CS3423

Michelin: Report

Team-2

Table of Contents

Team Details

- Name: Sai Sidhardha Grandhi
  Roll number: CS19BTECH11050
  Roll: Project Manager
  GitHub ID: G-Sidhardha

- Name: Krishn Vishwas Kher
  Roll number: ES19BTECH11015
  Roll: Language Guru
  GitHub ID: KrishnKher

- Name: Mukkavalli Bharat Chandra
  Roll No.: ES19BTECH11016
  Role: System Architect
  GitHub ID: chandra3000

- Name: Sujeeth Reddy
  Roll No: ES19BTECH11022
  Role: System Integrator
  GitHub ID: Sujeeth13

- Name : Sree Prathyush Chinta
  Roll No. : CS19BTECH11043
  Role : Tester
  Github ID: Prathyush-1886

- Name : Praneeth Nistala
  Roll No. : CS19BTECH11054
  Role : Tester
  Github ID: Praneeth-Nistala

- Name : VEMULAPALLI ADITYA
  Roll No: CS19BTECH11025
  Role: System Integrator
  GitHub ID: VEMULAPALLI-ADITYA

Introduction to Michelin

The beginning of an innovation

In recent times, technologies are growing rapidly. Time has become a valuable resource. We, humans, are exploring different ways to replace our day-to-day tasks with machines. Cooking is one such task which humans are trying to automate. In spite of having various technologies, there should be a well-written code that has a specific task.

One might think that there are hundreds and thousands of languages like c++, java, python to do coding, why do we need a new one. The problem with these languages is that the task we are doing is highly domain-specific. This will make the task of writing code in the languages you know very difficult. This might make some good recipes not worth coding. So, we are trying to solve these problems with our language.

A Better Way we think:

A simple and useful way to solve these problems is the Michelin. Some of our language specifications are

- In spite of being an object-oriented language, we are trying to make it simple.
- We are trying to make your codes run on multiple different platforms.
- We are aiming for a high-performance yet we don't want to compromise on the security as any mistake in the code might cost a lot of damage.

Our language is a portable, compiled, high-performance, simple, object-oriented programming language. In this chapter, we are trying to provide a brief look at the main design goals of our language and later examine these features in detail.

The Beginning

Our language is designed to cook dishes by writing code! In a more generic way, our language helps to program a hypothetical machine to cook. It tries to simulate the process of cooking even for someone with bare minimum knowledge of cooking and programming.

The motivation for making this language comes from the fact that cooking seems to innocuously incorporate many design features of modern OOPS languages and these seem to be very common-sense ideas to the common populace. Hence, it indeed seems feasible to design a language that uses the innate organizational qualities that people have to help cook food! It can help cook food with very precise measurements since it will be done through a machine.

Example code snippet:

```
Glass g = new Glass().        \*declaring a glass g*\
Ingredient ing1 = new Ingredient("Water", 100, g).
                            \*100ml Water will be placed in glass g and will  be
                              called as ing1*\
serve g.                     \*serving the glass*\
ready.                       \*this refers that the code is ended*\
```

Here, we are taking a glass and adding water to it and serving it. Your cup of water is ready!

Design Goals

Our language is highly OOPS-oriented. Even difficult recipes can be coded using this language and the rest of the task is left to the machine which will try to cook the required dish. We intend it to be similar to C++, with a very specific focus on cooking and adding on different ideas that are related to it. We would like to support all the basic paradigm functionalities that OOPS supports such as inheritance, polymorphism, operator overloading, etc. since we have found many instances in cooking a dish that has a very close resemblance to many of these ideas. We have multiple other ideas as well, but we are not sure about how many of those other ones we can implement as of now, so we will keep those for later.

Simple, Object-Oriented, and Familiar

One of the important characteristics of the language we wanted to implement is to simplify the language as much as we can so that it does not require any intense special training to code.

We are trying to provide a clean and efficient object-based development environment. This makes it easier to code a recipe in a more organized way which eases the readability. We are planning to provide different libraries which can be used by the programming to make his task much easier.

We want to design our language in such a way that it will be familiar not only to programmers but also to someone who never coded.

Robust and Secure

Our language is designed for creating highly reliable code. Since the machines that use our language to function are assumed to be super expensive, we are planning to design our language such that nothing bad happens to the machine due to the code.

We are discussing different methods to make the task as safe as possible. Some of the ideas we have were discussed in the latter parts. We are not planning to include pointers in our language as it might be confusing to beginners. Since our language expects a low programming experience, we want to provide an environment that identifies the errors in the code and makes it easy for the user to correct the code.

Coming to the part of the security of the code, since the code contains a recipe that might be valuable, we also want to take care of this issue so that the user can write a code without worrying.

Architecture Neutral and Portable

We are not completely sure about this aspect but we are thinking like we will decide based on further readings.

Performance

Performance is an important aspect to discuss. Although we want to achieve high performance, we also don't want to compromise on the risk of the machines. So, we are planning to find a way which tries to balance the performance and the risk factor.

Compiled and Threaded

We wanted to build a procedure for our language similar to c, c++. Like any other compilers, we want to follow the traditional compile, link, and test cycles to convert our code to machine code. Other details of these are and will be discussed later.

We are planning to provide multithreading in our language as it makes the job much faster and thus are planning to include basic synchronizing tools so that there won't be any conflicts between the threads.

The Michelin Environment:

Writing a code using our language is not only easy to code but also easy to read. We are trying to ensure safety and security with a decent performance which helps the programmers as well as the users. Due to the ease of using the language, a programmer only needs to focus mainly on the recipe/process compared to coding it.

Example code snippet:

```
Glass g = new Glass().
Ingredient ing1 = new Ingredient("Water", 100, g).
Bowl b = new Bowl().
b~>add(ing1).     \*adding ing1 to a bowl b and heating it for 5 mins at 60
                      degrees*\
b~>boil(5, 60).
g~>transfer(b).   \*adding the boiled water to the glass g(from b)*\
serve g.
ready.
```

We can see how the code is pretty intuitive and easy to understand.

Language tutorial

Prerequisites:

First ocaml must be installed.

Program Execution:

We have written a make file, which when compiled gives an executable file named parse. The extension for our language is miche, in order to compile our .miche file we have to run the following commands

```
make
./main path_to_file.miche > L2_output.txt
./parse
```

Functions:

Function can have any number of parameters but each parameter type must be specified.

Example:

```
int add(Glass g, Ingredient x) {
\*..... Some code.....*\
return 1.
}
```

The syntax for calling a function is its name, followed by left bracket, list of arguments separated by comma, followed by right bracket.

Example:

```
add(g, ing[ j ])
```

Variable Declarations:

We support inbuilt data types like int, double, float, string, bool

Examples:

```
int a  = 100.
string s = "Water".
bool v = true.
```

We support user defined classes.
Example:

```
class x{
      int a = 0.
      int y(int z1, string z2) {
            \*.... Some code....*\
            return 1.
      }
}.
```

We can create objects of user defined class using

Example:

```
Glass g = new Glass(1000).
```

Arrays:

We support arrays of both inbuilt and user defined data types.

Indexing of the array starts at 0. Number of elements between square brackets must be an integer only.

Example:

```
int a[2].
a[0] = 100.
a[1] = 20.
Ingredient ing[4]<
ing[0] = new Ingredient("CoolWater", 100, g).
ing[1] = new Ingredient ("Lemon juice", 2*get_volume(tb), g).
ing[2] = new Ingredient("Sugar", 3*get_volume(tb),  g).
ing[3] = new Ingredient("Salt", (get_volume(tb))/2, g).
```

```
>.
```

Statements:

We support if-else statements, for statements , while statements and two special instructions ready and serve.

Syntax for if-else declaration

```
if(\*some boolean expression*\)
{
\* some code*\
}
else{
\* some code*\
}
```

Syntax for for loop:

```
for(\*initialization*\;\*some boolean expression*\; \*increment or
decrement*\)
{
      \*Some code*\
}
```

Syntax for while loop:

```
while (\*boolean expression*\) {
       \*Some code*\
}
```

Syntax for ready statement:

ready.

Syntax for serve statement:

```
\*if G was a glass that contains something*\
serve G.
```

Language reference manual

A Programmer's Toolbox

Primitive Data Types

In the libraries of our language, we use various data types of C++ like int, double, string, etc as the ones we work with. To compare and measure different quantities in Michelin, we use classes to store a few basic data types related to the class and their scaled quantities as well.

Examples:

```
int a  = 100.
string s = "Water".
bool b = true.
```

Arithmetic and Relational Operators

Arithmetic Operators

Operator	Description
+	Adds two operands.
−	Subtracts the second operand from the first.
*	Multiplies both operands.
/	Divides numerator by de-numerator.
%	Modulus Operator and the remainder after an integer division.

