

]

Design Patterns - Using FAST

Design Patterns in FAST

😊 *agentzh* 😊

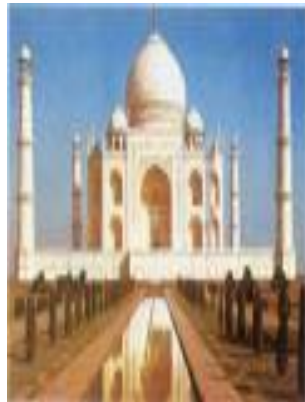
(章亦春)

2006.4

What is
Design Pattern?

Proven design idioms
for software development

Has its root in
architecture...



The *famous* book...

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



© 1995 by Addison-Wesley Longman, Inc.

Foreword by Grady Booch



ADDISON WESLEY PROFESSIONAL COMPUTING SERIES

Why *Design Pattern*?

Design Patterns

✓ Robustness

Something **proven** to work
is always *nice*!

Design Patterns

✓ Efficiency

We needn't start from *scratch*
every time!

Design Patterns

✓ Laziness



Never reinvent
the wheels!

But...

wait...

What is

FAST?

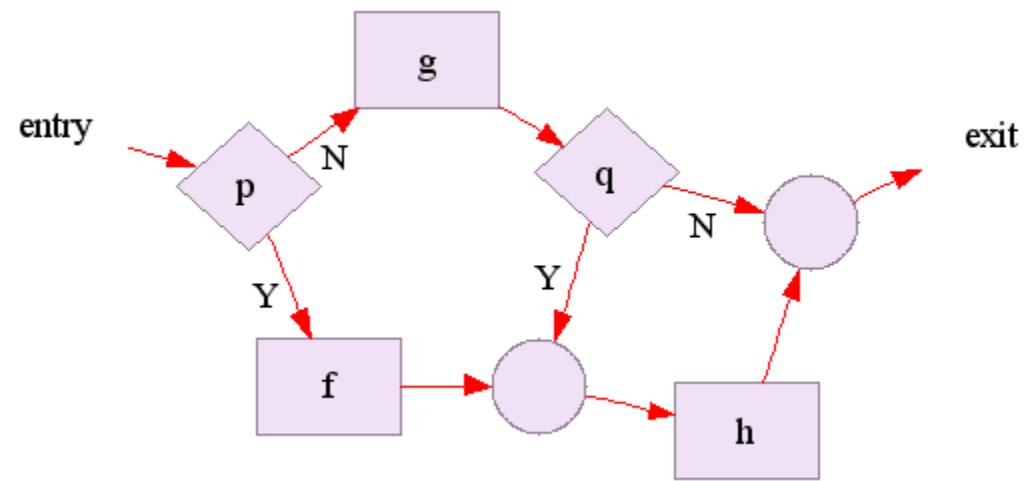
Flowchart Abstract Syntax Tree *transformer*



Recall our *second* homework
for Programming Methodology...

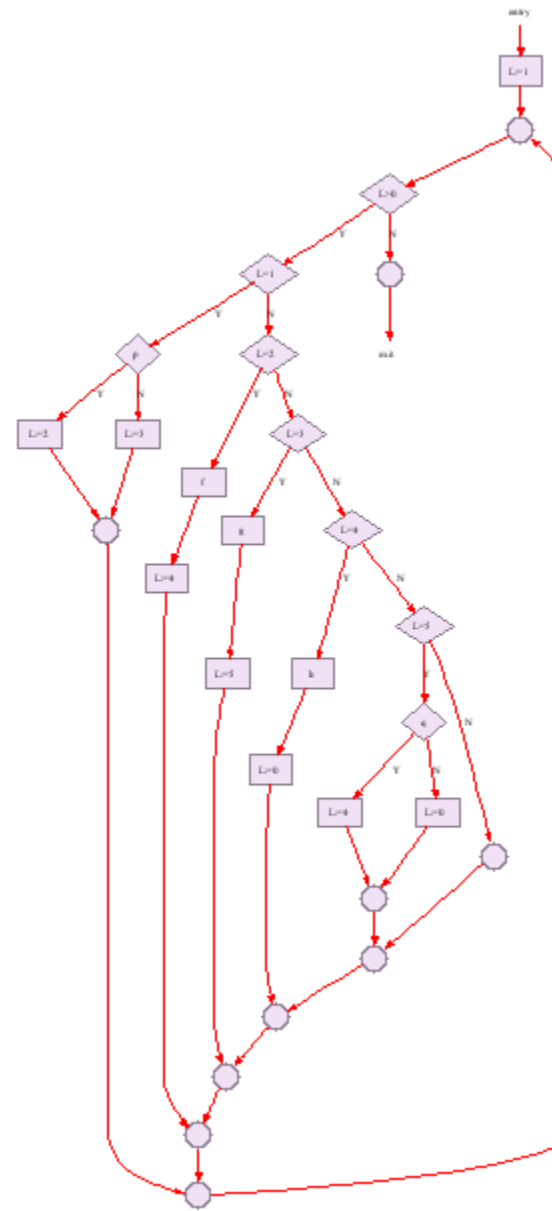
Given an arbitrary
flowchart program





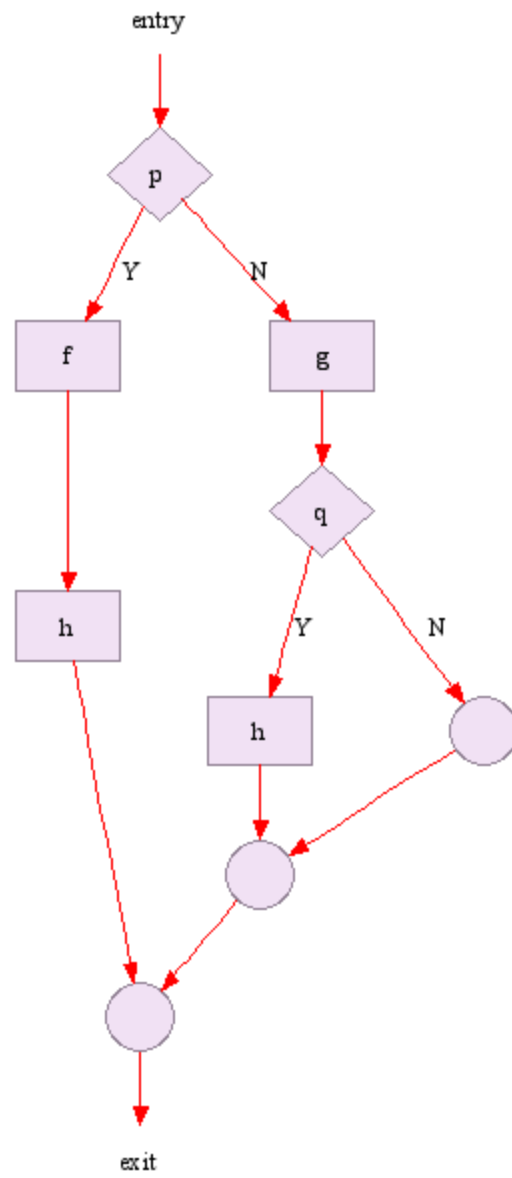
We were asked to convert that to
a *structural* program





And also a *recursive*
structural program







Any problems of this category
can be solved by **FAST** !

FAST is powered by
mature design patterns

FAST

✓ Robustness

FAST passes 500+ tests

FAST

✓ Efficiency

FAST's core was done
in 3 days!

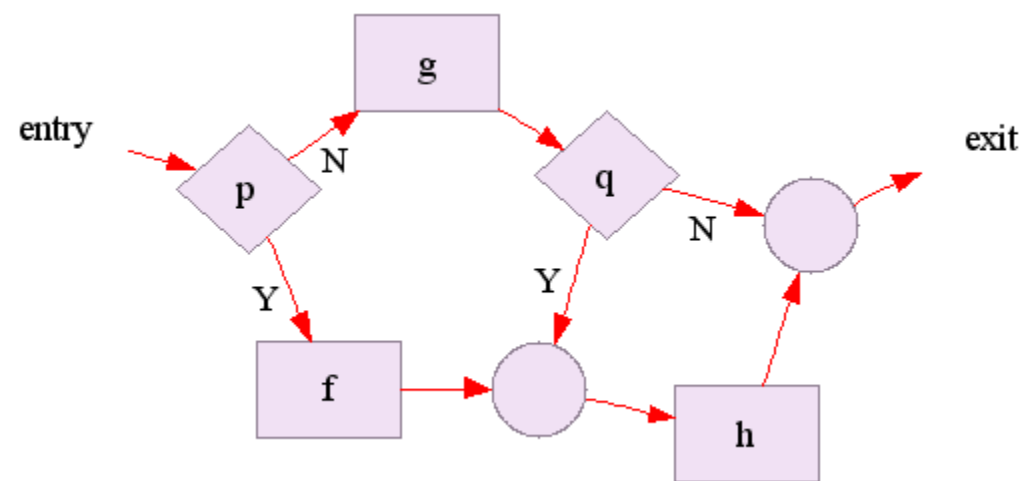
FAST

✓ Laziness

FAST is a frivolous project
optimized for *fun*

One Problem...

