Project

Poithon

Programming Language used : [C++]

Group Number **14** Members:

1) Tushar Kumar (IIT2017129)
2) Kaustubh Chitravanshi (IIT2017136)
3) Raghav Dhupar (IIT2017121)
4) Heimal Shukla (IIT2017104)
5) Ritesh Yadav (IRM2017001)

Introduction:-

Poithon is a compiler for the python based programming language.This program aims to be an experiment with Flex, Bison, and Abstract Syntax Trees, rather than any sort of Python replacement. It's intended to support the following features while keeping the code as small and simple as possible

1. Basic manipulation
2. if statement
3. if-else statement
4. while statements

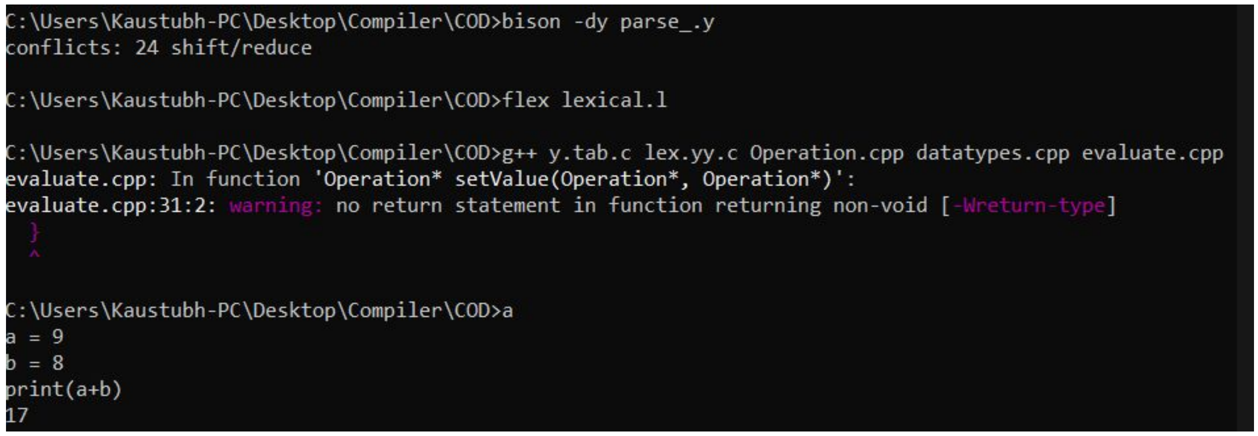
Different return codes for semantic/syntactic errors.

The interpreter will return the following values:

1. Warning
2. Syntax error
3. Semantic error
4. Other error (ex: source file not found)
5. Runtime error such as undefined variable.

The compiler is able to compile itself. You can see its code both as an implementation.

On the terminal hit the following commands as shown:



```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>bison -dy parse_.y
conflicts: 24 shift/reduce

C:\Users\Kaustubh-PC\Desktop\Compiler\COD>flex lexical.l

C:\Users\Kaustubh-PC\Desktop\Compiler\COD>g++ y.tab.c lex.yy.c Operation.cpp datatypes.cpp evaluate.cpp
evaluate.cpp: In function 'Operation* setValue(Operation*, Operation*)':
evaluate.cpp:31:2: warning: no return statement in function returning non-void [-Wreturn-type]
  }
  ^

C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
a = 9
b = 8
print(a+b)
17
```

As you can see, the compilation warning is of the class c++. Since code is written with love in c++ . Now as the compilation is done we get a file named a.exe in windows and a.out in unix.

After executing that file , the terminal observes a shell on which one can write the acceptable functionality.

Poithon source code is carefully written to be as concise and easy-to-read as possible, so that the source code becomes good study material to learn about various techniques used in compilers. You may find the lexer and the parser are already useful to learn how poithon’s source code is .

