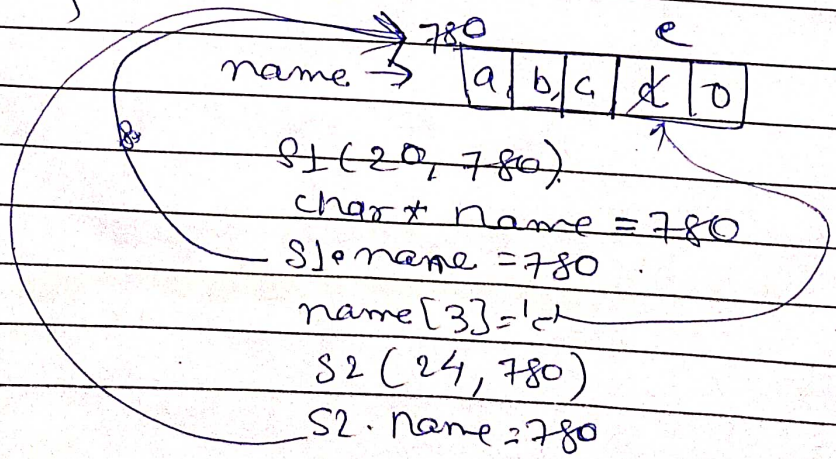
OOPS-2

Date
Page

Shallow-copy

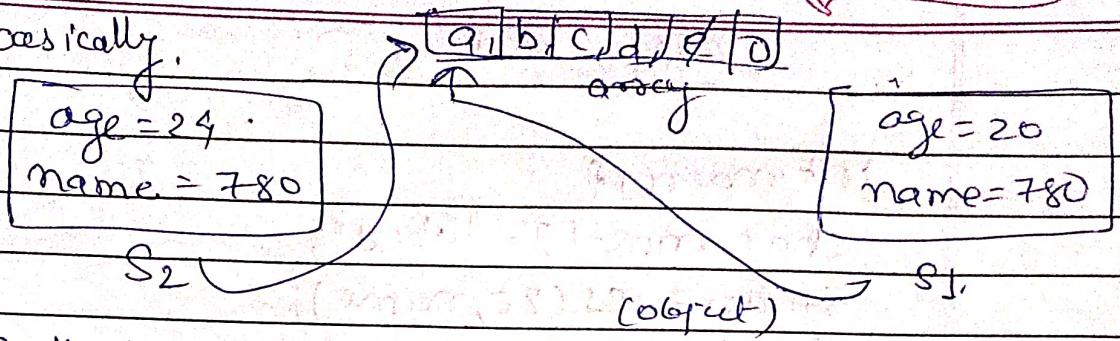
```
class Student {
  int age;
  char *name;
  Public:
  Student (int age, char *name)
  {
    this -> age = age
    //shallow copy (which needed to be avoided)
    this -> name = name;
  }
void display ()
  {
    cout << name << " " << age << endl;
  }
};

int main () {
  char name[] = "abcd";
  Student S1 (20, name);
  S1. display ();
  name [3] = 'e';
  Student S2 (24, name);
  S2. display ();
}
```