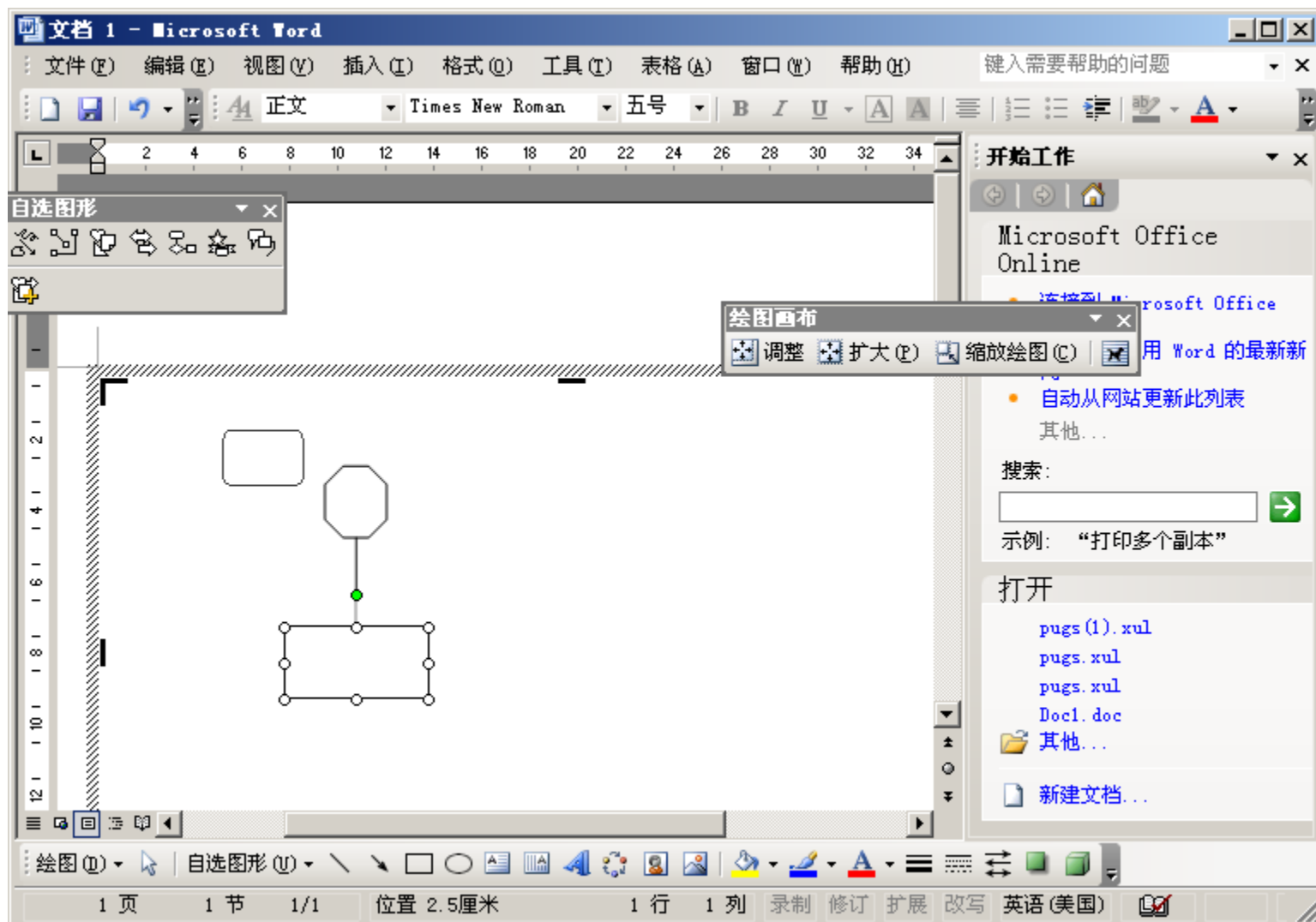
How to *type* the
original flowchart *into*
the machine?



Keyboard sucks...

Drag and drop by
a mouse ?

As in...



GUI *rocks*
for interfacing **beginners**

Hmm, however...



 Economy ?

☹ Scalability?

☹ flexibility?

☹ Testability?

GUI *sucks*
for *prototyping* FAST

GUI *sucks*
for testing FAST

GUI *sucks*
for interfacing apps

GUI *sucks*
for experienced users

Let's look *back*...

Does keyboard
really suck ?

Pattern #1

♡ Interpreter

Given a language, define a representation for its *grammar* along with an *interpreter* that uses the representation to interpret sentences in the language.

Make a language of
my own !

That's a *dream*
in my *childhood...*



Perhaps of **all** the creations of man
language is the most astonishing.

-- *Giles Lytton Strachey*

Why yet another
language?

FAST has a
user language
for entering flowcharts

entry => <p>

<p> => [f]

[f] => [h]

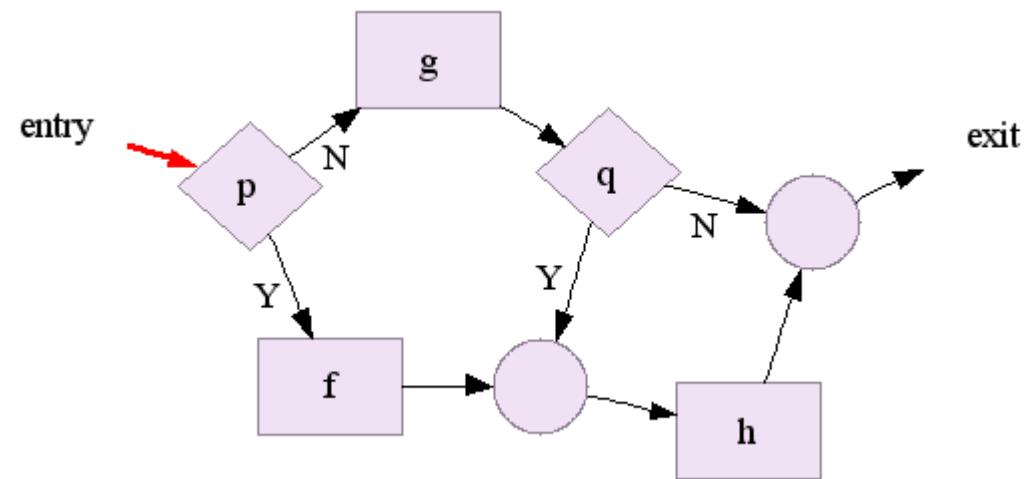
[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

<q> => exit



entry => <p>

<p> => [f]

[f] => [h]

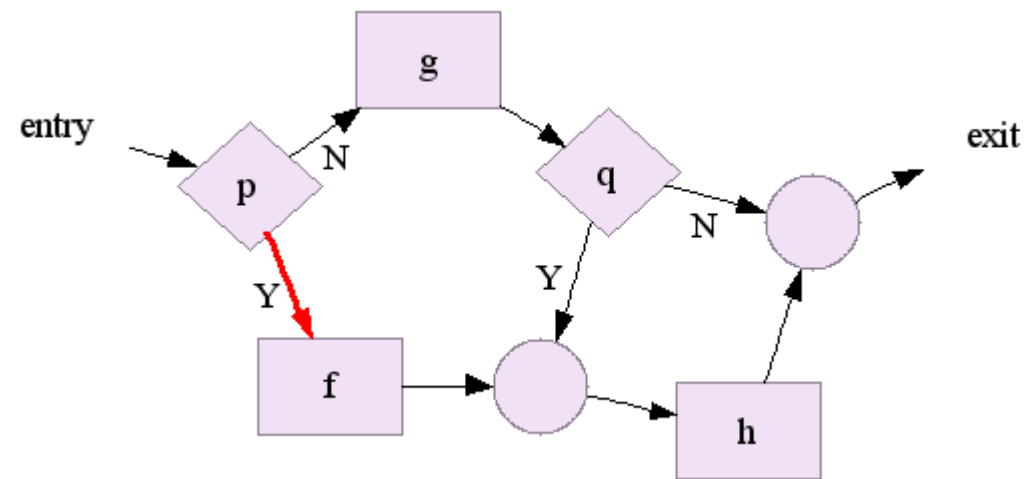
[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

<q> => exit



entry => <p>

<p> => [f]

[f] => [h]

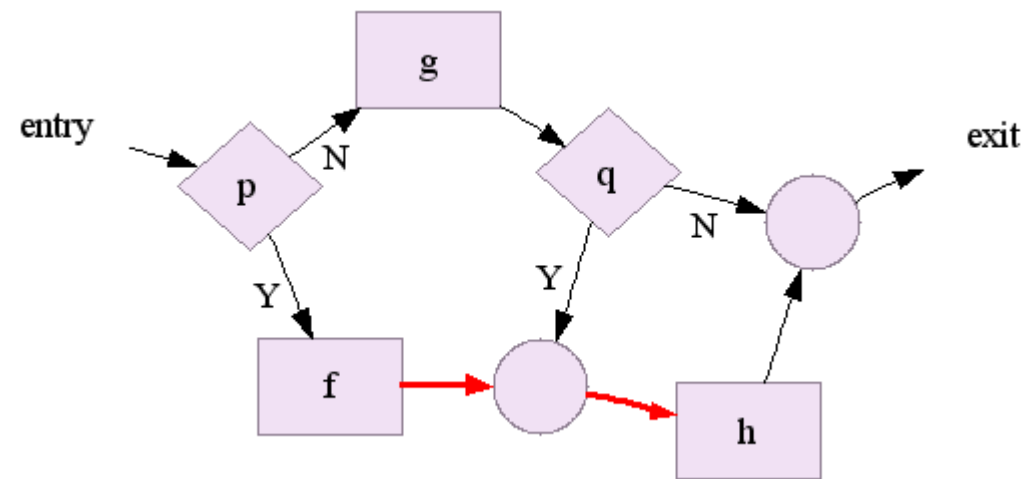
[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

