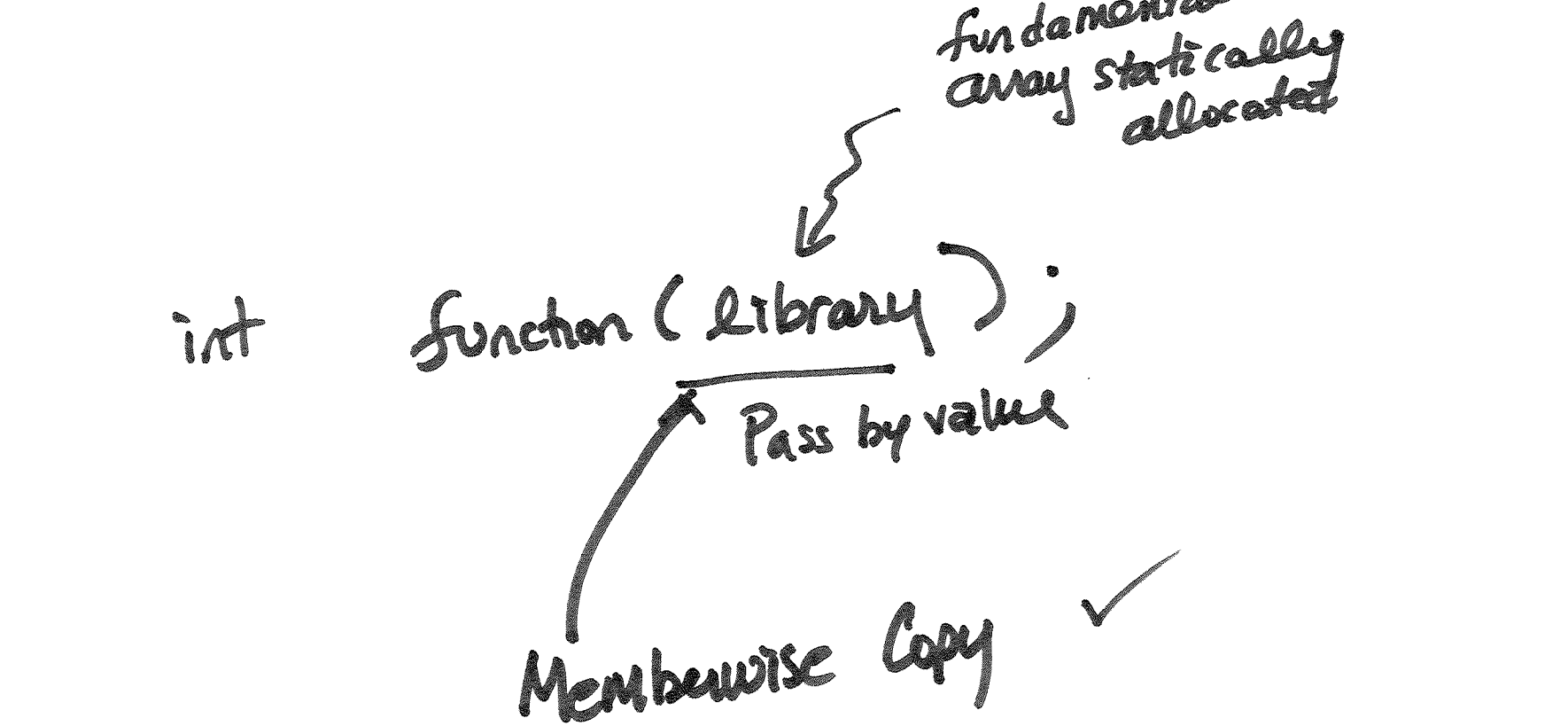


---

what if ... library node * head ;

Now Membewise Copy will break.

Main

Function

{

library

object;

int function(library local)

Pass by value

{

.

.

.

function(object);

Lifetime of "object" is over

}

Lifetime of "local" is over

Destructor will be invoked

}

Main

Function

const

int function (library &

reference)

library object;

{

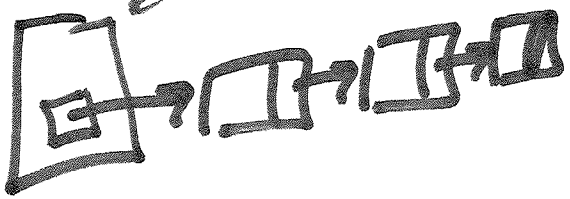
function (object);

}

← Nothing happens

Main

library object;



function (& object);

↑

Address of

function

int function (library * ptr)

{

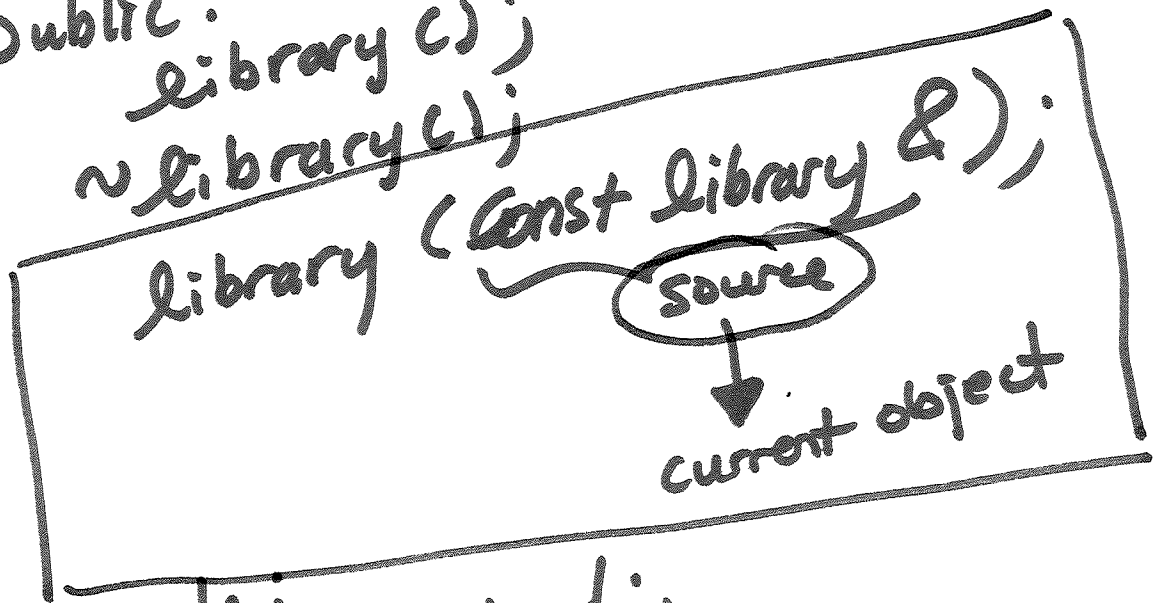
(*ptr). display();

