approach.
- The user is required to have access to a stable internet connection to access the online graphical user interface of the tool.

## HIGH LEVEL DESIGN DOCUMENT

- There could be issues in scalability after deployment. But could possibly be solved by migrating to a cloud service. The system is highly portable.

### 3. High Level System Design

#### Component Diagram

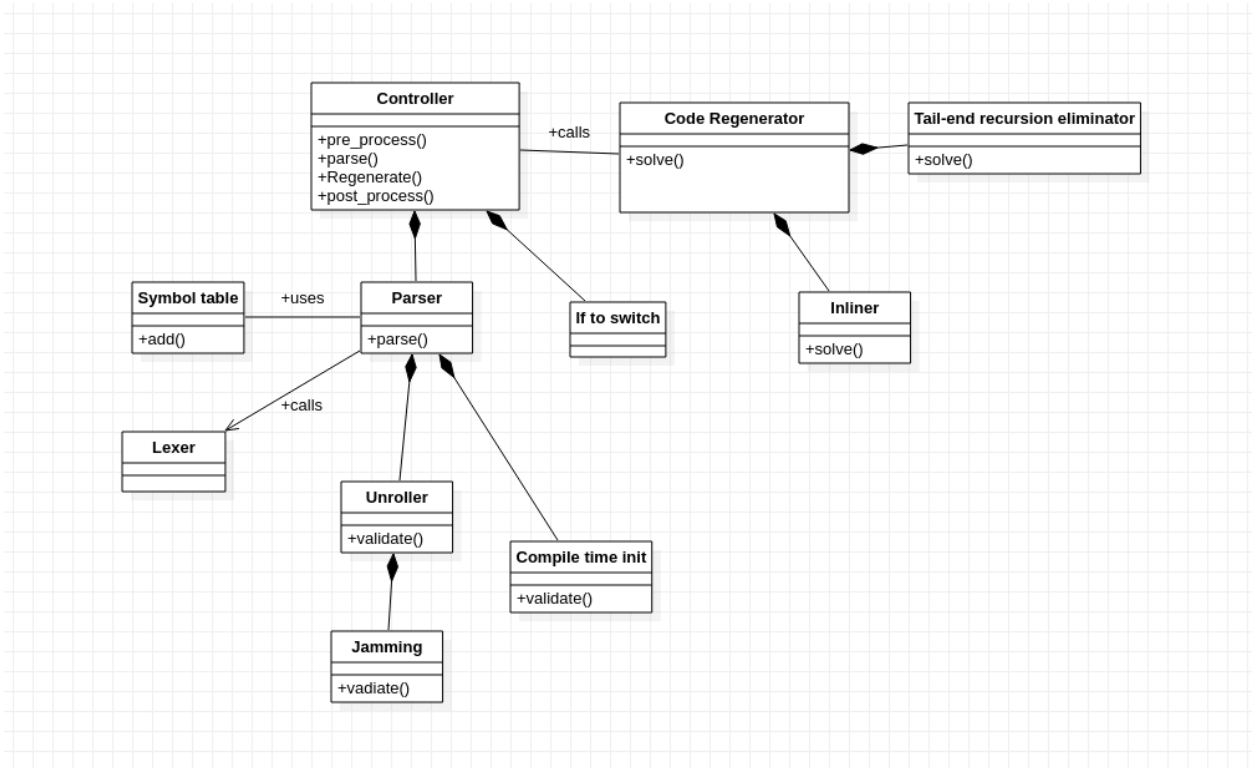


### 4. Design Description

interfaces listed:

1. parse
2. pre\_process
3. tail\_rec\_eli\_solve
4. fn\_inline\_solve
5. lookahead
6. compile\_init\_validate
7. for\_unroll\_validate
8. identify\_chains
9. make\_switch
10. make\_compile\_inits
11. post\_process
12. **Code Generator**

