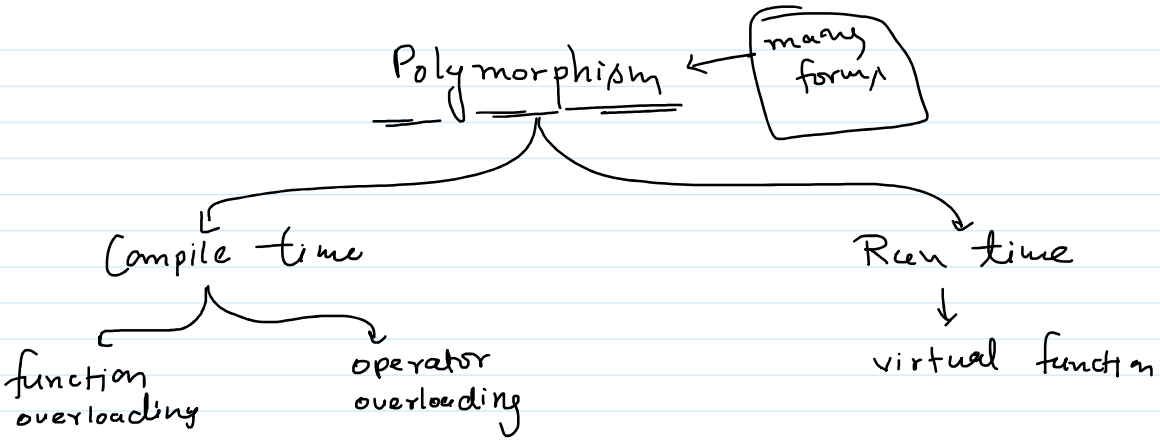
Day 07

24 March 2025 08:27 AM



① Function overloading :→ ① Same name
② Arg diff

```
int sum(int a, int b)
{
    return a+b;
}

double sum(double a, double b)
{
    return a+b;
}

string sum(string s1, string s2)
{
    return s1+s2;
}
```

```
sum(10, 20)
sum(7.5, 8.4)
sum("abcd", "xyz")
```

```
#include<iostream>
using namespace std;
int sum(int a, int b)
{
    cout<<"Int ";
    return a+b;
}
double sum(double a, double b)
{
    cout<<"Double ";
    return a+b;
}
int main()
{
    cout<<sum(10,20)<<endl;
    cout<<sum(7.5,8.7)<<endl;
    return 0;
}
```

Diff arguments

④ diff types

sum(int, int)

sum(float, float)

sum(double, double)

③ diff number of arguments

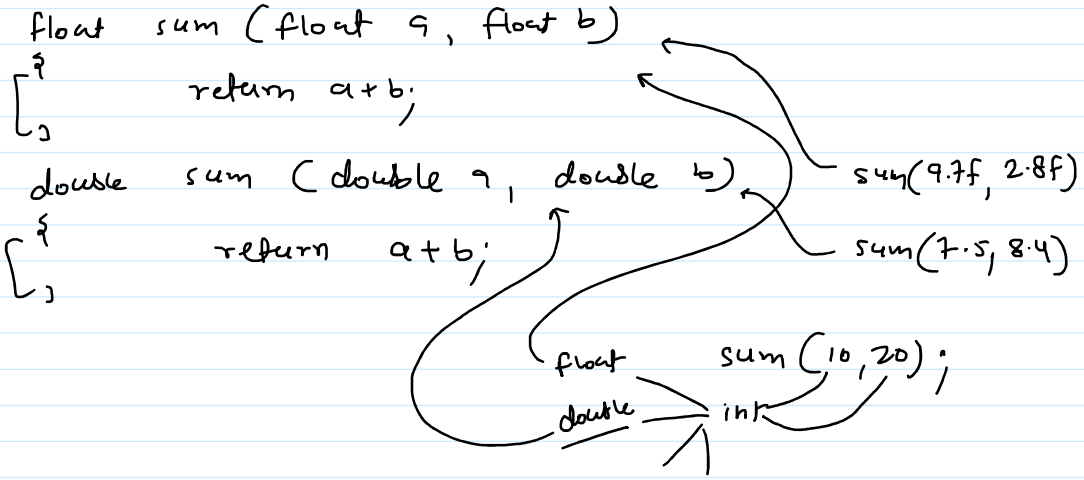
sum(int, int);

sum(int, int, int);

