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 become 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
demarcate blocks
- Used for:
  - Function
  - Definitions
  - Flow control
  - Certain
    declarations

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

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

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

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

Memory Management
- "Memory management is a real b****"
whitespace

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

```python
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

```json
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

```plaintext
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

```python
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 

Edge declarations

p2.type = " type of 

p3.type = " hate 

String concatenation

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)
A knode program

main():

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

edge p1 = [a->b]
edge p2 = [b->c]
edge p3 = [a->c]
node b
  role: "dogs"
node c
  role: "cats"

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

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)
Edge declarations

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

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

- **Precompiler**
  - Lexical Organizer
    - (whitelex.l)
  - Lexical Analyzer
    - (lex.l)
  - Syntactic Analyzer
Semantic Analyzer (absyn.c)

Tree Walker (walker.c)

Writer
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
- verifies individual components
- can be run independently
- ensures each component works as expected
- helps catch issues early

Integration Testing
- verifies components work together
- checks interfaces and dependencies
- ensures components integrate smoothly
- helps catch issues that arise from integration

Regression Test Suite
- retests a function after changes
- ensures changes don't break existing functionality
- helps maintain software quality

Performance Testing
- measures system performance
- evaluates under load conditions
- ensures system can handle expected loads
- helps identify bottlenecks and capacity issues
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
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)
- 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!
asdas209 opened this pull request 3 days ago

**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  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

---

**The Problem**
- Native code is too slow  
- Not easy to code in standard  
- Programmers don't want more work, but it's a project/new idea.  
- Need to code in a way that is more familiar  
- Existing systems need to be able to communicate with new systems

**Imagine...**
- An all-encompassing online system  
- The goal: a workflow of things  
- Each knode program is a single system  
- Uses code in a familiar and natural way  
- No need to learn a new language

**The Solution:**
**knode**
-ัสีผิวหมอนที่อยู่อย่างคุณค่า  
- เป็นผลิตภัณฑ์ที่มีคุณค่ามาก  
- ใช้งานง่าย และมีประสิทธิภาพ

---

**knode is...**
- A knode program

**Key Syntax**
- ใช้ในภาษา  
- มีรูปแบบที่ง่ายและง่ายต่อการดำเนินการ

---

**Project Management**
- Characteristics of a graph development process  
- Organizational aspects  
- Development strategies  
- Project timeline, deliverables, and milestones

---

**Development**
- Lex  
- Yacc  
- UTHash

---

**Translator Architecture**
- Reveals the structure of the translator  
- Allows for easier debugging and testing

---

**Testing**
- Evaluates the functionality and performance of the program

---

**Conclusions**
- Working on a group project has its challenges  
- Knodes 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