++	The increment operator increases the integer value by one.
--	The decrement operator decreases the integer value by one.

Example code snippet:

```
\*Example for + operator*\
\*let a, b be 2 vessels*\
int temp = a~>get_volume() + b~>get_volume().
\* Adding volumes of a, b & storing it in temp*\.
```

```
\*Example for  * operator*\
\*let g be a glass*\
Ingredient ing("water", 4*g~>get_volume(), b).
\*volume of water is 4 times that of volume of glass g*\
```

Relational Operators

Operator	Description
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.
>	Checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition becomes true.
<	Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition becomes true.

>=	Checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition becomes true.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition becomes true.

Example:

```
bool temp = ( i < j).
```

Logical Operators

Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false.

Example:

```
bool temp = ((i < j ) && (a < b)).
```

Assignment Operators

Operator	Description
=	Simple assignment operator. Assigns values from right side operands to left side operand
+=	Add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.

Example:

```
i += g~>get_volume().
```

Miscellaneous operators

Operator	Description
&	Returns the address of a variable.
? :	Conditional Expression.
~>	Accessing class methods.

Operator Precedence Table

Category	Operator	Associativity
Postfix	() [] ++ - -	Left to right
Accessor	~>	Left to Right
Unary	+ - ! ~ ++ - - & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right

Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= &= ^= |=	Right to left
Comma	,	Left to right

Arrays

Just like any other language, Arrays in Michelin are used to store multiple data of the same type together at one place.

Multidimensional arrays are not supported by Michelin.

There are 2 types of array declarations, one which is similar to c style declaration for primitive data types and another one is for inbuilt data types and classes.

1. Example:

```
int arr[0].
arr[0] = 2.
```

2. Example:

```
Plate p = new Plate().
Bowl b = new Bowl().
Ingredient x[3]<
x[0] = new Ingredient("Apples", 2, unit, p). \*name, quantity,
typeOfQuantity, storeIn*\
x[1] = new Ingredient("Grapes", 7, unit, b).
x[2] = new Ingredient("Berries", 7, unit, b).
>.
```

Strings

Strings exist in our language for multiple purposes. Some of them can be said as:

- To send in some specific instructions like saying to chop the onions into smaller pieces.
- For input/output requirements, etc.

Strings are also used in the language to specify the type(/state) of the ingredient stored, dishes, etc.

Example:

```
String s = "Wheat flour".
bowl b1.
Ingredient item(s , 250,b1).
```

Memory Cleanup

We use destructors to clean up the memory with which we can automatically free up the memory used by those particular classes.

Functions

Functions are a set of instructions bundled together to achieve a specific outcome. They are a good alternative to having repeating blocks of code in a program. Functions like boiling which do not require a specific instance of a class to call can be declared as global functions.

Polymorphism

```
Vessel a[2]<
a[0] = new Glass().
a[1] = new Bowl().
>.
```

We are implementing a simple polymorphism, as shown above since Vessel is a superclass of Glass and bowl, an array of Vessel is created in which the elements are of type Glass and Bowl.

Object-Oriented Programming Welcomes Michelin

When we initially thought of the language, it quite quickly came to our attention that this language is by nature, object-oriented! Why? Well, let's dive head-first into the principles of object-oriented programming and see how the whole idea of cooking a recipe blends into it.

Here are some of the primary principles of object-oriented programming (we'll just state them briefly here since the principles themselves are quite famous and our focus here is more on how Michelin incorporates these ideals) :

Encapsulation

Encapsulation is the mechanism of hiding data implementations by restricting access to public methods. Instance variables are kept private and accessor methods are made public to achieve this. It is basically a way of grouping data and "scripts" that tell us how to handle the data (functions, in a programmer's dictionary!) cleverly so as to make usage of the whole platform easy for the user.

This mechanism is required by our language to ensure that a user doesn't try and make changes to certain attributes declared in classes.

For example, a certain ingredient class will have a state attribute to which we may not want the user to have access, or another case would be when we may not want the users to change the dimensions of a container object. This mechanism ensures that such events don't occur.

Consider the following example:

```
\* User code *\

Vessel v.
v.dimensions = 1. \* Error. *\
```

This statement will return an error as dimensions is a private attribute of the vessel and the user cannot make changes to it.

Abstraction

Abstraction refers to the idea of describing the intent of a certain class or interface at a high level without delving into the low-level details of its implementation. Many times, all we need to know from a class is what it does and which subroutines or methods do we need to use in order for it to lend some desired functionality.

In the context of cooking, we don't need a lot of different types of objects. We came up with 3 high-level classes that describe most of the functionalities that we need. These classes are: "Container", "Tools", and "Ingredients." The Container class, for example, represents objects such as pan, vessel, cauldron, cooker, etc. The Tools class represents objects such as ladle, spoon, knife, peeler, stirrer, etc. The Ingredients class is slightly more complex to explain but for now, in the context of abstraction, we just mention that some of its child classes are "Potato", "Milk", etc.

Ok, so where is the abstraction in all this?

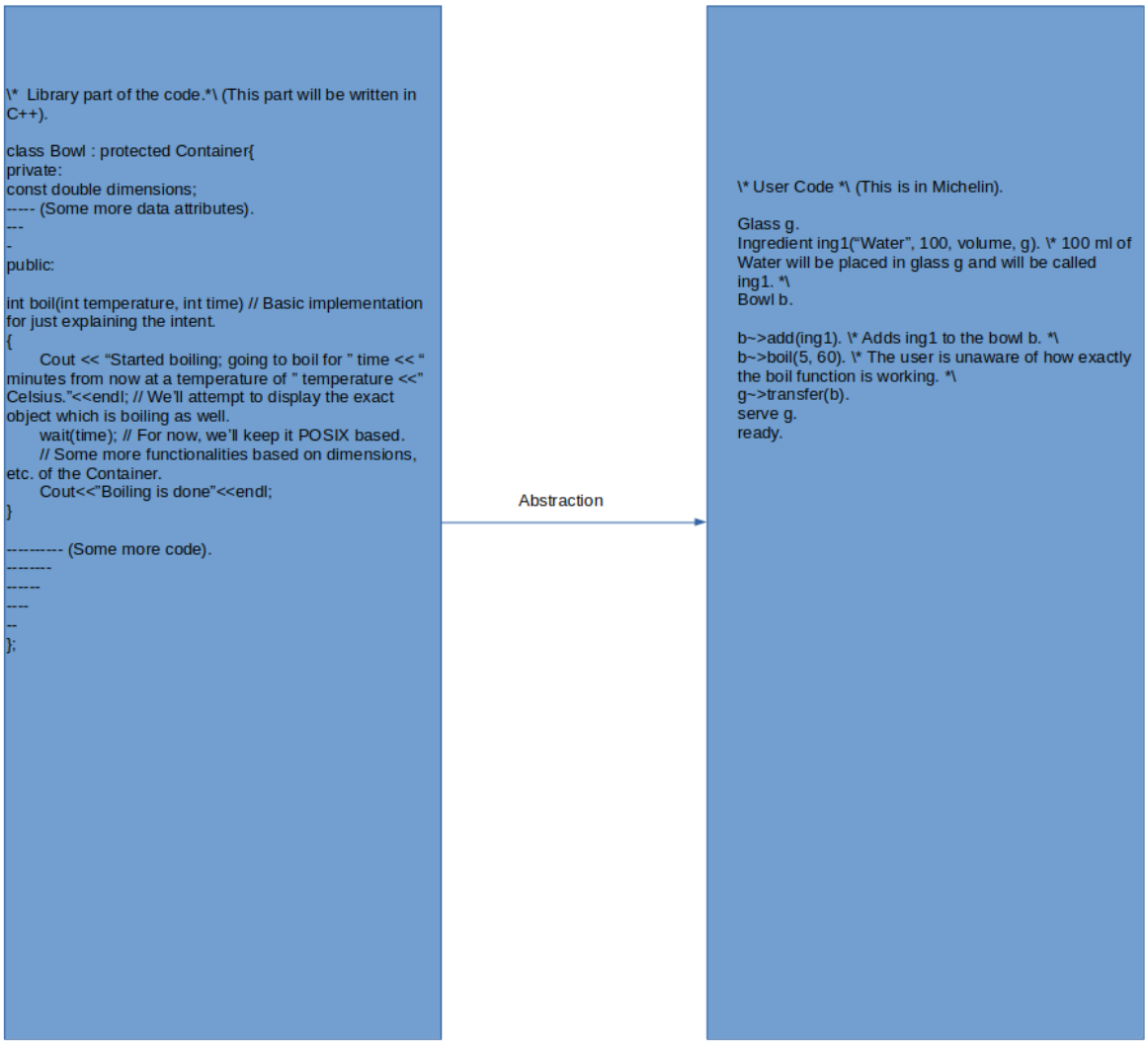
Well, the abstraction comes from the fact that all that has been mentioned above is what the classes represent and we haven't really revealed how these classes aka blueprints are actually internally storing these ideas.

