Topics for Midterm

- Inheritance
  - Single
  - ~~Multiple~~
  - ~~Virtual~~
- Dynamic Binding
  - pointers to base class
  - Abstract Base class
  - Pure Virtual Function

NO RTTI

- OOP - blackboard
- Copy Constructors ← Go program it!

Closed Book / closed Notes

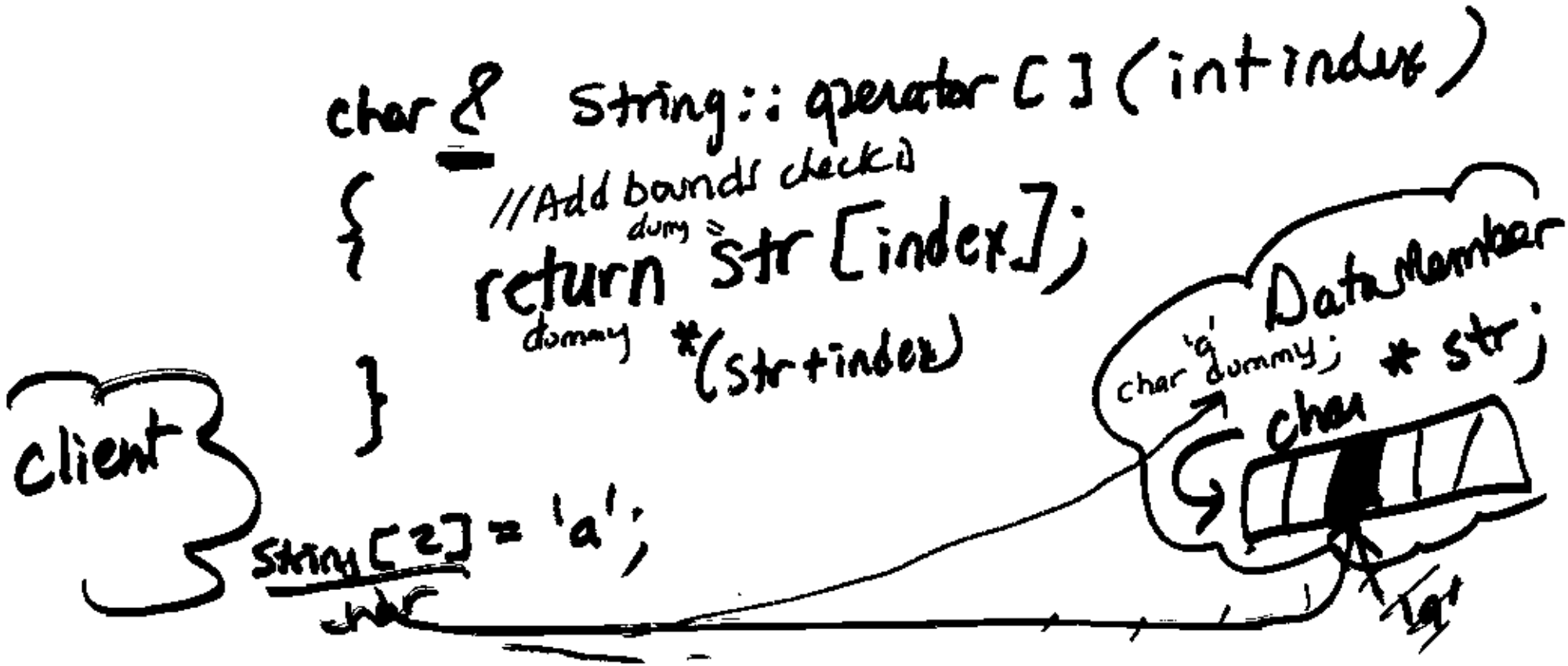
```
class data : public node
{

};
```

LLL - singly linked, linear
- adding at end
- removing at end
- remove the last 2 items
- make a copy

