C++ 13

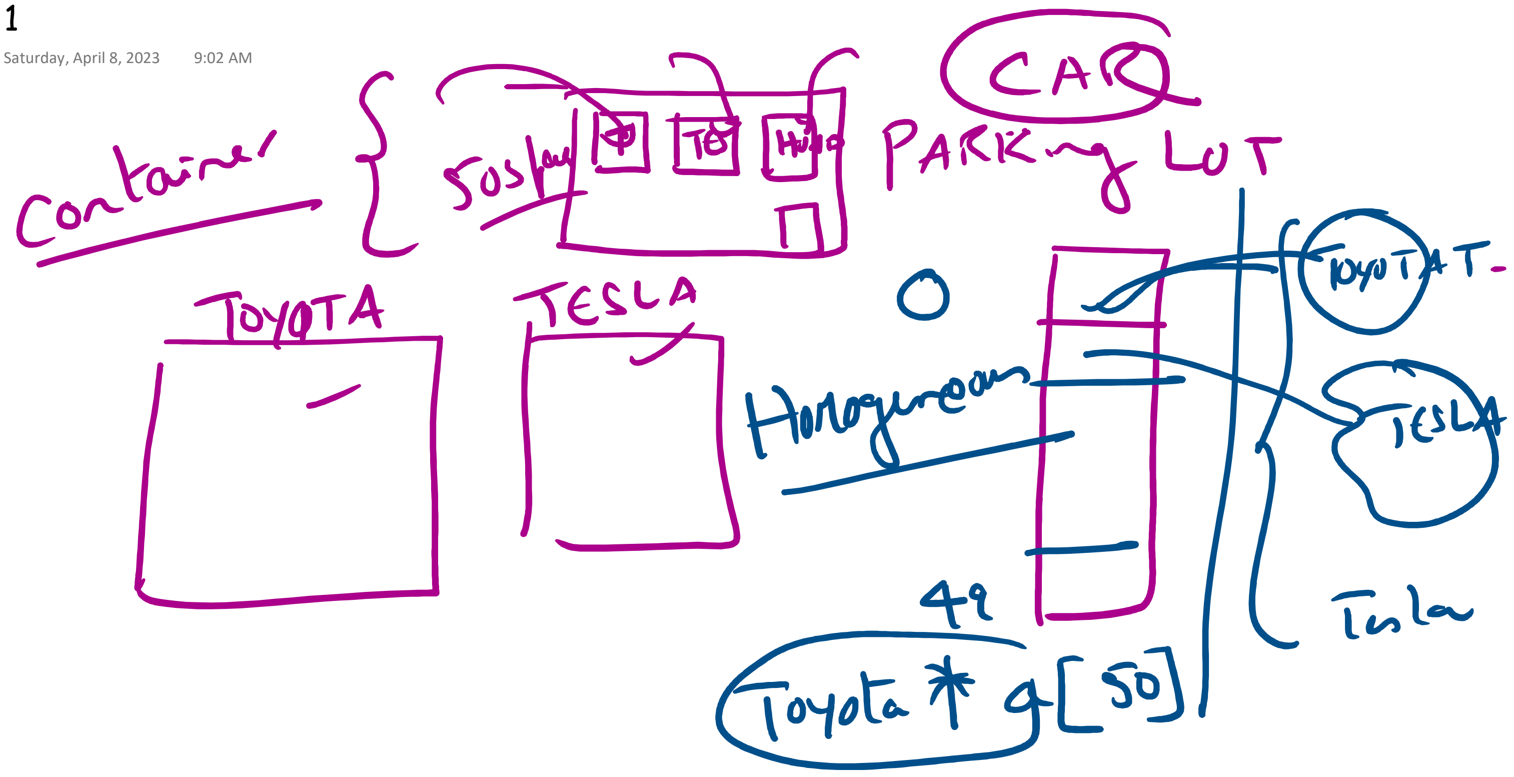
Friday, April 7, 2023 8:06 PM

C++

1. April 8
2. April 15
3. April 22 FINAL EXAM

1

Saturday, April 8, 2023 9:02 AM

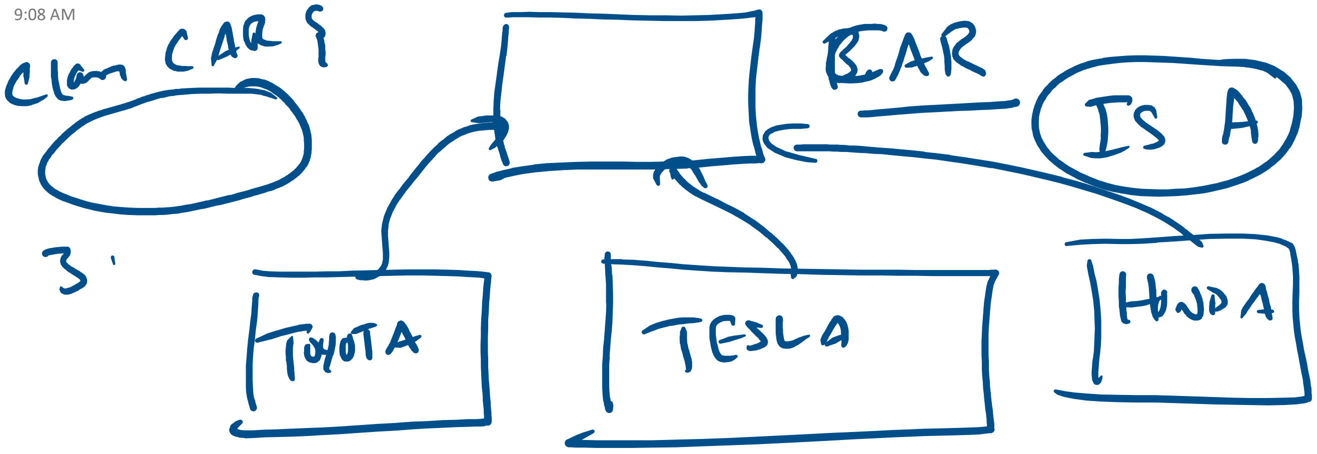


2

Saturday, April 8, 2023 9:08 AM

Clam CAR {

3 '



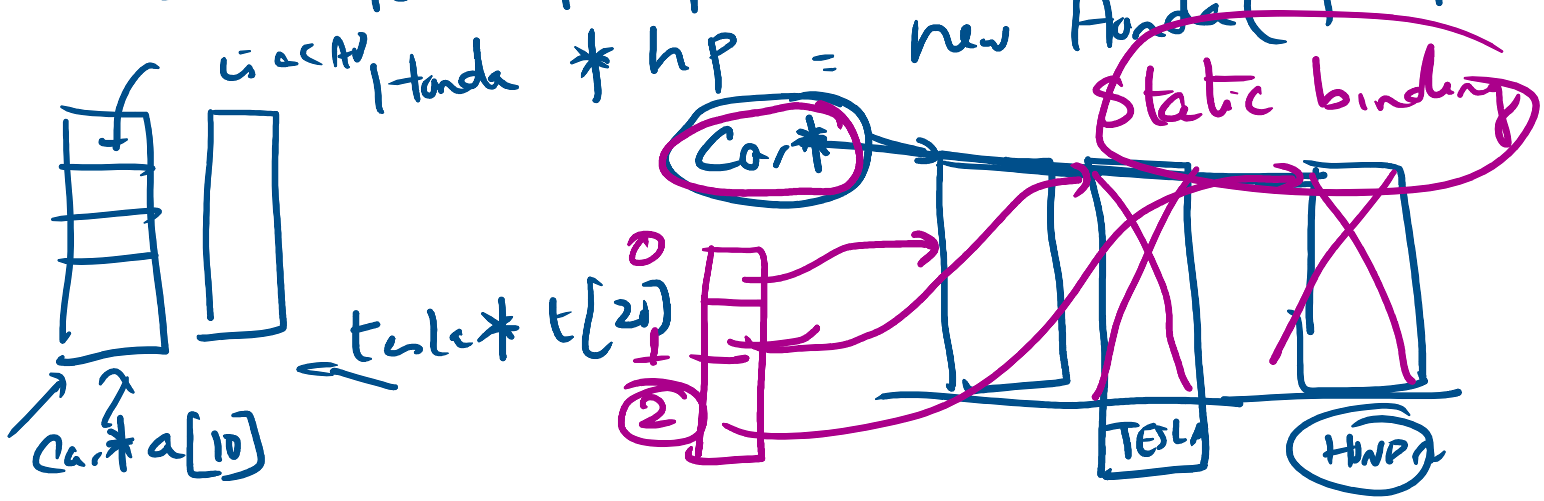
Saturday, April 8, 2023 9:09 AM

Car * cp = new Car( )

@:

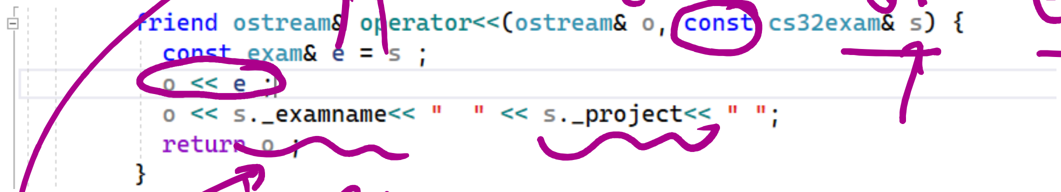
is a CAR Tesla * tp = new Tesla( )

is a CAR Honda * hp = new Honda( )



