SER502 - Emerging Languages and Programming Paradigms
Milestone - 1
Team 21

Aum Garasia
Samit Nikesh Shah
Shubham Chetan Shah
Siddesh Shetty

Project Description:

Language Name: EaZy (file extension = .ez)
GitHub Repository: Here

It is no secret that python is one of the most popular programming languages out there. What new improvements can be made to this language? What if that language was just, EaZy (not a good pun, but okay ;)). Our aim is to build on the already existing user-friendly nature of python and add minor changes on top of it.

Parsing Technique: The parser is going to be implemented using DCG rules using python3. We aim to build the parse tree recursively and check its validity in similar fashion as prolog.

Compiler / Interpreter runtime: The runtime environment is also going to be implemented in python3.

Data Structures: As of now, we are planning to incorporate dictionaries and/or hash maps for easier lookup. For Storing tokens, an array could be used.

Language Design:

Declarations :

```
char x;
bool axk23 = true;
integer ine = 2*3+4;
string str = “abcdes23431”
```

Loop :

```
for_Loop(i = 1; i< 10; i=i+1;) { z = z + 2; }
for_Loop(i in range(0,9)) { z = z + 2; }
while_Loop ( x \< 4 ) { z = z + 2; }
```

If- else:

```
If true do x=x+1;
If false do x=x-1; else do x=x+1;
If x \=3 ? z=z+2; : z=z+4;
```

Sample Program

```
begin
int x := 3;
while x \< 4 AND x \> 2:
display “hello world”;
x := x + 1;
fin
```

Language Grammar:

1. Tokens:

<Num> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.
<LChar> := a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z.
<UChar> := A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z.
<Space> := “ “ (Inverted Commas are not a part of <Space>)

2. Arithmetic Expressions:

<Arithmetic-Expr> := <Num> + <Num> | <Num> - <Num> | <Num> * <Num> | <Num> / <Num> | <Num> mod <Num> | (<Arithmetic-Expr>) | <Identifier> = <Num>.

3. Boolean Grammar:

<Bool> := true | false.
<Bool> := <Arithmetic-Expr> == <Arithmetic-Expr> | <Arithmetic-Expr>. (If <Arithmetic-Expr evaluates to zero : false, otherwise true)
<Bool> := <Var-Name> \< <Num> | <Var-Name> \< <Num> | <Var-Name> \= <Num>
<Bool> := <Bool> | <Bool> AND <Bool> | <Bool> OR <Bool>.
<Bool> := NOT <Bool>.

4. Looping Grammar:

<Statement> := for_Loop (<Statement> ; <Bool>; <Statement>;) {<Statement>;}
<Statement> := for_Loop (<Var-Name> in range (<Num>, <Num>)) {<Statement>;}
<Statement> := while_Loop (<Bool>) {<Statement>;}

5. Strings Grammar:

<Char> := <LChar> | <UChar> | <Num>
<String> := “, <Char>,” | “,<Char>, <Char-List>,” (Inverted Commas are part of the grammar)
<Char-List> := <Char> | <Num>
<Var-Name> := <LChar> | <Char-List>

6. Declarations:

<DataType> := bool | str | char | int.
<Dec> := <DataType> <Var-Name> | <DataType> <Var-Name> := <Value>
<Val> := <Bool> | <String> | <Char> | <Num>
<Dec> := <Var-Name> := <Value>
<Dec> := <Dec>; <Dec>

7. Conditionals:

<If> := if <Bool> do <Statement>; | if <Bool> do <Statement>; else do <Statement>;
<if> := if <Bool> ? <Statement>; : <Statement>;

8. Comments:

<Comment> := @ <Char-List> @

9. Print:

<Statement> := display <Var-Name> | display <Val> | display <String>

10. Program:

<Prog> := Begin (<Statement>)* Fin
<Statement> := <Statement>;<Statement>