Design Patterns - Using FAST

Design Patterns in ☺ FAST ☺

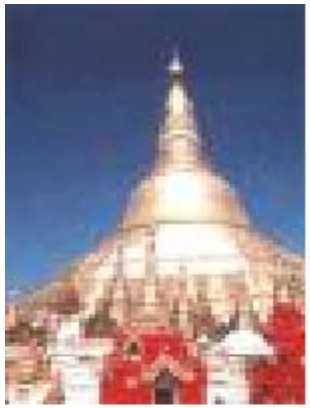
Agent Zhang

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

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...

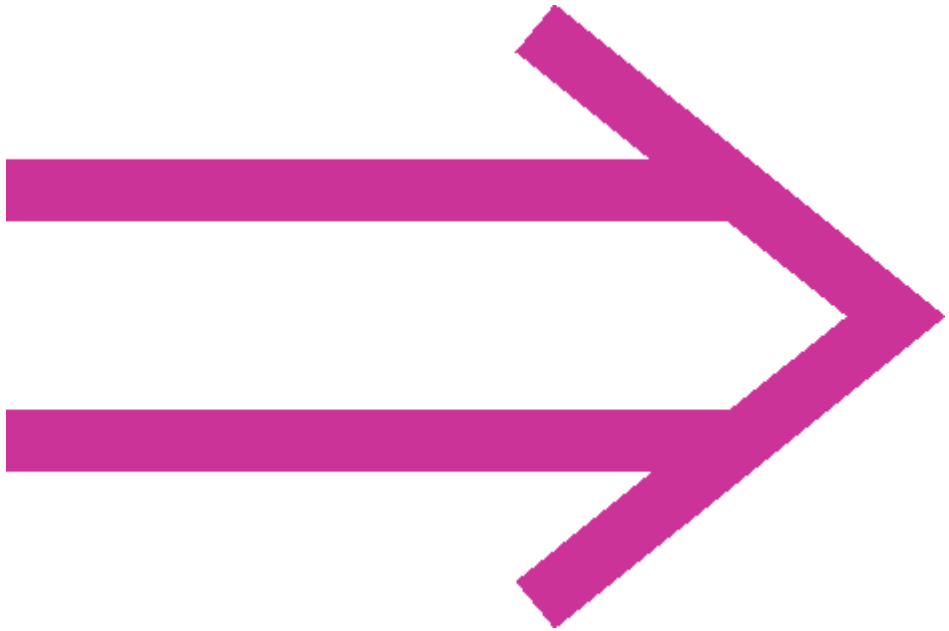
What is FAST?

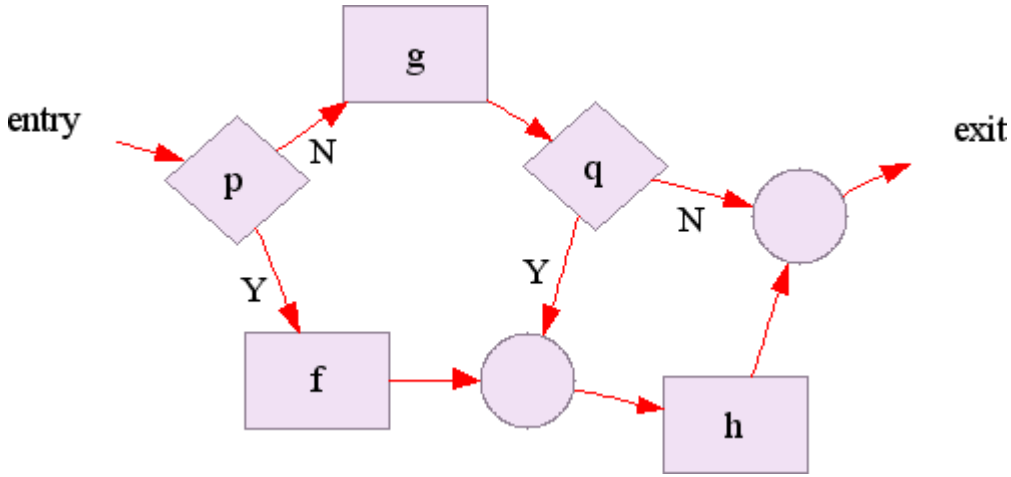
Flowchart Asbtract Syntax Tree

transformer

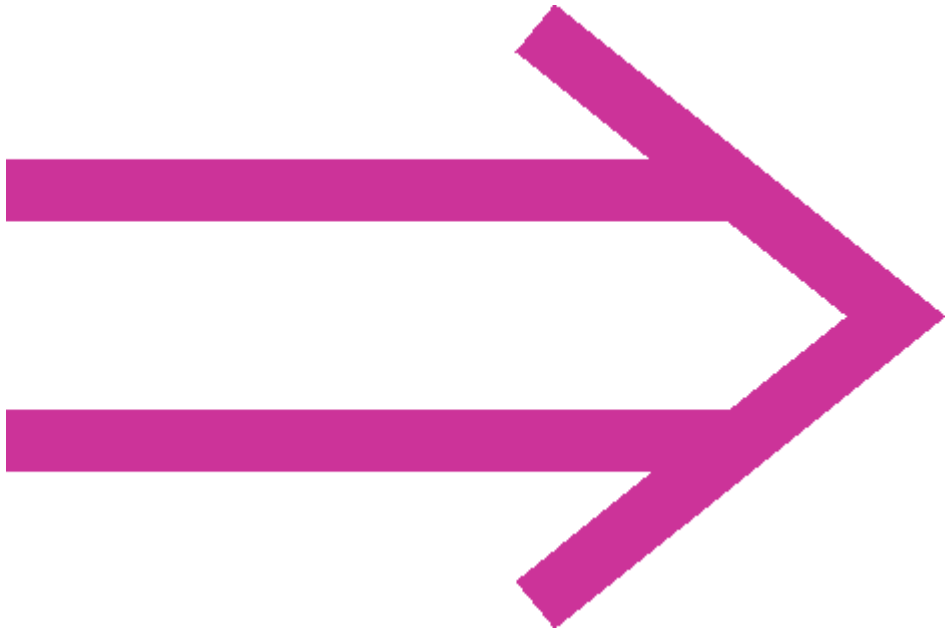
Recall our *second* homework for Programming Methodology...