<q> => exit



entry => <p>

<p> => [f]

[f] => [h]

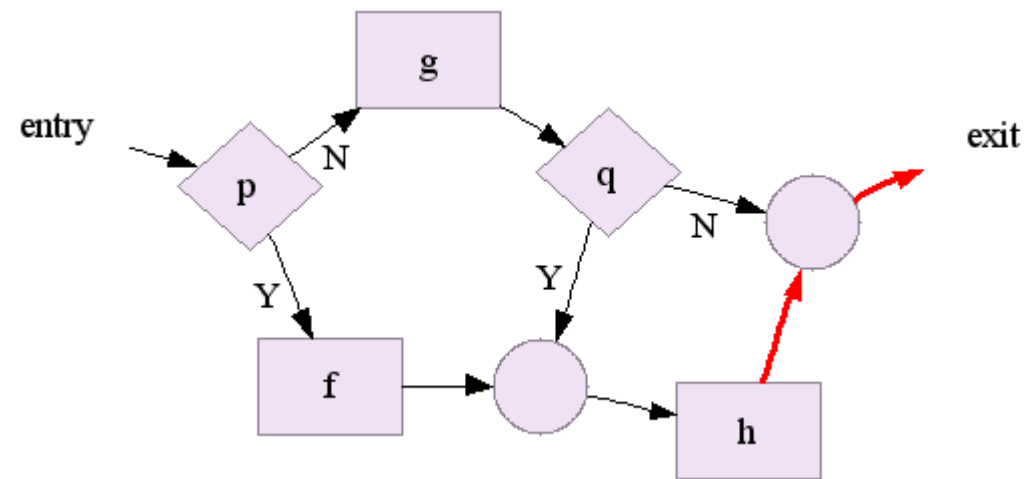
[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

<q> => exit



entry => <p>

<p> => [f]

[f] => [h]

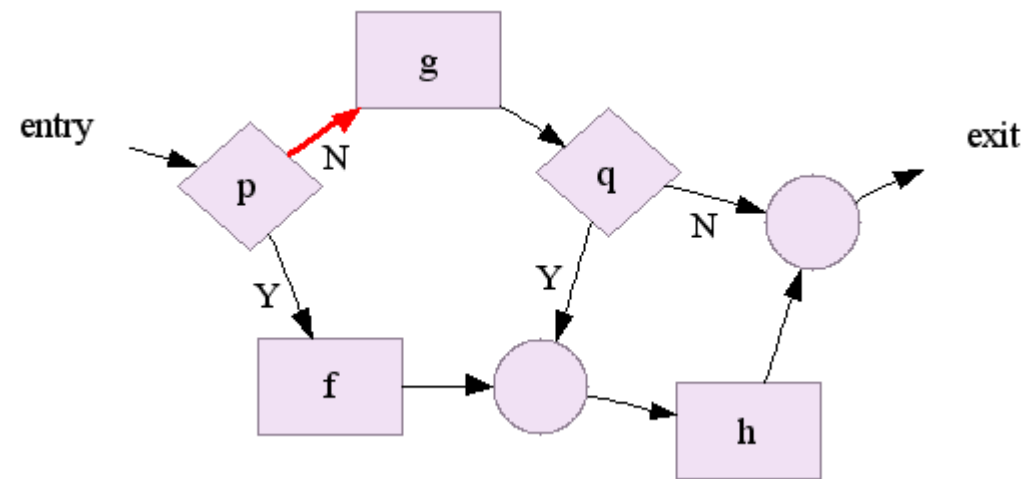
[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

<q> => exit



entry => <p>

<p> => [f]

[f] => [h]

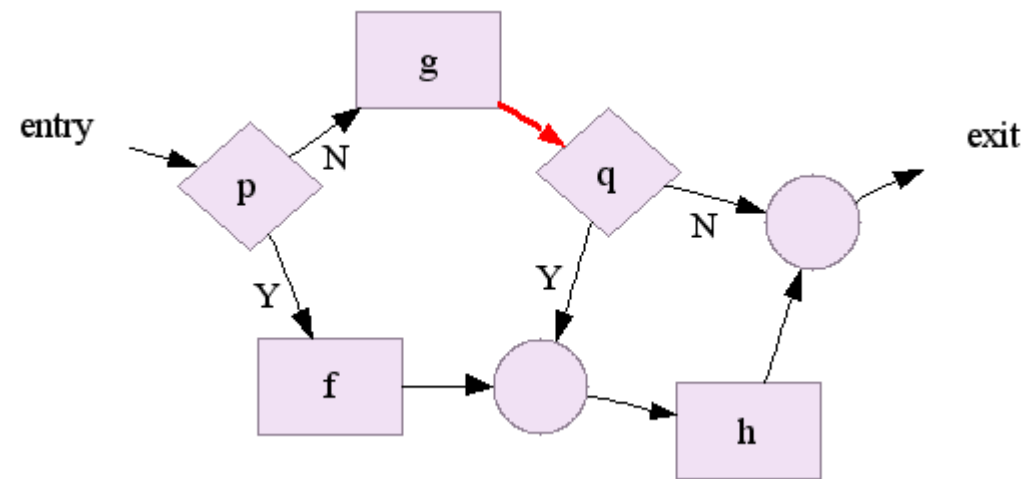
[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

<q> => exit



entry => <p>

<p> => [f]

[f] => [h]

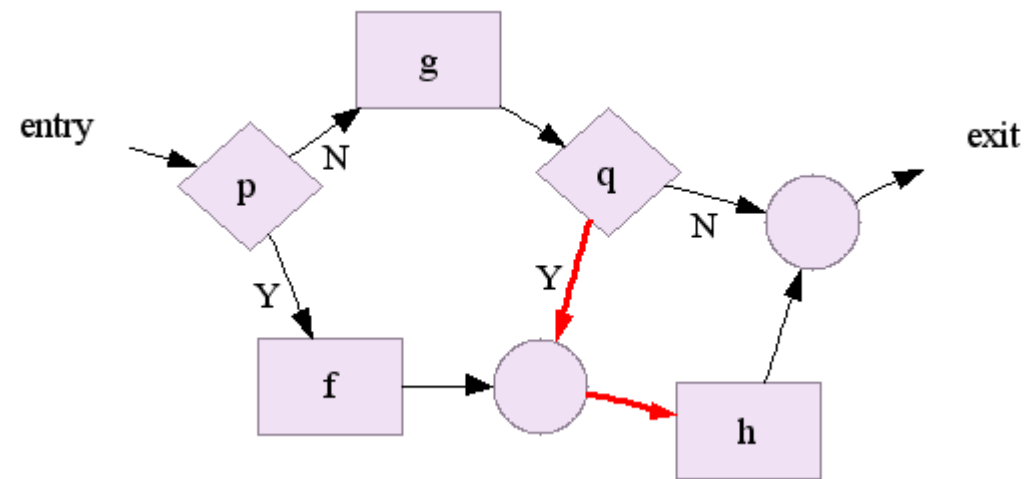
[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

<q> => exit



entry => <p>

<p> => [f]

[f] => [h]

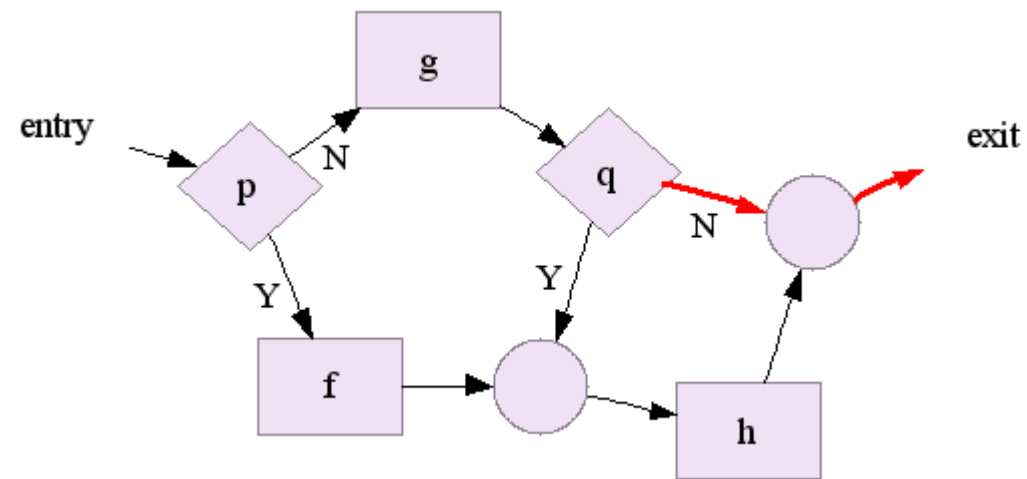
[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

<q> => exit



The *grammar* for this
mini-language...

program : statement(s)

statement : node '='>' node

node : 'entry'

| 'exit'

| '[' string ']'

| '<' string '>'

string : char(s)

How to *implement*
this grammar ?

It's *trivial* if you're
using Perl!

