Animated version at: http://chantalgalvez.com/PLT/nsbl.html

Role	Member	Email
Project Manager	Chantal Galvez	cg2486@columbia.edu
Language Guru	Jing Zhang	jz2300@columbia.edu
System Integrator	Lixing Pan	lp2441@columbia.edu
System Architect	Jing Zhang	jz2300@columbia.edu
System Tester	Kunal Mishra	ksm2135@columbia.edu

What is NSBL?

NSBL is an *interpreted* programming language that seeks to help data organization by offering a simple language to create, save, and perform complex queries and operations on backend data graphs

What can NSBL do?

What can NSBL do?

READ from

Write to

Query Data
from graph created/loaded

Query Data
from graph created/loaded

What is NSBL?

Graph-based

Highly Abstract

Familiar Syntax

Interpreted and Interactive

How to run

```
$ make
$ nsbl program.nsbl
$ ./a.out
```

What is NSBL?

Graph-based

Highly Abstract

Familiar Syntax

Interpreted and Interactive

How to run

$ make

$ nsbl program.nsbl

$ ./a.out

Language Features

Basic features of NSBL

NSBL Types

- void
- bool
- int
- float
- string
- vertex
- edge
- graph
- vlist
- elist

Operations

- assign =
- logical || &&
- equality == !=
- relational < > <= >=
- math math
- cast cast
- unary unary
- postfix postfix

NSBL flow control

- if else
- while
- for
- foreach
- break, continue, return

Function and Function Literal

IO

- print
- xml file

Other remarks

- name equivalence
- scope and live time
- static and dynamic
- garbage collector

Getting Started

No main

Compared to C

```
// NSBL hello world 1
string  world = "World";
print  << "Hello  " << world << "\n";
```

```
// C hello world
#include stdio.h>
void main() {
        printf("Hello ");
        printf("World");
        printf("\n");
}
```

However, use a function with arbitrary name to warp

```
// NSBL hello world 2
sayHello();

void sayHello() {
        string  world = "World";
        print  << "Hello  " << world << "\n";
        return ;
}
```

Scope and GC

```
                vertex v;
{  vertex vv; }
// END_FILE
```


```
                #include "nsbl.h"
VertexType * v_v0_s0;
int main() {

gcInit();
v_v0_s0 = NULL;
v_v0_s0 = NULL; assign_operator_vertex ( &( v_v0_s0 ) ,new_vertex() );
{
  VertexType * vv_v1_s1 = NULL;
        vv_v1_s1 = NULL; assign_operator_vertex ( &( vv_v1_s1 ) , v_v0_s0 );
  destroy_vertex( vv_v1_s1 );vv_v1_s1 = NULL;
} // END_COMP
destroy_vertex( v_v0_s0 );v_v0_s0 = NULL;
gcDel();

} // END_OF_MAIN
```

Control flow and function

```
// NSBL  factorial with static type
int i, n=5, f=1;
for ( i=1; i<=n; i = i+1 ) {
        f = f*i ;
}
print << "fact(5) =" << f << "\n";
```

```
// NSBL  factorial with fund 1
print << "fact(5) =" << fact(5) << "\n";
int fact (int n) {
        if (n==1) { return 1; }
        return fact(n-1)*n;
}
```

```
// NSBL  factorial with dynamic type
vertex v;
v.n=5, v.f=1;

for ( v.i=1; v.i<=v.n; v.i = v.i+1 ) {
        v.f = v.f*v.i ;
}

print << "fact(5) =" << v.f << "\n";
```

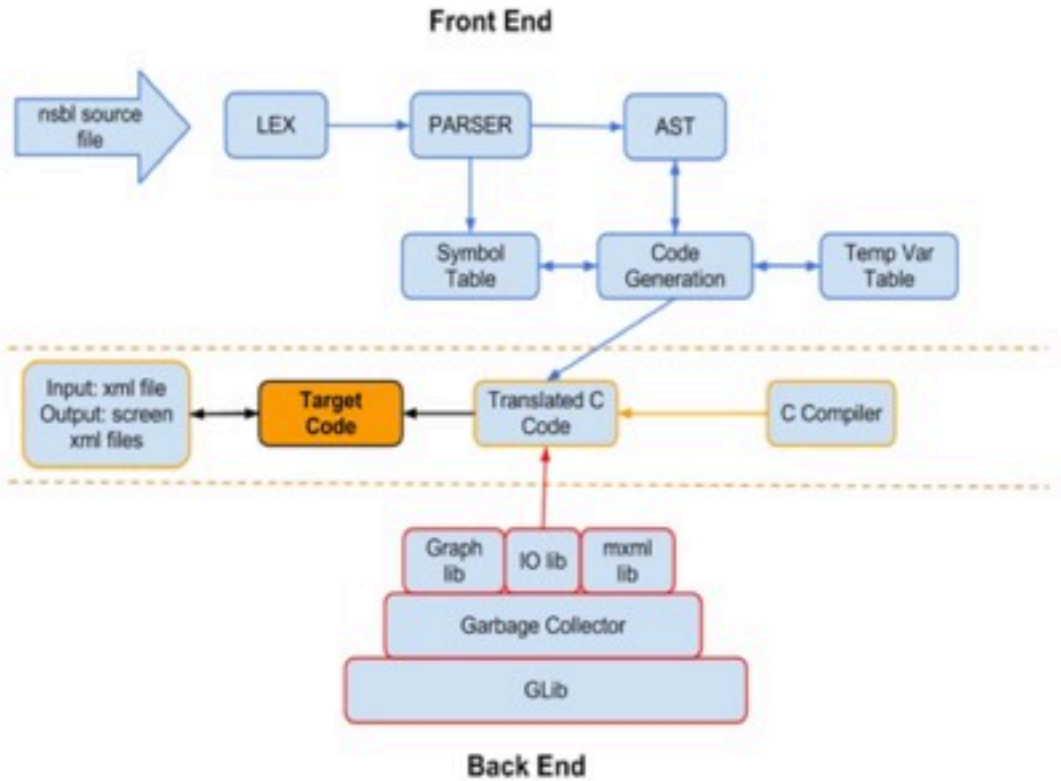
```
// NSBL  factorial with fund 2
vertex v;          v.n=5;
print << "fact(5) =" << vfact(v) << "\n";
int vfact( vertex v ){
        if (v.n==1) { return 1; }
        vertex vv;
        vv.n = v.n - 1;
        return vfact(vv)*v.n;
}
```