ptr → display();

}

← Nothing to happens to the object

```
class library
{  public:
     library();
     ~library();
     library(const library &);
   private:
     node * head;
};
```

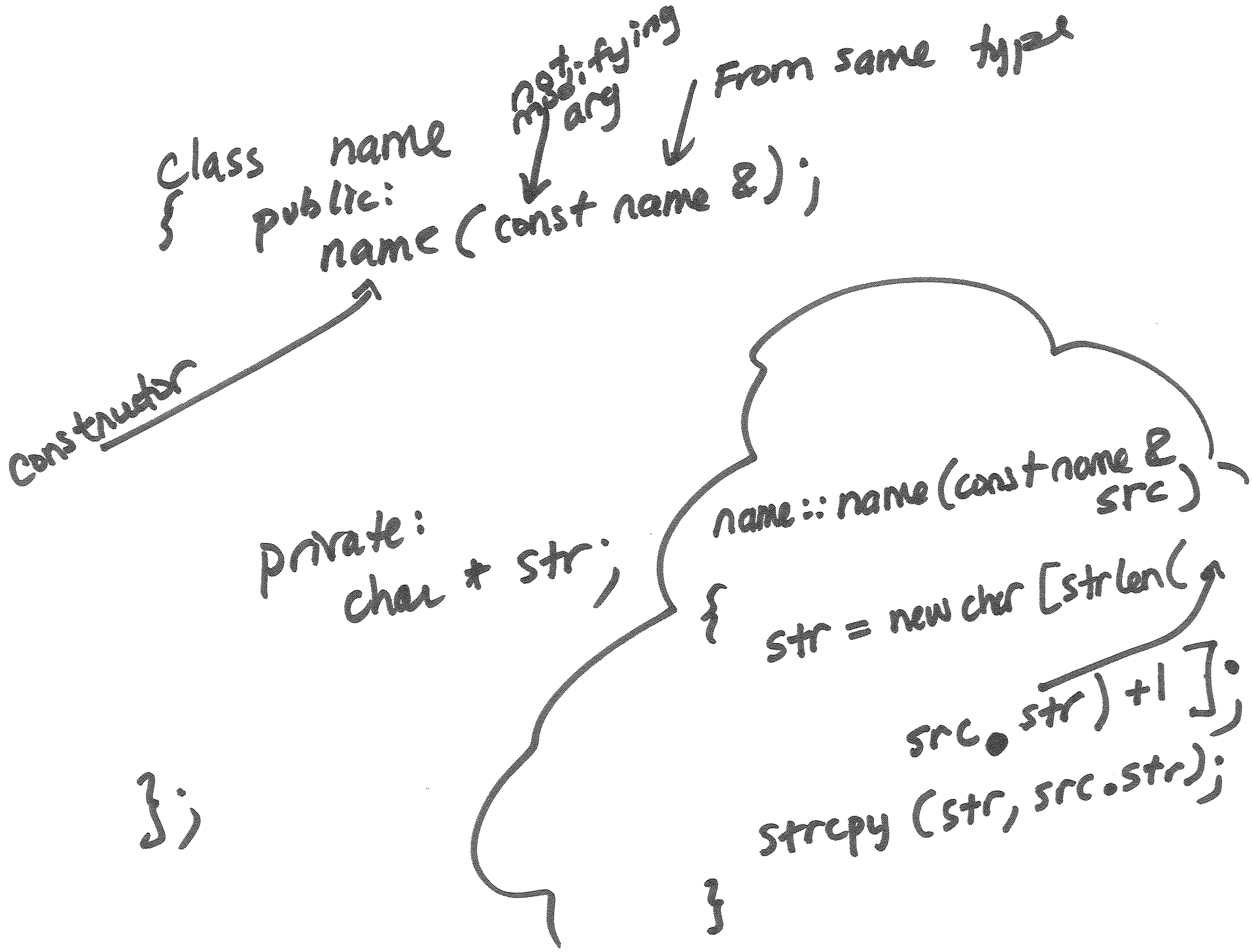


```
library::library ( const library & src)
{       copy (head, src.head);
}

void library::copy ( node *& dest,
                     node * source )

{   if (!source)
    {   dest = NULL;
        return;
    }
    dest = new node;
    dest-> copy ( source); <-- //Look at
    copy ( dest->next, source->next);

}
```


class name
{
public:
name(const name &);
};
constructor
modifying arg
From same type
private:
char * str;
name::name(const name &src)
{
str = new char [strlen(src.str)+1];
strcpy(str, src.str);
}

Using Copy Constructor

- Pass by Value
- Return by Value

- Initializing
  name my-name;
  name obj = my-name;

  or name obj (my-name);

1.

string (const string &)

String

So does this one

Name

Copy constructor that you write

Name::Name(const name & object)

String(object)

"is a"