It only costs me
21
lines of code !



```

1 sub parse {
2     my( $self, $infile ) = @_
3     open $in, $infile or die $!;
4     my( %edge_from, %edge_to );
5     while (<$in) {
6         if (/^\s* (.*\S) \s* => \s* (.*\S) \s*$/x) {
7             my( $from, $to ) = ( $1, $2 );
8             $edge_from{$to} ||= [];
9             $edge_from{$from} ||= [];
10            $edge_to{$from} ||= [];
11            $edge_to{$to} ||= [];
12            push @{$edge_from{$to}}, $from;
13            push @{$edge_to{$from}}, $to;
14        } else {
15            parse_error $fname, "syntax error: $_";
16        }
17    }
18    close $in;
19    $self->{edge_from} = \%edge_from;
20    $self->{edge_to} = \%edge_to;
21 }

```

Basic Usage of FAST

1 input file ``*bar*''

entry => <p>

<p> => [f]

[f] => [h]

[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

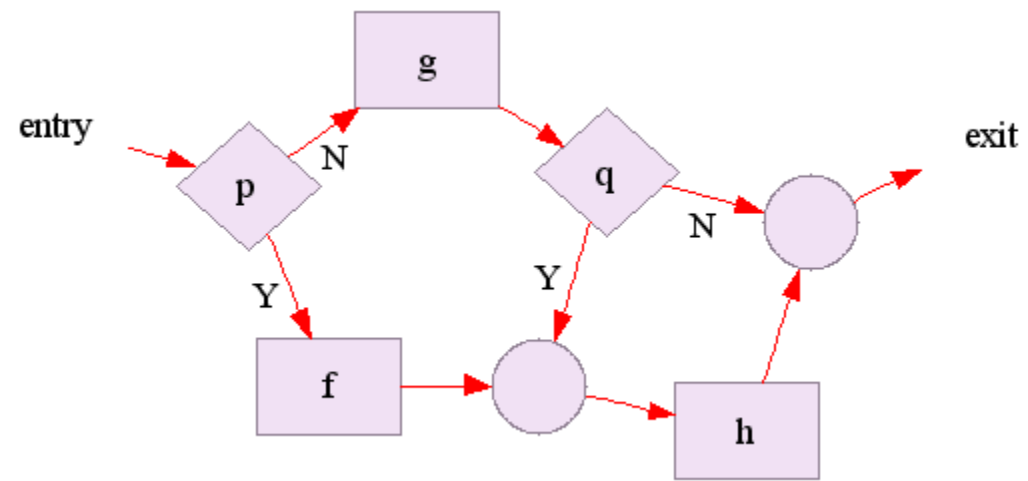
<q> => exit

C:\> fast *bar*



6 output files !

bar $\Rightarrow$ bar.png



bar $\Rightarrow$ bar.asm

test p

jno L1

do f

L2:

do h

L3:

exit

L1:

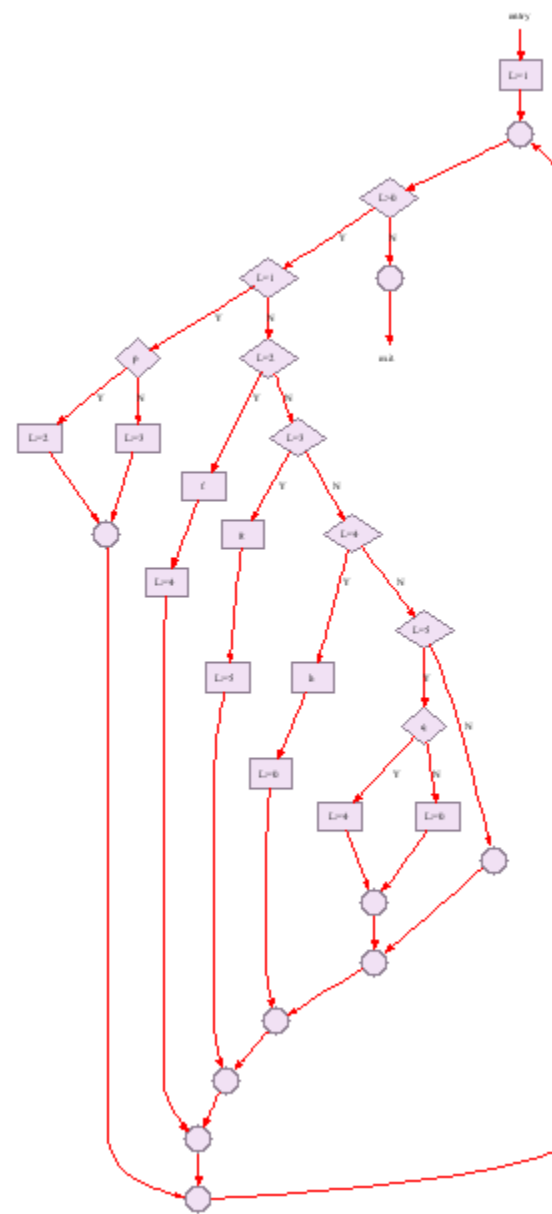
do g

test q

jno L3

jmp L2

bar $\Rightarrow$ bar.unopt.png



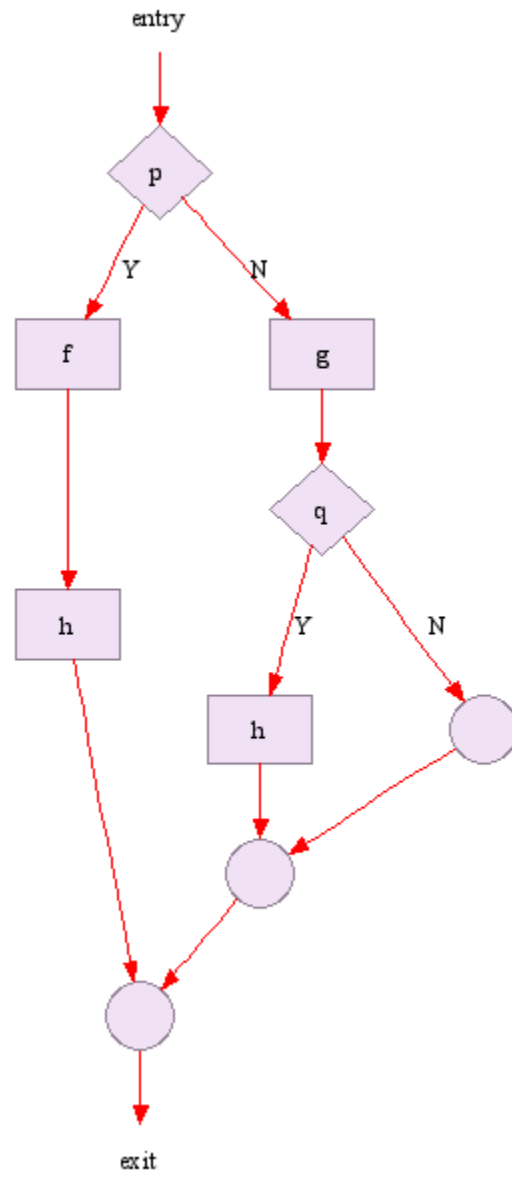
bar $\Rightarrow$ bar.unopt.c

```

do L:=1
while (L>0) {
  if (L=1) {
    if (p) {
      do L:=2
    } else {
      do L:=3
    }
  } else {
    if (L=2) {
      do f
      do L:=4
    } else {
      if (L=3) {
        do g
        do L:=5
      } else {
        if (L=4) {
          do h
          do L:=0
        } else {
          if (L=5) {
            if (q) {
              do L:=4
            } else {
              do L:=0
            }
          }
        }
      }
    }
  }
}

```

bar $\Rightarrow$ bar.opt.png



bar $\Rightarrow$ bar.opt.c

```
// bar.opt.c
```

```
if (p) {  
    do f  
    do h  
} else {  
    do g  
    if (q) {  
        do h  
    }  
}  
}
```

Looking into
the **FAST** *Internals*...

Structural flowcharts
are represented by
trees

What do those trees
look like?

Pattern #2

♡ Composite

Compose objects into *tree* structures to represent *part-whole* hierarchies. Composite lets clients treat individual objects and compositions of objects *uniformly*.

Yeah, that's a story
about *trees*...