DLL - linear
- make a copy
- remove at either end

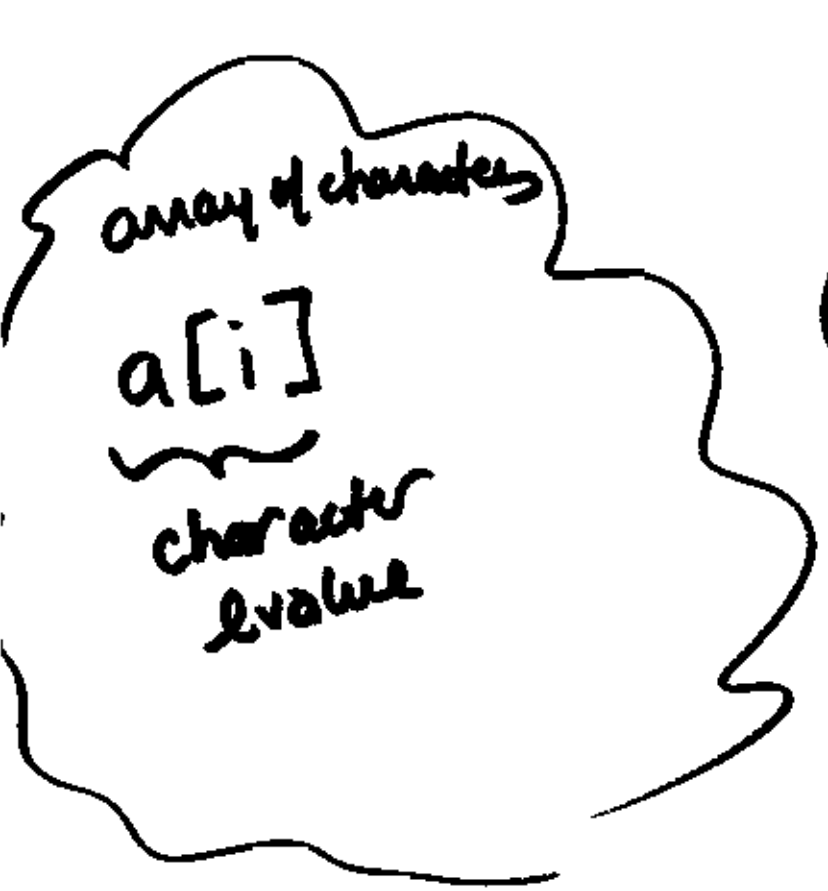
BST
- copy tree
- remove leaf / largest / smallest
- copy tree into LLL

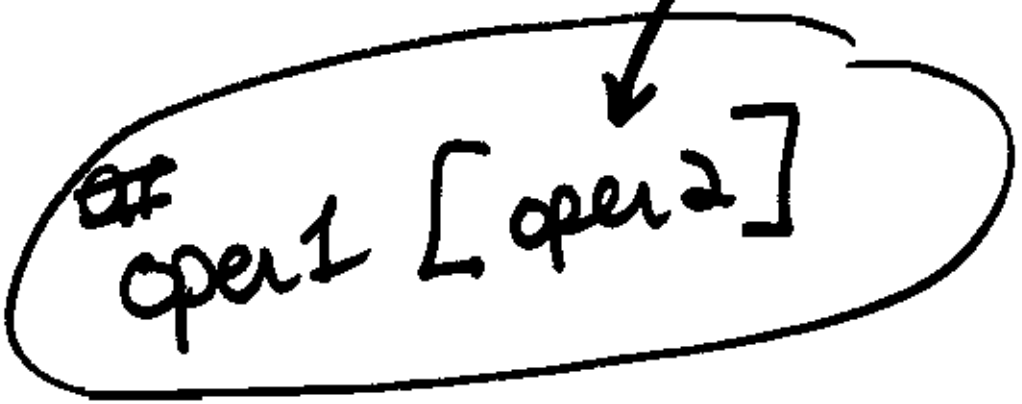
lvalue
array [0..1]
operand 1
operand 2
rvalue ?
lvalue ?
left =

