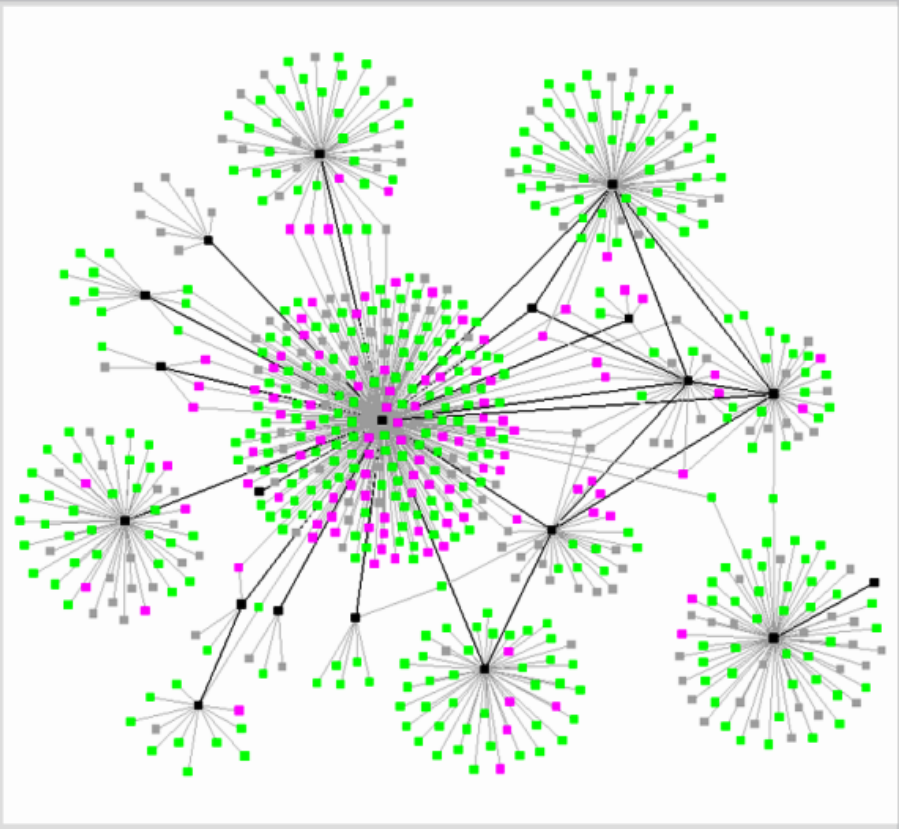
GPL: GPL is a Programming Language

Jacob Jensen - Language Guru
Kritika Kaul - System Tester
Mohan Kolli - Team Lead
Nitin Natarajan - System Integrator
Yufei Liu - System Architect

Why Graphs?

- Many algorithms are based on graphs – shortest path, breadth first search, topological sort
- Real-world networks are interesting objects of study in themselves
- Many languages can deal with graphs reasonably well, but often in an inflexible or cumbersome way.



What is GPL?

- General-purpose graph programming language
- Make graphs simple to work with
  - Manipulate data naturally in an adjacency list setting
  - Many common algorithms built in: no hassle
  - Interpreted for ease of use
- To graphs what MATLAB is to matrices

Features of GPL

- Interpreter based
- Procedural
- Weakly-typed
- Visualizable
- Easy to use
- Enforces Good Programming Practices
- Awesome

Syntax

- Simple: Nothing too crazy
- Spare: Skip the curly brackets
- Intuitive: Not hard to figure out
- Traditional: Stick to standards of other interpreted languages

Graph-Centric

- Three atomic data types
  - String: Symbols are important
  - Number: Quantitites are important
  - Graph: Collections of things and their binary relationships are most important!
- Graph has a set of Nodes and Edges
  - Encapsulated: No unsupervised access
  - Manipulate from Graph level, for simplicity and safety
- Graph can be visualized
  - Beauty of Swing representation unparalleled
  - 3D... representation of a perfectly flat surface

Built-in Libraries

- Graph Algorithms (CLRS style)
  - MST, BFS, BFS with target node, DFS
- Network Algorithms
  - PageRank
  - Matrices and Vectors that can manipulate stored quantities

Development Environment

- Eclipse: IDE for Java backend development
- 'Liu'Lex: custom-written lexical analyzer
- Jacc: Java-native version of yacc
- Perl: Interpreted Shell

Say Hello to Graph World?