4

Saturday, April 8, 2023 9:16 AM



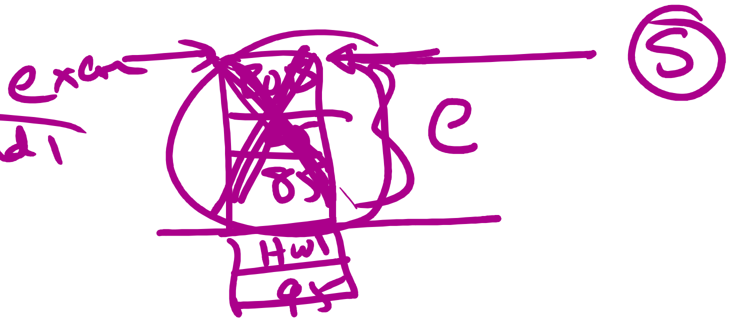
```
friend ostream& operator<<(ostream& o, const cs32exam& s) {
  const exam& e = s ;
  o << e ;
  o << s._examname<< "  " << s._project<< " ";
  return o ;
}
```

e = s ;

o<<e ;

cs32exam c("bob", "HW1", 25, 85, 95);

Cout << c << endl

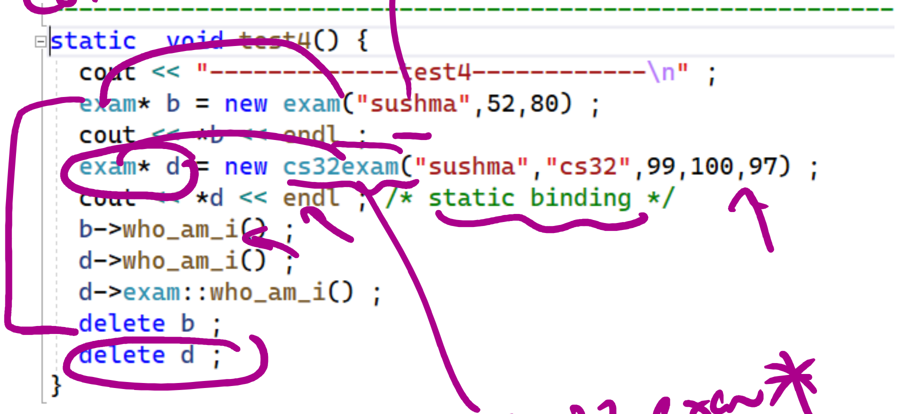


5

Saturday, April 8, 2023 9:20 AM

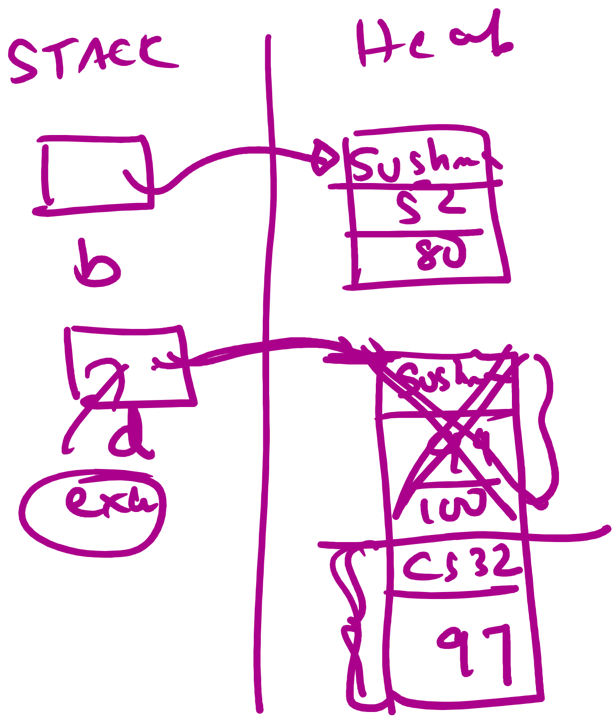
Cgr*

STL



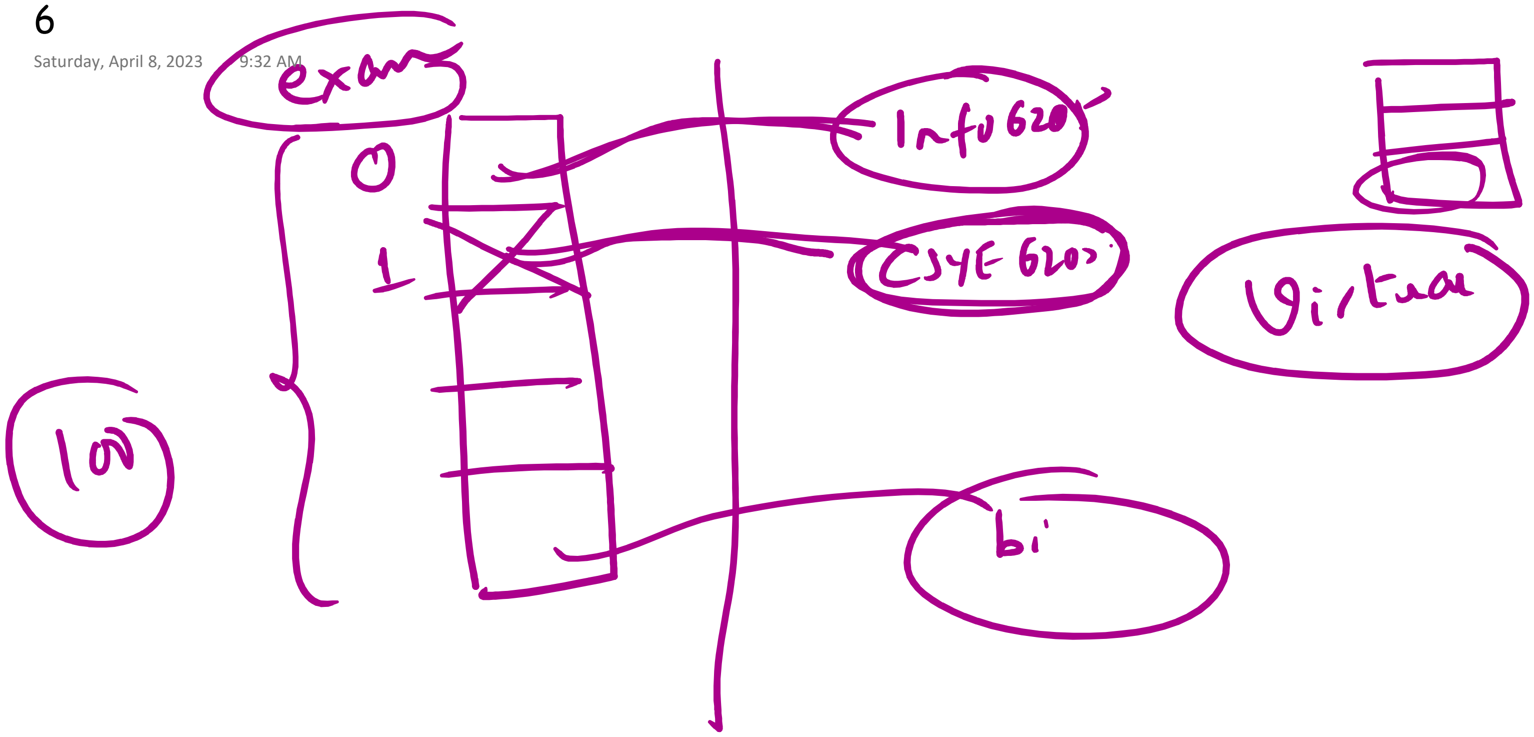
```
static  void test4() {
   cout << "--------------test4------------\n" ;
   exam* b = new exam("sushma",52,80) ;
   cout << *b << endl ;
   exam* d = new cs32exam("sushma","cs32",99,100,97) ;
   cout << *d << endl ; /* static binding */
   b->who_am_i() ;
   d->who_am_i() ;
   d->exam::who_am_i() ;
   delete b ;
   delete d ;
}
```

