UE18CS390A – Capstone Project Phase – 1

Project Progress Review #3

Project Title : Automated Tool for Source Code Optimization
Project ID : PW22NSK01
Project Guide : Prof. N S Kumar
Project Team : Khushei Meghana Meda, Sriram Subramanian, Shashank Vijay, Adithya Bennur

Abstract

- We propose to develop an automated tool for C source code optimization.
- We propose to analyze the structure of the source code.
- We propose to detect the possibility of optimization.
- We propose to implement Bentley’s rules along with some common optimization techniques.

Scope

- Cannot handle programs with bugs.
- Cannot achieve significant performance improvement in low run time programs.
- Cannot guarantee implementation of all Jon Bentley’s rules.

Suggestions from Review - 2

Provide the suggestions and remarks given by the panel members.
Suggestions-

- Implement a web based GUI for the tool
- Compare the performances of input code and generated optimized code.

Mention the feasibility on the same showing the progress.

Feasibility and progress-

- The performance metric being measured is execution time.
- Other analysis and profiling metrics are yet to be implemented.

Design Approach

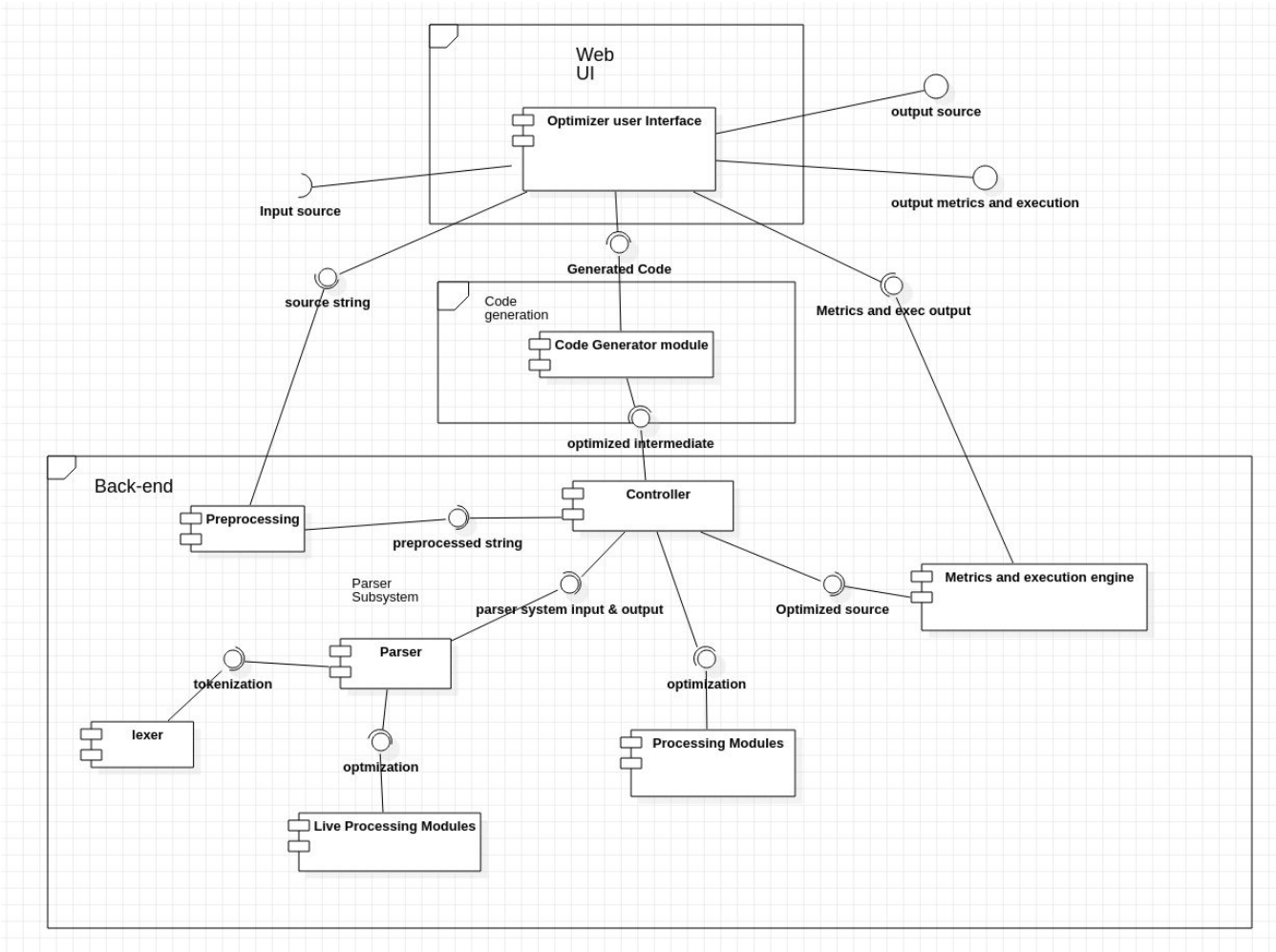
Our approach is incremental and iterative.

