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

MIS | Phase 2 | Formal Report

Objective: Build a client/server model for your MIS virtual machine. You wil be spliting your virtual machines into a client and server programs.

Brian Nguyen
Froy Valencia

bnguye54ucsc.edu
frvalenc@ucsc.edu

The Reworks of Phase 1 | Then & Now.

Initially for phase 1, we worked hard towards making a working MIS simulator that would ideally run a txt file of commands however, we were unable to complete the task in time. With UML diagrams, a formal report, and multiple files towards our implementation, we tried to best describe our process via comments and explanation through the deliverables.

Our original thought process in Phase 1 involved organization of the methods that would be used by particular objects. For example, we knew that DIV label would not be applied to strings so we defined Math operators like these under LABELS that would use them. This was a flaw in the first thought process of our design so we changed it in order for classes to be divided in a more appropriate manner. We looked into the similar way the Geometric objects were made, defined, and used in the class example slides that were provided to us. Using this, we first implemented this organization of methods to our math operator Labels being ADD,MUL,DIV,SUB. Since these were most similar to the geometric objects in the way that they have similar return types, it help us understand how we should implement our other methods.

We ended up making more classes than we initially did in our phase 1 because we rewrote how the organization of our Machine class would be. Machine would have an executeAll method that would ideally read in a txt file and with the help of our other classes like Parser, Instruction, and VAR.

Initially in Phase 1, the organization of our instructions had subclasses in areas we thought would best fit them. In our newer version, the Instruction class is our Base Abstract Class where each Label gets it's own class under instruction. The same application is applied to VAR with String, Char, Numeric, and Real where VAR is also a Base Abstract Class.

Server and Client | Putting it Together.

In this phase, we are asked to learn application of multithreading via the MIS machine we made in phase 1. Since we didn't completely finish phase 1 the first time around, as described in the reworks of Phase 1, it was rewritten, twice. With this new way that we wrote the MIS machine, we were able to work independently on files a lot easier. To each of our own, but together, one of us worked on server/client side while the other tested and reworked phase 1 to make sure our machine would work when it came to the time that server and client worked.

At first understanding how this whole client/server situation worked was a huge confusing game of chasing our tail however after a lot of reading the slides, pulling code from the slides to play with, reading textbooks, and our internet provider, we were able to slowly get a better understanding of what was asked of us. Is our implementation of client/server with MIS machine completely accurate to the tee and should be used professionally? I wouldn't recommend it but we feel confident in it.

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

MIS | Phase 2 | Formal Report

Objective: Build a client/server model for your MIS virtual machine. You wil be spliting your virtual machines into a client and server programs.

Brian Nguyen
Froy Valencia

bnguye54ucsc.edu
frvalenc@ucsc.edu

Server and Client | Putting it Together (contd)

When planning our phase 2, we wanted to be able to have it so that when calling client in g++ command line, you can include a textile as an argument so that multiple clients can be called from that line. Nobody wants to open up a cpp file and edit how many clients will go into it… So by calling the client 'n' times being that n is the amount of clients you want to run, it will take in the first argument as a txtfile string, 2nd as a port, and last as an ip.

The reason for our implementation of port and ip address outside of localhost is because we want it to be modular in the way that we are given the case of needing to test it over different computers with multiple ports. We weren't sure what kind of testing would be involved so we both that that it would be a good idea to include this as a feature as it might nearly be a need considering the fact that we are working with networking.

With our client taking in a txtString, it would send that txtfile name to server after having established a connection. This would be executed through a thread and the thread would lead it to the server where a machine pointer object is called with the txtfile name. There, the machine object will invoke an execute all method where it will handle the file accordingly.

In the case we have multiple clients going. There can be 2 txtfiles going without them conflicting since we have multiple threads going on, each locking and unlocking mutex's independently to their own need. Our server would infinitely have socket listeners that would wait for more client items to be loaded until the user exits the server by typing cntrl +c.

