Bliss Programming Language

Final Project-SER502

Languages and Prog Paradigms

Team Members in Team 27

Satya Partha Varun Ramaraju

Vatsal Malaviya

Ansh Agrawal

Rishabh Alkeshkumar Doshi

Contents

Introduction

Introduction

Bliss(.bs) is a programming language developed by Team27 as part of SER502 Final project implementation. The interpretation and parsing of bliss code is achieved using PLY (Python Lex - YACC) library.

PLY provides a powerful toolkit for building lexers and parsers, enabling the creation of interpreters and compilers for programming languages in Python.

Unique Features of Bliss and its design philosophy

1. Flexibility and Simplicity: Bliss is designed to be simple and flexible, similar to Python, which allows dynamic typing and uses indentation to define code blocks. This approach reduces the syntactic overhead for programmers and makes the language easy to learn and use. We also put traditional for loop in there for the java lovers which is not present in Python.
2. Efficient Parsing and Interpretation: Utilizing the PLY library, Bliss emphasizes efficient parsing and interpretation of the code. This is achieved through the use of Lex and YACC for tokenizing and parsing, which are powerful tools for building interpreters and compilers.

The design philosophy is centered around creating a user-friendly programming environment that leverages robust tools for language parsing and error management, thus making it accessible to learners and effective for educational purposes.

Technology and Tools

Technology and Tools Used

- Lexical Analysis and Parsing - PLY (Python Lex and YACC)
- Base Interpreter - Python
- Version Control - Git/Github

Design of the Language

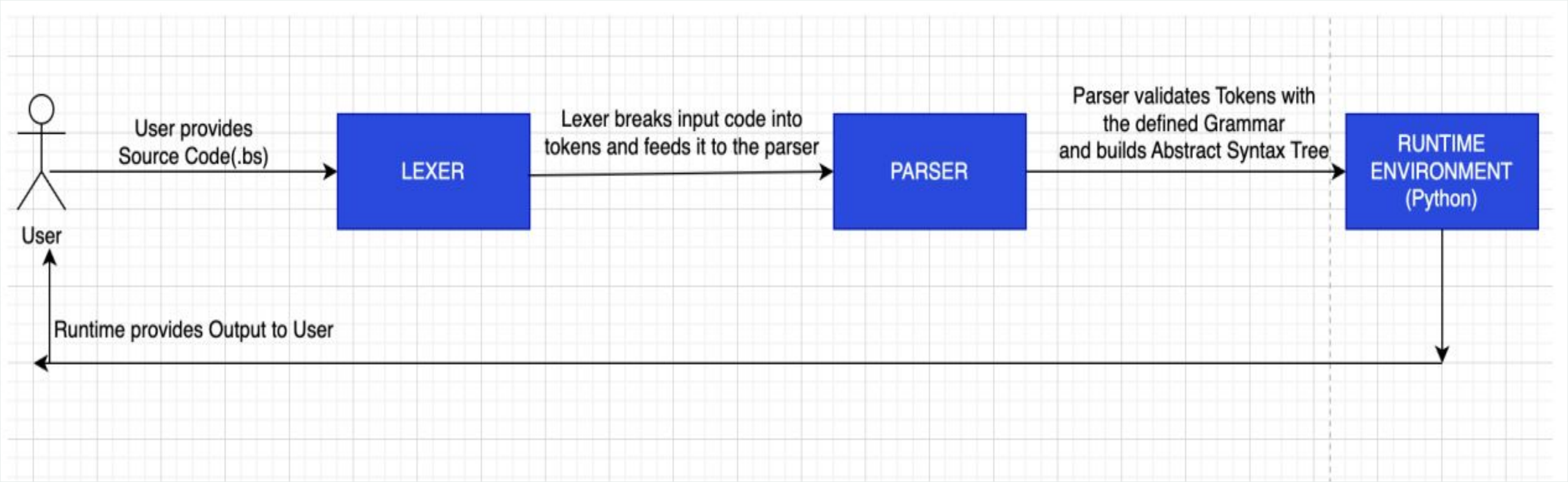
How the main components in Bliss work

1. Lexer Process:
   a. *Converts raw code into a stream of tokens using lexer.py.*
2. Parser Operations:
   a. *Takes the token stream from lexer.py.*
   b. *Uses grammar rules to parse tokens into complex structures (expressions, statements).*
3. AST Construction:
   a. *parser.py builds an Abstract Syntax Tree (AST) from tokens.*
   b. *Follows predefined grammar rules to ensure correct structure and syntax.*
   c. *AST represents the logical structure of the code.*