{ ... }

A

Doesn't

B

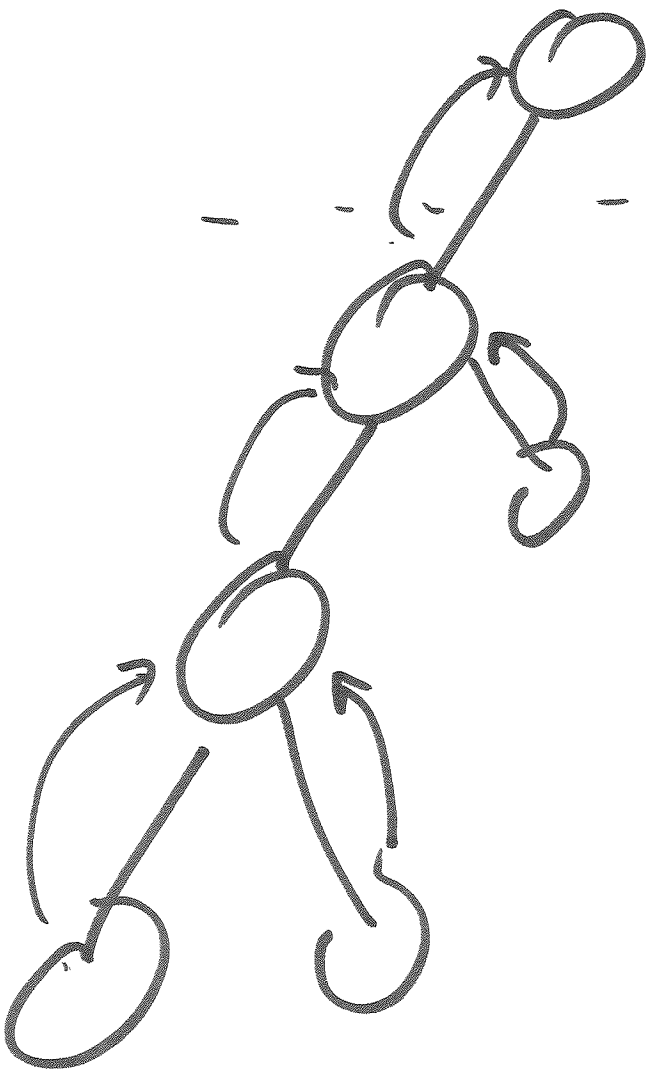
Has a copy constructor you have written

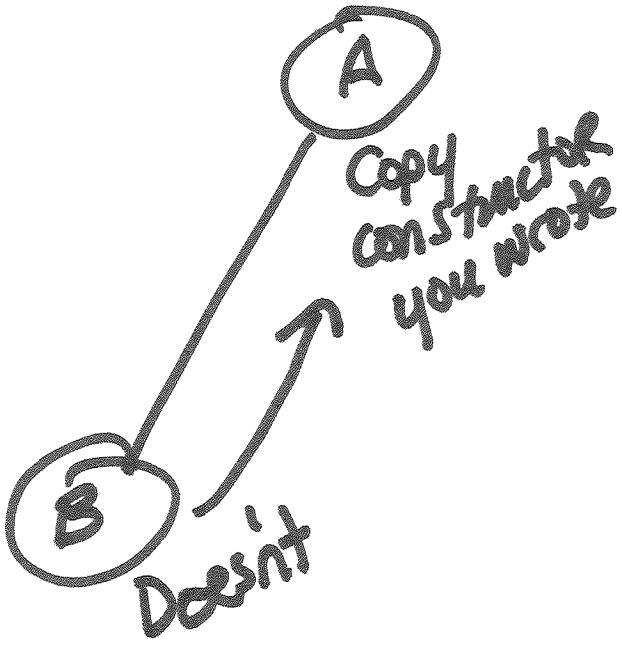
* I still NEED

B::B(const B& object)

A(object)