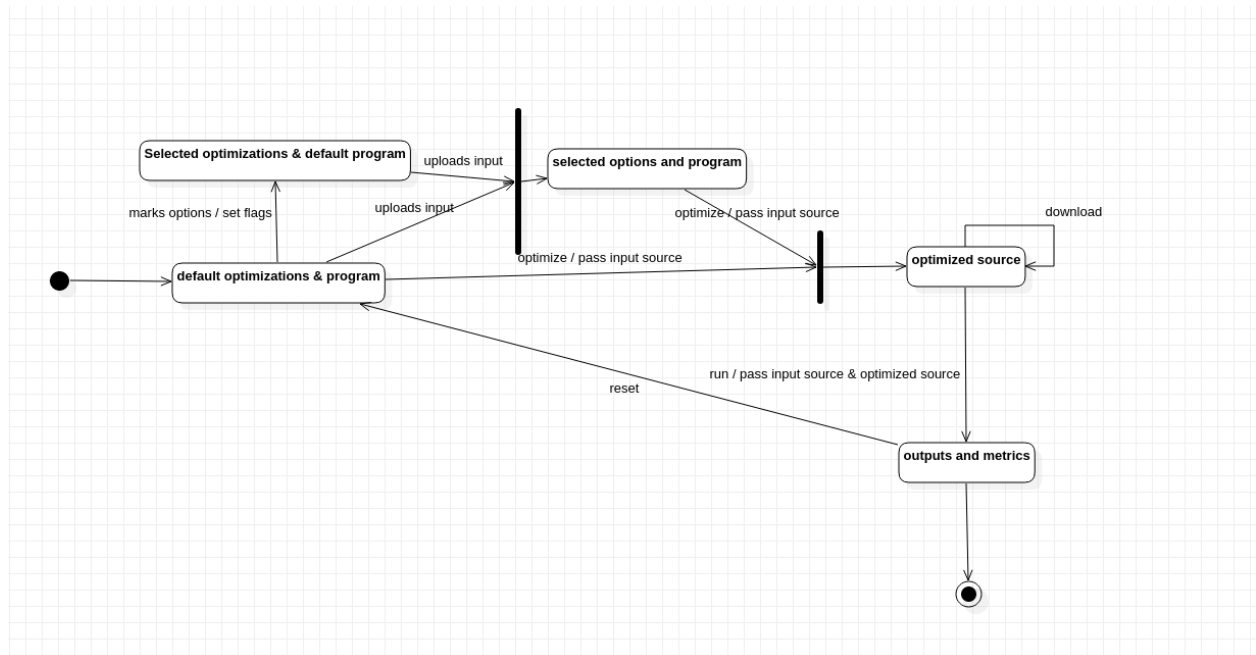
#### 4.1. Class Diagram



### 4.2. Reusability Considerations

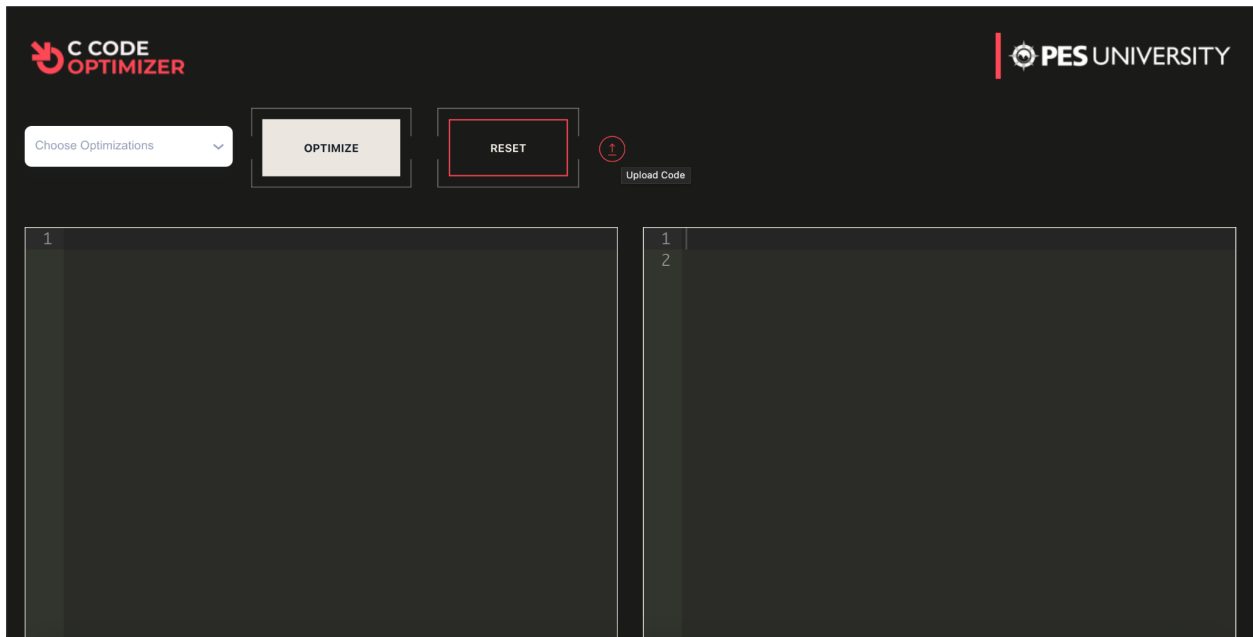
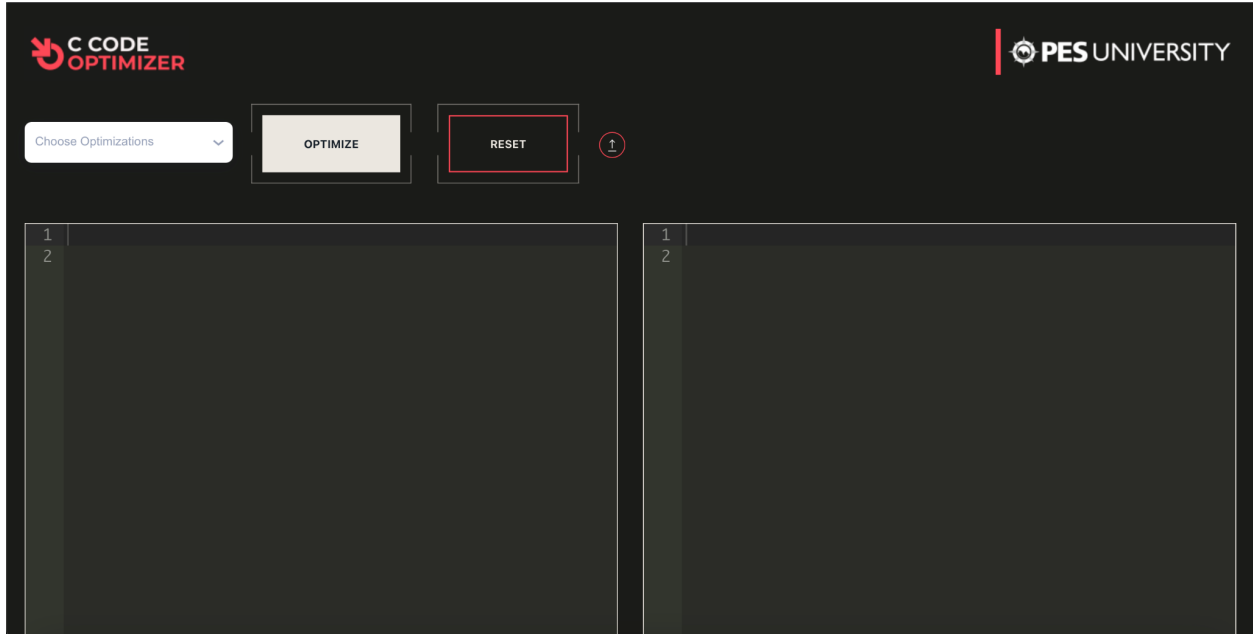
- Indent tool is used for indentation of output.
- The Clang-Tidy compiler is used to check correctness of programs written as well as preprocess the input program.
- Symbol Table Module is depended on by almost all optimization modules.
- Preprocessing and postprocessing modules can be used in general to clean code.

