CMPS109, Fall 16, Section 01

ADVANCED PROGRAMMING

UC SANTA CRUZ

# MIS | Phase 1 | Formal Report

**Objective: This is a programming project that is designed to let you utilize all the concepts and apply all the libraries, tools, and advanced programming techniques explained in class to build a middleware for the Machine Instruction Simulator (MIS), which is in the form of a virtual machine capable of executing an instruction set.**

Brian Nguyen
Froy Valencia

bnguye54ucsc.edu
frvalenc@ucsc.edu

## Design Decision

When collaborating on this assignment, we had to consider how to approach it in a modular, time efficient, and organized manner so it would present the best results. There are a lot of methods that were considered but each of their own have benefits and flaws with them. At a high level, we know that we need to use the STL containers from the library in order to maximize efficiency. Suppose we were to use arrays of a fixed size for the VARS and instructions. There would be a way to make this all work but it would be very difficult to change the code efficiently when you need to add or remove components from the code . In general, Overloading and templates to be able to pass generic objects or primitive types into our instruction methods. We were able to efficiently without having to know the object by creating operator overloads within the particular objects that would use a given function allowing us to get the primitive types from an object. We got over more of our design decision in the box to the right via our tools used and their application.

## Considered Concepts

A big consideration we wanted to make but decided against it was a hash table. Having a hashtable of the instructions after reading in lines of instruction from the txt file would be helpful in the way that we would have a key that we can point to. By using a node id to refer to line numbers and iterators to obtain vector values of the string of instructions, a hash table was of great consideration. The reason why we went with maps and templates was because we both knew the concept a little better and have practiced it more often in class. A disadvantage with hash tables would be the difficulty we face when coming across many collisions which would be very critical to us resulting in very poor performance. Another thing would be that a hash table would be really efficient if we had a very large amount of data we would work with, like a Records Store. Since we know that the instructions list given won't be that large, a map will work just fine.

## Tools & Application

Map - Maps were used many times in our program but was especially helpful when we were preloading the lines of instructions. The reason to preloading the instructions into a map would be so that we can refer to the map instead of the txt file when we need to JMP or refer back to certain values. The map that used an <int, vector<string>> for example, helped us load the line of string instructions into the vector<string>. This way, we can point to a particular element in that vector for what should be done. The int part of the map helps us determine which line number we are on. Having this will help us when we need to JMP in between chunks. We'll know where to restart or go back to when needed.

STD find - We used find in our Instruction class to particularly look through the instructions vector and compare it to the input string of a given input instruction. This would assure that the parameter being passed in matches the correct opcode prior to execution to avoid crash and error. It was used in a format of a range based for loop in a vector with .begin() and .end() values.

Iterators - This allows our program to be increased in it's general use and modulability. By pointing to the value of maps via key/value, we're able to access the information that we need directly. Iterators were used throughout our program due to it's overall usefulness and can be used with STL containers.

Templates - With most of the instructions being similar to each other like ADD, SUM, MUL, DIV, we needed a way to write concise code in a non redundant sense. We want to be working a lot with Generic programming, especially when working with this MIS project since we receiving input of an unknown type. The template can be used as a blueprint/model that would help create a generic class or function. Like I mentioned, this is definitely useful in our case in regards to the sixteen instructions we had to consider.

Inheritance - This was used often when we had to split up out VAR into Alphas and Numeric classes. VAR took a set of parameters that would be utilized in Numeric and Alpha but since Numeric handles numbers while Alpha handles alpha type, we split up using inheritance. It's important because there are certain methods that would only be used in either Numeric or Alpha. It's much more efficient and organized compared to piling all the VAR types into a single class that would be managing all the methods that it would correlate with.

ifstream - With ifstream, we were able to take in a txt file that would contain a list of instructions in the case that the user doesn't want to enter instruction by instruction and has a txt file which would contain it. We assumed that it would be much more efficient in the case a hypothetical client needed to run multiple lines of instruction. After loading in the file, we managed to handle the file with getLine() functions that would allows us to go through line by line to parse the instructions as needed.