How the main components in Bliss work

1. Integration of Components:
    a. *Lexer, parser, and Interpreter (AST) work in unison.*
    b. *Lexer supplies tokens to the parser.*
    c. *Parser builds the AST using these tokens.*
    d. *AST is utilized for further processing (code execution, optimization, code generation).*

Design Diagram



Lexer

- Tokenization Overview
    - *Converts raw code into tokens using Lex.*
    - *These basic syntactic units would be further used for parsing*
- Lexical Analysis Process
    - *Analyzes code character-by-character.*
    - *Utilizes regular expressions to recognize token patterns.*
- Token Definitions
    - *Defines token types: keywords, operators, identifiers, data types, indents, etc.*
    - *Additional information such as value, line number, and, lexical number defined.*

Lexer example

- Sample Code

  X = 5

- Output

  LexToken(IDENTIFIER,'x',1,0)

  LexToken(ASSIGN,'=',1,2)

  LexToken(INTEGER,5,1,4)

Parser

- Grammar Rules in Parsing
  - *Defines how tokens combine into higher-level constructs: expressions, control flows, functions, etc.*
  - *Establishes token sequence and structure for valid code blocks.*
- Parsing Method
  - *Employs LALR (Look-Ahead Left-to-Right) parsing strategy.*
  - *Ensures correct syntax interpretation for complex constructs.*
- Error Handling in Parsing
  - *Detects syntax errors by recognizing misalignments with grammar rules.*
  - *Provides feedback or halts execution to prevent unexpected behavior.*

Parser example

```
def p_if_statement(p):
    """
    if_statement : IF expression COLON block ELSE COLON block
                 | IF expression COLON block
    """
    if len(p) == 8:
        p[0] = IfStatement(p[2], p[4], p[7])
    else:
        p[0] = IfStatement(p[2], p[4])
```

Interpreter

- AST Node Definitions
  - *Contains classes or structures for node types: expressions, statements, functions, loops, conditionals.*
  - *Each node type has specific attributes and behaviors aligned with the language's grammar.*
- Node Relationships
  - *Establishes connections among nodes to form a hierarchical AST structure.*
  - *Example: A function node might include nodes for parameters, body, and return type.*
- Tree Construction
  - *AST built from parser's output based on defined grammar rules.*
  - *Serves as an intermediate representation for further processing: code generation, optimization, or execution.*

Interpreter example

```python
class IfStatement(ASTNode):
    def __init__(self, condition, then_block, else_block=None):
        self.condition = condition
        self.then_block = then_block
        self.else_block = else_block

    def eval(self, context):
        if self.condition.eval(context):
            return self.then_block.eval(context)
        elif self.else_block:
            return self.else_block.eval(context)
```

Grammar

```
program : statements

statements : statements statement
| statement

statement : assignment
| expression
| control_flow
| function_definition
| PRINT LPAREN expression RPAREN

expression : expression PLUS expression
| expression MINUS expression
| expression TIMES expression
| expression DIVIDE expression
| expression MODULO expression
| expression GREATER_THAN expression
| expression LESS_THAN expression
| expression GREATER_EQUAL expression
| expression LESS_EQUAL expression
| expression EQUAL expression
| expression NOT_EQUAL expression
| expression AND expression
| expression OR expression
| LPAREN expression RPAREN
| INTEGER
| FLOAT
| IDENTIFIER
| STRING_LITERAL
| BOOLEAN
| MINUS expression
| NOT expression
| expression QUESTION expression COLON expression

assignment : IDENTIFIER ASSIGN expression
| IDENTIFIER ADD_ASSIGN expression
| IDENTIFIER SUB_ASSIGN expression
| IDENTIFIER MUL_ASSIGN expression
| IDENTIFIER DIV_ASSIGN expression
| IDENTIFIER MOD_ASSIGN expression

expression : LBRACKET list_elements RBRACKET

list_elements : list_elements COMMA expression
| expression

expression : expression LBRACKET slice RBRACKET

slice : expression COLON expression COLON expression
| expression COLON COLON expression
| expression COLON expression
| expression COLON
| COLON COLON expression
| COLON expression
| COLON
| expression

expression : RANGE LPAREN range_args RPAREN

range_args : expression COMMA expression COMMA expression
| expression COMMA expression
| expression

control_flow : if_statement
| while_statement
| for_statement

if_statement : IF expression COLON block ELSE COLON block
| IF expression COLON block

while_statement : WHILE expression COLON block

for_statement : FOR IDENTIFIER IN expression COLON block

block : INDENT statements OUTDENT

function_definition : DEF IDENTIFIER LPAREN params RPAREN COLON block

params : params COMMA IDENTIFIER
| IDENTIFIER
| empty

empty :

expression : IDENTIFIER LPAREN arguments RPAREN

arguments : arguments COMMA expression
| expression
| empty
```