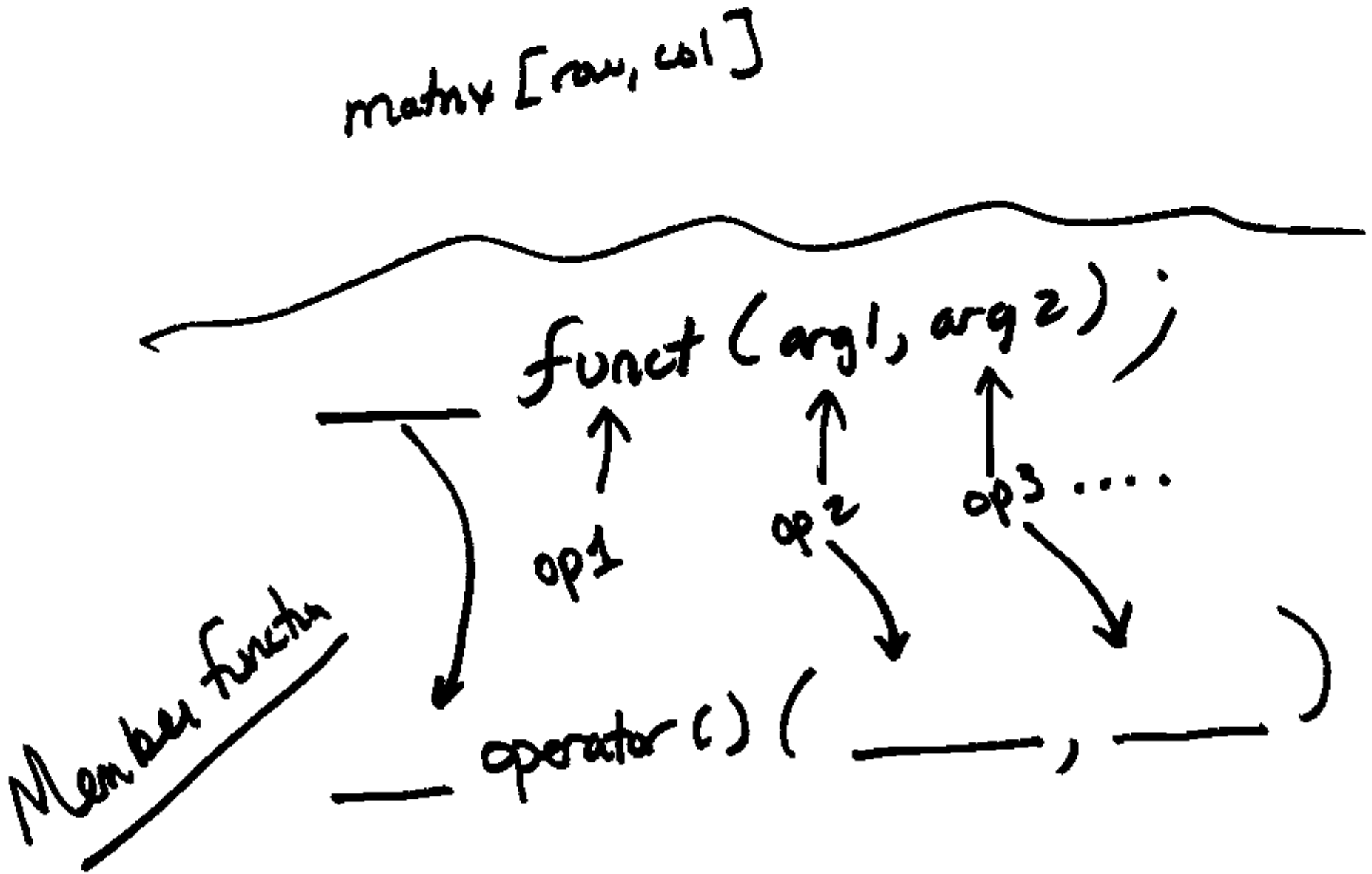

char & String:: operator [] (int index)
{
//Add bound check
return str [index];
}
dummy = *(str+index)
client
String[2] = 'a';
char 'a' dummy;
Data Member
char * str;
char
Tg!

Member function

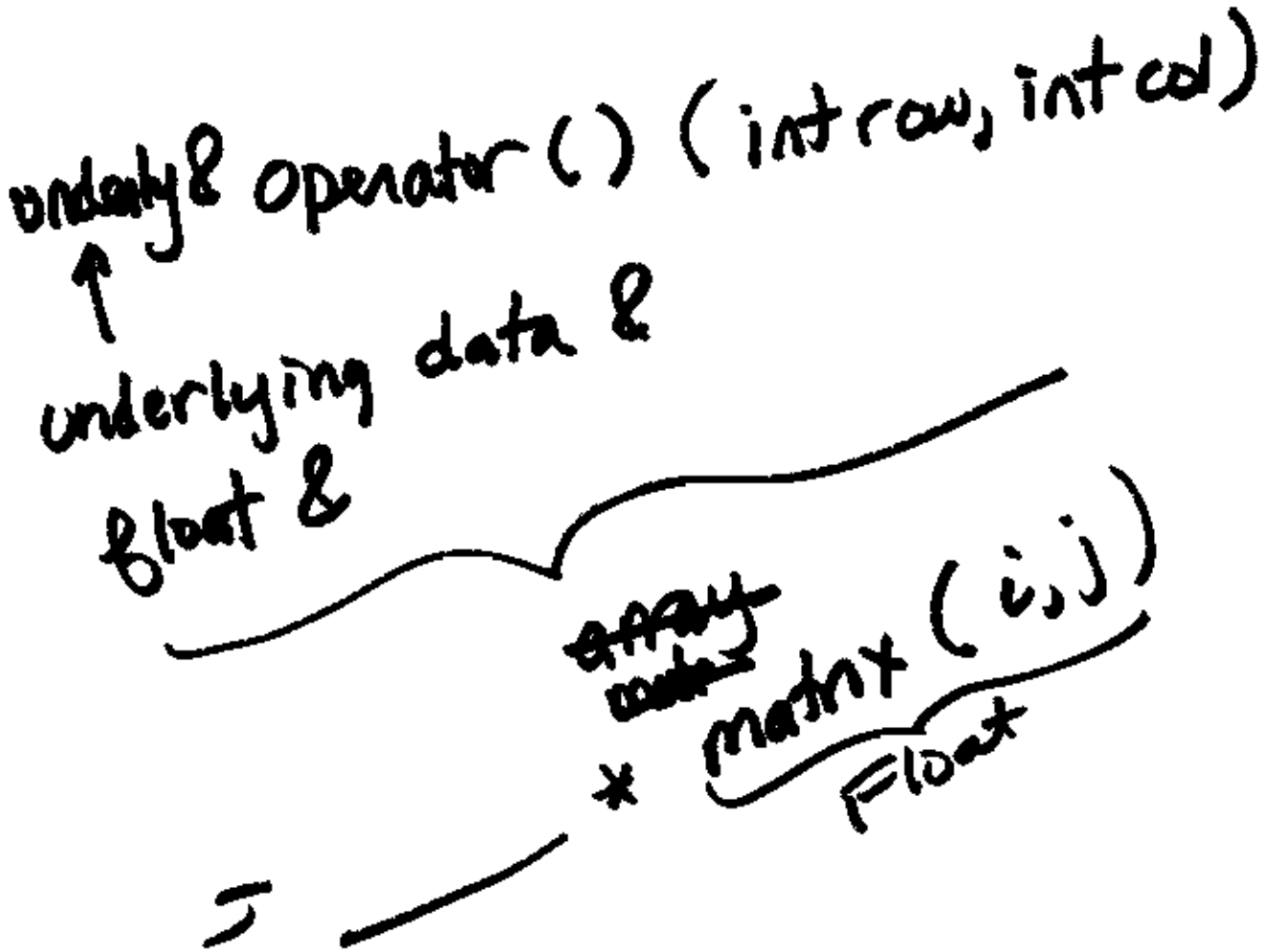
char & operator [] (int index)





