knode
a graph-based language

Project Manager: Krista Kohler
Language Guru: Jon Jia
System Architect: Jonathan Balsano
System Integrator: Maria Moresco
System Tester: Ruby Robinson

Imagine...
- You are a non-programmer who works with lots of data.
- You like the intuitive look of Python, but you don't like performance issues.
- You need the speed and efficiency of C without the headaches of pointers and memory management.

The Problem
- In our world, there is a LOT of data
- This data does not exist in a vacuum
- As our lives becomes more social, there's a greater need to access, share, and analyze interrelated data
- But data is not always easy to manage, especially in large quantities

The Solution: knode
- Declarative language focused on convenience
- Pretty syntax, snappy performance
- Speed of C, but memory managed
- Minimal programming experience required
- Built-in graph primitives for easy data viewing and manipulation

knode is...

A knode program

[illegible]

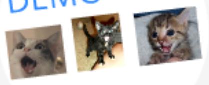
Node declarations

Edge declarations

String concatenation

Key Syntax

DEMO TIME



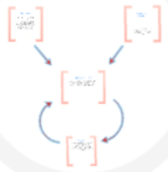
Project Management
- Traditional, five-phase development process
- Emphasis on collaboration and flexibility
- Weekly meetings: planning and working
- Asana for outline, task management, and deadlines
- "First-come, first-serve" assignment of responsibilities

Development
- Lex
- Yacc
- UTHash

Translator Architecture

Runtime Environment
- Output by GCC in machine code
- Just type ./a.out in *nix shell
- Output goes to stdout
- Can be used in combination with shell tools

Testing



Conclusions
- Working on a group project has its challenges
- Cat GIFs make late-night pull requests fun!
- Great people, great product
- Knode is highly capable and easy to use
- Low level efficiency, higher level coding
- Manage interrelated data easily
- Capable, convenient, and cross-platform

knode

a graph-based language

Project Manager: Krista Kohler

Language Guru: Jon Jia

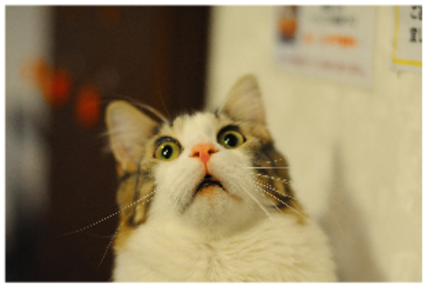
System Architect: Jonathan Balsano

System Integrator: Maria Moresco

System Tester: Ruby Robinson

The Problem

- In our world, there is a LOT of data
- This data does not exist in a vacuum
- As our lives becomes more social, there's a greater need to access, share, and analyze interrelated data
- But data is not always easy to manage, especially in large quantities



Imagine...

- You are a non-programmer who works with lots of data.
- You like the intuitive look of Python, but you don't like performance issues
- You need the speed and efficiency of C without the headaches of pointers and memory management

The Solution:

knode

- Declarative language focused on convenience
- Pretty syntax, snappy performance
- Speed of C, but memory managed
- Minimal programming experience required
- Built-in graph primitives for easy data viewing and manipulation

knode is...

user-friendly

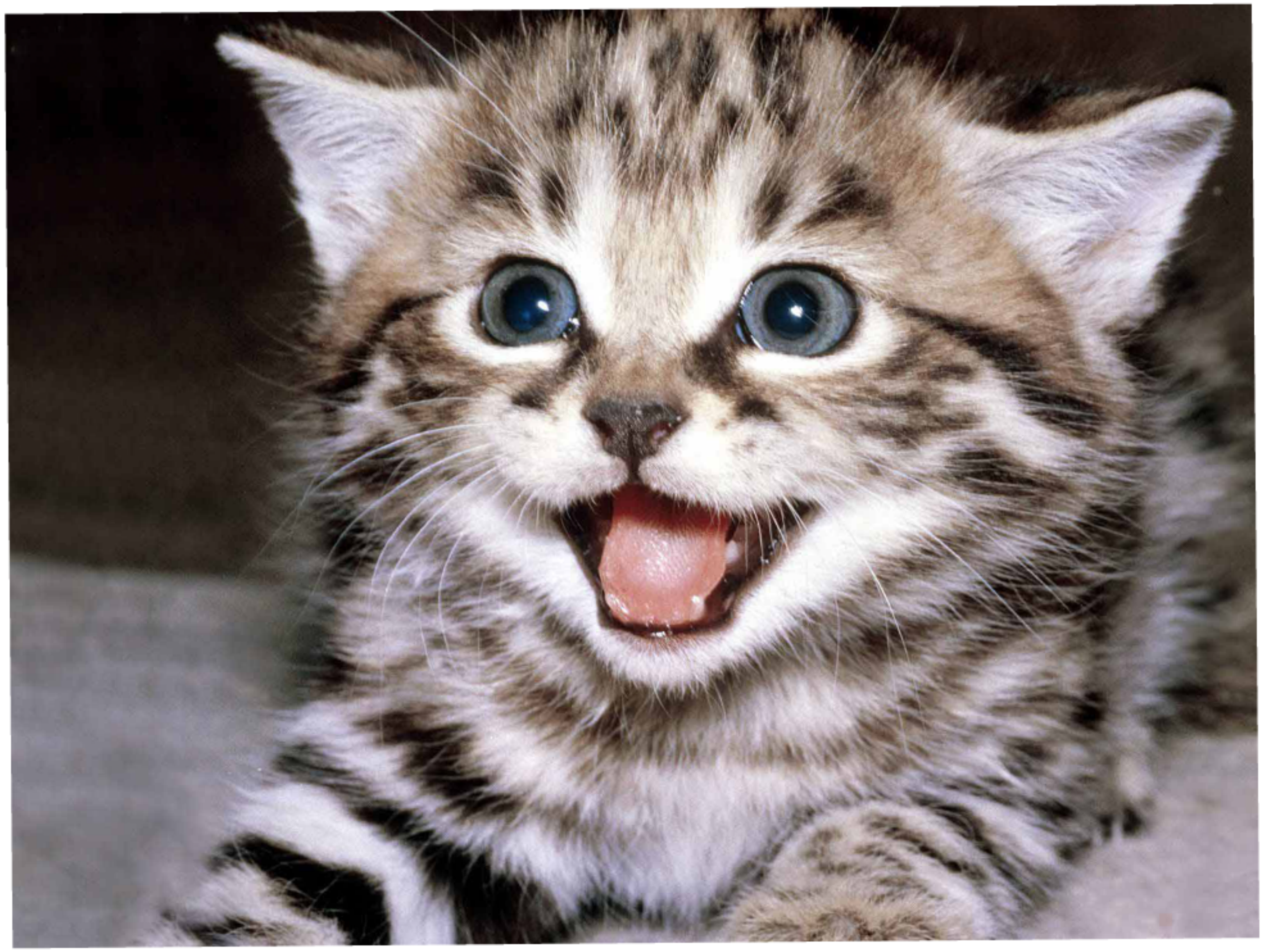
convenient

fast

cross-platform

user-friendly

convenient



fast

cross-platform

knode is...