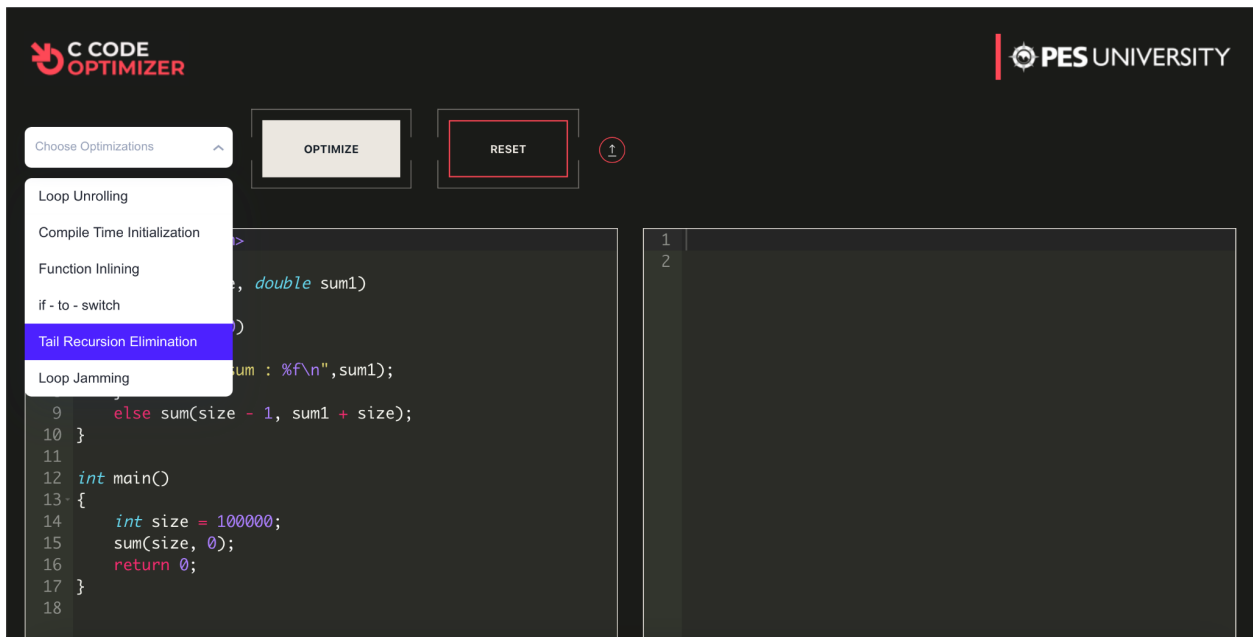
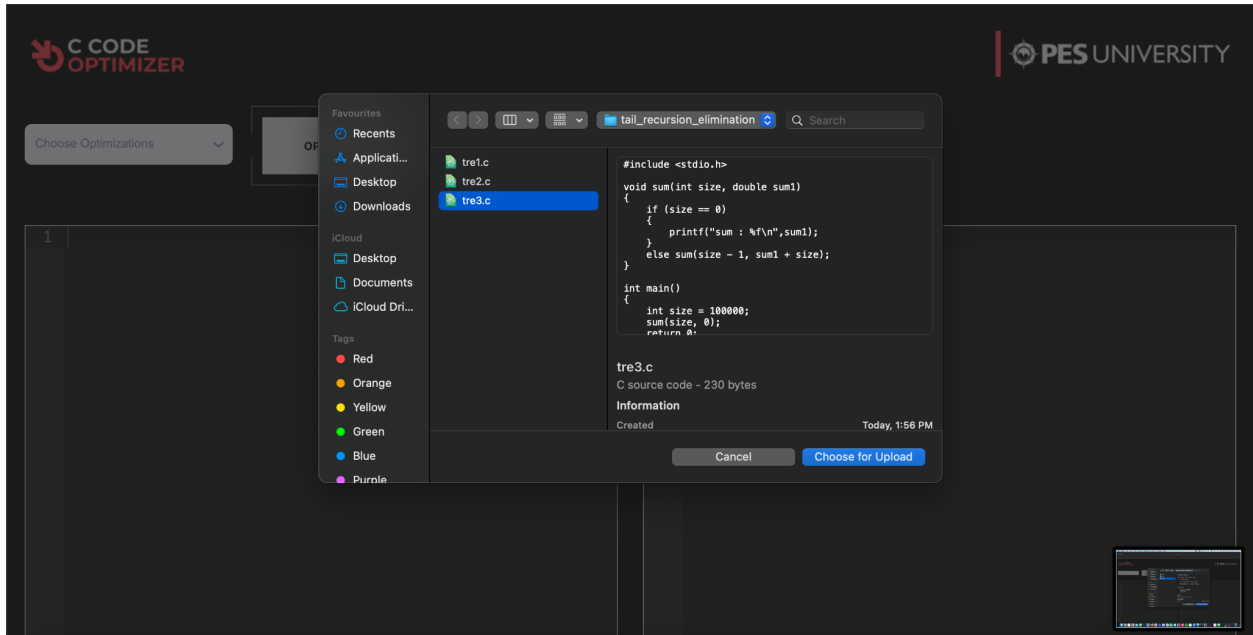
### 5. State Diagram