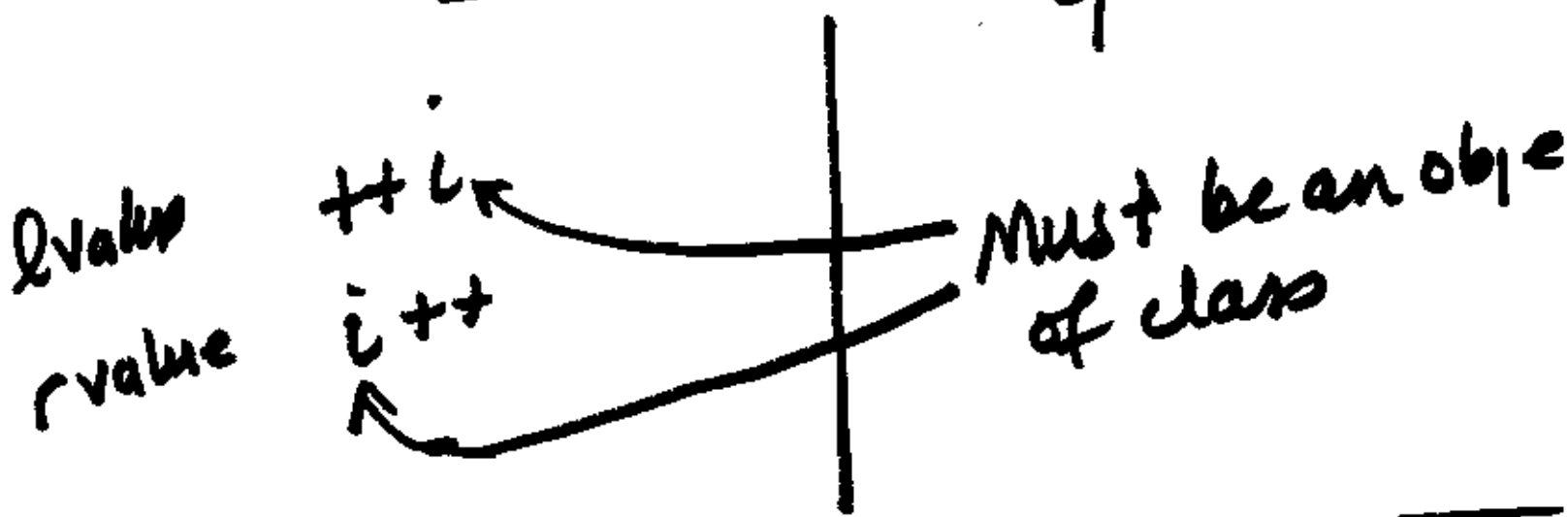

matrix [row, col]
funct (arg1, arg2);
op1
op2
op3 ....
Member functn
operator ( ) (,)

$$\underbrace{(i+1)*(md+i)*(md+i+1)}_{\text{multiplications}}$$


underlyingData& operator() (int row, int col)
underlying data &
float &
array index
matrix (i,j)
Float
*
=

```
counter obj;
++obj;   // obj.operator++();
obj++;   // obj.operator++(0);
```

Increment operator ov.