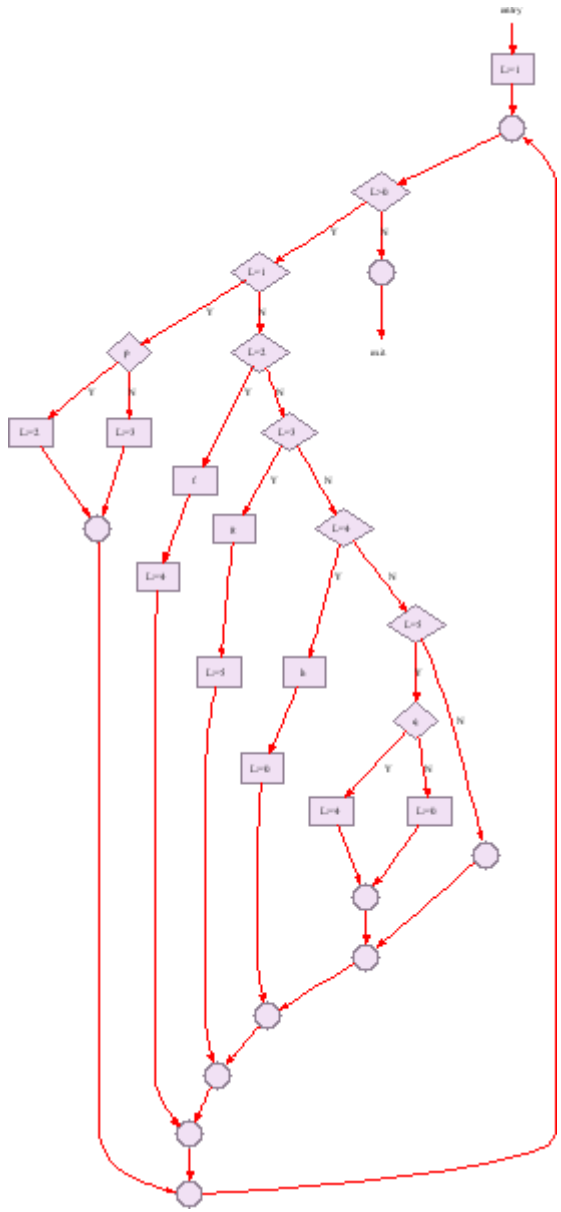
Given an arbitrary flowchart program



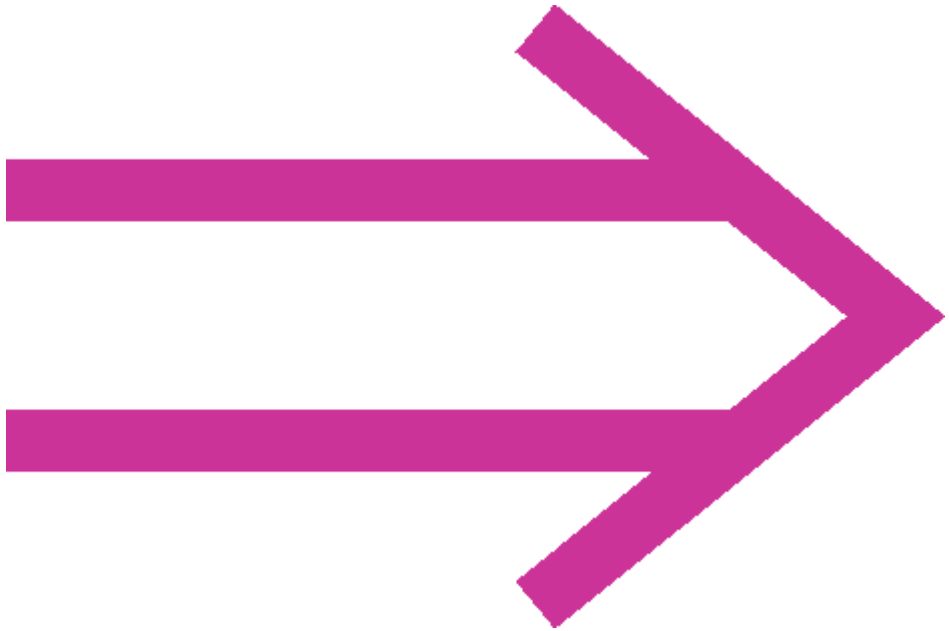

entry
p
g
N
Y
f
q
Y
N
exit
h

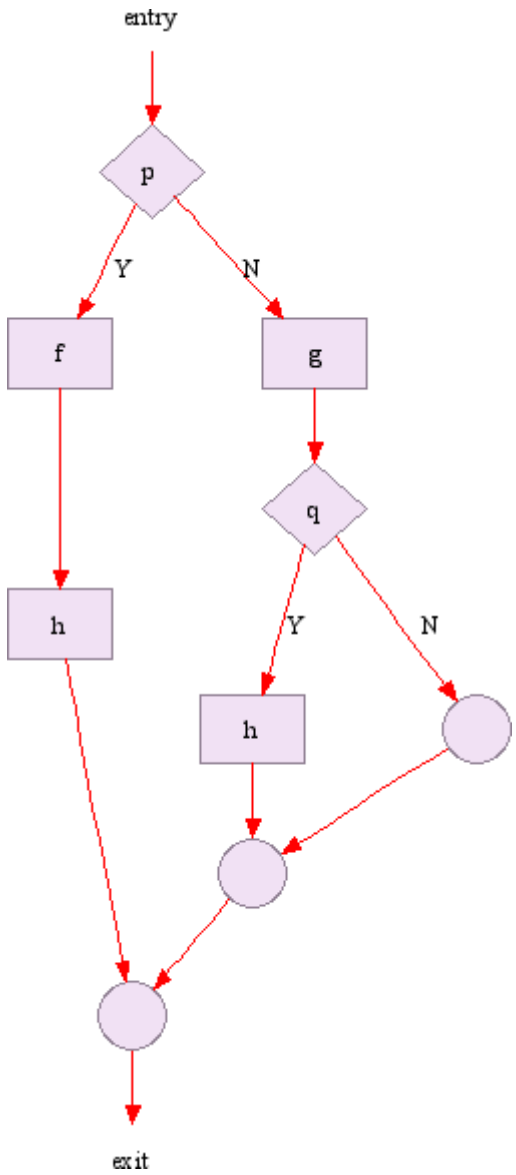
We were asked to convert that to a *structural* program




entry
L=1
L=6
Y
N
L=1
Y
N
exit
P
L=2
Y
N
L=3
L=2
Y
N
L=3
C
Y
N
L=4
B
L=4
Y
N
L=5
L=5
k
Y
N
L=6
L=4
L=6

And also a *recursive* structural program




entry
p
Y
N
f
g
q
h
Y
N
h
exit

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!

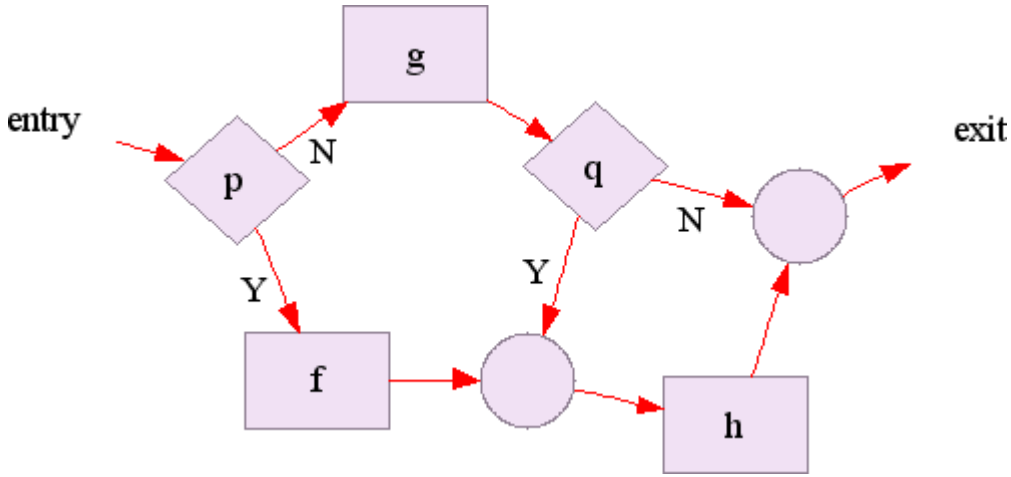
FAST

✓ Laziness

FAST is a frivolous project optimized for *fun*

One Problem…

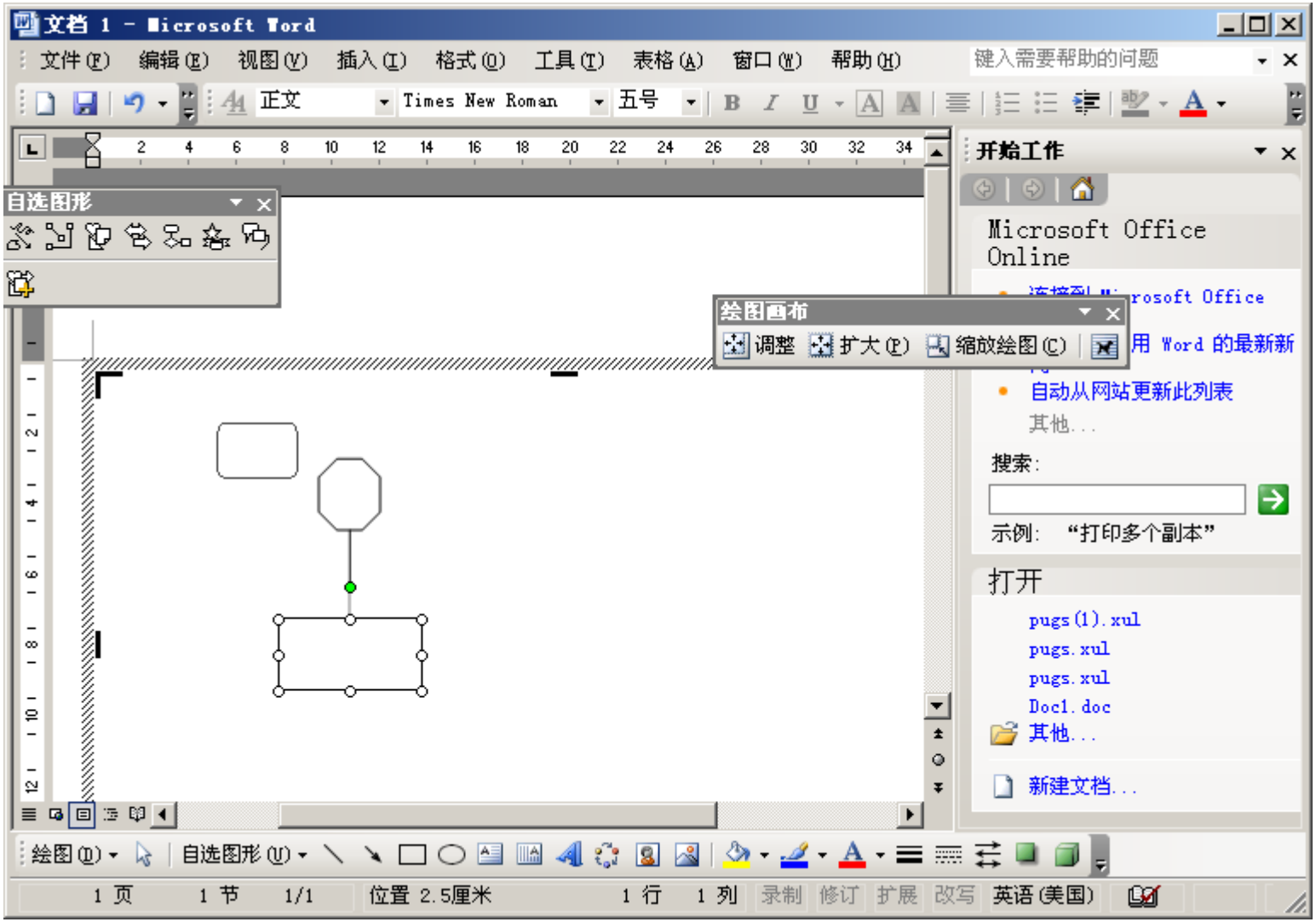
How to *type* the original flowchart *into* the machine?


entry
p
g
N
Y
f
q
Y
N
exit
h

Keyboard sucks...

Drag and drop by a mouse ?

As in...


文档 1 - Microsoft Word
文件(F) 编辑(E) 视图(V) 插入(I) 格式(O) 工具(T) 表格(A) 窗口(W) 帮助(H)
键入需要帮助的问题
正文
Times New Roman
五号
自选图形
绘图画布
调整
扩大(P)
缩放绘图(C)
开始工作
Microsoft Office Online
自动从网站更新此列表
其他...
搜索:
示例: “打印多个副本”
打开
pugs(1).xul
pugs.xul
pugs.xul
Doc1.doc
其他...
新建文档...
绘图(D)
自选图形(U)
1 页 1 节 1/1 位置 2.5厘米 1 行 1 列 录制 修订 扩展 改写 英语(美国)

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 ?

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...

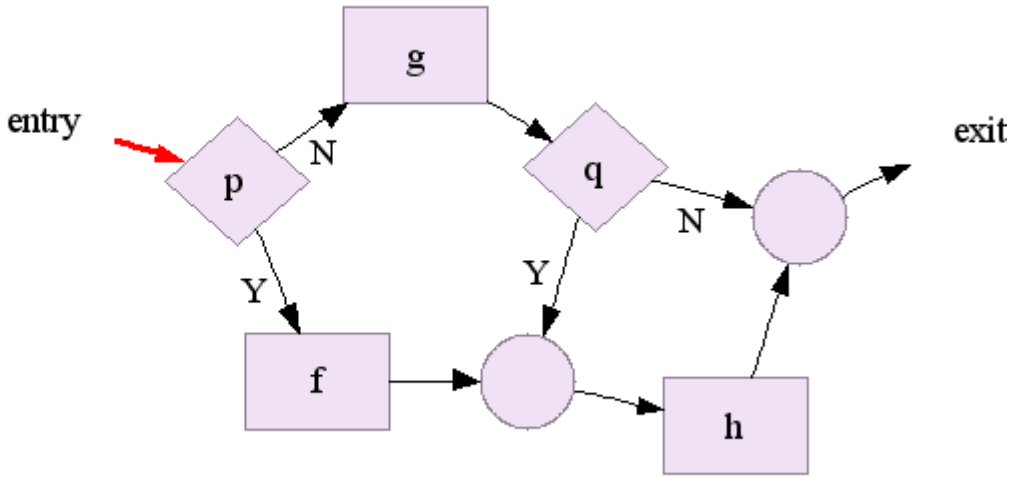
Perhaps of all the creations of man *language* is the most astonishing.