user-friendly

convenient

fast

cross-platform

Key Syntax

whitespace

- Two spaces used to denote blocks
- Used for:
  - Function definitions
  - Flow control
  - Certain declarations

```
main():
  node a
    role: "mammal"
  node b
    role: "dogs"
  node c
    role: "cats"
```

node

- Nodes are declared with keyword
- New block after declaration sets node data as dict

```
node a
  role: "mammal"
node b
  role: "dogs"
node c
  role: "cats"
```

dictionary

- Keys and values are separated by colon
- Values must be strings

```
node a
  role: "mammal"
node b
  role: "dogs"
node c
  role: "cats"
```

edge

- Edges set as relation to two nodes
- Specified by ->, <-, <->
- Type of edge set with type keyword

```
edge p1 = [a->b]
edge p2 = [b->c]
edge p3 = [a->c]
p1.type = " type of "
p2.type = " type of "
p3.type = " hate "
```

string concatenation

- String concatenation is done with +
- Strings, ints, doubles can be all concatenated
- Create new string or print directly

```
print (b.role + " are a" + p1.type + a.role)
print (c.role + " are a" + p2.type + a.role)
print (b.role + p3.type + c.role)
```

Memory Management

- "Memory management's a real b****"

whitespace

- Two spaces used to denote blocks
- Used for:
  - Function definitions
  - Flow control
  - Certain declarations

```
main():
  node a
    role: "mammal"
  node b
    role: "dogs"
  node c
    role: "cats"
```

node

- Nodes are declared with keyword
- New block after declaration sets node data as dict

```
node a
  role: "mammal"
node b
  role: "dogs"
node c
  role: "cats"
```

dictionary

- Keys and values are separated by colon
- Values must be strings

```
node a
  role: "mammal"
node b
  role: "dogs"
node c
  role: "cats"
```

edge

- Edges set as relation to two nodes
- Specified by ->, <-, <->
- Type of edge set with type keyword

```
edge p1 = [a->b]
edge p2 = [b->c]
edge p3 = [a->c]
p1.type = " type of "
p2.type = " type of "
p3.type = " hate "
```

string concatenation

- String concatenation is done with +
- Strings, ints, doubles can be all concatenated
- Create new string or print directly

```
print (b.role + " are a" + p1.type + a.role)
print (c.role + " are a" + p2.type + a.role)
print (b.role + p3.type + c.role)
```

Memory Management

- "Memory management's a real b****"

A knode program

```
main():
  node a
    role: "mammal"
  node b
    role: "dogs"
  node c
    role: "cats"
```

Node declarations

```
  edge p1 = [a->b]
  edge p2 = [b->c]
  edge p3 = [a->c]
  p1.type = " type of "
  p2.type = " type of "
  p3.type = " hate "
```

Edge declarations

```
  print (b.role + " are a" + p1.type + a.role)
  print (c.role + " are a" + p2.type + a.role)
  print (b.role + p3.type + c.role)
```

String concatenation

A knode program

```
main():
  node a
    role: "mammal"
  node b
    role: "dogs"
  node c
    role: "cats"
```

Node declarations

```
  edge p1 = [a->b]
  edge p2 = [b->c]
  edge p3 = [a->c]
  p1.type = " type of "
```

Edge declarations

```
node b
  role: "dogs"
node c
  role: "cats"
```

Node declarations

```
edge p1 = [a->b]
edge p2 = [b->c]
edge p3 = [a->c]
p1.type = " type of "
p2.type = " type of "
p3.type = " hate "
```

Edge declarations

```
print (b.role + " are a" + p1.type + a.role)
print (c.role + " are a" + p2.type + a.role)
print (b.role + p3.type + c.role)
```

Prezi

String concatenation

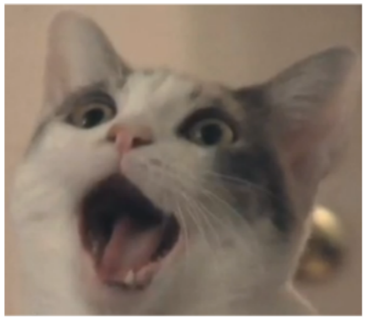
```
edge p2 = [b->c]
edge p3 = [a->c]
p1.type = " type of "
p2.type = " type of "
p3.type = " hate "
```

Edge declarations

```
print (b.role + " are a" + p1.type + a.role)
print (c.role + " are a" + p2.type + a.role)
print (b.role + p3.type + c.role)
```