Graph operation

```
// declaration
vertex v1, v2;
edge e1;
graph g;
// assign attributes
v1.name = "Joe";
v1.age = 20;
v2.name = "Mike";
v2.age = 25;
// assign edge
e1: v1->v2;
e1.rel = "knows";
// construct a graph
g <: v1; g <: v2;
g <: e1;
// save graph
string file = "g.xml";
file << graph;
```


```
// read graph
graph g;
string file = "g.xml";
file >> g;
// get vertices and edges
print << g.allV;
print << g.allE;
// pipe
print << g.allV|OutE|endV;
// match
print << g.allV?[@age>20];
// foreach
foreach ( vertex v : g.allV ) {
        print << v;
}
```

ARCHITECTURE



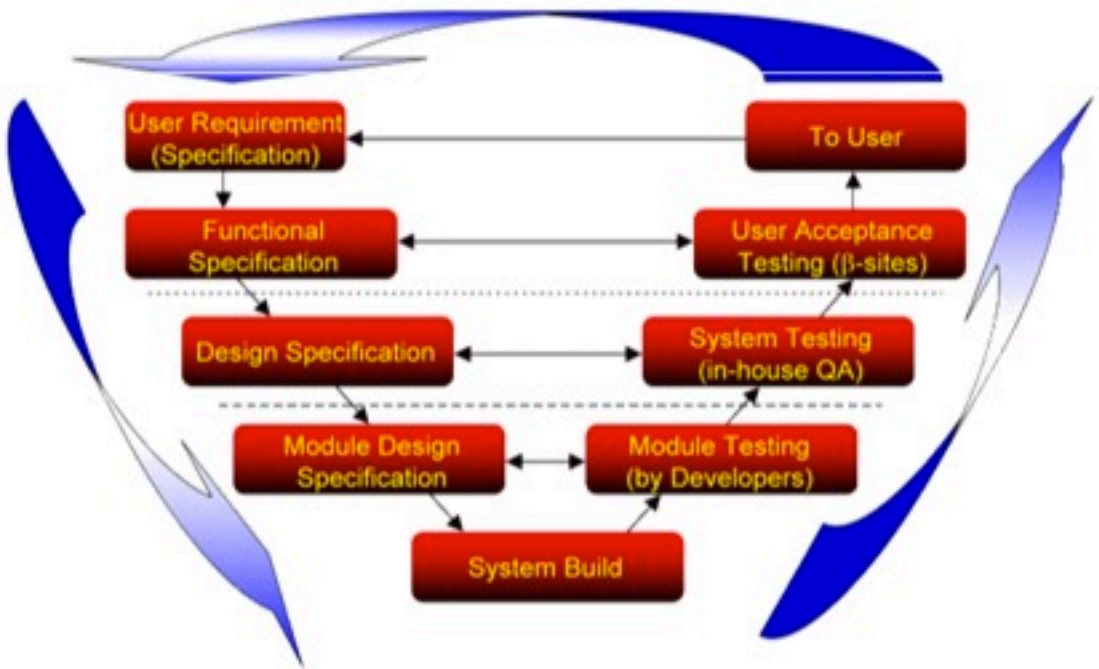
Development Environment

Language: C
OS: Linux
Compiler: gcc
Debug Tool: GDB
Source Control: git (github)
Editor: vi/vim
Libraries: Glib, Mini-XML

Test Model

V Model of Software Development

Focuses on Testing each phase of development lifecycle
Industry Standard for Quality Assurance



Phases of Testing

Project Requirement Testing

No formal approach

Intuitive group discussions helped narrow down the scope and filter potential implementation risks (For e.g., shell type console)

Unit Testing

Every Module of the compiler developed was tested by the developer manually (against unit test programs).

The developer logged the defects and fixed them

Incremental and Regression oriented.

Unit Testing of libraries Graph & I/O (most important)

Functional test cases

System Integration Testing

System Integration Testing

Most important testing phase.
Consumes most of testing effort.
NSBL being rolled out for the 1st time.
All functionalities are high priority.

Testing Approach

Manual Testing: Test cases are functionality specific programs in NSBL based on a pseudo-fuzzing approach.
Functionalities were grouped together; ensured compatibility and sped up testing (mutated performance testing).

Defect Management

No formal defect management document (due to time constraint)
On finding a bug, contact the pertinent developer via email or mobile.
Re-test the program after bug fix and close the defect.
Weekly meetings for tracking defects and testing progress.