README.md | How to build

README.md

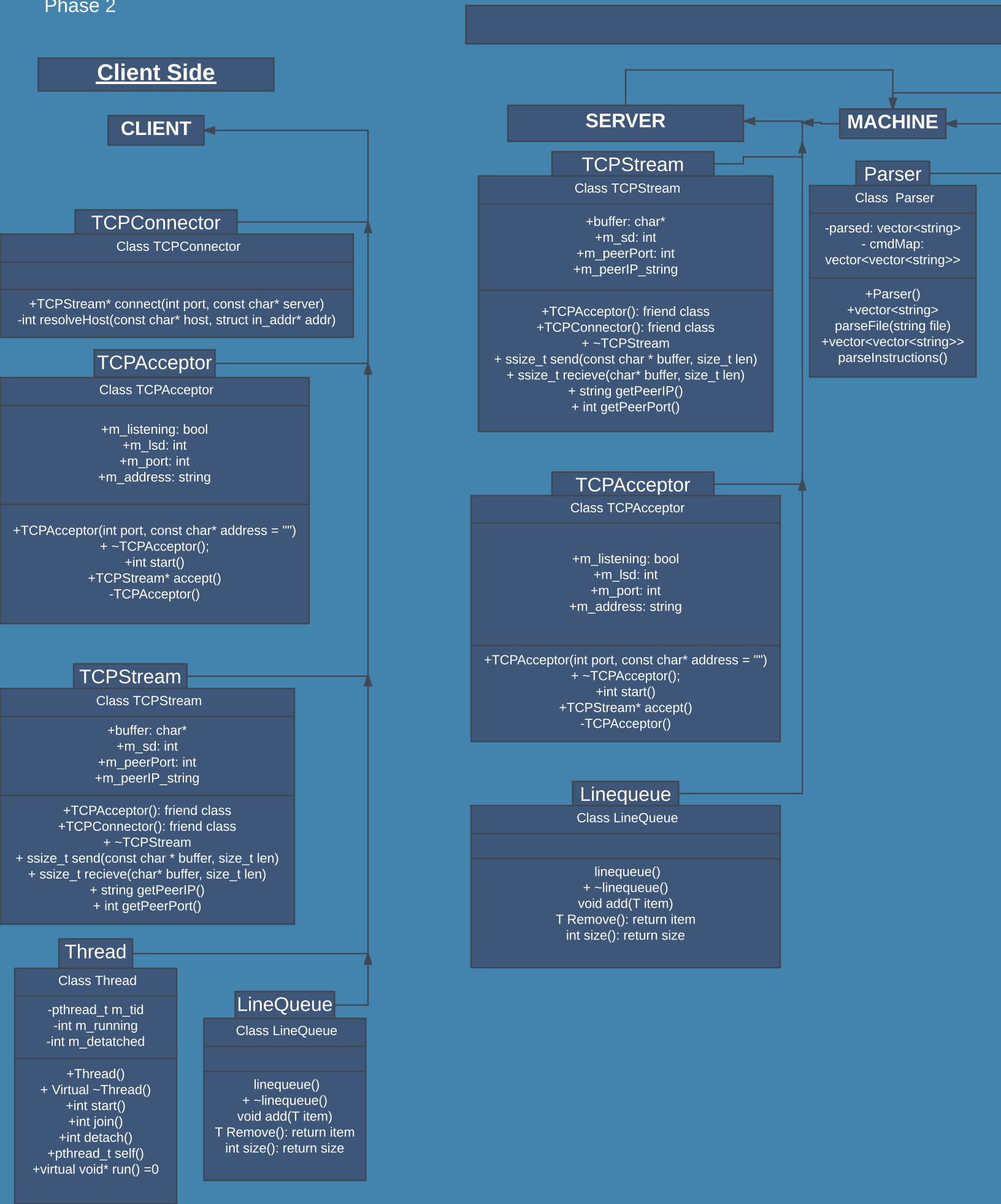
```
#How to build Server/Client

1.) Open 2 seperate terminal windows.
    - 1 for Server, cd to ServerSide folder
    - 1 for Clients, cd to Clientside Folder
2.) Run Server first:
    - $./server <port> <ip Address>
    - ie:  $./server 9999 localhost
3.) Run Client(s) following it:
    - For the amount of clients you want to run, do:
        - $./client <txtfile name> <port> <ip Address>
        - ie for 1 client:  $./client varTxt.txt 9999 localhost
        - ie for multiple clients:
            -$./client varTxt.txt 9999 localhost; ./client varTxt2.txt 9999 localhost; ./client varTxt3.txt 9999
            localhost
4.) To exit Server, do Cntrl+ C to exit.
```

MIS Class Diagram

Phase 2

Brian Nguyen & Froy Valencia | CMPS109 MIS Project



ct

Server Side

INSTR

Jump

Class Jump

Jump();
virtual ~Jump();
virtual void execute(Data * d,vector<string> line);
virtual Instruction * clone();

Add

Class Add

+ Add();
+virtual ~Add();
+ virtual void execute(Data *d, +vector<string> line);
+ virtual Instruction * clone();

Data

Class Data

- std::map<std::string,VAR *> varMap;
- std::map<std::string,int> labelMap;
- int current;

+ Data()
+ int getCurrent(): return current;
+ void setCurrent(int i)
+void addVar(string name, VAR * v)
+VAR * getVar(string name): return varMap[name];
+ void addLabel(string label, int i)
+int getLabel(string label): return labelMap[label];