String concatenation

DEMO TIME



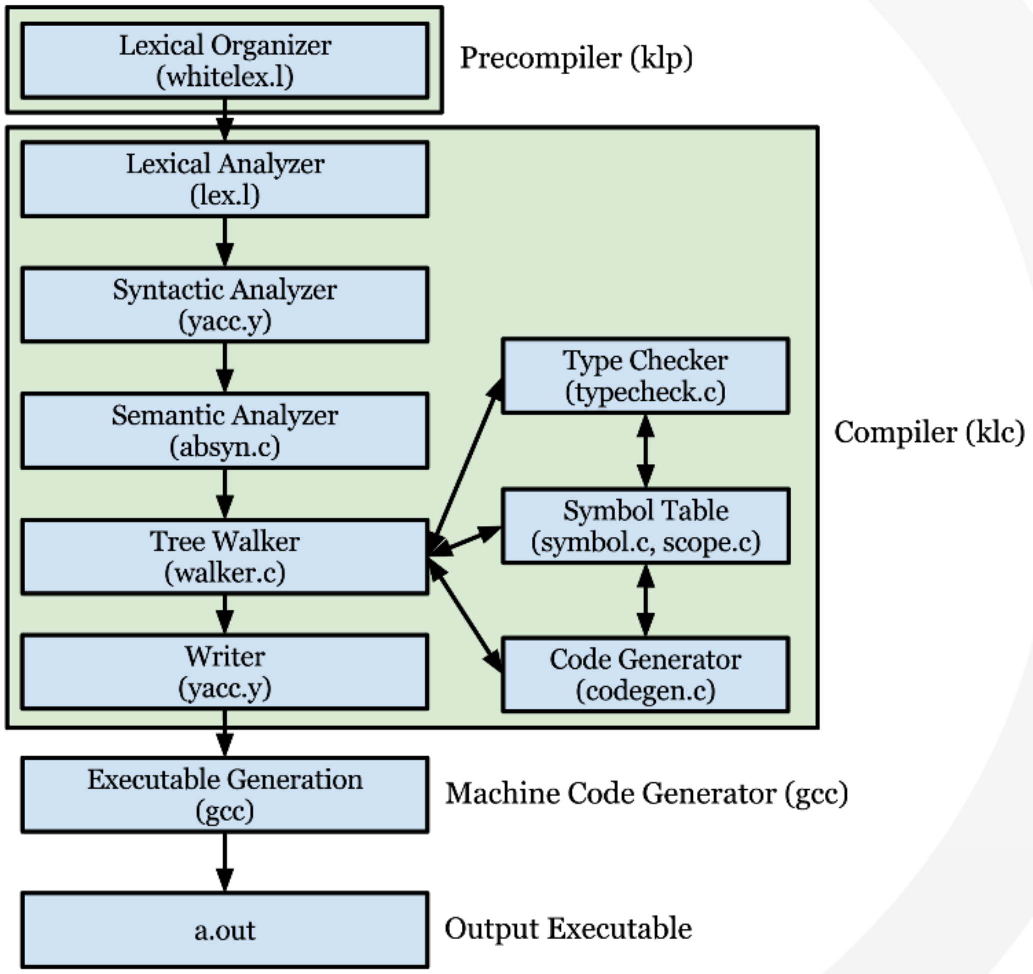
Project Management

- Traditional, five-phase development process
- Emphasis on collaboration and flexibility
- Weekly meetings: planning and working
- Asana for outline, task management, and deadlines
- "First-come, first-serve" assignment of responsibilities