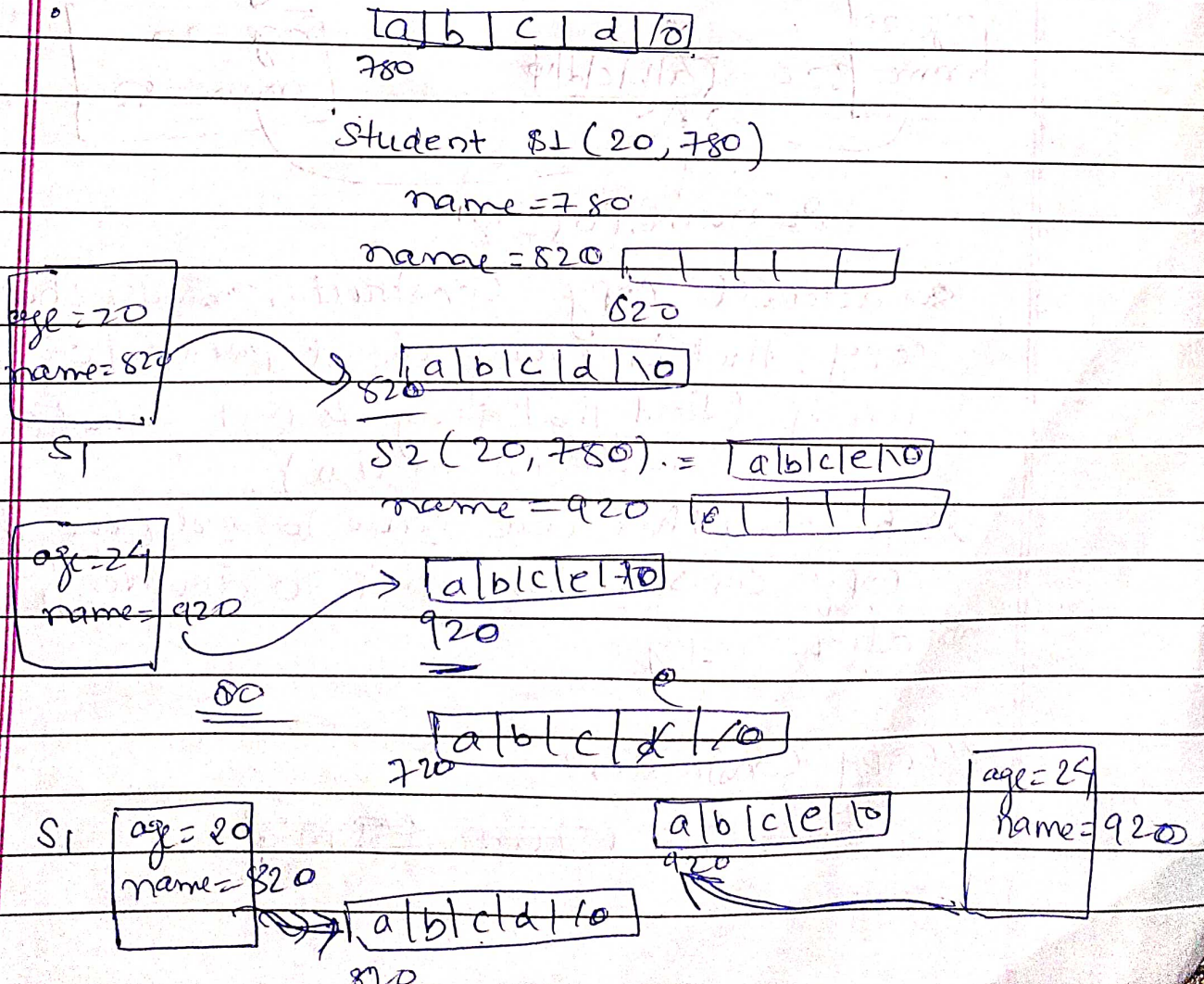
Date
Page

basically.



So that mean both block point to same array of char, but we do not want this. we want seperate array for each object.
this is called Shallow copy.
→ this is called Deep copy
→

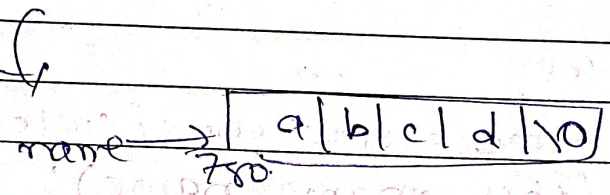
```
// Deep copy.                                   for null character
this -> name = new char[strlen(name)+1];
strcpy(this -> name, name);
```

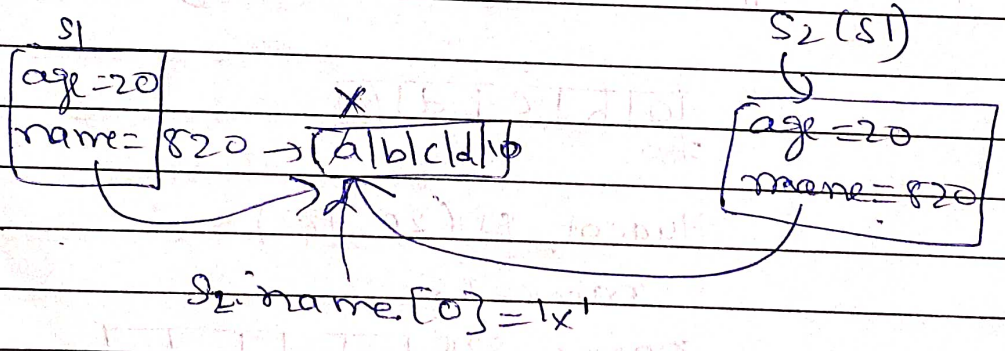


Date
Page

Copy Constructor

```
int main(){
 Char name[] = "abcd";
 Student S1(20, name);
 S1.display();
 Student S2(S1);
  S2.name[0]='x';
  S1.display();
  S2.display();
}
```





So actuall copy Constructor make shallow copy, that is every object point to same array (find that array is not what we made in main).

