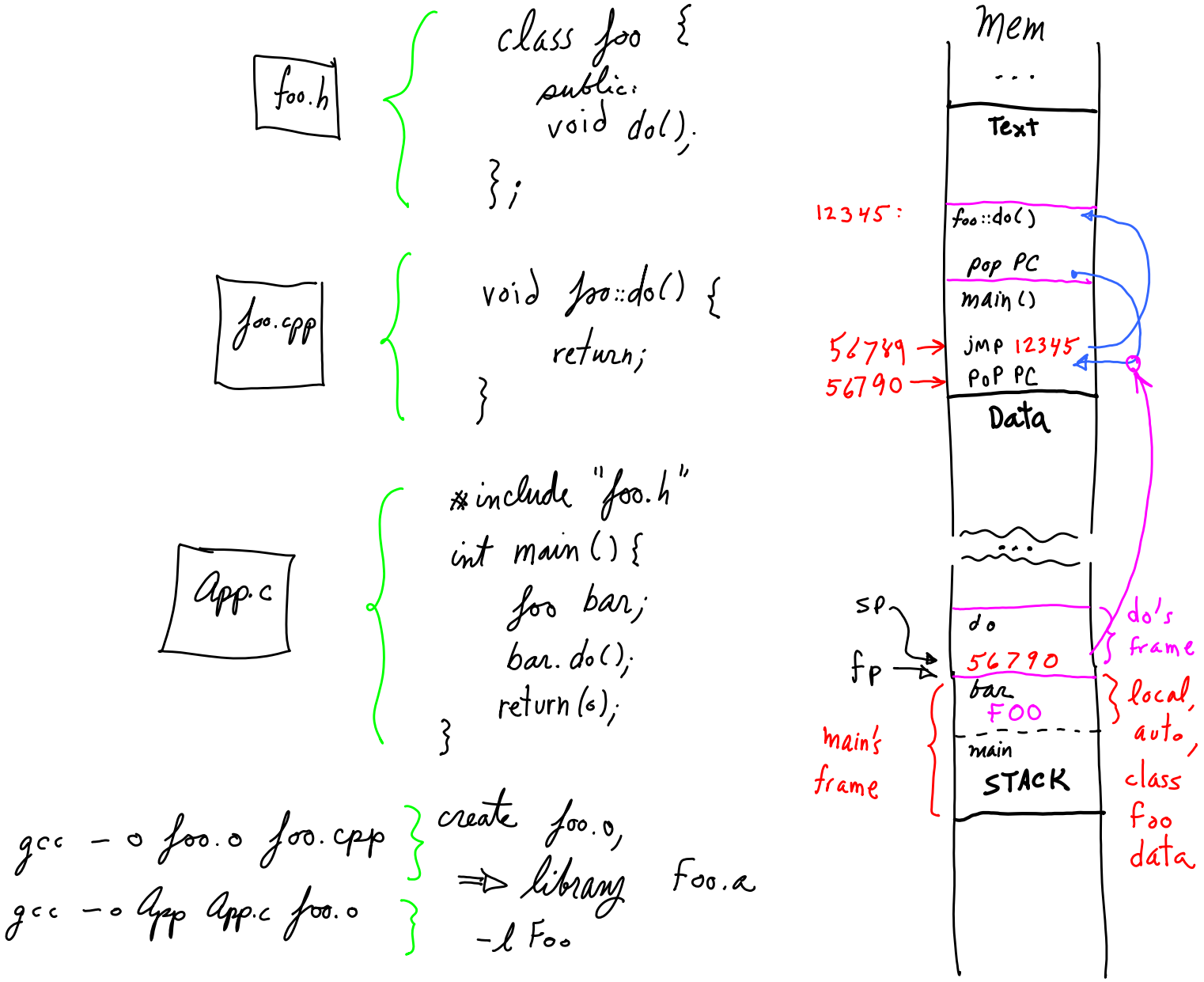
Syntax corner: cd template/src/syntaxCorner; svn up; svn log -v

```
------------------------------------------------------------------------
r413 | 250-374-developer | 2011-07-26 14:24:35 -0400 (Tue, 26 Jul 2011) | 3 lines
Changed paths:
   M /template/src/syntaxCorner/const.c

rks-- syntax: destructors, method overload.
Note the failure to call destructor for ptr.

------------------------------------------------------------------------
r411 | 250-374-developer | 2011-07-26 13:53:09 -0400 (Tue, 26 Jul 2011) | 2 lines
Changed paths:
   M /template/src/syntaxCorner/const.c

rks-- syntax: overloaded method.

------------------------------------------------------------------------
r410 | 250-374-developer | 2011-07-26 13:43:52 -0400 (Tue, 26 Jul 2011) | 2 lines
Changed paths:
   M /template/src/syntaxCorner/const.c

rks-- syntax const: a const object can't alter fields.

------------------------------------------------------------------------
r409 | 250-374-developer | 2011-07-26 13:38:48 -0400 (Tue, 26 Jul 2011) | 2 lines
Changed paths:
   M /template/src/syntaxCorner/const.c

rks-- syntax, const: f() const cannot alter object.

------------------------------------------------------------------------
r408 | 250-374-developer | 2011-07-26 13:36:51 -0400 (Tue, 26 Jul 2011) | 2 lines
Changed paths:
   M /template/src/syntaxCorner/const.c

rks-- syntax, const: usual object methods.

------------------------------------------------------------------------
r404 | 250-374-developer | 2011-07-26 12:52:21 -0400 (Tue, 26 Jul 2011) | 2 lines
Changed paths:
   A /template/src/syntaxCorner
   A /template/src/syntaxCorner/const.c

rks--Syntax for "const int".

------------------------------------------------------------------------
```