Development

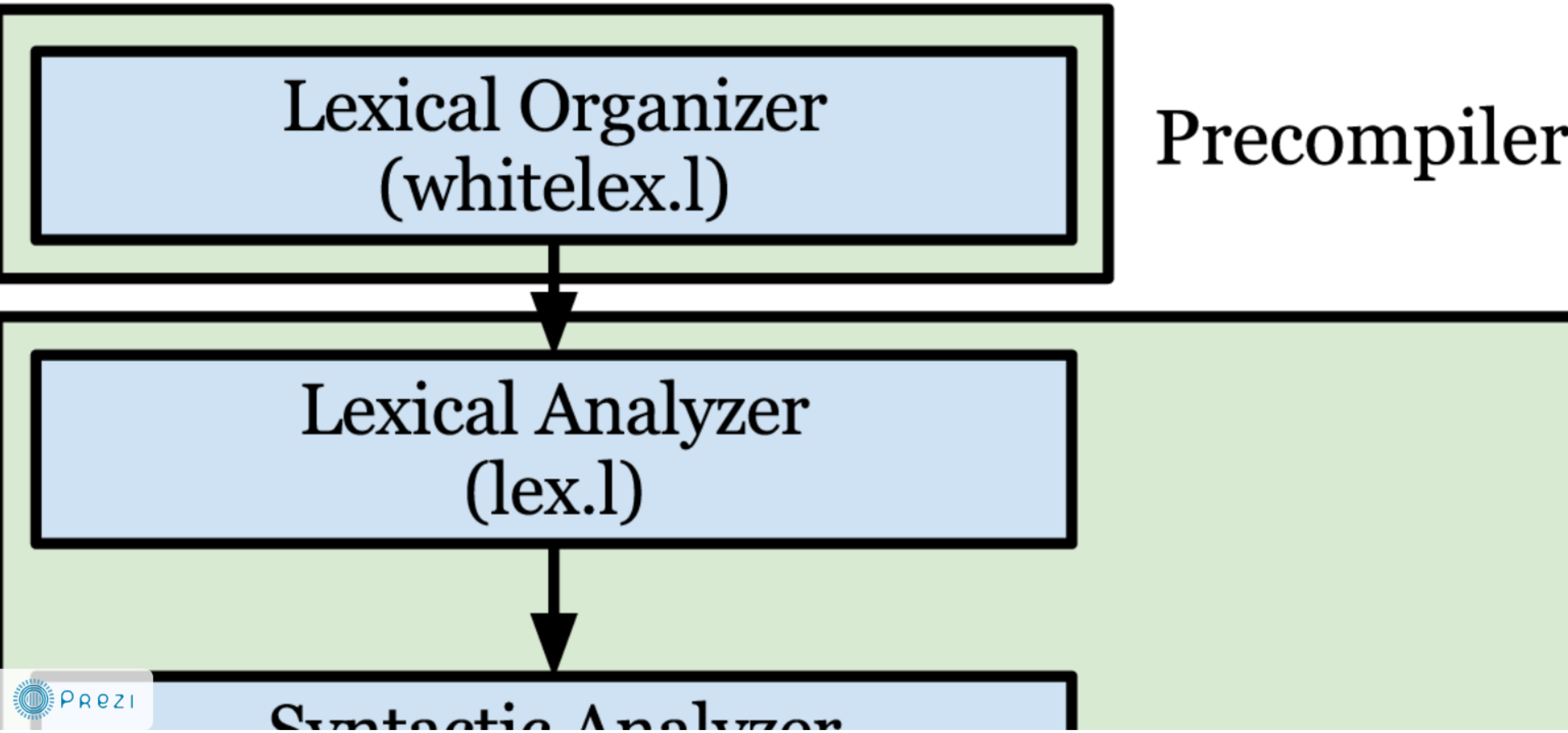
- Lex
- Yacc
- UTHash

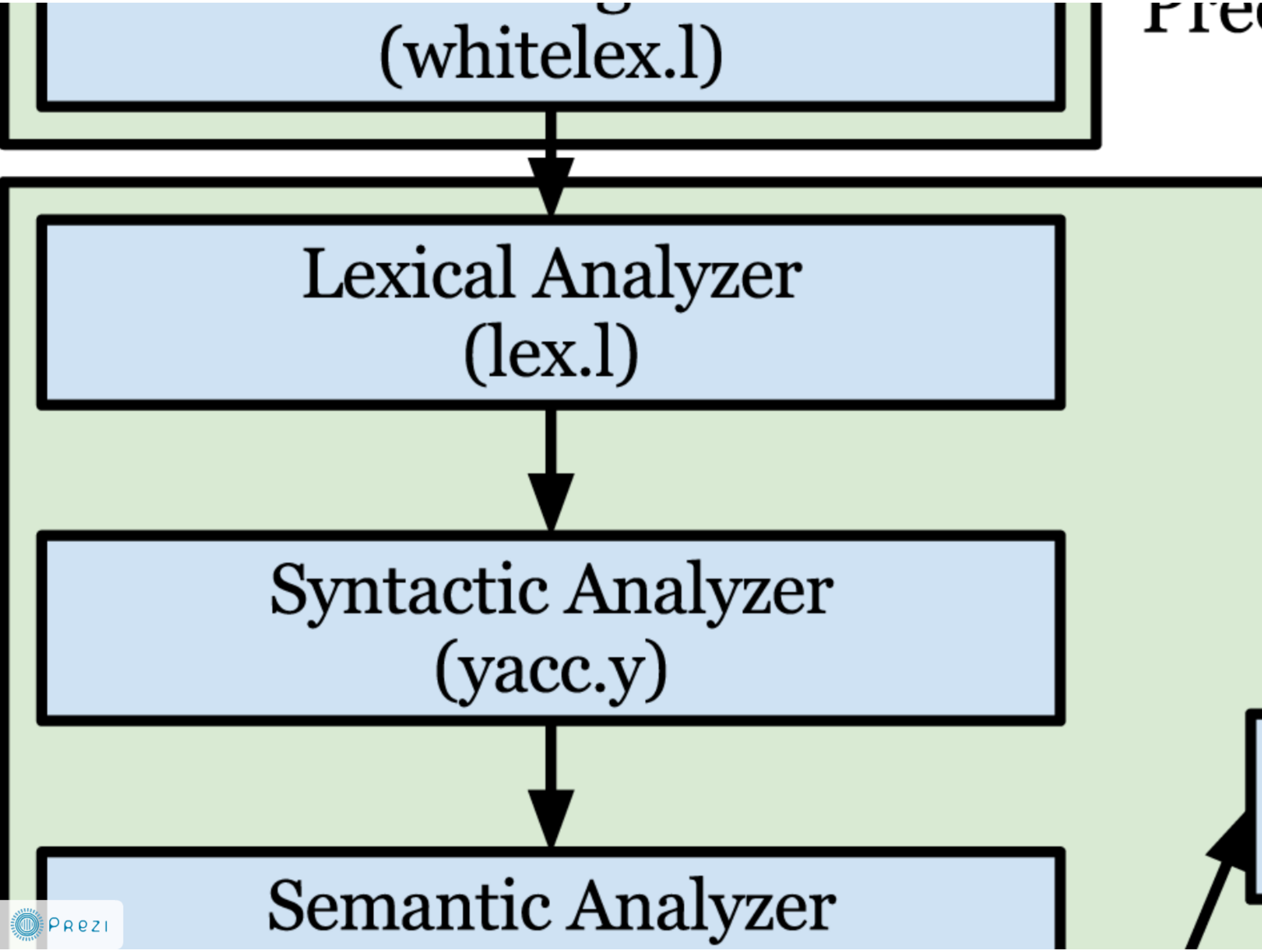
Translator Architecture

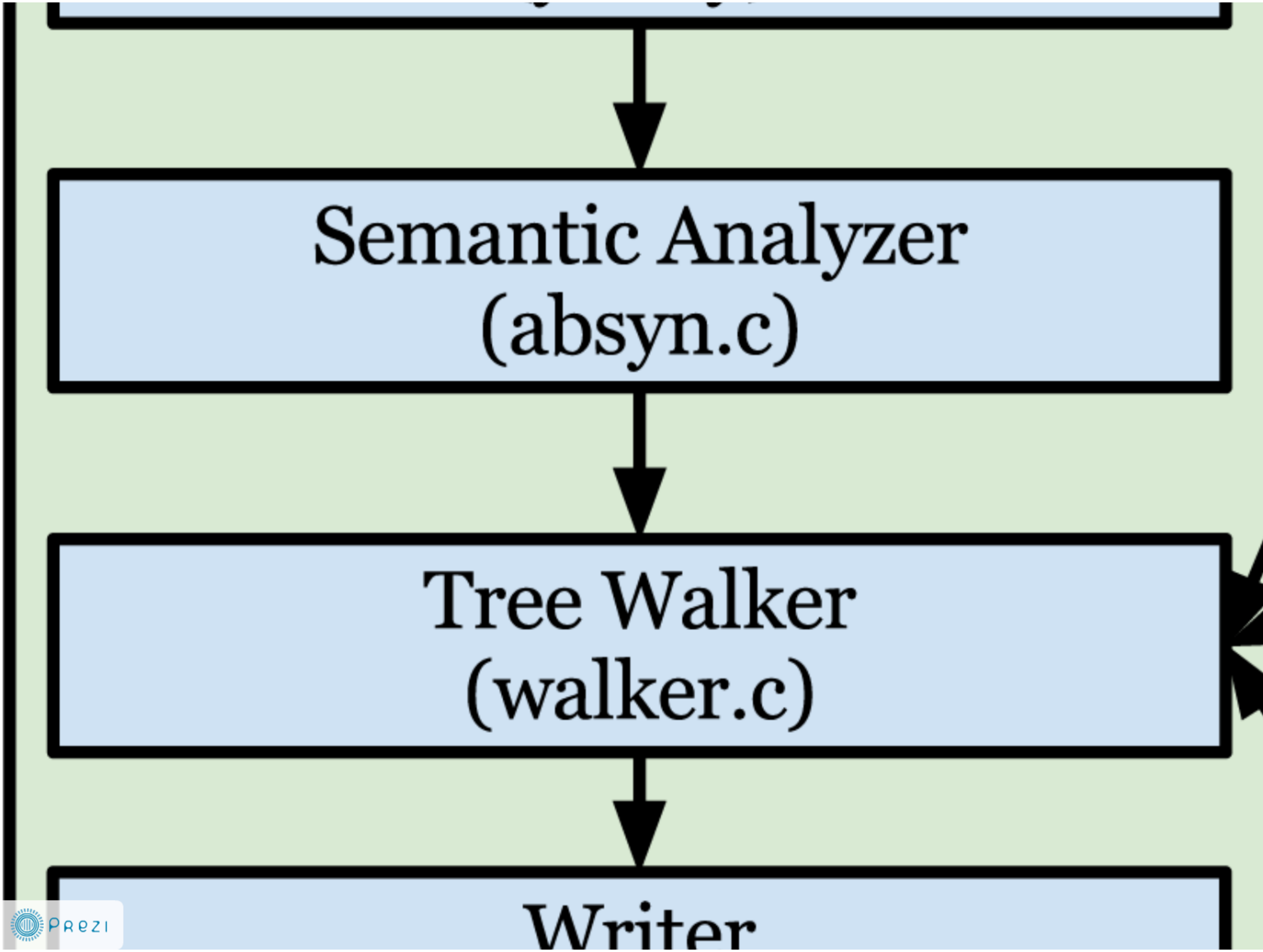


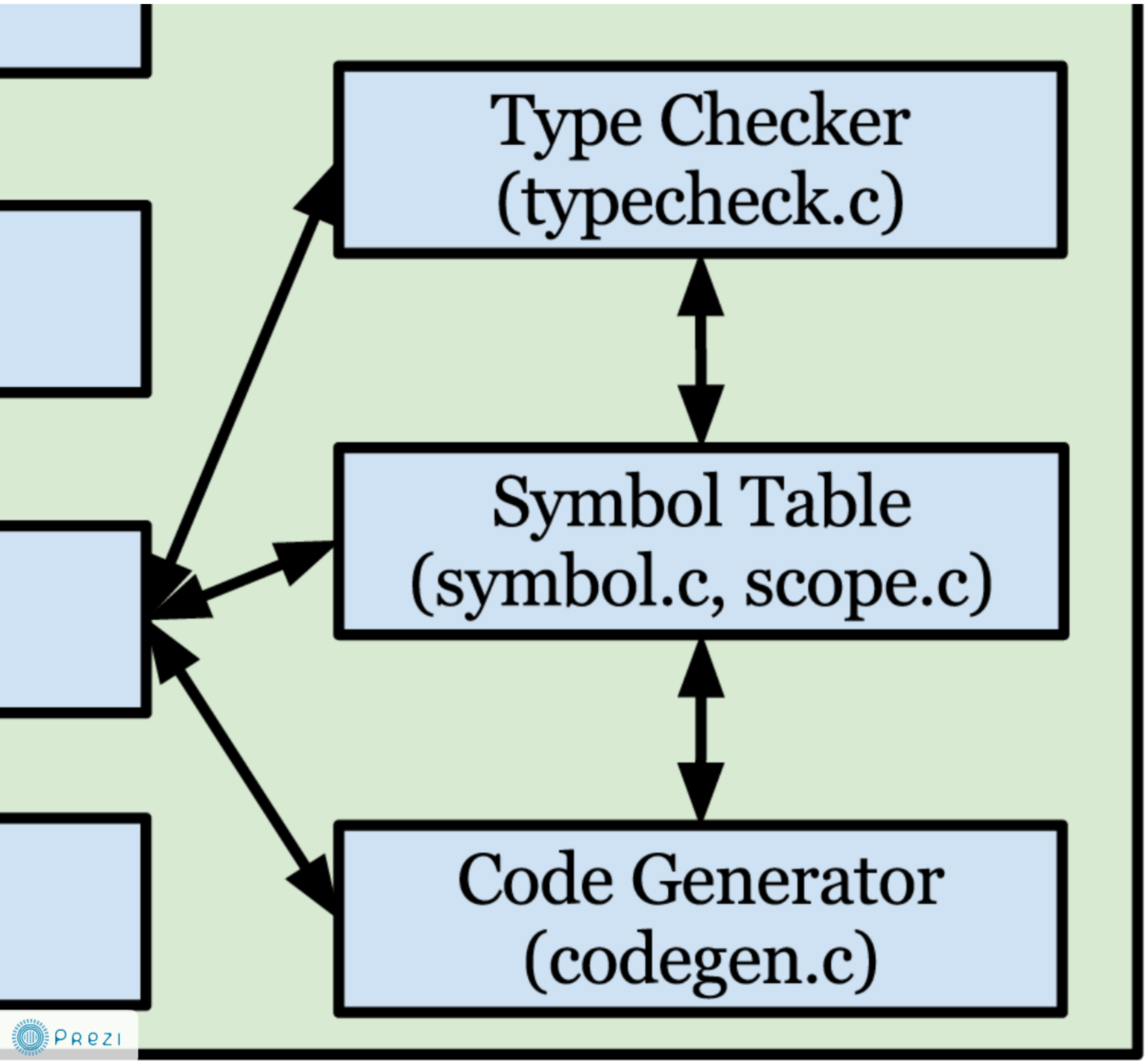
Lexical Organizer (whitelex.l)
Precompiler (klp)
Lexical Analyzer (lex.l)
Syntactic Analyzer (yacc.y)
Semantic Analyzer (absyn.c)
Tree Walker (walker.c)
