Poly Overview

- Polynomial manipulation language
- Implements a polynomial data type
- Supports mathematical operations on polynomials

Goals

- Simple language
- Easy to use
- Create constructs for operations specific to polynomials

Polynomial Operators

- Addition, Subtraction, Negation
- Multiplication (not between two polynomials)
- Division (only by a number)
- Concatenation

Polynomial Operators

- Order
- Coefficient
- Assignment
- Logical Operators (and, or, not)
- Relational Operators (==, !=, >, <)

What's POLY made of?

1) Lexical constructs
2) Types
3) Expressions
4) Statements
5) User-defined functions
6) Internal Functions

Lexical Constructs

- // - Single line comment
- Identifiers
- Keywords - if, else, while, and, or, not, prototype, function, return
- Numbers – int, float
- Polynomials – [1,2,3]
- Strings

Lexical Constructs

- Other tokens

{ } ( ) [ ] , ; + - *
/ % = > < ==

!= :

Types

- int
- float
- poly
- string constants
- functions

Expressions

- Primary Expressions
- Arithmetic Expressions
- Relational Expressions
- Logical Expressions

Primary Expressions

- Identifiers
- Constants
- Function Calls – funcname(arg1, …, argn)
- Polynomial co-efficient extraction – polyVar[i]
- Polynomial concatenations – polyA:polyB

[1,2,3]:[4,5,6] ==> [1,2,3,4,5,6]

- Order of polynomial - |polyVar|
- Parentheses – (expr)

Arithmetic Expressions

- Unary minus “–” -var
- Multiplicative operators – *,/,%
- Additive operators - +,-

Relational Expressions

- Equals ==
- Not equals !=
- Greater than >
- Less than <

Logical Expressions

- not – not(expr)
- and – expr1 and expr2
- or – expr1 or expr2

Statements

- Assignment ID=expr
- Variable declarations type var = expr;
- Conditional statements

if(logical expr){ statements }

if(logical expr){ statements }

else {statements}

- Iterative statements

while(logical expr){ statements }

- Return statements

return expr;

return;

Functions

- **Function prototypes**

prototype typename funcname (argType1, argType2, …, argTypen)

- **Function declaration**

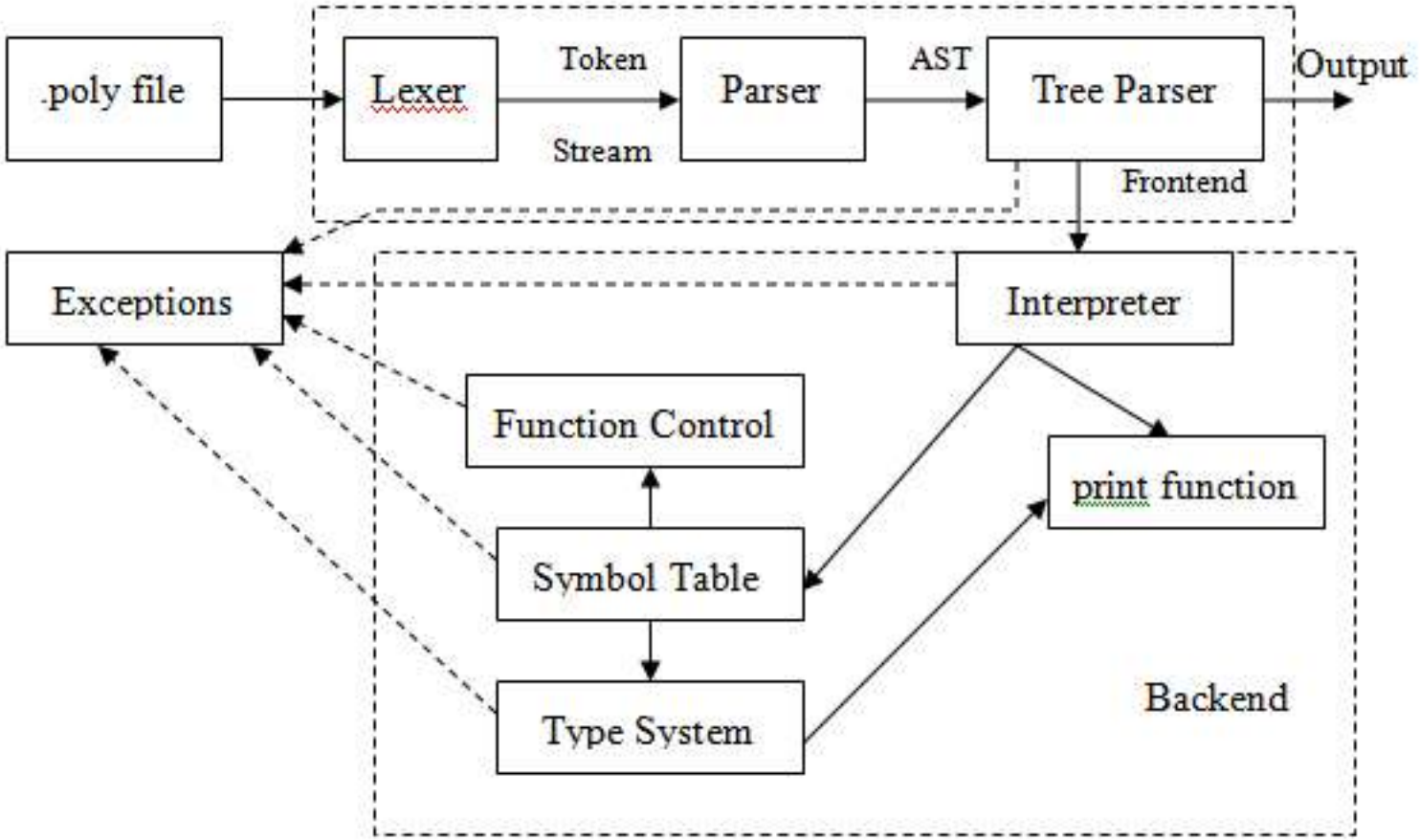
function typename funcname (type1 arg1, …, typen argn) {statements}