CS32 exam



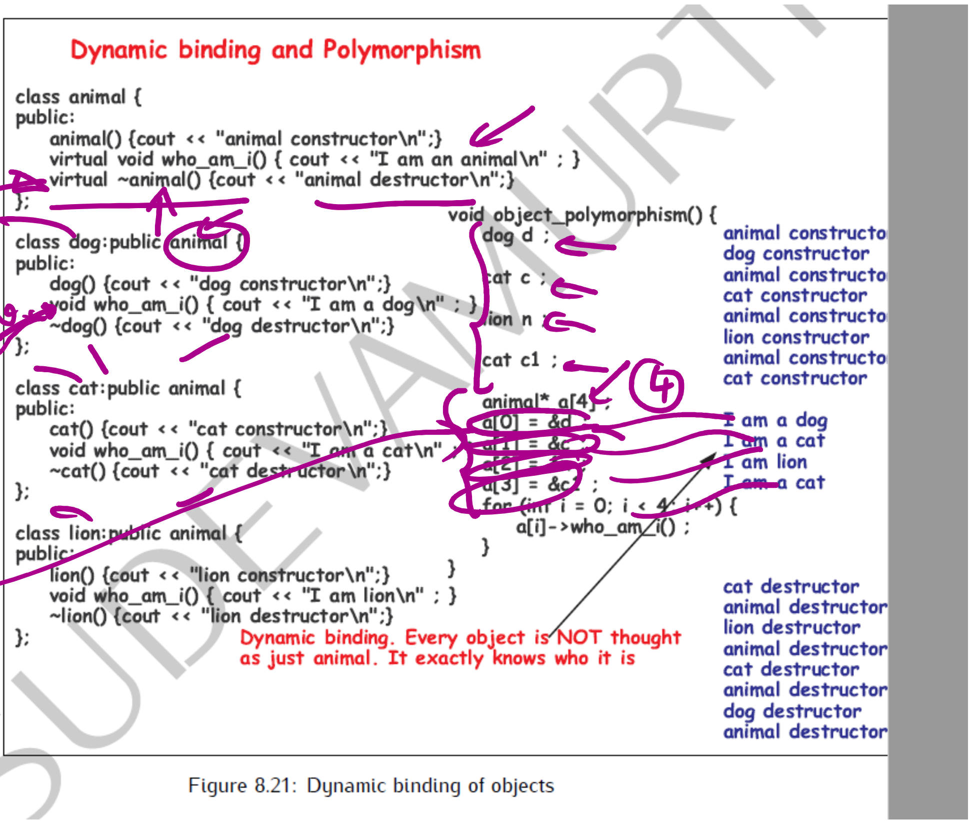
6

Saturday, April 8, 2023 9:32 AM



7

Saturday, April 8, 2023 9:34 AM



Dynamic binding and Polymorphism

```
class animal {
public:
    animal() {cout << "animal constructor\n";}
    virtual void who_am_i() { cout << "I am an animal\n" ; }
    virtual ~animal() {cout << "animal destructor\n";}
};

class dog:public animal {
public:
    dog() {cout << "dog constructor\n";}
    void who_am_i() { cout << "I am a dog\n" ; }
    ~dog() {cout << "dog destructor\n";}
};

class cat:public animal {
public:
    cat() {cout << "cat constructor\n";}
    void who_am_i() { cout << "I am a cat\n" ; }
    ~cat() {cout << "cat destructor\n";}
};

class lion:public animal {
public:
    lion() {cout << "lion constructor\n";}
    void who_am_i() { cout << "I am lion\n" ; }
    ~lion() {cout << "lion destructor\n";}
};
```

```
void object_polymorphism() {
    dog d ;
    cat c ;
    lion n ;
    cat c1 ;
    animal* a[4] ;
    a[0] = &d ;
    a[1] = &c ;
    a[2] = &n ;
    a[3] = &c1 ;
    for (int i = 0; i < 4; i++) {
        a[i]->who_am_i() ;
    }
}
```

```
animal constructo
dog constructor
animal constructo
cat constructor
animal constructo
lion constructor
animal constructo
cat constructor
I am a dog
I am a cat
I am lion
I am a cat

cat destructor
animal destructor
lion destructor
animal destructor
cat destructor
animal destructor
dog destructor
animal destructor
```

Dynamic binding. Every object is NOT thought as just animal. It exactly knows who it is

Figure 8.21: Dynamic binding of objects

SUDEVAMURT

Virtual

Virtual

4

Cat

Class xx {

Virtual

}

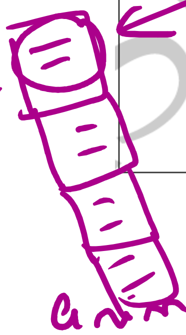
xxx(- - -)

xxx(- - -

PYTHON STL Ptn Ptn

Chork

Animal a[4]



Note that both base and derived destructors are called
because base destructor is virtual
virtual ~animal() {cout << "animal destructor\n";}

DREAM animal* a[4] which stores dog, cats and lion
Hetrogenous container

Figure 8.22: Dynamic binding of pointers to objects

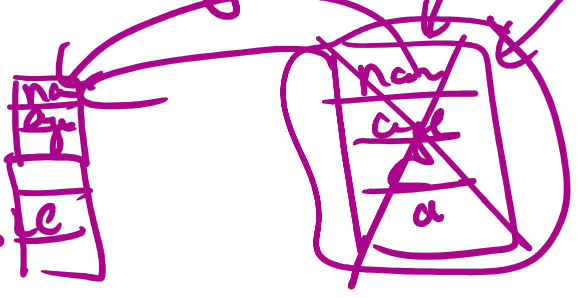
animal a[4]

dog d("nm", a, b, c, d)

a[0] = d;

a[0] = d

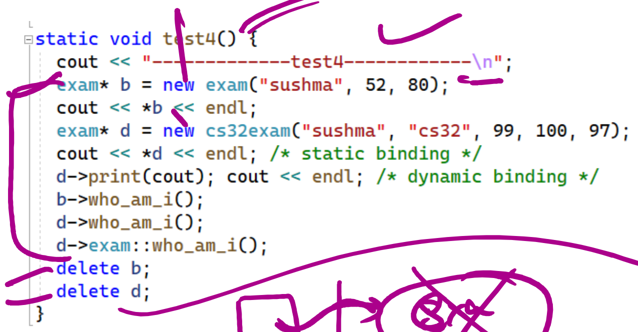
cs3



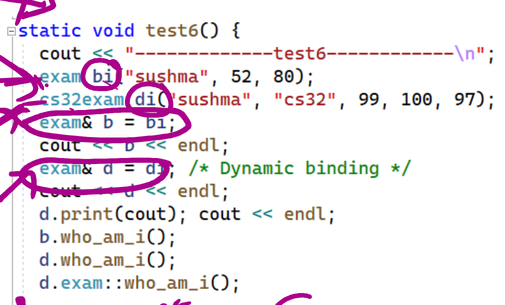
& &

9

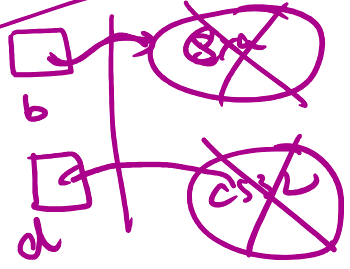
Saturday, April 8, 2023 9:47 AM

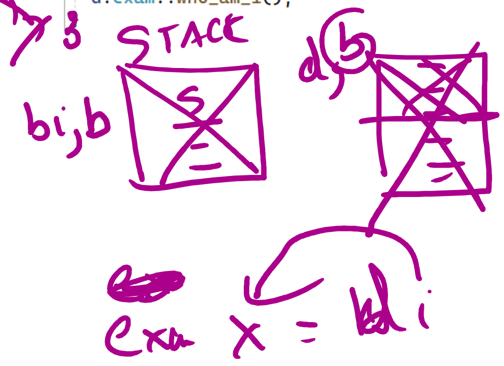


```
static void test4() {
    cout << "--------------test4-------------\n";
    exam* b = new exam("sushma", 52, 80);
    cout << *b << endl;
    exam* d = new cs32exam("sushma", "cs32", 99, 100, 97);
    cout << *d << endl; /* static binding */
    d->print(cout); cout << endl; /* dynamic binding */
    b->who_am_i();
    d->who_am_i();
    d->exam::who_am_i();
    delete b;
    delete d;
}
```



```
static void test6() {
    cout << "--------------test6-------------\n";
    exam bi("sushma", 52, 80);
    cs32exam di("sushma", "cs32", 99, 100, 97);
    exam& b = bi;
    cout << b << endl;
    exam& d = di; /* Dynamic binding */
    cout << d << endl;
    d.print(cout); cout << endl;
    b.who_am_i();
    d.who_am_i();
    d.exam::who_am_i();
```





10

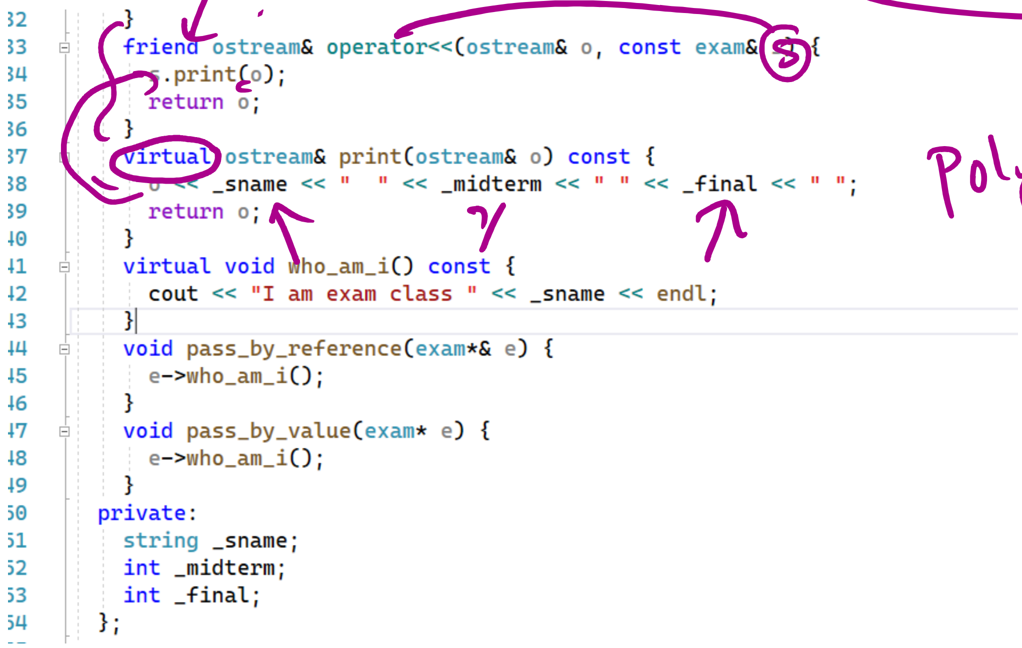
Saturday, April 8, 2023 9:52 AM

Op

cout << c

fri

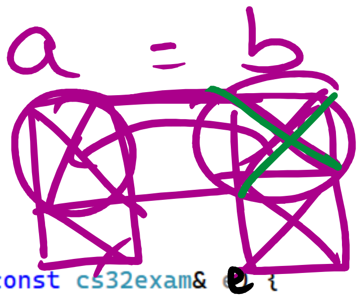
Poly

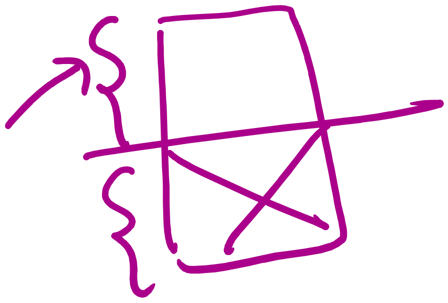


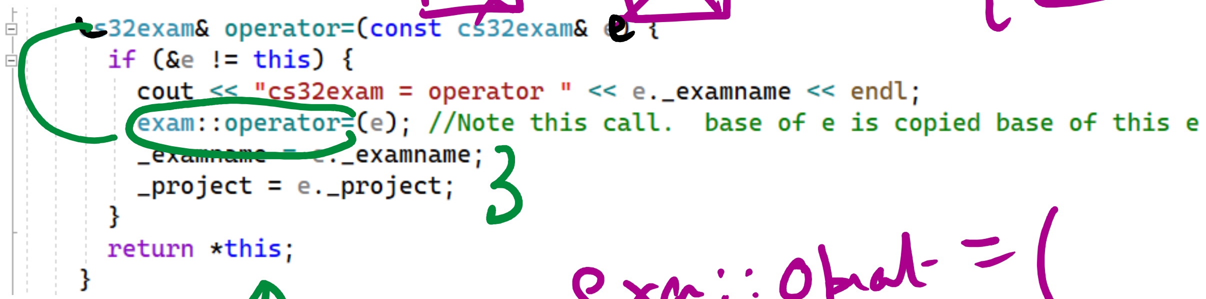
```
    }
    friend ostream& operator<<(ostream& o, const exam& s) {
      s.print(o);
      return o;
    }
    virtual ostream& print(ostream& o) const {
      o << _sname << "  " << _midterm << " " << _final << " ";
      return o;
    }
    virtual void who_am_i() const {
      cout << "I am exam class " << _sname << endl;
    }
    void pass_by_reference(exam*& e) {
      e->who_am_i();
    }
    void pass_by_value(exam* e) {
      e->who_am_i();
    }
  private:
    string _sname;
    int _midterm;
    int _final;
  };
```