② diff order of arguments

sum(int, float)

sum(float, int)




```
#include<iostream>
using namespace std;
float sum(float a, float b)
{
    cout<<"Float ";
    return a+b;
}
double sum(double a, double b)
{
    cout<<"Double ";
    return a+b;
}
int sum(int a, int b)
```

```
{
    return a+b;
}
int main()
{
    cout<<sum(10,20)<<endl;
    return 0;
}
```

default argument function :→

```
int sum (int a, int b)
{
    return a+b;
}
int sum (int a, int b, int c)
{
    return a+b+c;
}
int sum (int a, int b, int c, int d)
{
    return a+b+c+d;
}
```

sum(10,20)

sum(10,20,30)

sum(10,20,30,40)

```
int sum (int a, int b, int c=0, int d=0)
{
    return a+b+c+d;
}
```

default

sum(10,20);

sum(10,20,30);

sum(10,20,30,40);

✓ Call by Reference :-

Reference variable :→

Types of variable :-

C
1. value variables
2. Address variables (pointer)

C++
3. Reference variable

```
int a;
int &r = a;
```

a = 10 (100) → 20

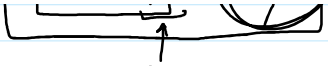
r

Reference

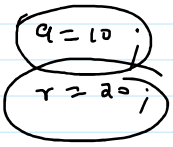
int &

↑ Reference

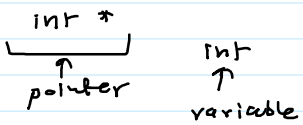
int *



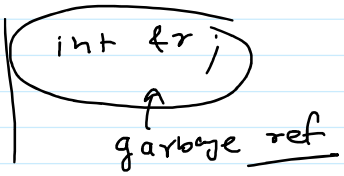
Reference



Reference



&r = b ; ← ✗

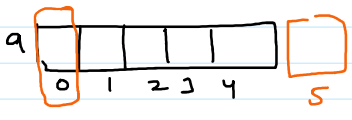



```
int main()
{
    int a=10;
    int &r = a;
    cout<<r<<endl;
    r=20;
    cout<<a<<endl;      //20
    return 0;
}
```

```
#include<iostream>
using namespace std;
int myswap(int &a, int &b)
{
    int temp = a;
    a=b;
    b=temp;
}
int main()
{

    int a=10,b=20;
    myswap(a,b);        //call by reference
    return 0;
}
```

int a[5] ;



int &r = a; ✗

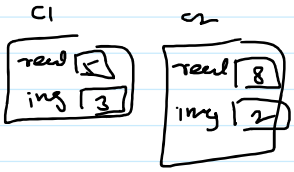
int &r = a[5] ; ✓

}

complex sum (Complex &obj)

{

complex c1 , c2



c1. sum (c2)

↑ by value / ref

c1.sum(c2)

↑ call by ~~value~~ Ref

Constructor & Destructor

Constructor → member function.
→ auto called when we create an object

→ Initialize an object
↑ (store value of member variable)

→ name → (same as class name)

```
#include<iostream>
using namespace std;
class data{
    int a;
    float b;
    public:
        data()
        {
            cout<<"Constructor called\n";
        }
};
int main()
{

    data d1,d2,d3,d4;
    return 0;
}
```