-- *Giles Lytton Strachey*

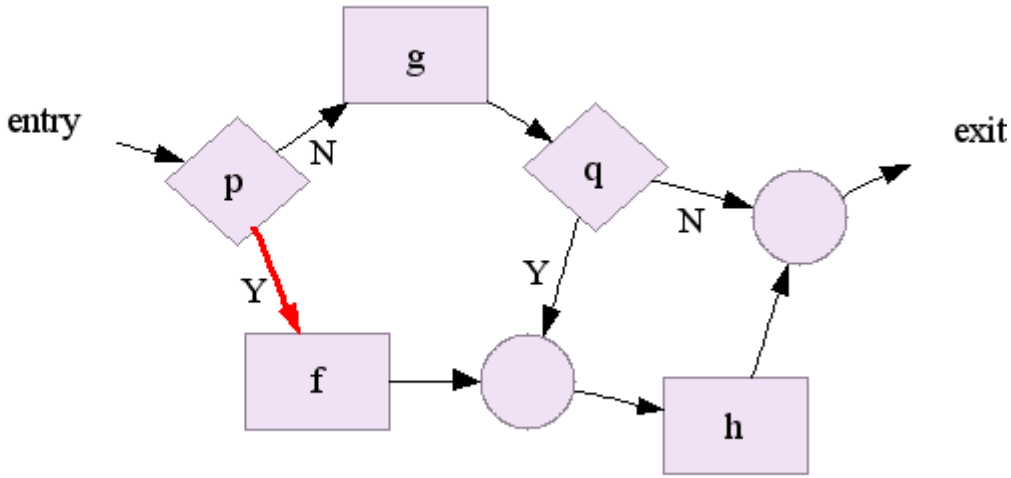
Why yet another language?

FAST has a
user language
for entering flowcharts

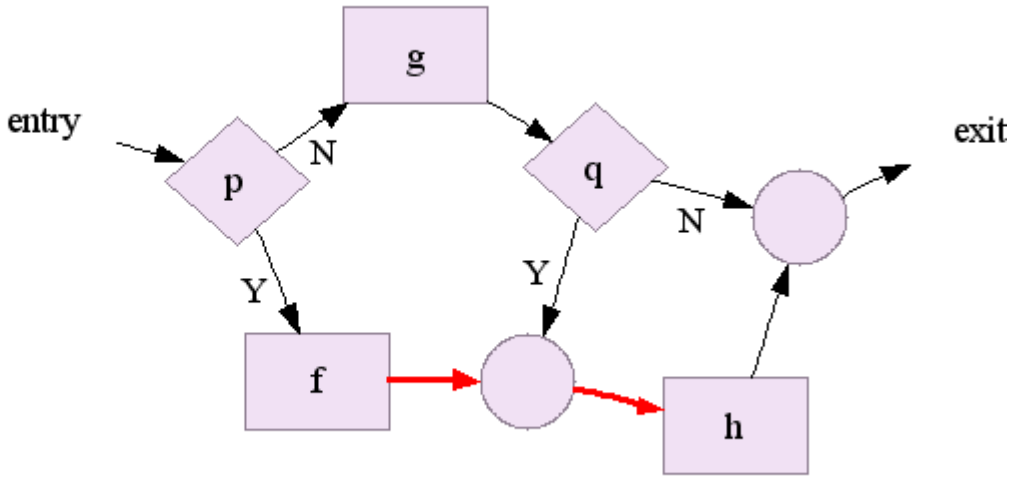
```
entry => <p>

<p> => [f]

[f] => [h]

[h] => exit

<p> => [g]

[g] => <q>

<q> => [h]

<q> => exit
```



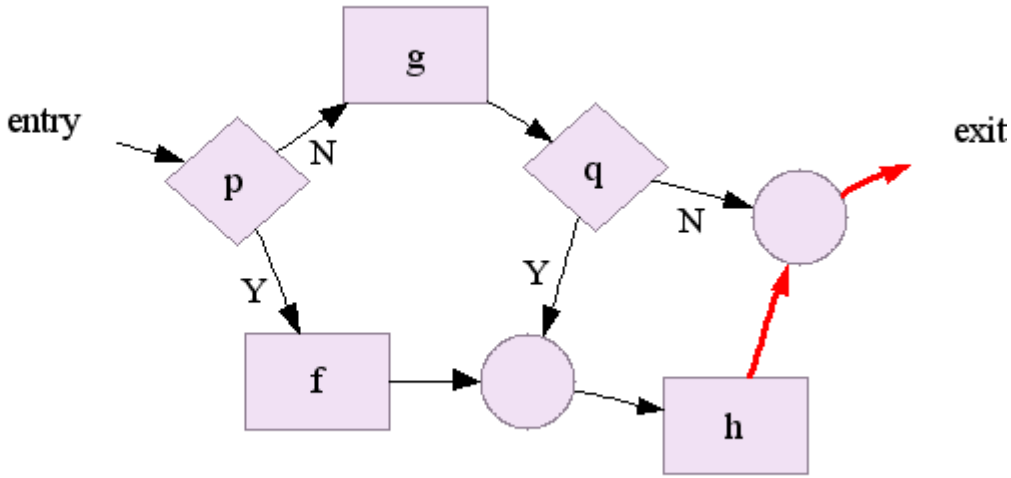
```
entry => <p>
<p> => [f]
[f] => [h]
[h] => exit
<p> => [g]
[g] => <q>
<q> => [h]
<q> => exit
```



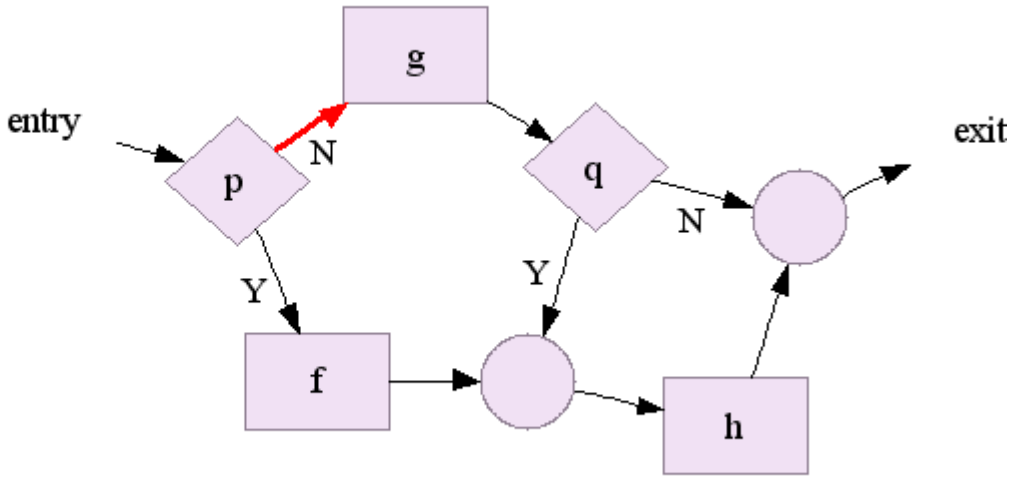
```
entry => <p>
<p> => [f]
[f] => [h]
[h] => exit
<p> => [g]
[g] => <q>
<q> => [h]
<q> => exit
```



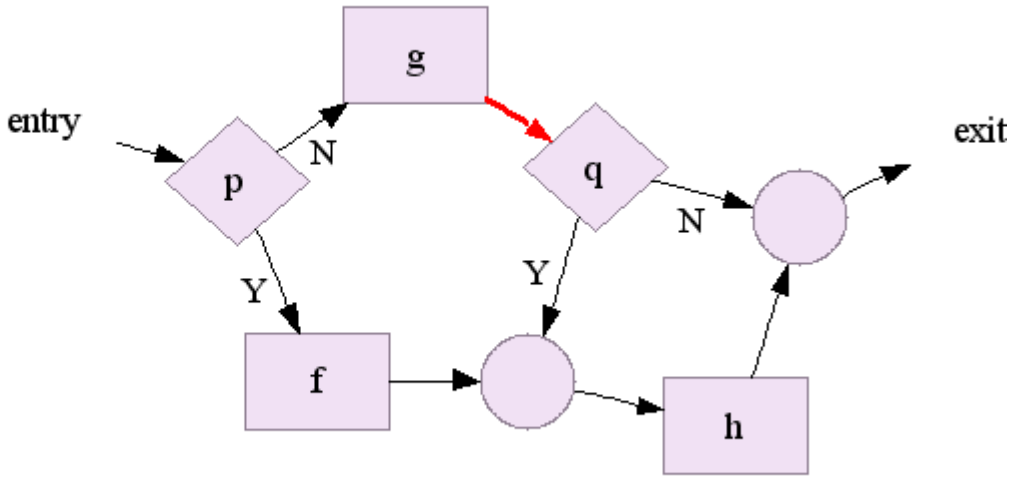
```
entry => <p>
<p> => [f]
[f] => [h]
[h] => exit
<p> => [g]
[g] => <q>
<q> => [h]
<q> => exit
```



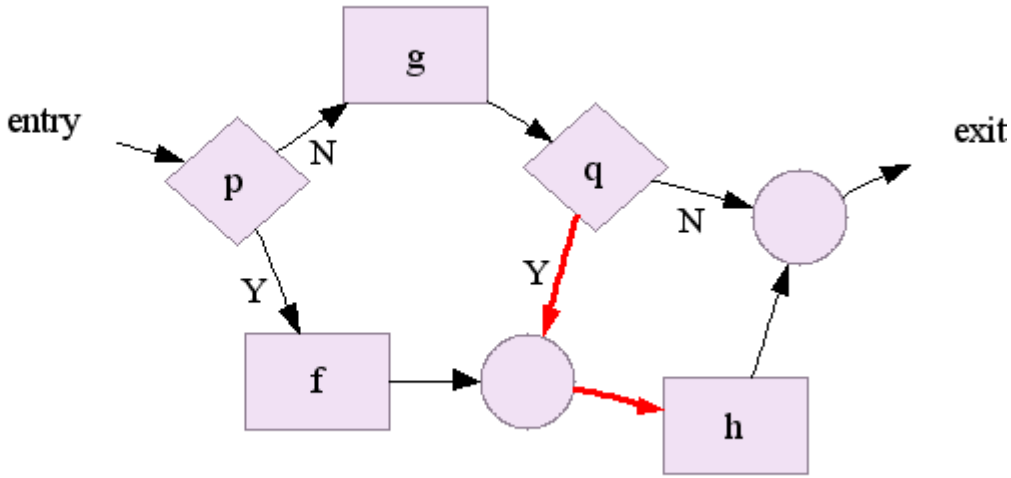
```
entry => <p>
<p> => [f]
[f] => [h]
[h] => exit
<p> => [g]
[g] => <q>
<q> => [h]
<q> => exit
```