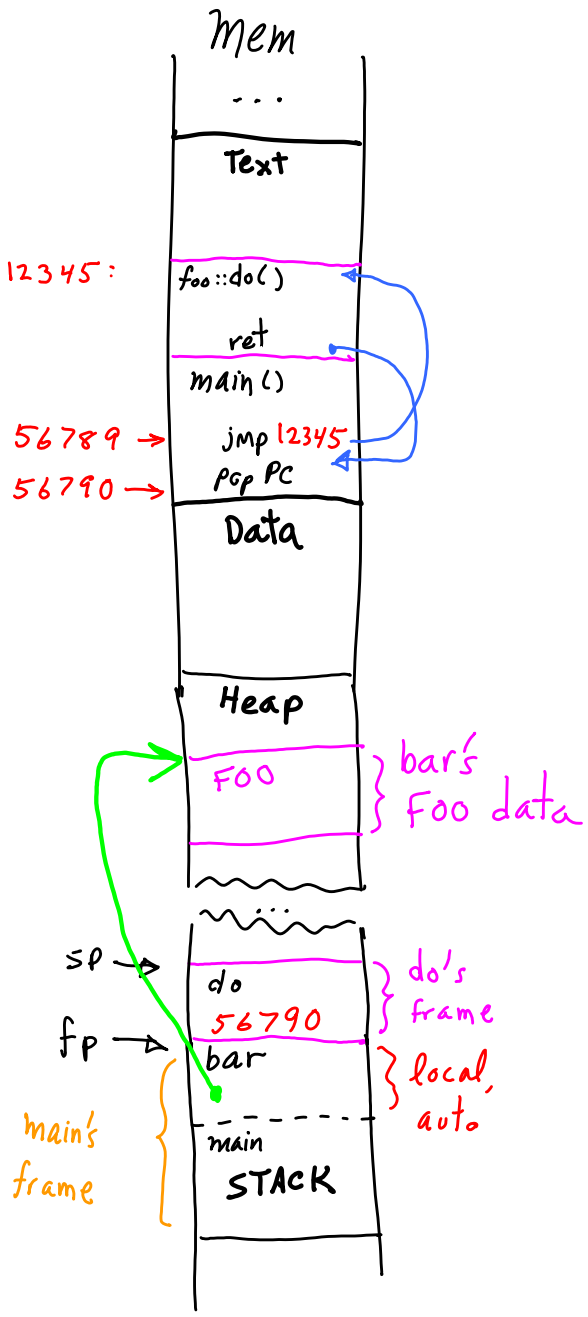


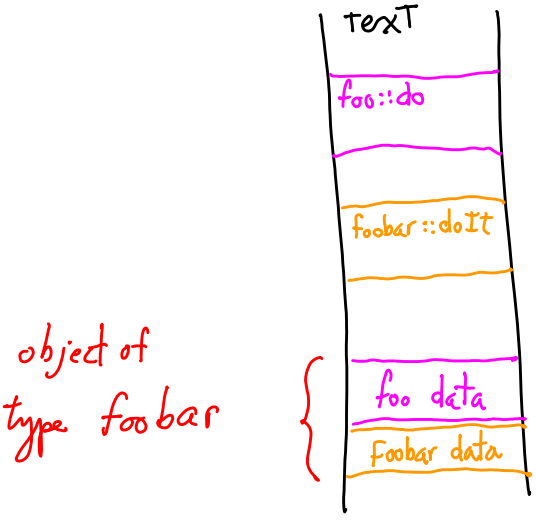
compile time: known address of function bar.do() ==> jmp 12345

run time: unknown return address is on top of stack ==> pop PC (= 12346)

```
#include "foo.h"
int main() {
    foo *bar = new foo;
    *bar.do();
    return (0);
}
```

```
class foobar : foo {
        public doIt();
}
```