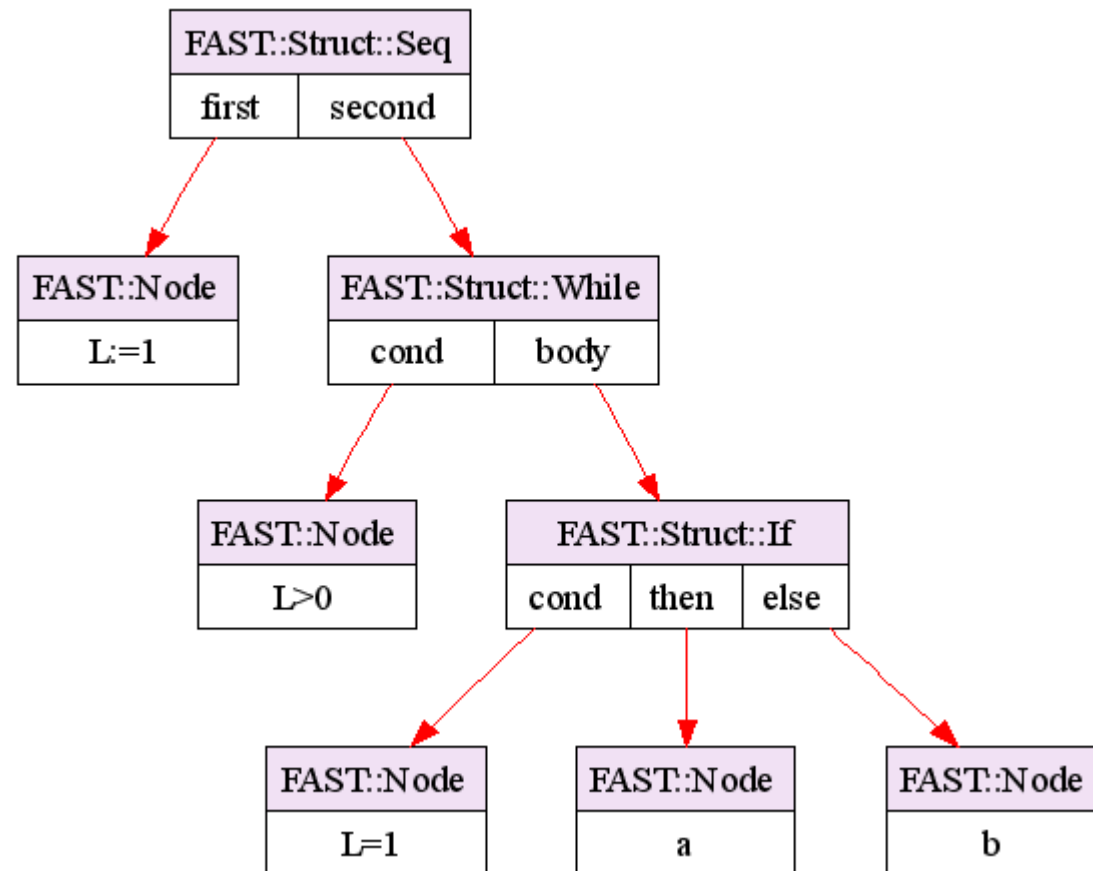


Every structural flowchart
can be represented by
a *tree*

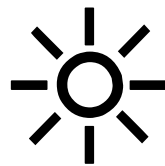
```

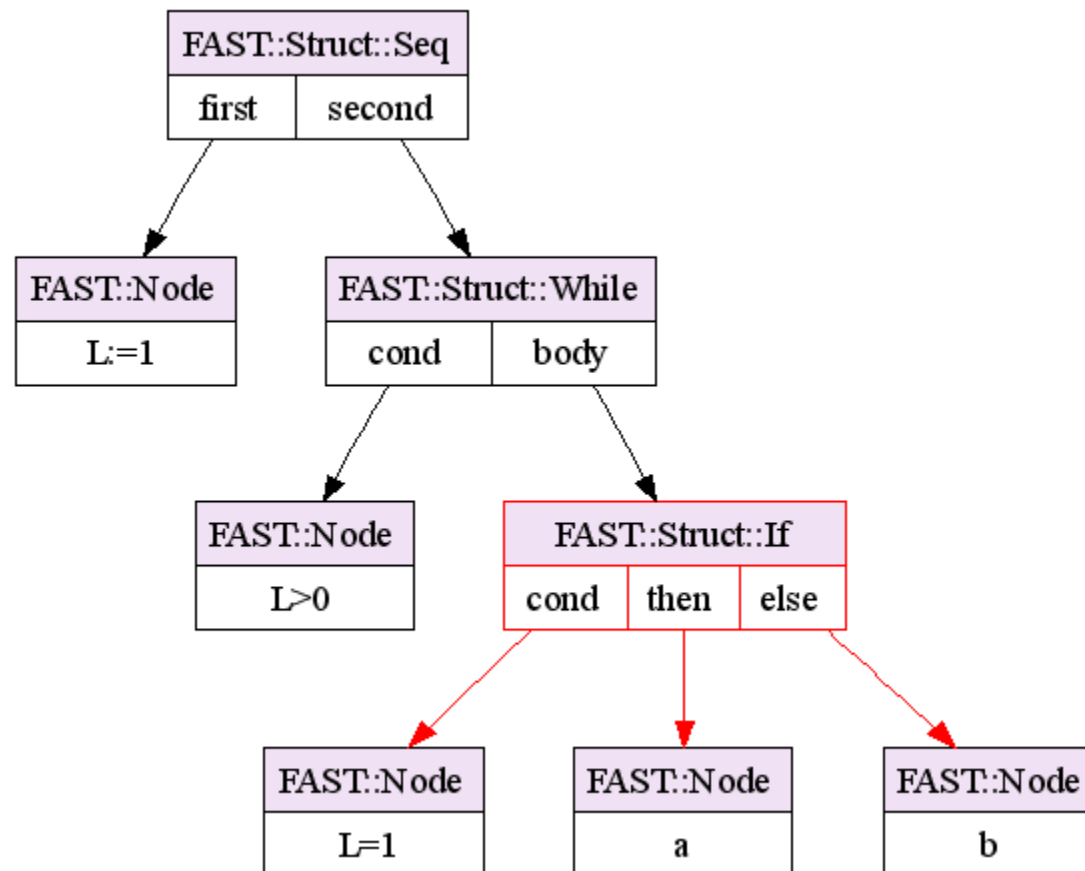
1  L:=1
2  while ( L>0 ) {
3      if ( L=1 ) {
4          do a
5      } else {
6          do b
7      }
8  }

```

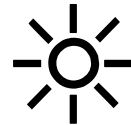


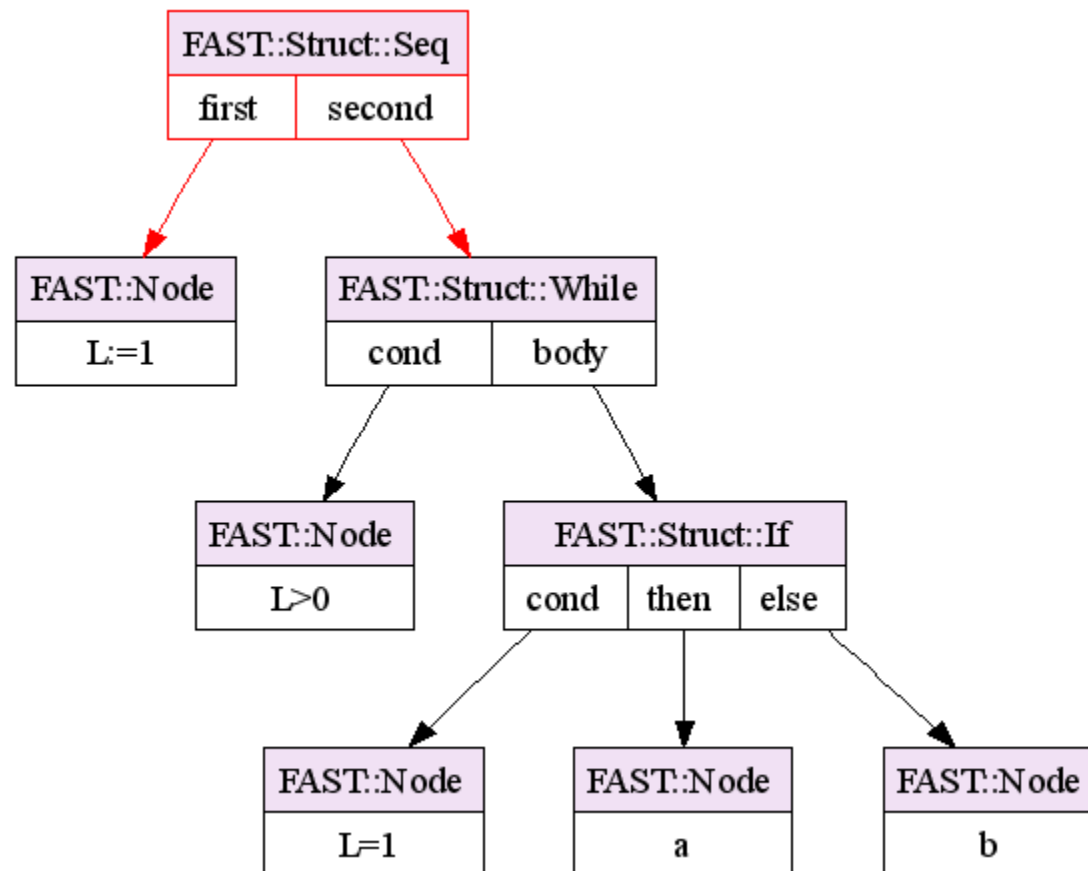
Every node is
also a **tree**.





Every tree is
represented by its
root node.





Hence...

