GDL
(pronounced goodəl)

Goodle Manager – Lindsey Heller
Goodle Guru – Joseph Corbisiero
Goodle Architect – Ilan Elkobi
Goodle Integrator – Henrique Maia
Goodle Tester – Elayna Tuck
Why GDL… makes programming decision trees easy!
Buzz Words

- Flexible
- Familiar
- User-Friendly / Easy to Use
- Useful
Target

- Professionals
- Students
- Publications
# Syntax: Keywords

<table>
<thead>
<tr>
<th>Similar to Java &amp; C</th>
<th>GDL Specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>for</td>
<td>begin</td>
</tr>
<tr>
<td>while</td>
<td>graph</td>
</tr>
<tr>
<td>if</td>
<td>state</td>
</tr>
<tr>
<td>else</td>
<td>start</td>
</tr>
<tr>
<td>return</td>
<td>accept</td>
</tr>
<tr>
<td>true/false</td>
<td>func</td>
</tr>
<tr>
<td></td>
<td>goto</td>
</tr>
</tbody>
</table>
## Syntax : Primitive Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>a Java String</td>
</tr>
<tr>
<td>number</td>
<td>the Java primitive double</td>
</tr>
<tr>
<td>bool</td>
<td>the Java primitive boolean</td>
</tr>
</tbody>
</table>
Syntax: Conditionals

- while
- for
- if/else
- goto

```plaintext
goto, {list_states}, condition;
```
Syntax: Graphs

- begin

  ```java
  begin( )
  {
    //states and functions
  }
  ```

- graph

  ```java
  graph <name>( )
  {
    //states and functions
  }
  ```
Syntax : States

- **start** = the start state of a graph

```
start <name>( )
{
    //actions
}
```

- **accept** = accepting state of a graph

```
accept <name>( )
{
    //actions
}
```

- **standard** = any state that is neither the start nor accept stat of a graph

```
state <name>( )
{
    //actions
}
```
func return_type : <name> (parameter_list)
{
    //actions
}

Syntax : Function Declaration
Lexical Analyzer

- Returns Tokens
- Keywords of the language

```c
/* CONTROLS */
{IF} { return Parser.IF; }
{ELSE} { return Parser.ELSE; }
{FOR} { return Parser.FOR; }
{WHILE} { return Parser.WHILE; }
{DO} { return Parser.DO; }
```
Using BYAAC/J

Creates an AST

```plaintext
stmt : decl SEMI {$$ = new Node(State.STMT, $1); }
     | stmt_assign SEMI {$$ = new Node(State.STMT, $1); }
     | begin {$$ = new Node(State.STMT, $1); }
     | graph_closure {$$ = new Node(State.STMT, $1); }
     | state_closure {$$ = new Node(State.STMT, $1); }
     | start_closure {$$ = new Node(State.STMT, $1); }
     | accept_closure {$$ = new Node(State.STMT, $1); }
     | goto_stmt SEMI {$$ = new Node(State.STMT, $1); }
     | if_stmt {$$ = new Node(State.STMT, $1); }
     | while_loop {$$ = new Node(State.STMT, $1); }
     | for_loop {$$ = new Node(State.STMT, $1); }
     | func {$$ = new Node(State.STMT, $1); }
     | func_call SEMI {$$ = new Node(State.STMT, $1); }
     | return_stmt {$$ = new Node(State.STMT, $1); }
     | NL { /* Nothing to do */ }
     | SEMI { /* Nothing to do */ }
;```

Semantic Analyzer

=  

A  /  

+  D  

*  B  C  

-  E  F  

Semantic Analyzer
public class GDLMain {
    HashMap<String, String> closedList = new HashMap<String, String>();
    HashMap<String, AbstractState> allStatesTable = new HashMap<String, AbstractState>();
    public GDLMain() {
        allStatesTable.put("beginStart", new beginStart());
        allStatesTable.put("begin_S1", new begin_S1());
        allStatesTable.put("begin_S2", new begin_S2());
        allStatesTable.put("terminalAccept_begin_acc", new terminalAccept_begin_acc());
    }
    public static void main(String[] args) {

        GDLMain gdl1 = new GDLMain();

        gdl1.runGraph();
    }
GUI Graph

creates DOT file to produce a graph
Execution Environment

• Makefile calls:
  • Lex and Yacc
    • Creates Parser
      • Parser generates files in output folder
        • Helper classes are compiled and used
          • GDLMain is created with user program
            • Program executes on terminal
              • GDL_graph.dot is generated
User sees the results of accept states immediately after running the program

A dot file is generated so the average user can better understand the results of the graph

It’s Useful!

GDL has already been put to use for one of our AI projects this year and is currently being used by two team members to create FSM for Embedded Architectures!
Development Tools

BYACC/J

graphviz

JFlex

git

T

Vim

>
Management Tools

# groupme

github

Google Drive

Google groups

Scrum
Efficiency Tools
## Sprint Schedule

<table>
<thead>
<tr>
<th>Sprint Dates</th>
<th>Sprint Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 6\textsuperscript{th} – March 9\textsuperscript{th}</td>
<td>Sprint 0</td>
</tr>
<tr>
<td>March 10\textsuperscript{th} – March 16\textsuperscript{th}</td>
<td>Sprint 1</td>
</tr>
<tr>
<td>March 17\textsuperscript{th} – March 30\textsuperscript{th}</td>
<td>Sprint 2</td>
</tr>
<tr>
<td>March 31\textsuperscript{st} – April 14\textsuperscript{th}</td>
<td>Sprint 3</td>
</tr>
<tr>
<td>April 15\textsuperscript{th} – April 27\textsuperscript{th}</td>
<td>Sprint 4</td>
</tr>
<tr>
<td>April 20\textsuperscript{th} – May 11\textsuperscript{th}</td>
<td>Sprint 5</td>
</tr>
</tbody>
</table>
Testing

- Unit Testing
- Black Box Testing
- Regression Testing
Testing

- Tested Parser
- Tested Semantics
- Tested general functionality
We learned our lesson...

- ensure all are using the same version
- always pull before you commit
- communication is key

- test
- test the test
- test the test that tested the test
Questions?