11

Saturday, April 8, 2023 10:00 AM







```
cs32exam& operator=(const cs32exam& e) {
    if (&e != this) {
        cout << "cs32exam = operator " << e._examname << endl;
        exam::operator=(e); //Note this call.  base of e is copied base of this e
        _examname = e._examname;
        _project = e._project;
    }
    return *this;
}
```

3

exa::opeat=(

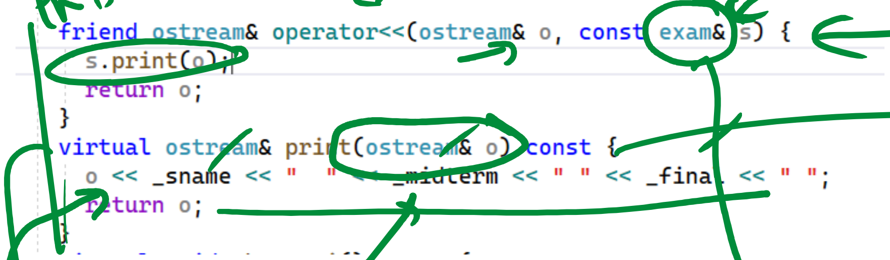
12

Saturday, April 8, 2023 10:03 AM

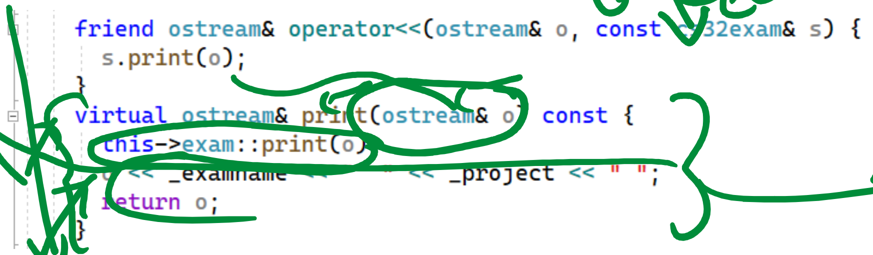
this

$\boxed{<<}$

Virtue



```
friend ostream& operator<<(ostream& o, const exam& s) {
  s.print(o);
  return o;
}
virtual ostream& print(ostream& o) const {
  o << _sname << " " << _midterm << " " << _final << " ";
  return o;
}
```



```
friend ostream& operator<<(ostream& o, const [illegible]32exam& s) {
  s.print(o);
}
virtual ostream& print(ostream& o) const {
  this->exam::print(o);
  o << _examname << " " << _project << " ";
  return o;
}
```