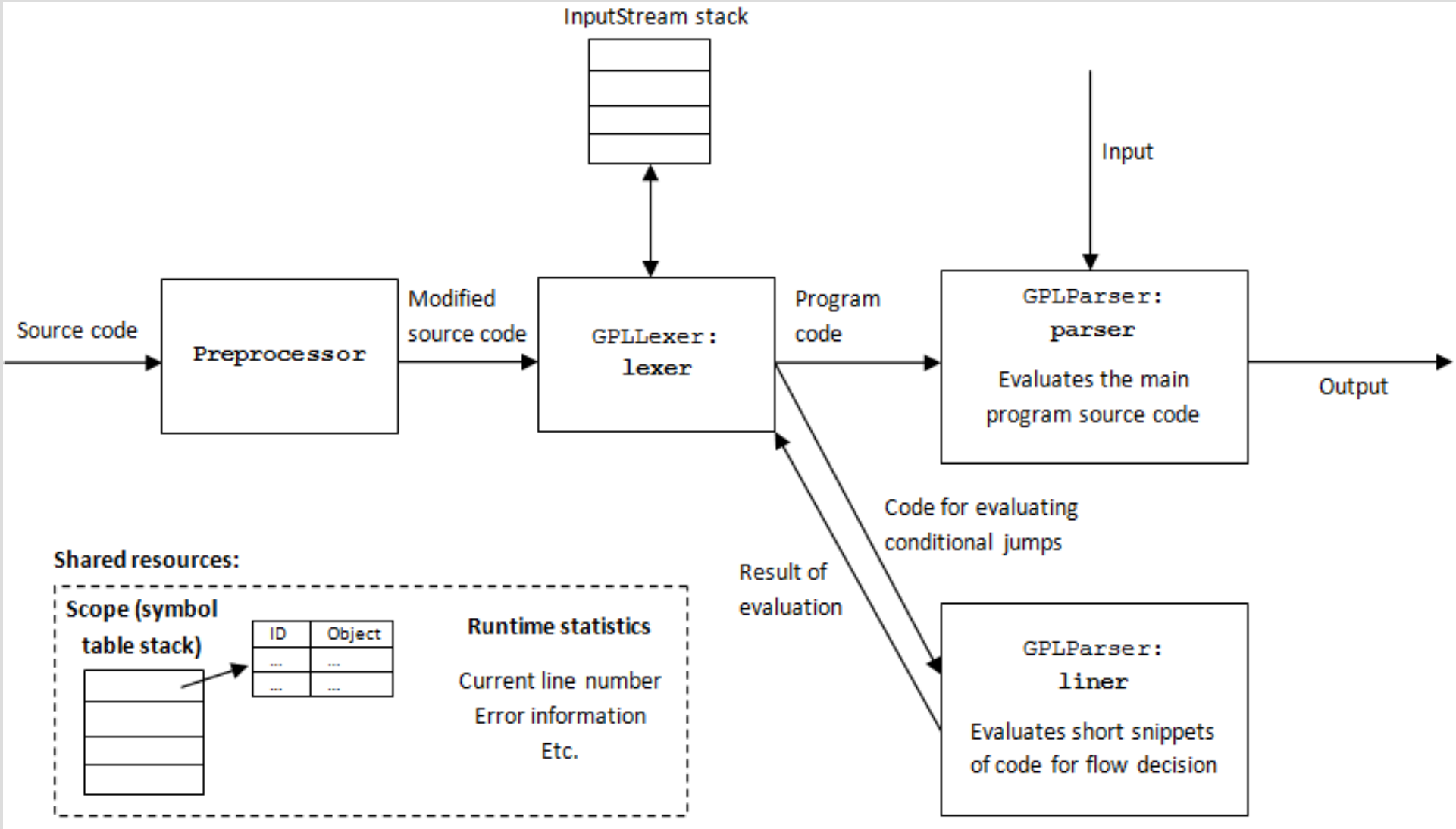
```
Graph h = ["h", "e", "l", "l", "o"]
String s = h.printNodes()
print(s)
```

Output:
h e l l o

Sample Code

```
def printNumber(Number n)
    print("Number is passed to printNumber ")
    print(n)
end

Number i = 2
Number j = 2
while (i>0)
    printNumber(i)
    i = i-1
end

i = 4

if (i%2 == 0)
    print("i is even ")
elsif (i%3 == 0)
    print("i is divisible by 3 ")
else
    print("i is neither even nor divisible by 3 ")
end

Graph h = ["h", "e", "l", "l", "o"]
print(h.printNodes())

h.addEdge(1,2,1)
h.addEdge(1,3,1)
h.addEdge(1,4,1)
h.addEdge(1,5,1)
h.addEdge(3,4,1)
h.addEdge(4,2,1)

h.bfs(1)
h.dfs(1)
h.topologicalSort()
```

Translator Architecture



Integration and Testing

- Unit tested every back end method.
- Every time a method was checked in or any change was made, we made it a point to run the tests so that the application is always in a stable state.
- Integration with the parser was made due to our script which enabled us to test the back end with the front end and run our test suite.
- We also had a command line interpreter to quickly check the state of the parser and how it behaves.