Virtual functions polymorphism

```
class Thing {
        virtual void show() const;
                void hello();
};
```

```
class Bunny : Thing {
        void show() const;
        void hello();
};      int x;
```

```
void Bunny::show() const {
}
...
```

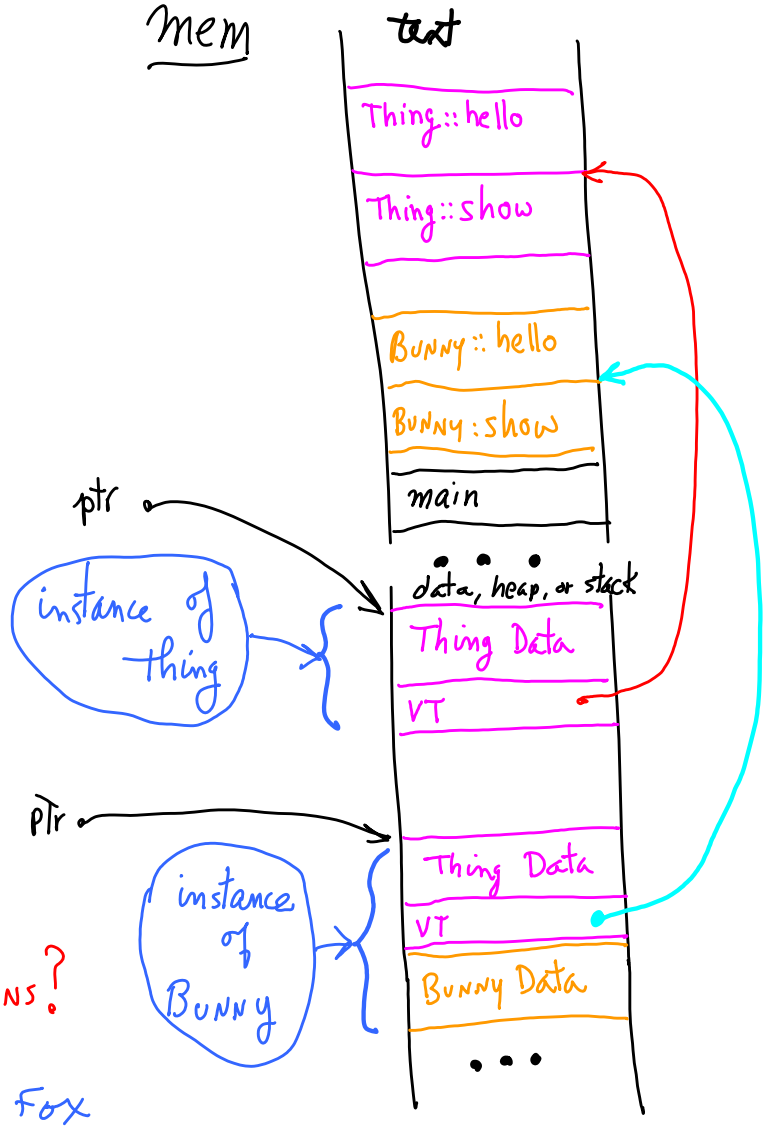
```
class Fox : Thing {
        void show() const;
        void hello();
};      int y;
```

```
void Fox::show() const {
}
...
```

```
Thing *ptr;
ptr = (Thing*) new Bunny;
list.insert(ptr);
ptr = (Thing*) new Fox;
list.insert(ptr);

ptr = new Thing;
ptr->show();
ptr->hello();

ptr = list.getNext();
ptr->show();
ptr->hello();

ptr = list.getNext();
...
```

