Rev Programming Language

SER502-SPRING2023-TEAM

- Chedvihas punnam
- Teja Lam
- Gnanadeep Settykara
- Naga Vamsi Bhargav Seelam
- Vikranth Reddy Tripuram

Table of Contents

Introduction

- We have named our language as Rev Language and in this language we generated tokens using python Lexer file.
- We Generated Parse Tree using prolog.
- For the files which we are giving as an input we have .rev extension
- In this Language, We start our program with “init” and end with “exit”.
- For Tokens we have used Python version 3.11.

Tools Used

- Tokens Generation : Python
- For Compilation, Parsing : SWI Prolog
- Other Tools : VS Code, Git

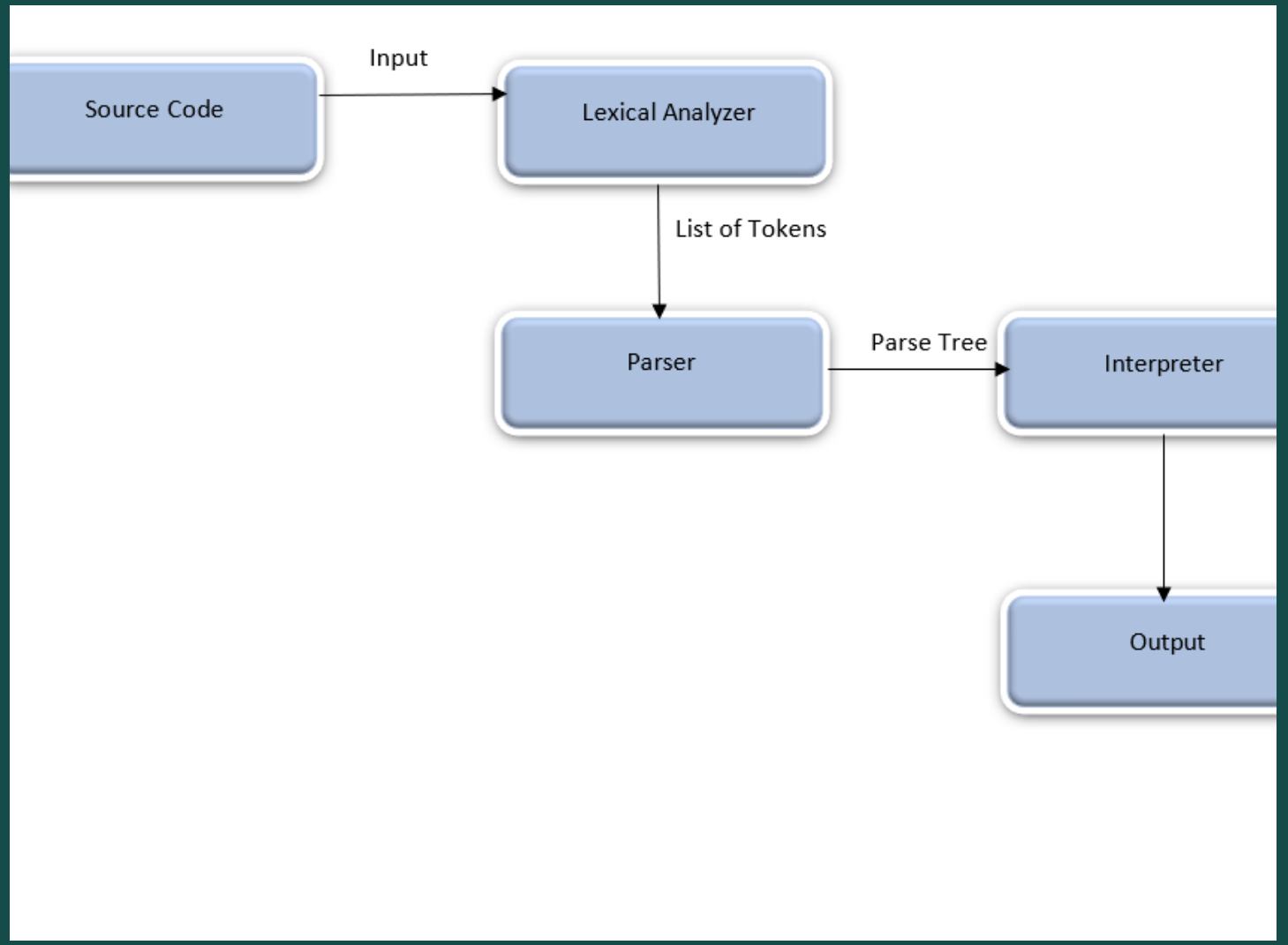
Program design and execution flow

Lexical Analyzer:

- Breaks down source code into meaningful tokens.
- Follows grammar rules set for the specific language being compiled.
- Produces a token list.

Parser:

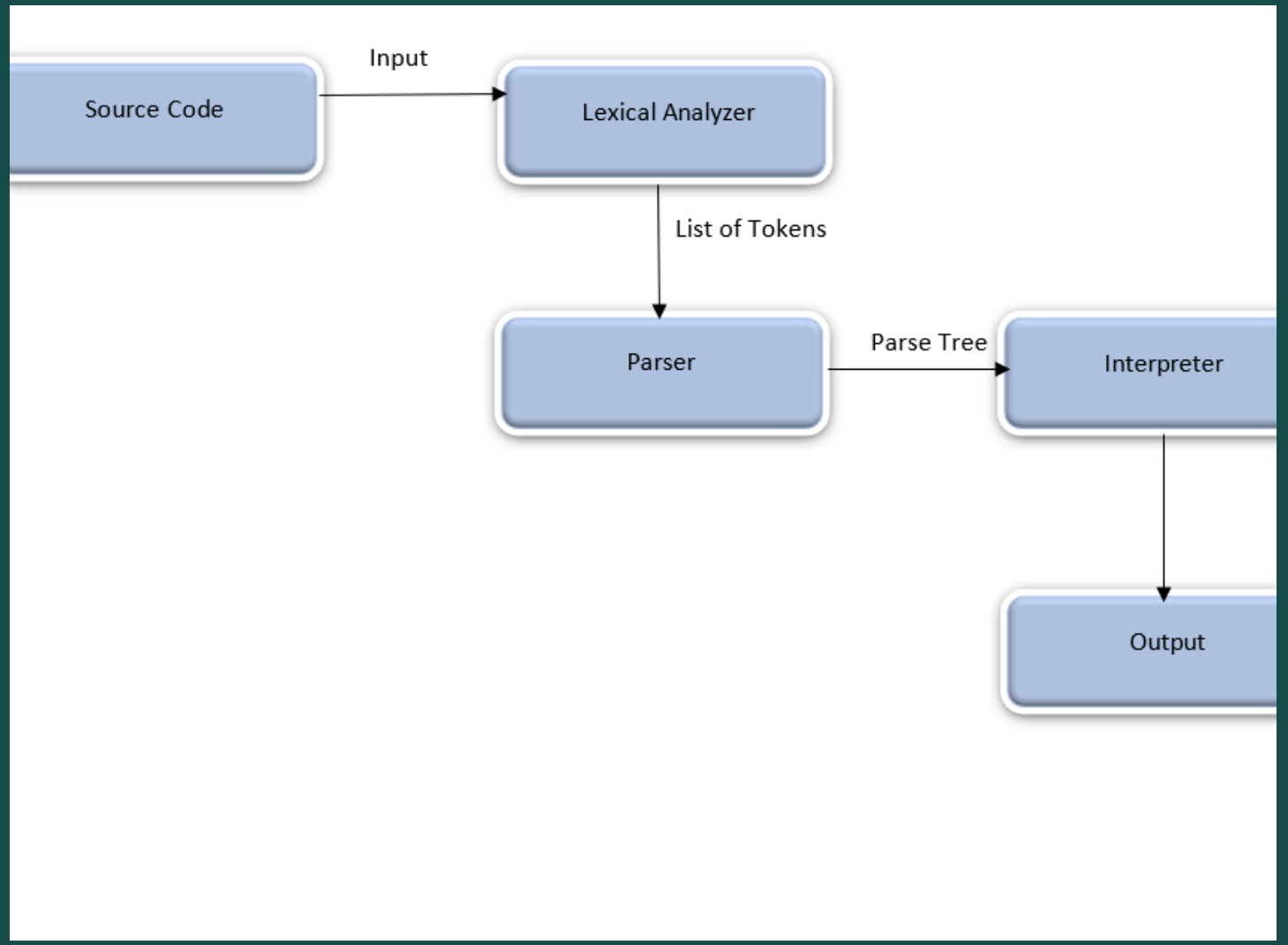
- Receives the token list produced by the lexer.
- Examines each token according to the grammar rules.
- Uses Prolog to construct a parse tree.
- Performs this stage once all tokens have been parsed.



