SAANP

#SastaPython

SAANP

Introductions

Team 26 from the SER-502 Spring 23 class has five members:

Manisha M Deshpande

ASU SE Grad

Love food, sleep, anime, programming. In that order.

Hardik Sakhuja

Software Engineering @ ASU

Cloud-Native developer and a Snooker/Pool guru.

Sahithya Cherukuri

Computer Software Engineering @ ASU

Code, Learn, Update, Hibernate.

Sameer Mungole

Computer Software Engineering, ASU.

Mobile developer by day, assignment finisher by night!

Saudamini Khare

Software Engineering Grad at ASU.

A computer vision enthusiast who loves to sing

Overview

- Language Introduction
- Features
- Language Grammar
- Language Specifications
- Tools used
- High Level Design
- Tokenization
- Parsing
- Evaluation
- Prerequisites for Execution
- Execution Steps
- Sample run

Language Introduction

SAANP is a programming language that is inspired by the simpleness of Python.

It is easy to code in, therefore useful for beginners. However, unlike python which relies on indentation, SAANP takes inspiration from C, but uses ‘.’ to separate the statements and ‘endif/endwhile/endfor’ to determine the end of block.

SAANP, similar to Python, is a dynamically typed language, and variable declaration can happen anywhere throughout the program.

NOTE: The declared variables are available in the global scope, for the rest of the program.

Features

Data types:

- integers
- string
- boolean

Mathematical operations:

- Addition
- Subtraction
- Multiplication (with precedence)
- Division
- Expression in Parenthesis

Comparison Operators:

- Equals (==)
- Not Equals (!=)
- Greater than (>)
- Less than (<)

Logical Operators:

- and
- or
- not
- Ternary expression

Features (Von Neumann Language)

Sequence:

- Series of statements are sequentially executed.
- Print method is supported and it prints variables, string literals, numbers, and booleans followed by a newline.

Selection:

- if statement
- if-else statement

Iteration:

- while loop
- traditional for-loop
- Enhanced for-loop

Language Grammar


```
PROG ::= BLK
BLK ::= DEC | IFE | FOR | WHILE | EFOR | PRINT

DEC ::= ID = EXP. | ID = TER. | DEC, BLK

IFE ::= if LOG: BLK endif | if LOG: BLK else: BLK endif | IFE, BLK
TER ::= LOG? EXP : EXP

FOR ::= for ID = NUM, LOG, INC: BLK endfor | FOR, BLK
INC ::= ID = EXP

WHILE ::= while LOG: BLK endwhile | WHILE, BLK

EFOR ::= for ID in range(NUM, NUM): BLK endfor | EFOR, BLK

PRINT ::= print(ID).
PRINT ::= print(STR).
PRINT ::= print(NUM).
PRINT ::= print(BOOL).
PRINT ::= PRINT, BLK

LOG ::= CMP and CMP | CMP or CMP | not CMP | CMP
CMP ::= EXP == EXP | EXP != EXP | EXP < EXP | EXP > EXP | ID | BOOL

EXP ::= TERM | TERM + EXP | TERM - EXP | STR | BOOL
TERM ::= FACTOR | FACTOR * TERM | FACTOR / TERM
FACTOR ::= ID | NUM | (EXP)

ID ::= [a-z]+[a-z0-9_]* - {True|False|if|else|endif|for|endfor|while|endwhile|range|print}
STR ::= "[^"]*"
NUM ::= [0-9]+ | -[0-9]+
BOOL ::= True | False
```

Language Specifications - If and If-Else

If statement:

```
if x == 1:
    y = 2.
    if z != 3:
        y = y + 1.
    endif
endif
```

* nesting is supported

If-else statement:

```
if x > 1:
    y = 2.
else:
    y = 3.
endif
```

Language Specifications - Ternary expression and While

Ternary expression:

```
x = 3.

y = 9.

print(x).

print(y).

z = x > y ? x : y.

print("Greater Number:").

print(z).
```

While statement:

```
print(“Remainder of 15/3”).

num1 = 15.

while num1 > -1:

    num1 = num1 - 3.

endwhile

print(num1).
```

* nesting is supported

Language Specifications - Traditional For-Loop

Traditional FOR loop:

```
print(“Even numbers:”).

for i = 2, i < x , i = i + 2:

    print(i).

endfor
```

* nesting is supported

- The ‘for’ loop requires three things:
  - Variable initialisation
  - A boolean expression
  - Variable increment

Note:

- The variable initialized is still present outside the scope of the for loop.

