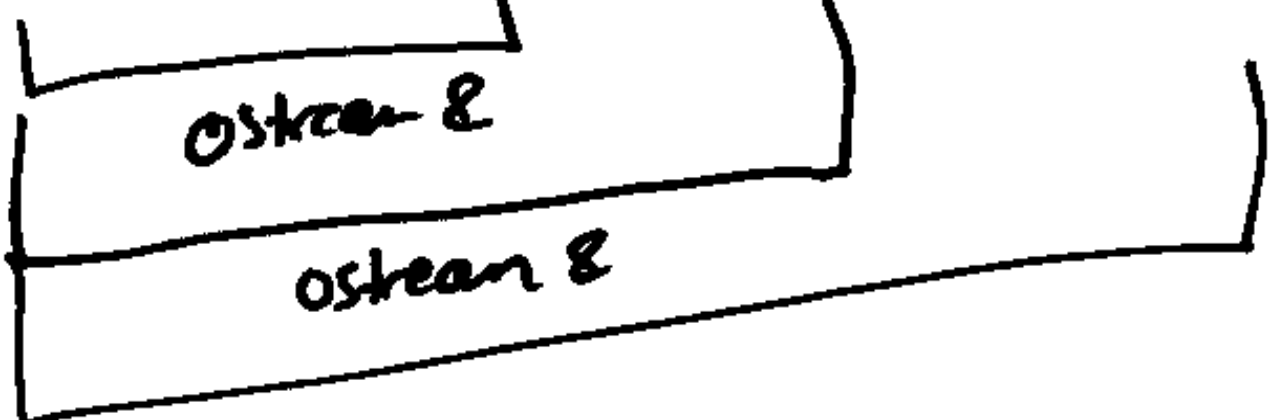
cin >> s1;

cout << s1;

cout << i << s1 << "more";



I/O

cin >> variable;

cout << variable;

istream

cin • operator >> (variable)

string

For Member operators

```
class String
{
  public:
    String & operator=(const String &);
    void f(const String &);
```

client

```
s1 = s2;
```

s1 → current object

s2 → Argument

```
s1.operator=(s2);
```

```
s1.f(s2);
ptr->f(s2);
```

if (*this == argument)

object string class

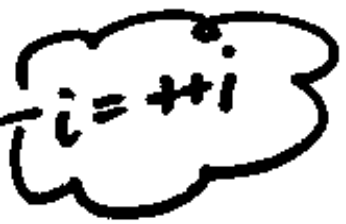
Reference to a string object

overload ==

Compare every element/character of strings to see if they are the same

Did you know....

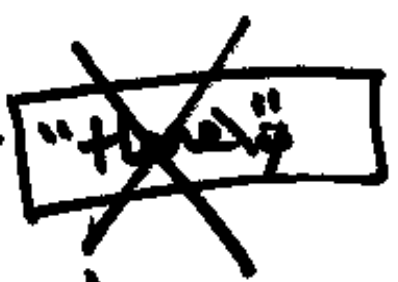
i = i;



i = f(i);

Self assignment

s1 = s1;



if (this == &argument)

pointer to current object

address of

Assignment

String s1,s2;

s1 = s2;

```
class list
{   public:
        void display () const;
};
```

```
void func (const list& obj)
{
    obj.display();
}
```

Constant member function

When you have a constant object ONLY constant member functions can be used.

string& string::operator=(const string&)

1 arg

↑

2nd operand

int a = 10;