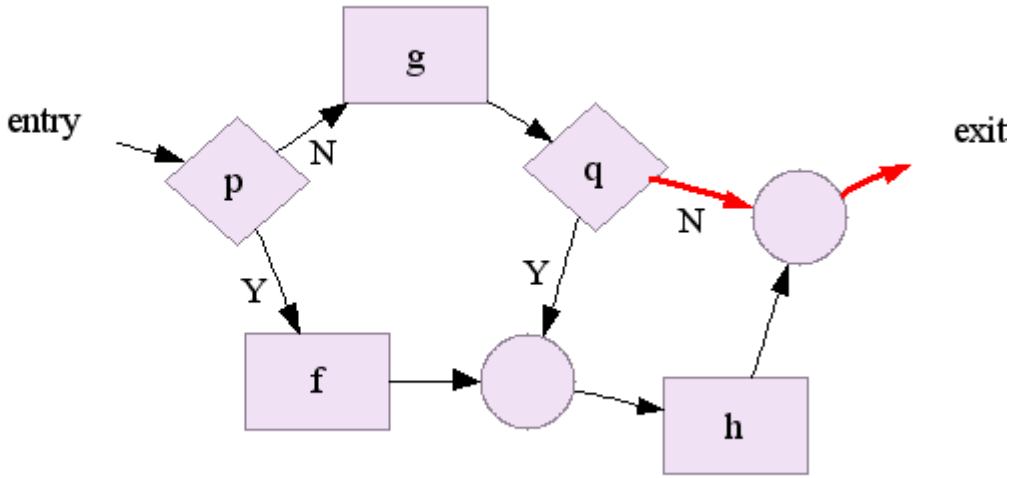
```
entry => <p>
<p> => [f]
[f] => [h]
[h] => exit
<p> => [g]
[g] => <q>
<q> => [h]
<q> => exit
```



```
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 !

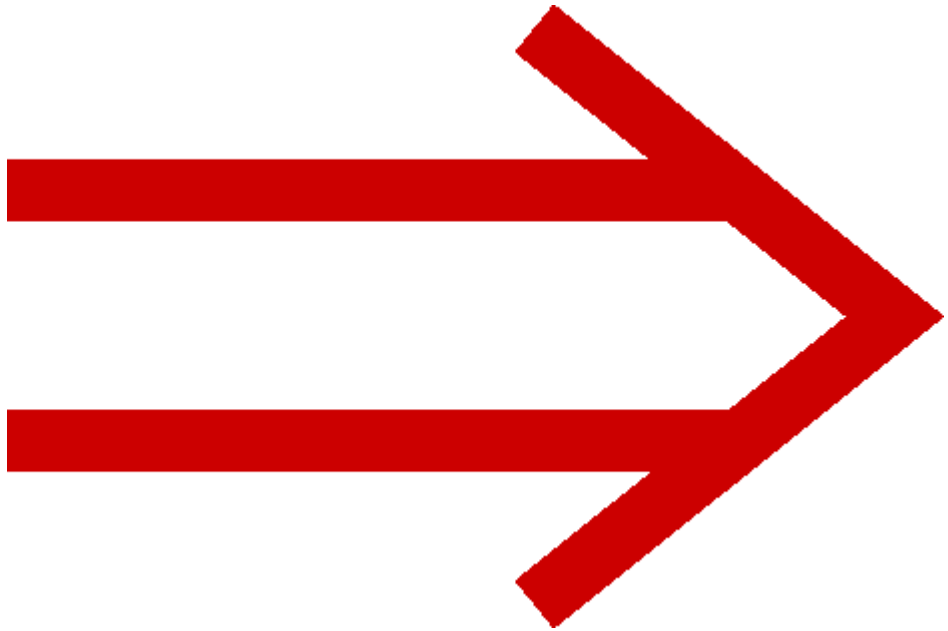
```
 1 sub parse {
 2     my( $self, $infile ) = @_
 3     open $in, $infile or die $!;
 4     my( %edge_from, %edge_to );
 5     while (<$in>) {
 6         if (/^\s* (.*\S) \s* => \s* (.*\S) \s*$/x) {
 7             my( $from, $to ) = ( $1, $2 );
 8             $edge_from{$to}   ||= [];
 9             $edge_from{$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*''

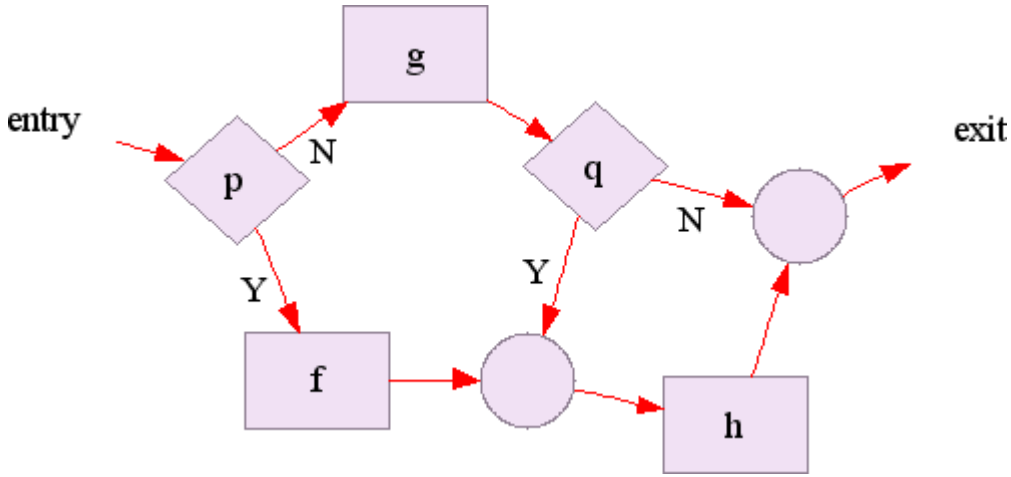
```
entry => <p>
<p>  =>  [f]
 [f] =>  [h]
 [h] =>  exit
<p>  =>  [g]
 [g] =>  <q>
<q>  =>  [h]
<q>  =>  exit
```

C:> fast *bar*



6 output files !

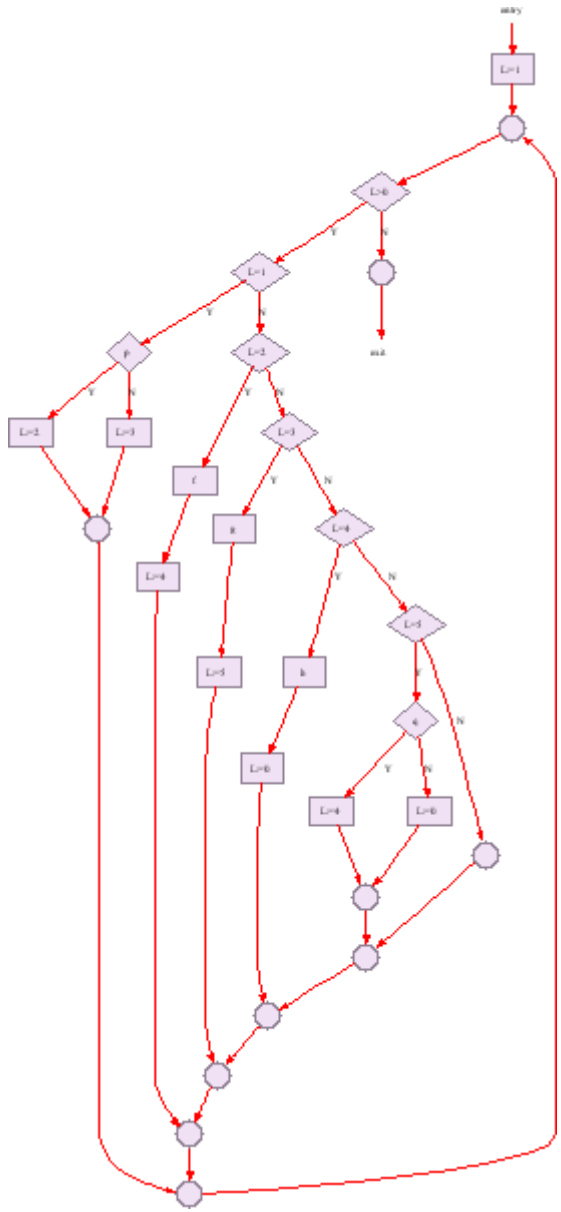
bar ⇒ bar.png


entry
p
g
N
Y
f
q
Y
N
exit
h

bar ⇒ bar.asm

```
        test  p
        jno   L1
        do    f
L2:
        do    h
L3:
        exit
L1:
        do    g
        test  q
        jno   L3
        jmp   L2
```