Q. Which hello() runs?

} access a Fox



Tree traversals

given: An expression tree

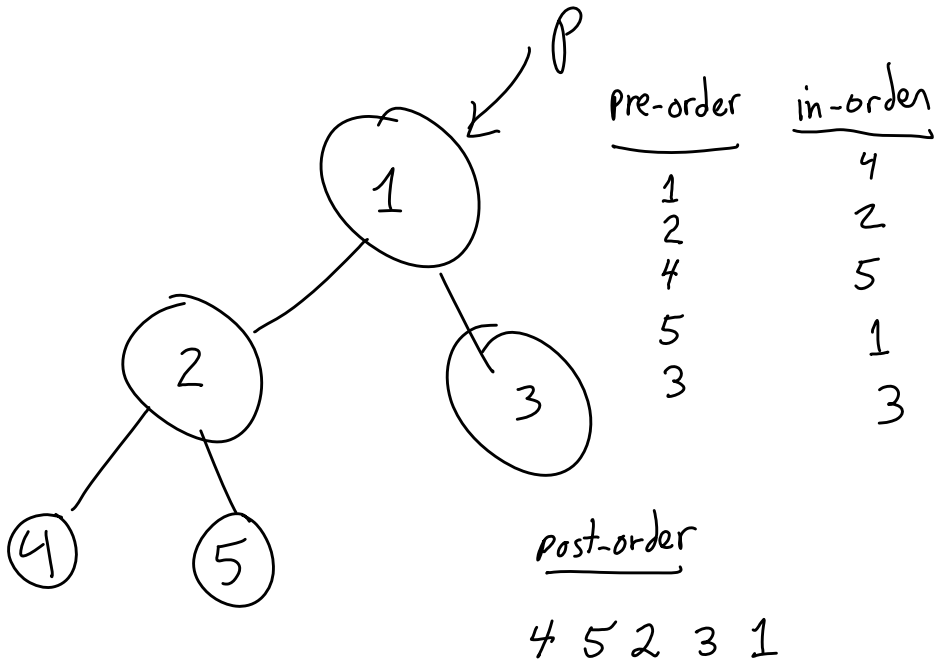
usual algebra expression (inorder):

```
Walk( p->left );
do( p );
Walk( p->right );
```

function notation (preorder):

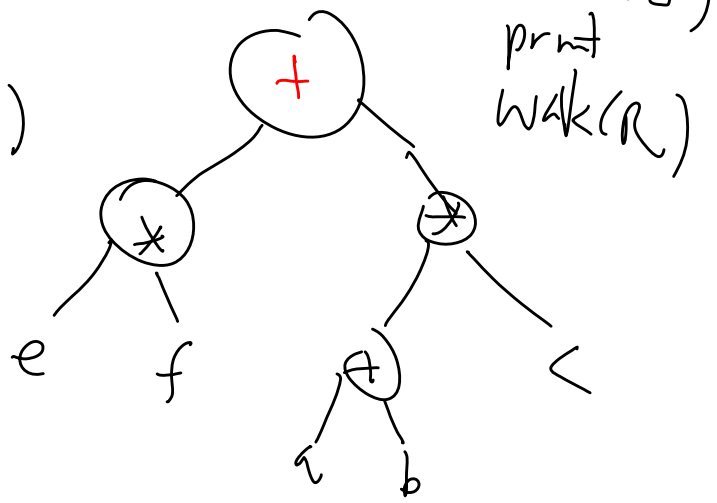
```
do( p );
Walk( p->left );
Walk( p->right );
```

reverse polish notation (postorder):



in-order

((e * f)) + ((a+ b) * c)



Walk(L)
print
Walk(R)

pre-order

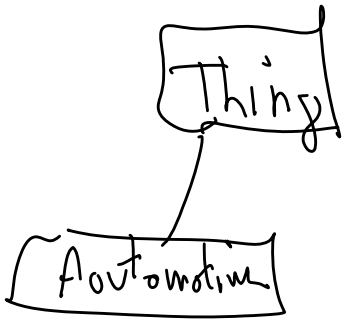
+(*(e, f) , *(+(a, b), c))

polish

ef* ab+ c*+

} use LIFO, pop args, pop func, push result

```
class Thing {
    virtual void draw();
    int x;
    int y;
};
```