### 6. User Interface Diagrams






Tail Recursion Elimination   
Function Inlining 

OPTIMIZE

RESET



```



1 #include <stdio.h>
2
3 void sum(int size, double sum1)
4 {
5     if (size == 0)
6     {
7         printf("sum : %f\n",sum1);
8     }
9     else sum(size - 1, sum1 + size);
10 }
11
12 int main()
13 {
14     int size = 100000;
15     sum(size, 0);
16     return 0;
17 }
18



```

```

1
2


```





Tail Recursion Elimination   
Function Inlining 

OPTIMIZE

RESET





Download Code

```

1 #include <stdio.h>
2
3 void sum(int size, double sum1)
4 {
5     if (size == 0)
6     {
7         printf("sum : %f\n",sum1);
8     }
9     else sum(size - 1, sum1 + size);
10 }
11
12 int main()
13 {
14     int size = 100000;
15     sum(size, 0);
16     return 0;
17 }
18

```

```

1 // optimized code
2
3 #include<stdio.h>
4 void sum(int size, double sum1)
5 {
6     label_1b728cc193614adf9e36b62180df9ee8: {
7     }
8     if (size == 0) { {
9         printf("sum : %f\n", sum1);
10    }
11    } else { { { // tail recursion eliminated
12        int par_size_1b728cc193614adf9e36b62180df9
13        size;
14        double par_sum1_1b728cc193614adf9e36b62180
15        = sum1;
16        size =
17        par_size_1b728cc193614adf9e36b62180df9
18        1;
19        sum1 =

```

```

5  if (size == 0)
6  {
7      printf("sum : %f\n",sum1);
8  }
9  else sum(size - 1, sum1 + size);
10 }
11
12 int main()
13 {
14     int size = 100000;
15     sum(size, 0);
16     return 0;
17 }
18

```

```

5- {
6- label_1b728cc193614adf9e36b62180df9ee8:{
7- }
8- if (size == 0) { {
9-     printf("sum : %f\n", sum1);
10- }
11- } else { { { // tail recursion eliminated
12-     int par_size_1b728cc193614adf9e36b62180df9
13-     size;
14-     double par_sum1_1b728cc193614adf9e36b62180
15-     = sum1;
16-     size =
17-     par_size_1b728cc193614adf9e36b62180df9
18-     1;
19-     sum1 =
20-     par_sum1_1b728cc193614adf9e36b62180df9
21-     par_size_1b728cc193614adf9e36b62180df9
22-     1;
23- }
24- }
25- }
26- }
27- }
28- }
29- }
30- }
31- }
32- }
33- }
34- }
35- }
36- }
37- }
38- }
39- }
40- }
41- }
42- }
43- }
44- }
45- }
46- }
47- }
48- }
49- }
50- }
51- }
52- }
53- }
54- }
55- }
56- }
57- }
58- }
59- }
60- }
61- }
62- }
63- }
64- }
65- }
66- }
67- }
68- }
69- }
70- }
71- }
72- }
73- }
74- }
75- }
76- }
77- }
78- }
79- }
80- }
81- }
82- }
83- }
84- }
85- }
86- }
87- }
88- }
89- }
90- }
91- }
92- }
93- }
94- }
95- }
96- }
97- }
98- }
99- }
100- }

```

RUN

### OUTPUTS & METRICS

ubuntu@ubuntu: ~CCodeOptimizer (Non-Optimized Output)