For example in the Containers class, we will be having a "boil" function common across all subclasses of the Containers class, because we believe that most cooking containers can be used to boil a particular ingredient.

Now when the user writes code and uses the “boil” function, the user is unaware of how exactly it has been implemented for the particular instance of the Container class that he declared (say vessel)!

However, the overall blueprint self-explained by the class is sufficient for the user to use the desired functionality.

Consider the following example:

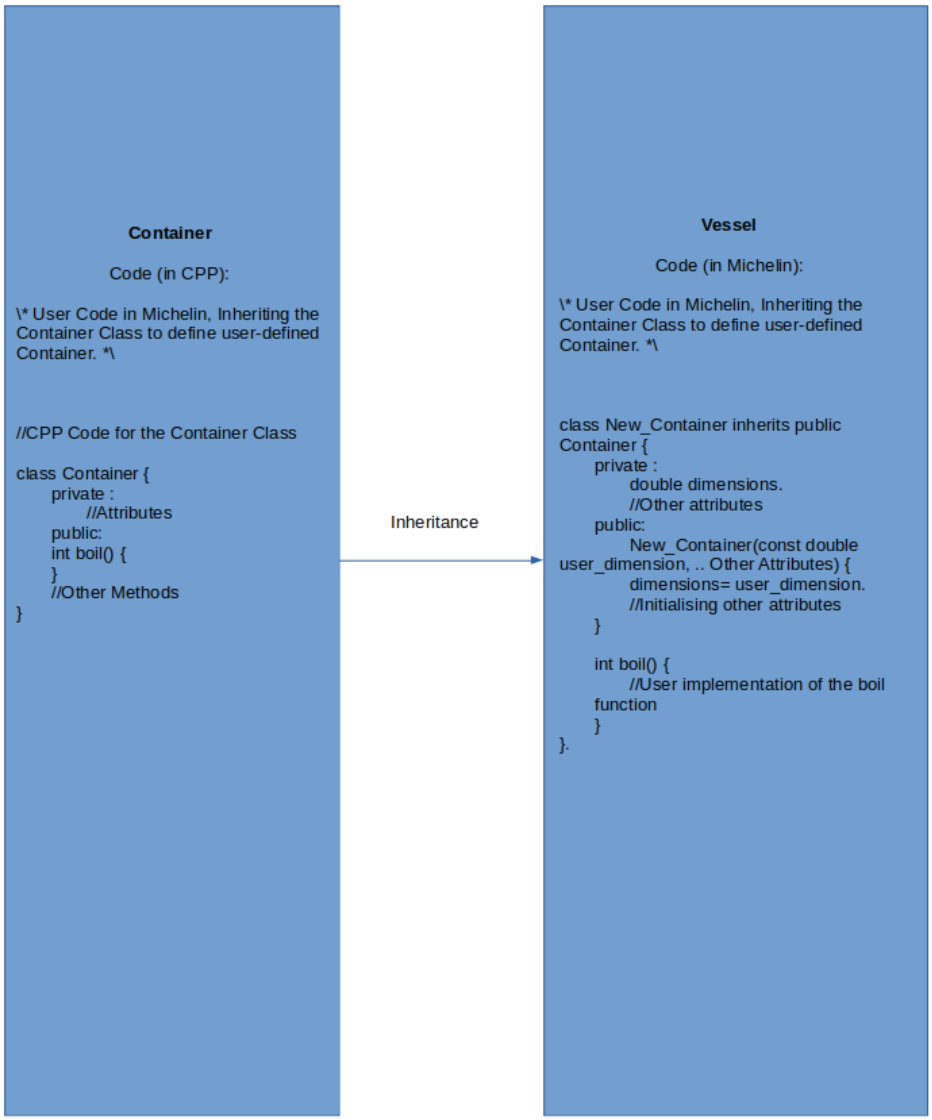


Inheritance

Inheritances express an “is-a'' relationship between two classes. We can reuse the code of existing superclasses in the derived classes using inheritance. It also helps us better understand how one class depends on the functionality of another class, which is usually called the parent class.

Extending from the previous description, we see that inheritance is crucial for the way we conceive of the cooking process in Michelin! In fact, if we take note of the Containers class, we see that within the Containers class, we have some state-based subclasses. One example of a state-based class could be based on the Vessel subclass present as a child class of the Containers class. We have SmallVessel and LargeVessel as subclasses of the Vessel class. Now both SmallVessel and LargeVessel will inherit the boil function present in the Vessel class, and might make a few modifications to the boil class present in the Vessel class in relation to the size (e.g. constants related to max_temperature that the vessel can be boiled given that the Small/Large Vessel object has initially been filled to x% of the vessel capacity).

There is also another kind of behavior that is closely related to, if not identical to inheritance that we wish to add into our language. This is in relation to the idea that sometimes one may want to use the output of a particular sub recipe in another larger recipe (somewhat analogous to C++’s lambda functions)



The above example shows Inheritance in Michelin.

Polymorphism

In Greek, poly refers to “many/multiple” while morph refers to “shape/form.” As can be inferred from the etymological break up of the word, polymorphism refers to the idea of a single entity externally having the same form but performing different actions. Some common ways polymorphism is demonstrated is through the ideas of runtime and compile-time polymorphisms.

A simple example of this in our language would be in the following scenario.

Let's suppose that we want to add multiple kinds of ingredients to a single pan, so for this example, we could take milk and sugar. Now milk and sugar are fundamentally different entities in regards to the state itself (milk is a liquid ingredient while sugar is a solid ingredient).

But we provide the ability to store both of these "different" entities in a single array of type "Ingredient." This is an example of polymorphism in the context of storing data. This is just the tip of the iceberg in terms of its functionality.

Here's an example code snippet demonstrating the use of overriding a particular function.

Example:

```
\* User Code in Michelin. *\
class Custom_container inherits public Container{
      Double dimensions.
      override int boil(int time){
            //my own implementation of the boil function
      }
}
```

Here what we are doing is creating our own custom class and inheriting a predefined class and overriding the boil method to allow the user to make his/her own implementation of the boil method. This acts as another example of abstraction that the user herself/himself can use.

Other Details

1. All the classes defined in Michelin Language will be redefined in CPP internally.
2. By default, all the functions in class Container and class Tools are virtual.
3. The user cannot define a function as virtual for her/his classes.

Now from the above-given examples, it can clearly be seen why we choose to design our language using the OOP paradigm. To summarize in a nutshell everything in a kitchen can be visualized to be an object with certain properties and our goal is to make use of these properties to make a certain desired delicious output.

Security In Michelin

Memory Layout

Michelin does not make use of pointers like in C and C++. The Michelin compiled code references memory via symbolic handles that are resolved to real memory addresses at run time.

Limitation checks

Since our language is used to instruct a machine to prepare a delicious dish, it is our duty as language designers to run a check on whether or not the given recipe input is feasible for the hardware system or not.

So we have implemented a security check level that runs a simulation of the given recipe and checks whether or not it can run within the limitations of the hardware system. If these checks weren't done then although logically it may seem correct, it may cause wastage of ingredients.

For example, if a vessel object of a particular volume is created and is used to store milk. If the user adds excess milk into the vessel it will overflow and lead to lots of wastage. So before the code is executed we throw an error and ask for this to be fixed. Similarly, we don't only check for semantic errors, we do device-dependent checks as well like ensuring the user doesn't try cooking on stoves that aren't present on the device or accessing a mode not present on the device

Project plan

Week - 0

- Fixed the different aspects of the language like the programming paradigm, features and other logical aspects of our language
- Researched more about the lexical analyzer in theory
- Discussed the likely implementations which are above the scope of this project.
- Wrote a Language Specification Document on the lines of Java Whitepaper.

Week - 1

- Finalised the syntax for the language.
- Started exploring OCaml, Ocamllex, Ocamlyacc
- Started working on the code of the lexer

Week - 2

- Finished the lexer by using OCaml.
- Testing and debugging the lexer with more examples
- Working to make a few changes to the Language Specification Document.
- Worked on more examples for the language
- Explored more about the theoretical aspects of the Parser

Week - 3

- Improved the Lexer after the debugging
- Submitted the modified Language Specification Document
- Started with the code of Parser

- Worked on the basic templates for the presentations and created presentations for the Lexer Phase

Week - 4

- Completed the basic code for Parser
- Trying to add further features into it
- Added a makefile for ease of execution

Week - 5

- Created presentations for the Parser phase.
- Working on modifying the parser
- Started the theory part of the Semantic Analyzer

Week - 6

- Implementing a few of the features in the parser
- Exploring more of Ocaml to utilize the inbuilt data structures
- Completed theoretical base required for the Semantic Analysis phase
- Made a few changes to the grammar thereby refining it and reflecting them in the previous code files

Week - 7

- Started working on the symbol table and Ast
- Started writing tests for the Semantic Analyzer
- Started writing some preliminary error handling in the Analyzer

Week - 8

- Working on the code to build the symbol table
- Building our Semantic Analyzer with further added test case codes

- Modified the Parser accordingly

Week - 9

- Completed checks related to Redeclaration of variables, type mismatches and function arguments' type error checks
- Completed the presentation and demo for the semantic phase.
- Working on finishing implementing the additional features.