Operator Overloading - This was used a lot throughout our program by allowing generic input of information to be worked with easily. In a case that we have an instruction line that requires us to modify a value, operator overloading would help adjust the type into a value that can be utilized without error.

CMPS109, Fall 16, Section 01
ADVANCED PROGRAMMING
UC SANTA CRUZ

# MIS | Phase 1 | Formal Report

Brian Nguyen

Froy Valencia

## Planning Process ,Execution Details & Approach

We attacked the problem first by evaluating what we need to make in chunks. The first chunk we handled were the VARS. With the VARS, we knew that we would have repeated chunks of code that slightly differ from each other. The operator overloads definetly helped make our project a lot cleaner. With templates and dividing the VARS into sub classes helped us figure out which classes will beusing what kind of methods.

After handling VARS, we would apply the same kind of mental approach when it came to the machine instructions. With the machine instructions, we listed out which instructions would be similiar, diffferent, and how they can be organized. At first we made multiple files that wouldn't have functional code but commented sections of what we want and expect to be put there. This would help everybody in the group understand what is going on. With the commented sections, we would figure out return types, parameters, and variables that would be neeeded. This was helpful for some of us that understand looking at code better than looking at a UML diagram.

At the top of our files, we've commented a brief description of the class' purpose. Within the code itself where it is appropriate, we have also applied the same brief description for more complex methods that might be harder to understand at a glimpse.

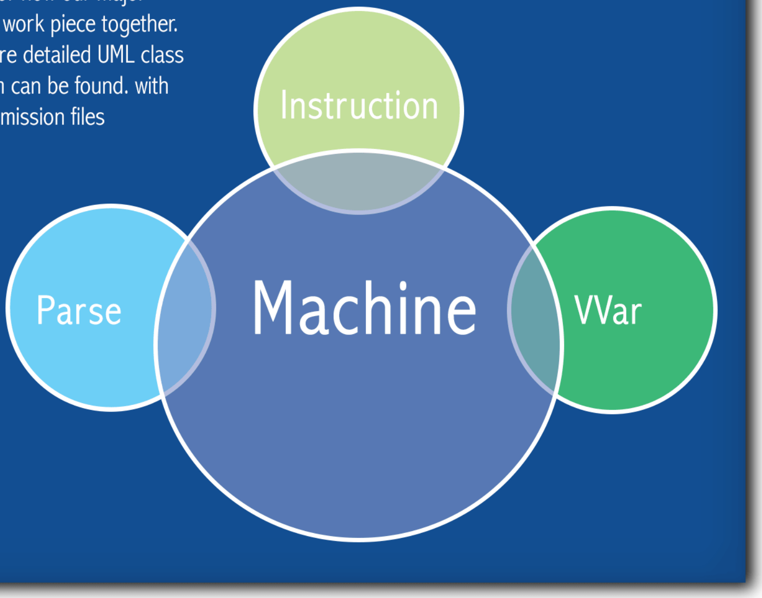
The commented out pseudo code classes were helpful to understand our approach but we slowly found a flaw being that it leads to compilation errors due to little symantic issues. It was tedious but not too problematic for the reward of understanding the program well.
To test our classes individually, we would make tester classes to see how the classes would behave prior to the big compliation with all the files. This was a good thing to do because it helped us narrow down any small kinks or issues that can be easily traced in a small test file.
To keep track of our progress, we used google hangouts, trello, and meetings to discuss what we should do and touch bases on what should be done. By practicing multiple communication methods, we are preparing for what is ahead of us after we graduate in the "real world". Unfortunaetly during our program development, we lost a group partner and were down to two people . This made the work:person ratio higher than we intially were prepared for. This would lead to us having us turn our assignment in late. Despite the lost, we will be able to still provide deliverables and files as needed.