JumpGT

Class JumpG

JumpGTE()
virtual ~JumpG
virtual void execute(Data * d,
virtual Instruction *

Char

Class Char

-value: char

+Char();
+ Char(std::string n, char v);
+ ~Char();
+ void initialize(vector<string> line);
+ VAR * clone(vector<string> line);
+ char getValue() const;
+ void setValue(char c);
+ friend std::ostream& operator<<(std::ostream& os, + const Char& var);

JumpGT

Class JumpGT

JumpGT();
virtual ~JumpGT();
virtual void execute(Data * d,vector<string> line);
virtual Instruction * clone();

Jumpl

Class Jum

JumpLT
virtual ~Jum
virtual void execute(Data *
virtual Instructio

Real

Class Real

double value;

Real();
Real(std::string n, double v);
virtual ~Real();
virtual void initialize (vector<string> line);
VAR * clone (vector<string> line);
void setValue(double v);
double getValue() const;
Real operator*(const Real& other);
Real operator/(const Real& other);
Real operator-(const Real& other);
Real operator+(const Real& other);
Real& operator=(const Real& other);
Real& operator=(const int& n);
Real& operator+=(const Real& other);
Real& operator+=(const int& i);
Real& operator+=(const double& d);
Real& operator+=(const Numeric& num);
Real& operator*=(const Real& other);
Real& operator*=(const int& i);
Real& operator*=(const double& d);
Real& operator*=(const Numeric& num);
friend std::ostream& operator<<(std::ostream& os, const Real& var);

SET_STR_CHAR

Class SET_STR_CHAR

SET_STR_CHAR();
virtual ~SET_STR_CHAR();
virtual void execute(Data * d,vector<string> line);
virtual Instruction * clone();

JumpL

Class Jum

JumpLTE
virtual ~Jum
virtual void execute(Data *
virtual Instructio

GET_STR_CHAR

Class GET_STR_CHAR

GET_STR_CHAR();
virtual ~GET_STR_CHAR();
virtual void execute(Data *d, vector<string> line);
virtual Instruction * clone();