Test Environment

Team member's laptops and CLIC machines.

Assumptions and Risks

(Assumption)Mini-XML library is stable and does not bring in unintended bugs.
(Risk) In case a bug cannot be fixed before the deadline, the mitigation plan was to not pass the functionality for production

SIT Test Plan and Coverage

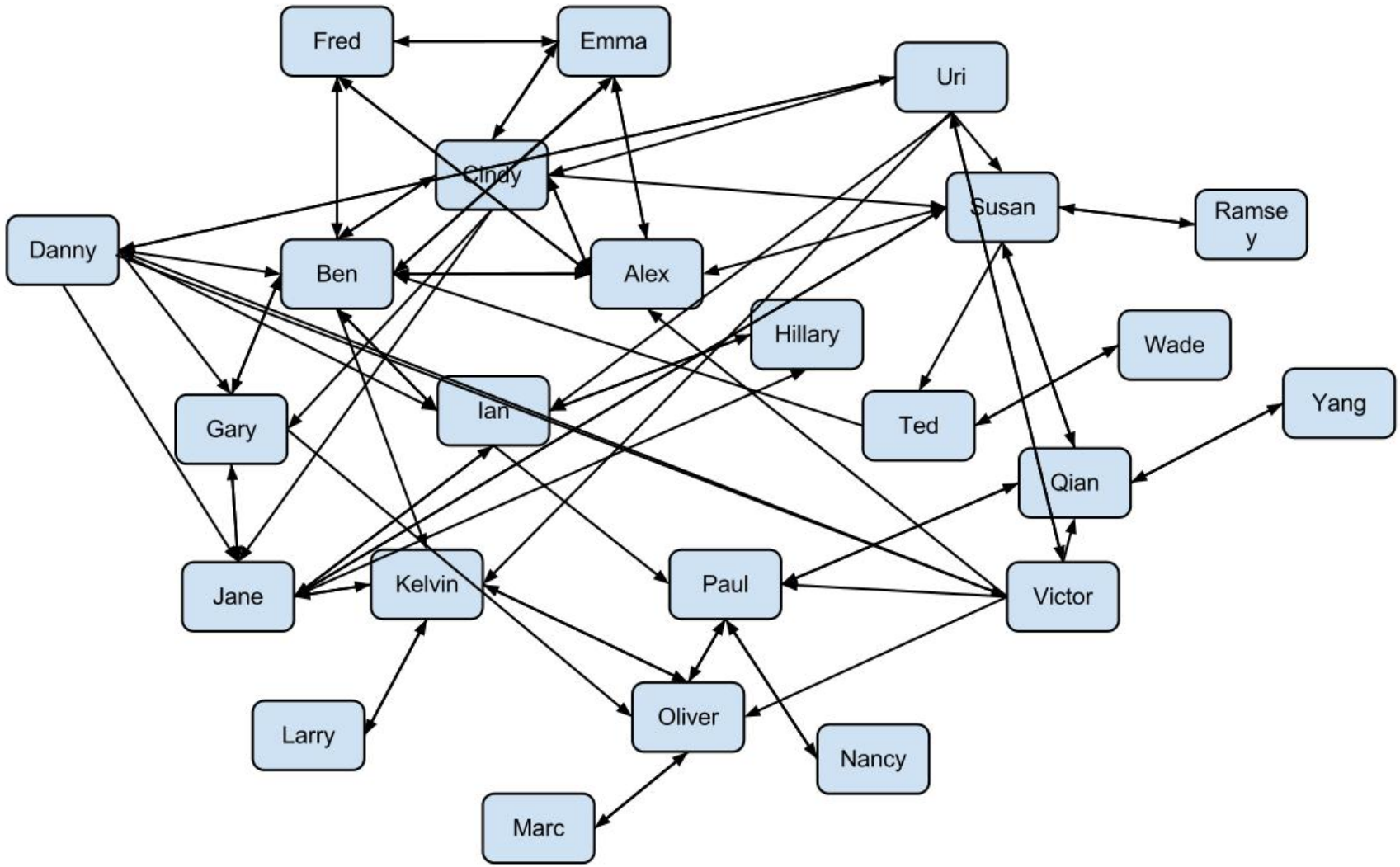
Priority	Functionality	Coverage	Result
High	int, float, string, bool, graph, edge, vertex and list types declaration and creation	Covered	Pass
High	Function declaration and usage	Covered	Pass
High	Functioning of Relational and Arithmetic operators	Covered	Pass
High	Functioning of loops (for/foreach/while)	Covered	Pass
High	Functioning of jump statements (return/break)	Covered	Pass
High	Functioning of if, if/else statements	Covered	Pass
High	Functioning of vertex property functions (outE and inE)	Covered	Pass
High	Functioning of edge property functions (strtV and endV)	Covered	Pass
High	Vertex attribute assignment	Covered	Pass
High	Edge attribute assignment	Covered	Pass
High	Proper functioning of Scope logic	Covered	Pass
High	File read/write	Covered	Pass
High	Graph Query (pipe and match)	Covered	Pass
High	Proper working of function literals	Covered	Pass
High	Proper working of print statement	Covered	Pass
High	Proper functioning of graph property functions (allV and allE)	Covered	Pass