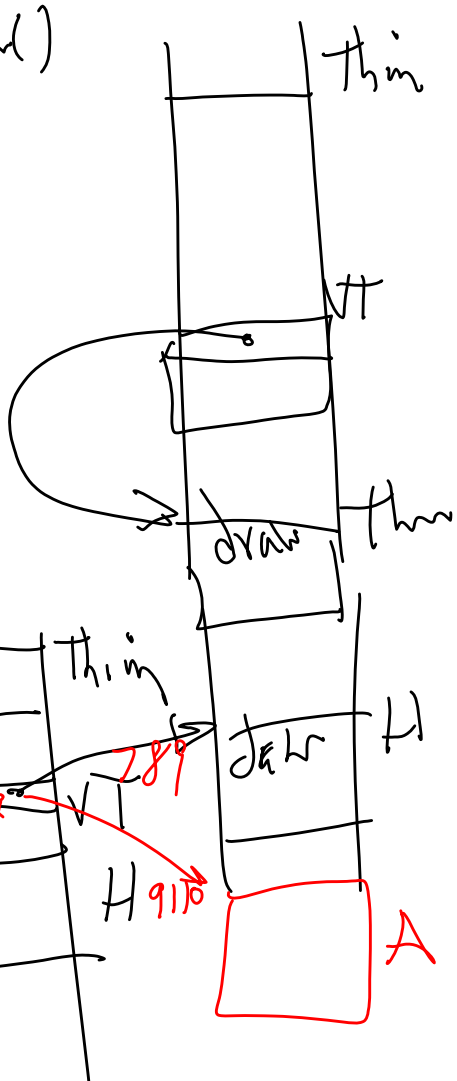

```
class Automotive:Thing{
    void move(?)
    void draw()
};
```

```
A b;
b.x=1
b.y=2
b.draw()
```

```
class H:Thing
    draw()
```

```
H c;
c.draw()
```

```
Thing * ptr = queue.get()
12340  ptr->draw()
((H*)ptr)->draw()
```