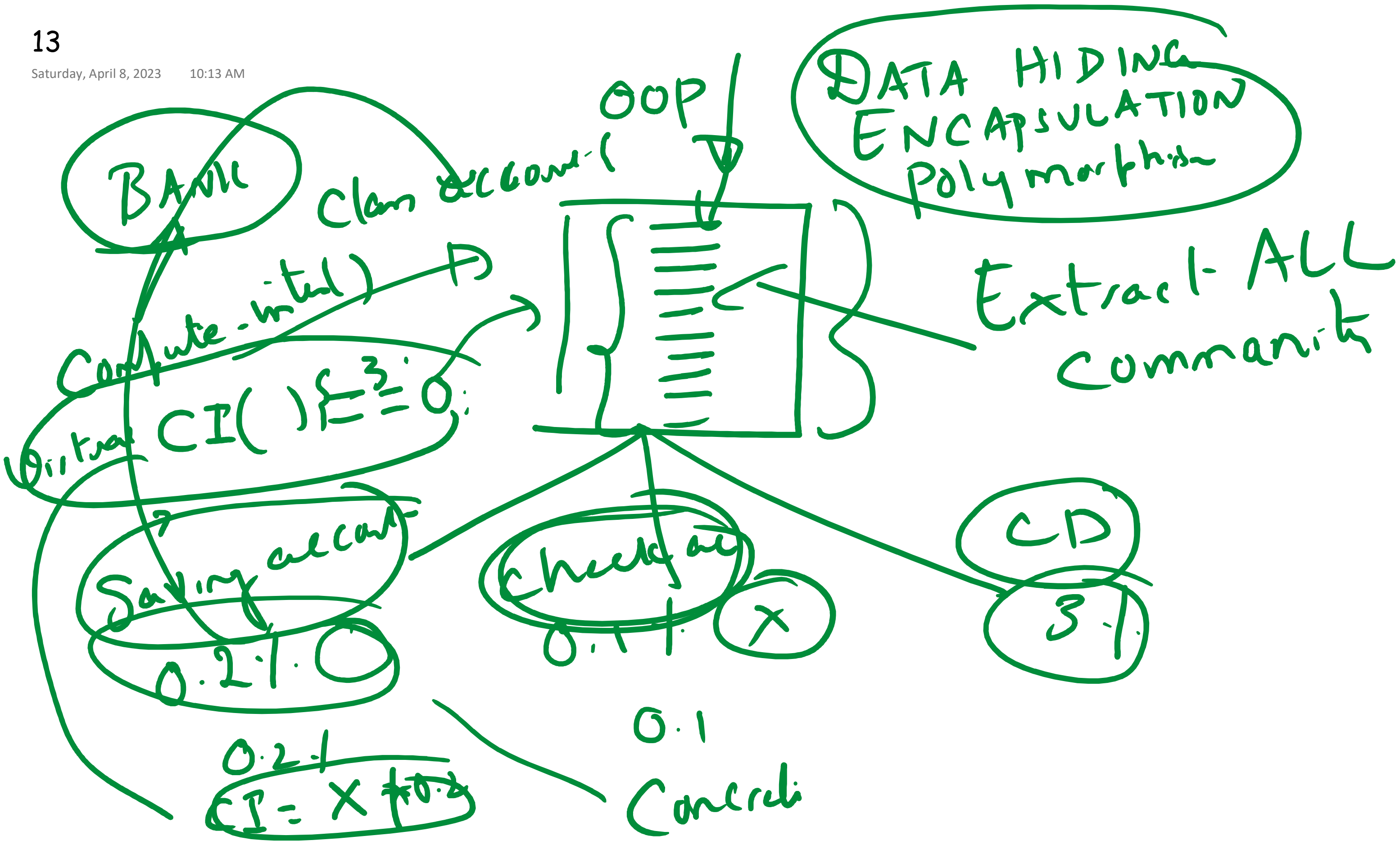
exam

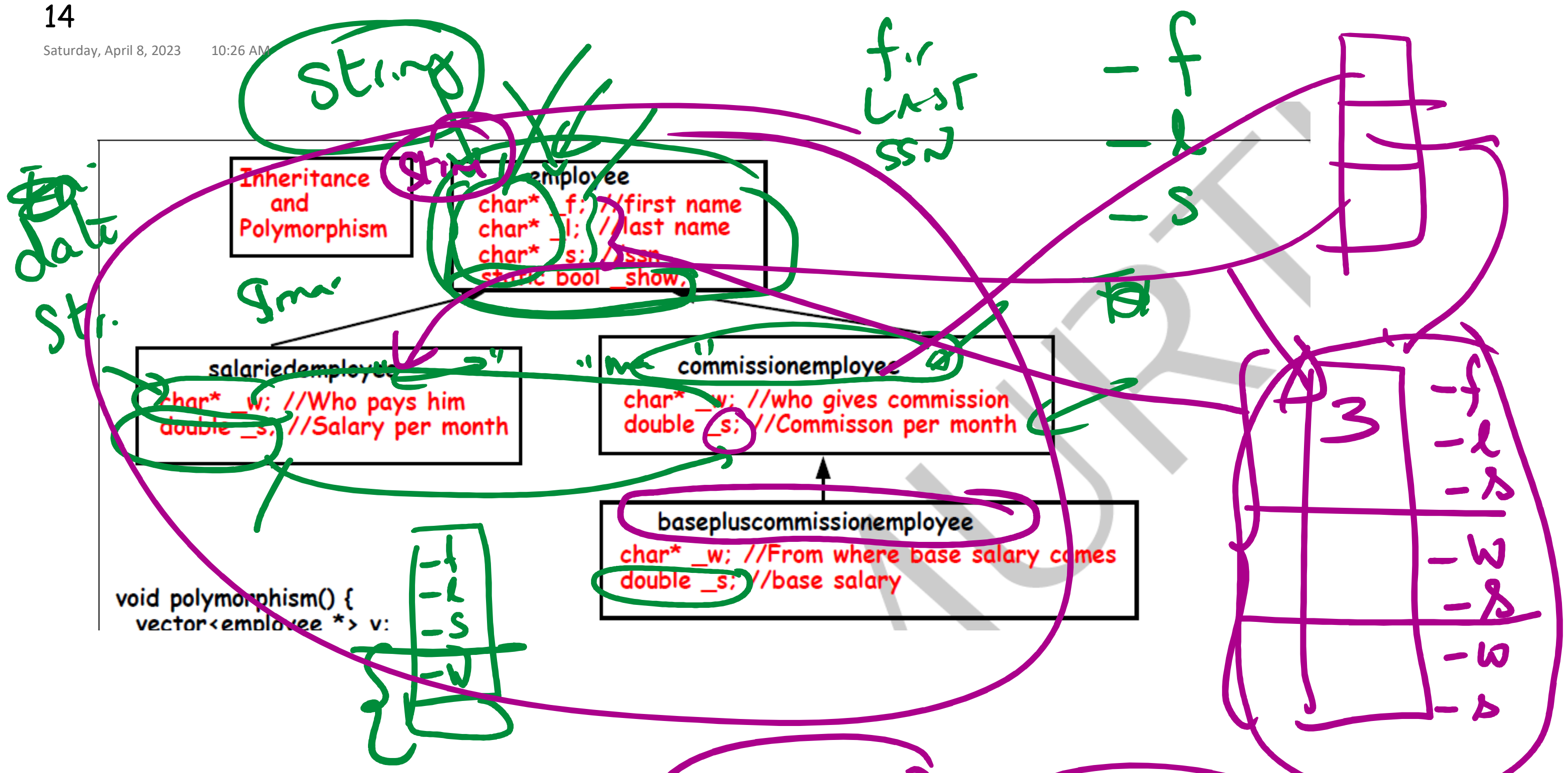
cout << $\bigcirc$

$\boxed{exam}$

13

Saturday, April 8, 2023 10:13 AM





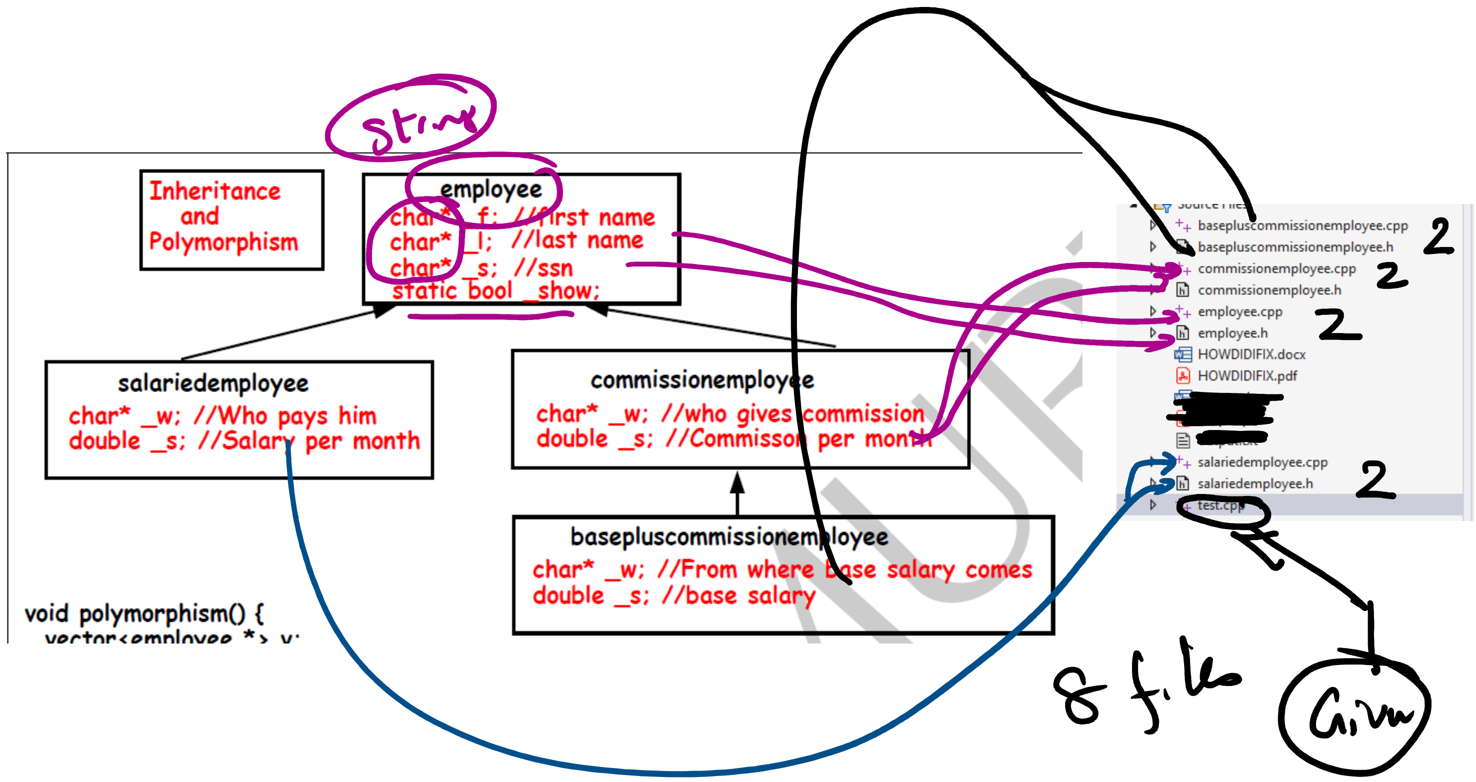
```
void polymorphism() {
  vector<employee *> v;
```

10:50

Record

15

Saturday, April 8, 2023 10:50 AM



Inheritance and Polymorphism

```
employee
char* _f; //first name
char* _l; //last name
char* _s; //ssn
static bool _show;
```

```
salariedemployee
char* _w; //Who pays him
double _s; //Salary per month
```

```
commissionemployee
char* _w; //who gives commission
double _s; //Commisson per month
```

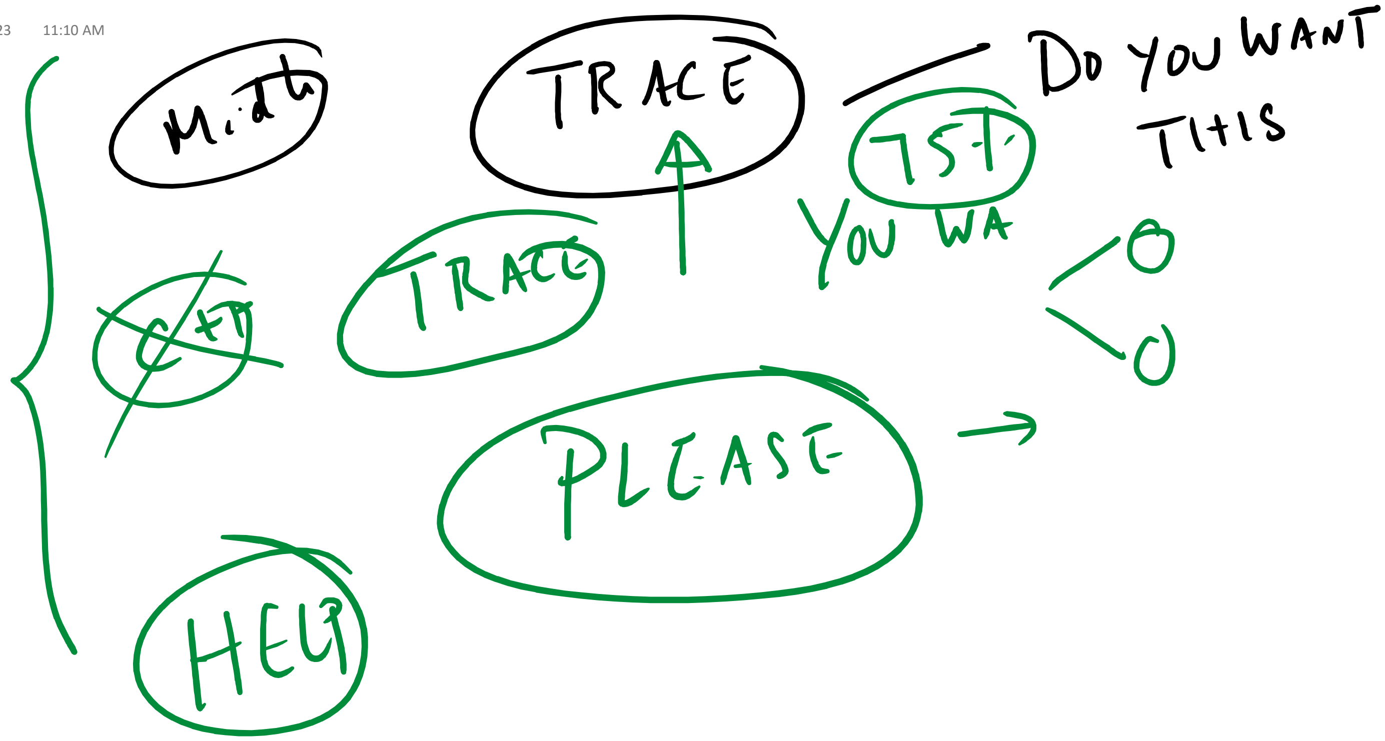
```
baseplusscommissionemployee
char* _w; //From where base salary comes
double _s; //base salary
```

```
void polymorphism() {
```

baseplusscommissionemployee.cpp
baseplusscommissionemployee.h
commissionemployee.cpp
commissionemployee.h
employee.cpp
employee.h
HOWDIDIFIX.docx
HOWDIDIFIX.pdf
[redacted]
salariedemployee.cpp
salariedemployee.h
test.cpp