Member counter class

counter& operator++ ( ); //prefix

counter operator++ (int); //postfix

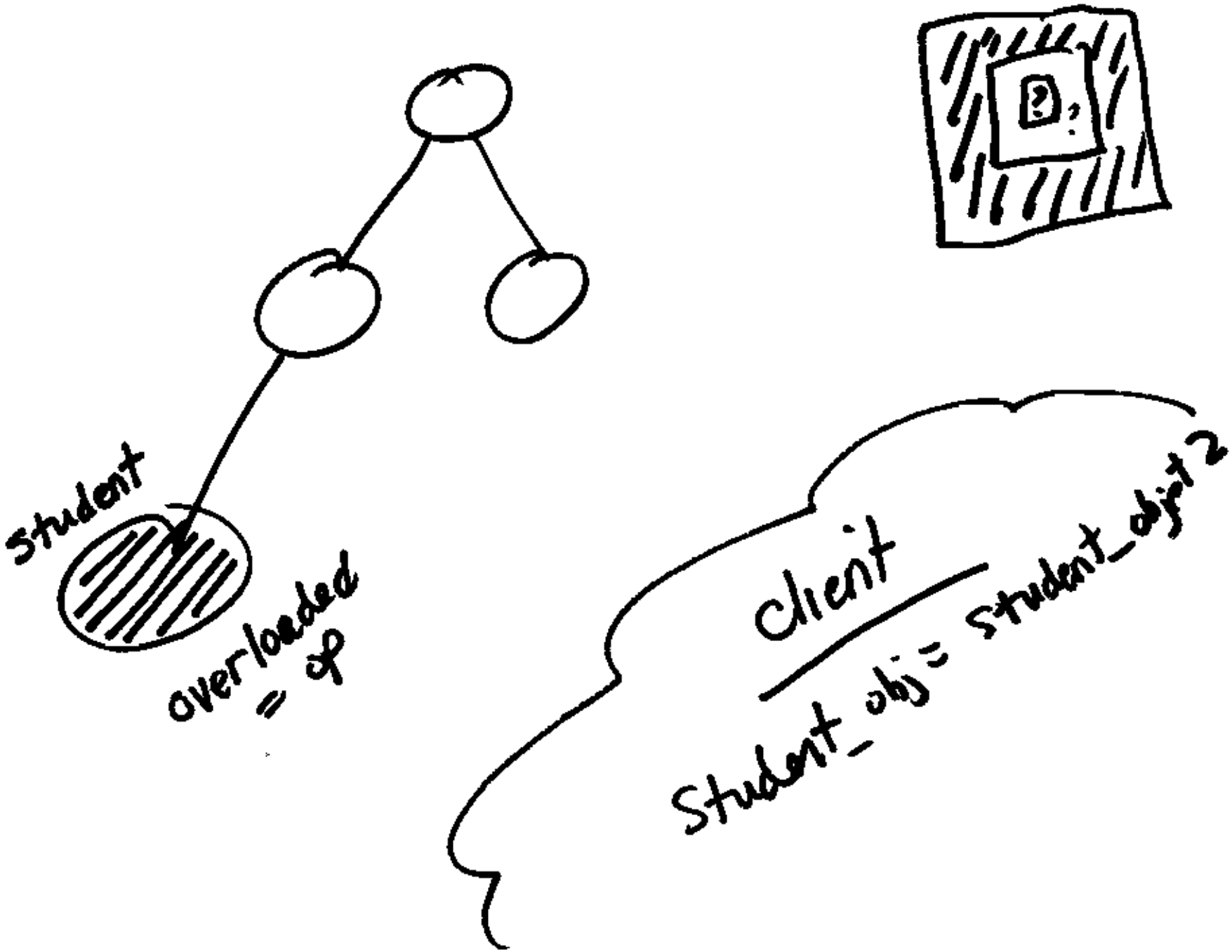
```
counter counter::operator++(int)
{
    counter temp(*this);
    data+=1;
    return temp;
}
```