ptr->x

12340+0

ptr->y

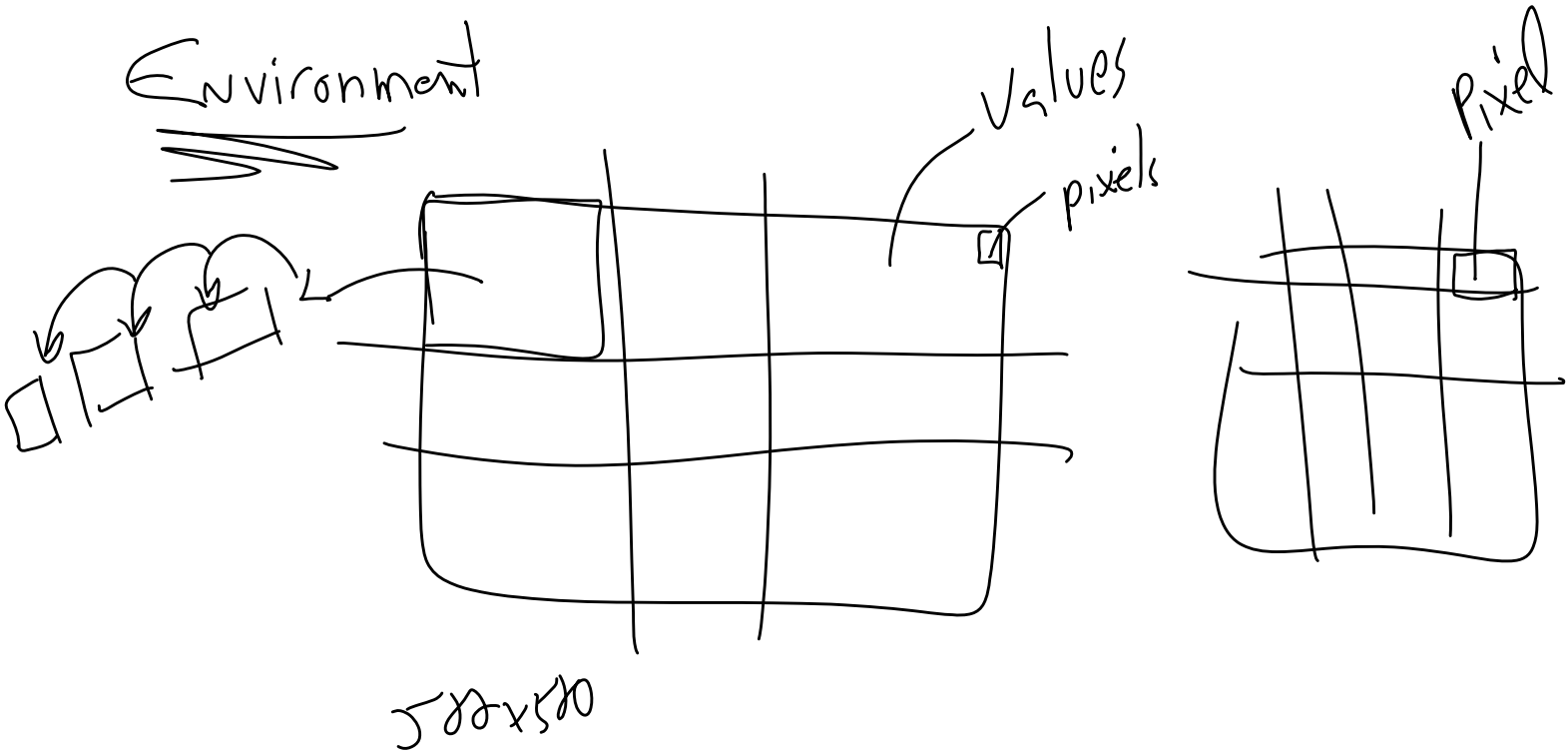
12340+8

ptr->draw 12340 : y

12356 : 289

12240+16+0

PC ← MEM[123456]