This approach has been chosen to maintain a balance between the time spent in feasibility study and implementation.

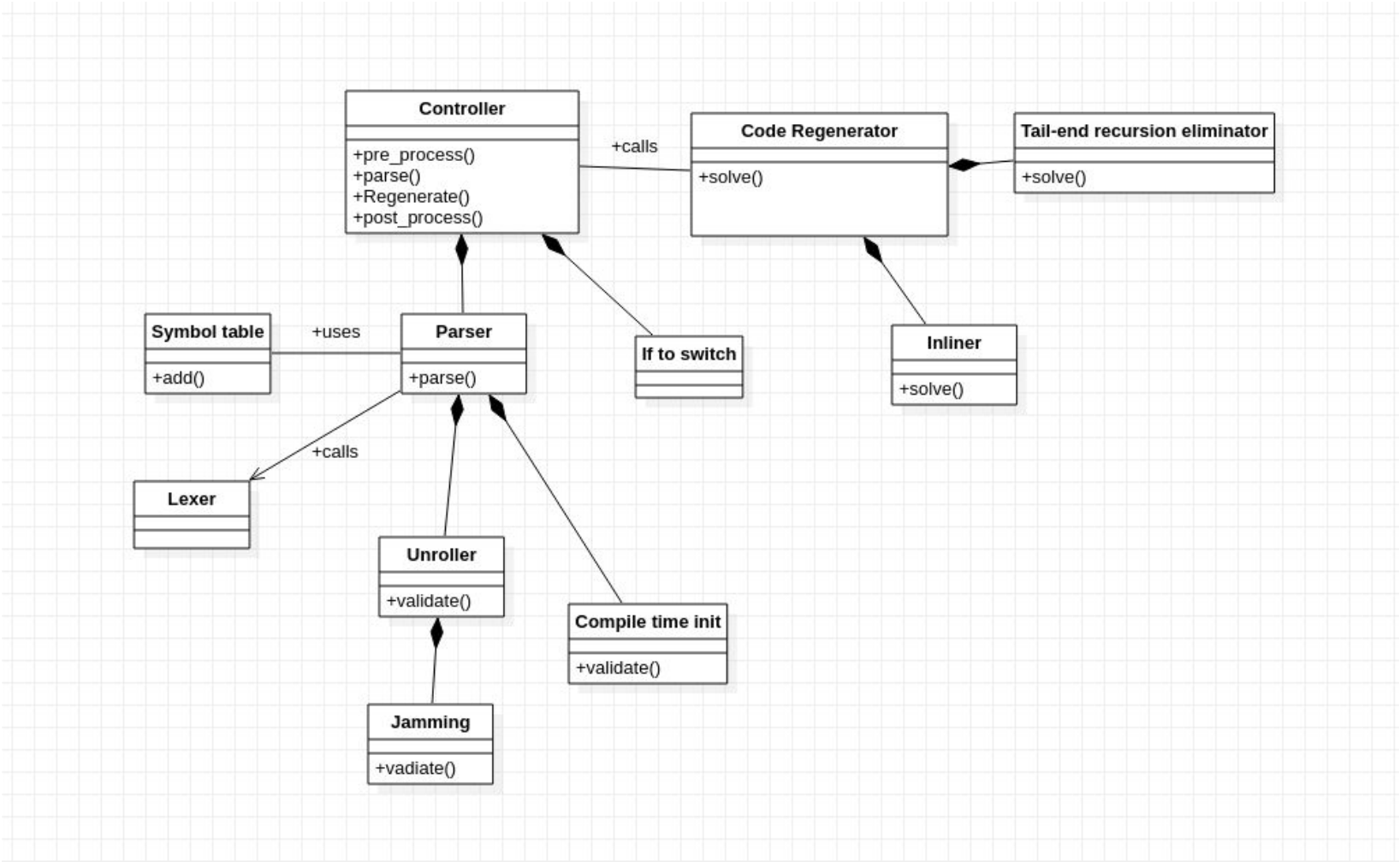
Benefits: Parallelism in implementation of various components.