Code:-

DATA TYPES (*CPP FILE*):

```
#include <bits/stdc++.h>
#include "datatypes.h"
#include "Operation.h"
using namespace std;


extern int threeadd;
Arithmetic* Arithmetic::findValue(string x, Arithmetic* b)
{
	if(x[0] == '+')
		return add(b);
	else if(x[0] == '/')
		return divide(b);
	else if(x[0] == '%')
		return mod(b);
	else if(x[0] == '*')
		return multiply(b);
	else if(x[0] == '-')
		return subtract(b);
	else if(x[0] == '<')
		return lessthan(b);
	else if(x[0] == '>')
		return greaterthan(b);
	else if(x == "==")
		return equalto(b);
	else if(x == "<=")
	{
		Integer* x = new Integer(((dynamic_cast<Integer*> (lessthan(b)))->value) &&
((dynamic_cast<Integer*> (equalto(b)))->value));
		return x;
	}
	else if(x == ">=")
	{
```

```
                Integer* x = new Integer(((dynamic_cast<Integer*> (greaterthan(b)))->value)
&& ((dynamic_cast<Integer*> (equalto(b)))->value));
                return x;
        }
        else if(x == "!=")
        {
                Integer* x = new Integer((!(dynamic_cast<Integer*> (equalto(b)))->value));
                return x;
        }
        return NULL;
}

Operation* Arithmetic::perform()
{
        return this;
}

string Integer::print()
{
        string x = to_string(value);
        return x;
}

Arithmetic* Integer::add(Arithmetic* a)
{
        Integer* x = new Integer(value + ((dynamic_cast<Integer*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Integer::subtract(Arithmetic* a)
{
        Integer* x = new Integer(value - ((dynamic_cast<Integer*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Integer::multiply(Arithmetic* a)
{
        Integer* x = new Integer(value * ((dynamic_cast<Integer*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Integer::divide(Arithmetic* a)
{
        Integer* x = new Integer(value / ((dynamic_cast<Integer*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Integer::mod(Arithmetic* a)
{
        Integer* x = new Integer(value%((dynamic_cast<Integer*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Integer::lessthan(Arithmetic* a)
{
        Integer* x = new Integer((value < ((dynamic_cast<Integer*> (a))->value))?1:0);
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Integer::greaterthan(Arithmetic* a)
{
        Integer* x = new Integer((value > ((dynamic_cast<Integer*> (a))->value))?1:0);
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Integer::equalto(Arithmetic* a)
{
        Integer* x = new Integer((value == ((dynamic_cast<Integer*> (a))->value))?1:0);
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

string Float::print()
{
        string x = to_string(value);
        return x;
}

Arithmetic* Float::add(Arithmetic* a)
{
        Float* x = new Float(value + ((dynamic_cast<Float*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Float::subtract(Arithmetic* a)
{
        Float* x = new Float(value - ((dynamic_cast<Float*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}
Arithmetic* Float::multiply(Arithmetic* a)
{
        Float* x = new Float(value * ((dynamic_cast<Float*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Float::divide(Arithmetic* a)
{
        Float* x = new Float(value / ((dynamic_cast<Float*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Float::mod(Arithmetic* a)
{
        Float* x = new Float();
        return x;
}

Arithmetic* Float::lessthan(Arithmetic* a)
{
        Integer* x = new Integer((value < ((dynamic_cast<Integer*> (a))->value))?1:0);
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Float::greaterthan(Arithmetic* a)
{
        Integer* x = new Integer((value > ((dynamic_cast<Integer*> (a))->value))?1:0);
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}

Arithmetic* Float::equalto(Arithmetic* a)
{
        Integer* x = new Integer((value == ((dynamic_cast<Integer*> (a))->value))?1:0);
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}


string String::print()
{
        string x = (value);
        return x;
}

Arithmetic* String::add(Arithmetic* a)
{
        String* x = new String(value + ((dynamic_cast<String*> (a))->value));
        x->name = "t" + to_string(threeadd);
        threeadd++;
        return x;
}
```

```
Arithmetic* String::subtract(Arithmetic* a)
{
        throw "No such operation allowed";
        String* x = new String();
        return x;
}

Arithmetic* String::multiply(Arithmetic* a)
{
        int l = (dynamic_cast<Integer*> (a))->value;
        string str;
        for(int i = 0; i < l; i++)
        str = str + value;
        return new String(str);
}

Arithmetic* String::divide(Arithmetic* a)
{
        String* x = new String();
        return x;
}

Arithmetic* String::mod(Arithmetic* a)
{
        String* x = new String();
        return x;
}


Arithmetic* String::lessthan(Arithmetic* a)
{
        Integer* x = new Integer((value < ((dynamic_cast<String*> (a))->value))?1:0);
        return x;
}

Arithmetic* String::greaterthan(Arithmetic* a)
{
        Integer* x = new Integer((value > ((dynamic_cast<String*> (a))->value))?1:0);
        return x;
}
```

```
Arithmetic* String::equalto(Arithmetic* a)
{
        Integer* x = new Integer((value == ((dynamic_cast<String*> (a))->value))?1:0);
        return x;
}
```