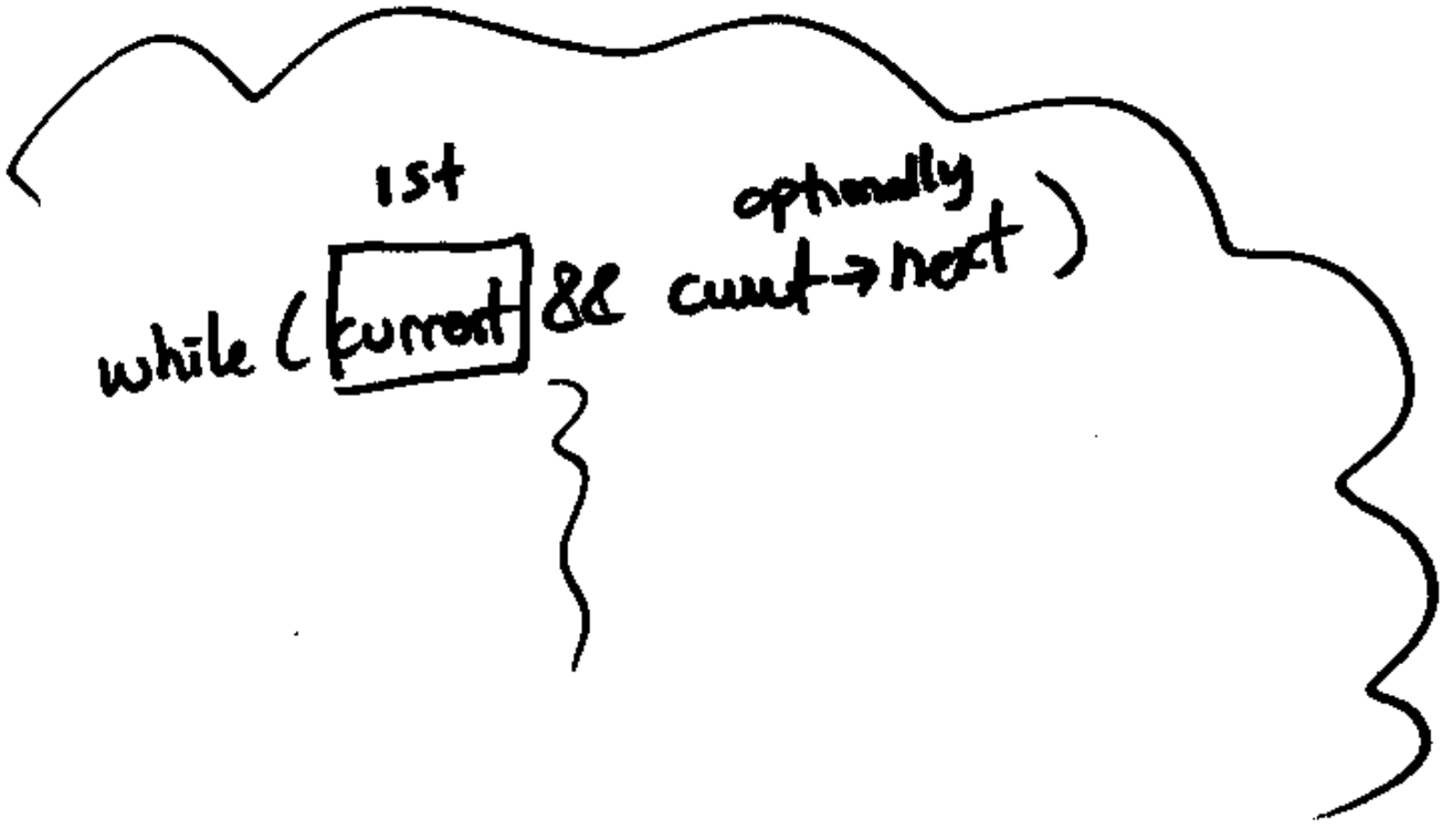
f(++a, a)