File Edit View Search

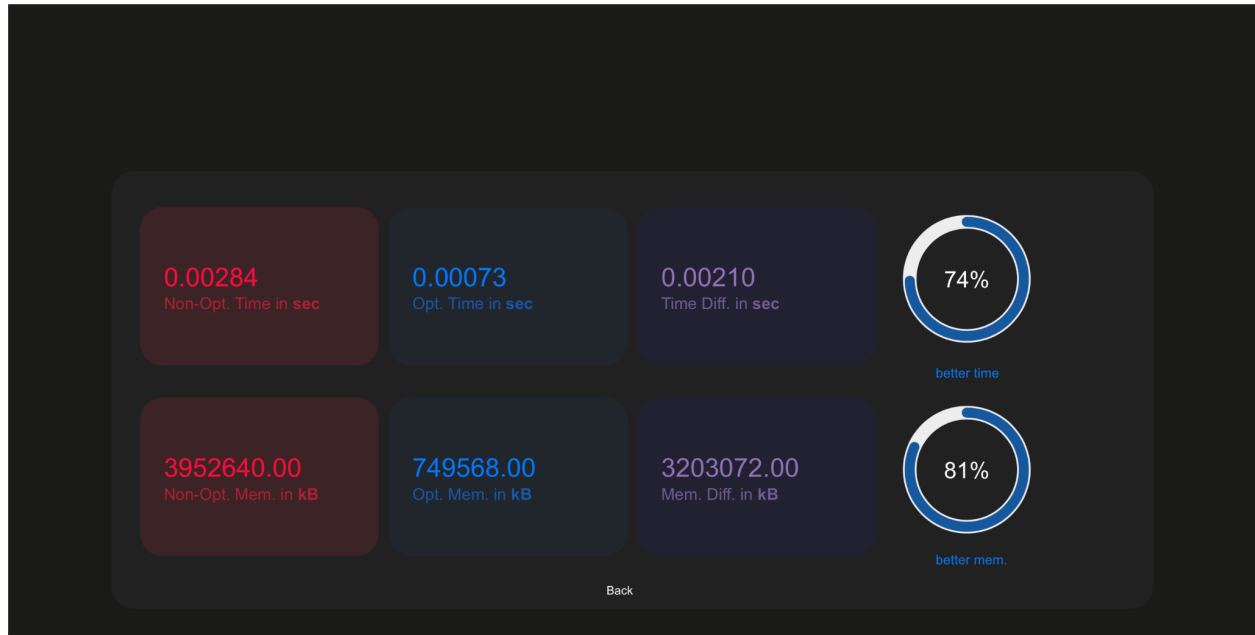
sum : 5000050000.000000

ubuntu@ubuntu: ~CCodeOptimizer (Optimized Output)