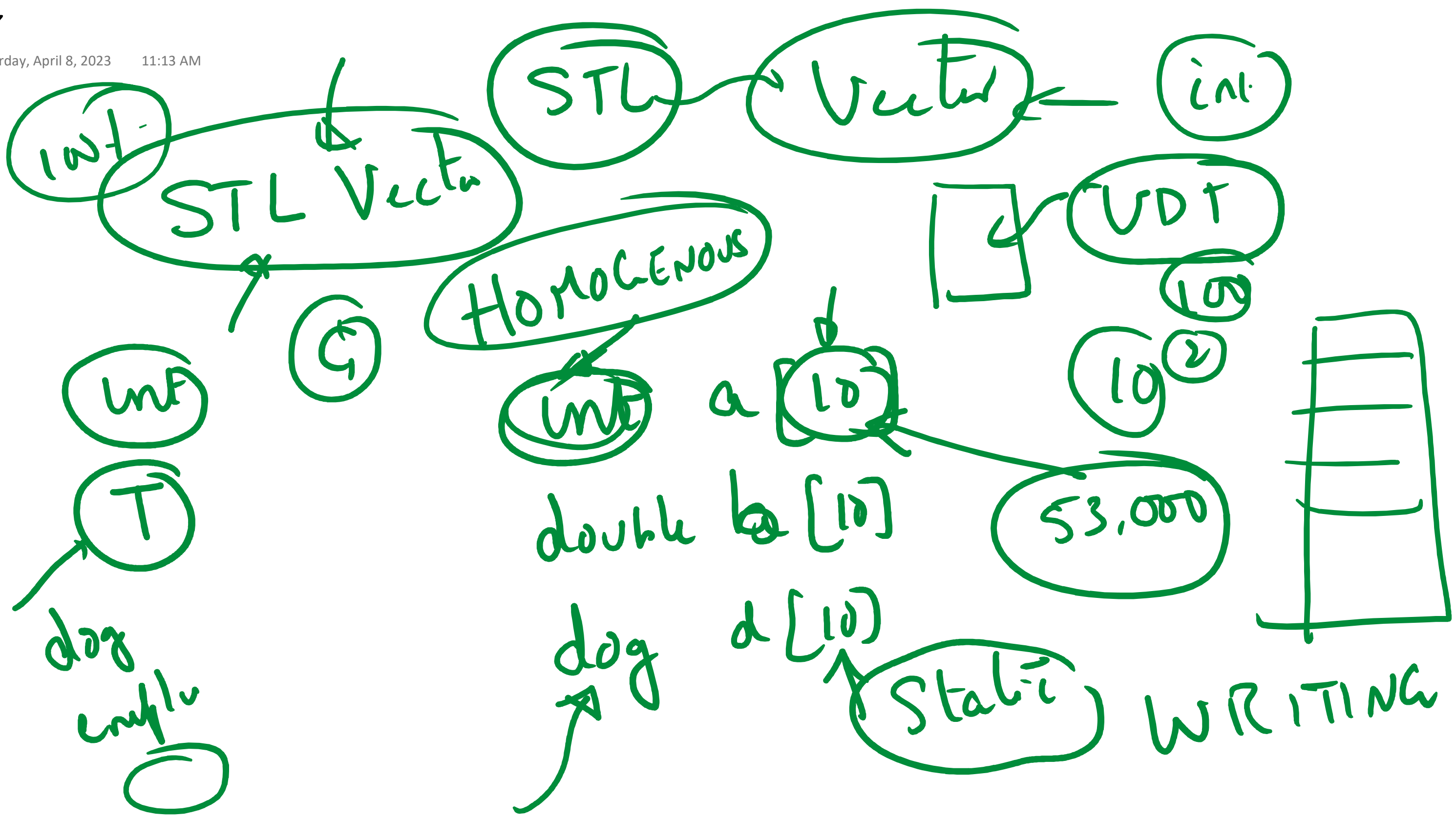
16

Saturday, April 8, 2023 11:10 AM



17

Saturday, April 8, 2023 11:13 AM



18

Saturday, April 8, 2023 11:16 AM

T a[10]

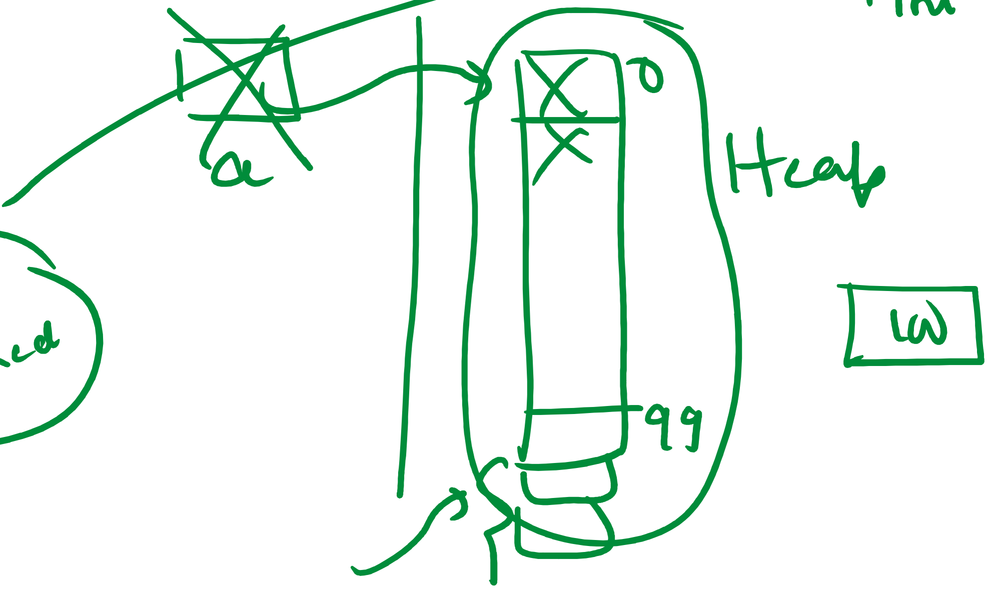
STACK

$$\texttt{int * a = new int}[\overset{100}{\underset{1000}{10}}]$$

1 int

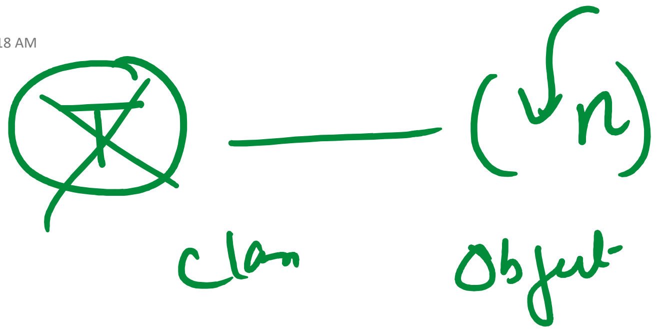
155

Dynamic + Fixed

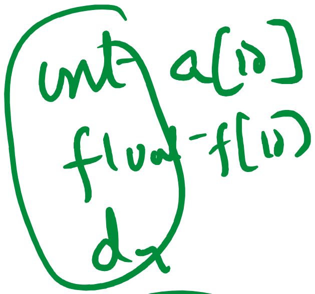


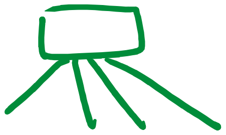
19

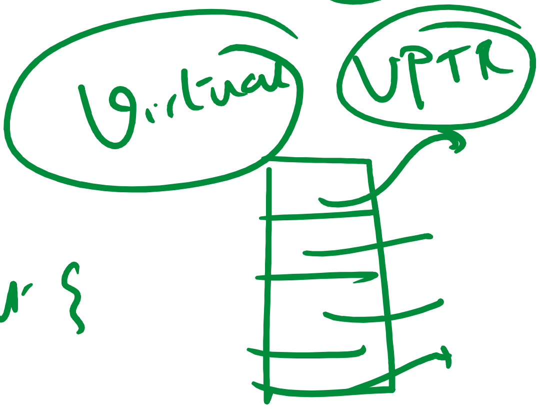
Saturday, April 8, 2023 11:18 AM

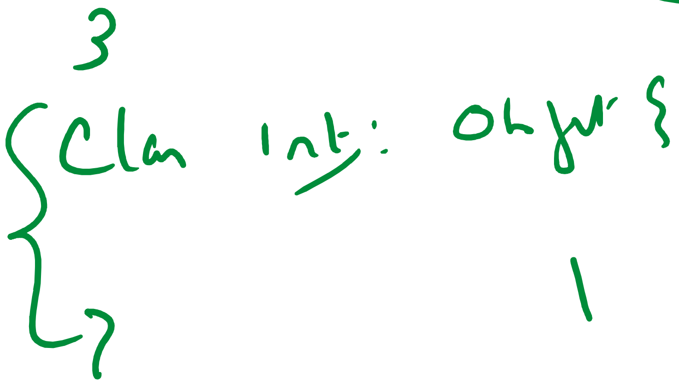


S



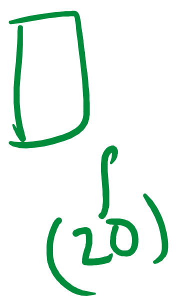






20

Saturday, April 8, 2023 11:20 AM



```
Vector < int > a ;
Vecto < dog > b ;
Vecto < int * > c ;
Vecto < dog * > d ;
```

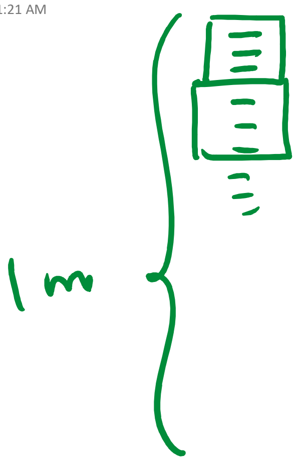
T

STL

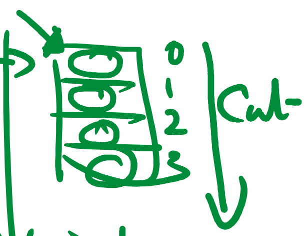
emplye[5]