Language Specifications - Enhanced For-Loop

Enhanced FOR loop:

```
sum = 0.
print("Sum of numbers 1 to 5").
for i in range(1, 5):
    sum = sum + i.
endfor
print(sum).
```

* nesting is supported

- The enhanced ‘for’ loop requires:
  - A variable (which is automatically initialized to the first value of the range during interpretation).
  - A range which is a tuple of two integers.

Note:

- SAANP currently supports only integer literals in the range method. It can be enhanced in the future to support integer variables.

Language Specifications

Variables names: can be a combination, of any length(>0), of the small alphabets(a-z), underscore(_) and digits(0-9), but should not begin with a number. It cannot match the reserved keywords.

Reserved Keywords: True, False, if, else, endif, for, endfor, while, endwhile, range, print.

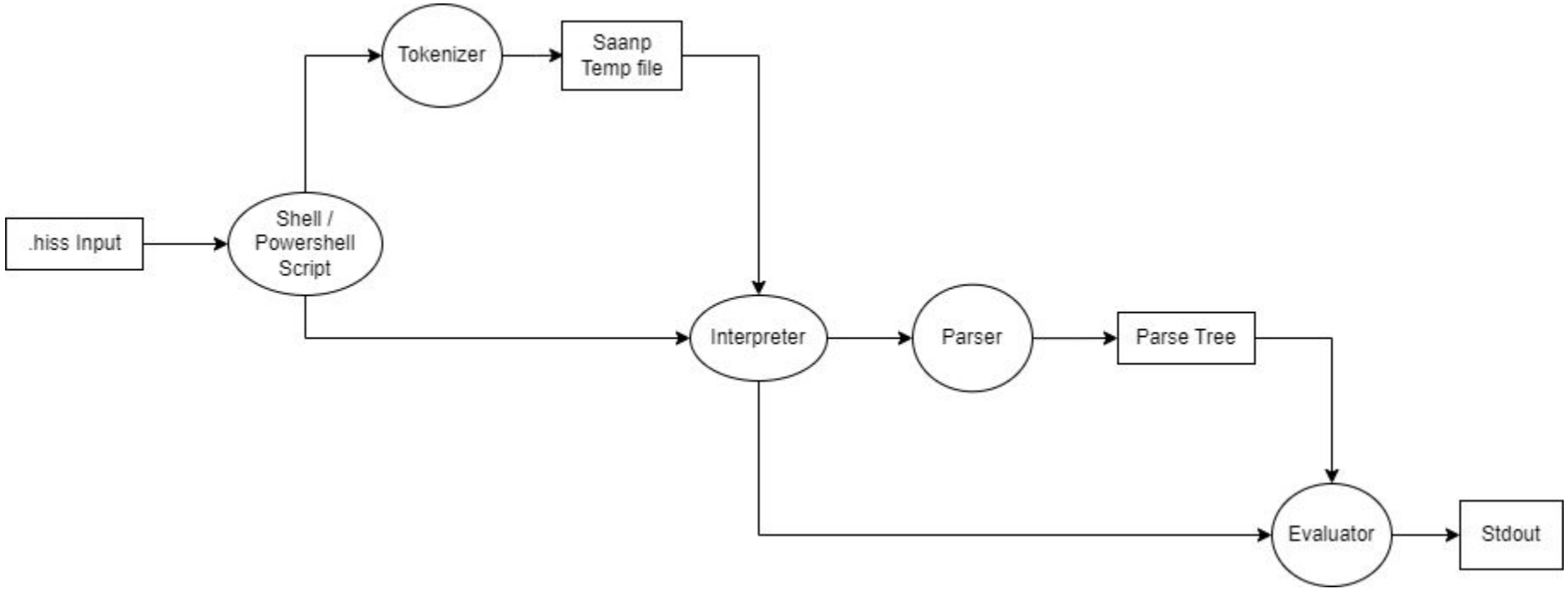
NOT supported:

- More than 1 logical operator in a boolean expression (`ex1 and not ex2`).
- Assignment in an expression (`if x=1: ...`).
- integer or string value as a boolean expression(`x=2. if x: ...`).
- Double quotes within a string literal (`x=”He said “Ok”.”`.).