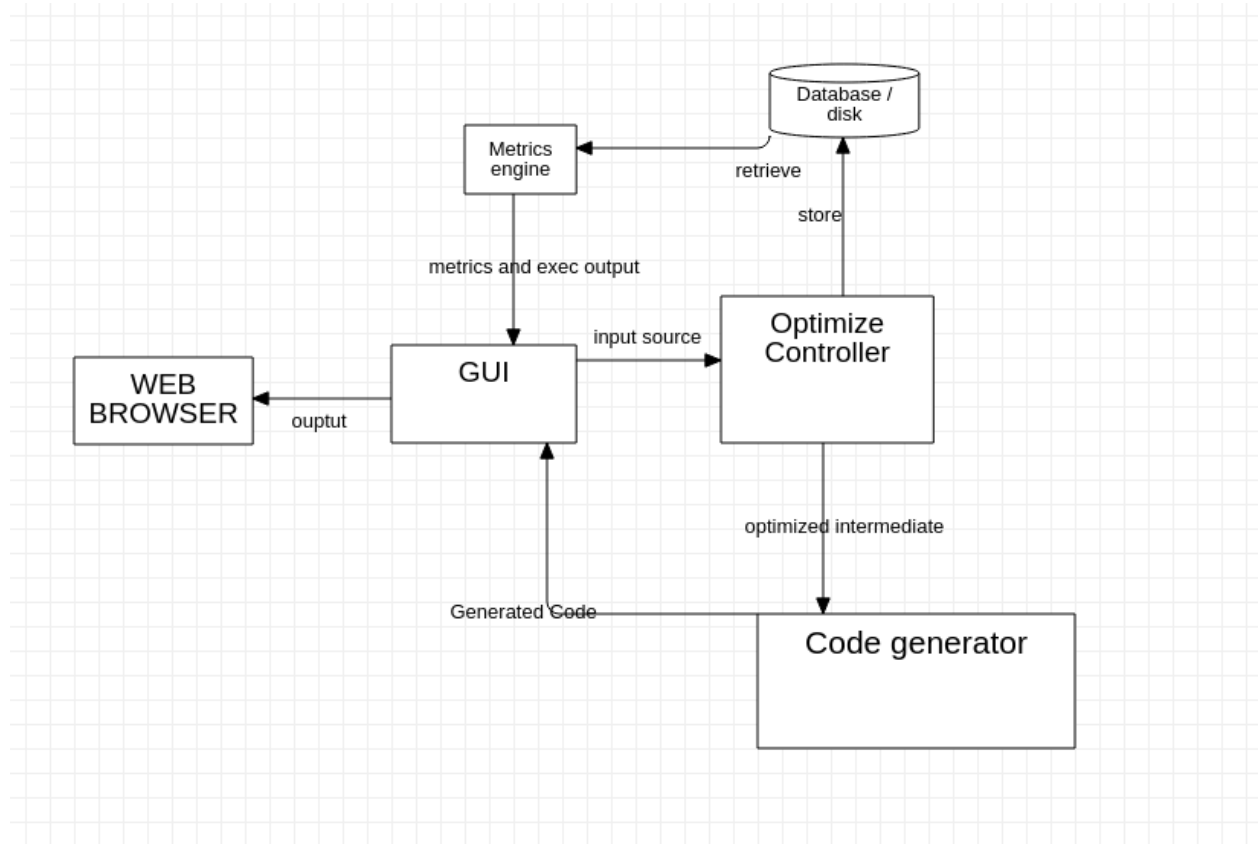
File Edit View Search

sum : 5000050000.000000

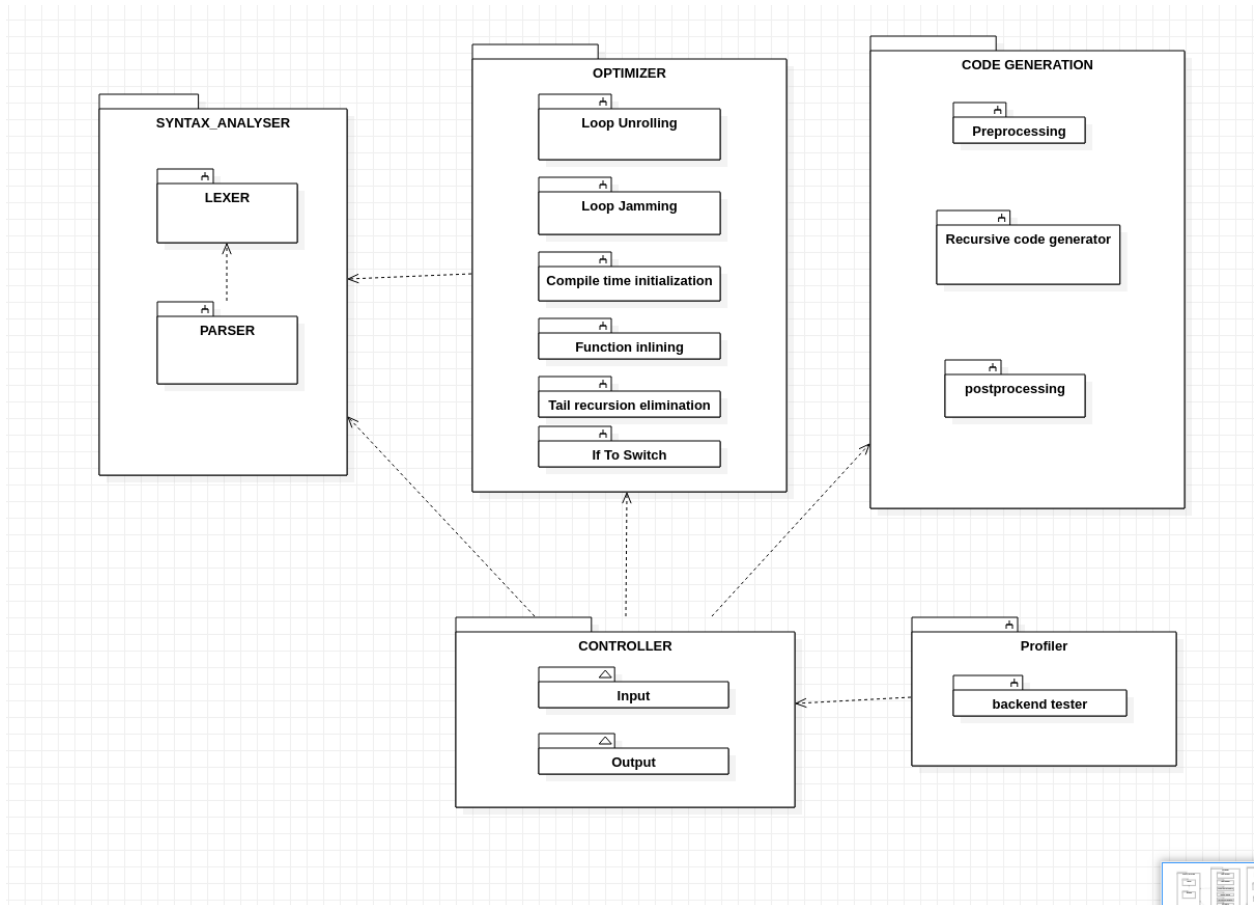
[View Metrics](#)