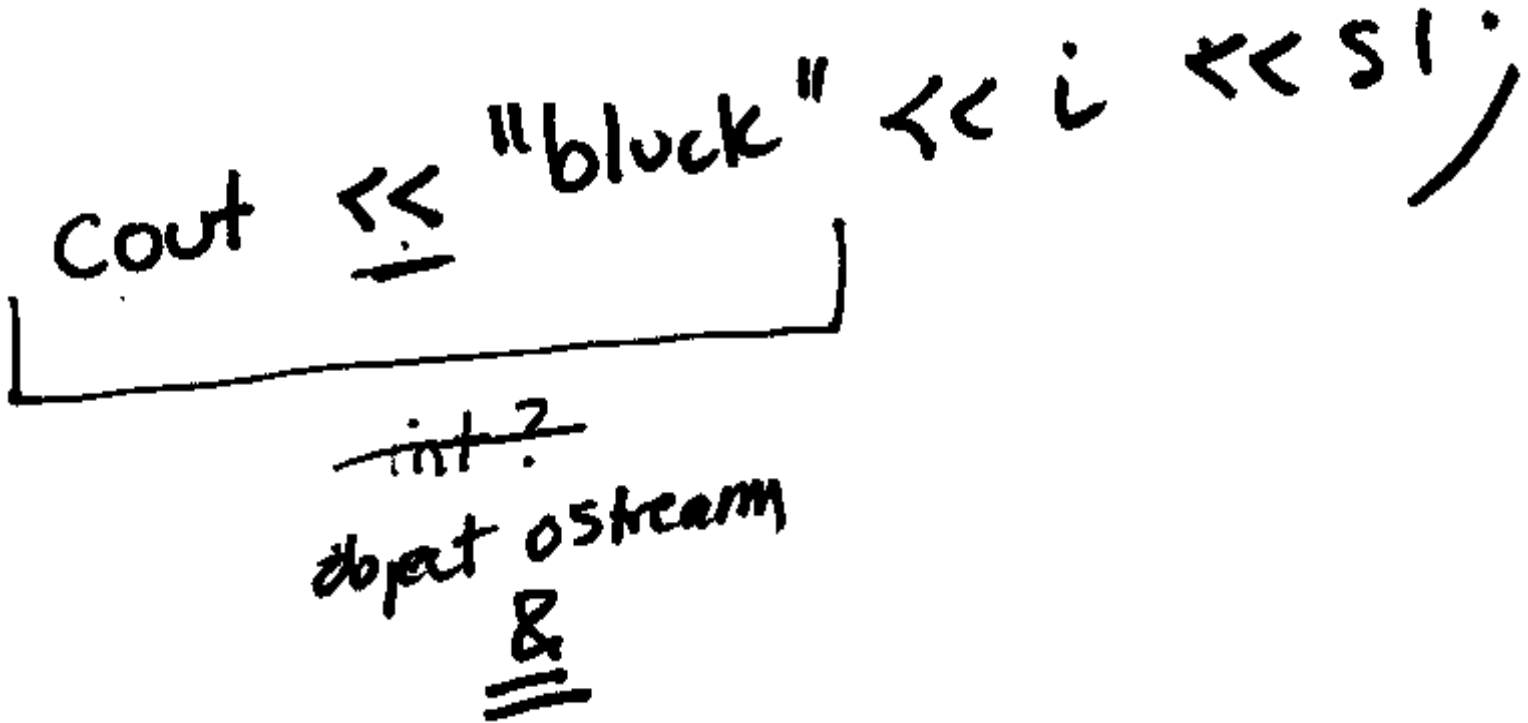
10 a = 12

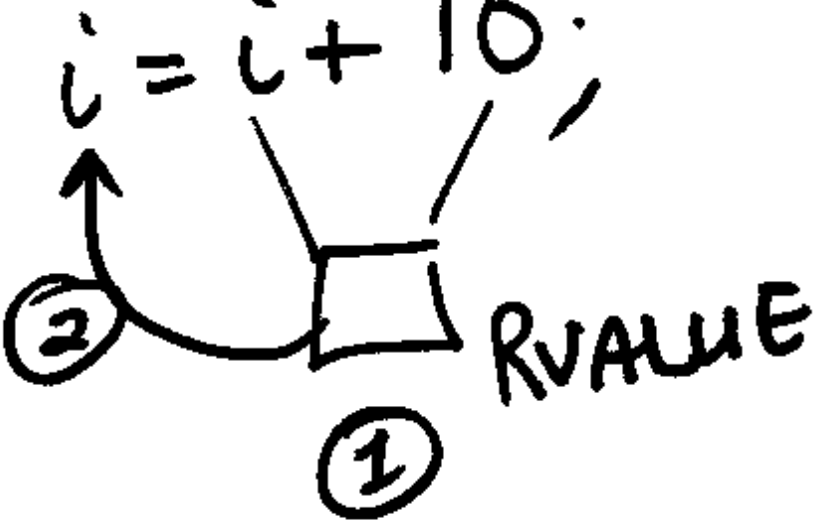
I believe - not overload

&& || ,

bool operator && (op1 , op2)