EXAMPLES

Graph creation and Querying
File I/O and BFS

EXAMPLE 1

Graph creation and Querying

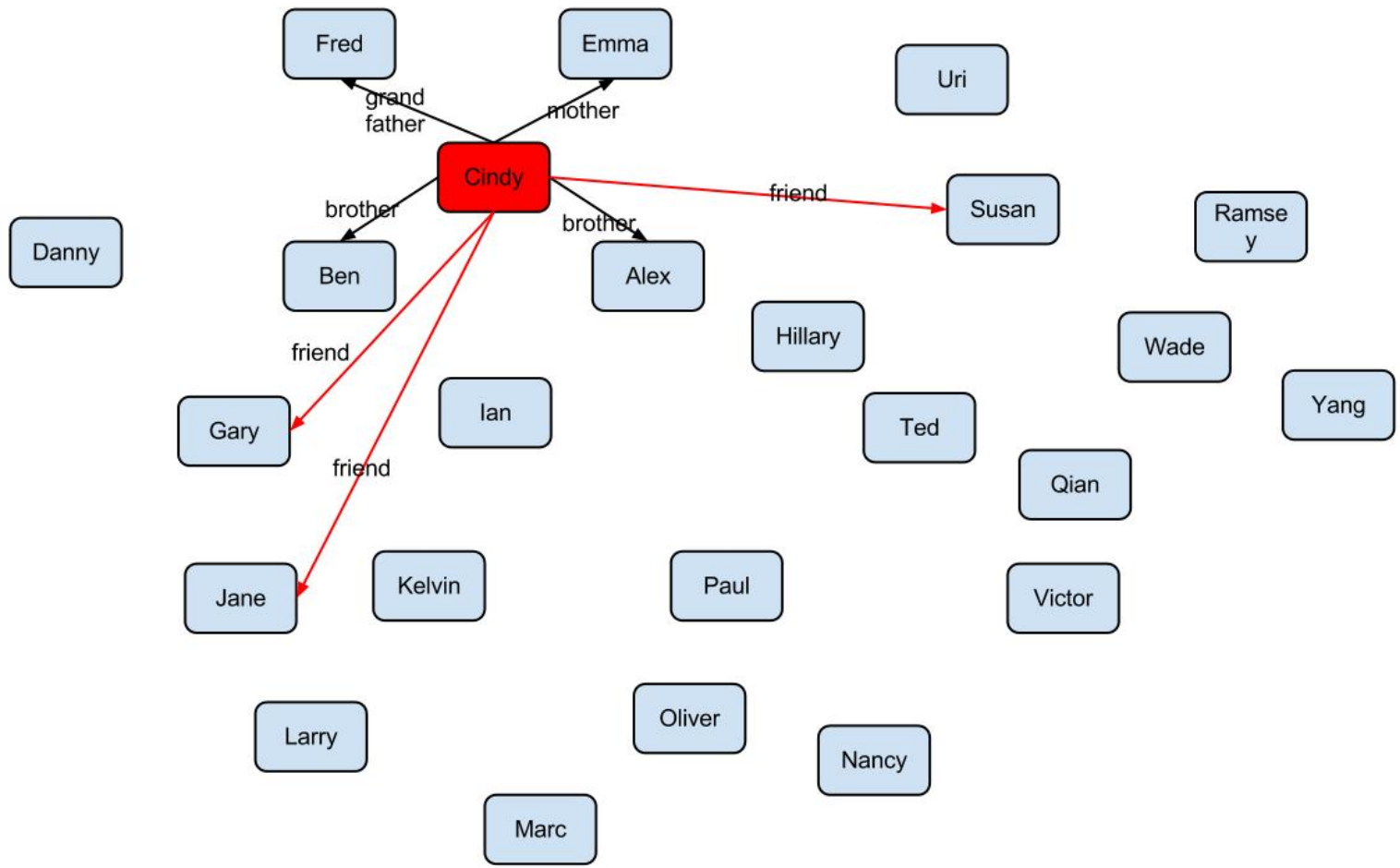

Fred
Emma
Uri
Cindy
Susan
Ramsey
Danny
Ben
Alex
Hillary
Wade
Gary
Ian
Ted
Yang
Qian
Jane
Kelvin
Paul
Victor
Larry
Oliver
Nancy
Marc

```
1 vertex  Alex, Ben, Cindy, Danny, Emma,
2         Fred, Gary, Hillary, Ian, Jane, Kelvin, Larry,
3         Marc, Nancy, Oliver, Paul, Qian, Ramsey, Susan,
4         Ted, Uri, Victor, Wade, Yang;
5
6 //n: name
7 //w. weight
8 //g: gender
9 //a: age
10 Alex.n = "Alex"; Alex.w = 130.5; Alex.g = "m"; Alex.age = 23;
11 Ben.n = "Ben", Ben.w = 140.3; Ben.g = "m"; Ben.age = 24;
12 Cindy.n = "Cindy"; Cindy.w = 124.3; Cindy.g = "f"; Cindy.age = 22;
13 Danny.n = "Danny"; Danny.w = 150.7; Danny.g = "m"; Danny.age = 31;
14 Emma.n = "Emma"; Emma.w = 138.5; Emma.g = "f"; Emma.age = 51;
15 Fred.n = "Fred"; Fred.w = 120.5; Fred.g = "m"; Fred.age = 79;
16 Gary.n = "Gary"; Gary.w = 132.3; Gary.g = "m"; Gary.age = 24;
```