Tools Used

1. **Tokenizer** - Python (3.8.0+)
2. **Parser** - Prolog (SWI-Prolog version 9.0.4)
3. **Evaluator** - Prolog (SWI-Prolog version 9.0.4)
4. **Execution in MacOS/Linux** - Bash/Shell
5. **Execution in Windows** - Powershell

High Level Design - Timeline Graph



Tokenization

Program to find sum of numbers 1 to 4.

```
sum = 0.
print("Sum of numbers 1 to 4").

for i in range(1, 5):
    sum = sum + i.
endfor

print(sum).
```

Generated Tokens:

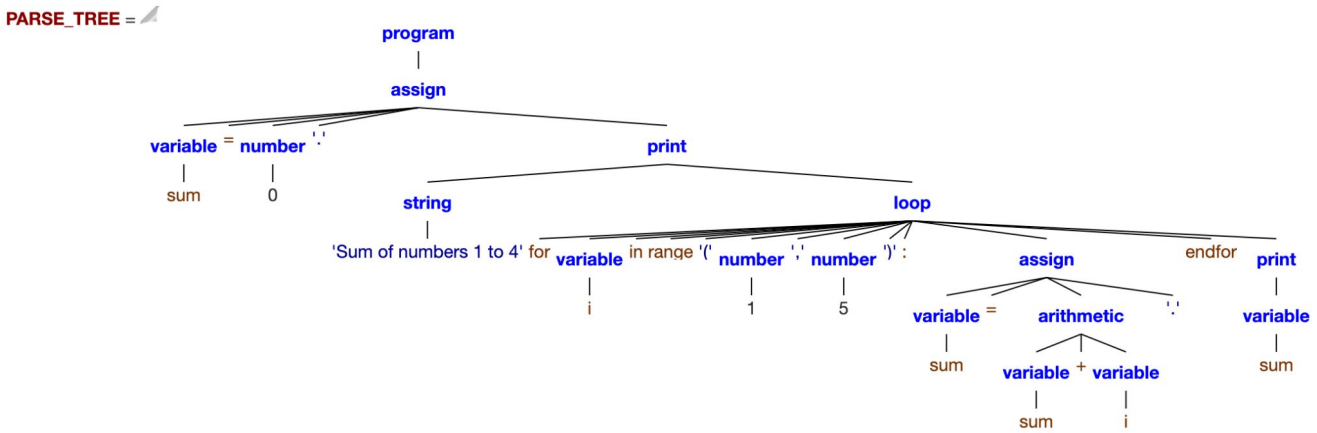
```
[
    'sum', '=', '0', '.',
    'print', '(', '"', 'Sum of numbers 1 to 4', '"', ')', '.',
    'for', 'i', 'in', 'range', '(', '1', ',', '5', ')', ':',
    'sum', '=', 'sum', '+', 'i', '.',
    'endfor',
    'print', '(', 'sum', ')', '.'
].
```

Parsing

Tokens:

```
[
    'sum', '=', '0', '.',
    'print', '(', '"', 'Sum of numbers 1 to 4', '"', ')', '.',
    'for', 'i', 'in', 'range', '(', '1', ',', '5', ')', ':',
    'sum', '=', 'sum', '+', 'i', '.',
    'endfor',
    'print', '(', 'sum', ')', '.'
].
```

Generated Parse Tree:



Evaluation

Parse Tree:

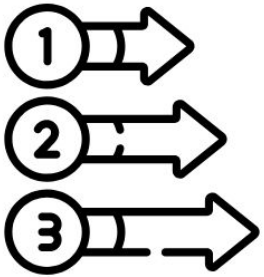
```
PARSE_TREE = program(
    assign(
        variable(sum),
        =,
        number(0),
        '.',
        print(
            string('Sum of numbers 1 to 4'),
            loop(
                for,
                variable(i),
                in,
                range,
                '(',
                number(1),
                ',',
                number(5),
                ')',
                :,
                assign(
                    variable(sum),
                    =,
                    arithmetic(
                        variable(sum),
                        +,
                        variable(i)
                    ),
                    '.'
                ),
                endfor,
                print(variable(sum))
            )
        )
    )
)
```

Output:

Prerequisites for execution

1. In Mac/Linux, execution permission will be required to run saanp.sh. One can do so by running the following command -
   chmod +x saanp.sh
2. In Windows, permission is required to run powershell scripts. One can do so by running the following command in Powershell **(Run as administrator)** -
   set-executionpolicy remotesigned
3. Note - **parser.pl** is using a third-party library for **regex** and if one does not have the package installed in their prolog installation, SAANP will provide a prompt upon execution, to install the package. One will require to say **yes** to the prompt.