## Classes in Phase 1

Alpha.cpp
Alpha.h
Char.cpp
Char.h
Client.cpp
Deliverables.pdf
Instruction.cpp
Instruction.h
Machine.cpp
Machine.h
Number.cpp
Number.h
Numeric.cpp
Numeric.h
Parser.cpp
Parser.h
Real.cpp
Real.h
String.cpp
String.h
VAR.cpp
VAR.h
makefile

## Basic Flow of MIS Classes

This shows the high level aspect of how our major classes work piece together. The more detailed UML class diagram can be found. with our submission files

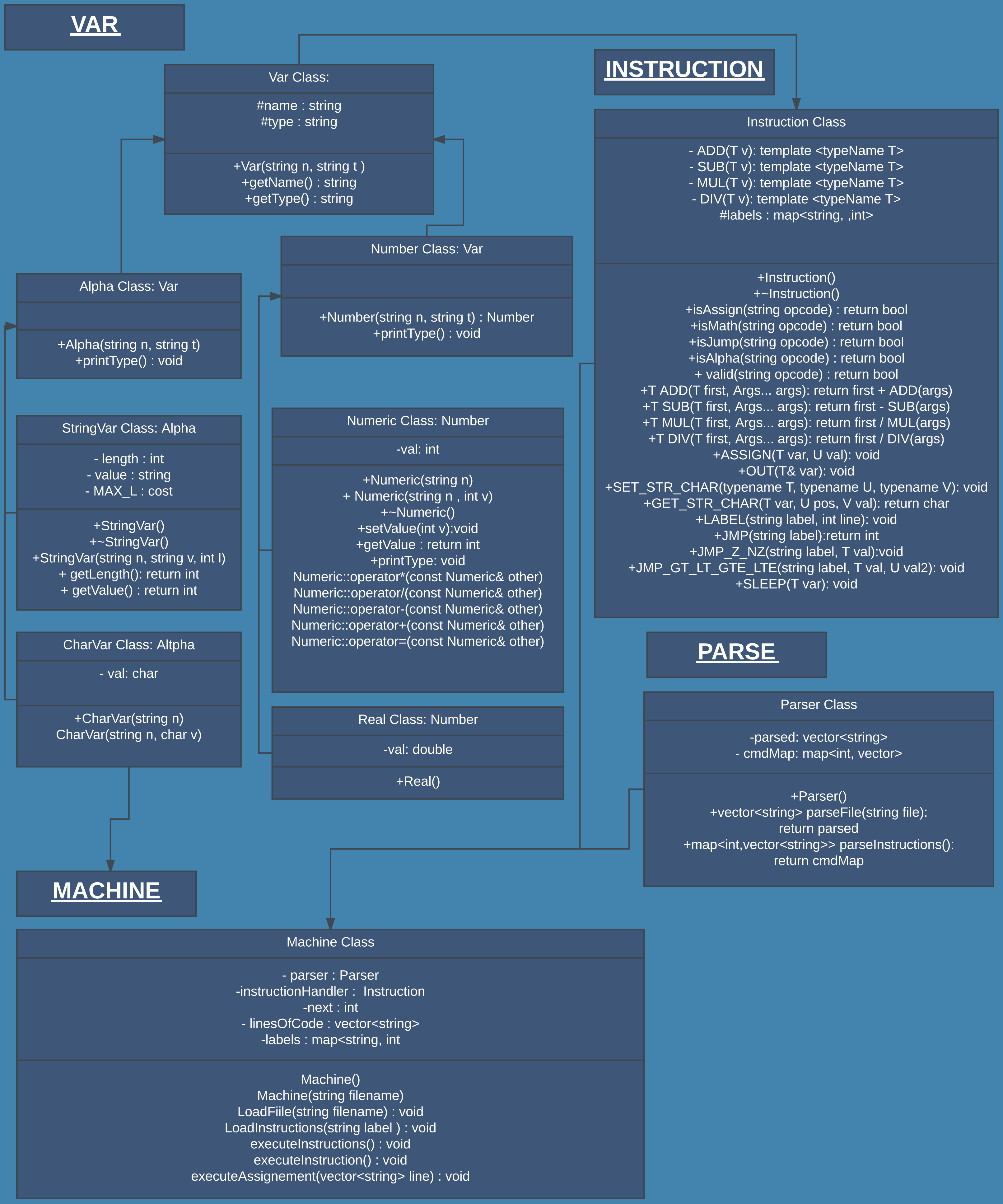


A big thanks to...