char A[] = "abc";

const char * A = &"abc"

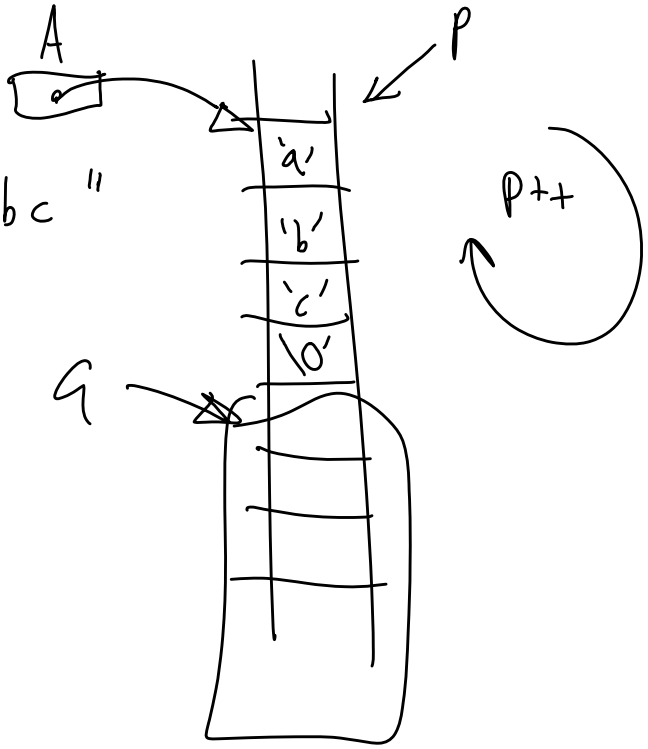
Char a[10]

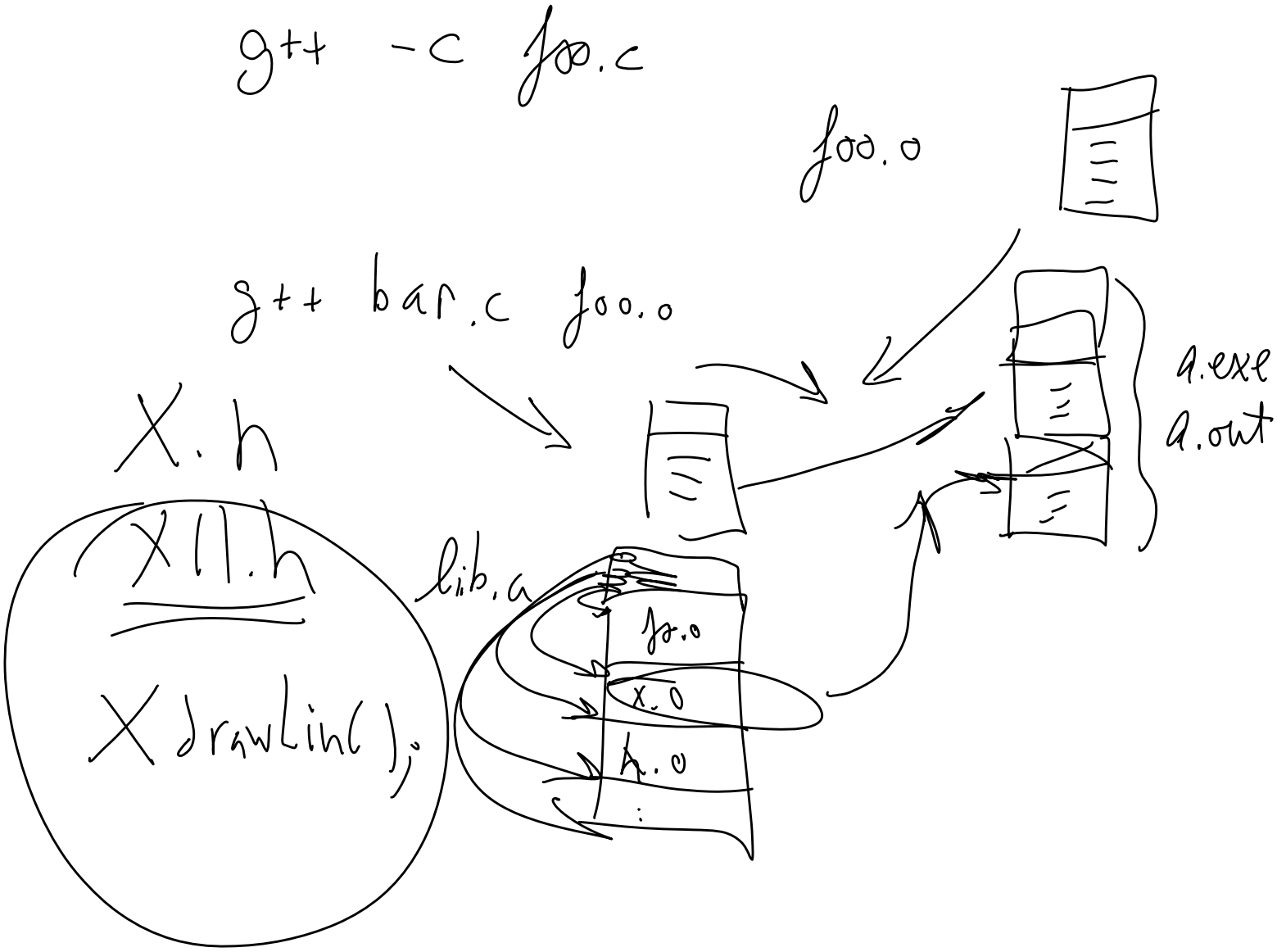
Char *p;