Writer (yacc.y)
Type Checker (typecheck.c)
Symbol Table (symbol.c, scope.c)
Code Generator (codegen.c)
Compiler (klc)
Executable Generation (gcc)
Machine Code Generator (gcc)
a.out
Output Executable

Translator Archi

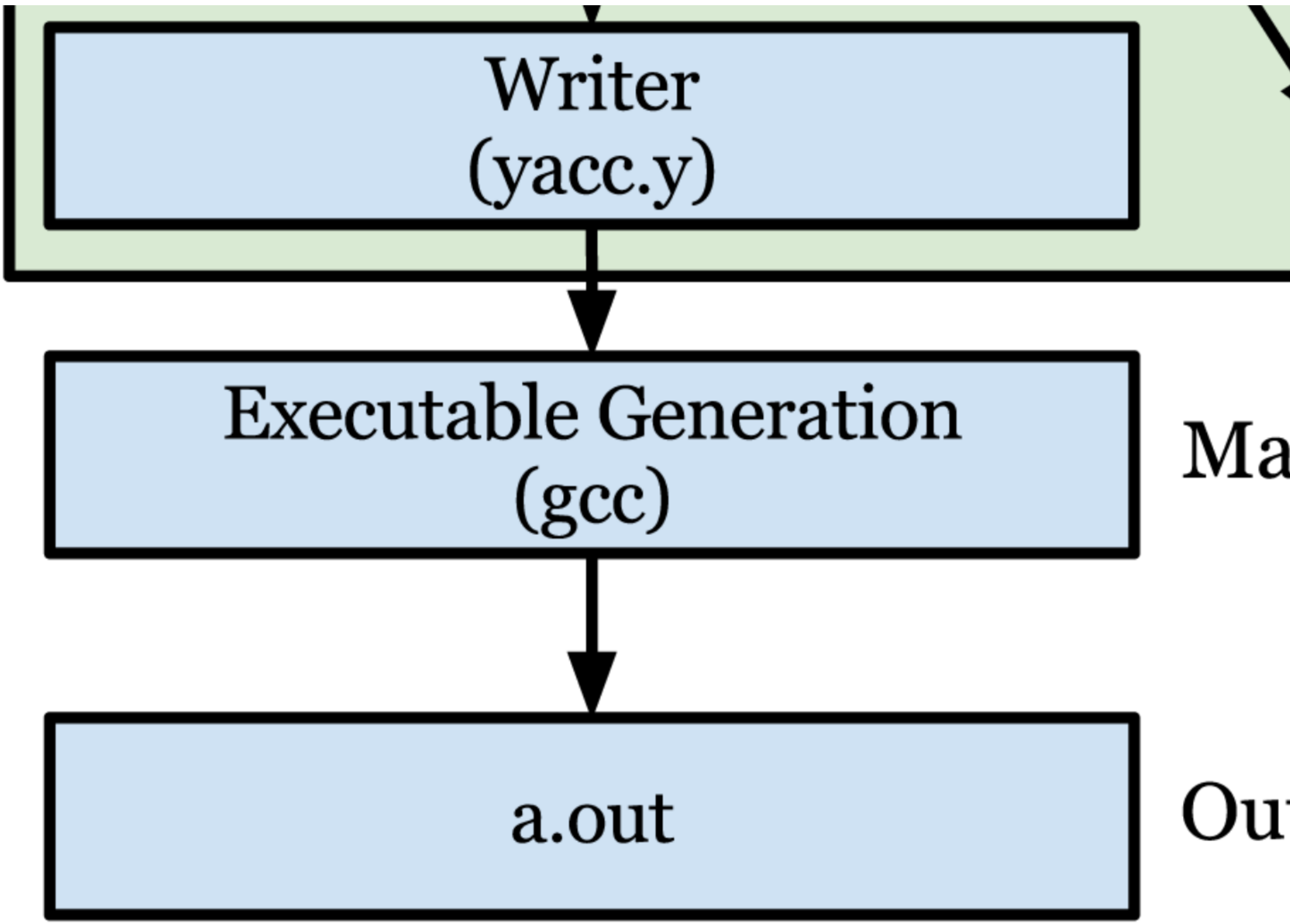



(whitelex.l)
Pre
Lexical Analyzer
(lex.l)
Syntactic Analyzer
(yacc.y)
Semantic Analyzer
Prezi