GitHub Trello Lucidchart

# MIS Class Diagram

Brian Nguyen & Froy Valencia | CMPS109 MIS Project



## VAR

Var Class:

#name : string
#type : string

+Var(string n, string t )
+getName() : string
+getType() : string

Alpha Class: Var

+Alpha(string n, string t)
+printType() : void

Number Class: Var

+Number(string n, string t) : Number
+printType() : void

StringVar Class: Alpha

- length : int
- value : string
- MAX_L : cost

+StringVar()
+~StringVar()
+StringVar(string n, string v, int l)
+ getLength(): return int
+ getValue() : return int

CharVar Class: Altpha

- val: char

+CharVar(string n)
CharVar(string n, char v)

Numeric Class: Number

-val: int

+Numeric(string n)
+ Numeric(string n , int v)
+~Numeric()
+setValue(int v):void
+getValue : return int
+printType: void
Numeric::operator*(const Numeric& other)
Numeric::operator/(const Numeric& other)
Numeric::operator-(const Numeric& other)
Numeric::operator+(const Numeric& other)
Numeric::operator=(const Numeric& other)

Real Class: Number

-val: double

+Real()

## INSTRUCTION

Instruction Class

- ADD(T v): template <typeName T>
- SUB(T v): template <typeName T>
- MUL(T v): template <typeName T>
- DIV(T v): template <typeName T>
#labels : map<string, ,int>

+Instruction()
+~Instruction()
+isAssign(string opcode) : return bool
+isMath(string opcode) : return bool
+isJump(string opcode) : return bool
+isAlpha(string opcode) : return bool
+ valid(string opcode) : return bool
+T ADD(T first, Args... args): return first + ADD(args)
+T SUB(T first, Args... args): return first - SUB(args)
+T MUL(T first, Args... args): return first / MUL(args)
+T DIV(T first, Args... args): return first / DIV(args)
+ASSIGN(T var, U val): void
+OUT(T& var): void
+SET_STR_CHAR(typename T, typename U, typename V): void
+GET_STR_CHAR(T var, U pos, V val): return char
+LABEL(string label, int line): void
+JMP(string label):return int
+JMP_Z_NZ(string label, T val):void
+JMP_GT_LT_GTE_LTE(string label, T val, U val2): void
+SLEEP(T var): void

## PARSE

Parser Class

-parsed: vector<string>
- cmdMap: map<int, vector>

+Parser()
+vector<string> parseFile(string file):
return parsed
+map<int,vector<string>> parseInstructions():
return cmdMap