Features of the Language

Data Types

1. Integer
   a. *5, 0, -5*
2. Float
   a. *1.0*
   b. *10e10*
   c. *6.74e-22*
3. String
   a. *“Hello, World!”*
   b. *‘Bye’*
4. Boolean
   a. *True, False*

Identifiers

In the Bliss programming language, identifiers serve as symbolic names that developers can assign to store and manipulate data during program execution. The naming conventions for identifiers in Bliss follow a specific set of rules to ensure consistency and readability.

Naming Rules:

- Variable names can start with either a lowercase or an uppercase letter.
- Variable names can contain alphanumeric characters (letters and digits) and underscores (_).
- Variable names are case-sensitive, meaning that myVariable and myVariable are treated as different identifiers.

Declaration and Assignment

In Bliss, variables do not need to be explicitly declared with a specific data type. Instead, the data type of a variable is dynamically determined by the value assigned to it, similar to how variables work in languages like Python.

Example

name = “John”

age = 30

isStudent = false

Operators

Arithmetic Operators:

- + (Addition)
- - (Subtraction)
- * (Multiplication)
- / (Division)
- % (Modulus)

Assignment Operators:

- = (Simple assignment)
- += (Addition and assignment)
- -= (Subtraction and assignment)
- *= (Multiplication and assignment)
- /= (Division and assignment)
- %= (Modulus and assignment)

Operators continued...

Comparison Operators:

- == (Equal to)
- != (Not equal to)
- > (Greater than)
- < (Less than)
- >= (Greater than or equal to)
- <= (Less than or equal to)

Logical Operators:

- and
- or
- not

Reserved Keywords and Built-in Functions

Reserved Keywords:

```
reserved = {
    "if": "IF",
    "else": "ELSE",
    "elif": "ELIF",
    "print": "PRINT",
    "int": "INTEGER",
    "str": "STRING",
    "for": "FOR",
    "while": "WHILE",
    "and": "AND",
    "or": "OR",
    "in": "IN",
    "range": "RANGE",
    "break": "BREAK",
    "continue": "CONTINUE",
    "def": "DEF",
    "not": "NOT",
}
```

Pre-defined functions:

- print()
- range()

Conditional Statements

Conditional Statements:

- if

```
if -y < 0:
    print(x+y)
```

- if…else

```
if i > 3:
    print("Value of i is greater than 3!")
else:
    print("Value of i is not greater than 3!")
```