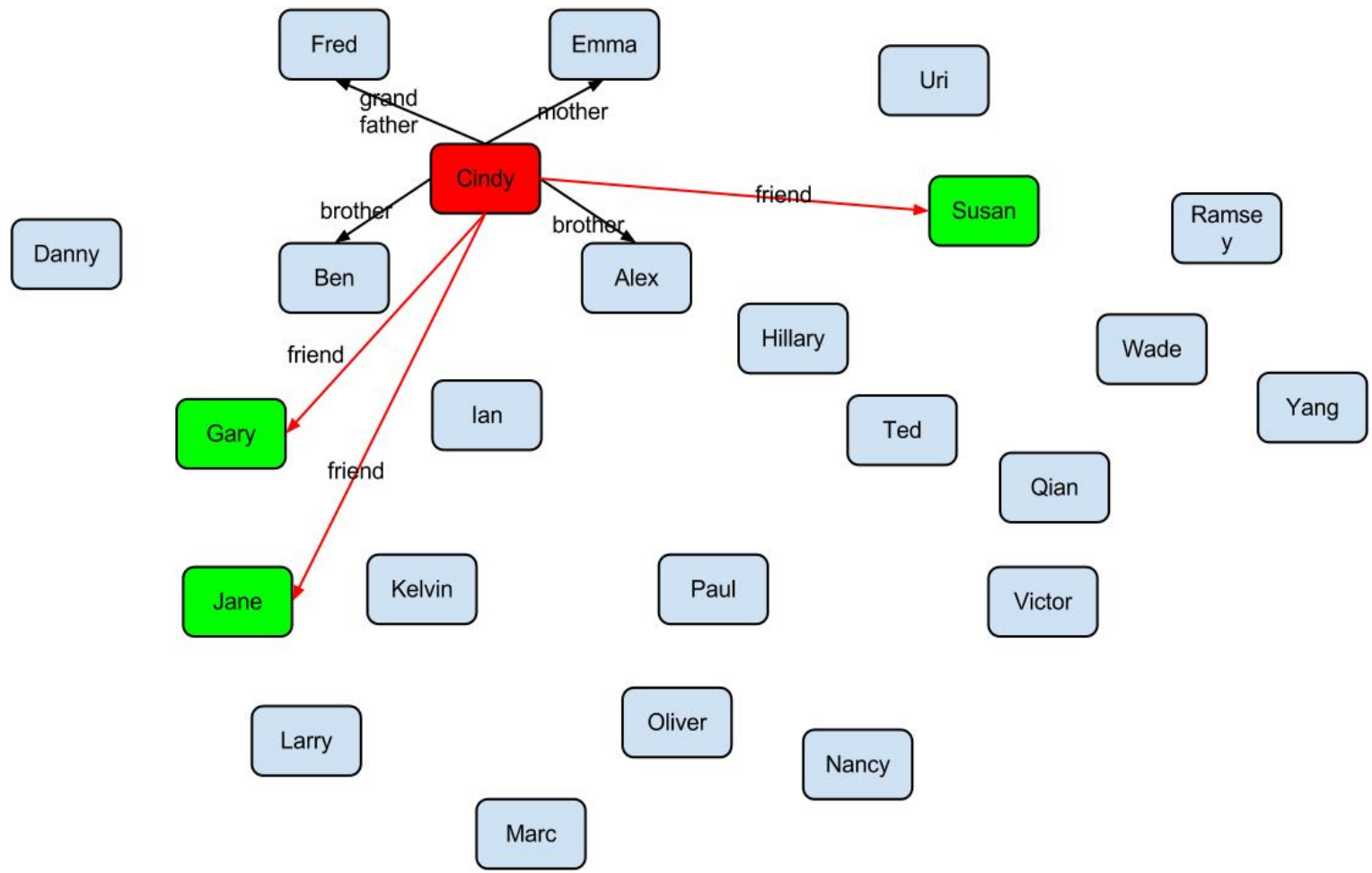
```
35 graph g1;
36 vlist vl_all = [Alex, Ben, Cindy, Danny, Emma,
37         Fred, Gary, Hillary, Ian, Jane, Kelvin, Larry,
38         Marc, Nancy, Oliver, Paul, Qian, Ramsey, Susan,
39         Ted, Uri, Victor, Wade, Yang];
40 g1 <: vl_all;
```

```
106 follow(g1, Cindy, Ben);
107 te = get_edge(Cindy, Ben);
108 te.level = 3;
109 te.rel = "brother";
110 follow(g1, Cindy, Emma);
111 te = get_edge(Cindy, Emma);
112 te.level = 5;
113 te.rel = "mother";
```