## MACHINE

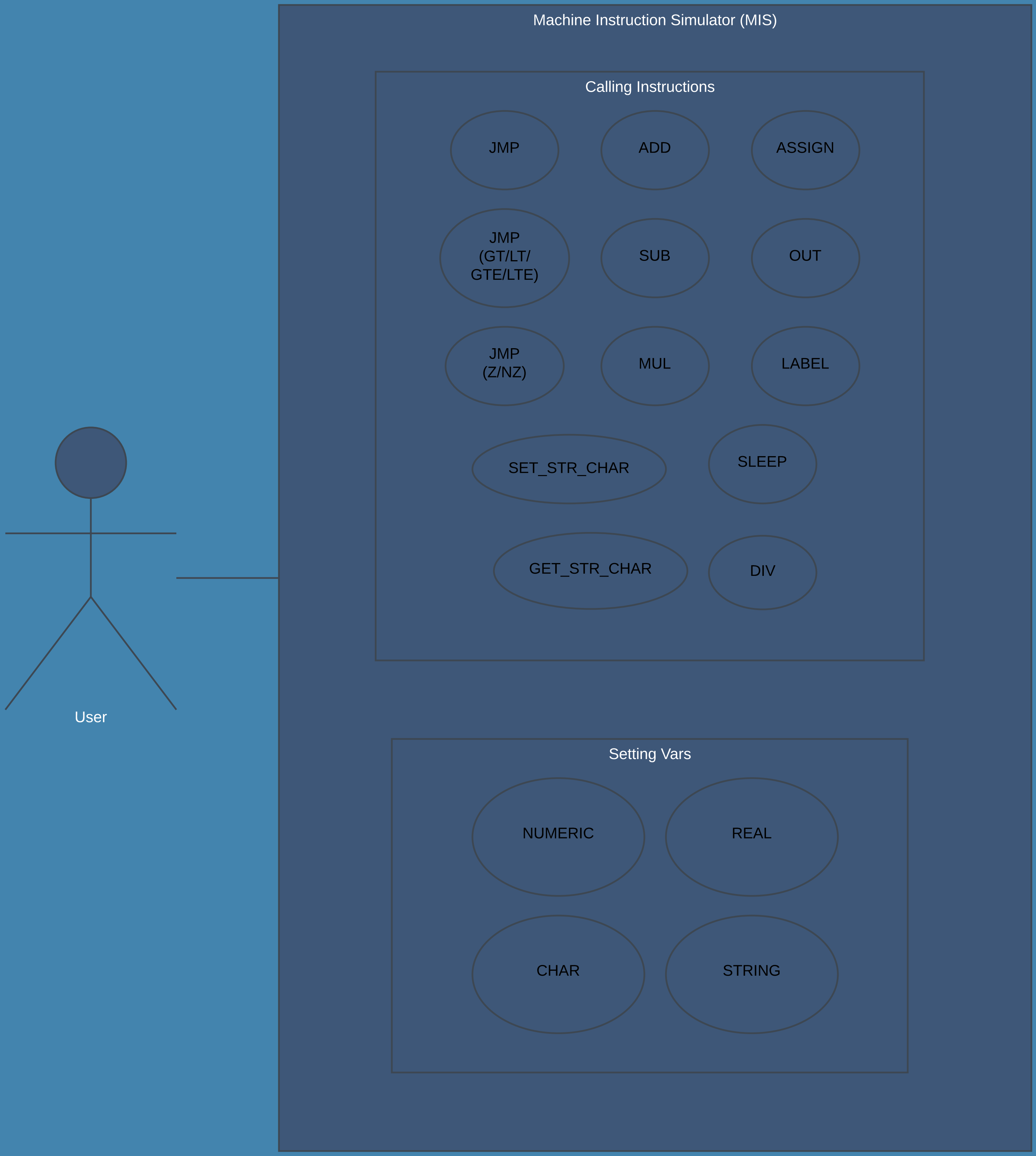
Machine Class

- parser : Parser
-instructionHandler : Instruction
-next : int
- linesOfCode : vector<string>
-labels : map<string, int

Machine()
Machine(string filename)
LoadFiile(string filename) : void
LoadInstructions(string label ) : void
executeInstructions() : void
executeInstruction() : void
executeAssignement(vector<string> line) : void

# MIS Use Case Diagram

Brian Nguyen & Froy Valencia | CMPS109 MIS Project



Machine Instruction Simulator (MIS)
Calling Instructions
JMP
ADD
ASSIGN
JMP (GT/LT/ GTE/LTE)
SUB
OUT
JMP (Z/NZ)
MUL
LABEL
SET_STR_CHAR
SLEEP
GET_STR_CHAR
DIV
User
Setting Vars
NUMERIC
REAL
CHAR
STRING