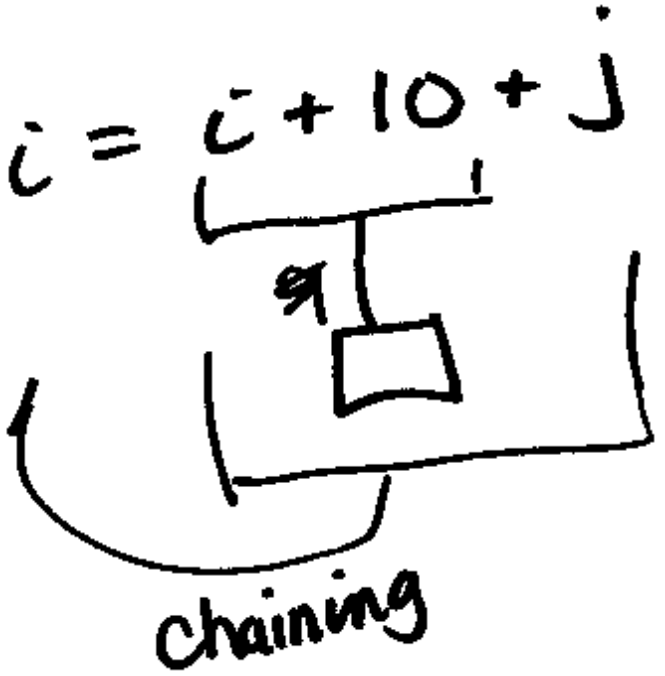



cout << "block" << i << 5;
int ?
object ostream
&



i += 10;

LVALUE



Operator Overloading

name ~~string~~ operator + ( )

by value

const name & op1,
const name & op2

RVALUE

name & operator = (const name & op2)

Member function

Second operand

Return by Reference

First operand is represented by this pointer

LVALUE

RVALUE:

↑

right

K = i + j;

RVALUE

All BINARY arithmetic operators are RVALUE

LVALUE:

↑

Left

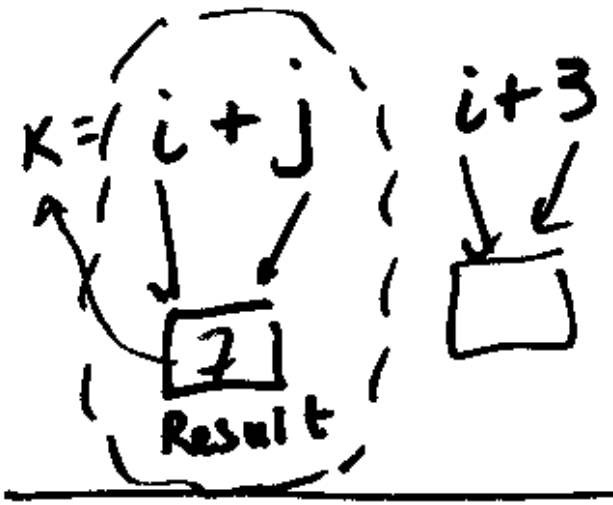
CAN be on left of = op

a[i] = blue;

LVALUE

Built-in

int i, j;



if(i = j)