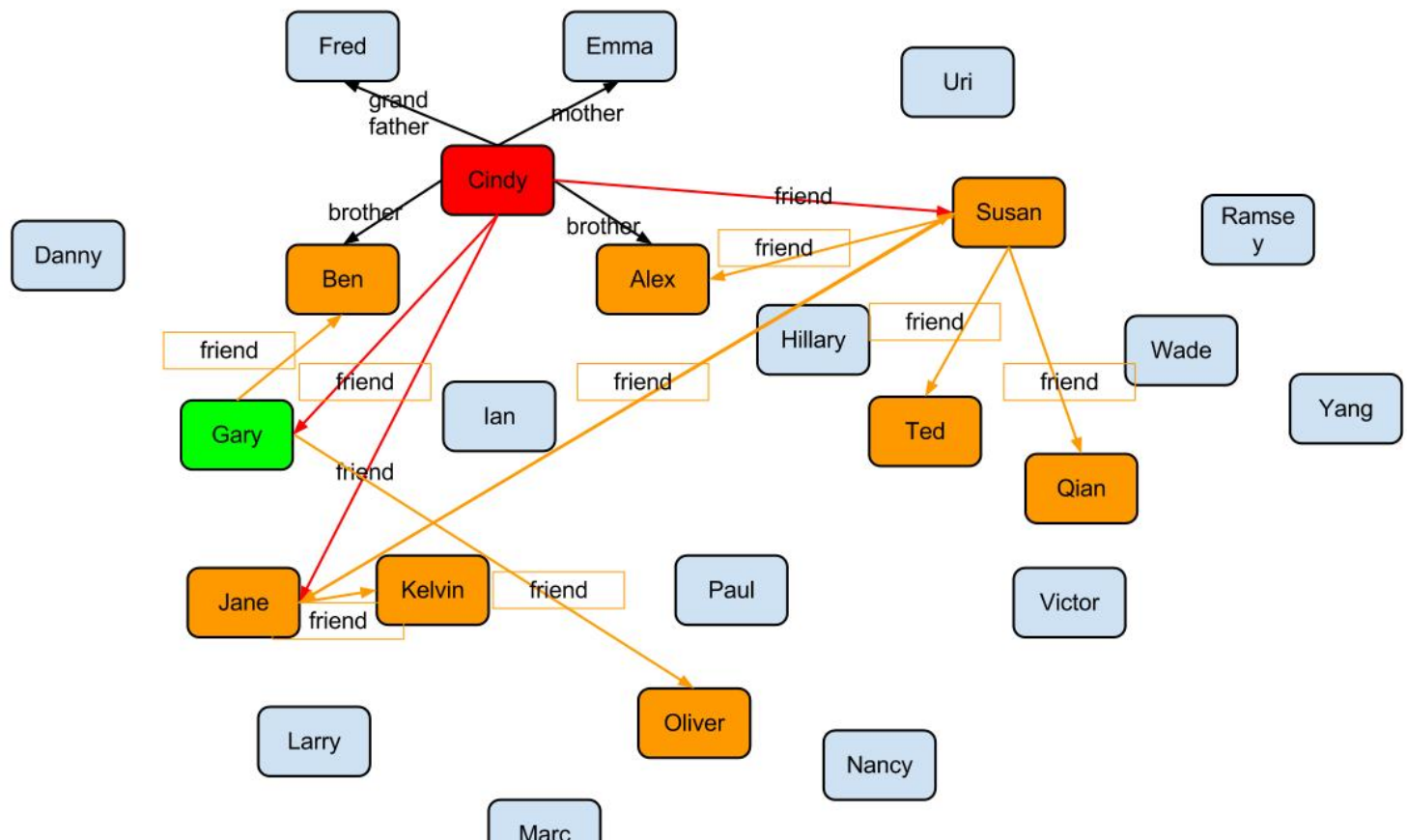
```
441 //get Cindy's friend
442 vlist vl_Cindy_friends = g1.allV?[@n=="Cindy"]|outE?[@rel=="friend"]|endV;
443 print_vlist(vl_Cindy_friends);
444
```



```
441 //get Cindy's friend
442 vlist vl_Cindy_friends = g1.allV?[@n=="Cindy"]|outE?[@rel=="friend"]|endV;
443 print_vlist(vl_Cindy_friends);
444
```