Types of constructor :-

1. Default Constructor (zero parameterized Cons)
2. Parameterized constructor
   - → one | two .... n parameters
   - → Default argument parameterized
   - → Dynamic constructor
3. Copy Constructor

```
#include<iostream>
using namespace std;
class data{
    int a;
    float b;
    public:


        data(int a=0, float b=0.0f)
        {
            this->a = a;
            this->b = b;
            cout<<"Two Constructor called\n";
```

```
        }
};
int main()
{

    data d1;
    data d2 = data(10);     // data d2(10);        // data d2 = 10;
    //  data d3 = 10,2.5f  error
    data d3 = data(10,2.5f);     // data d3(10,2.5f);
    return 0;
}
```

Dynamic Constructor :-

```
int a[5];
int *p = a;
```

Dynamic memory :→

C

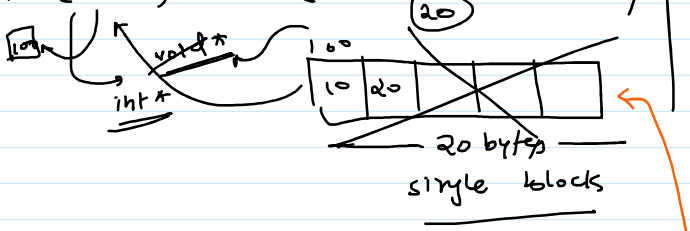
malloc

n [5] ← num of elements

```
int *p;
p = (int *)malloc(n * sizeof(int));
```

type casting

5 * 4 = 20




```
p[0] = 10;
p[1] = 20;
free(p);
```

dynamic

C++

new ← keyword

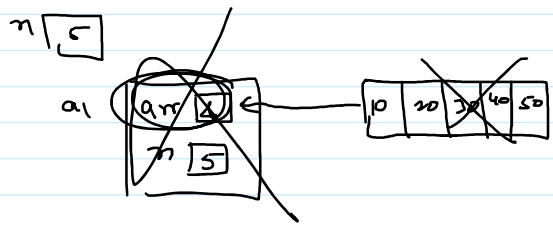
```
int *p;
n [5]
p = new int[n];
```


```
delete [] p;
```

```
#include<iostream>
using namespace std;
class Array{
    int *arr;
    int n;
    public:
        Array(int n)    //dynamic constructor
        {
            this->n = n;
            arr = new int[n];       //dynamic memory
        }
        void input()
        {
            cout<<"Enter "<<n<<" values:";
            for(int i=0;i<n;i++)
                cin>>arr[i];
        }
        void output()
        {
            for(int i=0;i<n;i++)
```

5

```
    }
    void output()
    {
        for(int i=0;i<n;i++)
            cout<<arr[i]<<" ";
        cout<<endl;
    }
    void free()
    {
        delete []arr;
    }
};
int main()
{
    int n;
    cout<<"Enter number of elements:";
    cin>>n;
    Array a1=n;
    a1.input();
    a1.output();
    a1.free();
    return 0;
}
```