JumpN

Class Jum

JumpNZ
virtual ~Jum
virtual void execute(Data *
virtual Instructio

Div

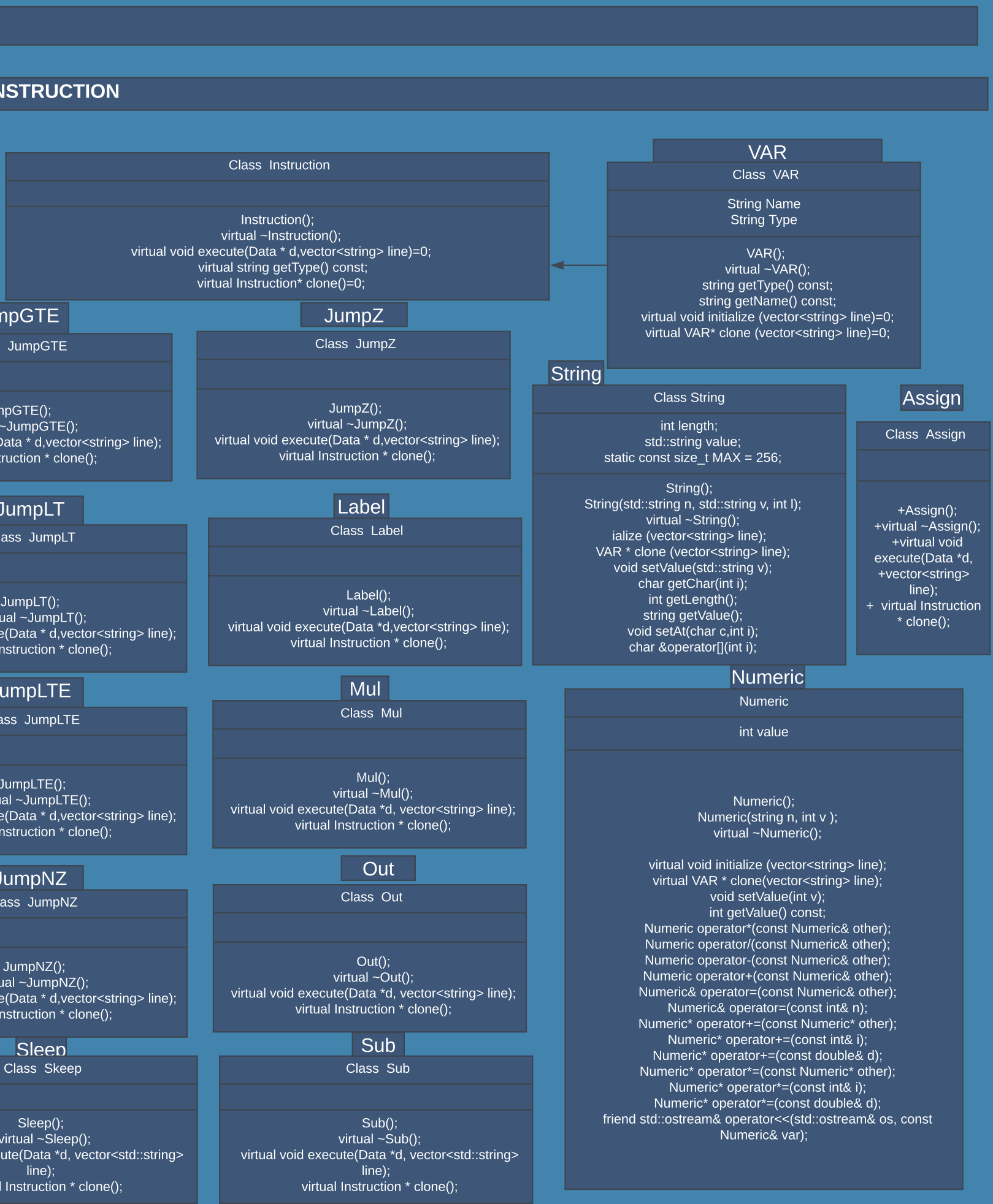
Class Div

+ Div();
+ virtual ~Div();
+virtual void execute(Data *d, vector<string> line);
+ virtual Instruction * clone();

Slee

Class S

Sleep
virtual ~S
virtual void execute(Data
line
virtual Instruct


NSTRUCTION
Class Instruction
Instruction();
virtual ~Instruction();
virtual void execute(Data * d,vector<string> line)=0;
virtual string getType() const;
virtual Instruction* clone()=0;
VAR
Class VAR
String Name
String Type
VAR();
virtual ~VAR();
string getType() const;
string getName() const;
virtual void initialize (vector<string> line)=0;
virtual VAR* clone (vector<string> line)=0;
mpGTE
JumpGTE
npGTE();
~JumpGTE();
Data * d,vector<string> line);
truction * clone();
JumpZ
Class JumpZ
JumpZ();
virtual ~JumpZ();
virtual void execute(Data * d,vector<string> line);
virtual Instruction * clone();
String
Class String
int length;
std::string value;
static const size_t MAX = 256;
String();
String(std::string n, std::string v, int l);
virtual ~String();
ialize (vector<string> line);
VAR * clone (vector<string> line);
void setValue(std::string v);
char getChar(int i);
int getLength();
string getValue();
void setAt(char c,int i);
char &operator[](int i);
Assign
Class Assign
+Assign();
+virtual ~Assign();
+virtual void
execute(Data *d,
+vector<string>
line);
+ virtual Instruction
* clone();
JumpLT
ass JumpLT
JumpLT();
ual ~JumpLT();
e(Data * d,vector<string> line);
nstruction * clone();
Label
Class Label
Label();
virtual ~Label();
virtual void execute(Data *d,vector<string> line);
virtual Instruction * clone();
umpLTE
ass JumpLTE
JumpLTE();
ual ~JumpLTE();
e(Data * d,vector<string> line);
nstruction * clone();
Mul
Class Mul
Mul();
virtual ~Mul();
virtual void execute(Data *d, vector<string> line);
virtual Instruction * clone();
Numeric
Numeric
int value
Numeric();
Numeric(string n, int v);
virtual ~Numeric();
virtual void initialize (vector<string> line);
virtual VAR * clone(vector<string> line);
void setValue(int v);
int getValue() const;
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);
Numeric& operator=(const int& n);
Numeric* operator+=(const Numeric* other);
Numeric* operator+=(const int& i);
Numeric* operator+=(const double& d);
Numeric* operator*=(const Numeric* other);
Numeric* operator*=(const int& i);
Numeric* operator*=(const double& d);
friend std::ostream& operator<<(std::ostream& os, const
Numeric& var);
JumpNZ
ass JumpNZ
JumpNZ();
ual ~JumpNZ();
e(Data * d,vector<string> line);
nstruction * clone();
Out
Class Out
Out();
virtual ~Out();
virtual void execute(Data *d, vector<string> line);
virtual Instruction * clone();
Sleep
Class Skeep
Sleep();
virtual ~Sleep();
ute(Data *d, vector<std::string>
line);
Instruction * clone();
Sub
Class Sub
Sub();
virtual ~Sub();
virtual void execute(Data *d, vector<std::string>
line);
virtual Instruction * clone();