Semantic Analyzer
(absyn.c)
Tree Walker
(walker.c)
Writer

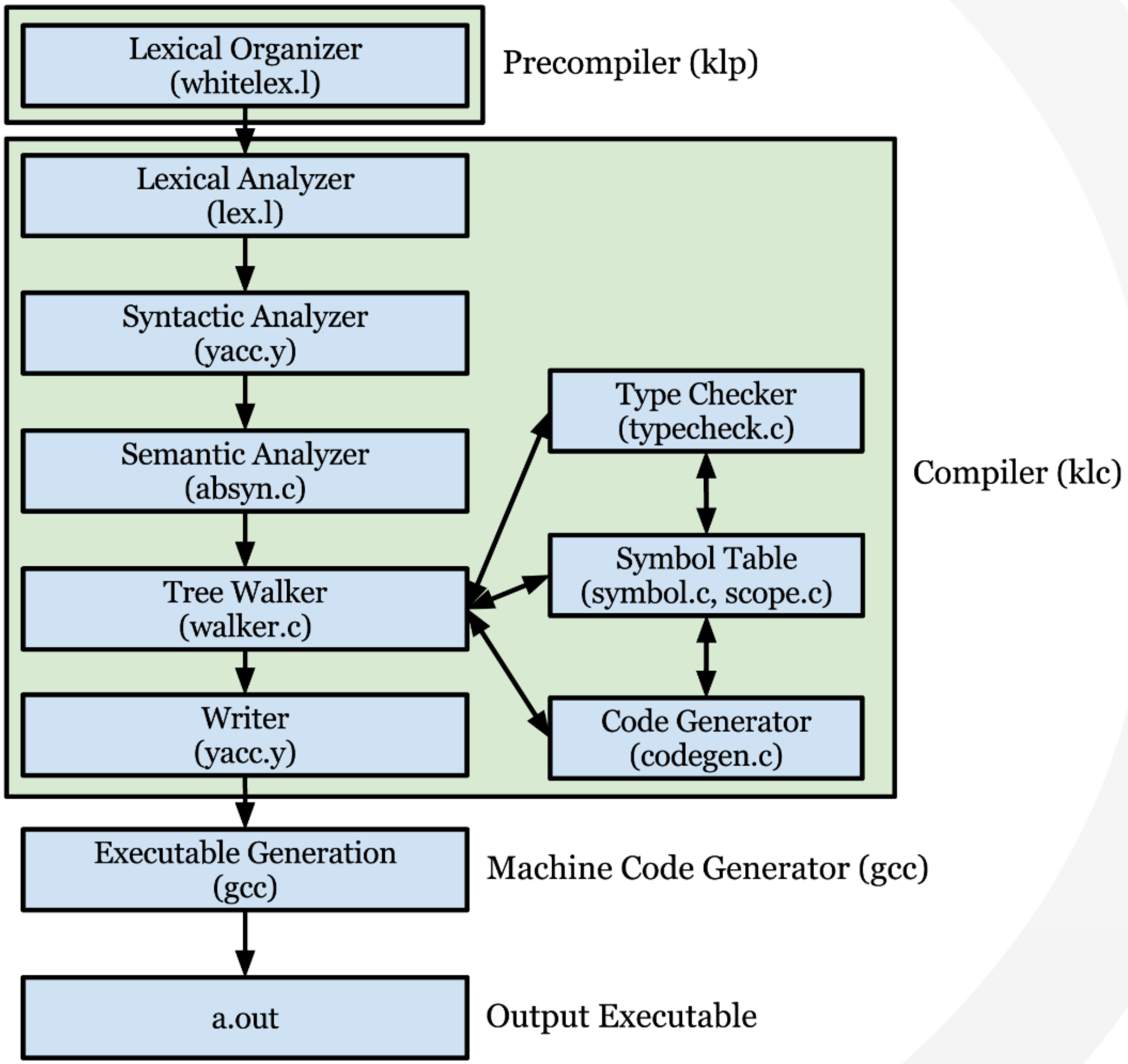

Type Checker
(typecheck.c)
Symbol Table
(symbol.c, scope.c)
Code Generator
(codegen.c)

Compil



Prezi

Translator Architecture



Runtime Environment

- Output by GCC in machine code
- Just type ./a.out in *nix shell
- Output goes to stdout
- Can be used in combination with shell tools

Testing

Unit Testing

- Individual elements and types tested
- Each represented in .lin file
- Different outputs were produced during each stage of production depending on the compilation status of files at that stage

Integration Testing

- After unit tests were verified… .lin integration test files were written
- Same output methodology as unit tests

Regression Test Suite

- Shell script file iterated through all files in test suite
- Produced a log, output files, and errors from file
- Printed the status of each file (OK or FAILED)

Verification

- Verification process combined test plan and Github functionality
- Once all tests were passed it was possible to move on to other features
- If tests failed it was back to the drawing board!

Unit Testing

- -Individual elements and types tested
- Each represented in .kn file
- Different outputs were produced during each stage of production depending on the compilation status of files at that stage

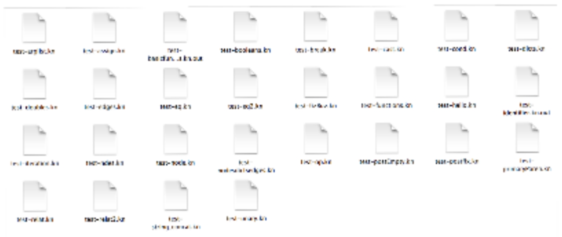
Prezi

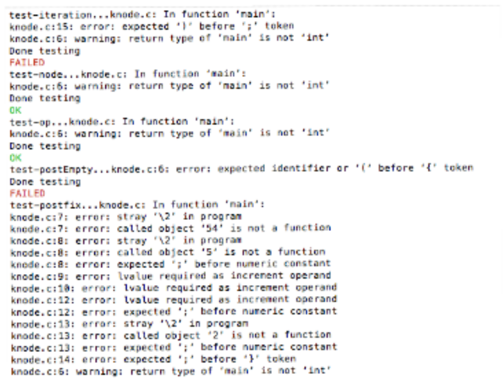
Integration Testing

- After unit tests were verified , .kn integration test files were written
- Same output methodology as unit tests