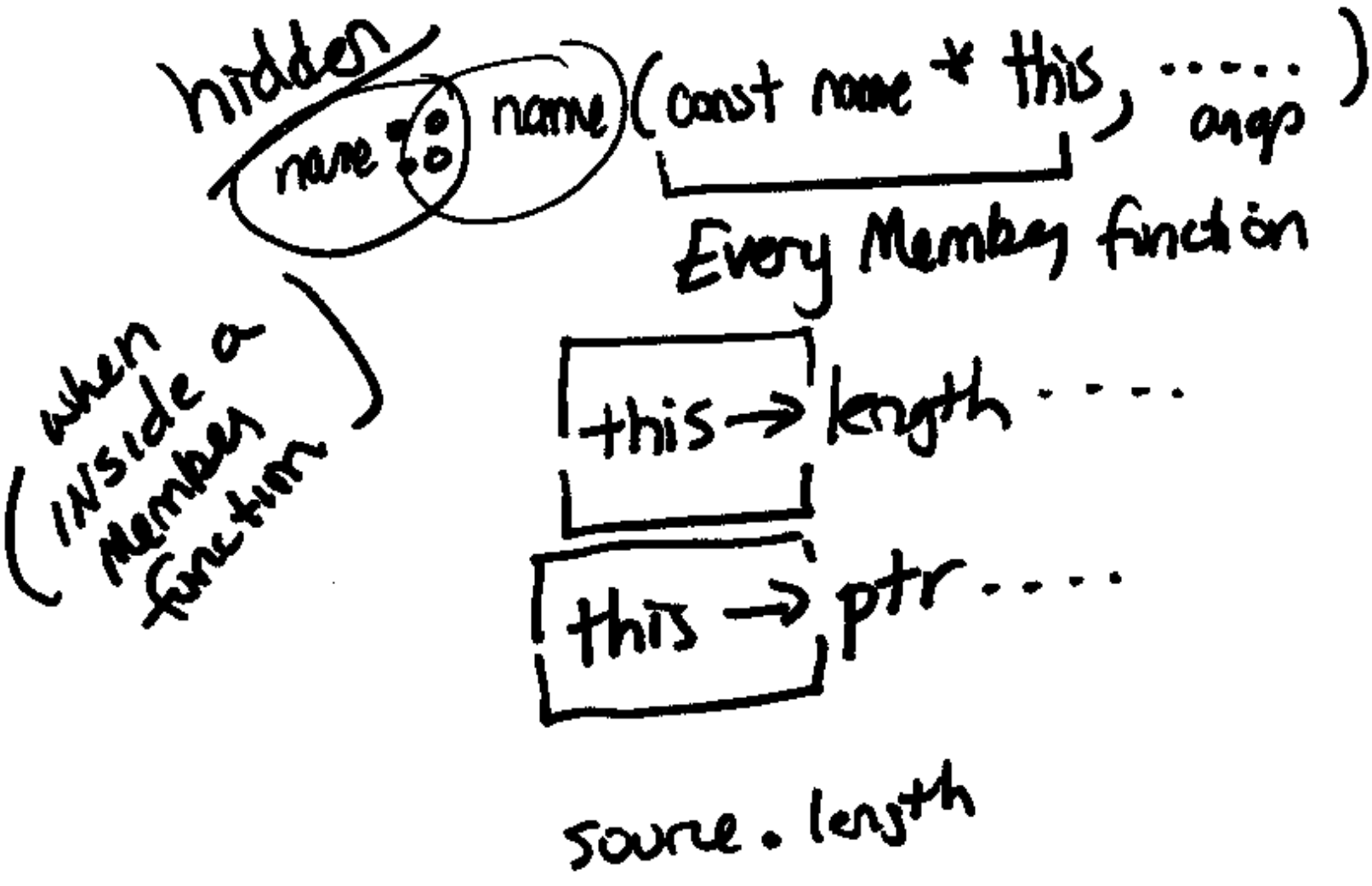
Result

User Defined Data Types

name s1, s2;

s1 = s2

Result


hidden
name
name(const name * this, ...)
args
Every Member function
(when inside a Member function)
this → length ....
this → ptr ....
source.length

