C++ big 3:

1. Copy constr.
2. destructor
3. assignment operator

// no mem mgmt
dynamic ok

like Constructor but gets an Object (class) as argument

already has mem. allocated

MEMORY MGMT

DEEP COPY

Constructor:

if needed, use new

to allocate mem

destructor!

Use delete to free

memory

⇒ if no new default

destr. ok

assignment op:

operator = (const ... & rhs)

① if (this != &rhs) {

// copy all
from rhs
to this

}

pass by ref

standard alias test

② return *this;

Array class

just an array class, templated

build your own version of STL

vector< >

templates !!!

generic programming

typical code separation

C++ interface:

the .h header file

API, like an advert of what your class does.

C++ implementation

the .cpp file

- Makefiles
  - handles: .cpp files
    - Compile (.cpp → .o)
    - link .o files → a.out
      ~~executable~~
  - dependencies

— main.cpp :

"driver" program

contains main()

```
#include <iostream>
   :
#include "array.h"
```

search locally ./

/usr/include search system headers

- NOTE: (in main.cpp)

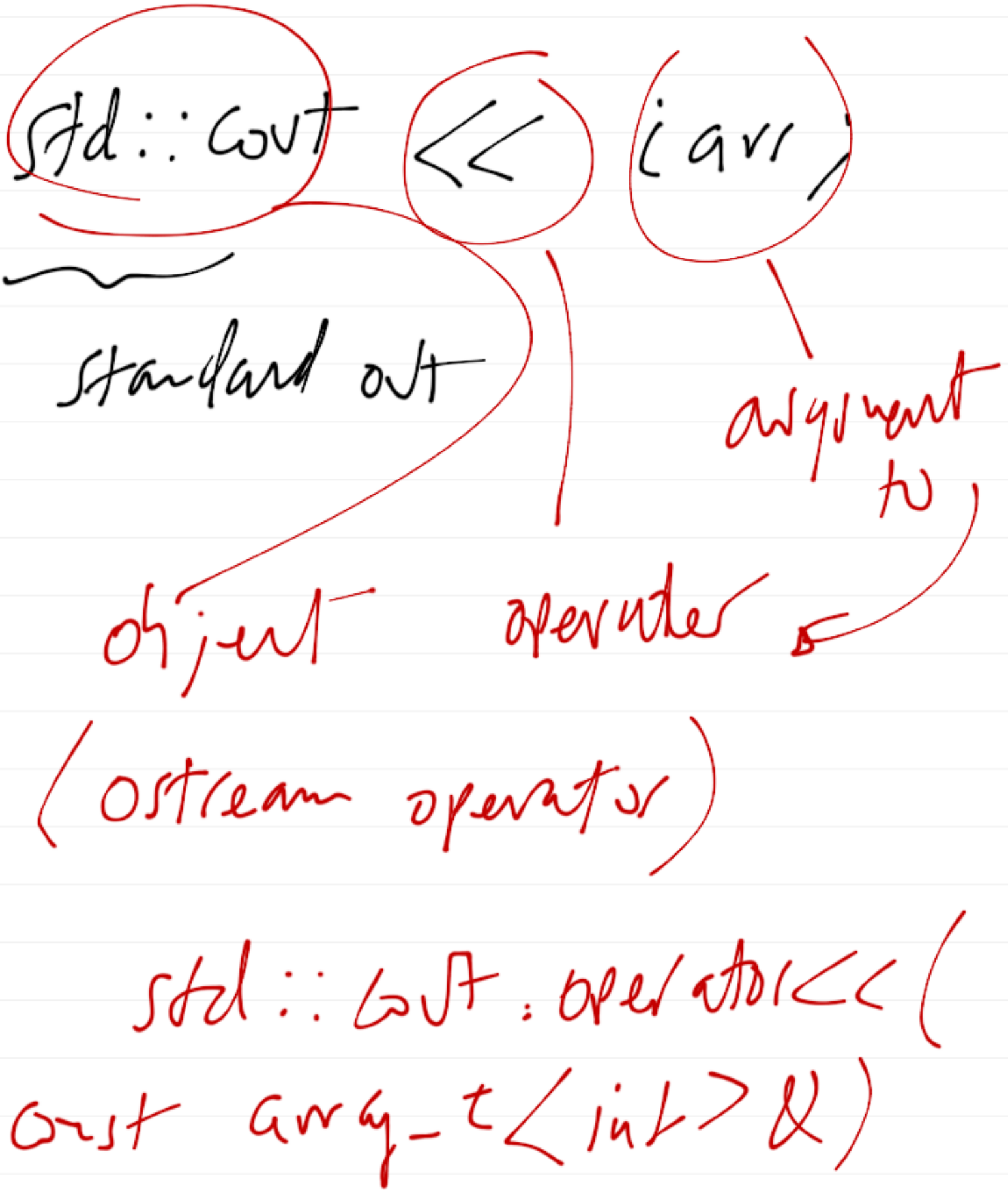
```
#include "pixel.h"
#include "array.h"
```

order matters

Usage of array:

```
int main()
{
    array_t<int> iarr;
    array_t<float> farr;
}
```

generic, templated "container" (see stl)


std::cout
<<
(arr);
standard out
object
operator
argument to
(ostream operator)
std::ostream::operator<<(
const array_t<int>&)

templating (how to):

① forward declaration

on top of .h file:

template <typename T>

class array_t;

generic type

(int, double...)

(2)

```
template <typename T>
class array_t {
public:
    T
    array_t(int *arr,
            int sz);
```

③ specialization

at bottom of .cpp file

template class array_t<int>;

template class array_t<float>;

— Operators : [ ]

in main ()

for(int i = 0; i < iarr.size(); i++)

iarr[i] += 10;

↑

array_t<int>.operator[]

LHS of ⟶

iarr[i] = iarr[i+1]

/

Operator []

Rhs of eg.

Const