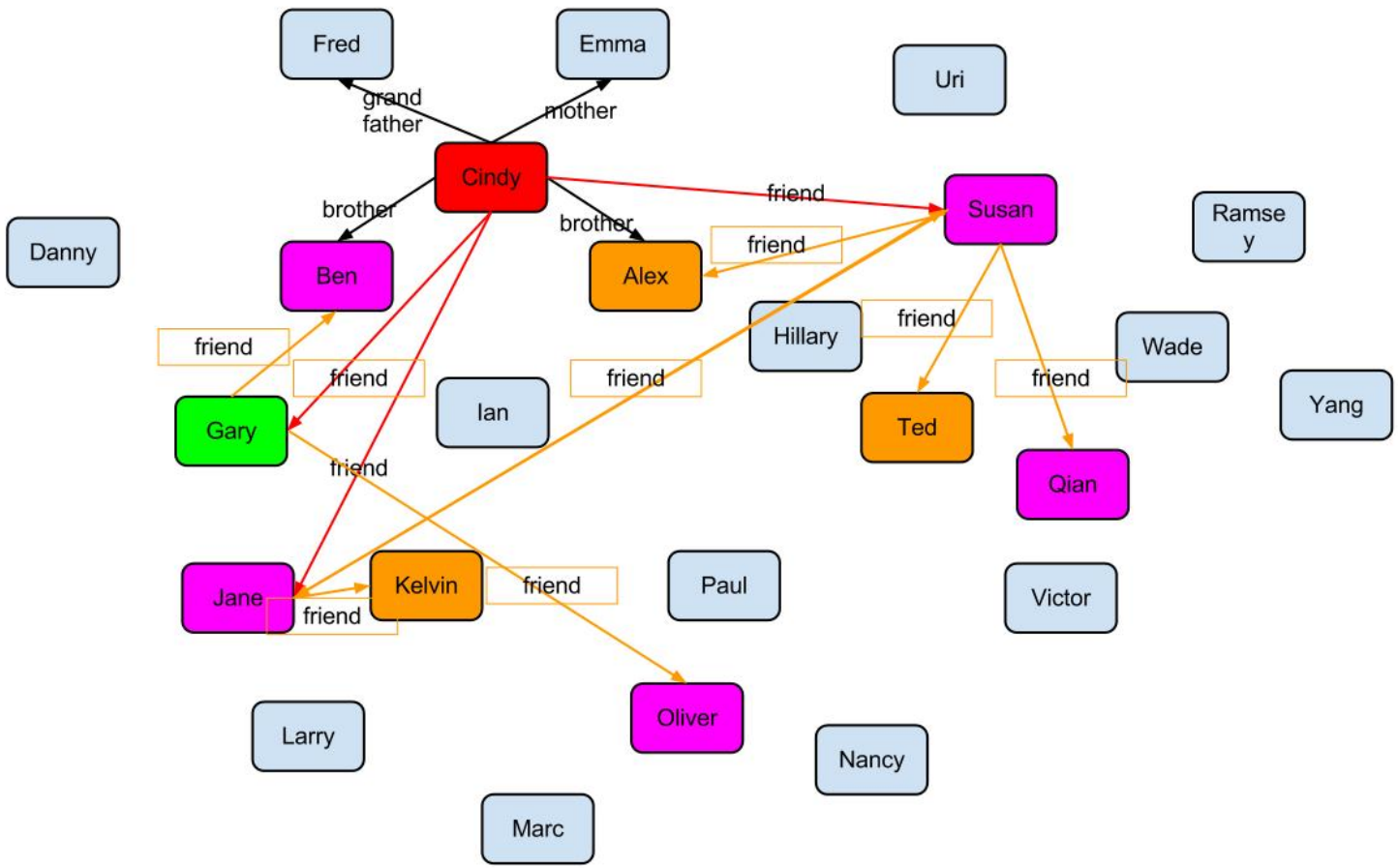


```
445 //get Cindy's friend's friends
446 vlist vl_Cindy_friends_friends = g1.allV?[@n=="Cindy"]|outE?[@rel=="friend"]|endV|outE?[@rel=="friend"]|endV?[@n!="Cindy"];
447 print_vlist(vl_Cindy_friends_friends);
448
```



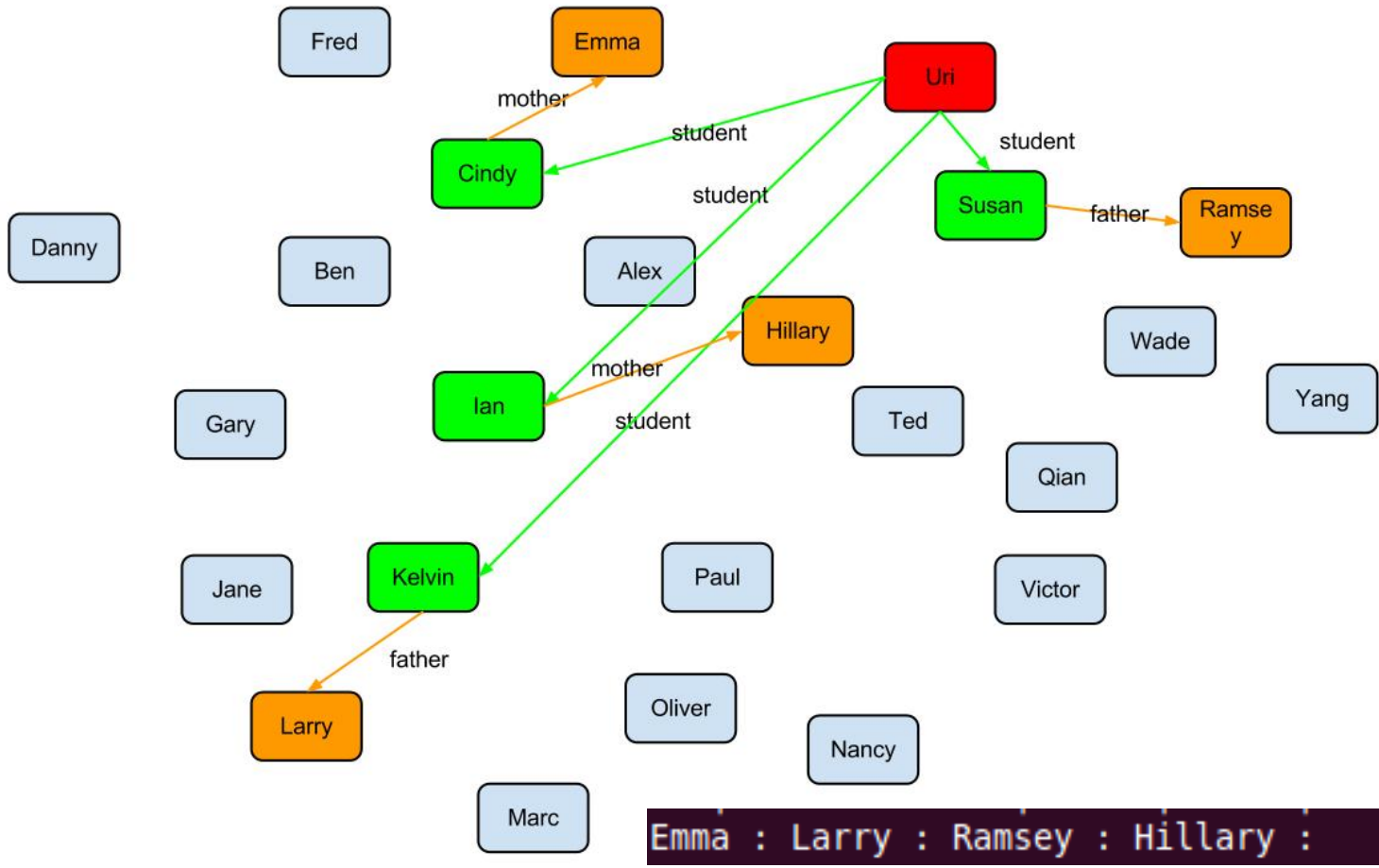

```
Kelvin : Susan : Qian : Alex : Jane : Ted : Ben : Oliver :
```

```
449 vlist vl_Cindy_friends_friends_2 = g1.allV?[@n=="Cindy"]|outE?[@rel=="friend"]|endV|outE?[@rel=="friend"]|endV?[@n!="Cindy"&&(@g=="f"||@age>23)];
450 print_vlist(vl_Cindy_friends_friends_2);
451
```




```
Susan : Qian : Jane : Ben : Oliver :
```

```
452 //get student's parent
453 vlist vl_uri_student_parent = g1.allV?[@n=="Uri"]|outE?[@rel=="student"]|endV|outE?[@rel=="father"||@rel=="mother"]|endV;
454 print_vlist(vl_uri_student_parent);
```