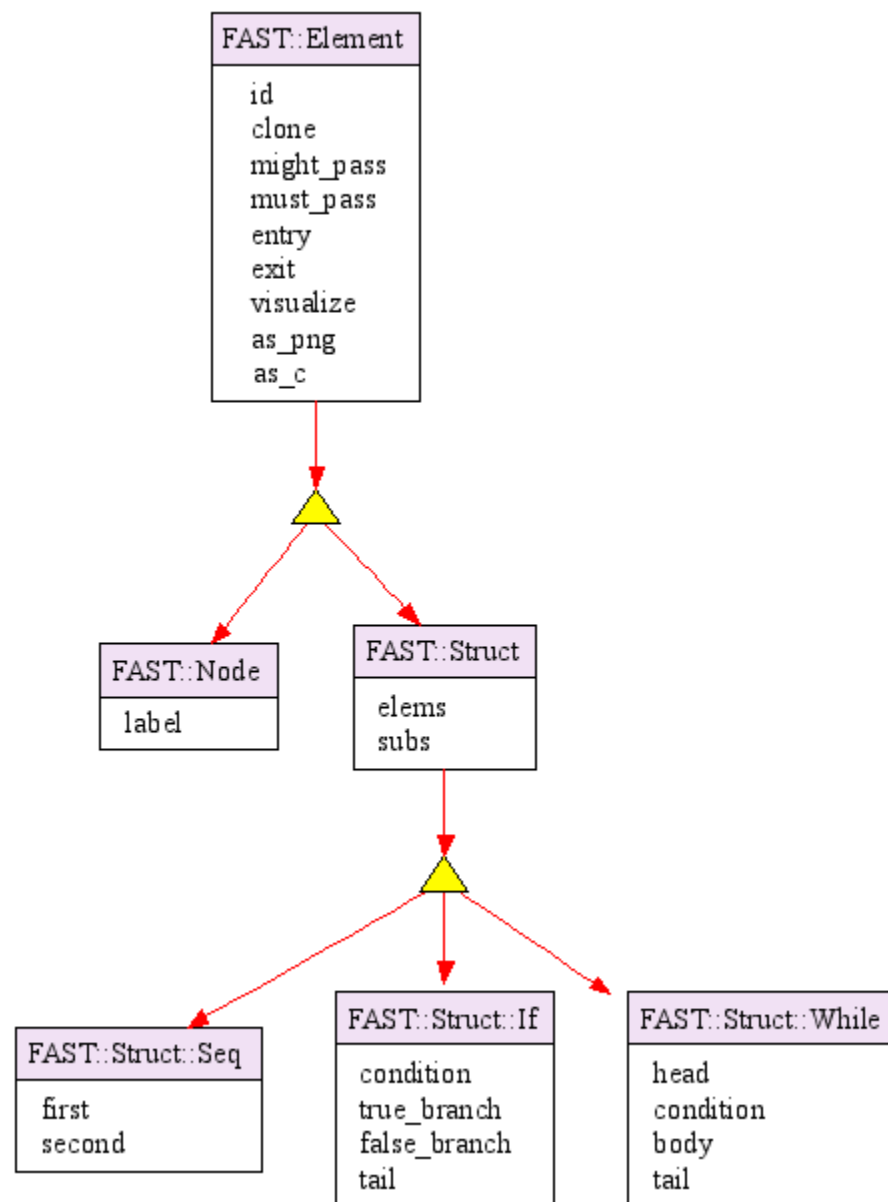


Trees and nodes
share the *same* interface.



There is *no*
Tree class any more

As evidenced in
the *UML* thingy...

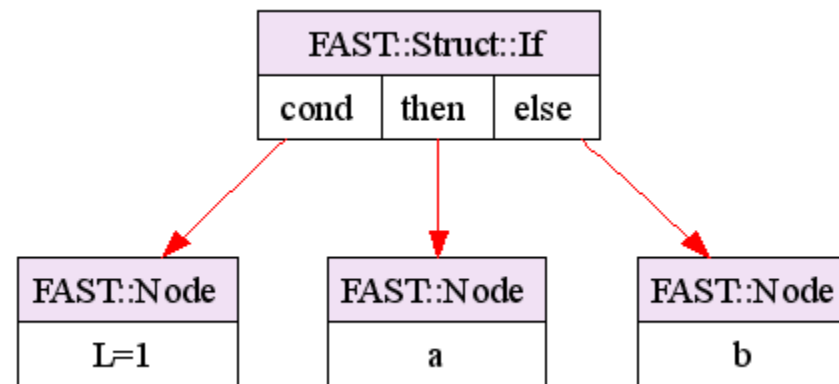


Composite Pattern

- ✓ Building the tree is easy

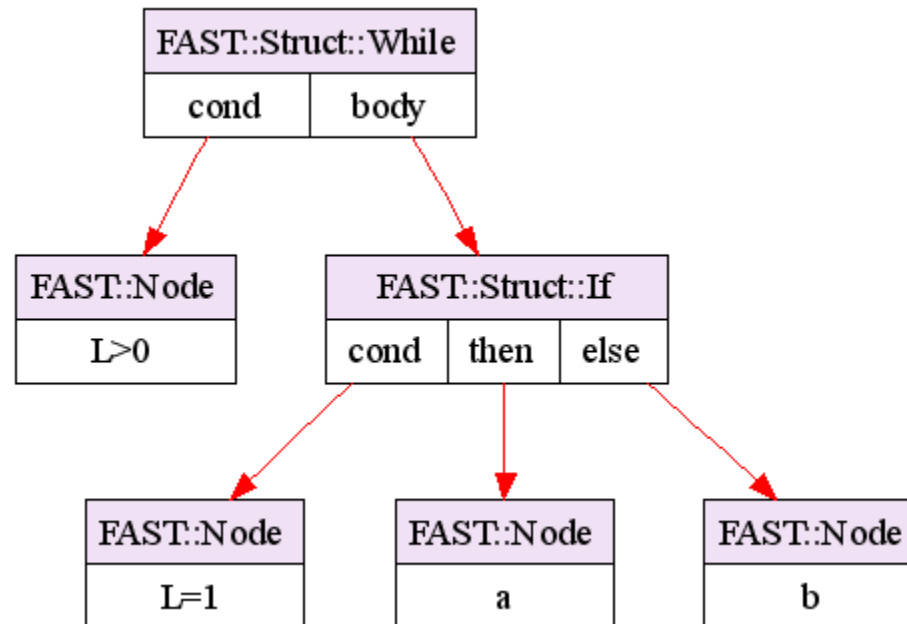
```
$if = FAST::Struct::If->new('<L=1>', '[a]', '[b]');
```

```
$if = FAST::Struct::If->new('<L=1>', '[a]', '[b]');
```



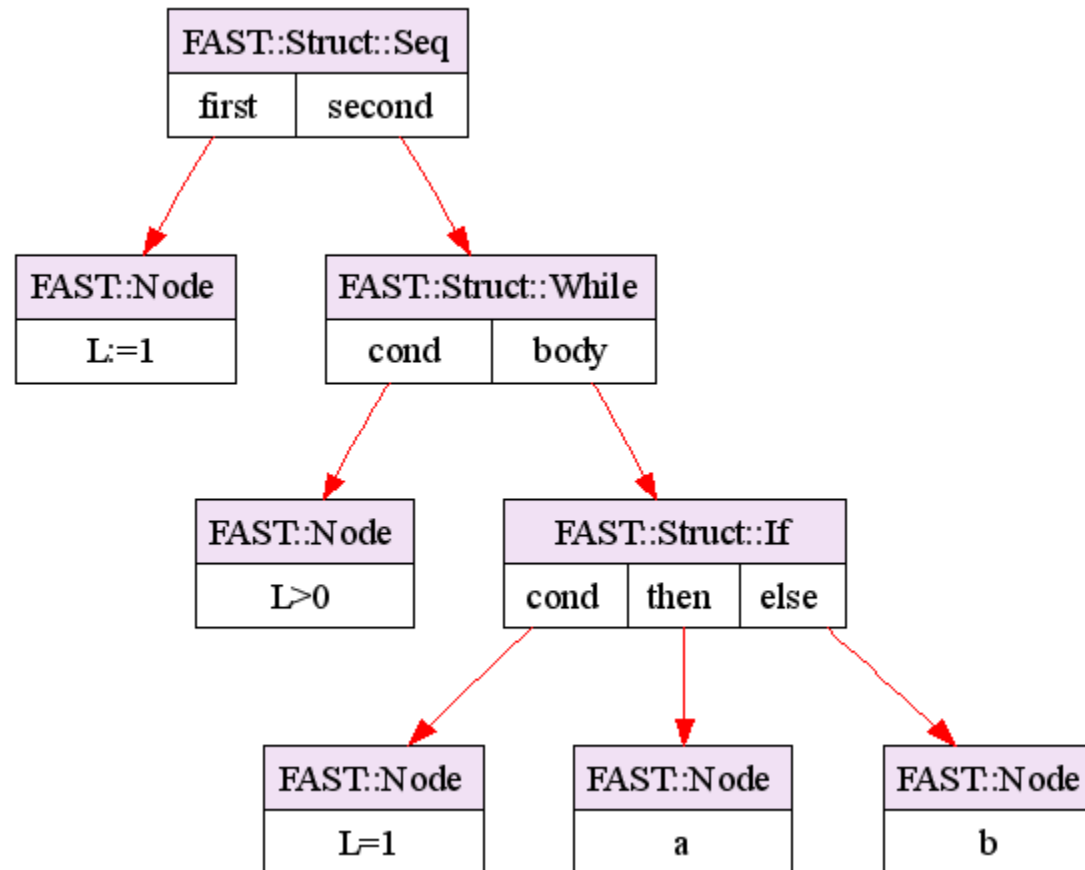
```
$if = FAST::Struct::If->new('<L=1>', '[a]', '[b]');  
$while = FAST::Struct::While->new('<L>0>', $if);
```

```
$if = FAST::Struct::If->new('<L=1>', '[a]', '[b]');  
$while = FAST::Struct::While->new('<L>0>', $if);
```



```
$if = FAST::Struct::If->new('<L=1>', '[a]', '[b]');  
$while = FAST::Struct::While->new('<L>0>', $if);  
$seq = FAST::Struct::Seq->new('L:=1', $while);
```

```
$if = FAST::Struct::If->new('<L=1>', '[a]', '[b]');  
$while = FAST::Struct::While->new('<L>0>', $if);  
$seq = FAST::Struct::Seq->new('L:=1', $while);
```

