**12.06.20**

**Polymorphism:**

One object behaves as different objects in different situation.

A close up of a map

Description automatically generated

Example:

One Person 🡪 a. Student in college

b. Intern at Company

c. Passenger in College bus

d. Son/Daughter at home etc

A close up of text on a white background

Description automatically generated

Agenda:

1. Function/Method Overloading
2. Function/Method Overriding
3. **Operator Overloading**
4. **Function Overloading: Two or more functions can have Same name & different Parameters**

#include<iostream>

using namespace std;

class Add

{ public:

void sum(int a, int b)

{

cout<<a+b<<endl;

}

void sum(int a, int b, int c)

{

cout<<a+b+c<<endl;

}

void sum(double a, double b, int c)

{

cout<<a+b+c<<endl;

}

void sum(double a, int b)

{

cout<<a+b<<endl;

}

};

int main()

{

Add obj;

obj.sum(10,20);

obj.sum(4.1125,2);

obj.sum(1,1,1);

obj.sum(4.1,6.3,1);

obj.sum(4.2,1);

}

1. **Function Overriding: Two or more functions can have Same name & same Parameters**

**Mandate: Use inheritance concept must be der.  
  
Reason: cannot be done within a class.. therefore we use inheritance – (Super class & Child Class)- During execution its confusing**

**Hence we require – Derived class & Base Class**

Code 1:

#include<iostream>

using namespace std;

class Base

{

public:

void show()

{

cout<<"Base Class"<<endl;

}

};

class Derived: public Base

{

public:

void show()

{

cout<<"Derived Class"<<endl;

}

};

int main()

{

Base b;

Derived d;

b.show();

d.show();

return 0;

}

Code 2:

#include<iostream>

using namespace std;

class Add

{

public:

void sum(int a, int b)

{

cout<<a+b<<endl;

}

};

class ChildAdd: public Add

{

void sum(int a, int b)

{

cout<<a+b<<endl;

}

};

int main()

{

Add obj;

obj.sum(10,20);

obj.sum(1,1);

}

**Conclusion Note:**

**1.if using same name & same parameter use – overriding – use super & child class**

**2. if using same name & differen parameter – use overloading – same/single class is enough**

1. **Operator Overloading:**

It is a type of polymorphism in which operator is overloaded to give user defined meaning.  
Ex:   
a . add 2 numbers – we use + operator  
b. concatenate two strings use + operator  
So we are using both + & + … this is called operator overloaded.

Q) How to overload the operator???

Ans: To overload a operator, operator function is defined inside a class.

Syntax:

**class class\_name  
{  
public:  
return\_type operator\_sign(args)  
{  
….  
}  
};**

**Code:**

#include<iostream>

using namespace std;

class Rectangle

{

int l,b;

public:

Rectangle()

{

l=0;

b=0;

}

void operator++()

{

l+=2;

b+=2;

}

void display()

{

cout<<l<<endl;

cout<<b<<endl;

}

};

int main()

{

Rectangle R;

cout<<"Before increment"<<endl;

R.display();

++R;

cout<<"After increment"<<endl;

R.display();

return 0;

}

**output:**

0  
0  
2  
2

**Rules:**

1. **Only existing member can be overloaded, we cannot create your own operator to overload.**
2. **The Overload operator must have at least one operand of user defined type.**
3. **It follows syntax rules of original operator. This means we cannot change the basic meaning of operator.**
4. **Some operators cannot be overloaded** 
   1. **(.) dot – member access operator**
   2. **(.\*) - pointer to member operator**
   3. **(::) scope resolution**
   4. **(sizeof)**
   5. **(?:) ternary operators**
   6. **= etc**