Week - 10

- Refining the Semantic Analyzer and fixing a few oversights in the design
- Started working on the theoretical aspects Code generation phase
- Worked on a few features of the language and their implementation

Week - 11

- Resolving issues with the Semantic Analyzer and handling the code accordingly
- Made necessary changes to Parser
- Working to get more insight on the Code generation phase of the compiler

Week - 12

- Worked on refining the Semantic Analyzer
- Started working with the code of the Code Generator
- Working on the Code generator's design and trying to fix a low-level language for it.

Week - 13

- Worked on the presentation of the Code Generator
- Improved the code of the Code generation
- Started working on other documents related to the Language

Language evolution

It started with trying to think of some creative ideas as was told to us. We wanted to avoid referring to the Internet for new ideas, as we felt that it might prevent us from thinking more freely in different directions.

Most of us in the team had taken the PoPL-II course so we had a fair idea of some of the most common programming paradigms such as object-oriented, procedural, functional, etc. We felt that coming up with a functional language would be hard and implementing a compiler for it would be even harder.

We felt more comfortable with the awesome modularity features that object oriented programming as a concept incorporates within itself. They felt much more intuitive to comprehend for us in particular, and in general for most people!

Hence we decided to try to come up with some sort of an object-oriented language for which it would not be too hard to develop a basic compiler as well as have some particularly creative ideas behind it.

We were planning for a more domain specific language then a general purpose one as we felt that it would be both easier and provide a better learning experience for us if we were to go for the former.

Having been trained to “see” object orientedness in our surroundings in previous courses, we tried to think of some regularly faced problems/inconveniences that could be solved via a programming language that in particular is object oriented.

So there were multiple suggestions on the same, some of which are listed below:

1. **Idea**: Have a language that could take a directory (on GitHub or even on the local PC for instance) and output a tree kind of structure, or some relevant information that would help in searching for things that we wish to look at in codebases.

Inspiration: many hours spent on being unable to find relevant bits of information or code that we wanted to see while searching through GCC and LLVMs’ codebases (from previous assignments/courses)!

2. **Idea**: Calculating the units of different quantities related to physics and implementing some language which could possibly make some computations commonly done in physics or other such stuff more viable to do via a programming language.

Inspiration: we have been fairly acquainted with physics for quite a while during JEE and some of us are still fascinated by the concepts involved in it and so we just thought of coming up with something useful for understanding or performing operations useful for physics.

3. **Idea**: Come up with a programming language that could assist common people as well as industries in cooking efficiently and using precise measurements and recipes, which are easily programmable.

Inspiration: Something which we see so often every single day is the process of cooking! As we started thinking more on it in depth, we realized how naturally modular cooking can actually be and actually is when done by professional chefs. The idea of having a machine (yes, some minor ideas from the theory of computation in regards to Turing machines!) that could cook in a modular fashion by having separate slots for ingredients, tools and vessels.

Note that the above 3 components that we mentioned also incorporate modularity within themselves! For instance, within vessels we usually have small sized vessels, medium-sized, and larger sized ones. That could be one point of having separate subclasses. There are other sub classification metrics possible.

In addition, many times, the way an algorithm is introduced is as a “recipe”. So we thought of reversing the correspondence, i.e use algorithmic toolboxes (a.k.a computers via which we could program machines) for cooking!

4. **Idea**: An extension of the above idea (3^{rd}) to some manufacturing domains, and even to inventory and shipping!

Inspiration: The 3^{rd} idea could be in most cases, extended to any domain which incorporates a reasonable amount of modularity in its design and is feasible enough to program in.

We finally decided to go ahead with the 3rd idea. Some of the reasons behind coming up with this decision are stated as below:

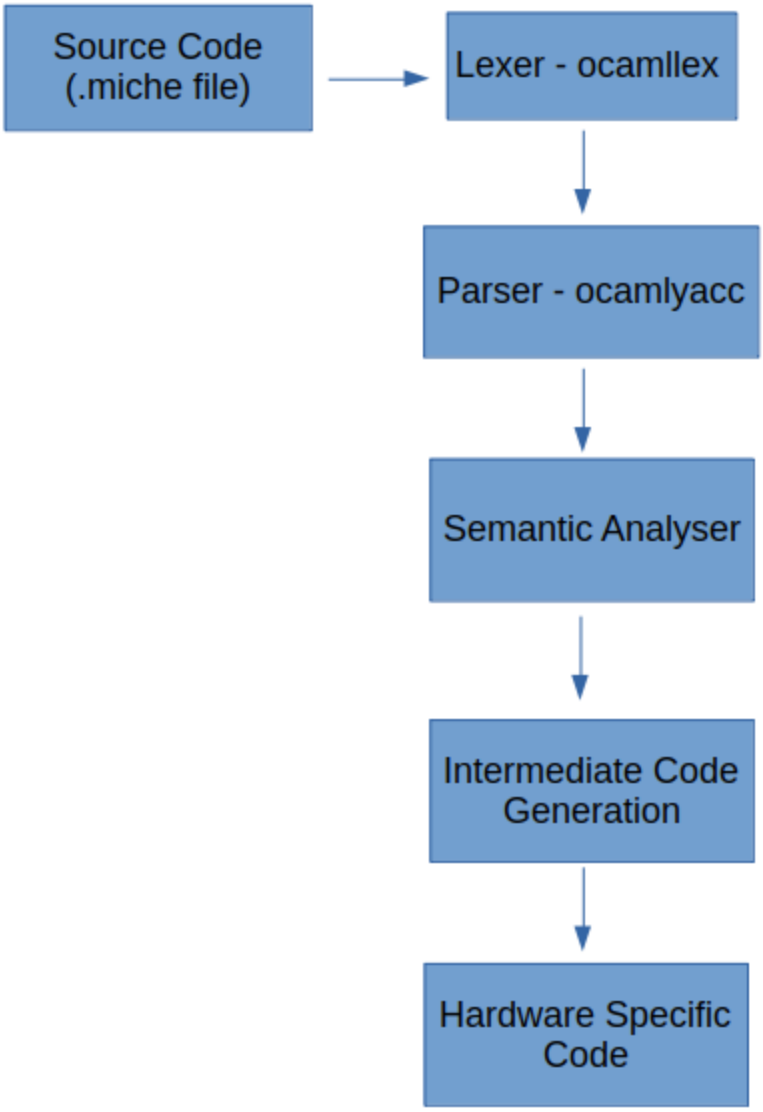
- Some ideas, although greatly useful, didn't seem to require a whole programming language to be invented for them.
- We felt that we didn't have enough domain knowledge to implement some other great ideas mentioned in the list.
- For some of the ideas, we had a very high level idea of how it would be useful but we weren't able to pin down what exactly would we need to achieve those ideals in a programming language for the same.
- Testing the outputs of the languages was going to be hard for at least some of the proposals.

After this, we needed to come up with a good naming for our language. The idea behind naming our language as Michelin is explained here. We understood that it would be hard to test it, but once we explained the intended output and how we might want to go forward with it, the TAs were fine with it, and so decided to pursue the idea.

The idea definitely, we believe, must be having some internal inconsistencies that we might not have been able to think of, but at that moment of time, that seemed the best way to go forward. And luckily, we happened to learn many things in the process of developing this project!

Compiler architecture

The Compiler Architecture of the Michelin language is as follows:



Lexer

The lexer takes in the ".miche" (The extension of Michelin Language - .miche) program and converts it to tokens. The lexer generates tokens using regular expressions and pattern rules. The lexer discards all the white spaces and the comments present in the program. The stream of tokens generated are passed to the Parser. Michelin's lexer has been implemented using OCamllex.

Parser

The parser takes in the stream of tokens generated by the Lexer. The parser generates the syntax tree using the Context Free Grammar. The parser detects any syntax errors present in the program. The implemented parser is a "Look Ahead LR" parser. The parser for Michelin has been implemented using OCamlyacc.

Semantic Analyser

The semantic Analyser uses the syntax tree generated by the parser to check the source code for semantic consistency with the language definition. One of the important purposes of semantic analyser is type-checking. If there are no semantic errors, the semantically checked syntax tree is used for intermediate code generation.