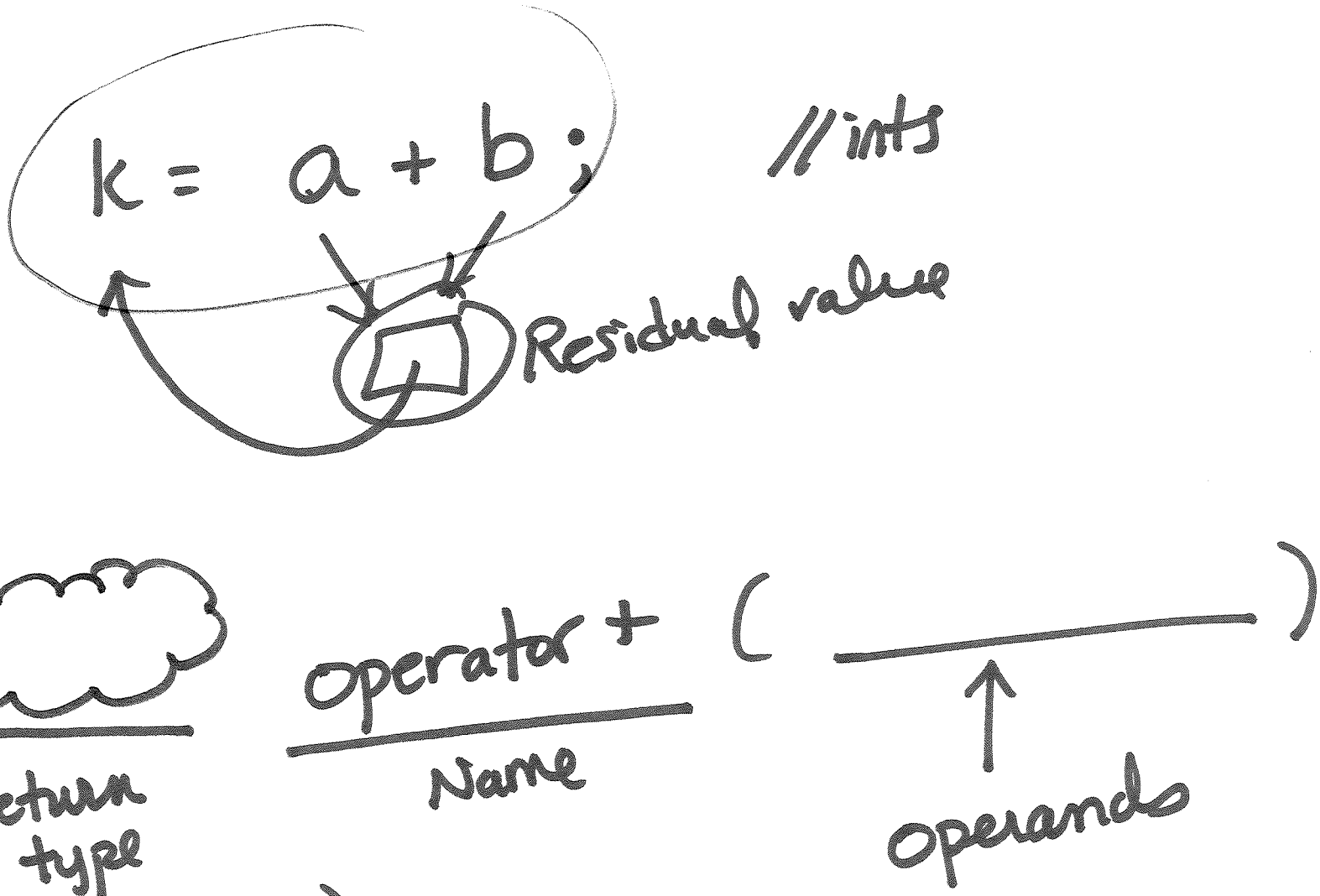
{ }

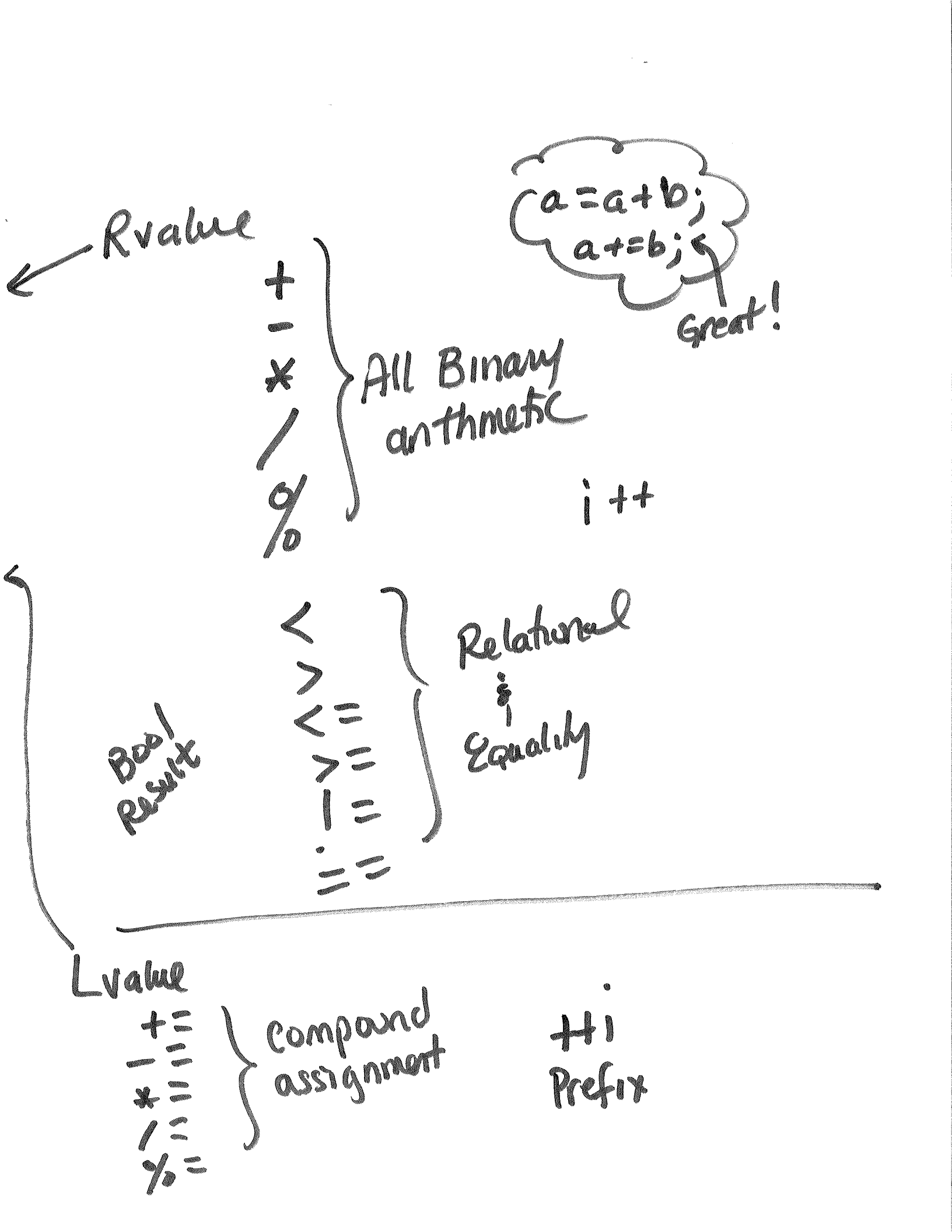




WE ARE Free!