So to solve this we have to redefine the copy constructor, so it do deep copy not shallow copy.

```
//Copy Constructor
 student (student S1){
```

// copy Constructor, copy assignment operator always make shallow copy that is not make it's own new array

```
Student (Student S)
{
    this->age = S.age;
    // Deep copy.
    this->name = new char [strlen(S.name)+1];
    Strcpy (name, S.name);
}
```

but wait there is a problem in it

first in main S2(S1) → it call copy constructor. which we redefine so as we redefined it so default copy constructor has gone.

call copy constructor

```
Student (Student S) {
    Student S = main.S1
    S(S1)
}
```

Call copy constructor.

Note if & if only if we make copy Constructor by own Hence and thereby default copy constructor gone

So by this it will go to infinite loop.

to solve this we have to do something so that at [Student (Student S) { } ;] at that place copy Constructor not called again.

So to solve we use refference variable.

~~which not~~ if we not used refference variable vs if we use refference variable.

```
int i = 5 → [5] i
int j = i  (i) = [5]
```

(basically make new j and assign value as in i)

```
int i = 5
int &j = i
```

[5] i

that block is now access by i & j both, no new block created

[5] i, j

so Student(Student const & s) {

}

default copy constructor मे इसी पे होता है।

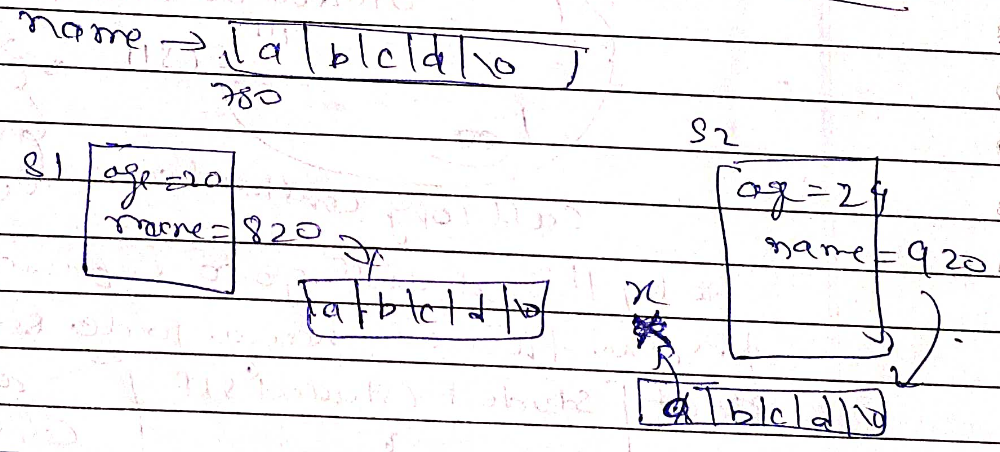
So that why we use refference variable so that real (main original) S1 from main passed and we have to use Constant as well so that S1 value not get manuplated.

So after making it constant

[5] (i के through तो i++ कर सकते है but
i, j

j++ नही कर पाएगा as j के through changes allowed नही है।)

So now this is pictorial Representation



Initializing list

- int a = 5; [5] a ✓ ≠ int a; [garbage] a
  a = 5; [5] a

- int const b = a; ✓ ≠ int const b; ← error at
  b = a ✗

Date
Page

(r) `int i = 5;` `int &j = i` ✓

`int i = 5;` `int &j;` `j = 5` ✗ error at

Q Now we want to make our Students class properties rollno to be constant so that no one can modify it latter.

```
Class Student {
    public:
        int age
        const int rollNO;

};
int main() {
    Student S1;
    S1.age = 20;
    S1.rollNumber = 101;
}
```

error at bcz this will fill garbage value to a constant variable.

Problem in this code is that, whenever we make object of student class, respective Constructor is called and that means an object of student with age and rollNo. as property created in a memory, so all variables garbage value before it get initilize in constructor so it give error.