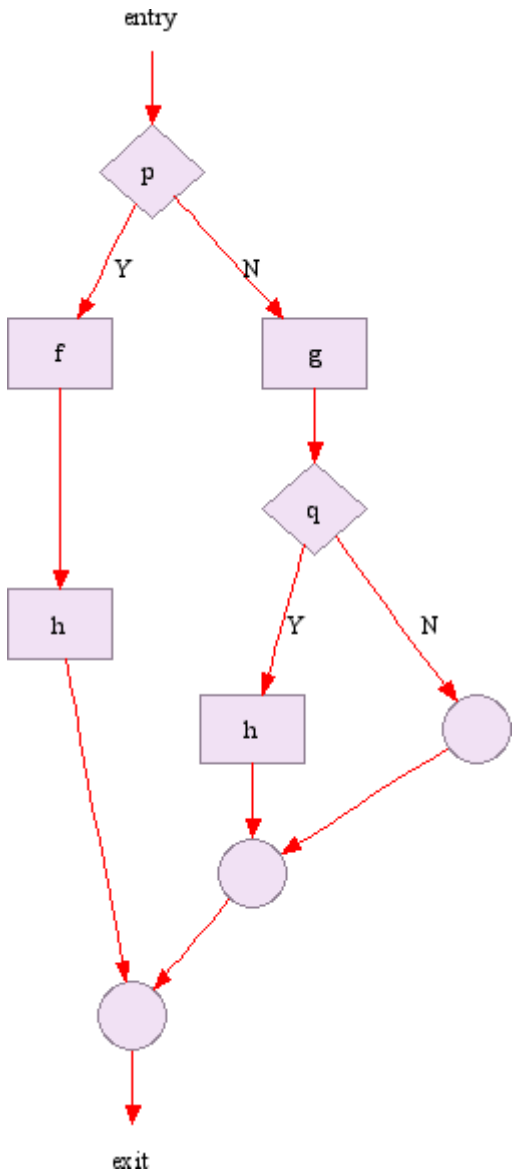

bar ⇒ bar.unopt.png


entry
L=1
L=6
Y
N
L=1
Y
N
exit
P
L=2
Y
N
L=3
L=2
Y
N
L=3
C
Y
N
L=4
B
L=4
Y
N
L=5
L=5
k
Y
N
L=6
L=4
L=6

bar ⇒ 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 ⇒ bar.opt.png


entry
p
Y
N
f
g
q
h
Y
N
h
exit

bar ⇒ 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

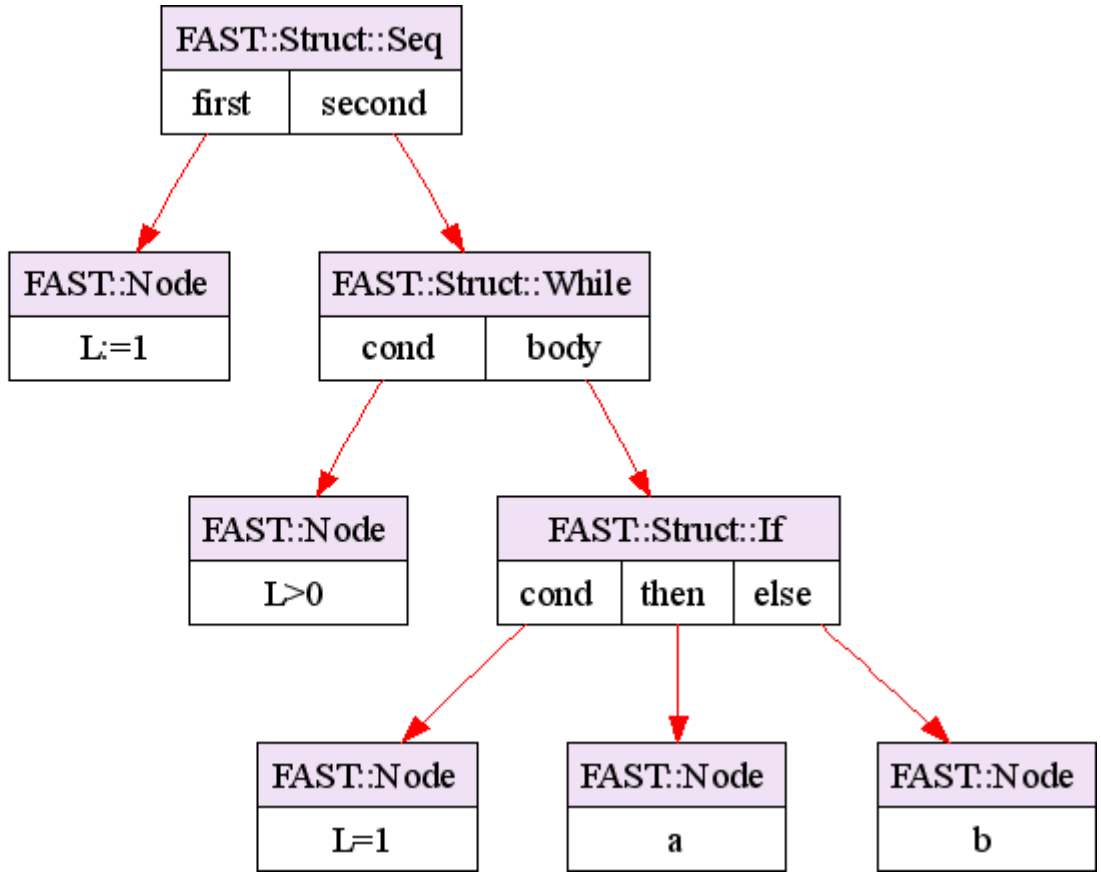
♡ Composite

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

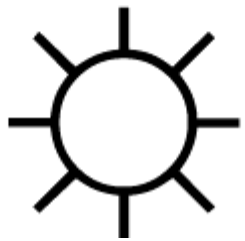
Yeah, that's a story about *trees*...

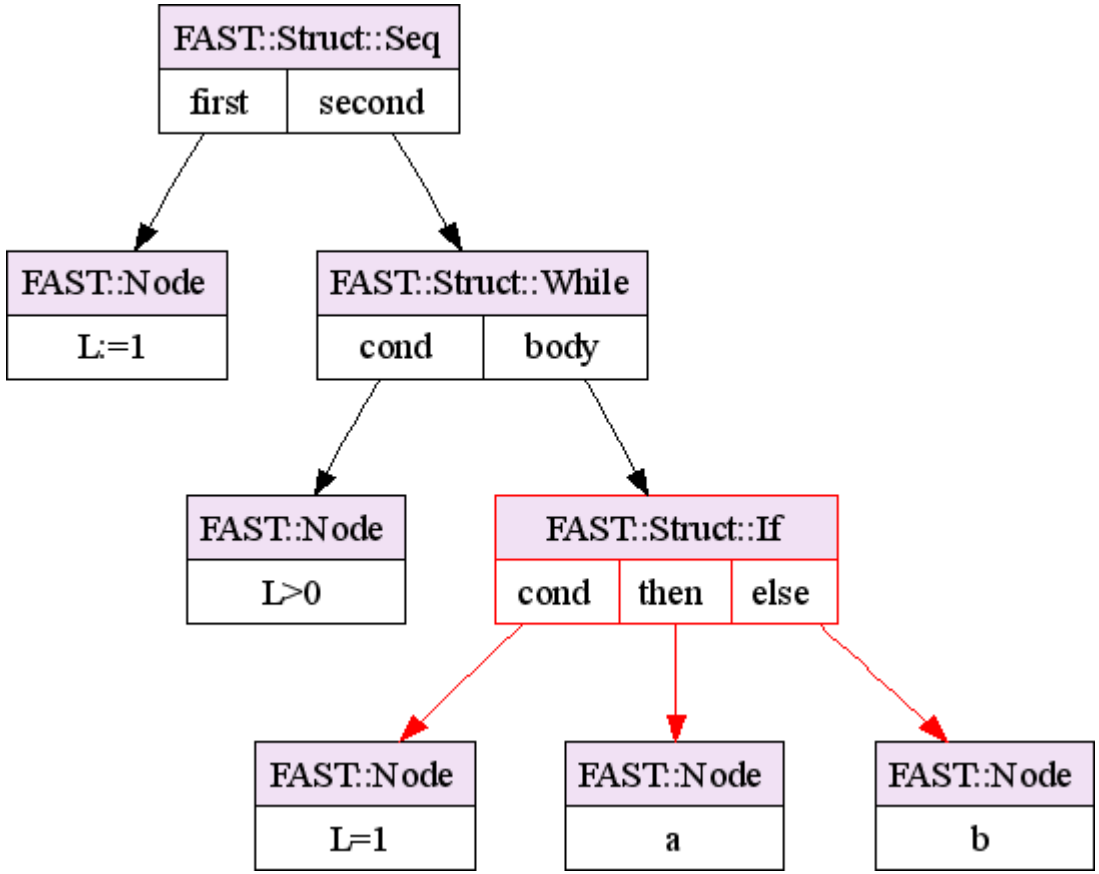
Every structural flowchart can be represented by a *tree*

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

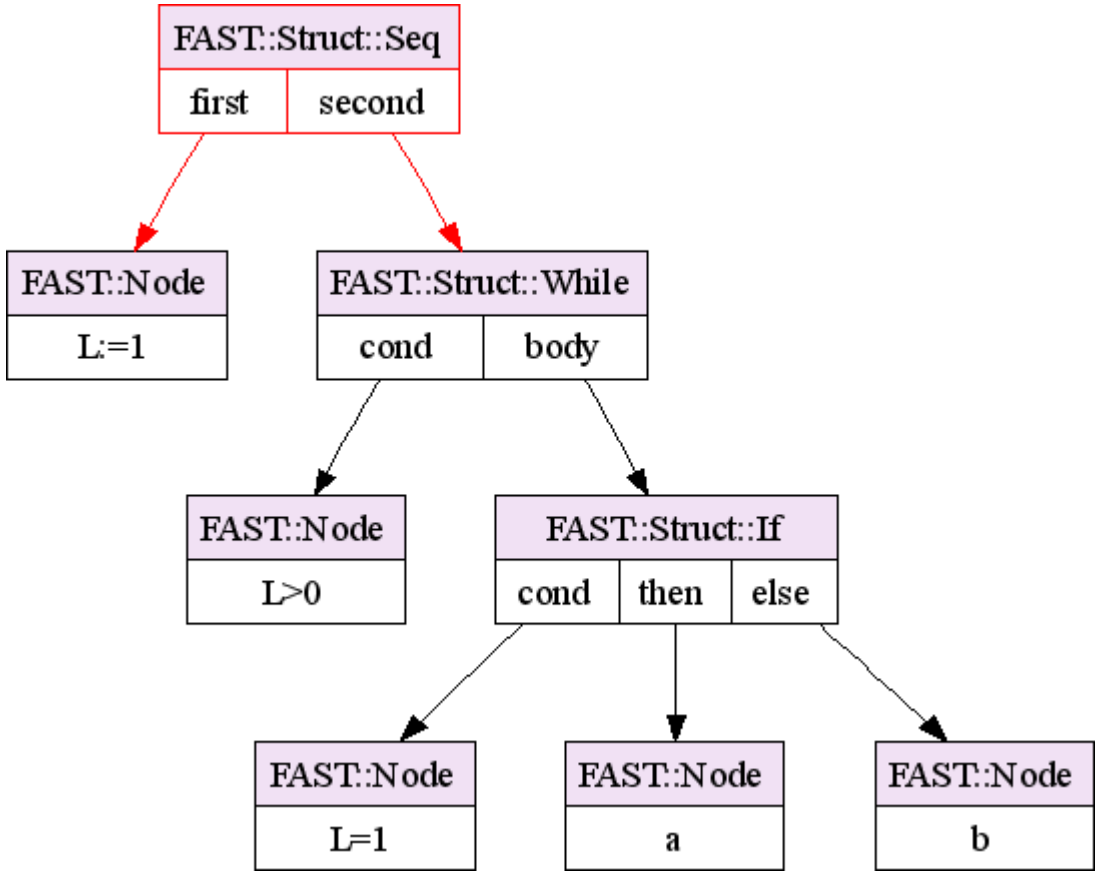


Every node is also a tree.