We don't have to do anything..


k = a + b; //ints
Residual value
return type
(Residual value)
operator +
Name
operands


Rvalue
a = a + b;
a += b;
Great!
+
-
*
/
%
All Binary arithmetic
i ++
Bool result
<
>
<=
>=
!=
==
Relational & Equality
Lvalue
+=
-=
*=
/=
%=
compound assignment
++i
Prefix

Return by value

"right" Temporary Memory

rvalue - CAN ONLY be on the right hand side of =

Return by Reference

"Left"

lvalue - CAN be on the Left of an assignment operation

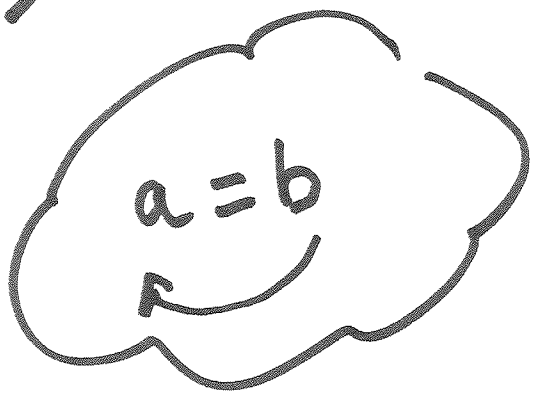
- Memory YOU control
- Variable

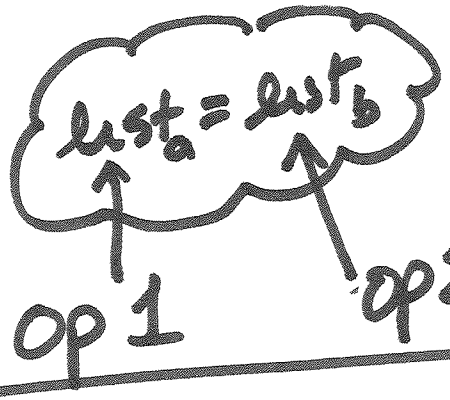
a = 10;

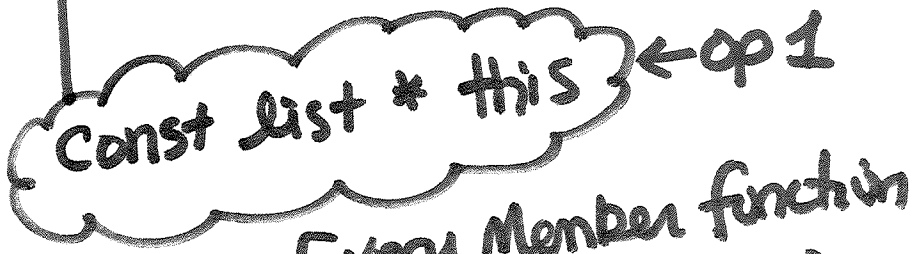
↑ lvalue

Lvalue

list & list::operator = (const list & op2);