Internal Functions

- **print function**

print strORexpr1, strORexpr2, …, strORexprn

Architecture Diagram



Front-end

- Grammar.g
  - Lexer
  - Parser
  - Independent
- Walker.g
  - Static + semantic walking
  - Execution
  - Simultaneous

The Walker

1) Coefficient only called on poly with int

2) Function exists on invocation

3) Function indeed returns var; var is correct type

4) Declare only if not already locally declared

5) When retrieving variable value, check existence

Ex: Coefficient Replacement

```
prototype poly coeffAssign(poly, int, float);
function poly coeffAssign(poly A, int x, float p) {
  int y = x+1;
  int z = |A|;
  poly B = [0];
  while(not (z<y)){
    B = A[z]:B;
    z=z-1;
  }
  B = p:B;
  z = z-1;
  while(not (z<0)) {
    B = A[z]:B;
    z=z-1;
  }
  return B;
}

poly A = [0,1,2,0,3,0,4,5,0];
float x = -99.0;
print "Replacing coefficient:", A[4], coeffAssign(A, 4, x);
```

Execution

- Interpreter
  - addProto(rettype, name, args[])
  - funcRegister(rettype, name, args[], varnames[], AST)
  - funcInvoke(walker, func, params[])
  - assign(a,b)
- Symbol Table
  - getParent()
  - containsVar(name, enable_global)
  - getVar(name)
  - setVar(name, data)

Data Types

- General
  - setName(string)
  - getName()
  - print()
  - boolVal()
  - concat(PolyDataType)
- Specific
  - mod(PolyDataType)
  - coeff(PolyDataType)
  - simplify()

Testing Plan

- Initial stages
  - Elaborate files
  - No regression suite
- After AST generation
  - Evolving tests
  - Formal regression testing

Test Suite

- Two Modes
  - make test
    - Run until failure
    - Output detailed failure information
  - make testall
    - Success or failure for each test

Test Programs

- Slowly increasing difficulty
  - Declarations
  - Assignments
  - Expressions
  - Functions
  - Recursion

Example Test 1

```
int a = 1*4;
float b = 6*4;
float c = 5*5.3;
float d = 6.7*3;
float e = 2.9*6.3;
poly f = [1,2,3]*4;
poly g = 5*[6,7,8,9];
poly h = [2,3,4]*5.67;
poly i = 98.7*[6,5,43,2,1];

print a;
print b;
print c;
print d;
print e;
print f;
print g;
print h;
print i;
```

Lessons Learned

- Easy stuff at home
- Build incrementally
- Lots of coding
- Learn ANTLR
- Coding style
- Control flow is hard
- KISS