DATA TYPES (*HEADER FILE*)

```
#ifndef DATA_MY
#define DATA_MY
#include "Operation.h"
using namespace std;
class Integer;
class Float;
class String;
class Operation;
extern int threeadd;
class Arithmetic: public Operation
{

        public:
        Arithmetic(){}
        ~Arithmetic(){delete A;}
        Arithmetic(string t, Arithmetic* x)
        {
                type = t;
                A = x;
        }

        string type;
        Arithmetic* A;
        string name;
        Operation* perform();
        Arithmetic* findValue(string x, Arithmetic* b);
```

```
	virtual Arithmetic* add(Arithmetic* x) = 0;
	virtual Arithmetic* divide(Arithmetic* x) = 0;
	virtual Arithmetic* mod(Arithmetic* x) = 0;
	virtual Arithmetic* multiply(Arithmetic* x) = 0;
	virtual Arithmetic* subtract(Arithmetic* x) = 0;
	virtual Arithmetic* lessthan(Arithmetic* x) = 0;
	virtual Arithmetic* greaterthan(Arithmetic* x) = 0;
	virtual Arithmetic* equalto(Arithmetic* x) = 0;
	virtual string print() = 0;
};

class Integer : public Arithmetic
{
	public:
	Integer(){}
	Integer(int x, string nm = "")
	{
		type = "integer";
		A = this;
		value = x;
		if (nm == "")
		{
			nm = "t" + to_string(threeadd);
			threeadd++;
		}
		name = nm;
	}

	int value;
	Arithmetic* add(Arithmetic* a);
	Arithmetic* subtract(Arithmetic* a);
	Arithmetic* multiply(Arithmetic* a);
	Arithmetic* divide(Arithmetic* a);
	Arithmetic* mod(Arithmetic* a);
	Arithmetic* lessthan(Arithmetic* a);
	Arithmetic* greaterthan(Arithmetic* a);
	Arithmetic* equalto(Arithmetic* a);
	string print();
};

class Float : public Arithmetic
```

```
{
	public:
	Float(){}
	Float(double x, string nm = "")
	{
		type = "float";
		A = this;
		value = x;
		if (nm == "")
		{
			nm = "t" + to_string(threeadd);
			threeadd++;
		}
	name = nm;
	}

	double value;
	Arithmetic* add(Arithmetic* a);
	Arithmetic* subtract(Arithmetic* a);
	Arithmetic* multiply(Arithmetic* a);
	Arithmetic* divide(Arithmetic* a);
	Arithmetic* mod(Arithmetic* a);
	Arithmetic* lessthan(Arithmetic* a);
	Arithmetic* greaterthan(Arithmetic* a);
	Arithmetic* equalto(Arithmetic* a);
	string print();
};

class String : public Arithmetic
{
	public:
	String(){}
	String(string x, string nm = "")
	{
		type = "string";
		A = this;
		value = x;
		if (nm == "")
		{
			nm = "t" + to_string(threeadd);
			threeadd++;
		}
```

```
                name = nm;
        }

        string value;
        Arithmetic* add(Arithmetic* a);
        Arithmetic* subtract(Arithmetic* a);
        Arithmetic* multiply(Arithmetic* a);
        Arithmetic* divide(Arithmetic* a);
        Arithmetic* mod(Arithmetic* a);
        Arithmetic* lessthan(Arithmetic* a);
        Arithmetic* greaterthan(Arithmetic* a);
        Arithmetic* equalto(Arithmetic* a);
        string print();
};

#endif
```

EVALUATE (*CPP FILE*)

```
#include <bits/stdc++.h>
#include "evaluate.h"
#include "Operation.h"
#include "datatypes.h"

extern std::map<int, std::map<string, Arithmetic*> > symboltable;

Operation* findValue(Operation* a, string x, Operation* b)
{
        Arithmetic* xx = dynamic_cast<Arithmetic*>(a);
        Arithmetic* yy = dynamic_cast<Arithmetic*>(b);
        return xx->findValue(x, yy);
}

Operation* setValue(Operation* a, Operation *b)
{
        Arithmetic * bb = dynamic_cast<Arithmetic*>(b);
        if(bb->type == "integer")
        {
                Integer *c = new Integer(((dynamic_cast<Integer*> (bb))->value));
                a= c;
        }
        else if(bb->type == "float")
        {
                a = new Float(((dynamic_cast<Float*> (bb))->value));
        }
        else
        {
                String *c = new String(((dynamic_cast<String*> (bb))->value));
                a=c;
        }
        return a;
 }
```

EVALUATE (*HEADER FILE*)

```
#ifndef EVAL_MY
#define EVAL_MY
#include "datatypes.h"

using namespace std;
class Operation;
class Arithmetic;
Operation* findValue(Operation* a, string x, Operation* b);

Operation* setValue(Operation* a, Operation *b);

#endif
```

LEXICAL (*.L FILE*)

```
%option noyywrap
%option yylineno

%{
#include <bits/stdc++.h>
#include "evaluate.h"
#include "y.tab.h"

using namespace std;

extern void yyerror(const char*, char);
%}

ident [_a-zA-Z][_0-9a-zA-Z]*
int   [0-9]+
float [0-9]*\.[0-9]+
string "\""[^"\n]*"\""
```

```
%%

"#"[^\r\n]* { /* comments */    }
[ \t]         { /* whitespace */ }
[\n]          { return yytext[0]; }
"+"           { return yytext[0]; }
"-"           { return yytext[0]; }
"*"           { return yytext[0]; }
"/"           { return yytext[0]; }
"%"           { return yytext[0]; }
"^"           { return yytext[0]; }
"("           { return yytext[0]; }
")"           { return yytext[0]; }
"="           { return yytext[0]; }
":"           { return yytext[0]; }
","           { return yytext[0]; }
"<"           { return yytext[0]; }
">"           { return yytext[0]; }
"+="          { return PLUSEQ;    }
"-="          { return MINUSEQ;   }
"*="          { return MULTEQ;    }
"/="          { return DIVEQ;     }
"%="          { return MODEQ;     }
"^="          { return POWEQ;     }
"++"          { return INC;       }
"--"          { return DEC;       }
"<="          { return LE;        }
">="          { return GE;        }
"=="          { return EQ;        }
"!="          { return NE;        }
"if"          { return IF;        }
"else"        { return ELSE;      }
"def"         { return DEF;       }
"while"       { return WHILE;     }
"print"       { return PRINT;     }
"end"         { return END;       }
"return"      { return RETURN;    }

{ident}
              {
                      yylval.ident = new char[strlen(yytext)+1];
                      strcpy(yylval.ident, yytext);
```

```
                return IDENT;
            }

{float}
            {
                istringstream(string(yytext)) >> yylval.floatVal;
                return FLOATNUM;
            }

  {int}
           {
                istringstream(string(yytext)) >> yylval.intVal;
                return INTNUM;
            }

{string}
           {
                string x = yytext;
                x = x.substr(1, x.size() - 2);
                yylval.data = (new string(x));
                return DATA;
           }

.
          {
                yyerror("undefined token", yytext[0]); yyterminate();
          }
%%
```