So solution is to use initialisation list so that rollNo get value (initilized) at a same time when it get memory space

```
Class Student {
Public:
  int age
  Const int rollno;

  Student(int r) : rollNo(r) {
    }
};
int main() {
  Student S1 (213);
  S1.age = 20;
}
```

≡ Const int rollNO = r;

initilazing list

We can pass multiple value in initilizing list

```
Student(int r, int age) : rollno(r), age(age) {
    }
Student S1(213, 20)
```

not needed to use 'this'

So Initilizing list is used whenever we have to Initilize const or refference data member

for a refference variable we can use like this

```
Class student {
Public:
  int age;
```

```
const int rollno;
int &x;
student (int r, int age) : rollNo(r), age(age), x(age){
    }
};
int main(){
  Student S1(100,10);
}
```

meaning of this line

```
// const int rollNo = r
// int age = age
// int &x = age
```

Constant function

- Fraction const F3;
  - ↳ it will create Constant obj F3 for Fraction class
  - so we cannot change ~~variable~~ data member of F3 and we can't even call a ~~function which changes those datatype~~ any normal function.

Note:- by Constant obj we can only call Constant function.

Constant function :- which doesn't change any property of curr obj and const keyword is used after name of that function

(✓ Constant obj sirf Constant function ko call kr sakti)

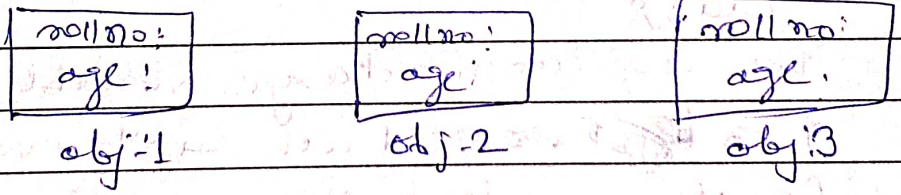
Static Member

- there are some Variable or Property which does not belong to a particular object but instead of that it belong to all object.

ex. total student in class Student
So for that we have make it static member which is belong to class, not to a object

```
Class Student {
  public:
  int rollno;      } — non-static
  int age;         }
  static int total_Student; } — static
};
```

So whenever we create obj :- obj look like this



* obj1.rollno } — How to call to non-static ~~mem~~ Variable member.
  obj2.age }

* ~~obj~~. How to access static Member.
  Student :: total_student;
  className :: name of Property;
  :: → Scope resolution opr

* How to initialize static member
Ans static member can be initilized outside the class only.

eg :-

```
int Student :: totalStudents = 0; // initialize static data member.
```

Note

S1. totalStudents // this will not give error but logically wrong.

S1.totalStudents = 20; // this is ideally wrong but not give error instead of that it will change actual static totalStudent Variable = 20;

Static function

function which belong to a class not to a obj

ex:- function to calculate total no. of student

```
static int getTotalStudent() {
    return totalStudents;
}
```

=> to access that function.

```
cout << Student :: getTotalStudent() << endl;
```

Note

- Static function can only access static data member (so we cannot use any non static data member or function inside static function)
- Static function के पास "this" keyword नहीं होता है। (access नहीं होता है)

Operator Overloading

↳ means how to extend the use of operator in our user define class

eg adding two complex no.

$c_1 + c_2$ for this we have to overload the '+' operator

or

eg Fraction $F_3 = f_1 + F_2$

to overload this operator we need function like this

↳ Fraction operator+ (fraction const &f2) {

};

Fraction $F_3 = f_1 + f_2$

call this ↑ this में जाता है ↖ yeas a argument pass होता है

Overloading + operator

```
Fraction operator+(fraction const &f2) {
    int lcm = denominator * f2.denominator;
    int x = lcm / denominator;
    int y = lcm / f2.denominator;
    int num = x * numerator + (y * f2.numerator);
    Fraction fNew(num, lcm);
    return fNew;
}
```

Similarly for overloading * operator

```
Fraction operator* (fraction f2) {
    int n = numerator * f2.numerator;
```

int d = denominator * f2.denominator;
Fraction fNew(n, d);
return fNew;
}

f1 = f2 * f3
return ya pe hoga.
go on "this" keyword
pass as argument

- Same for == operator
- Note the above overloaded + and * overloading function can be consider as a constant function as it's not changing the value of 'this' obj but changing the value of newly created obj.

→ Overload unary operator like ++
++ → Pre increment
Post increment

// Pre-increment
when nothing to return
Version 1
void operator ++() {
numerator = nerator + denominator;
Simplify();
}

++f1;
Pass as 'this' keyword.

Date
Page

Version-2. when we have to return fraction,

```
fraction operator ++() {
    numerator = numerator + denominator;
    simplify();
    return *this;
}
```

f1 = 2/3

- fraction F3 = ++f1 ✓ work fine. //F3 = 5/3

f1 = 10/2

- ++(++f1)
  f1.Print() // f1 = 6/1 X

f1 = 10/2

- fraction f2 = ++(++f1)
  f2.Print() // f2 = 7/1 ✓ f1 = 6/1 X
  f1.Print()

So why f1 is not updating?