The semantic analysis in Michelin is performed in parallel to the parsing, which seems to be contrary to the diagram shown above. During the parsing stage, whenever the token matches with a particular rule or action, the token is either added to the symbol table (a hashmap) or the constraints against the token are checked in the symbol table. If there are no conflicts with the tokens and the symbol table the semantic analysis and the parsing are successfully completed. If there is a conflict between the matched tokens and the symbol table, an error is raised and both the processes are terminated. This behaviour is similar to that of an interpreter.

In our semantic analysis we have implemented certain functions for handling variable/function declarations along with some preliminary scope checks. We have tried to

implement separate functions for separate kinds of data types as can be seen from our code. We believe that this kind of design helps in easy maintenance and debugging of the code. It also improves readability.

Intermediate Code Generation

The intermediate code that is generated is not architecture-neutral. Our main aim is to generate a code that is understood by particular hardware (like in Arduino programming as an example). In this case, the hardware is the particular device (hypothetical machine) that will actually be cooking the dishes as per our programming commands (e.g the hardware requirement for the machine shown here will be those as required for the pizza making robot).

Moreover, there were some commands that we planned to translate into database queries. An example of such a statement is where we want to fetch a certain amount of rice for a particular dish that we wish to make from the food inventory (connected to the device). We wanted to try to make the language in such a way that the user just has to write a very simple statement for fetching that but which would internally get translated into a query from the map in which we would be storing the ingredients.

However, we weren't sure how exactly we could construct this device. In particular since we weren't sure about which target (robotic) architecture we needed to generate the code for, we decided to generate the intermediate code in LLVM.

Hardware Specific Code

The final code that is generated is compatible with a hypothetical machine that can cook. As mentioned above, this stage doesn't have much use for us currently, since we can't design the hardware for the hypothetical machine that we have proposed right now.

Development environment

Ocaml:

We decided to use ocaml as the programming language since we felt that it provided good library support to make the process of developing a compiler nice and smooth.

We also decided to use Ocaml since it was a functional programming language and we wanted to move out of the conventional imperative style and try out a new paradigm.

Now the main features provided by Ocaml are two program generators namely ocamllex and ocamlyacc.

These program generators are very close to the well-known lex and yacc commands that can be found in most C programming environments.

Ocamllex:

The ocamllex command produces a lexical analyzer from a set of regular expressions with attached semantic actions, in the style of lex. Assuming the input file is lexer.mll, executing “ocamllex lexer.mll” on the terminal produces OCaml code for a lexical analyzer in file lexer.ml.

Ocamlyacc:

The ocamlyacc command produces a parser from a context-free grammar specification with attached semantic actions, in the style of yacc. Assuming the input file is parser.mly, executing “ocamlyacc options parser.mly” on the terminal produces OCaml code for a parser in the file parser.ml, and its interface in file parser.mli.

The generated module defines one parsing function per entry point in the grammar. These functions have the same names as the entry points. Parsing functions take as arguments a lexical analyzer and a lexer buffer, and return the semantic attribute of the corresponding entry point. Lexical analyzer functions are usually generated from a lexer specification by the ocamllex program. Lexer buffers are an abstract data type

implemented in the standard library module Lexing. Tokens are values from the concrete type token, defined in the interface file parser.mli produced by ocamlyacc.

GNU Make:

We created a makefile since we had to run a lot of commands to compile and then link the object files before running the executable. Also in Ocaml it is necessary to remove all previous intermediary object files before recompiling. This task would be extremely tedious without the make clean command.

The advantages of Make are:

- Make enables the end user to build and install your package without knowing the details of how that is done.
- Make figures out automatically which files it needs to update based on which source files have changed.
- GNU Make has many powerful features for use in makefiles which other Make versions don't have. It can also regenerate, use and then delete intermediate file which need not be saved.

Git:

We used Git as our version control system. We pushed our work into a remote on GitHub whenever one of us finished working on some feature. We have done some code review and then modified, pulled and pushed etc.

VSCode:

We used VSCode to do all our coding. The main reason to use **VSCode** was due to **Liveshare** which is an extension that allows real time interactive coding sessions allowing all of our team members to work in a collaborative fashion to make updates to the directories. It also provides features like Terminal sharing, Code sharing, syntax highlighting etc which helped in completing the project in a very collaborative and comfortable way.

Test plan and test suites

Introduction:

We wrote some test cases, which uses all necessary program structures like arithmetic, loops, conditional statements, classes etc which can be run to check the correctness of the project. We manually tested our code to ensure that the outputs for the respective phases (lexer,parser etc.) are in accordance with the correct outputs.

Example files:

Example 1:

This is a basic program in our language michelin which is an analogous hello world program in C, C++ languages.

```
class bowl{
bool boil(int time, int temp) {
            \* some code *\
            return true.
}
}.
int kitchen( )
{
bowl b = new bowl().
Ingredient ing1 = new Ingredient("Water", 100, b).

\* 100ml Water will be placed in bowl b  and will be called as ing1 *\

b~>boil(5, 60).
serve g.
ready.
}
```

Example 2:

This is a program that peels and cuts an apple giving the resultant pieces of apple on a plate to the user.

```
class peeler{
      bool peel(Ingredient ing) {
            return true.
      }
}.

class knife{
      bool cut(Ingredient ing) {
            return true.
      }
}.
int kitchen()
{
      Plate p = new Plate(25).
      Ingredient ing = new Ingredient("Apple", 2, p).
      peeler pl = new peeler().
      pl~>peel(ing).
      knife k1 = new knife().
      k1~>cut(ing).
      serve p.
      ready.
}
```

Example 3:

This is a simple program that can create a salad, here we are creating a fruit salad which consists of apples, grapes and berries that are added back to the same container, mixing this and serving the salad on a plate.

```
class knife{
      bool cut(Ingredient ing) {
            return true.
      }
}.
class bowl{
      bool mix(int time) {
            return true.
      }
}
```

```
bool transfer(bowl b, Plate p) {
      return true.
}
int kitchen(int y)
{
    Plate p = new Plate(25).
    bowl b = new bowl().
    Ingredient x[3]<
    x[0] = new Ingredient("Apples", 2, p).
    x[1] = new Ingredient("Grapes", 7, b).
    x[2] = new Ingredient("Berries", 7, b).
    >.
    knife k = new knife().
    k~>cut(x[0]).       \*cuts x[0] and places back in same container*\
    transfer(b, p).     \*transfers contents of p to b*\
    b~>mix(0.1).
    serve b.
    ready.
}
```

Example 4:

This example depicts a program for the creation of a fruit milkshake where in this case it's an apple milkshake.

```
class peeler{
      bool peel(Ingredient ing){
            return true.
      }
}.
class knife{
      bool cut(Ingredient ing) {
            return true.
      }
}.
```

```
class foodProcessor{
      bool add(Ingredient ing) {
            return true.
      }
      bool grind(int time) {
            return true.
      }
      bool pourInto(Glass g) {
            return true.
      }
}.

int get_volume(TableSpoon tb) {
      int val.
      return val.
}

int kitchen(int y )
{
    Plate x = new Plate(1).
    Glass g = new Glass(1000).
    Ingredient ing1= new Ingredient("Apple", 2, x).
    Ingredient ing2 = new Ingredient("Milk", 100, g).
    peeler p = new peeler().
    ing1=p~>peel(ing1).
    knife k = new knife().
    ing1=k~>cut(ing1).
    foodProcessor fp = new foodProcessor().
    fp~>add(ing1).
    fp~>add(ing2).
    for(int i = 0; i < 3; i++) {
            TableSpoon tb.
            Ingredient ing("Sugar",get_volume(tb), tb).
            fp~>add(ing).
    }
    fp~>grind(2).       \*Grinding for 2 minutes*\
    fp~>pourInto(g).
    serve g.
    ready.
}
```

Example 5:

This example depicts the program for the creation of lemonade. In this example we have showcased some of the function creation features of michelin as well as a few simple loops.

```
int add(Glass g, Ingredient x) {
     \*..... Some code*\
     return 1.
}

int stir(Glass g, int time)
{
      \*..... Some code*\
      return 1.
}
int x(int vol){
      Glass g = new Glass(vol).
      Ingredient ing[4]<
      ing[0] = new Ingredient("CoolWater", 100, g).
      ing[1] = new Ingredient ("Lemon juice", 2*get_volume(tb), g).
      ing[2] = new Ingredient("Sugar", 3*get_volume(tb),  g).
      ing[3] = new Ingredient("Salt", (get_volume(tb))/2, g).
      >.
      int j = 0.
      while( j < 4)  {
            add(g, ing[ j ]).
            stir(g, 0.5).
            j++.
      }
      serve g.
}
int kitchen(int y)
{
    int i = 0.
    for( i = 0; i < 5; i++)
    {
         \*Here lemonade function is not declared*\
         x(1000).
    }
```

```
    ready.
}
```