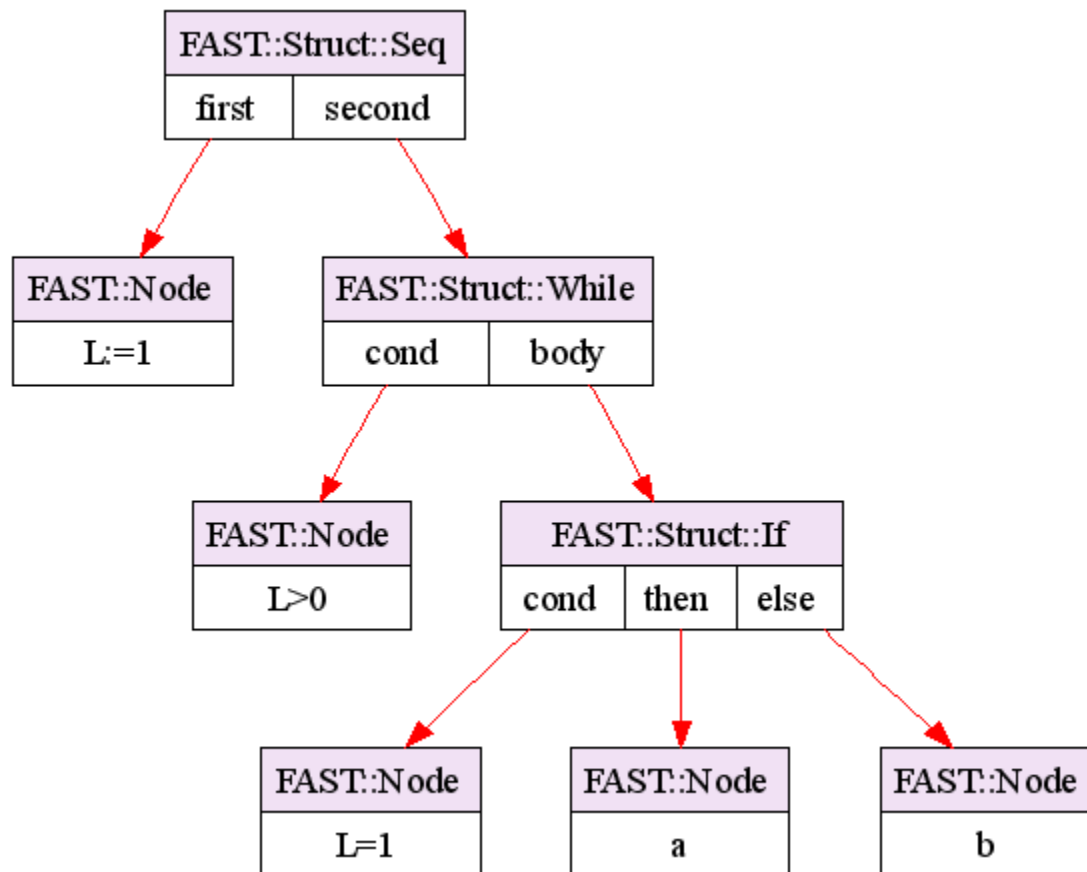


Composite Pattern

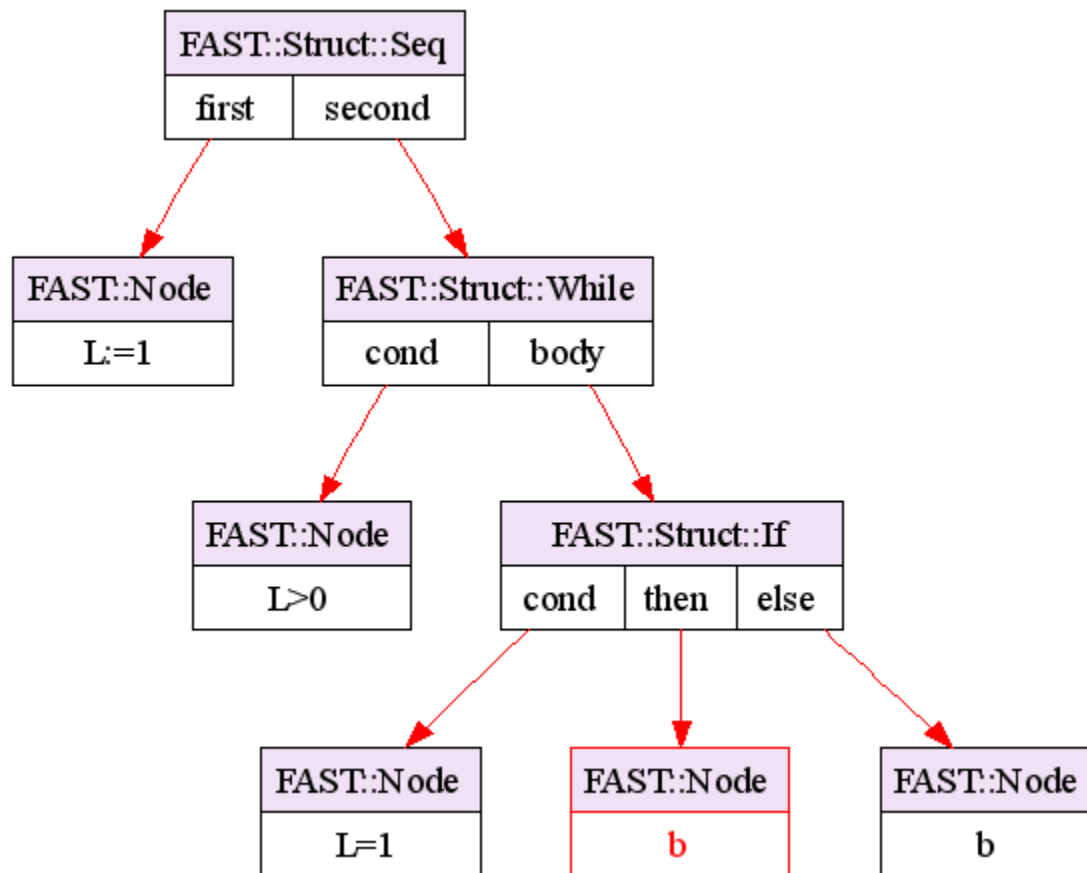
- ✓ Handling the tree is easy

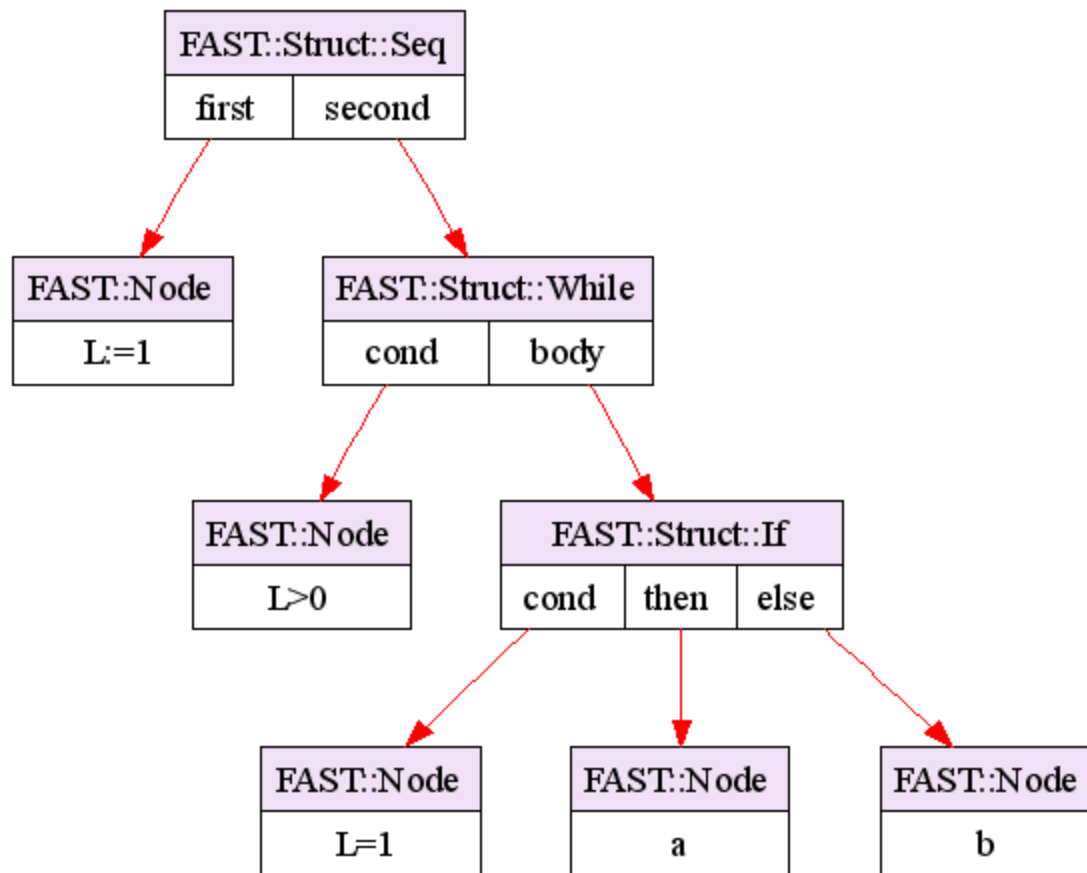
```
$tree1->subs($node_name, $tree2);
```

Substitute **\$tree2** for every node
named **\$node_name** in **\$tree1**.