Drawbacks: Requirement of dedicated time for integration of various components.

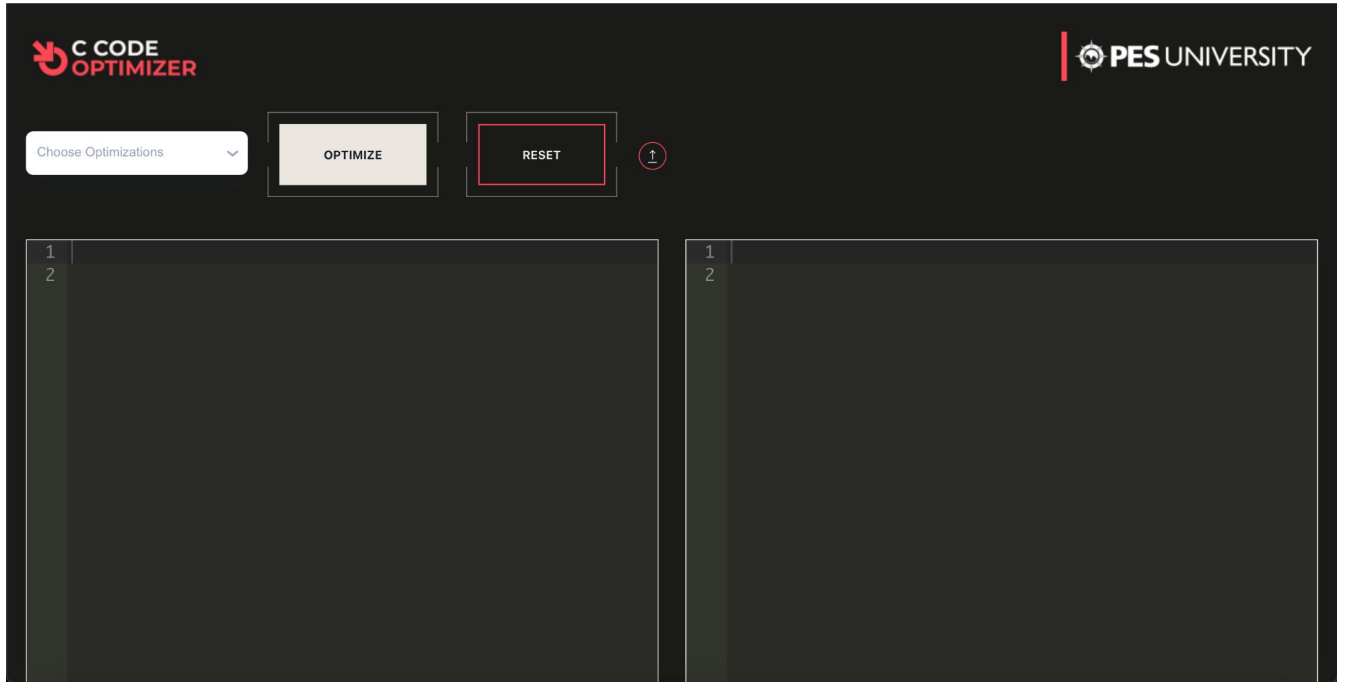
Architecture

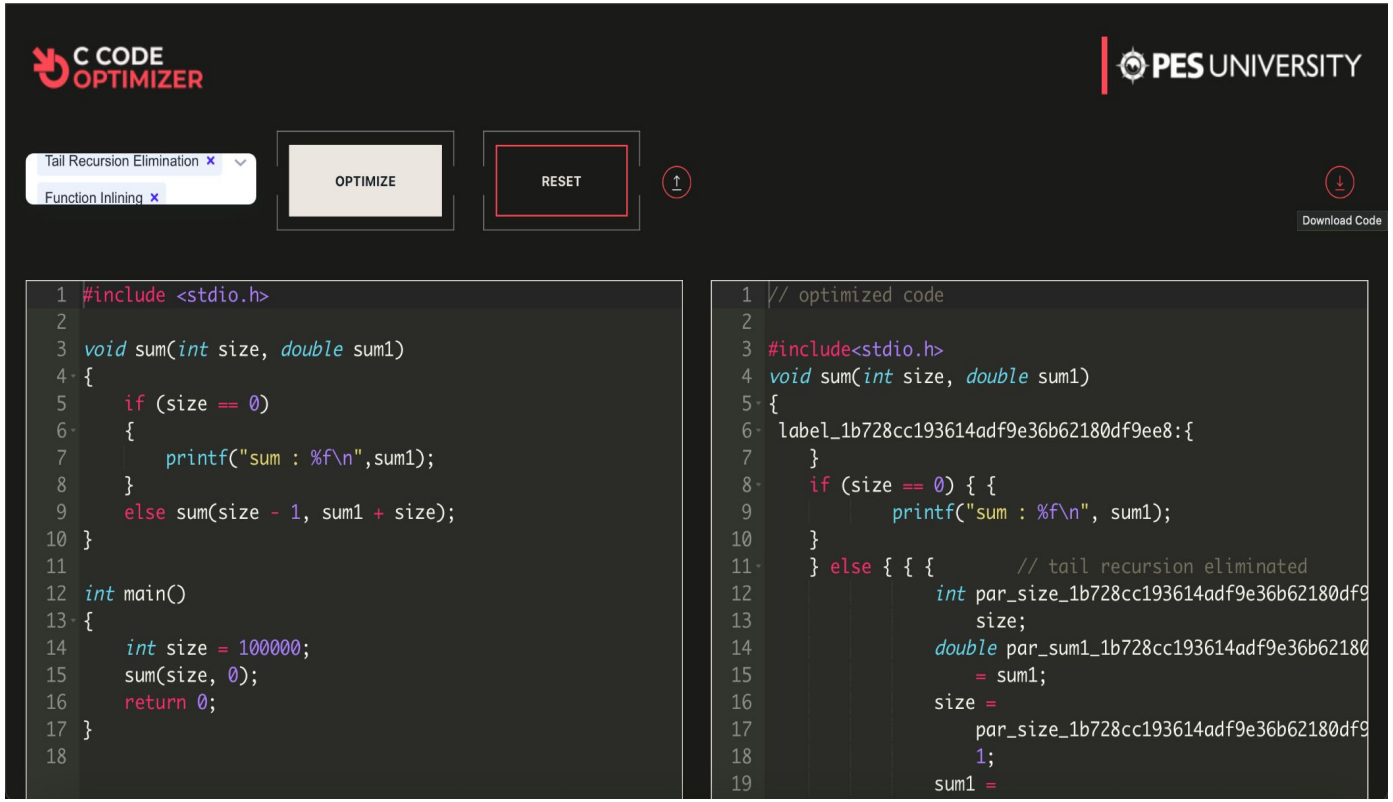


Master Class Diagram

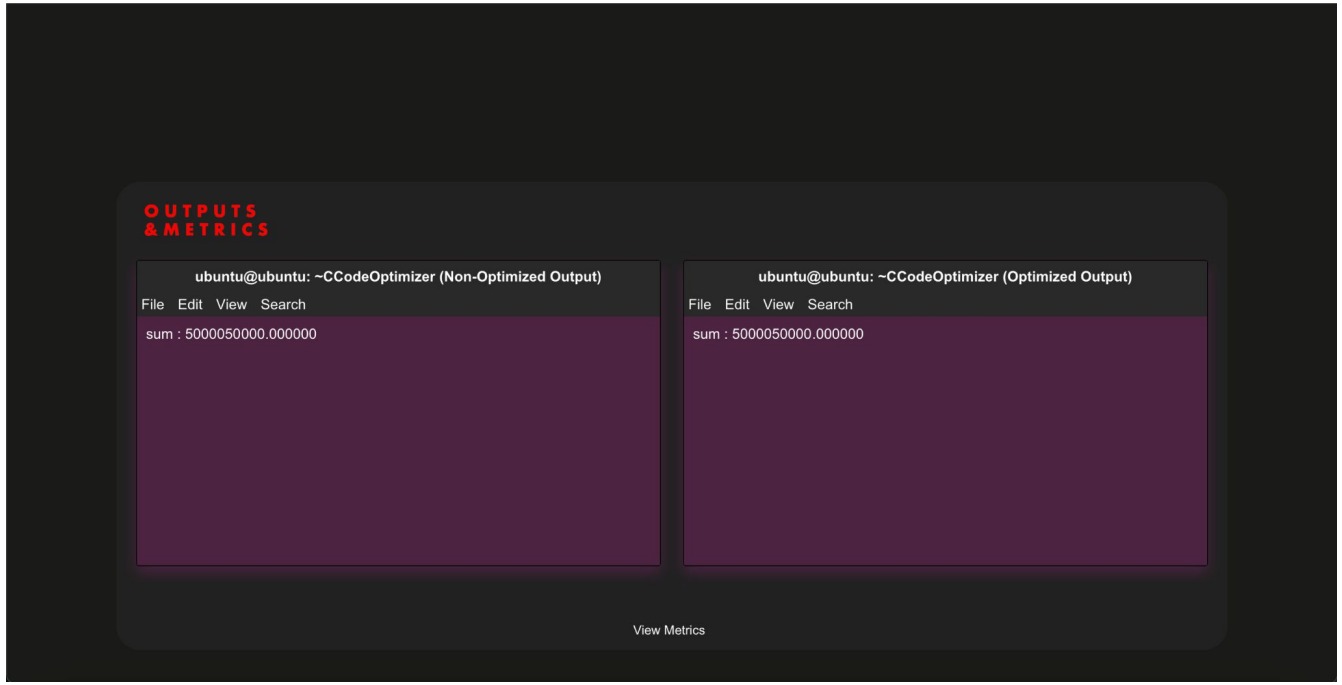


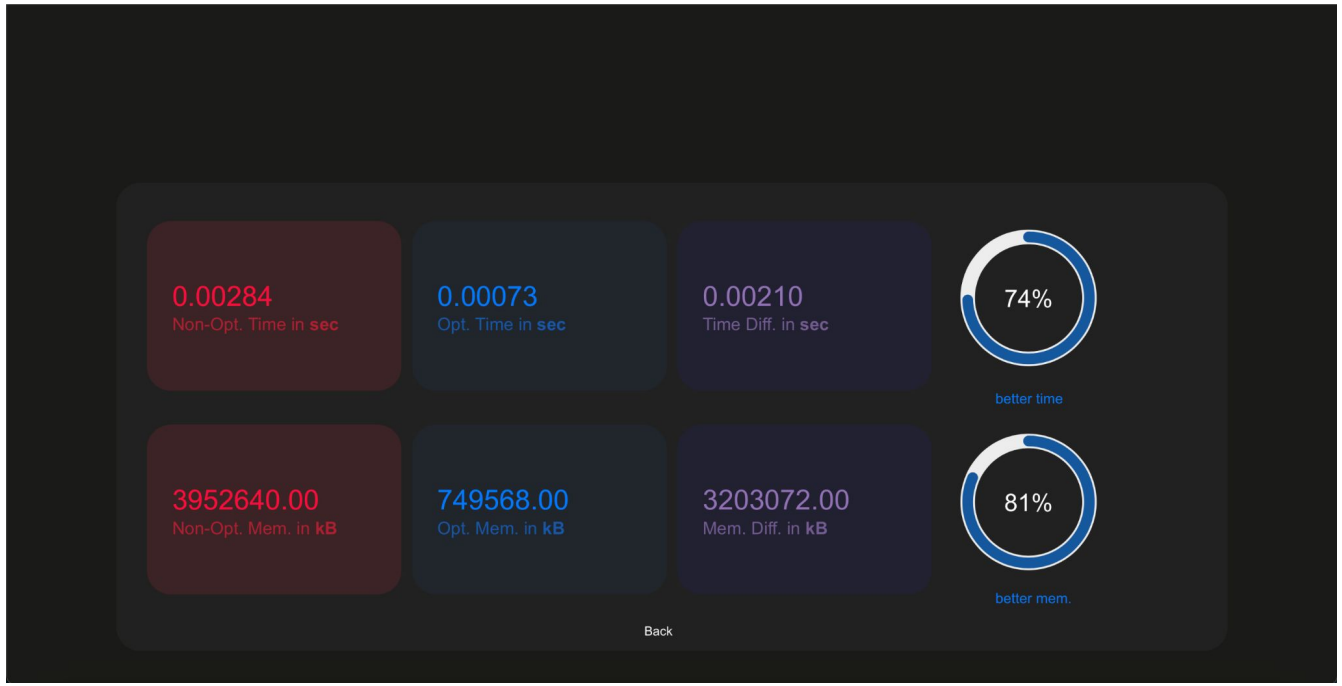
UI


C CODE OPTIMIZER
PES UNIVERSITY
Choose Optimizations
OPTIMIZE
RESET
1
2
1
2


C CODE OPTIMIZER
PES UNIVERSITY
Tail Recursion Elimination
Function Inlining
OPTIMIZE
RESET
Download Code
#include <stdio.h>
void sum(int size, double sum1)
{
if (size == 0)
{
printf("sum : %f\n",sum1);
}
else sum(size - 1, sum1 + size);
}
int main()
{
int size = 100000;
sum(size, 0);
return 0;