## 7. External Interfaces



### 8. Packaging Diagram



### 9. Help

1. A page in the UI will be reserved for providing information about the interface itself and on how to use it.
2. A catalogue of all possible actions that the user can perform will be provided.
3. A catalogue of all the optimization options that a user can select will be provided.

## **10. Design Details**

**10.1 Novelty** - The tool we are developing is completely new and there is no existing tool with the same set of features.

**10.2 Innovativeness** - Completely innovative in terms of implementation, given that there is no existing open source codebase for these optimizations.

**10.3 Interoperability** - The UI can be accessed by the user through any system as long as it has a modern web browser.

**10.4 Performance** - The tool is expected to deliver results in second scale latencies for the average input program size.

**10.5 Reliability** - The tool will not change the underlying logic of the input program provided that the user inputs syntactically and semantically correct programs.

**10.6 Maintainability** - The inherent code base is easy to maintain as it is modular and an object based model.

**10.7 Portability** - Clients can access UI from any system of choice in the presence of a modern browser. The server functions best in a unix/linux based environment due to component dependencies. However, substitutes for the components are available and the server will function in other systems.

**10.8 Reusability** - Code base consists of a few modules that can prove to be useful in other applications.

## **Appendix A: Definitions, Acronyms and Abbreviations**

### **Abbreviations:**

OS - Operating Systems

UI - User Interface

### **Definitions:**

Bentley's rules - refers to the set of rules formulated by John Bentley which are known to improve performance of programs when applied.

## **Appendix B: References**

<https://www.geeksforgeeks.org/basic-code-optimizations-in-c/>

[https://www.keil.com/support/man/docs/armclang\\_intro/armclang\\_intro\\_fnb1472741490155.htm](https://www.keil.com/support/man/docs/armclang_intro/armclang_intro_fnb1472741490155.htm)

[https://medium.com/@riddhipandya\\_82734/code-optimization-techniques-266c2199e62b](https://medium.com/@riddhipandya_82734/code-optimization-techniques-266c2199e62b)



<https://alibabatech.medium.com/gcc-vs-clang-llvm-an-in-depth-comparison-of-c-c-compilers-899ede2be378>

<https://easyaspi314.github.io/gcc-vs-clang.html>

[https://en.wikipedia.org/wiki/High-level\\_design](https://en.wikipedia.org/wiki/High-level_design)

<https://tallyfy.com/uml-diagram/>

### Appendix C: Record of Change History

#	Date	Document Version No.	Change Description	Reason for Change
1.	07 - 04-21	2	Changed component diagram	addition of a new component.
2.	10-04 -21	3	Appended to interface listing	New interface

### Appendix D: Traceability Matrix

Project Requirement Specification Reference Section No. and Name.	DESIGN / HLD Reference Section No. and Name.
3. Functional Requirements	3. High level system design - component diagram
5. Non-functional requirements	11. Design details