3.Working Test Cases:

Case 1:

```
int funct(){
      string g.
      int x = 1.
      double b.
      return 0.
}
```

Case 2:

```
int kitchen(bool a)
{
    if (a==True)
    {
        return 24.
    }
    else
    {
        return 12.
    }
}
int func(int x)
{
    bool a = False.
    int res.
    res=kitchen(a).
    return 0.
}
```

Case 3:

```
int kitchen(int x)
{

    string s = "abc".
```

```
    int a = 123.
    int b = 345.
    int c = 678.
    if(a > b)
    {
        a = a * 123.
        c = c + 678.
    }
    else{
        b = b / 345.
        c = c + 2.
    }
}
```

Case 4:

```
int main() {
      int arr[5].
      int i.
      for(i = 0; i < 5; i++) {
            arr[i] = i.
      }
      return 0.
}
```

4.Error Generating Test Cases:

Case 1:

This generates an error as identifiers in our language cannot begin with a numeric digit.

```
class Peeler{
      bool peel(Ingredient ing){
            return true.
      }
}.
class Knife{
      bool cut(Ingredient ing) {
            return true.
      }
}.
```

```
int kitchen(int y)
{
      Plate p.
      Ingredient ing("Apple", 2, unit, p).
      Peeler 1p.
      ing=p~>peel(ing).
      Knife k1.
      ing=k1~>cut(ing).
      serve p.
      ready.
}
```

Case 2:

This code ends up generating an error due to the fact that the multiline comment that has been opened hasn't been closed in the code.

```
class glass{
      bool add(Ingredient ing)
      {
            return true.
      }
}
int kitchen(int y)
{
    glass g.
    Ingredient ing1("Water", 60, g).
    bowl b.
    Ingredient ing2("Ice Cubes",  4, b). \* comments should end *\
    glass g.                    \* This is a glass of volume 120
    g~>add(ing2).
    serve g.
    ready.
}
```

Case 3:

This ends up generating an error as the variable itemname that has been declared as a string is now equated to an integer value which is not allowed.

```
class glass{
      bool stir(int time)
      {
            return true.
      }
}
int kitchen(int y)
{
      glass g.
      string itemname= 435.     \*Here the error is that variable of type
                                  string has been equated to integer.*\
      Ingredient ing1(itemname,300,g).
      Ingredient ing2("sugar",4,g).
      Ingredient ing3("salt", 4,g).
      g->stir(5).
}
```

Case 4:

This example generates an error as symbols like @, $ and so on cannot be used in the identifier names.

```
class bowl{
      \*Some code*\
      bool add(Ingredient ing)
      {
            return true.
      }
      bool mix(int time)
      {
            return true.
      }
}
```

```
class plate{
      \*Some code*\
}



class knife{
      \*Some code*\
      bool cut(Ingredient ing)
      {
            return true.
      }
}

int kitchen(int y)
{
    bowl b = new bowl().
    plate p1 = new plate().
    plate p2 = new plate().
    Ingredient x = new Ingredient("Apples", @2, p1).
    Ingredient y = new Ingredient("Grapes", 7, b).
    Ingredients z= new Ingredient("Bananas", 1, p2).
    knife k = new knife().
    k~>cut(x).
    k~>cut(z).
    b~>add(x).
    b~>add(z).
    b~>mix(0.2).
    serve b.
    ready.
}
```

Case 5:

This program generates an error as the opening bracket of the kitchen function doesn't have a corresponding closing bracket.

```
int kitchen(int y){
    string g.
    int x = 1.
    double b.
```

Case 6:

This programming ends up generating an error telling that there is no such function funct as our usage of funct in the kitchen function took an integer and a string as parameters, while our declaration for funct was two integers as parameters.

```
int funct(int a , int b) {
    return a.
}
int kitchen(int p) {
    string var = "1".
    int t = 0.
    int y.
    t = funct(t,var).
}
```

Conclusions & Lessons learnt

The idea we started off with Michelin was to build a coding language and a compiler which will be connected to some kind of a machine that will cook based on the instructions from the code. We named it Michelin as inspired from Michelin Star, the ultimate hallmark of culinary excellence. But the more we aimed into the front end functionalities, the more we were made aware of the depths of the compiler base it needed.

Coming into this project with a very little idea of how compilers break down the language and compile/execute it. Our only previous experience with it was a brief idea from the previous course(Compilers-1). Even though we were put to work on making a totally new language in POPL-1, we were not aware of its plausibility and feasibility. And coming to this course from that knowledge, was groundbreaking and we had to reconstruct our ideas of thinking according to the scope of making it into a working compiler.

With this course, we got to know in detail about the different parts of a compiler and what exactly is the functionality of each of them. This was a whole new experience. Trying to work with a relatively new programming language like Ocaml for the first time was pretty challenging, but with the time and references, we managed to get a grasp of its structure and syntax and wield it well for our Compiler.

We struggled at many different places like while making the symbol trees, writing hashtables to keep a check on things in the semantic analyzer, Trying to use Lexer output into Parser and Parser output into Semantic Analyzer, trying to convert it into LLVM for the code generation purposes and many more. But we also came through all of those. We got to know how makefiles and shell scripts work. We got to know how the changes in the language grammar affect the compiler and also the building of a compiler.

For further extensions into it, one can think of trying to implement libraries into it, try to explore the ways of parallel computing, work on the electrical machine to make it possible and also optimize the code in hand. We learnt that there is a lot to explore more under every aspect of the compiler and one can never endingly work on efficiency.