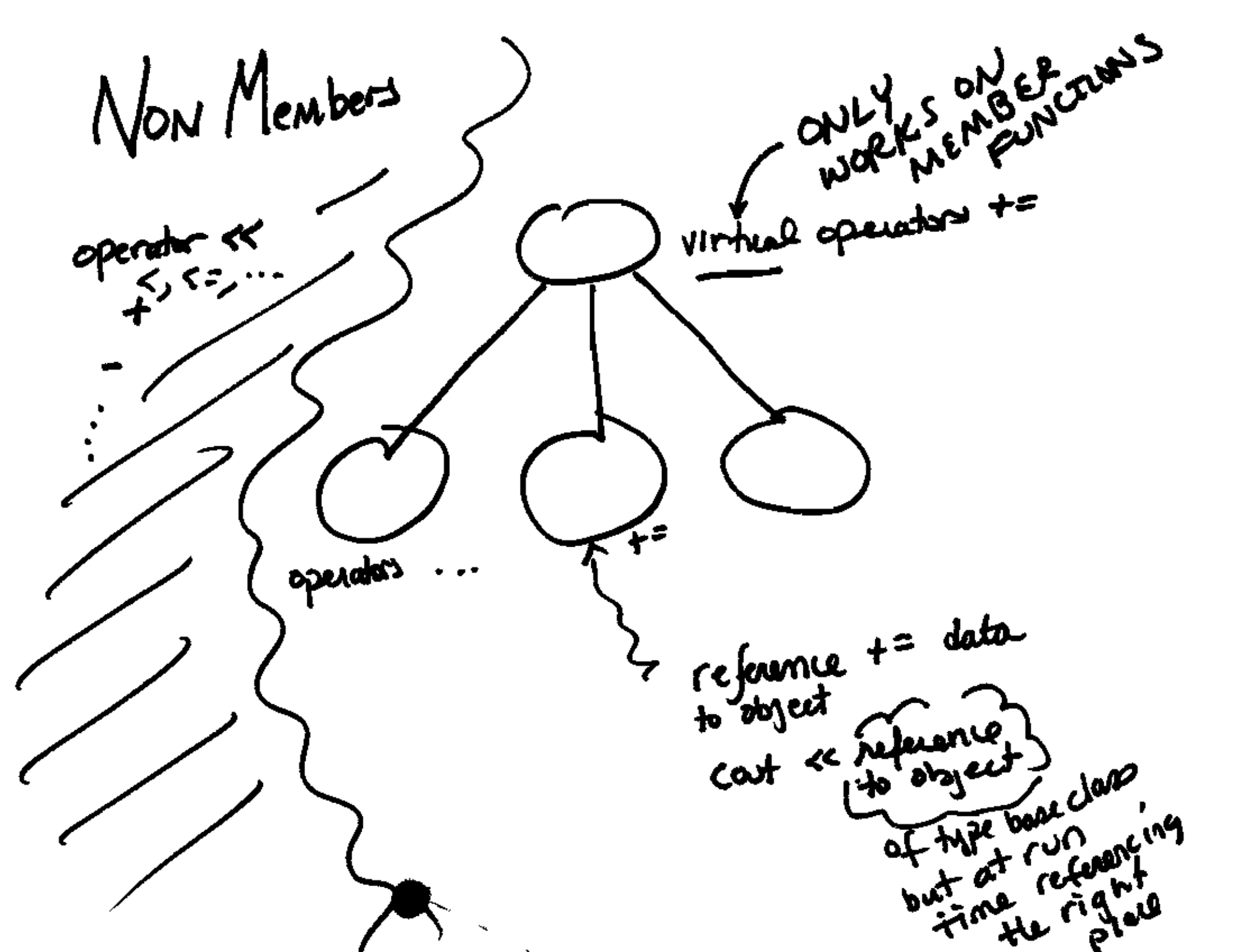
current object

i++

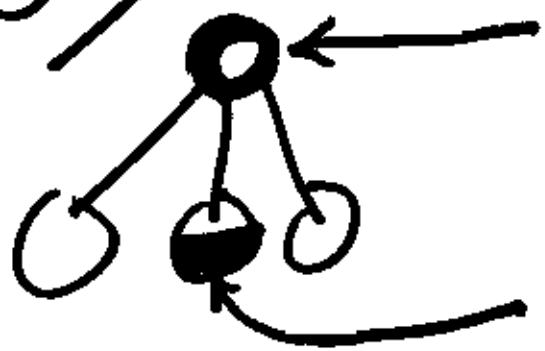
1) Grab value i (temp)
2) Increment i
3) Residual value is temp

(s1) const student & operator = (const student & s1)
{
checking::operator = (s1);
}
thus → checking::operator = (s1);
checking
=
=
=
assign
op
our


Student
overloaded
= op
Client
Student_obj = student_object2


NON Members
operator <<
+, +=, ...
-
...
ONLY WORKS ON MEMBER FUNCTIONS
virtual operators +=
operators ...
+=
reference += data
to object
cout << reference to object
of type base class
but at run time referencing the right place