Regression Test Suite

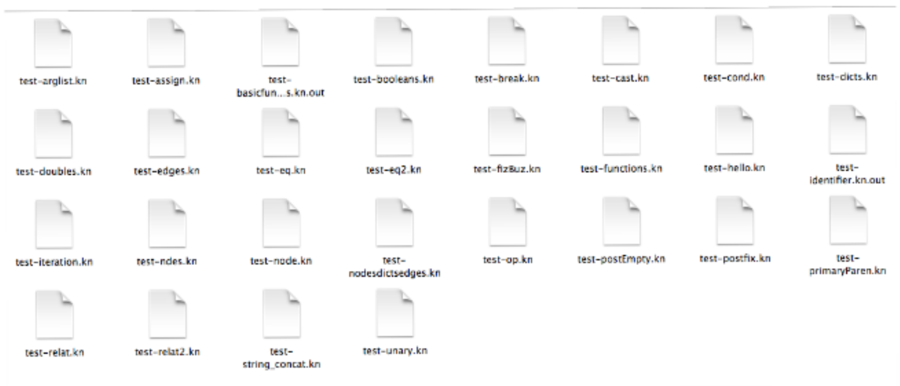
- Shell script file iterated through all files in test suite
  - Produced a log, output files, and errors from file
    - Printed the status of each file (OK or FAILED)

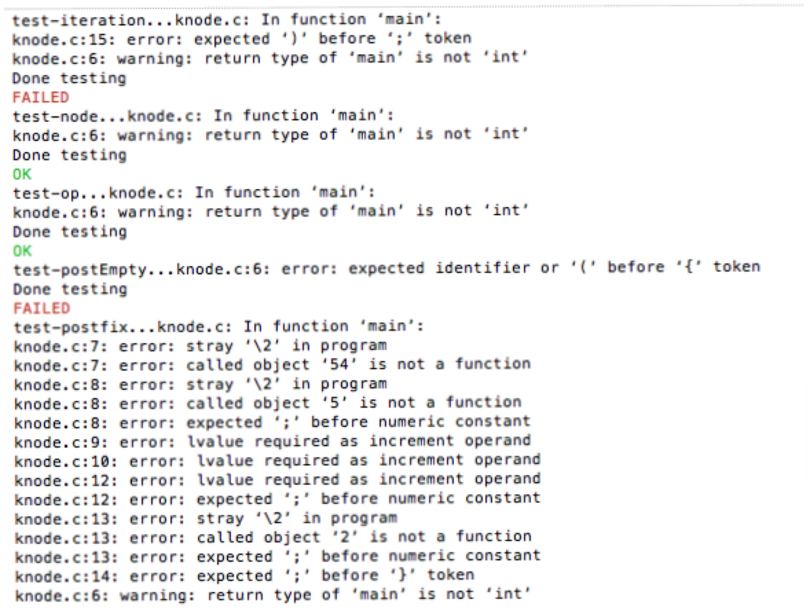




```
test-iteration...knode.c: In function 'main':
knode.c:15: error: expected ')' before ';' token
knode.c:6: warning: return type of 'main' is not 'int'
Done testing
FAILED
test-node...knode.c: In function 'main':
knode.c:6: warning: return type of 'main' is not 'int'
Done testing
OK
test-op...knode.c: In function 'main':
knode.c:6: warning: return type of 'main' is not 'int'
Done testing
OK
test-postEmpty...knode.c:6: error: expected identifier or '(' before '{' token
Done testing
FAILED
test-postfix...knode.c: In function 'main':
knode.c:7: error: stray '\2' in program
knode.c:7: error: called object '54' is not a function
knode.c:8: error: stray '\2' in program
knode.c:8: error: called object '5' is not a function
knode.c:8: error: expected ';' before numeric constant
knode.c:9: error: lvalue required as increment operand
knode.c:10: error: lvalue required as increment operand
knode.c:12: error: lvalue required as increment operand
knode.c:12: error: expected ';' before numeric constant
knode.c:13: error: stray '\2' in program
knode.c:13: error: called object '2' is not a function
knode.c:13: error: expected ';' before numeric constant
knode.c:14: error: expected ';' before '}' token
knode.c:6: warning: return type of 'main' is not 'int'
```

- Shell script file iterated through all files in test suite
  - Produced a log, output files, and errors from file
    - Printed the status of each file (OK or FAILED)

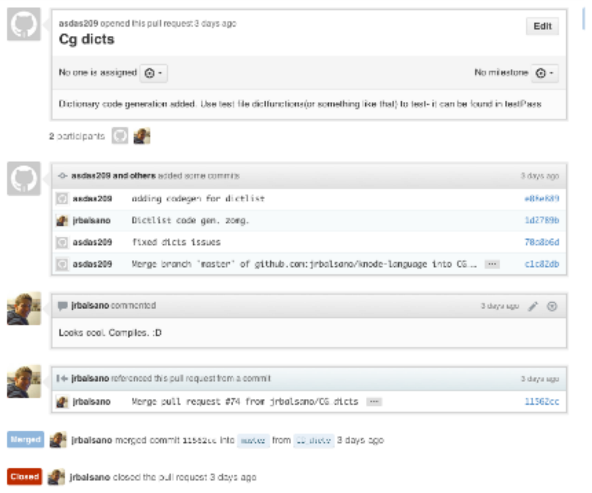




```
test-iteration...knode.c: In function 'main':
knode.c:15: error: expected ')' before ';' token
knode.c:6: warning: return type of 'main' is not 'int'
Done testing
FAILED
test-node...knode.c: In function 'main':
knode.c:6: warning: return type of 'main' is not 'int'
Done testing
OK
test-op...knode.c: In function 'main':
knode.c:6: warning: return type of 'main' is not 'int'
Done testing
OK
test-postEmpty...knode.c:6: error: expected identifier or '(' before '{' token
Done testing
FAILED
test-postfix...knode.c: In function 'main':
knode.c:7: error: stray '\2' in program
knode.c:7: error: called object '54' is not a function
knode.c:8: error: stray '\2' in program
knode.c:8: error: called object '5' is not a function
knode.c:8: error: expected ';' before numeric constant
knode.c:9: error: lvalue required as increment operand
knode.c:10: error: lvalue required as increment operand
knode.c:12: error: lvalue required as increment operand
knode.c:12: error: expected ';' before numeric constant
knode.c:13: error: stray '\2' in program
knode.c:13: error: called object '2' is not a function
knode.c:13: error: expected ';' before numeric constant
knode.c:14: error: expected ';' before '}' token
knode.c:6: warning: return type of 'main' is not 'int'
```

Verification

- Verification process combined test plan and Github functionality
- Once all tests were passed it was possible to move on to other features
- If tests failed it was back to the drawing board!



asdas209 opened this pull request 3 days ago

Edit

Cg dicts

No one is assigned | No milestone

Dictionary code generation added. Use test file dictfunctions(or something like that) to test- it can be found in testPass