FAST::Struct::Seq
first
second
FAST::Node
L:=1
FAST::Struct::While
cond
body
FAST::Node
L>0
FAST::Struct::If
cond
then
else
FAST::Node
L=1
FAST::Node
a
FAST::Node
b

Every tree is represented by its root node.

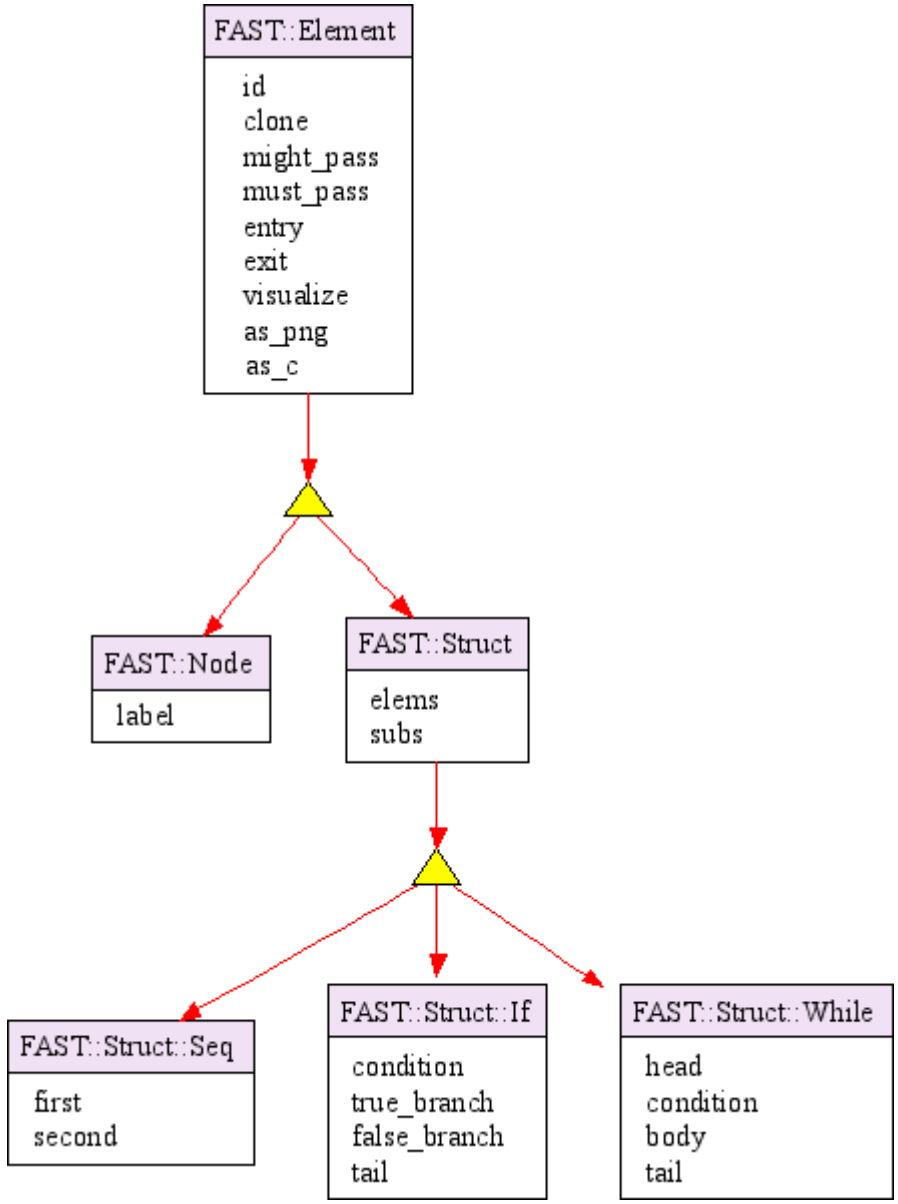

FAST::Struct::Seq
first
second
FAST::Node
L:=1
FAST::Struct::While
cond
body
FAST::Node
L>0
FAST::Struct::If
cond
then
else
FAST::Node
L=1
FAST::Node
a
FAST::Node
b

Hence…

Trees and nodes
share the *same* interface.

There is *no*
Tree class any more

As evidenced in the *UML* thingy...

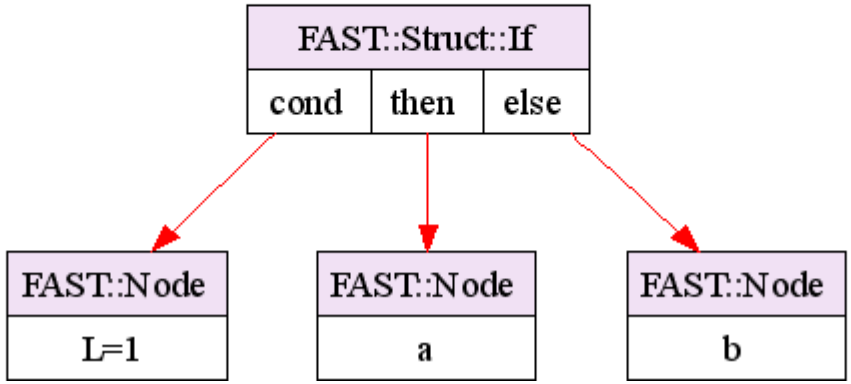

FAST::Element
id
clone
might_pass
must_pass
entry
exit
visualize
as_png
as_c
FAST::Node
label
FAST::Struct
elems
subs
FAST::Struct::Seq
first
second
FAST::Struct::If
condition
true_branch
false_branch
tail
FAST::Struct::While
head
condition
body
tail

Composite Pattern

✓ Building the tree is easy

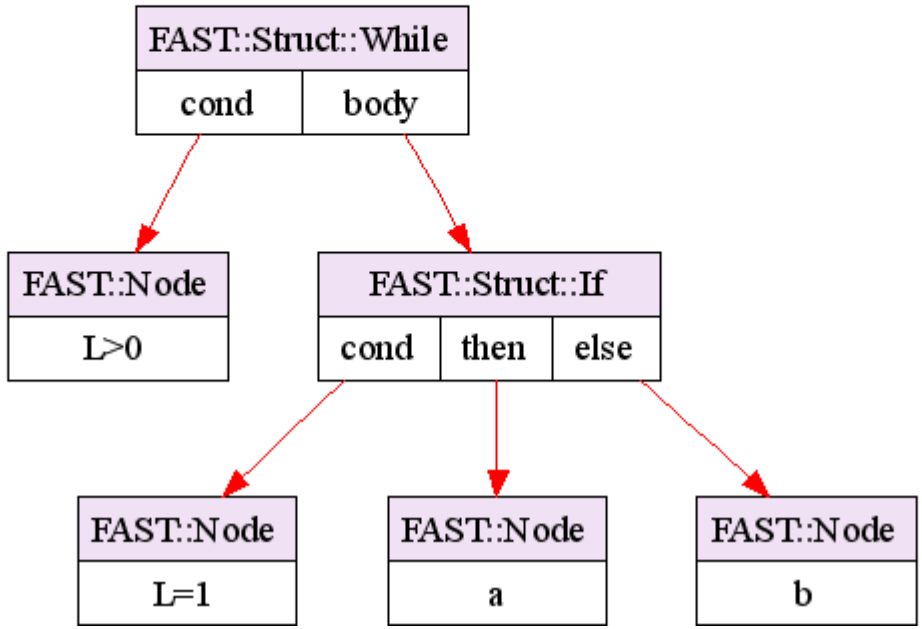
```
$if = FAST::Struct::If->new('<L=1>', '[a]', '[b]');
```

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



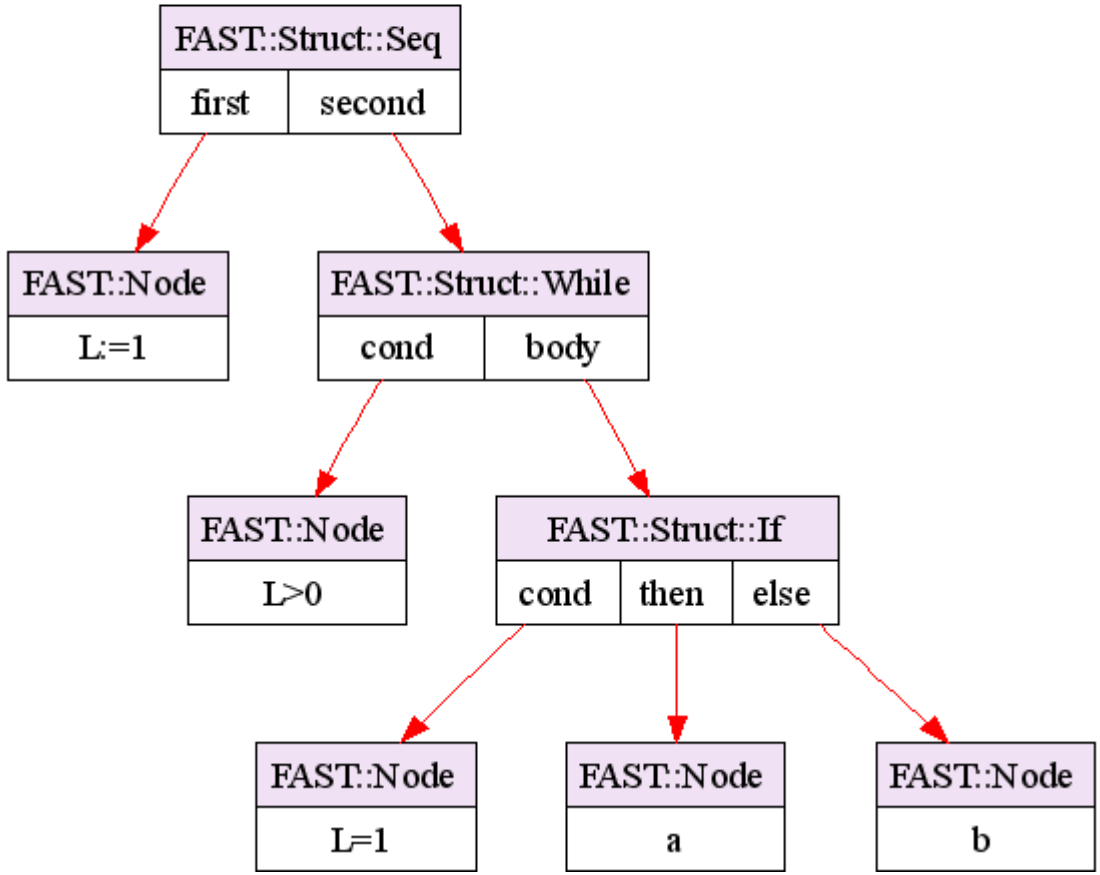
```
$if = FAST::Struct::If->new('<L=1>', '[a]', '[b]');
$while = FAST::Struct::While->new('<L>0>', $if);
```

```
$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);
```