Execution Steps



LINUX/ MAC OS:

1. Go to the “src” folder -
   cd src
2. Run saanp.sh with a.hiss file as an argument -
   ./saanp.sh <Path to .hiss file>

Windows:

1. Go to the “src” folder -
   cd src
2. Run saanp.ps1 with a .hiss file as an argument -
   .\saanp.ps1 <Path to .hiss file>

Sample Run - fizzBuzz.hiss

```
number = 15.

num1 = number.
while num1 > 0:
    num1 = num1 - 3.
endwhile
if num1 == 0:
    print("Fizz").
endif

num2 = number.
while num2 > 0:
    num2 = num2 - 5.
endwhile
if num2 == 0:
    print("Buzz").
endif

if num1 != 0 or num2 != 0:
    print(number).
endif
```

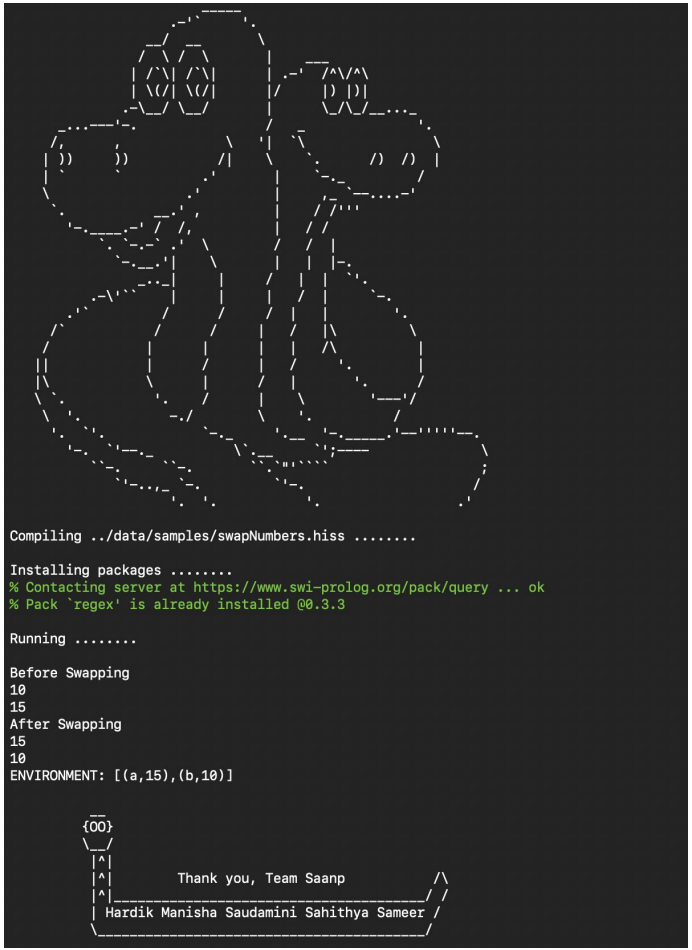
Sample Run - Swap Numbers

```
a = 10.
b = 15.

print("Before Swapping").
print(a).
print(b).

a = a + b.
b = a - b.
a = a - b.

print("After Swapping").
print(a).
print(b).
```



```
Compiling ../data/samples/swapNumbers.hiss ........

Installing packages ........
% Contacting server at https://www.swi-prolog.org/pack/query ... ok
% Pack `regex' is already installed @0.3.3

Running ........

Before Swapping
10
15
After Swapping
15
10
ENVIRONMENT: [(a,15),(b,10)]

        {OO}
        \__/
         |^|
         |^|       Thank you, Team Saanp          /\
         |^|_____________________________________/ /
         | Hardik Manisha Saudamini Sahithya Sameer /
         \_______________________________________/
```

Sample Run - Power

```
number = 10.
print("10 ^ 3").

if number == 1:
    print(1).
else:
    res = 1.
    for _ in range(0, 3):
        res = res * number.
    endfor
    print("result").
    print(res).
endif
```

```
Compiling ../data/samples/power.hiss ........

Installing packages ........
% Contacting server at https://www.swi-prolog.org/pack/query ... ok
% Pack `regex' is already installed @0.3.3

Running ........

10 ^ 3
result
1000
ENVIRONMENT: [(number,10),(res,1000),(_,3)]

        __
       {OO}
       \__/
        |^|
        |^|       Thank you, Team Saanp          /\
        |^|_____________________________________/ /
        | Hardik Manisha Saudamini Sahithya Sameer /
        \_______________________________________/
```