Appendix

Project Log

```
commit 3140718f967e38395cbe41a3dc17b5702683defa (HEAD -> main, origin/main,
origin/HEAD)
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Mon Dec 6 14:51:59 2021 +0530

        Update Improvements.md

commit a0eee8999a280c73850e0510c82d50ba04754c14
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Mon Dec 6 14:07:33 2021 +0530

        Rename correct3.miche to correct2.miche

commit e8d000734ffdd5afcd2feb1eb4839f6679712216
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Mon Dec 6 14:07:19 2021 +0530

        Delete correct4.miche

commit 1be7cb5a59f0490a089e43db03093f7a4149c80c
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Mon Dec 6 14:07:07 2021 +0530

        Delete correct2.miche

commit d4b778fd2281cb91fe029d6936d5080f256df4a7
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Mon Dec 6 13:57:15 2021 +0530

        Update incorrect2.miche

commit 410be33561f1f1af4c5f33c0f7835a68a831e298
Author: Praneeth-Nistala <74588016+Praneeth-Nistala@users.noreply.github.com>
```

```
Date:   Sun Dec 5 16:00:58 2021 +0530

        Update Parser.mly

commit 6827efc5e191db231cf3dd98a05b4839221aee4b
Author: G-Sidhardha <74396985+G-Sidhardha@users.noreply.github.com>
Date:   Sun Dec 5 11:28:10 2021 +0530

        Add files via upload

commit cc036094ea378aef408777852c3ec249b9d46782
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sun Dec 5 10:46:58 2021 +0530

        Delete Init.txt

commit 2a22a87e3c1c61c38e7d1446e651848de5bc3a74
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sun Dec 5 10:46:11 2021 +0530

        Update Video_links.md

commit 098085ee5811d0fe2f7eaa0cce76e826ef5dd8e5
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sun Dec 5 10:45:41 2021 +0530

        Update Video_links.md

commit 99ddf6fec7335b51e1c43cd67b785f227f06700c
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sun Dec 5 10:42:41 2021 +0530

        Create Video_links.md

commit b592672c2004c758fa07805c38bf24eb37e2b624
Author: Sujeeth13 <72500256+Sujeeth13@users.noreply.github.com>
Date:   Sat Dec 4 22:49:56 2021 +0530

        Update Semantic.ml

commit c9ce69b59558e97d8beee66cc8d55a9f168af4ec
Author: Sujeeth13 <72500256+Sujeeth13@users.noreply.github.com>
```

```
Date:   Sat Dec 4 22:48:44 2021 +0530

        Update Ast.ml

commit a4e8e0217b038351dca8f9d5831e807402185b03
Author: Sujeeth13 <72500256+Sujeeth13@users.noreply.github.com>
Date:   Sat Dec 4 22:48:09 2021 +0530

        Update Ast.ml

commit e65b0457a30aa26de2d45c199e68077fc6232a7a
Author: Sujeeth13 <72500256+Sujeeth13@users.noreply.github.com>
Date:   Sat Dec 4 22:45:46 2021 +0530

        Update CodeGeneration.ml

commit b5d46d678b0699d0ce9b38bb1efcd89b447d85ee
Author: Sujeeth13 <72500256+Sujeeth13@users.noreply.github.com>
Date:   Sat Dec 4 22:44:02 2021 +0530

        Update Parser.mly

commit f8fdb99c360da4a17d2ac2f587d4b8ece3fefc10
Author: Sujeeth13 <72500256+Sujeeth13@users.noreply.github.com>
Date:   Sat Dec 4 22:43:18 2021 +0530

        Update Parser.mly

commit d5a5cf58916a07d2ebd35067016dd369f96d770e
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 21:47:13 2021 +0530

        Update README.md

commit ab4d5643c3b0728802985f42170ccaa8de9f8143
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 21:46:27 2021 +0530

        Delete Init.txt

commit 9d73effada1d1d10f824a611b1292cb6914dc3cb
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
```

```
Date:   Sat Dec 4 21:46:18 2021 +0530

    Create README.md

commit b74e36212386f50a18513e84061e3d01737247a7
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 21:38:14 2021 +0530

    Add files via upload

commit 63688abc4c49ace0c93ba7556f20a685a0d91613
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 21:36:46 2021 +0530

    Create Init.txt

commit 22a063d25176ec02cb73b7a8072c432c488b2ed8
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 20:50:13 2021 +0530

    Update incorrect4.miche

commit be2f50e5a2228b33bf03710d7f156b020d34e277
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 20:48:50 2021 +0530

    Delete Init.txt

commit 4ecae910c5649481136ebbb3ac226a6fbb94f7b7
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 20:48:39 2021 +0530

    Delete Init.txt

commit 5b11a5b3c127163e613f034b20afc48d49ae2696
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 20:44:08 2021 +0530

    Update Installation.md

commit 91657bc8fda1519f960ecc3008c59bf253b1c2f1
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
```

```
Date:   Sat Dec 4 20:43:54 2021 +0530

        Update Installation.md

commit 567ae034bcdc898be45ad48df0ab7d435a7a0cce
Author: Praneeth-Nistala <74588016+Praneeth-Nistala@users.noreply.github.com>
Date:   Sat Dec 4 20:41:31 2021 +0530

        Update install.sh

commit 173b1e5ee55c8f250898f6a3877ff206666d4550
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 20:40:45 2021 +0530

        Update Installation.md

commit cc54b0e42d11bb7b3dcd92818fee63c20d4c30c6
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 20:40:23 2021 +0530

        Update Installation.md

commit 316ebf04be39f5c16a753e2f319cd5720a73a92f
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 20:29:19 2021 +0530

        Create install.sh

commit 96425e0071fd52166d75747eb15a825cced1adbe
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 20:29:03 2021 +0530

        Update Installation.md

commit 70d5b4ccbd6f7f73984cf5e858b22b3be9d362d2
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Sat Dec 4 20:28:19 2021 +0530

        Update Installation.md

commit e42b70f7c585069bac907aca71b30fc7509632c4
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
```

```
Date:   Fri Dec 3 21:49:02 2021 +0530

        Create CodeGeneration.ml

commit d0360eb475e74e1d1653a1c133b5d6778637c4b2
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Fri Dec 3 15:43:38 2021 +0530

        Create Installation.md

commit 001e7ef82ad436a02db514fdf0fe53ef70698a5f
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Nov 29 09:59:08 2021 +0530

        Delete Init.txt

commit 22f8b30e262b72c712661a6b75c2d678d03d8c01
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Nov 29 09:58:59 2021 +0530

        Create README.md

commit 80c0a0aef15dd65a42dff46e3f78d8cdc2976b9b
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Nov 29 09:58:47 2021 +0530

        Create Init.txt

commit 536e963e6d708cc12764d5c1a09000c5774e74cf
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Nov 29 09:58:30 2021 +0530

        Create Init.txt

commit 4e66520d7e0f977d5209dd783f7392340bc5ff84
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Nov 29 09:57:37 2021 +0530

        Delete Init.txt

commit 58a3efbda533f905767fb6f60b7453f25c364583
```

```
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Nov 29 09:13:17 2021 +0530

        Create README.md

commit e300b915113557578774343ca56befd5bdcc243d
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Nov 29 09:13:04 2021 +0530

        Create Init.txt

commit 3d75b9e3e37de7d1493f899b67763fa6f04efb6f
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Nov 29 09:11:15 2021 +0530

        Create Init.txt

commit ed3c29e7f8d54a693d40606ebc00da28e36944a4
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Thu Nov 25 10:09:58 2021 +0530

        Update FAQ.md

commit 4a60a42a6fc95b5bd71752a86860e474e8cd8c5e
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Thu Nov 25 10:09:06 2021 +0530

        Update FAQ.md

commit 9ef54f2a648e2eefc9086fe841cb04c21cd264e0
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Thu Nov 25 10:08:44 2021 +0530

        Update FAQ.md

commit fdf64ea2cd57818160d175257da954a36606a3e4
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Thu Nov 25 10:07:48 2021 +0530

        Update FAQ.md

commit 0cc93c2c28f7144525184d0fa4a18f33899ae542
```

Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Mon Nov 8 09:06:05 2021 +0530

    Create incorrect5.miche

commit cd0b072a58f658c7e08416710d61230aa5bea913
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Mon Nov 8 09:03:59 2021 +0530

    Update Ast.ml

commit 8aa9820b15a83118f3c97701a806878ccbff4e35
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Mon Nov 8 09:02:08 2021 +0530

    Update Semantic.ml

commit c3711590ae66f332c5949a7e239361182a56c61f
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Mon Nov 8 09:01:35 2021 +0530

    Update Parser.mly

commit 42998be88865aa08eed605dbf8d68229627d25d8
Author: G-Sidhardha <74396985+G-Sidhardha@users.noreply.github.com>
Date: Mon Nov 8 02:21:34 2021 +0530

    Delete Semantic_Analyser_Design.pdf

commit 41f75122ac8dfce7d5d69497f79fd785454604f8
Author: G-Sidhardha <74396985+G-Sidhardha@users.noreply.github.com>
Date: Mon Nov 8 02:20:42 2021 +0530

    Add files via upload

commit 0975da8abb0c5fca0d962f4bdd59c16866f7f2a5
Author: G-Sidhardha <74396985+G-Sidhardha@users.noreply.github.com>
Date: Mon Nov 8 02:17:35 2021 +0530

    Rename Video_links_final to Video_links_final.md

commit f0c09c28ceb0a03fc15d7f770bb6435941b8ec38

Author: G-Sidhardha <74396985+G-Sidhardha@users.noreply.github.com>
Date: Mon Nov 8 02:17:15 2021 +0530

    Create Video_links_final

commit 6fd5b0f9ab63a075013b355979f14d0fd82cd838
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Mon Nov 8 00:01:14 2021 +0530

    Rename Video_links.md to Video_links_(Nov 2).md

commit e24b58e1cd3776ac4c1cc0147b4a402044568dd7
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Sun Nov 7 23:59:44 2021 +0530

    Add files via upload

commit d44d979404f8d078765925b997aa8324537e50f0
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Sun Nov 7 23:58:17 2021 +0530

    Delete Semantic_Analyser_Design.pdf

commit 2f7608c326bd85047fa1cf5c7f8dbded1a52e69a
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date: Sun Nov 7 23:49:43 2021 +0530

    Add files via upload

commit d11e627e828f6ca7ded5578543270e5080a99b27
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date: Sun Nov 7 23:49:02 2021 +0530

    Delete Semantic_Analyser_Design.pdf

commit b93aac32c36f227f4cb2716fa8ef265eb3c5b023
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date: Sun Nov 7 23:48:28 2021 +0530