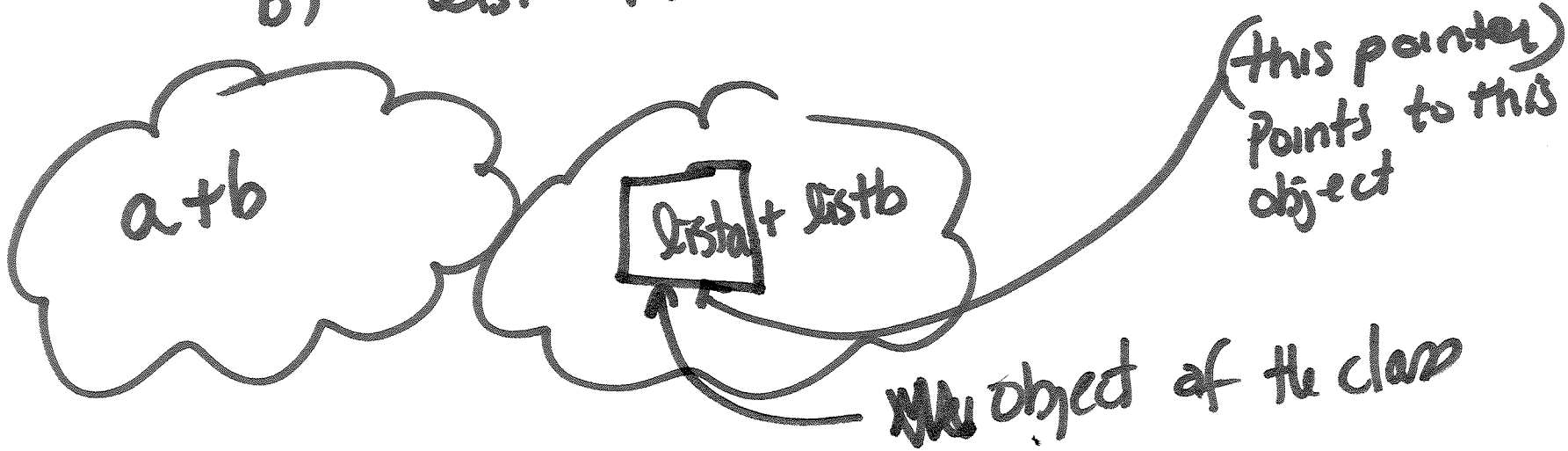


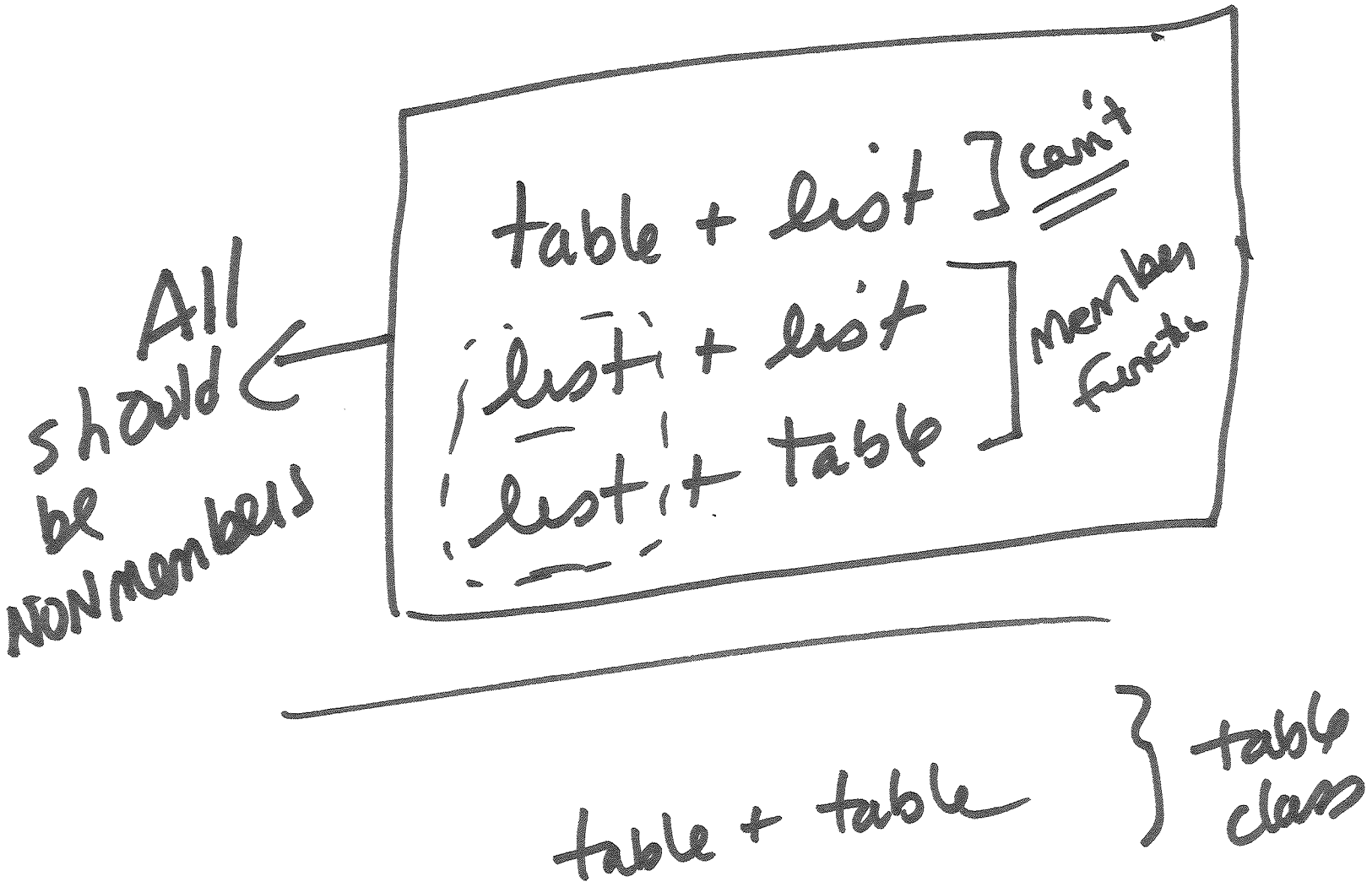


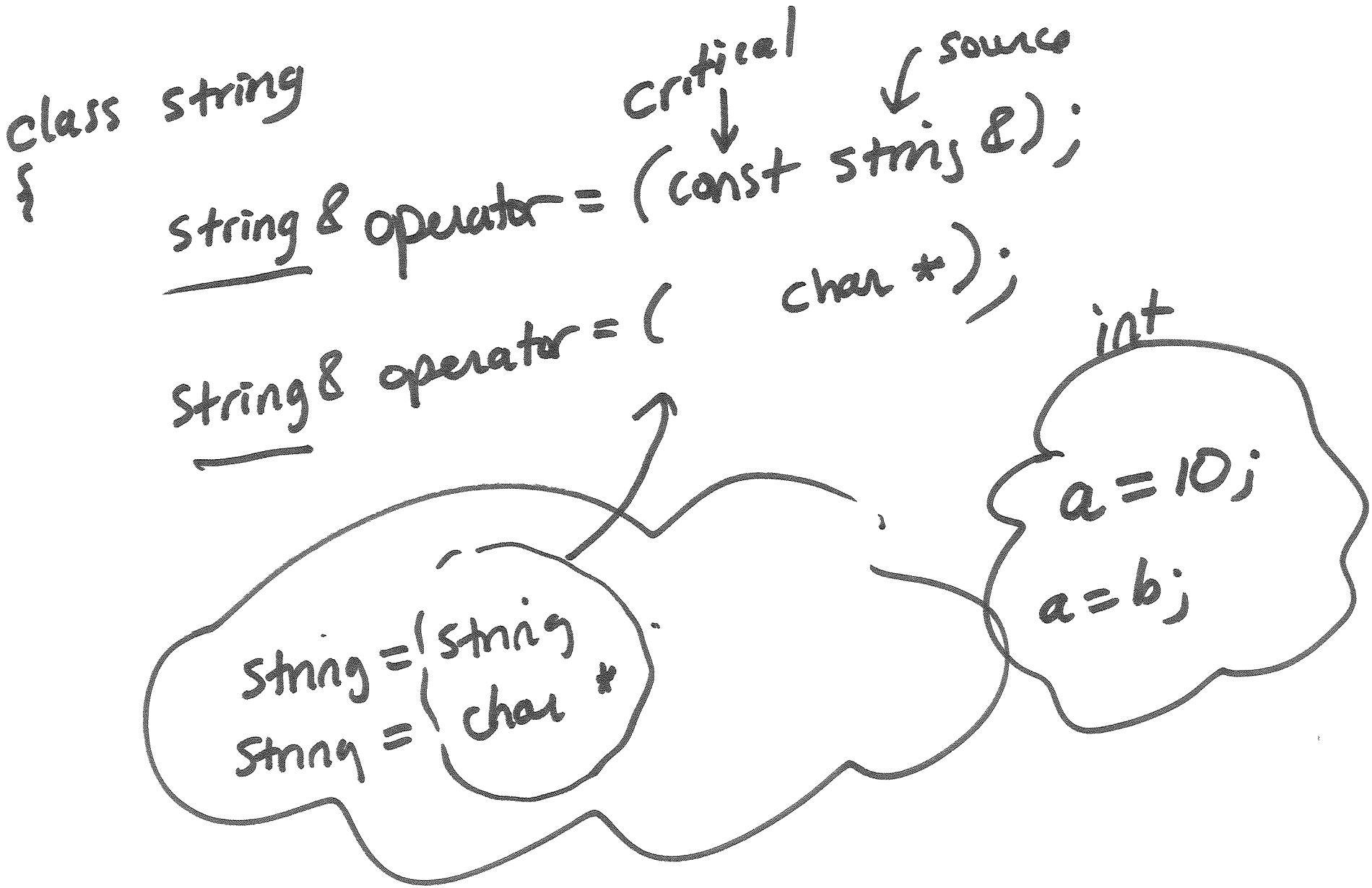
A) list operator + (const list & op1, const list & op2);