MAIN (*CPP FILE*)

```
#include <iostream>
#include <cstdio>

using std::cerr;
using std::endl;

extern int yyparse();
extern int parseResult;
extern FILE* yyin;

int main()
{

	// if a filename was passed on the command line, attempt to open handle for parsing
	// else, we are reading from STDIN
	yyparse();

	return parseResult;
}
```

OPERATION (*CPP FILE*)

```
#include <bits/stdc++.h>
#include "Operation.h"
#include "datatypes.h"
#include "evaluate.h"
using namespace std;

extern std::map<int, std::map<string, Arithmetic*> > symboltable;
extern int currscope;
extern std::vector<int> allscopes;
extern int threeadd;
```

```
Operation* Print::perform()
{
	Arithmetic* x = dynamic_cast<Arithmetic*>(str->perform());
	cout << (x->print()) << "\n";
	return NULL;
}

Operation* Assign::perform()
{
	Operation* x = dynamic_cast<Operation*>((a2)->perform());
	// cout << name << " " << ((dynamic_cast<Integer*>(x))->value) << endl;
	Arithmetic* xx = dynamic_cast<Arithmetic*>(setValue(NULL, x));
	// cout << name << " " << ((dynamic_cast<Integer*>(xx))->value) << endl;
	symboltable[0][name] = xx;
	return xx;
}

Operation* CreateInteger::perform()
{
	return new Integer(val);
}

Operation* CreateString::perform()
{
	return new String(val);
}

Operation* CreateFloat::perform()
{
	return new Float(val);
}

Operation* GetVal::perform()
{
	// cout << "hi\n";
	if (symboltable[0].find(name) == symboltable[0].end())
	{
		throw "Undefined variable " + name;
	}
	return symboltable[0][name];
}
```

```
Operation* Arithm::perform()
{
        Operation* y = NULL;
        Operation *a, *b;
        a = a1->perform();
        b = a2->perform();

        if(((dynamic_cast<Arithmetic*> (a))->type) !=

              ((dynamic_cast<Arithmetic*> (b))->type)    && !(x[0] == '*'

              && ((dynamic_cast<Arithmetic*> (a))->type) == "string" &&

              ((dynamic_cast<Arithmetic*> (b))->type) == "integer"))

                  throw "Invalid data types " + ((dynamic_cast<Arithmetic*> (a))->type) + " and "

                  +((dynamic_cast<Arithmetic*> (b))->type) + " for operation " + x;

         Arithmetic* zz = (dynamic_cast<Arithmetic*> (a));

         Arithmetic* yy = (dynamic_cast<Arithmetic*> (b));

         y = findValue(a, x, b);

         // cout << ((dynamic_cast<Arithmetic*>(y))->name) << " = " <<

                 ((dynamic_cast<Arithmetic*>(a))->name) << " " << x << " " <<

                  ((dynamic_cast<Arithmetic*>(b))->name) << endl;
        return y;
}

Operation* If_Else::perform()
{
        vector<Operation*> v = *(a2);

        Operation* x = a1->perform();

        Integer* y = dynamic_cast<Integer*>(x);

        if(y->value != 0)
        for(int i = 0; i < v.size(); i++)
        {
                v[i]->perform();
        }
        else if(a3 != NULL)
        {
                vector<Operation*> v_ = *(a3);
                for(int i = 0; i < v_.size(); i++)
                {
                        v_[i]->perform();
                }
        }
        return NULL;
}

Operation* While::perform()
{
        vector<Operation*> v = *(a2);
        Operation* x = a1->perform();
        Integer* y = dynamic_cast<Integer*>(x);
        // cout << (y->value) << "\n";
        int cnt = 0;
        while(y->value != 0)
        {
                for(int i = 0; i < v.size(); i++)
                {
                         v[i]->perform();
                }
                x = a1->perform();
                y = dynamic_cast<Integer*>(x);
                // cout << (y->value);
                // cout << ((dynamic_cast<Integer*>((symboltable[0]["a"])))->value) << "\n";
                // cout << "x" << endl;
         }
        return NULL;
}
```