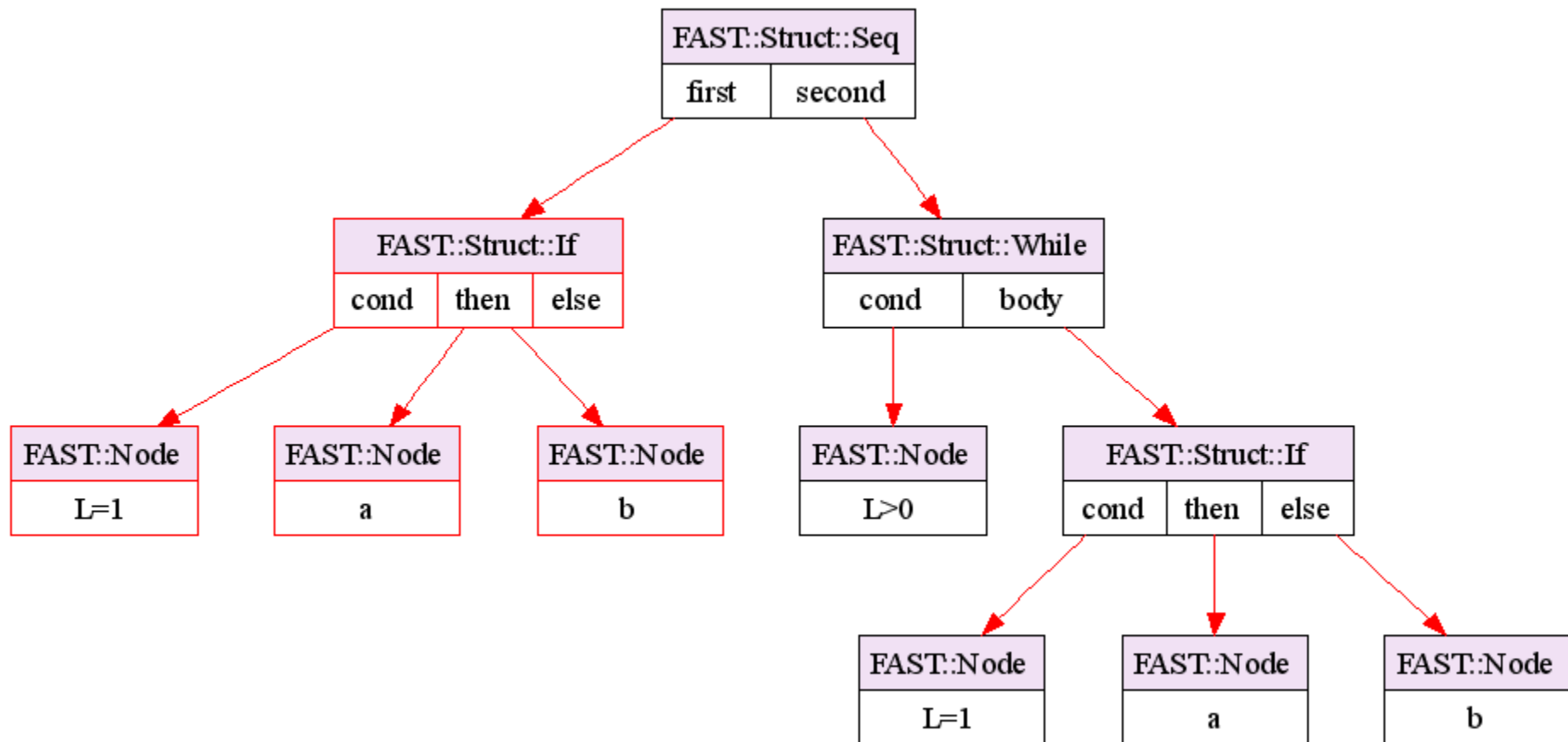


\$seq->subs(['a'], ['b']);





`$seq->subs(['L:=1'], $if);`



The **subs** method
is *also*

The **subs** method
is *also*
21 lines of code !

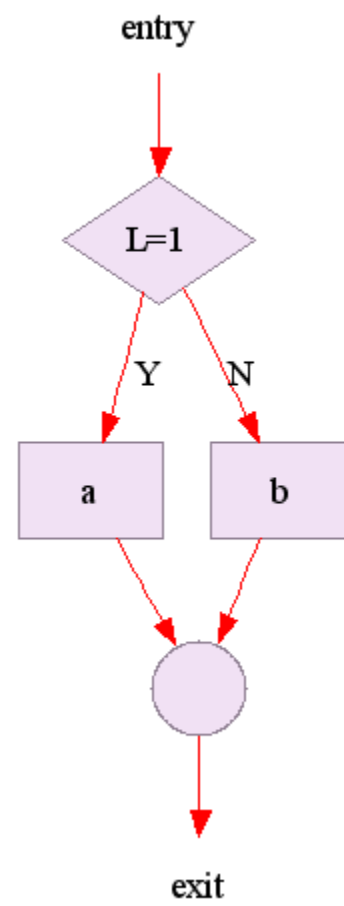
```
1 package FAST::Struct;
2
3 # Substitue a copy of $dest for every node named $label
4 # in the current FAST::Struct subtree:
5 sub subs {
6     my ($self, $label, $dest) = @_ ;
7     $dest = $self->_node($dest);
8     my $relems = $self->elems;
9     my $done;
10    for my $e (@$relems) {
11        if ($e->isa('FAST::Node')) {
12            if ($e->label eq $label) {
13                $e = $dest->clone;
14                $done = 1;
15            }
16        } else {
17            $done = $e->subs($label, $dest) || $done;
18        }
19    }
20    return $done;
21 }
```

Composite Pattern

- ✓ Visualizing the tree is easy

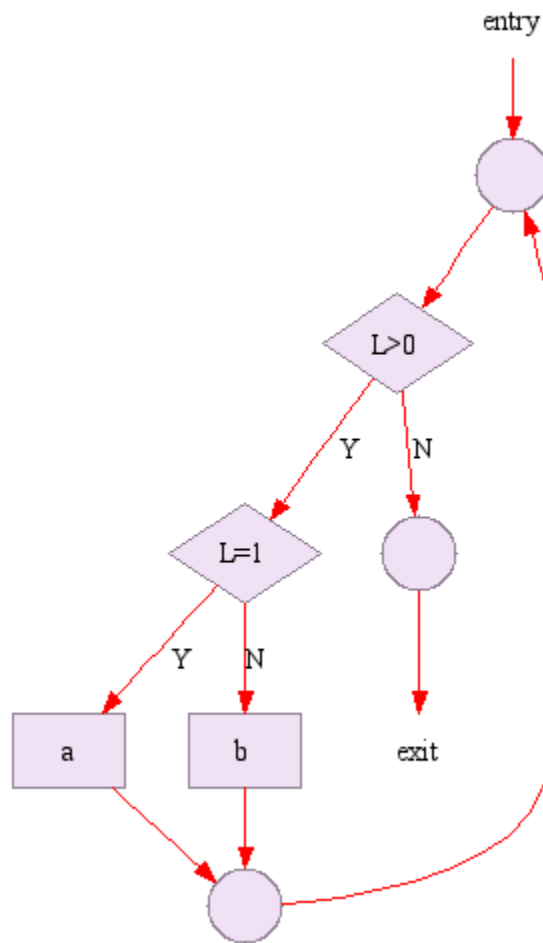
```
$if->as_png('if.png');
```

`$if->as_png('if.png');`



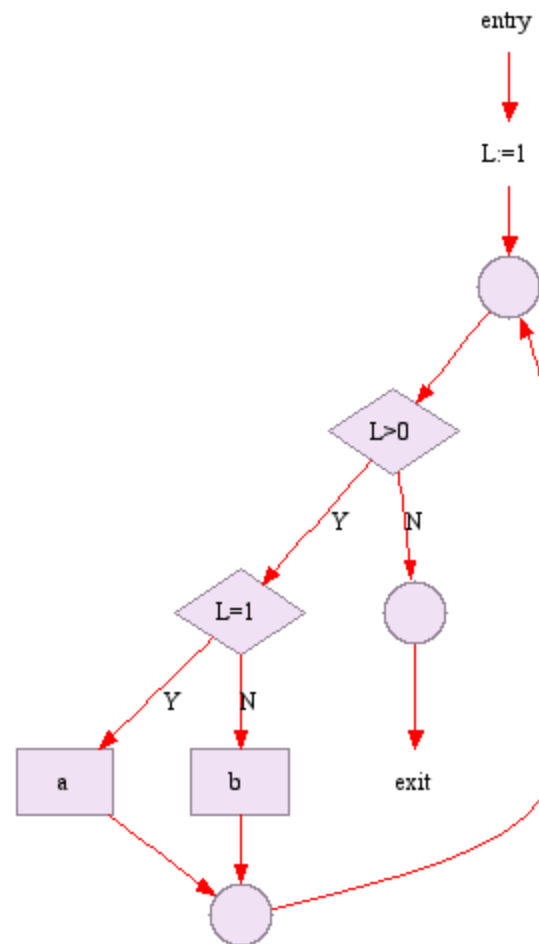
```
$while->as_png('while.png');
```

`$while->as_png('while.png');`



```
$seq->as_png('seq.png');
```

`$seq->as_png('seq.png');`



The operations of **trees** and **nodes**
are *unified* !



There are many, many
more design patterns
in **FAST**...

You can find them
yourself.



Get **FAST** and the **slides** today!



<http://agentzh.org/misc/slides/patterns.pdf>

These slides are
powered by
Sporx and...

Takahashi++



Most of the **images** used here
were *dawn* by
AT&T's **Graphviz**

Thank you!