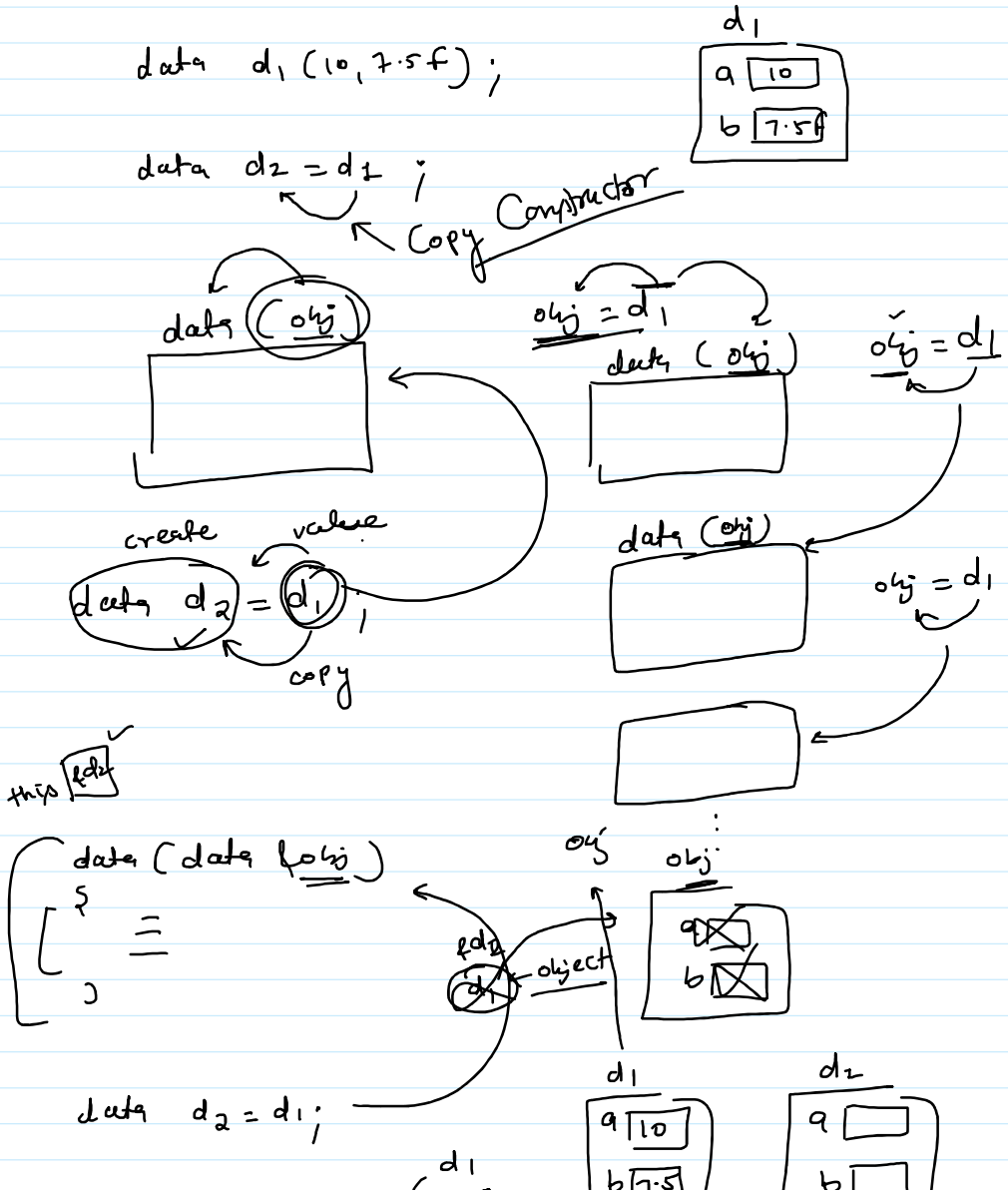
Destructor → member func
→ auto call –
→ same name as class name prefix ~ (↑ Tilde)
~ array
→ No parameters
→ Destructor overloading not allowed