OPERATION (*HEADER FILE*)

```
#ifndef ABC_MY
#define ABC_MY
#include <bits/stdc++.h>
using namespace std;

class Operation
{
        public:
        virtual Operation* perform() = 0;
};

class Print: public Operation
{
        public:
        Print(Operation* st)
        {
                str = st;
        }
        Operation* str;
        Operation* perform();
};

class Assign: public Operation
{
        public:
        Operation *a2;
        string name;
        Assign(char* a, Operation* b)
        {
                name = string(a);
                a2 = b;
        }
        Operation* perform();
 };
```

```
class CreateInteger: public Operation
{
        public:
        int val;
        CreateInteger(int x)
        {
                val = x;
        }
        Operation* perform();
};

class CreateString: public Operation
{
        public:
        string val;
        CreateString(string x)
        {
                val = x;
        }
        Operation* perform();
};

class CreateFloat: public Operation
{
        public:
        double val;
        CreateFloat(double x)
        {
                val = x;
        }
        Operation* perform();
};

class GetVal: public Operation
{
        public:
        string name;
        GetVal(char* x)
        {
        name = string(x);
        }
        Operation* perform();
```

```
};

class Arithm: public Operation
{
	public:
	Operation* a1, *a2;
	string x;
	Arithm(Operation* a, string b, Operation* c)
	{
		Operation* xx = a;
		a1 = dynamic_cast<Operation*>(xx);
		xx = c;
		a2 = dynamic_cast<Operation*>(xx);
		x = b;
	}
	Operation* perform();
};

class If_Else: public Operation
{
	public:
	Operation* a1;
	vector<Operation*> *a2, *a3;
	If_Else(Operation* a, vector<Operation*> * c)
	{

		a1 = a;
		a2 = c;
		a3 = NULL;
	}
	If_Else(Operation* a, vector<Operation*> * c, vector<Operation*> * d)
	{

		a1 = a;
		a2 = c;
		a3 = d;
	}
	Operation* perform();
};

class While: public Operation
{
        public:
        Operation* a1;
        vector<Operation*> *a2;
        While(Operation* a, vector<Operation*> * c)
        {

                a1 = a;
                a2 = c;
        }
        Operation* perform();
};
#endif
```

PARSE (.*YACC FILE*)

```
%error-verbose

%{
#include <iostream>
#include <map>
#include <vector>
#include <string.h>
#include "evaluate.h"
#include "Operation.h"

using std::cerr;
using std::endl;
using std::string;
using std::cout;

extern int yylex();
extern int yylineno;
extern int threeadd;
bool error = false;

int parseResult = 1;
extern std::map<int, std::map<string, Arithmetic*> > symboltable;
extern int currscope;
extern std::vector<int> allscopes;
class Arithmetic;
void yyerror(const char* s)
 {
        // bision seems to call both versions of yyerror, check a flag to suppress the duplicate
message
        if (!error)
        {
                cerr << "error (line " << yylineno << "): " << s << endl;
        }
}

void yyerror(const char* s, char c)
{
        cerr << "error (line " << yylineno << "): " << s << " \"" << c << "\"" << endl;
        error = true;
}

%}

%union
 {
   int                          intVal;
   double                          floatVal;
   char*                          ident;
   string*                        op;
   Operation*                       atm;
   string*                        data;
   vector<Operation*>*             stmt;
   vector<vector<Operation*>*>*      pr;
}

%token<intVal>   INTNUM    "int"
%token<floatVal> FLOATNUM  "float"
```

```
%token<ident>    IDENT     "identifier"
%token<data>     DATA      "string data"
%token PLUSEQ  "+="
%token MINUSEQ "-="
%token MULTEQ  "*="
%token DIVEQ   "/="
%token MODEQ   "%="
%token POWEQ   "^="
%token INC     "++"
%token DEC     "--"
%token LE      "<="
%token GE      ">="
%token EQ      "=="
%token NE      "!="
%token IF      "if"
%token ELSE    "else"
%token WHILE   "while"
%token DEF     "def"
%token PRINT   "print"
%token RETURN  "return"
%token END     "end"

%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE
%left "<" LE ">" GE EQ NE
%left "+" "-"
%left "*" "/" "%" "^"
%nonassoc UMINUS


%type<op> assignment_op relational_op arithmetic_op
%type<atm> expression zz print assignment list if_else while_loop
%type<stmt> program
%type<pr>   if_else_val


%%
```

```
main            : program
                  {
                          // cout << "1\n";
                          if($1 != NULL)
                          {
                                  vector<Operation*> v = *($1);
                                  for(int i = 0; i < v.size(); i++)
                                  {
                                          try
                                          {
                                                  v[i]->perform();
                                          }
                                          catch(string x)
                                          {
                                                  cout << x << endl;
                                          }
                                  }
                          }
                  }
            ;

program         :   program list
                    {
                          // cout << "2\n";
                          if($1 == NULL && currscope == 0)
                          {
                                  try
                                  {
                                          ($2)->perform();
                                  }
                                  catch(string x)
                                  {
                                          cout << x << endl;
                                  }
                                  $$ = NULL;
                          }
                          else
                          {
                                  ($1)->push_back($2);
$$ = $1;
                          }
                     }
                  |   list
                     {
                          if(currscope == 0)
                          {
                                  try
                                  {
                                          ($1)->perform();
                                  }
                                  catch(string x)
                                  {
                                          cout << x << endl;
                                  }
                                  $$ = NULL;
                          }
                          else
                          {
                                  // cout << "22" << endl;
                                  vector<Operation*>* ab = (new vector<Operation*>());
                                  ab->push_back($1);
                                  $$ = ab;
                          }
                     }
            ;

list        :   assignment '\n' {$$ = $1;}
            |   print '\n'      {$$ = $1;}
            |   if_else '\n'    {$$ = $1;}
            |   while_loop '\n' {$$ = $1;}
            ;

if_else       :       "if" expression ':' '\n' {currscope++;} if_else_val  {
                                                  $$ = new If_Else($2, (*($6))[0], (*($6))[1]);
                                                  currscope--;
                                               }


                                                ;
```