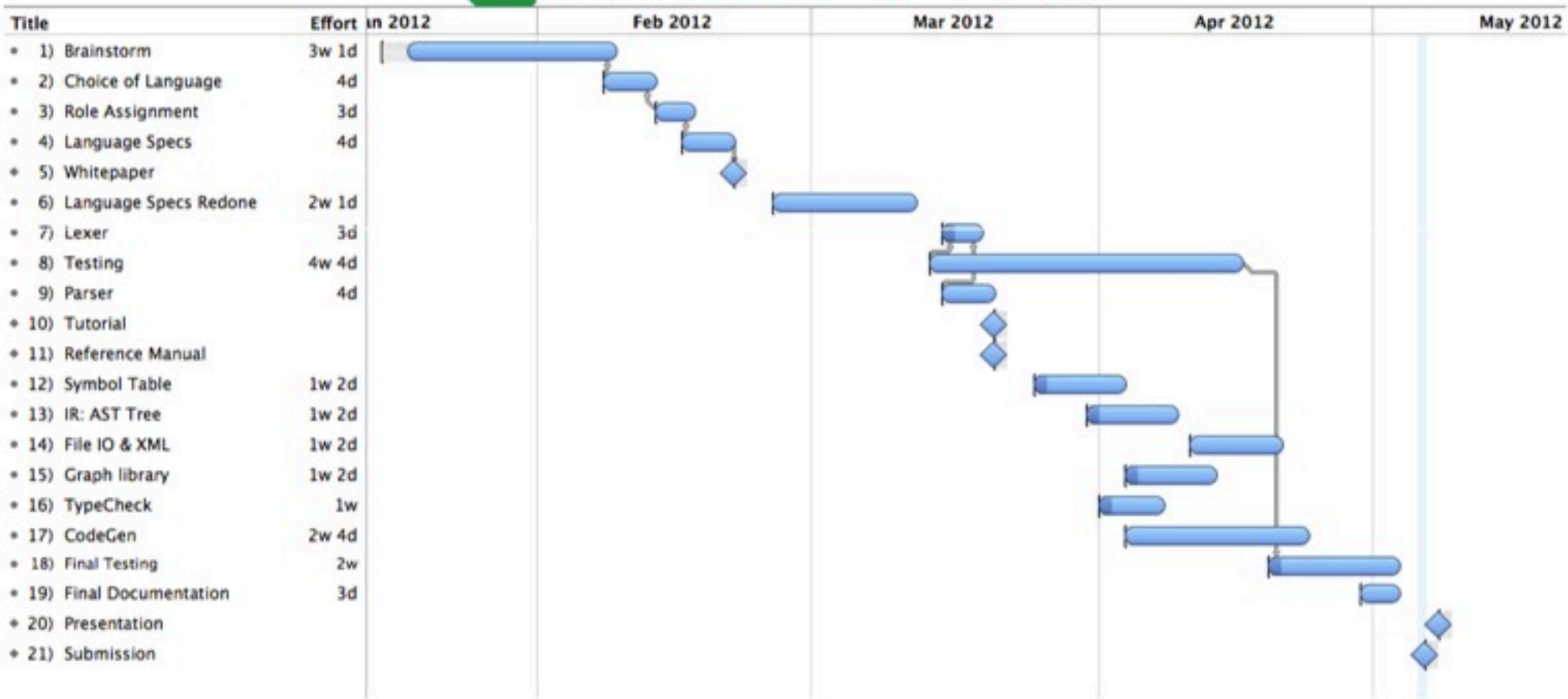



```
Emma : Larry : Ramsey : Hillary :
```

EXAMPLE 2: FILE I/O & BFS

Create a graph
Save it to the disk as XML
Read the XML and re-create a graph from it.
Do BFS on a specified vertex on it.

Conclusions



What we learnt

Keep it simple, or add complexity in layers

Start early

Try out things in the compiler by sections, not all at once.

Divide the work by people strengths.