Loops

- Traditional For Loop

```
for(x=0;x<10;x++):
    print(x)
```

- While Loop

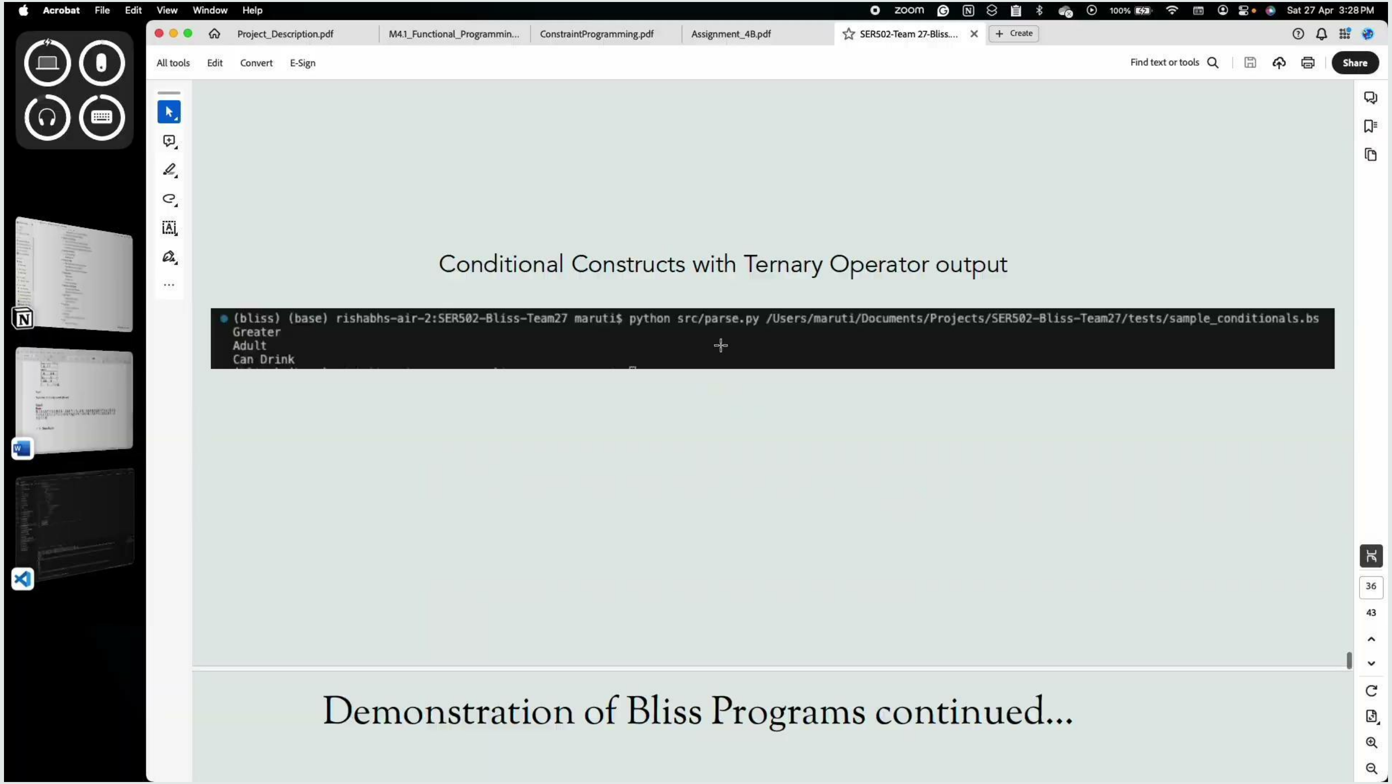
```
i = 1
while i < 3:
        print("Value of i is less than 3!")
        i = i + 1
```

- Advanced For Loop

```
for i in range(0,10,2):
    print(i)
```

Demonstration of Sample Programs

How to run a Bliss file:



Demonstration of Bliss Programs

Basic Arithmetic and Boolean Operations

tests > samplearith_bool.bs

```
# Demonstrating Boolean and arithmetic operations
x = 10
y = 20


sum = x + y
print(sum)
difference = y - x
print(difference)
product = x * y
print(product)
division = y / x
print(division)

is_greater = x > y
print(is_greater)
and_operation = (x < y) and (y > 10)
print(and_operation)
or_operation = (x > y) or (y == 20)
print(or_operation)
not_operation = not (x == y)
print(not_operation)
```

Demonstration of Bliss Programs continued…

Basic Arithmetic and Boolean Operations output

```
(bliss) (base) rishabhs-air-2:SER502-Bliss-Team27 maruti$ python src/parse.py /Users/maruti/Documents/Projects/SER502-Bliss-Team27/tests/samplearith_bool.bs
30
10
200
2.0
False
True
True
True
```

Demonstration of Bliss Programs continued...

String Operations and Assignments

tests > sample_string.bs

```
# String value assignments and printing
greeting = "Hello"
name = "World"
message = greeting + " " + name
print(message)
```

Output

```
(bliss) (base) rishabhs-air-2:SER502-Bliss-Team27 maruti$ python src/parse.py /Users/maruti/Documents/Projects/SER502-Bliss-Team27/tests/sample_string.bs
Hello World
```

Demonstration of Bliss Programs continued...