Program design and execution flow

Interpreter:

- Examines the parse tree produced by the parser.
- Traverses each node in the parse tree.
- Analyzes each node using semantic rules.
- Modifies each node accordingly.



Features of Rev Language

- Data Types:

We have data types like integer,boolean and string.

Samples

int a = 23;

bool b = true;

varchar s = ‘ser502’;

- Arithmetic Operators:

Addition : ‘+’

Subtraction : ‘-’

Multiplication : ‘ * ’

Division : ‘ / ’

Samples

a = a * 5;

c = c/4;

Features of Rev Language

- **Boolean Operators:**

AND operation

Sample : p and q;

OR operation

Sample : p or q;

NOT operation

Sample : !q

- **Assignment Operator**

‘ = ‘

Sample code :

int a = 57;

varchar v = ‘lang’;

Features of Rev Language

➢ **Loops:**

For Loop

Sample For:

```
 for(int a = 2;a < 13;a++)
{
print a;
}
```

➢ **Loops**

While Loop

Sample While:

```
int f = 0;
while(f!=8)
{
f = f +1;
}
```

Features of Rev Language

- **Loops:**

ForEach Loop

Sample Loop:

```
int r=5;
for x in each(1:10)
{
        x=x+r;
}
print x;
```

- **Conditional Statements**

If, else and else if statements,

Ternary Operator

Samples:

```
int s = 45;
If(s>=10)
{
s = s+5;
}
else
{ s = s-5;}
print s;
```

Syntax description:

- { ---------- > Start of the block
- } ---------- > End of the block
- int ---------- > Integer data type
- bool ---------- > Boolean data type
- if-then-else ---------- > Else if condition statement
- If-then ----------> If condition statement
- true ----------→ True
- false ----------→ False
- = ----------→ Assignment Operator

Grammar

prg ::= INIT stmnt DLMTR END.