Class



Base class

virtual / helper member functions

virtual void display(ostream &) =0;

optional

void derived::display(ostream &out)
{
    out << derivedata;
}

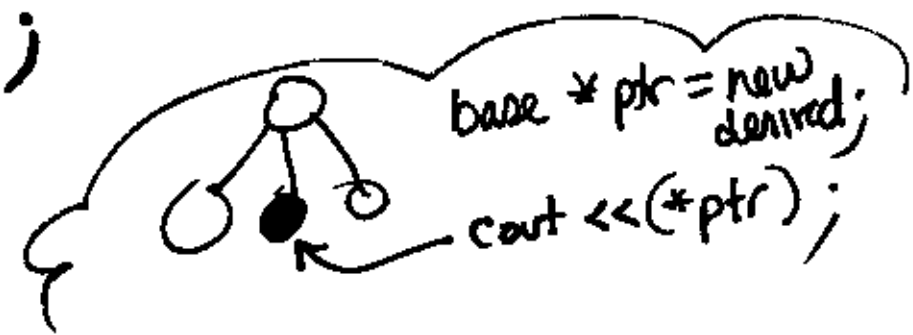
Non Member

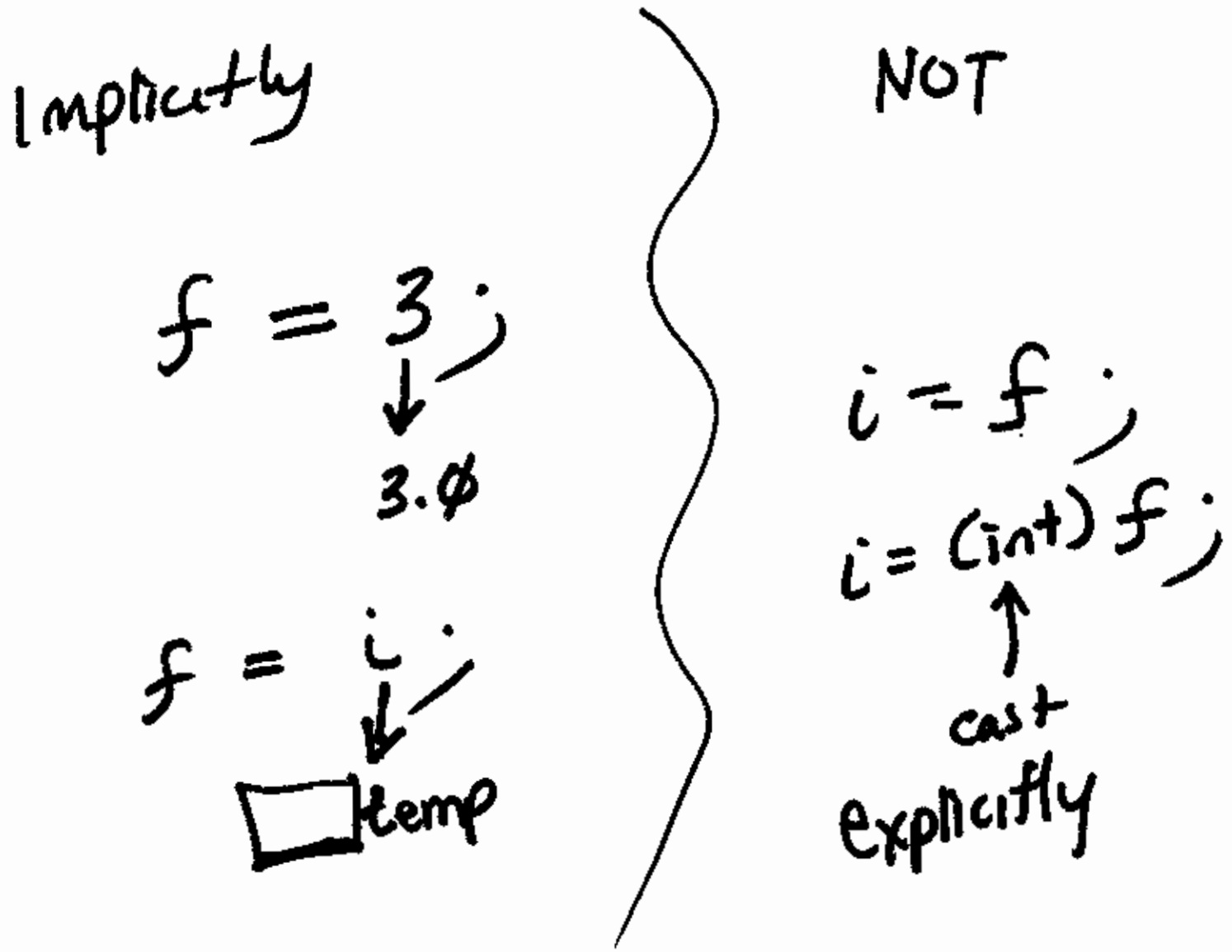
ostream & operator<<(ostream &out,
const base & obj)