}
// optimized code
#include<stdio.h>
void sum(int size, double sum1)
{
label_1b728cc193614adf9e36b62180df9ee8:{
}
if (size == 0) { {
printf("sum : %f\n", sum1);
}
} else { { {
// tail recursion eliminated
int par_size_1b728cc193614adf9e36b62180df9
size;
double par_sum1_1b728cc193614adf9e36b62180
= sum1;
size =
par_size_1b728cc193614adf9e36b62180df9
1;
sum1 =

PES UNIVERSITY




0.00284
Non-Opt. Time in sec
0.00073
Opt. Time in sec
0.00210
Time Diff. in sec
74%
better time
3952640.00
Non-Opt. Mem. in kB
749568.00
Opt. Mem. in kB
3203072.00
Mem. Diff. in kB
81%
better mem.
Back

PES UNIVERSITY

Technologies Used

- ply3.11
- clang-tidy LLVM10.0.0
- gcc
- python3.6 or higher
- indent
- valgrind and kcachegrind
- gcc compiler
- Javascript
- PHP

Source Code Generation

```
solve (start_index, end_index, PARSE_TREE, output_program) :
    if start_index=end_index // base case 1
        stop recursion
    if type (PARSE_TREE[start_index]) = 'string'  // base case 2
        if (PARSE_TREE[start_index] is a keyword)
            output_program.append (PARSE_TREE[start_index] + space_char)
        else
            output_program.append (PARSE_TREE[start_index])
    if type (PARSE_TREE[start_index]) = 'int'   // base case 3
        output_program.append (string(PARSE_TREE[start_index])
    if type (PARSE_TREE[start_index]) = 'list')
        solve (0, length(PARSE_TREE[start_index]), PARSE_TREE, output_program)
    solve (start_index+1, end_index, PARSE_TREE, output_program)  // continuing the recursion
```

Project Progress

- We have been progressing at a fairly consistent pace
- We have implemented the following optimizations-
  - Loop unrolling
  - Function inlining
  - If-else if-else to switch
  - Compile-time initialization
  - Constant folding and constant propagation (in progress)
  - Tail recursion elimination (in progress)
  - Loop jamming (in progress)
- We have developed a basic web based GUI for the user.
- We have completed 50% of the project.

Thank
You