stmnt ::= stmntop DLMTR stmnt | stmnt

stmntop ::= declare | assign | if-else | print | for | while

declare ::= DATATYPE IDNTFR | DATATYPE IDNTFR ASSGN data

data ::= FLOAT | CHAR | BOOL | VARCHAR| INT

comment ::= COMMENT VARCHAR

assign ::= IDNTFR ASSGN expressn

expressn ::= trm | trm ADD expressn | trm SUB expressn

trm ::= fctr | fctr MUL trm | fctr DIV trm

fctr ::= data | IDNTFR | OPENPRNTHS expressn CLOSEPRNTHS

Grammar

if-else ::= IF OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt

DLMTR CLOSEBRC | IF OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt

DLMTR CLOSEBRC DLMTR ELSE OPENPRNTHS stmnt DLMTR CLOSEPRNTHS | IF

OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt DLMTR CLOSEBRC

DLMTR, elif DLMTR ELSE OPENBRC stmnt DLMTR CLOSEBRC

elif ::= elif1 | elif DLMTR elif1

elif1 ::= ELSEIF OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt DLMTR

CLOSEBRC

print ::= PRINT IDNTFR | PRINT STRTQT STRING ENDQT

while ::= WHILE OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt

CLOSEBRC

Grammar

for ::= FOR forRng OPENBRC stmnt CLOSEBRC

forRng ::= IDNTFR IN RANGE OPENPRNTHS expressn COMMA

expressn CLOSEPRNTHS | OPENPRNTHS IDNTFR ASSGN expressn DLMTR

IDNTFR CMP expressn DLMTR CLOSEPRNTHS | OPENPRNTHS IDNTFR

ASSGN expressn DLMTR IDNTFR CMP expressn DLMTR expressn

CLOSEPRNTHS

condition ::= BOOL | IDNTFR CMP expressn | IDNTFR CMP expressn CNDOPR

condition

Sample Program for finding sum of first 5 numbers

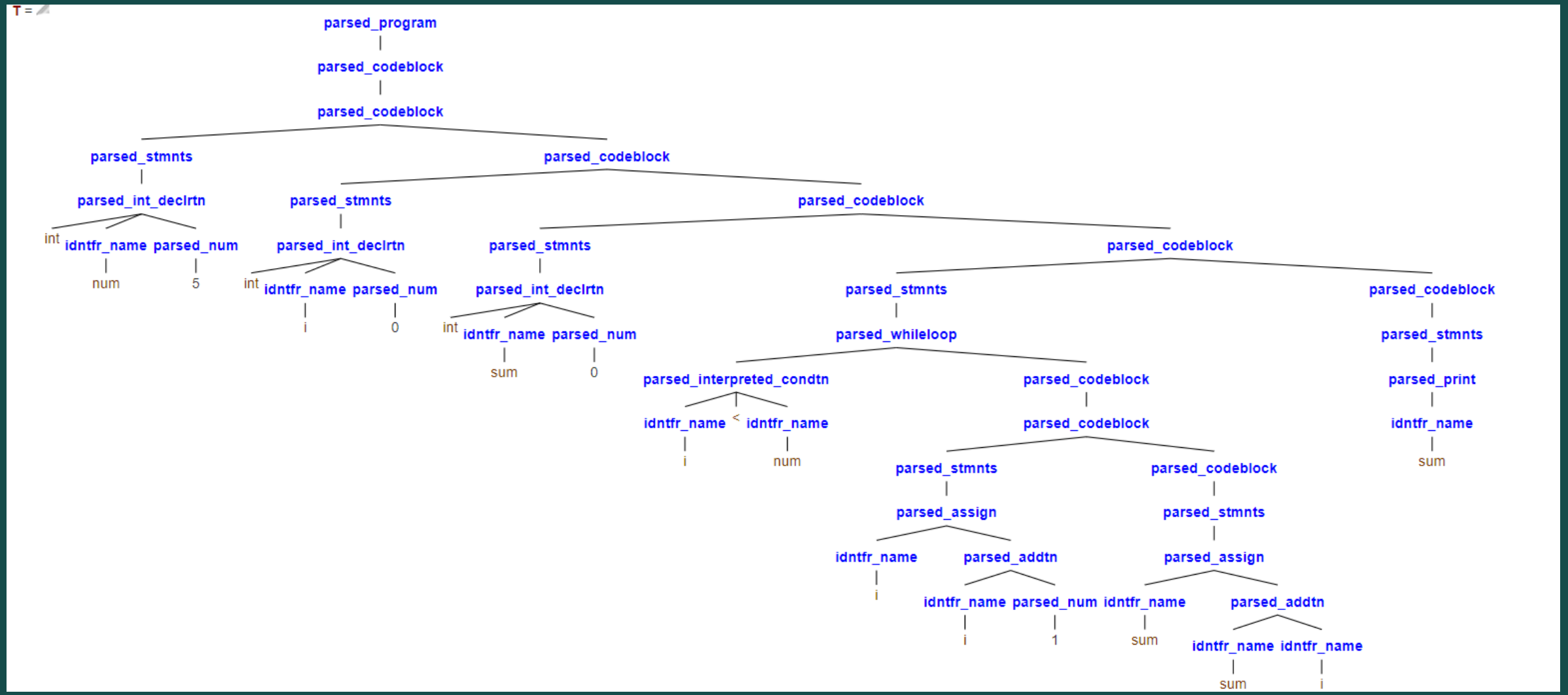
```
init
{
int  num=5;
int  i=0;
int  sum=0;
while(i<num)
{
    i=i+1;
    sum=sum+i;
}
print  sum;
}
exit
```

```
Program-Output:
15
true
```