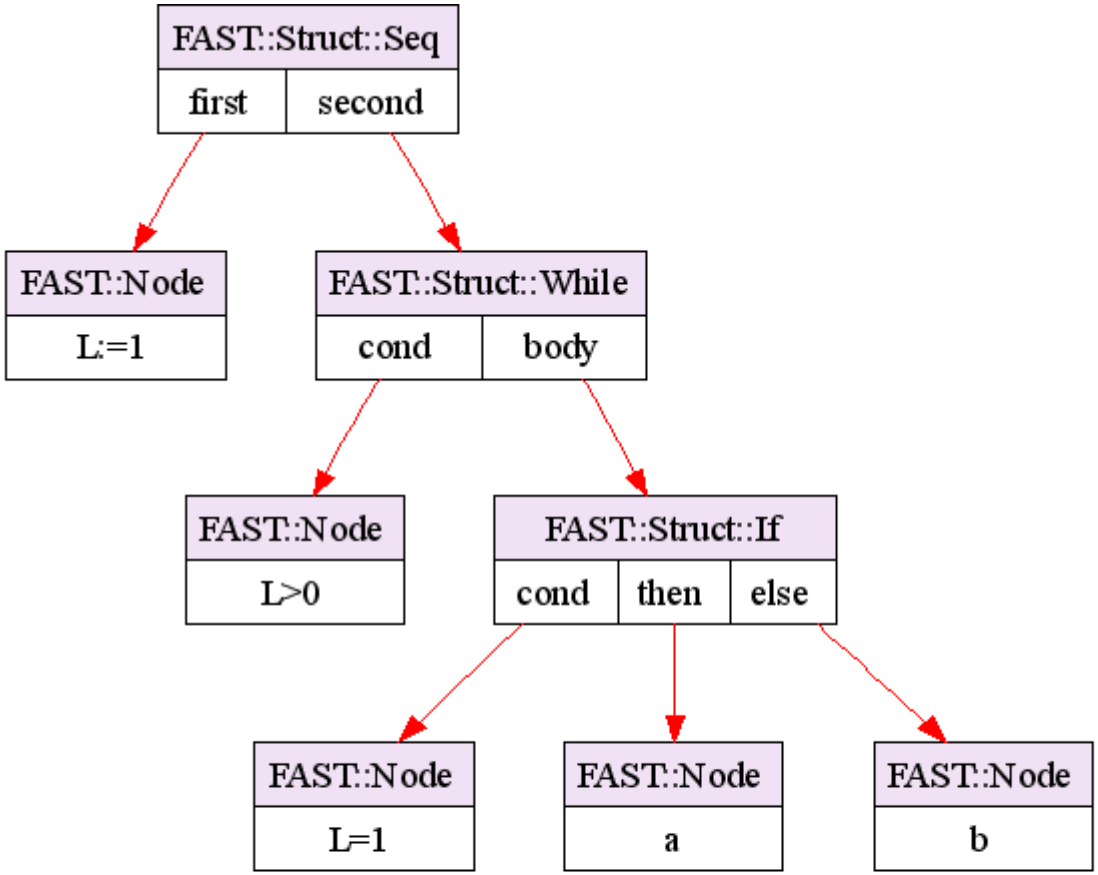


Composite Pattern

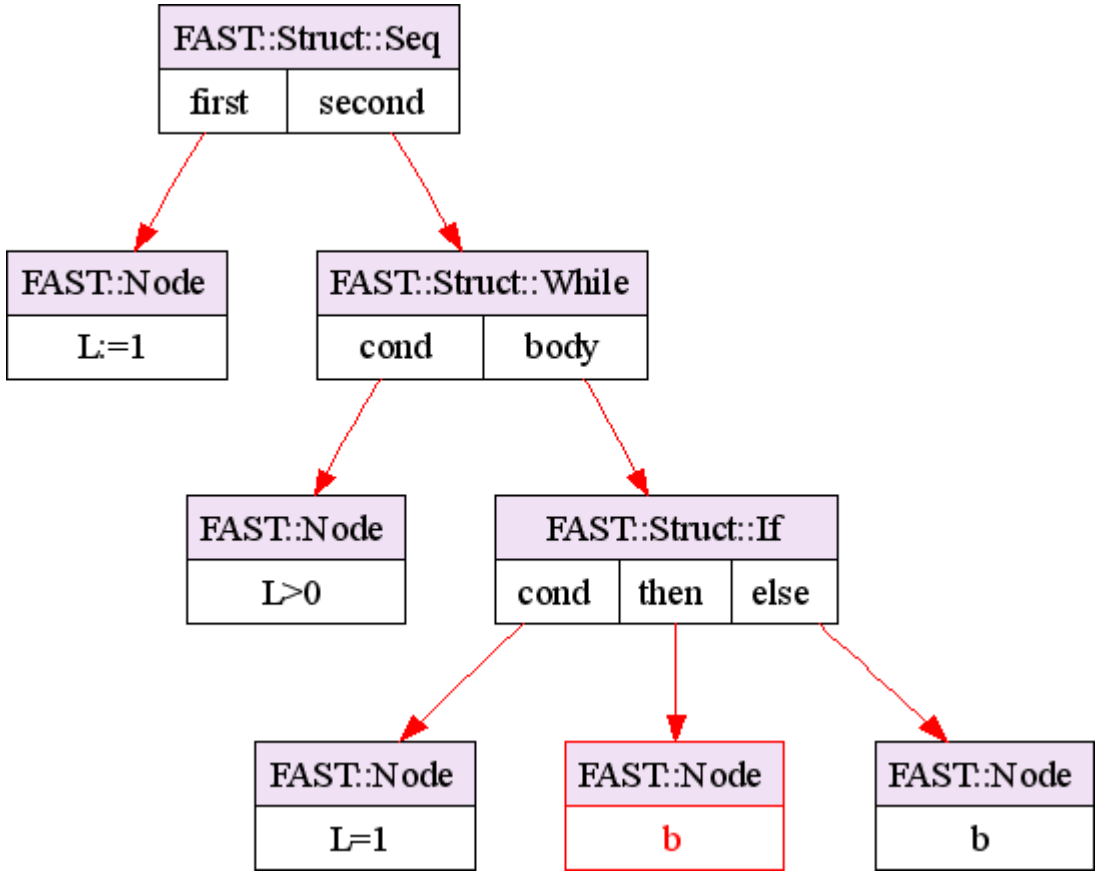
✓ Handling the tree is easy

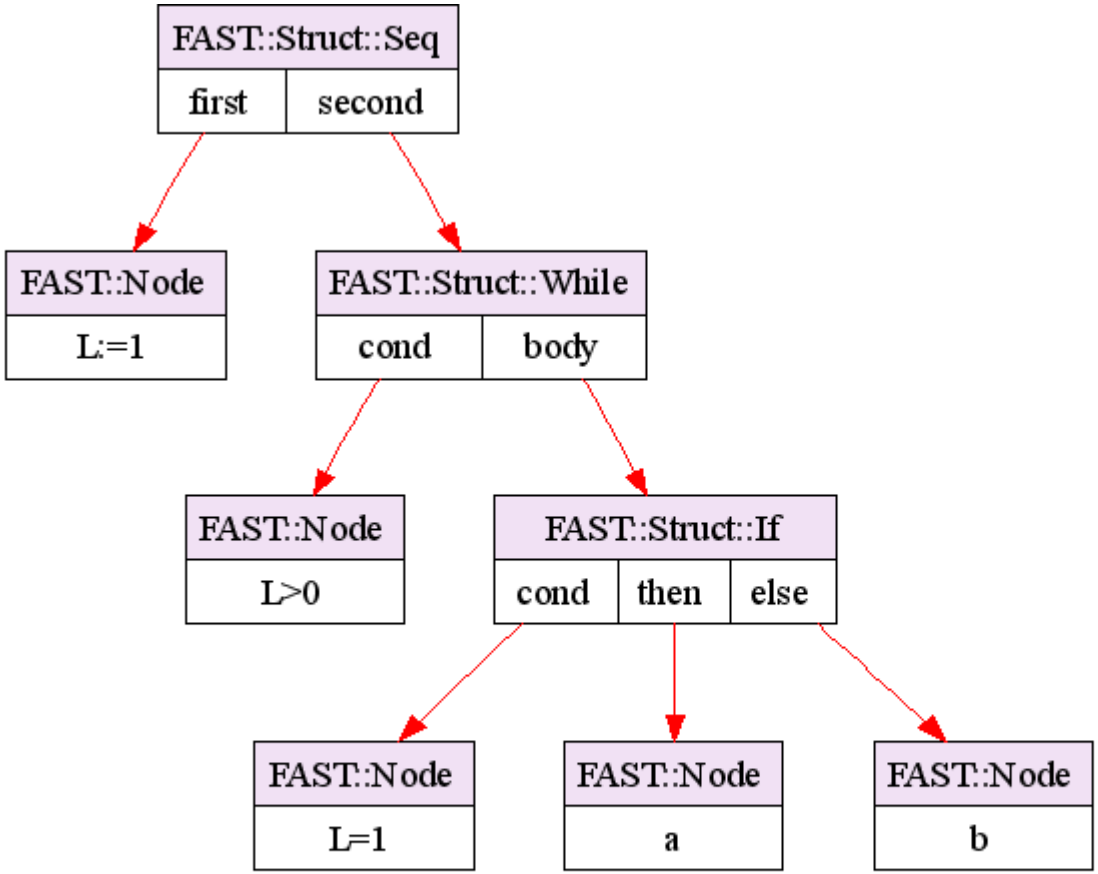
```
$tree1->subs($node_name, $tree2);
```