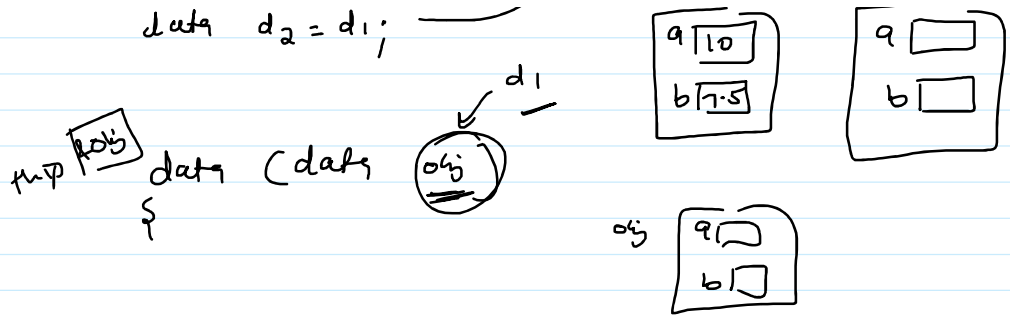
```
#include<iostream>
using namespace std;
class data{
    int a;
    public:
        data(int a)
        {
            this->a = a;
            cout<<a<<" Constructor called\n";
        }
        ~data()
        {
            cout<<a<<" Destructor called\n";
        }
};
int main()
{
    data d1=10,d2=20,d3=30;
}
```

```
#include<iostream>
using namespace std;
class Array{
    int *arr;
    int n;
    public:
        Array(int n)    //dynamic constructor
        {
            this->n = n;
            arr = new int[n];       //dynamic memory
        }
```

```
        void input()
        {
            cout<<"Enter "<<n<<" values:";
            for(int i=0;i<n;i++)
                cin>>arr[i];
        }
        void output()
        {
            for(int i=0;i<n;i++)
                cout<<arr[i]<<" ";
            cout<<endl;
        }
        ~Array()
        {
            delete []arr;
            cout<<"Dynamic memory Deleted\n";
        }
};
int main()
{
    int n;
    cout<<"Enter number of elements:";
    cin>>n;
    Array a1=n;
    a1.input();
    a1.output();
    return 0;
}
```

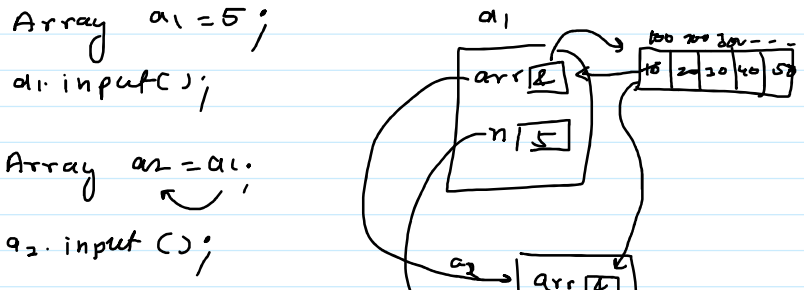
Copy Constructor :-






```
#include<iostream>
using namespace std;
class data{
    int a;
    float b;
    public:
        data(int a1=0, float b1=0.0f)
        {
            a=a1;
            b=b1;
            cout<<"Constructor called\n";
        }
        data(const data &obj)
        {
            a = obj.a;
            b = obj.b;
            cout<<"Copy Constructor called\n";
        }
        void output()
        {
            cout<<a<<'\t'<<b<<endl;
        }
};
int main()
{
    data d1(10,7.5f);
    d1.output();
    data d2 = d1;
    d2.output();
    return 0;
}
```

Shallow copy :- member to member




```
#include<iostream>
using namespace std;
class Array{
    int *arr;
    int n;
    public:
        Array(int n)    //dynamic constructor
        {
            this->n = n;
            arr = new int[n];       //dynamic memory
        }
        void input()
        {
            cout<<"Enter "<<n<<" values:";
            for(int i=0;i<n;i++)
                cin>>arr[i];
        }
        void output()
        {
            for(int i=0;i<n;i++)
                cout<<arr[i]<<" ";
            cout<<endl;
        }
        Array(const Array &obj)
        {
            n = obj.n;
            arr = new int[n];
            for(int i=0 ; i<n ;i++)          //deep copy
                arr[i] = obj.arr[i];
        }
        ~Array()
        {
            delete []arr;
            cout<<"Dynamic memory Deleted\n";
        }
};
int main()
{
    int n;
    cout<<"Enter number of elements:";
    cin>>n;
    Array a1=n;
    a1.input();
    a1.output();
    Array a2=a1;
    a2.output();
    a2.input();
    a1.output();
    a2.output();
    return 0;
}
```