(value)

B) list list :: operator + (const list & op2);




All should be NONmembers
table + list
list + list
list + table
can't
member functions
table + table
table class


class string
{
string & operator = (const string &);
critical
source
string & operator = (char *);
int
a = 10;
a = b;
string = (string
string = (char *

1st

① Self Assignment

i = i;

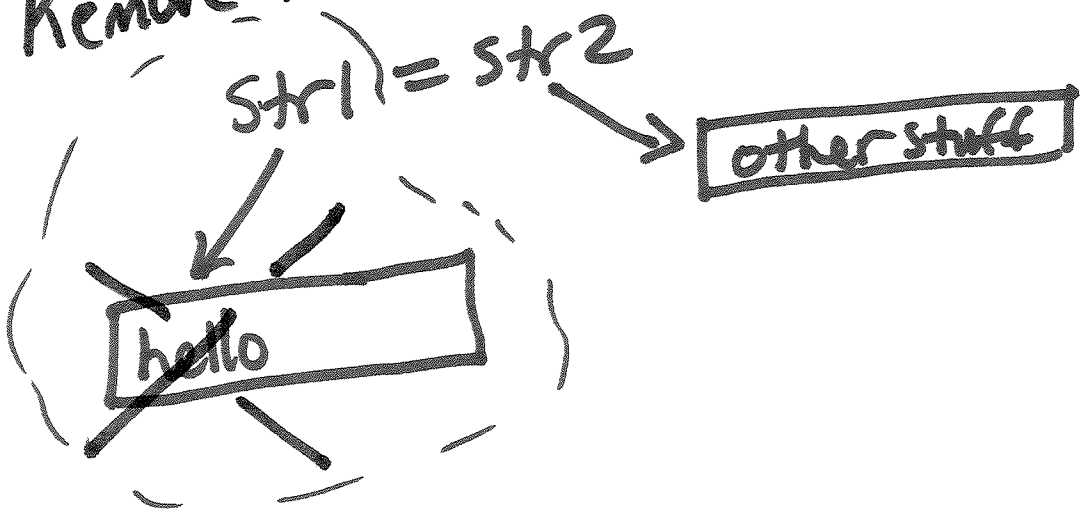
if(this == &s2)

i = f(i);