Sample Intermediate Code

```
program(T, [init,'{',int,num,=,5,;,int,i,=,0,;,int,sum,=,0,;,while,'(',i,<,num,')','{',i,=,i,+,1,;,sum,=,sum,+,i,;,'}',print,sum,;,'}',exit], []), exec_program(T,F)
15
F = [(int,num,5), (int,i,5), (int,sum,15)],
T =
parsed_program(
parsed_codeblock(
parsed_codeblock(parsed_stmnts(parsed_int_declrtn(int,idntfr_name(num),parsed_num(5))),
parsed_codeblock(parsed_stmnts(parsed_int_declrtn(int,idntfr_name(i),parsed_num(0))),
parsed_codeblock(parsed_stmnts(parsed_int_declrtn(int,idntfr_name(sum),parsed_num(0))),
parsed_codeblock(
parsed_stmnts(
parsed_whileloop(parsed_interpreted_condtn(idntfr_name(i),<,idntfr_name(num)),
parsed_codeblock(
parsed_codeblock(parsed_stmnts(parsed_assign(idntfr_name(i),parsed_addtn(idntfr_name(i),parsed_num(1)))),
parsed_codeblock(parsed_stmnts(parsed_assign(idntfr_name(sum),parsed_addtn(idntfr_name(sum),idntfr_name(i))))))
)
)
)
?- program(T, [init,'{',int,num,=,5,;,int,i,=,0,;,int,sum,=,0,;,while,'(',i,<,num,')','{',i,=,i,+,1,;,sum,=,sum,+,i,;,'}',print,sum,;,'}',exit], []), exec_program(T,F)
```

Sample Intermediate Code Parse Tree



Sample Execution 1

```
init
{
int r=1;
int e=5;
int v=8;

if(r>e and e<v)
{
   print r;

}
else
{
 print e;

}

if(r>e or e<v)
{
print r;
}
else
{
print e;
}

if(r!=e)
{
print v;
}

}
exit
```

```
Program-Output:
5
1
8
true
```

Sample Execution 2

```
init
{
 int r=5;
 for x in each(1:10)
 {
        x=x+r;
 }
 print x;
}
exit
```

```
Program-Output:
10
true
```

Sample Execution 3

```
init
{
int a=158;
varchar b='one fifty eight';
bool c=false;

print a;
print b;
print c;



}
exit
```

```
Program-Output:
158
one fifty eight
false
true
```

Sample Execution 4

```
init
{
int a=2;
int b=5;
int sum;
int sub;
int mul;
int div;


sum=a+b;
print sum;

sub=a-b;
print sub;

mul=a*b;
print mul;

div=b/a;
print div;


}
exit
```

```
Program-Output:
7
-3
10
2
true
```

Sample Execution 5

```
init
{
  int r=10;
  varchar tern='Output using Ternary operator:';
  r>=5? r=r*2; : r=r/2;;
  print tern;
  print r;
}
exit
```

```
Program-Output:
Output using Ternary operator:
20
true
```

THANK YOU