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

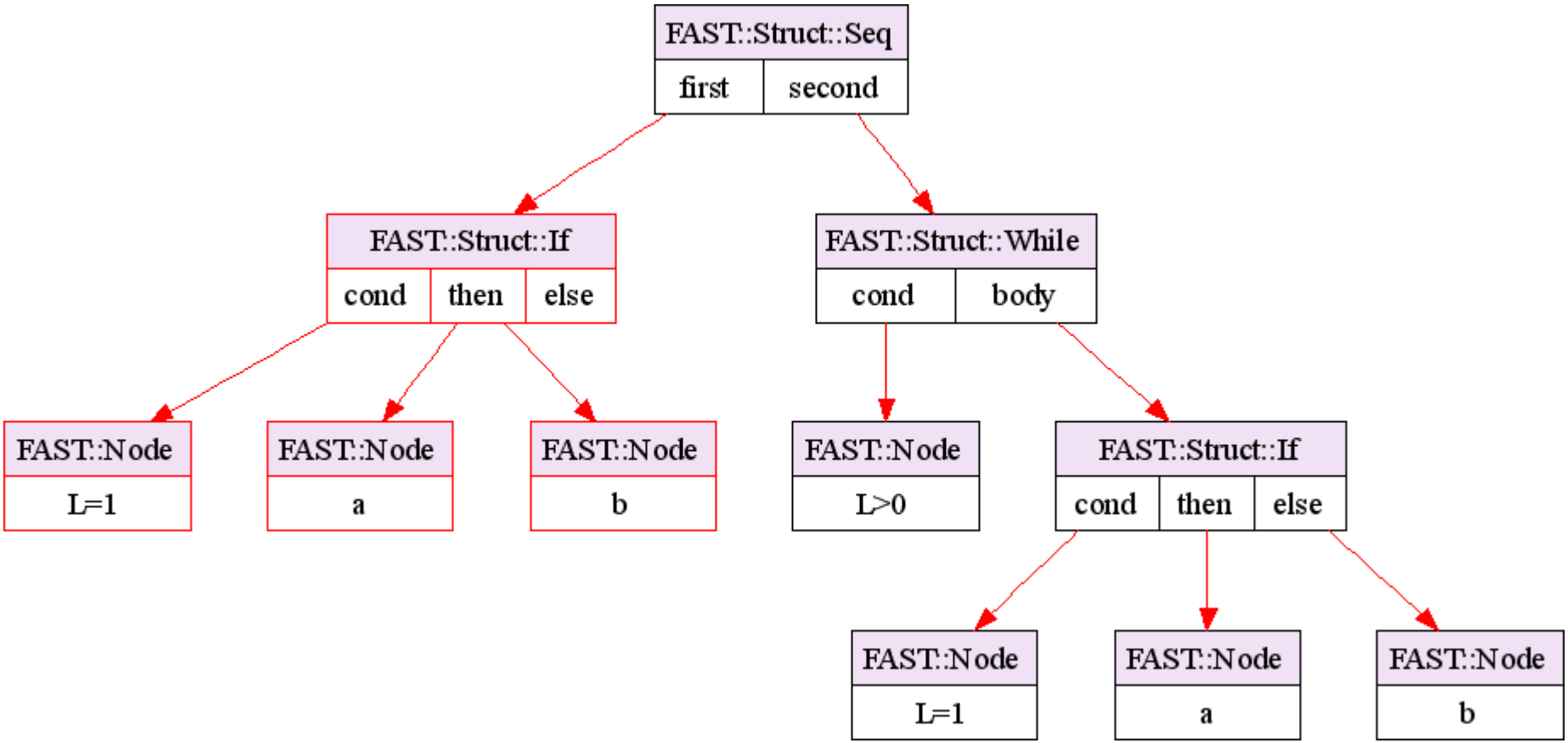

FAST::Struct::Seq
first
second
FAST::Node
L:=1
FAST::Struct::While
cond
body
FAST::Node
L>0
FAST::Struct::If
cond
then
else
FAST::Node
L=1
FAST::Node
a
FAST::Node
b

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


FAST::Struct::Seq
first
second
FAST::Node
L:=1
FAST::Struct::While
cond
body
FAST::Node
L>0
FAST::Struct::If
cond
then
else
FAST::Node
L=1
FAST::Node
b
FAST::Node
b




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


FAST::Struct::Seq
first
second
FAST::Struct::If
cond
then
else
FAST::Struct::While
cond
body
FAST::Node
L=1
FAST::Node
a
FAST::Node
b
FAST::Node
L>0
FAST::Struct::If
cond
then
else
FAST::Node
L=1
FAST::Node
a
FAST::Node
b

The subs method is *also*

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

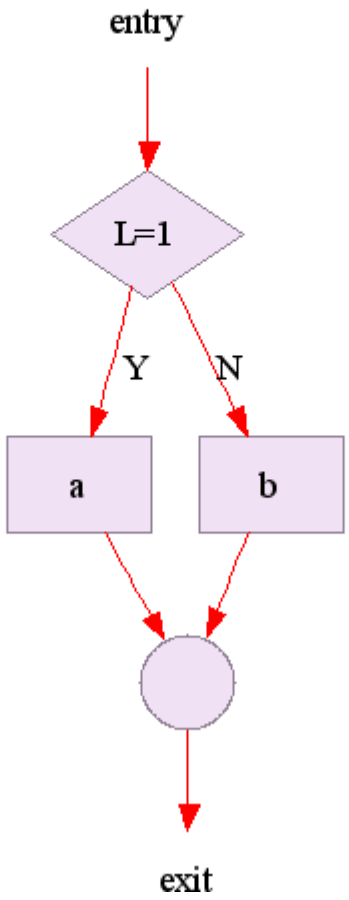
```
 1 package FAST::Struct;
 2
 3 # Substitue a copy of $dest for every node named $label
 4 # in the current FAST::Struct subtree:
 5 sub subs {
 6     my ($self, $label, $dest) = @_;
 7     $dest = $self->_node($dest);
 8     my $relems = $self->elems;
 9     my $done;
10     for my $e (@$relems) {
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

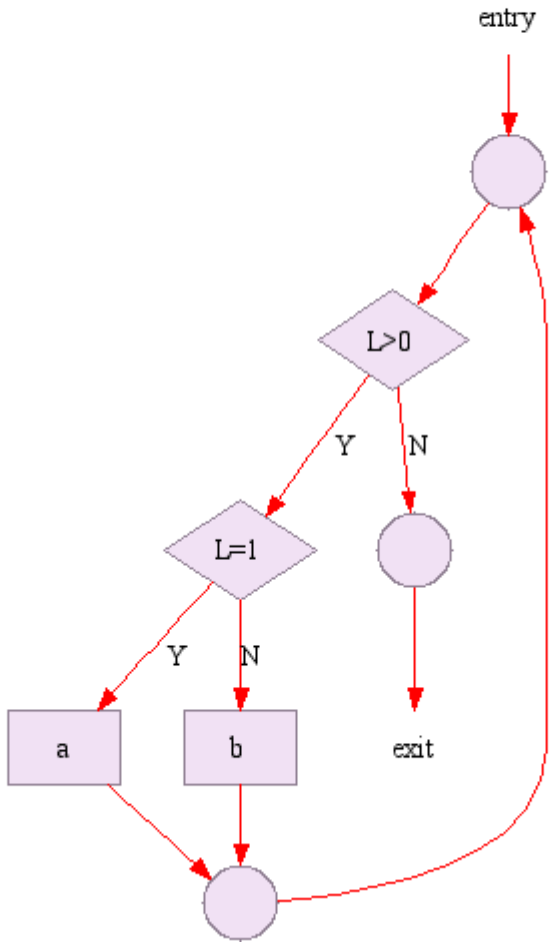
```
$if->as_png('if.png');
```

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



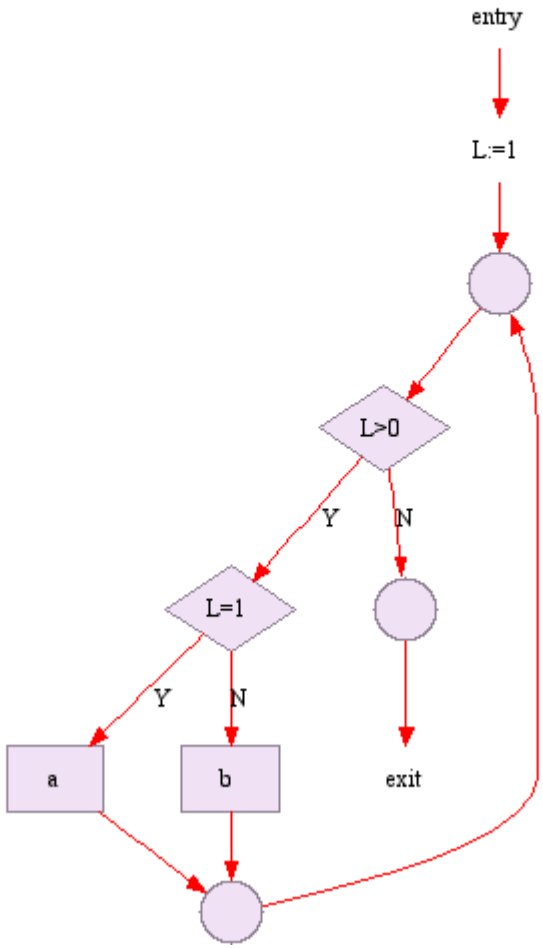
```
$while->as_png('while.png');
```

```
$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://perlcabal.org/agent/slides/patterns/patterns.xul
http://perlcabal.org/agent/slides/patterns/patterns.ppt
http://perlcabal.org/agent/slides/patterns/patterns.pdf

These slides are *powered* by

These slides are *powered* by Sporx and...

Takahashi++

Most of the images used here were *dawn* by

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

Thank you!