Sample Run - Prime Number

```
x = 13.

if x > 1:
    is_prime = True.
    for i = 2, i < x and is_prime, i = i + 1:
        if (x/i) * i == x:
            is_prime = False.
        endif
    endfor
    print(is_prime).
else:
    print(False).
endif


y = 15.

if y > 1:
    is_prime = True.
    for i = 2, i < y and is_prime, i = i + 1:
        if (y/i) * i == y:
            is_prime = False.
        endif
    endfor
    print(is_prime).
else:
    print(False).
endif
```

```
Compiling ../data/samples/primeNumber.hiss ........

Installing packages ........
% Contacting server at https://www.swi-prolog.org/pack/query ... ok
% Pack `regex' is already installed @0.3.3

Running ........

true
false
ENVIRONMENT: [(x,13),(is_prime,false),(i,4),(y,15)]


         __
        {OO}
        \__/
         |^|
         |^|       Thank you, Team Saanp          /\
         |^|_____________________________________/ /
         | Hardik Manisha Saudamini Sahithya Sameer /
         \_______________________________________/
```

Sample Run - Greater Number

```
x = 3.
y = 9.

print(x).
print(y).

z = x > y ? x : y.

print("Greater Number:").
print(z).
```

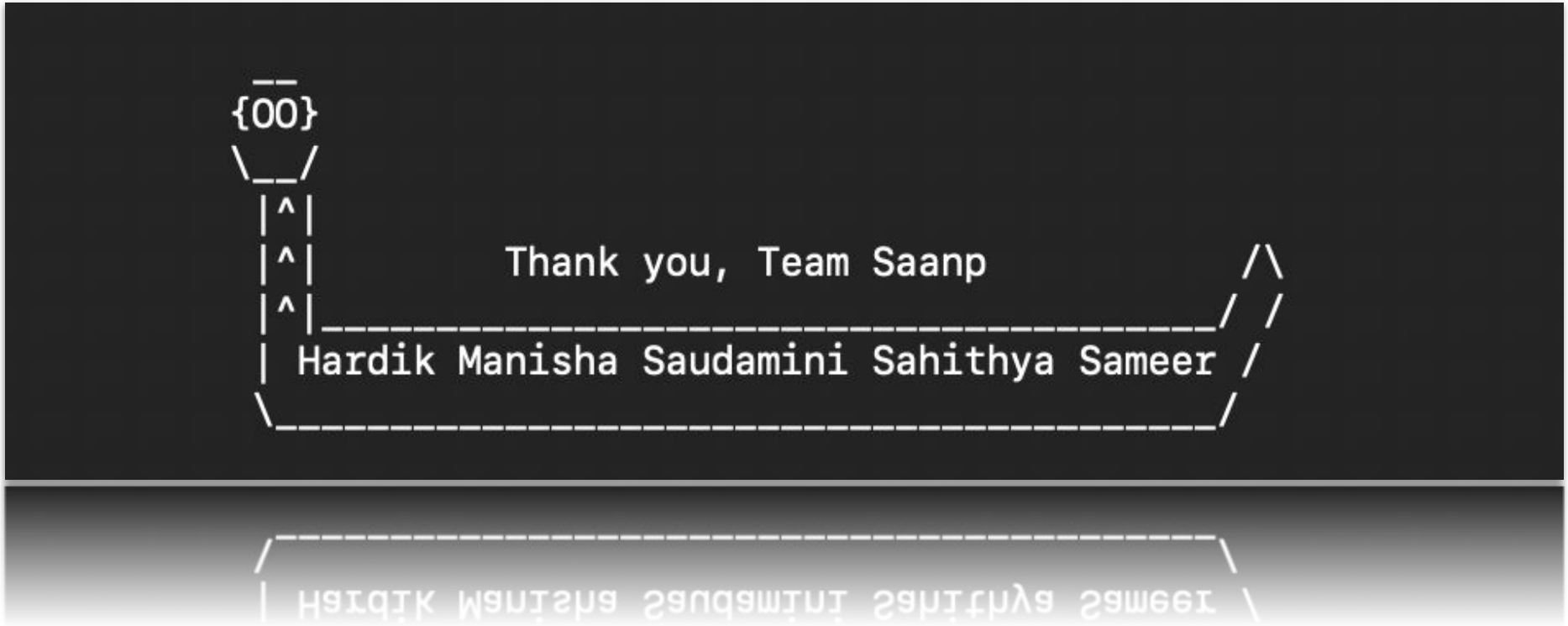
```
Compiling ../data/samples/GreatNum.hiss ........

Installing packages ........
% Contacting server at https://www.swi-prolog.org/pack/query ... ok
% Pack `regex' is already installed @0.3.3

Running ........

3
9
Greater Number:
9
ENVIRONMENT: [(x,3),(y,9),(z,9)]


        __
       {OO}
       \__/
        |^|
        |^|       Thank you, Team Saanp          /\
        |^|_____________________________________/ /
        | Hardik Manisha Saudamini Sahithya Sameer /
        \_______________________________________/
```


Thank you, Team Saanp
Hardik Manisha Saudamini Sahithya Sameer