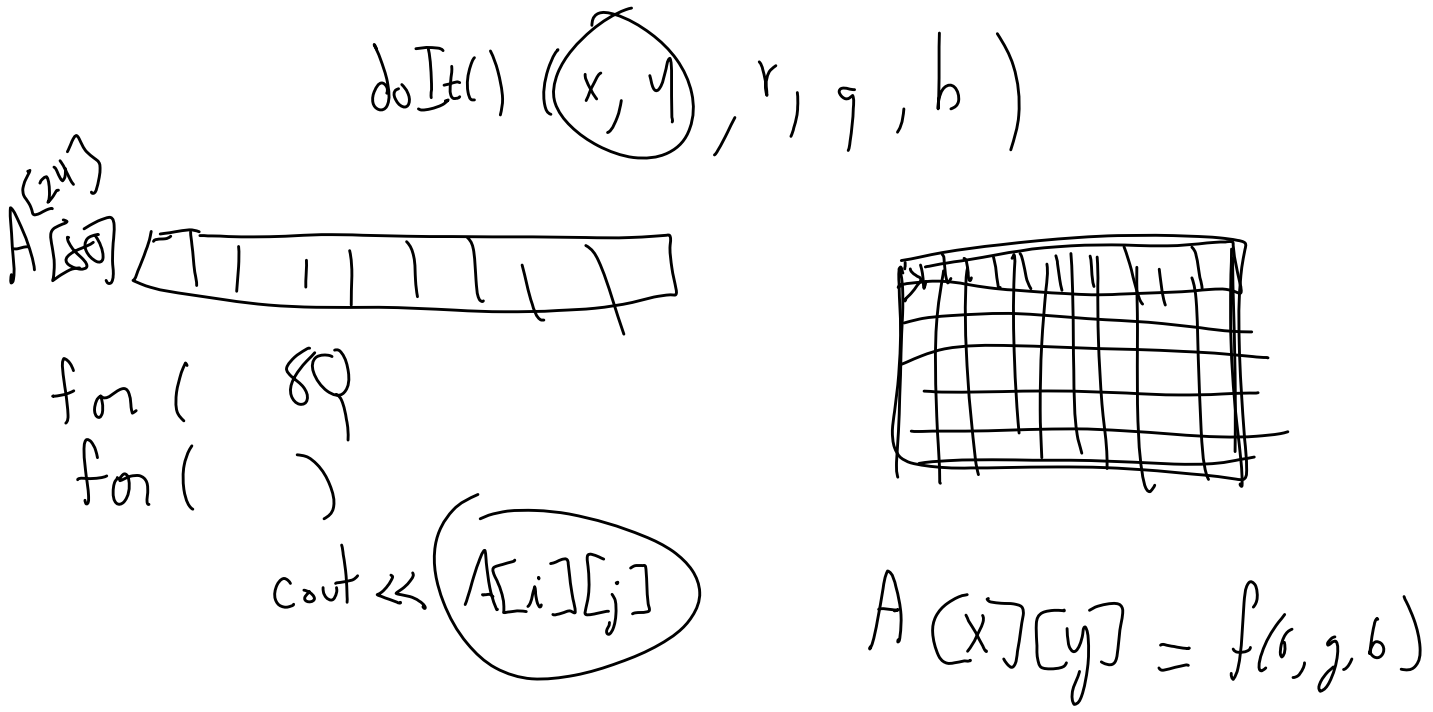
p = A

cost ← p[0]

cost ← *p




g++ -c foo.c
foo.o
g++ bar.c foo.o
X.h
X11.h
XdrawLine();
lib.a
fo.o
x.o
h.o
a.exe
a.out



Implement class Graphics based on plain text output as your low-level interface to the display. Starting ideas:

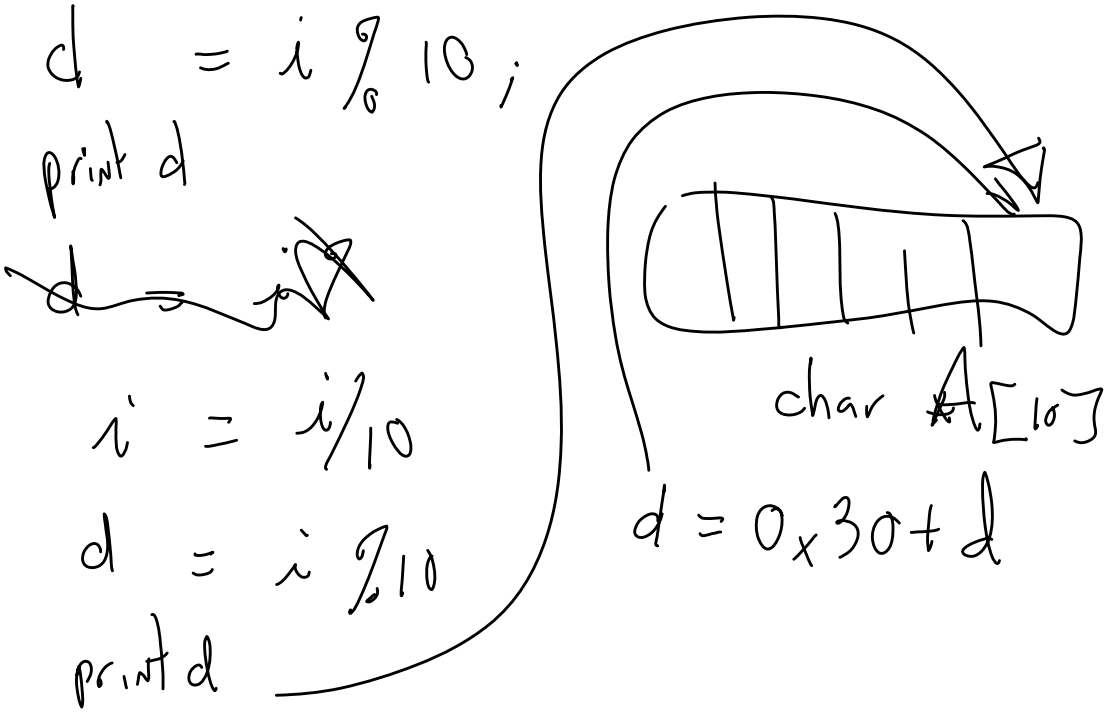
-- define a frame buffer of 50X50 (rows, cols): char fb[50][50].

-- size the window you run sim in to be exactly 50X50.

-- transfering the frame buffer to the screen is a display "refresh".

```
for( row = 0; row < RowSize; row++ ){
   for( col = 0; col < ColSize; col++ ) {
       cout << fb[row][col] << endl;
   }
}
```

-- the big job: deciding how to map (r,g,b) values to chars. E.g., (0,0,0) ==> '#', use ranges of values per char, ...



SyntaxCorner

-r 404 const int
-r 408 const methods, cpp (the pre-compiler) and #define, #ifndef in .h files,

sim.cpp: ints to strings, <string> and <sstream> , overloaded "<<" conversions, templates.