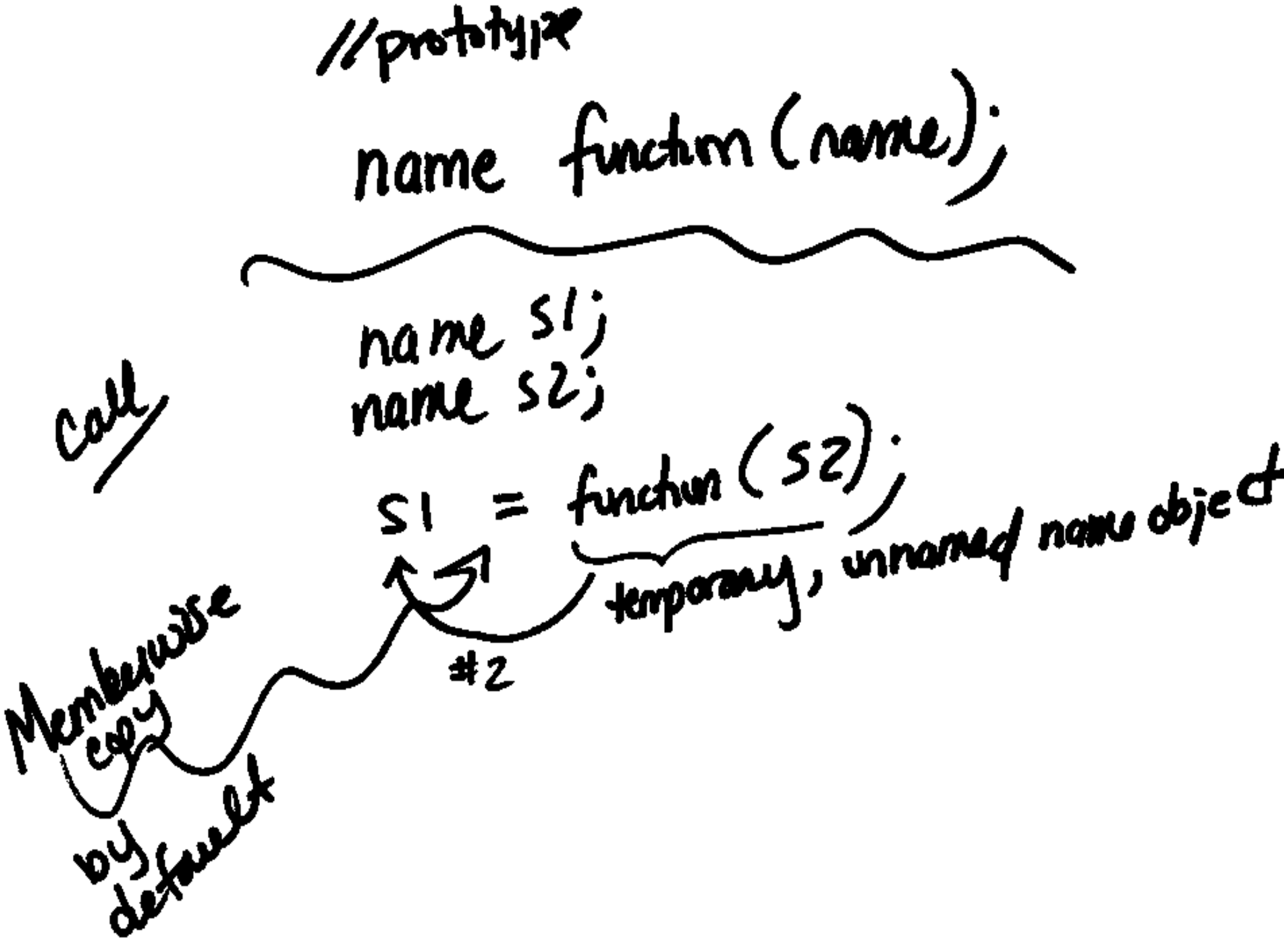
name array[100];

Class name
{
public:
name(); // default
name(const name &); // Copy Constructor

private: char* ptr; int length
};

```
name::name(const name & source)
// copying from argument into current object
{
    length = source.length;
    ptr = new char[length + 1];
    strcpy(ptr, source.ptr);

}
```


//prototype
name functun(name);
name s1;
name s2;
call
s1 = functun(s2);
temporary, unnamed name object
#2
Membewise copy by default

Shallow copy — default (automatic) copy constructor