Ans:-

~~f2 = ++(++f1)~~ Whenever we return something from function to main and not take this in defined variable the compiler itself make a temporary variable and save there

So in ~~f2 = ++(++f1)~~   let f1 = 10/2 first

Fraction F2 = ++(++f1)

so first (++f1) is passed

*this

f1 reflected as this

f1 is inc by 1 so f1 = 6/1

↳ ~~[illegible]~~

in main also as f1 he to pass hua ha (this me)

So when it return to main, as there is no one to take the value so it's save in temporay variable let say x
now (f2=++(x)) . x is passed and inc by 1 and save to f2 so
f2 = 7/1 and f1 = 6/1
So what is soln to solve this so f1 also inc twice.
So In.

return as reference.

```
fraction & operator++() {
    = -i
    return *this;
}
```

so this pass. as &f1
and in main receive as x = &f1 so it not make new variable but instead of that now x is now refference to f1 as well.
So now [6/1] so if f1 = ++(x) so
x, f1
it increse x by 1 i.e inc f1 by 1 so
Now f1 = 7/1 , f2 = 7/1

So in that function we can not use constant as it is changing "this" object always so it's not a constant function.

Post increment :- 1st use then increase
Pre increment :- increase then use.

Eg if int i=5;

i++	++i
cout<<i;	cout<<i;
// 6	// 6 ⇒ no diff in

i=5	i=5
int j = ++i;	int j = i++;
// i = 6	// i = 6
// j = 6	// j = 5 ⇒ diff

Overload Post increment

~~fraction operator ++ (int)~~ f1 = 10/2

Fraction f4 = f1++;

So we have to return (old value) and increase f1 by 1

use int to differentiate between pre and Post increment

```
fraction operator ++(int)
{
    fraction fnew(numerator, denominator);
    numerator = numerator + denominator;
    simplify();
    return fNew;
    fNew.simplify()
    return fNew;
}
```

So $f_1 = 10/2$

Fraction $f_3 = f_1$++

// $f_1 = 6/1$

// $f_2 = 5/1$

So Now what is the value if we do nesting as earlier

fraction $f_3 = (f_1++)++$; ✗

nesting is not allowed in Post-increment

If you do then it give ~~unpredictable~~ garbage value as answer.

So no nesting allowed in post increment.

~~overload~~

* Overload += operator

~~$f_1 = 10/f_2$~~ ~~$f_2 =$~~

~~fraction f4 (+f~~

int i=5, j=3

i+=j // i=8 j=3 } so we have to do i=i+j;

int i=5, j=3

(i+=j) +=j; } for nesting (nesting is allowed here)

// i=

// j=3

@ void Operator += (fraction cas+(f2) {

}

> $f_1 += f_2$; ✓ work fine

> $(f_1 += f_2) += f_2$ ✗ give error as it not return anything.

Fraction operator+= (Fraction const &f2){

return *this;

}

for

f1 = 3/1
f2 = 4/1

f1+=f2; ✓ work fine

(f1+=f2)+=f2; ✗ f1 = 7/1 ✗
f2 = 4/1 ✓

Value of f1 is wrong bcz when it return (*this) to main then it make new varible let x and than x+=f2 is called when x = this so f1 में inc नहीं होगा। f1 inc एक बार ही होगा।

So we have to return as refference so x = &f1 ; so when, 1st time x में inc होगा तब f1 में भी होगा that means f1 में दो बार inc होगा।

Fraction& oporator+= (Fraction const &f2){

int lcm = denominator * f2.denominator;
int X = lcm/denominator;
int Y = lcm/f2.denominator;
int num = X*numerator + (Y*f2.numator);
numerator = num;
denominator = lcm;
Simplify();
return *this;

}

f1 = 3/1
f2 = 4/1

f1+=f2 ✓
(f1+=f2)+=f2 ✓ f1 = 11/1 ✓ f2 = 4/1 ✓

Date
Page

20 Dynamic Array class

Here we want to make a class, where array size can be ~~increasing~~ increased.

ex :- let say we have array a => [1][2][3]

now we want to insert 4th element then we just create array of length 2x times of a. and make a to point this new array

a → [1][2][3][ ][ ][ ] so now we can insert 4 as well

// See code

Assignment

Implement Polynomial class

do yourself