Nothing to do

② Remove Memory used current object

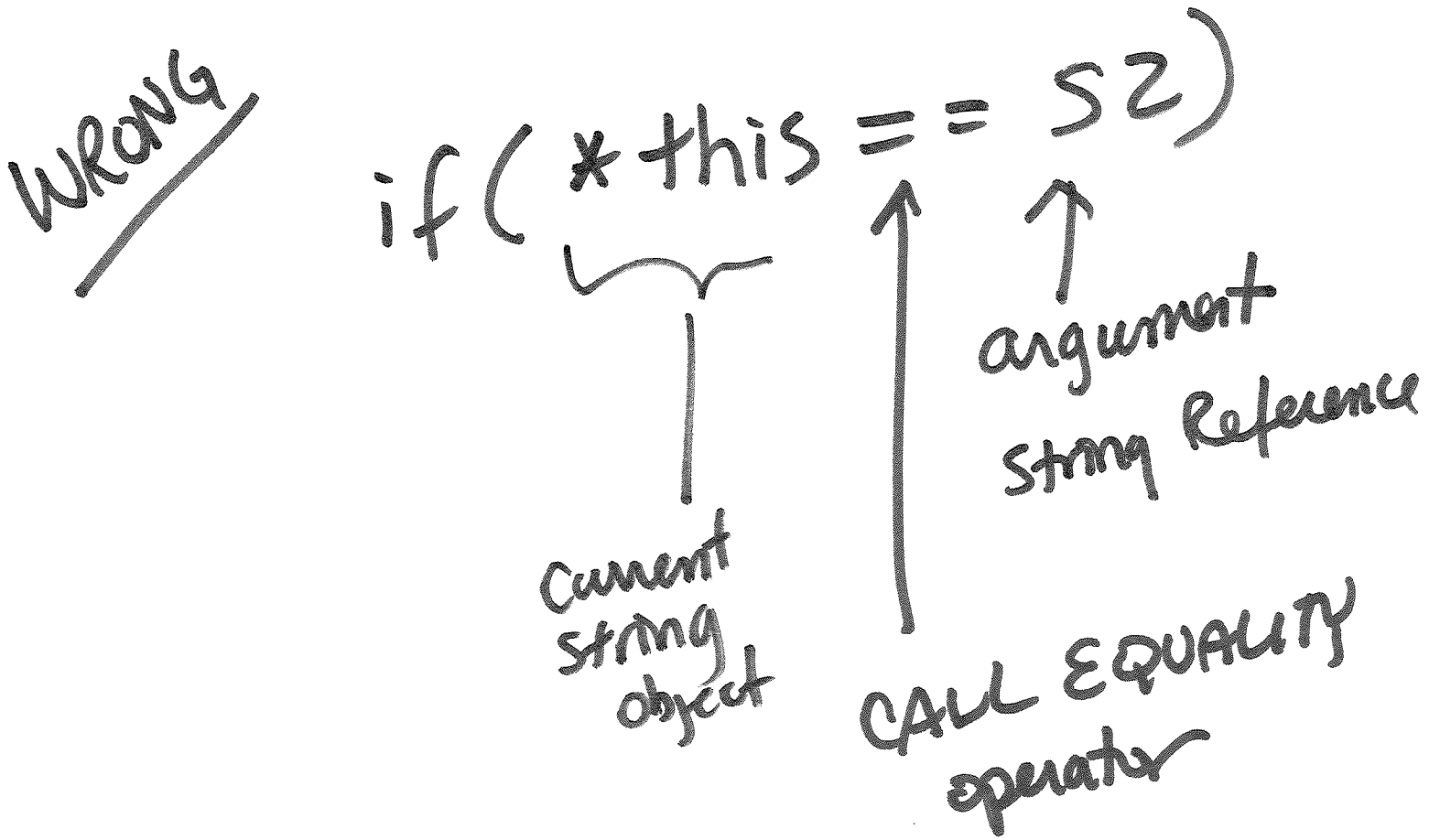
else

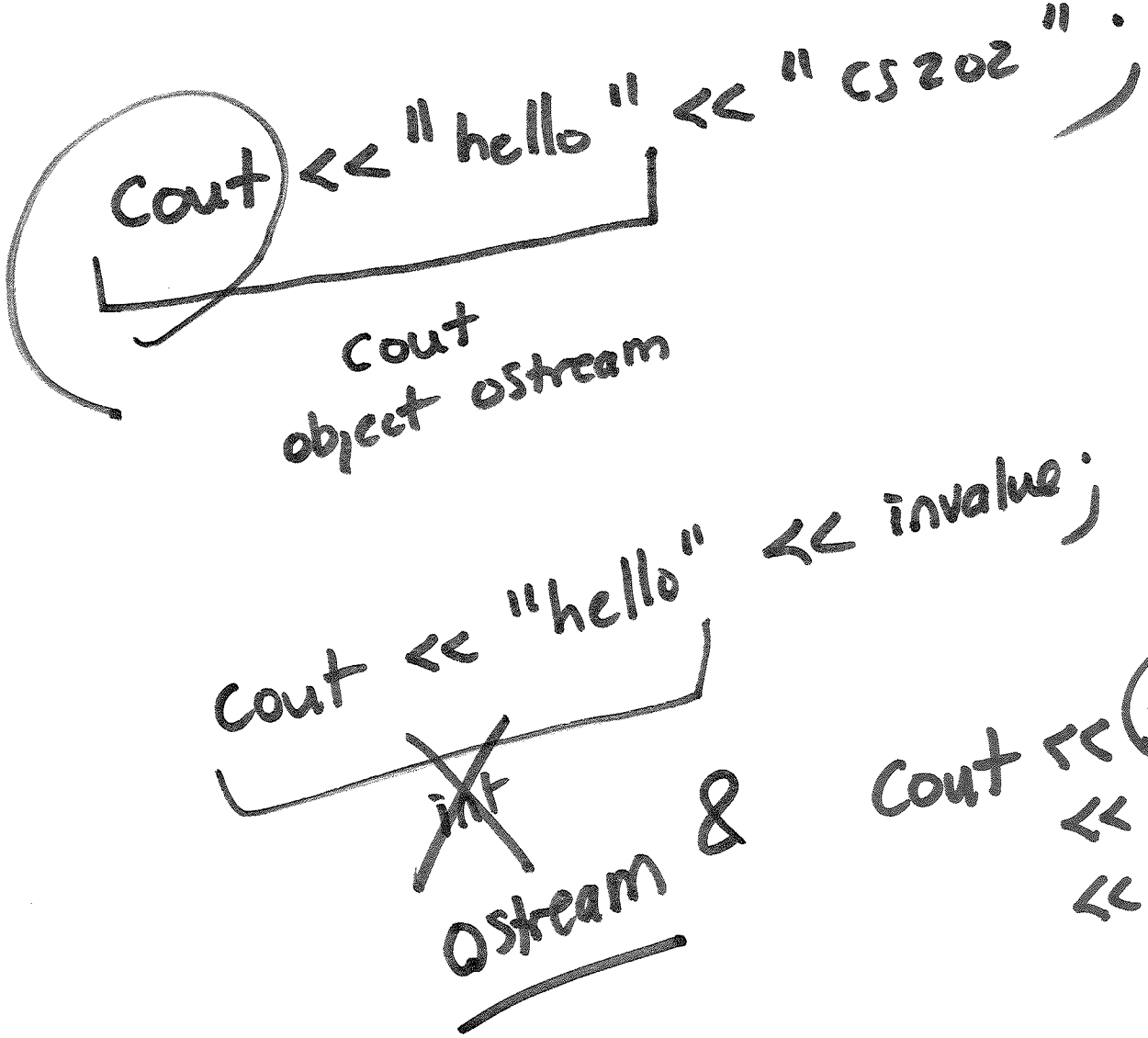


③

str1 = str1;

Risks seg fault


Wrong
if(*this == s2)
current string object
CALL EQUALITY operator
argument string Reference


cout << "hello" << "cs202";
cout
object ostream
cout << "hello" << invalue;
int
ostream &
cout << str << "block" << intv;