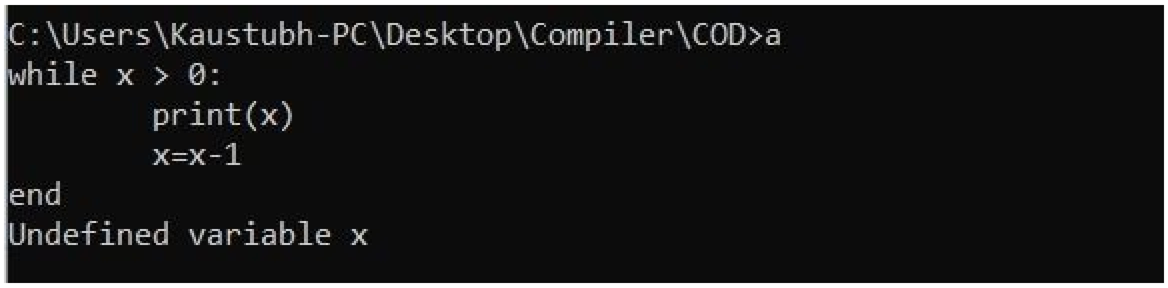
```
if_else_val     :   program "else" ':' '\n' program "end"  {
                                                            vector<vector<Operation*>* >* p = new
vector<vector<Operation*>* >();
                                                            p->push_back($1);
                                                            p->push_back($5);
                                                            $$ = p;
                                                        }
                |   program "end"   {
                                        vector<vector<Operation*>* >* p = new vector<vector<Operation*>*
>();
                                                            p->push_back($1);
                                                            p->push_back(NULL);
                                                            $$ = p;
                                    }
                ;


while_loop      :   "while" expression ':' '\n' {currscope++;} program "end" {
                                                    $$ = new While($2, $6);
                                                    currscope--;
                                                }
                ;

// stmtlst          :   stmtlst assignment '\n' { ($1)->push_back($2);
//                                              $$ = $1;
//                                          }
//                  |   stmtlst print '\n'      { ($1)->push_back($2);
//                                              $$ = $1;
//                                          }
//                  |   print '\n'              { $$ = (new vector<Operation*>());
//                                              $$->push_back($1);
//                                          }
//                  |   assignment '\n'         { $$ = (new vector<Operation*>());
//                                              $$->push_back($1);
//                                          }
                ;
```

```
// return          :   "return" expression
//                 ;

// start           :   start assignment '\n'   { ($2)->perform(); }
//                 |   start print '\n'        { ($2)->perform(); }
//                 |   assignment '\n'         { ($1)->perform(); }
//                 |   print '\n'              { ($1)->perform(); }
//                 ;




assignment      :   IDENT '=' expression    {
                                            // cout << "aye\n";
                                            $$ = new Assign($1, $3);

                                        }
                |   IDENT assignment_op expression   {
                                            Operation* k = new GetVal($1);
                                            Operation* t = new Arithm(k, *($2), $3);
                                            $$ = new Assign($1, t);
                                        }
                ;

assignment_op   :   "-="    { $$ = new string("-="); }
                |   "+="    { $$ = new string("+="); }
                |   "/="    { $$ = new string("/="); }
                |   "%="    { $$ = new string("%="); }
                |   "*="    { $$ = new string("*="); }
                ;

expression      :   expression relational_op expression {
                                                    $$ = new Arithm($1, *($2), $3);
                                                }
                |   expression arithmetic_op expression {
                                                    $$ = new Arithm($1, *($2), $3);
                                                }
                |   '(' expression ')' {
```

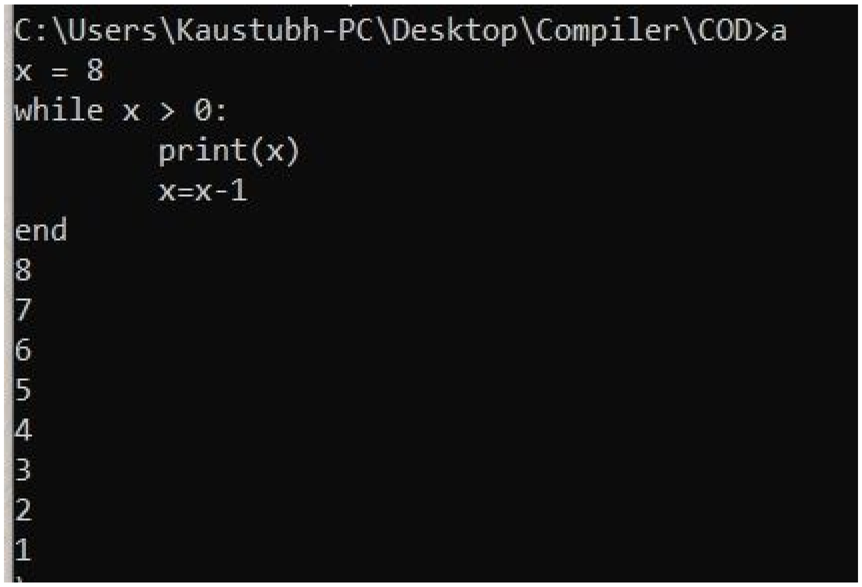
```
  $$ = $2;
                                        }
                |   zz   {
                            $$ = $1;
                         }
                |   INTNUM  {
                                $$ = new CreateInteger($1);
                            }
                |   FLOATNUM{
                                $$ = new CreateFloat($1);
                            }
                |   DATA{
                            $$ = new CreateString(*($1));
                        }
                ;

zz              :   IDENT   {
                                $$ = new GetVal(($1));
                            }
                ;

relational_op   :   "<=" {$$ = new string("<=");}
                |   ">=" {$$ = new string(">=");}
                |   '<' {$$ = new string("<");}
                |   '>' {$$ = new string(">");}
                |   "==" {$$ = new string("==");}
                |   "!=" {$$ = new string("!=");}
                ;

arithmetic_op   :   '+' {$$ = new string("+");}
                |   '-' {$$ = new string("-");}
                |   "**" {$$ = new string("**");}
                |   '*' {$$ = new string("*");}
                |   '/' {$$ = new string("/");}
                |   '%' {$$ = new string("%");}
                ;

print           :   PRINT expression  {
```

```
Operation* yy = ($2);
                                    $$ = new Print(yy);
                                }
            ;
%%

std::map<int, std::map<string, Arithmetic*> > symboltable;
int currscope = 0;
int threeadd = 1;
std::vector<int> allscopes;
int main(){
    allscopes.push_back(1);
    cout << ">> ";
    return yyparse();
}
int yywrap(){
    return(1);
}
```