    Add files via upload

commit cff76c7a6f808e8c652f87cd7a7a7da3cd62432d

```
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Sun Nov 7 11:48:21 2021 +0530

        Add files via upload

commit 678f9dabca9244f2c6882f45b40304e76adb4b2e
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Sun Nov 7 11:47:02 2021 +0530

        Update FAQ.md

commit b123daab37ac07a90e64547a9429d454b1beb665
Author: G-Sidhardha <74396985+G-Sidhardha@users.noreply.github.com>
Date:   Wed Nov 3 01:08:44 2021 +0530

        Update Video_links.md

commit 7e185c557ad1fb72df7cb89550628a82a5cc0404
Author: chandra3000 <74546888+chandra3000@users.noreply.github.com>
Date:   Wed Nov 3 00:39:17 2021 +0530

        Create Video_links.md

commit de59e1668900b7ceaae5cd282172bdca341fded9
Author: chandra3000 <74546888+chandra3000@users.noreply.github.com>
Date:   Wed Nov 3 00:38:16 2021 +0530

        Update Video_links.md

commit 5ac636214c25dd770315a74a945634852b5fa15c
Author: chandra3000 <74546888+chandra3000@users.noreply.github.com>
Date:   Wed Nov 3 00:37:45 2021 +0530

        Rename Video links.md to Video_links.md

commit 926968e889e7a7df7162f371d9bceffbcc66312d
Author: chandra3000 <74546888+chandra3000@users.noreply.github.com>
Date:   Wed Nov 3 00:37:01 2021 +0530

        Update Video links.md
```

```
commit 0485e81c28c5e9f2e3e221b805d0a6f13b7d3495
Author: chandra3000 <74546888+chandra3000@users.noreply.github.com>
Date:   Wed Nov 3 00:34:04 2021 +0530

        Delete Init0.txt

commit c653095fd068abfbad12a58fdc761310bb54b421
Author: chandra3000 <74546888+chandra3000@users.noreply.github.com>
Date:   Wed Nov 3 00:33:34 2021 +0530

        Add files via upload

commit 83bbc1df521e08e6732862baaf5e28631003f31b
Author: chandra3000 <74546888+chandra3000@users.noreply.github.com>
Date:   Wed Nov 3 00:32:29 2021 +0530

        Create Init0.txt

commit cb27e6fa08b60effef01450eeb5dbed81aff70da
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Tue Nov 2 23:08:50 2021 +0530

        Update README.md

commit 5ca2303c0cfe3e7a2e107e63629287a2fdb7151c
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Tue Nov 2 23:01:52 2021 +0530

        Delete Semantic.ml

commit 7520cb95180c2045e66757de5cf581ff062fbb66
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Tue Nov 2 22:58:05 2021 +0530

        Update correct1.miche

commit d427215e1f536a4219a9ca0c6de7e14304d05965
Author: Praneeth-Nistala <74588016+Praneeth-Nistala@users.noreply.github.com>
Date:   Tue Nov 2 22:56:35 2021 +0530
```

```
        Update Ast.ml

commit b17c0686e7bb10e4cc261ad6072874231ba1318c
Author: Praneeth-Nistala <74588016+Praneeth-Nistala@users.noreply.github.com>
Date:   Tue Nov 2 22:56:04 2021 +0530

        Update Ast.ml

commit 375b648987055e642e4dd628ab6ee344f421a747
Author: Praneeth-Nistala <74588016+Praneeth-Nistala@users.noreply.github.com>
Date:   Tue Nov 2 22:55:20 2021 +0530

        Update Parser.mly

commit 9886d77c95a41423e0684a65fc32d310357b6972
Author: Praneeth-Nistala <74588016+Praneeth-Nistala@users.noreply.github.com>
Date:   Tue Nov 2 22:54:50 2021 +0530

        Update Parser.mly

commit f7c97994876fb090635f7fd66a818a57a010b9ac
Author: Praneeth-Nistala <74588016+Praneeth-Nistala@users.noreply.github.com>
Date:   Tue Nov 2 22:51:37 2021 +0530

        Update Semantic.ml

commit 5d311c1bc0c5f1c5743dcd752dfeeb4a10a56753
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Tue Nov 2 22:03:14 2021 +0530

        Update correct3.miche

commit d9b3e7d9a70fb9d92d24e0376706847b9fa13cea
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Tue Nov 2 21:51:12 2021 +0530

        Create correct1.miche

commit f3ab6f55e71cfb5a238585093348a24889511a73
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
```

```
Date:   Tue Nov 2 21:48:24 2021 +0530

        Update correct1.miche

commit ab55fda2401f56f2612cf887faf5494984c64990
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date:   Tue Nov 2 21:46:39 2021 +0530

        Update correct1.miche

commit 4a30be2f4b000ce61ac421fff0dd4b79afededf6
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Tue Nov 2 21:15:16 2021 +0530

        Delete Init.txt

commit f780133c04485496645252d5794d7d47721541fa
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Tue Nov 2 21:14:50 2021 +0530

        Create correct4.miche

commit cea04cb4d7bba94e0af8b47f6d5915c60d6b4086
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Tue Nov 2 21:14:39 2021 +0530

        Create correct3.miche

commit 055971848bc9915e3b10dcd489a599490b261c43
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Tue Nov 2 21:14:27 2021 +0530

        Create correct2.miche

commit 69f24e7f053482f57d9fa48774cfccef7e31a8c0
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Tue Nov 2 21:14:05 2021 +0530

        Create correct1.miche

commit 9d99ae89e280802fa09617fe9f242d3a148831a2
```

Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date: Tue Nov 2 20:53:12 2021 +0530

    Update incorrect1.miche

commit 0ec571accc6c2a84790f52030ea1d76fb623f581
Author: VEMULAPALLI-ADITYA
<74454067+VEMULAPALLI-ADITYA@users.noreply.github.com>
Date: Tue Nov 2 20:50:30 2021 +0530

    Update incorrect1.miche

commit cad8880dd4b87cd5815d5c76440b96e171984fc1
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Tue Nov 2 20:44:40 2021 +0530

    Create Ast.ml

commit 3080b82cd0ab5156eb38380806ec2b138ea86137
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Tue Nov 2 20:44:19 2021 +0530

    Update Parser.mly

commit 121aa6ff3ba8d3e7e87531867c6ec6a0564115a2
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Tue Nov 2 20:43:47 2021 +0530

    Create Ast.ml

commit a141e33f7bbb0542c6c8e2db6b22025db79cf7f2
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Tue Nov 2 20:43:26 2021 +0530

    Create Parser.mly

commit cfb7e090abdbab9b7a5472a6a4706416afc973c5
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date: Tue Nov 2 20:42:52 2021 +0530

    Update Semantic.ml

```
commit 365e2899dd6e02b52f2709e7e3aa47420d186e08
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Tue Nov 2 20:42:26 2021 +0530

        Delete Init.txt

commit a66e1f8ca6caa9122e7145b5c109120a0604fc4f
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Tue Nov 2 20:41:13 2021 +0530

        Delete Init2.txt

commit aef83733809e0514633778d15a2618f745e4ded0
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Tue Nov 2 20:40:22 2021 +0530

        Create Semantic.ml

commit f1624b27bf7df166690371efd519b3025991a1be
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Wed Oct 27 17:24:40 2021 +0530

        Update incorrect4.miche

commit 426d4930aa921866698631075f01017ce30947c6
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Mon Oct 25 20:26:27 2021 +0530

        Update incorrect3.miche

commit 1a1ac043bee39918b51355f8e27190bdd4fc7f0d
Author: Prathyush-1886 <82052242+Prathyush-1886@users.noreply.github.com>
Date:   Mon Oct 25 20:26:01 2021 +0530

        Update incorrect4.miche

commit 382618f23253b905c4b784e644f6eeec4b08a381
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Oct 25 10:50:33 2021 +0530

        Update README.md
```

```
commit 29c90f10ee273b516a95ae8d2a6034b3d1fb5258
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Oct 25 10:50:20 2021 +0530

        Delete Init.txt

commit 977987e1dfda377a66332a7692eaf7bff9b4a2ea
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Oct 25 10:50:12 2021 +0530

        Create README.md

commit d326b4207ce918845f6e448722abf3ae97cb97f3
Author: KrishnKher <72561577+KrishnKher@users.noreply.github.com>
Date:   Mon Oct 25 10:50:03 2021 +0530

        Create Semantic.ml
```

Code Listing and other files

All the files can be found on Michelin Github.

Bibliography

- https://www.iith.ac.in/~ramakrishna/Compilers-Aug14/
- https://ocaml.org/manual/lexyacc.html
- https://www.youtube.com/watch?v=GpPIzjJSWls
- https://www.javatpoint.com/lex
- https://youtu.be/54bo1qaHAfk
- https://youtu.be/_-wUHG2rfM
- https://github.com/jengelsma/lex-tutorial
- https://ocaml.org/learn/tutorials/up_and_running.html
- https://youtu.be/JTEwC3HihFc
- https://ocaml.org/manual/lexyacc.html#s%3Aocamllex-overview
- https://www.youtube.com/watch?v=kearNtiYWr8
- https://www.youtube.com/watch?v=3xCJMyawoxg

- http://www.cs.columbia.edu/~sedwards/classes/2014/w4115-fall/reports/
- https://ocaml.org/learn/tutorials/data_types_and_matching.html
- https://web.eecs.umich.edu/~weimerw/2016-ldi/pa4.html
- https://ocaml.org/learn/tutorials/lists.html
- https://releases.llvm.org/8.0.0/docs/tutorial/OCamlLangImpl3.html
- https://www.theseus.fi/bitstream/handle/10024/166119/Nguyen_Anh%20.pdf;jsessionid=05DF1C24F4695AF989E4E79A8305B460?sequence=2
- https://ocaml.org/learn/tutorials/hashtbl.html