reference to base

{
    obj.display(out);
    return out;
}

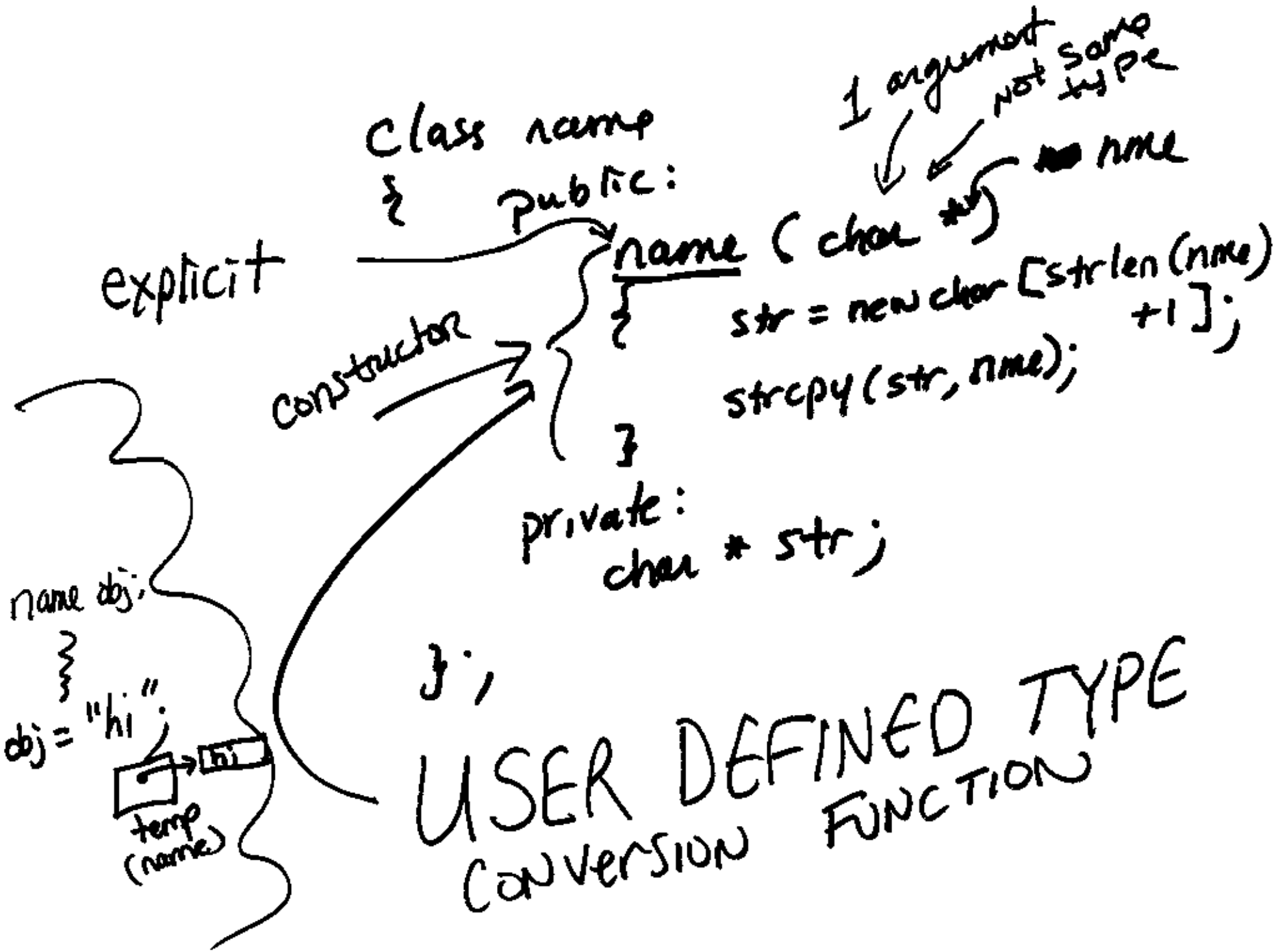
Static Binding




Implicitly
NOT
f = 3;
3.0
f = i;
temp
i = f;
i = (int) f;
cast
explicitly

f(char[])

f(char*)


Class name
{ public:
explicit
name (char *)
constructor
{
str = new char [strlen (nme)
+1];
strcpy (str, nme);
}
private:
char * str;
};
name obj;
obj = "hi";
temp (name)
USER DEFINED TYPE
CONVERSION FUNCTION

Explicit Type Conversion

Cast // C & C++

```
i = (int) f;
```

```
i = int (f);
```

functional notation
// Not in C

You can only put 1 name here

unsigned int

char *

Typedef

char **

charP.

C

Struct node

Node

obj = name ("hi"); ✓

1 copy

~~operator (string~~

Member
Name class

string operator ().

Makes 2 copies

↑ type conversion

MUST
Return by value

Name → string

string temp
temp = *str;
return temp;