21

Saturday, April 8, 2023 11:21 AM



int. $a[5]$
decs



$a[i]$ in $\Theta(1)$ tm

$a[5] = \boxed{a + 5}$

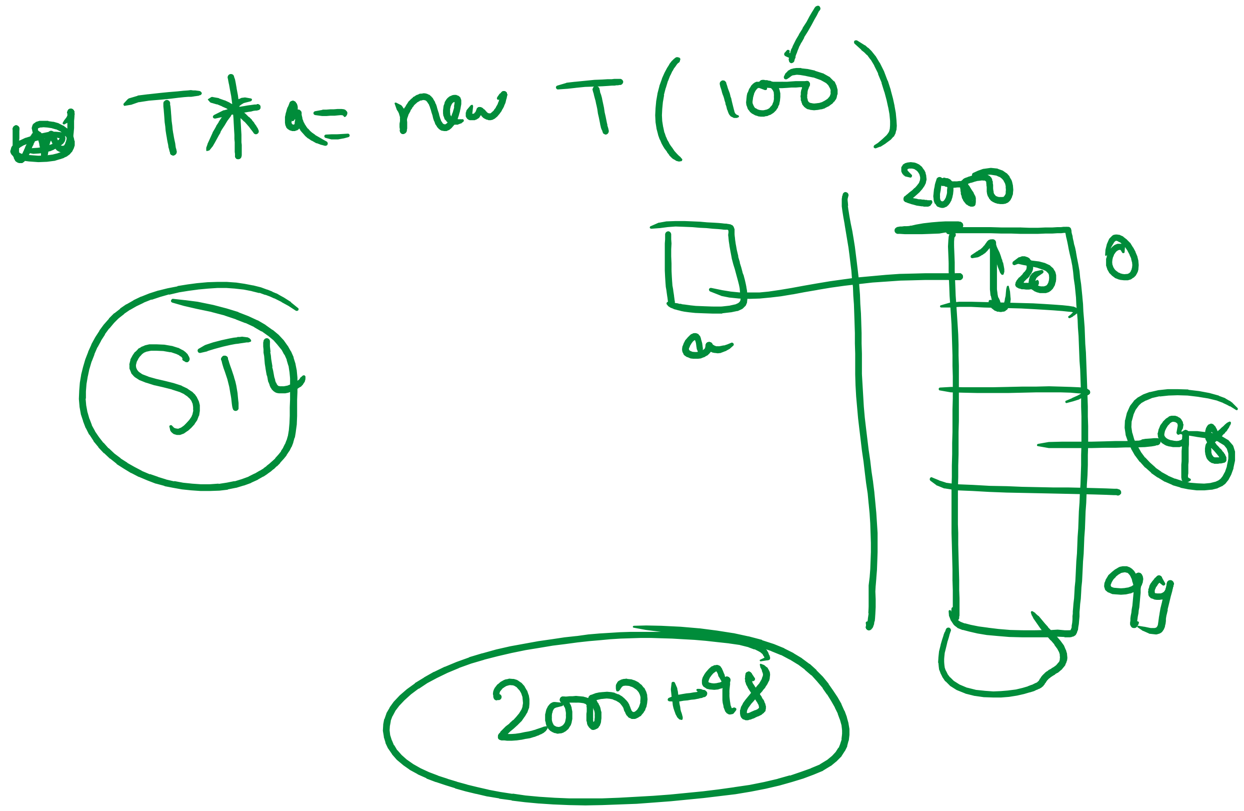
$a[i] = a + i$

$\nearrow$ $\Theta(1)$

22

Saturday, April 8, 2023 11:23 AM

T* a = new T(100)



Saturday, April 8, 2023 11:24 AM

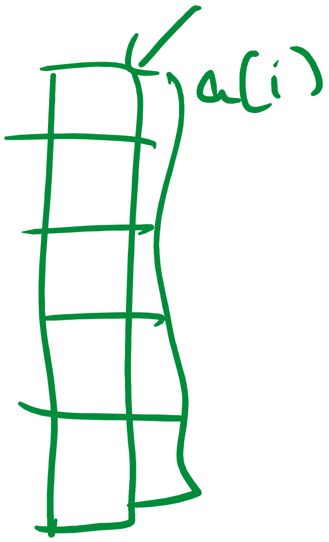
LIST

Vector
Arra
List

[ ] TABLE DOUBLING

Dynamic Arra

$a[i]$ in $\Theta(1)$



Vector<int> a;

$\Theta(1)$

$\rightarrow \boxed{4}$

a.push_back(100)

200

88

77

$\boxed{200}$ $\boxed{30}$

n

$\boxed{2}$

4
8

8
9

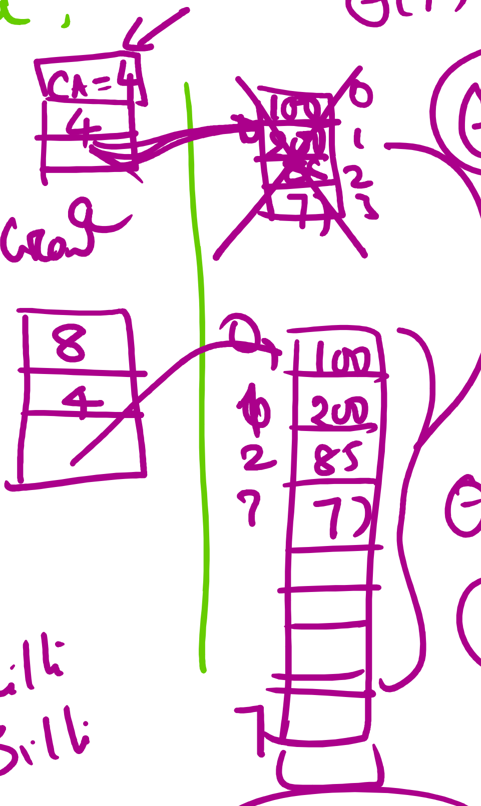
4
8
16
32
64
128
256

go crazy

1 milli
1 Billi

CAP

4 scal-car
~~8~~ 18 sec

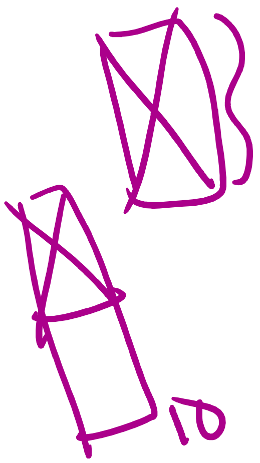


$\boxed{\Theta(n)}$

$\Theta(1)$

$\boxed{8}$

$\boxed{1}$



Grow <<< $\boxed{a[i]}$

$\boxed{\Theta(1)}$

$\boxed{1.5}$

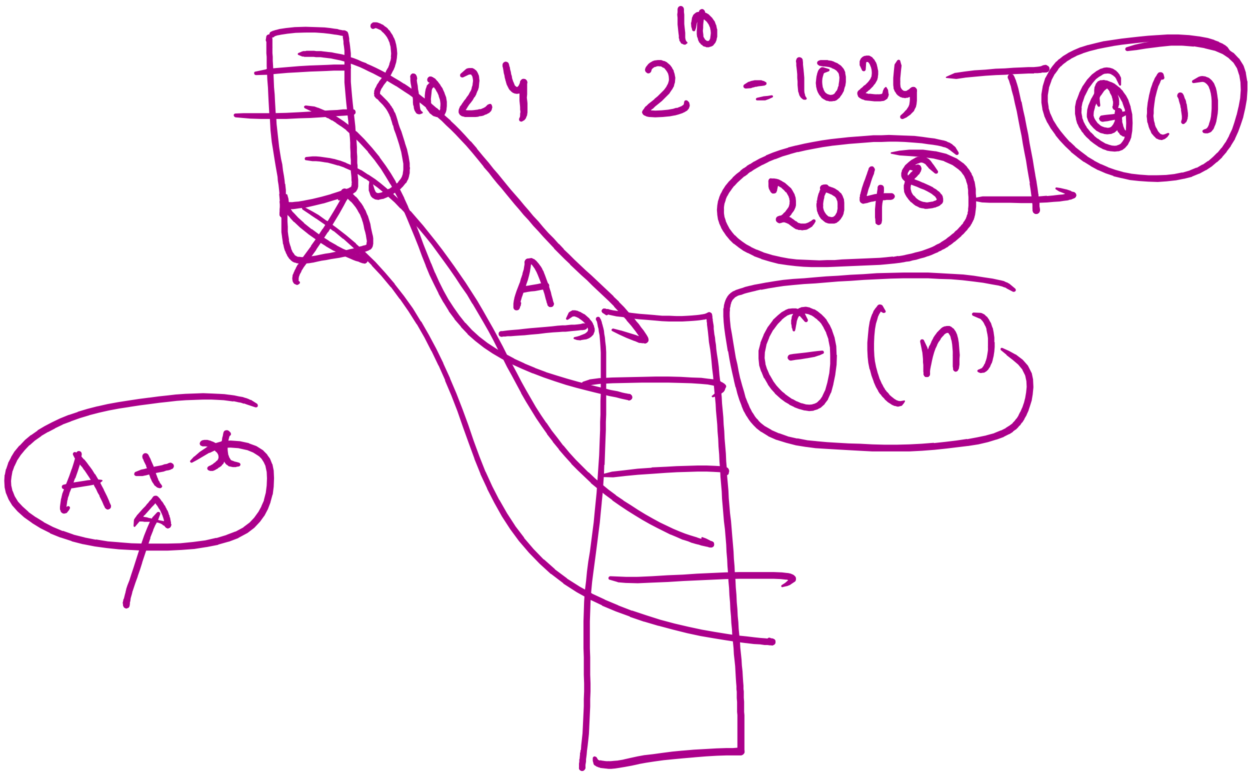
Amortized O(1)