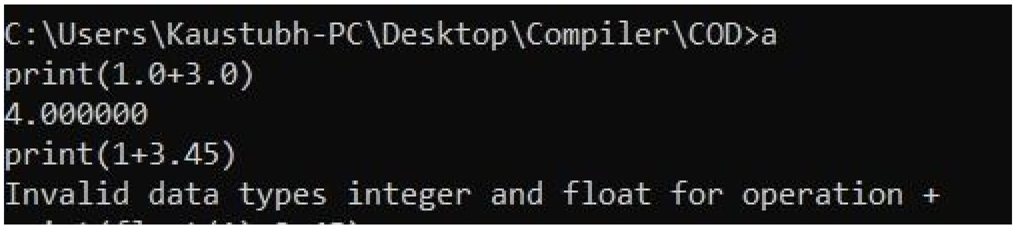
Output:-



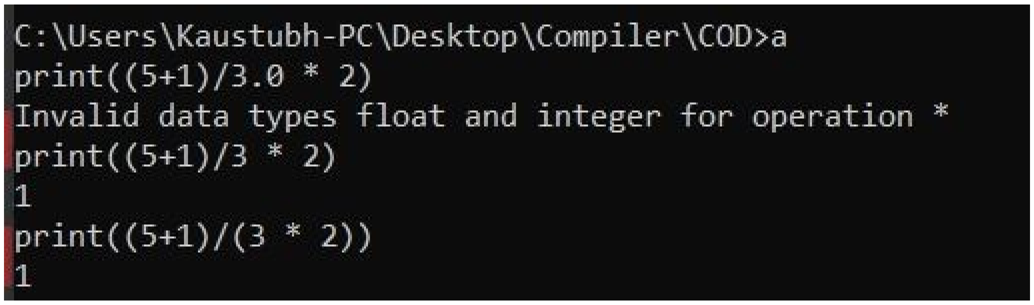
```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
while x > 0:
        print(x)
        x=x-1
end
Undefined variable x
```



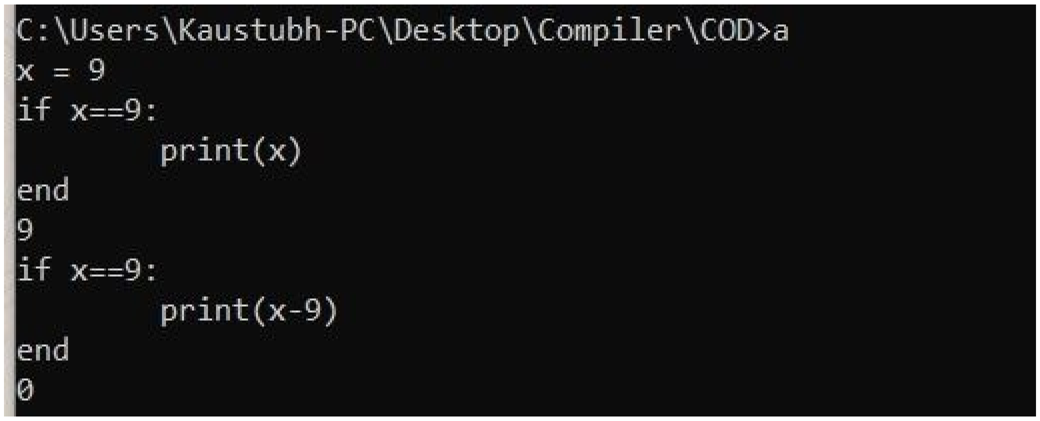
```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
x = 8
while x > 0:
        print(x)
        x=x-1
end
8
7
6
5
4
3
2
1
```