Conditional Constructs with Ternary Operator

tests > sample_conditionals.bs

```
# Using ternary operator and traditional if-then-else and just if
x = 5
result = x > 3 ? "Greater" : "Less or Equal"
print(result)

age = 20
if age < 18:
    print("Underage")
else:
    print("Adult")

age = 25
if age > 21:
    print("Can Drink")
```

Demonstration of Bliss Programs continued…

Conditional Constructs with Ternary Operator output

```
(bliss) (base) rishabhs-air-2:SER502-Bliss-Team27 maruti$ python src/parse.py /Users/maruti/Documents/Projects/SER502-Bliss-Team27/tests/sample_conditionals.bs
Greater
Adult
Can Drink
```

Demonstration of Bliss Programs continued...

Loops

tests > sample_loops.bs

You, 14 minutes ago | 2 authors (RishabhDoshi98 and others)

```
# Traditional for loop
for (i=0;i<5;i++):
    print(i)

# Traditional while loop
j = 5
while (j > 0):
    print(j)
    j = j - 1

# For loop with range similar to Python's range
for i in range(2, 5):
    print(i)

# Loop through the list and print each name
bliss = ["B", "L", "I", "S", "S"]
for i in bliss:
    print(i)

```

Demonstration of Bliss Programs continued...

Loops output

```
(bliss) (base) rspvarun97@rsps-air SER502-Bliss-Team27 % /Users/rspvarun97/miniconda3/envs/bliss/bin/python /Users/rspvarun97/Documents/ASU-Spring-2023/SER502/SER502-Final-Pr
oject-Bliss/SER502-Bliss-Team27/src/parse.py /Users/rspvarun97/Documents/ASU-Spring-2023/SER502/SER502-Final-Project-Bliss/SER502-Bliss-Team27/tests/sample_loops.bs
0
1
2
3
4
5
4
3
2
1
2
3
4
B
L
I
S
S
(bliss) (base) rspvarun97@rsps-air SER502-Bliss-Team27 %
```

Demonstration of Bliss Programs continued...

Fibonacci

tests > sample3.bs

You, 14 seconds ago | 2 authors (sramara6 and others)

```
def Fibonacci(n):
    if n < 0:
        result = "Invalid input"
    else:
        if n == 0:
            result = 0
        if n == 1:
            result = 1
        if n > 1:
            a = 0
            b = 1
            for i in range(2, n + 1):
                c = a + b
                a = b
                b = c
            result = b

    print(result)

Fibonacci(7)
Fibonacci(0)
Fibonacci(5)
```

You, 1 second ago • Uncommitted changes

We have also implemented the creation of custom functions by using the 'def' keyword

Output

```
(bliss) (base) rishabhs-air-2:SER502-Bliss-Team27 maruti$ python src/parse.py /Users/maruti/Documents/Projects/SER502-Bliss-Team27/tests/sample3.bs
13
0
5
```

Demonstration of Bliss Programs continued...

Another program using 'def'

tests > sample4.bs

You, 8 minutes ago | 2 authors (You and others)

```
def rowspattern(rows):
    for i in range(rows):
        line = ""  # Initialize an empty string for each row
        for j in range(i + 1):
            line += "* "  # Concatenate stars with a space
        print(line)

rowspattern(10)      You, 8 minutes ago • Uncommitted changes
```

Output

```
(bliss) (base) rishabhs-air-2:SER502-Bliss-Team27 maruti$ python src/parse.py /Users/maruti/Documents/Projects/SER502-Bliss-Team27/tests/sample4.bs
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
```

Future Scope

Future Scope

1. **Performance Optimization:** We plan to refine the compiler to increase execution speed and improve memory efficiency for complex applications.
2. **Error Handling Enhancements:** Future versions will focus on enhancing error detection and providing more descriptive error messages to ease debugging.
3. **Support for Additional Data Types:** We aim to introduce more data types like sets, dictionaries, etc., to broaden the language's applicability.
4. **Graphical Execution Demonstrations:** We plan to improve the interpreter to visually demonstrate execution processes and aid in understanding and debugging code flow through graphical displays.

Thanks!