$\Theta(1)$

1 milli

25

Saturday, April 8, 2023 11:32 AM



Saturday, April 8, 2023 11:35 AM

int a[10]

Mage

a[5] = 100

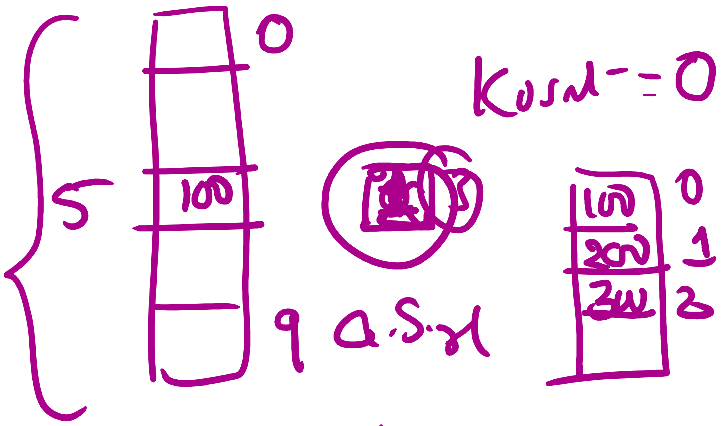
Mage

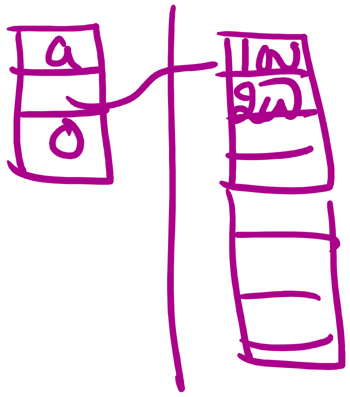
Vect<int> a;

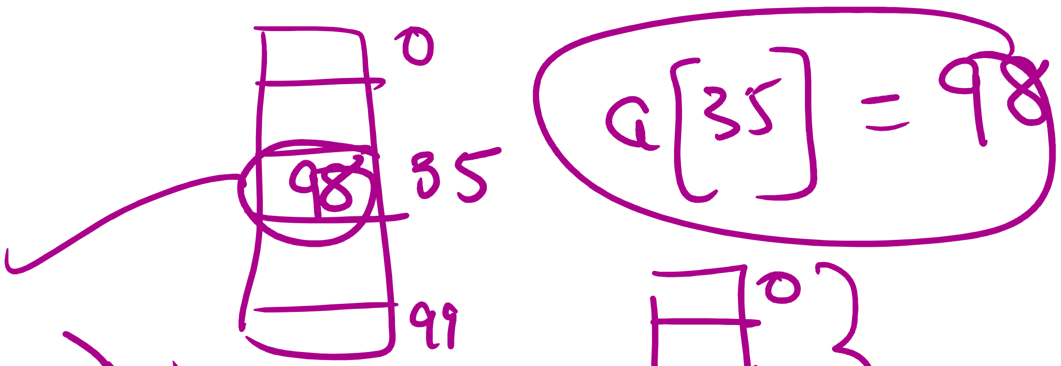
a(~~5~~, 100

a.push_back(100)

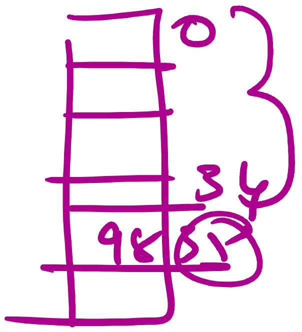
a.size()



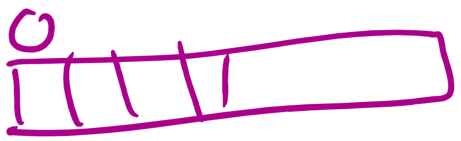




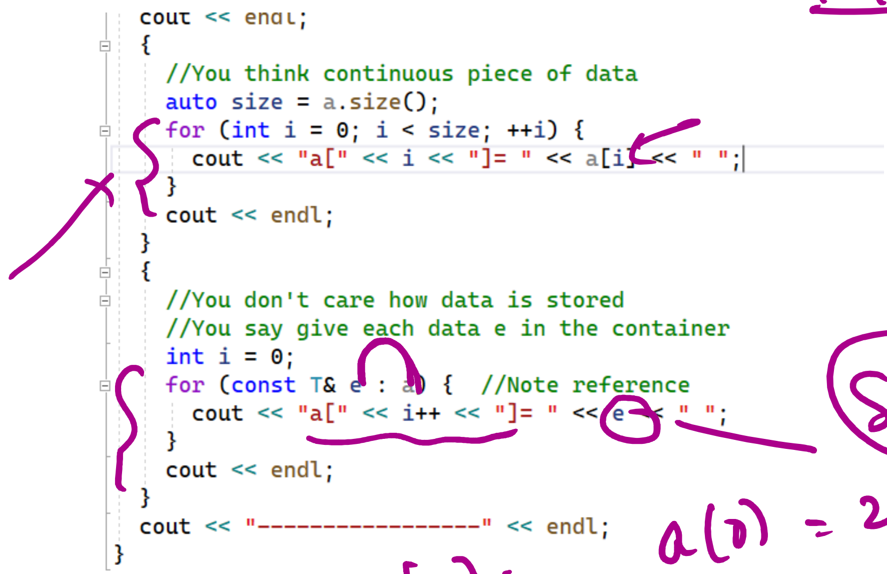
a.Pushbe(98)



Saturday, April 8, 2023 11:50 AM



$i = 0$
1
2



```
    cout << endl;
    {
      //You think continuous piece of data
      auto size = a.size();
      for (int i = 0; i < size; ++i) {
        cout << "a[" << i << "]= " << a[i] << " ";
      }
      cout << endl;
    }
    {
      //You don't care how data is stored
      //You say give each data e in the container
      int i = 0;
      for (const T& e : a) {  //Note reference
        cout << "a[" << i++ << "]= " << e << " ";
      }
      cout << endl;
    }
    cout << "-----------------" << endl;
}
```

e : a

a(0) = 28

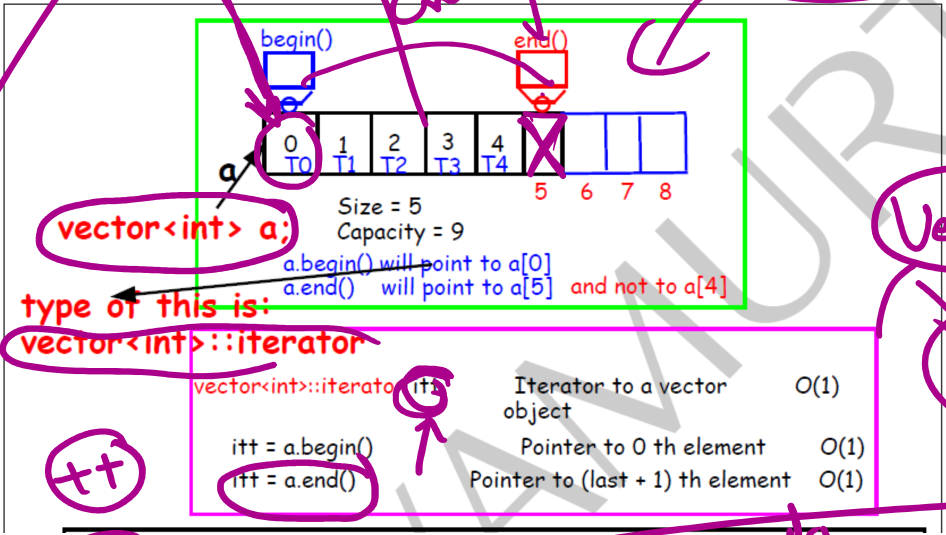
a[0]:

Saturday, April 8, 2023 11:55 AM

a.beg

NOT in end

Vector



vector<int> a;

Size = 5
Capacity = 9

a.begin() will point to a[0]
a.end() will point to a[5] and not to a[4]

type of this is:
vector<int>::iterator

vector<int>::iterator itt	Iterator to a vector object	O(1)
itt = a.begin()	Pointer to 0 th element	O(1)
itt = a.end()	Pointer to (last + 1) th element	O(1)

dog

Vector<int> a;

++

--

++

--

++

--

++

--

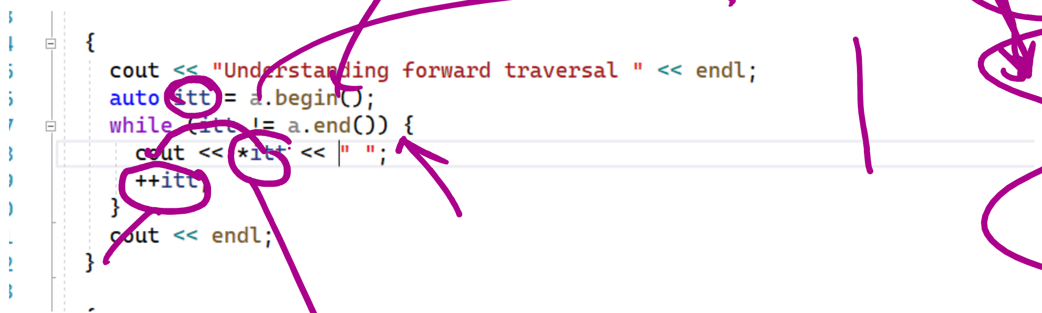
dog

Vector<int>::iterator

AMURTY

29

Saturday, April 8, 2023 12:00 PM



```
{
  cout << "Understanding forward traversal " << endl;
  auto itt = a.begin();
  while (itt != a.end()) {
    cout << *itt << " ";
    ++itt;
  }
  cout << endl;
}
```

Pre

++itt

~~itt++~~

++