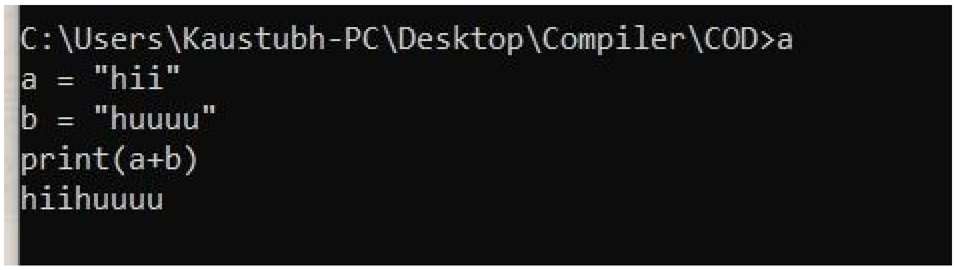
```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
print(1.0+3.0)
4.000000
print(1+3.45)
Invalid data types integer and float for operation +
```



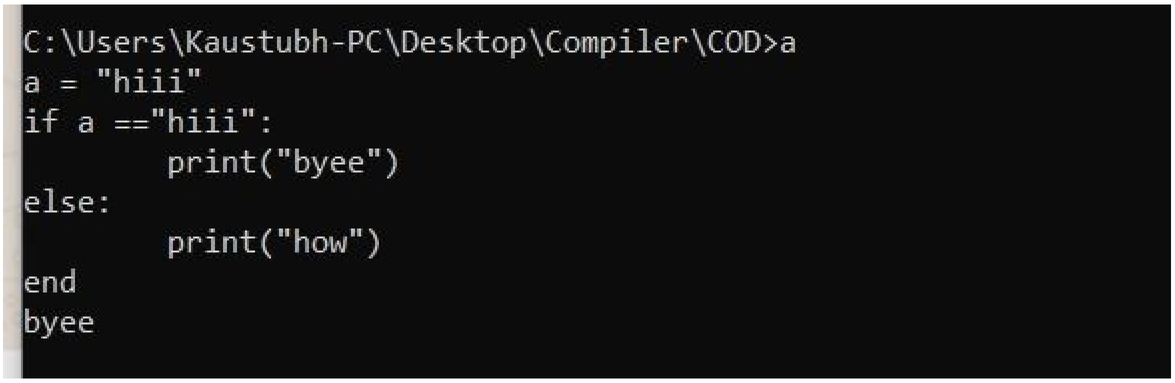
```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
print((5+1)/3.0 * 2)
Invalid data types float and integer for operation *
print((5+1)/3 * 2)
1
print((5+1)/(3 * 2))
1
```



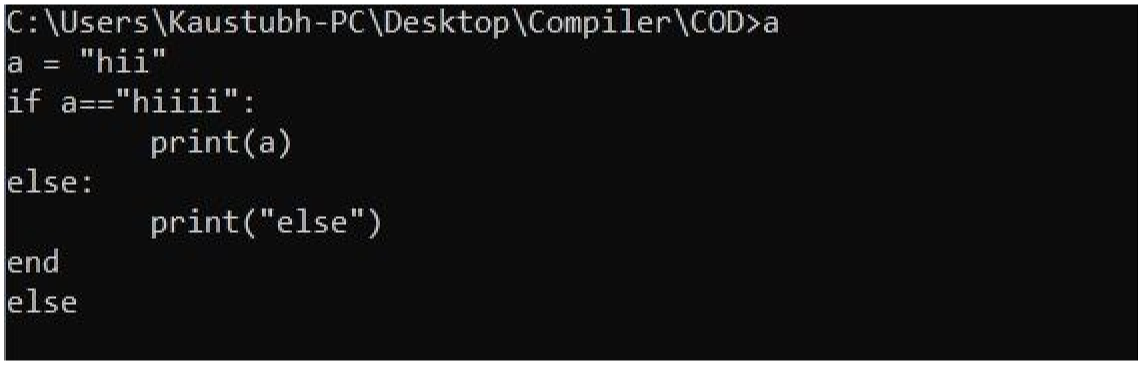
```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
x = 9
if x==9:
        print(x)
end
9
if x==9:
        print(x-9)
end
0
```



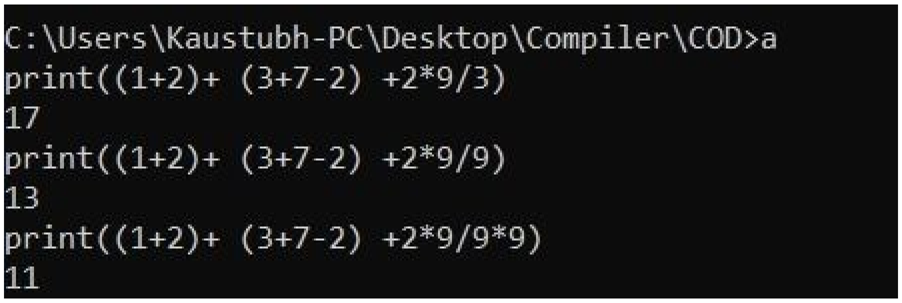
```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
a = "hii"
b = "huuuu"
print(a+b)
hiihuuuu
```



```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
a = "hiii"
if a =="hiii":
        print("byee")
else:
        print("how")
end
byee
```



```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
a = "hii"
if a=="hiiii":
        print(a)
else:
        print("else")
end
else
```



```
C:\Users\Kaustubh-PC\Desktop\Compiler\COD>a
print((1+2)+ (3+7-2) +2*9/3)
17
print((1+2)+ (3+7-2) +2*9/9)
13
print((1+2)+ (3+7-2) +2*9/9*9)
11
```