2 participants

asdas209 and others added some commits — 3 days ago

asdas209	adding codegen for dictlist	e8fe889
jrbalsano	Dictlist code gen. zomg.	1d2789b
asdas209	fixed dicts issues	78c8b6d
asdas209	Merge branch 'master' of github.com:jrbalsano/knode-language into CG...	c1c82db

jrbalsano commented — 3 days ago

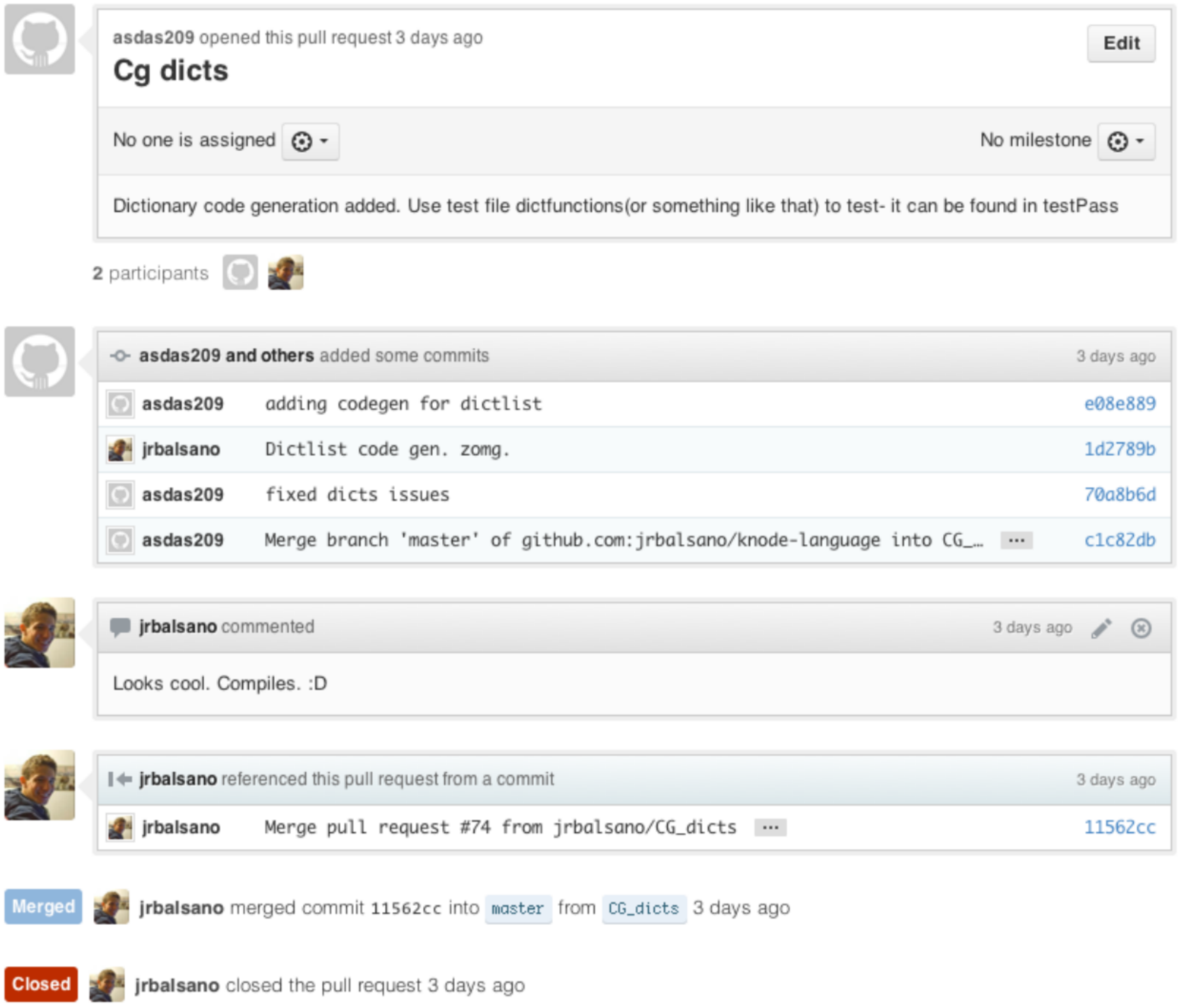
Looks cool. Compiles. :D

jrbalsano referenced this pull request from a commit — 3 days ago

jrbalsano	Merge pull request #74 from jrbalsano/CG dicts	11562cc

Merged — **jrbalsano** merged commit 11562cc into master from CG_dicts 3 days ago

Closed — **jrbalsano** closed the pull request 3 days ago



asdas209 opened this pull request 3 days ago

Cg dicts

Edit

No one is assigned

No milestone

Dictionary code generation added. Use test file dictfunctions(or something like that) to test- it can be found in testPass

2 participants

asdas209 and others added some commits 3 days ago

asdas209	`adding codegen for dictlist`	e08e889
jrbalsano	`Dictlist code gen. zomg.`	1d2789b
asdas209	`fixed dicts issues`	70a8b6d
asdas209	`Merge branch 'master' of github.com:jrbalsano/knode-language into CG_…`	c1c82db

jrbalsano commented 3 days ago

Looks cool. Compiles. :D

jrbalsano referenced this pull request from a commit 3 days ago

jrbalsano	`Merge pull request #74 from jrbalsano/CG_dicts`	11562cc

Merged jrbalsano merged commit `11562cc` into `master` from `CG_dicts` 3 days ago

Closed jrbalsano closed the pull request 3 days ago

Conclusions

- Working on a group project has its challenges
- Cat GIFs make late-night pull requests fun!
- Great people, great product
- Knode is highly capable and easy to use
- Low level efficiency, higher level coding
- Manage interrelated data easily
- Capable, convenient, and cross-platform

knode
a graph-based language

Project Manager: Krista Kohler
Language Guru: Jon Jia
System Architect: Jonathan Balsano
System Integrator: Maria Moresco
System Tester: Ruby Robinson

Imagine...

- You are a non-programmer who works with lots of data.
- You like the intuitive look of Python, but you don't like performance issues.
- You need the speed and efficiency of C without the headaches of pointers and memory management.

The Problem

- In our world, there is a LOT of data
- This data does not exist in a vacuum
- As our lives becomes more social, there's a greater need to access, share, and analyze interrelated data
- But data is not always easy to manage, especially in large quantities

The Solution:
knode

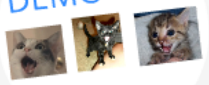
- Declarative language focused on convenience
- Pretty syntax, snappy performance
- Speed of C, but memory managed
- Minimal programming experience required
- Built-in graph primitives for easy data viewing and manipulation

knode is...

Key Syntax

A knode program

DEMO TIME



Project Management

- Traditional, five-phase development process
- Emphasis on collaboration and flexibility
- Weekly meetings: planning and working
- Asana for outline, task management, and deadlines
- "First-come, first-serve" assignment of responsibilities

Development

- Lex
- Yacc
- UTHash

Translator Architecture

Runtime Environment

- Output by GCC in machine code
- Just type ./a.out in *nix shell
- Output goes to stdout
- Can be used in combination with shell tools

Testing

Conclusions

- Working on a group project has its challenges
- Cat GIFs make late-night pull requests fun!
- Great people, great product
- Knode is highly capable and easy to use
- Low level efficiency, higher level coding
- Manage interrelated data easily
- Capable, convenient, and cross-platform