::draftWatermark ::

Agenda

- this pointer
- Types of Member Functions
- constructor and their types
- Constant
- Dynamic Memory Allocation
- ~~Reference~~

this pointer

- If we call member function on object then compiler implicitly pass address of that object as a argument to the function implicitly.
- To store address of object compiler implitly declare one pointer as a parameter inside member function. Such parameter is called this pointer.
- this is a keyword. "this" pointer is a constant pointer.
- General type of this pointer is:
    - Classname * const this;

Member Functions

- The functions declared inside the class are called as Member Functions.
- The member functions according to their behaviour are classified into following types

1. constructor
2. destructor
3. Mutator
4. Inspector
5. Facilitator

Constructor

- It is a member function of a class which is used to initialize object.
- Due to following reasons, constructor is considered as special function of the class:
    1. Its name is same as class name
    2. It doesn't have any return type.
    3. It is designed to call implicitly.
    4. In the life time of the object is gets called only once.
- We can not call constructor on object, pointer or reference explicitly. It is designed to call implicitly.
- constructor does not get called if we create pointer or reference.
- We can use any access specifier on constructor.
- If ctor is public then we can create object of the class inside member function as well as non member function but if constructor is private then we can create object of the class inside member function only.
- We can not declare constructor static, constant, volatile or virtual. We can declare constructor only inline.

- constructor can contain return statement,it cannot return any value from constructor as return statement is used only to return controll to the calling function.

Types of Constructor

1. Parameterless Constructor

    - A constructor, which do not take any parameter is called Parameterless constructor.
    - It is also called zero argument constructor or user defined default constructor.
    - If we create object without passing argument then parameterless constructor gets called.

```
class Point
{
    int x;
    int y;
public:
    Point()
    {
        x = 1;
        y = 1;
    }
}

int main(){
 Point pt1;
 Point pt2;
//Point::Point( )
//Point::Point( )
}
```

2. Parameterized Constructor

    - If constructor take parameter then it is called parameterized constructor.
    - If we create object, by passing argument then paramterized constructor gets called.
    - Copy constructor is a single parameter constructor hence it is considered as parameterized constructor.

```
 //Point *const this;
 Point( int xPos, int yPos )
 {
 this->xPos = xPos;
 this->yPos = yPos;
 }

 Point pt1(10,20);
 //Point::Point(int,int)
 Point pt2; //Point::Point( )
```

3. Default constructor

   - If we do not define constructor inside class then compiler generates default constructor for the class.
   - Compiler do not provide default parameterized constructor. Compiler generated default constructor is parameterless.
   - If we want to create object by passing argument then its programmers responsibility to write parameterized constructor inside class.
   - Default constructor do not initialize data members declared by programmer. It is used to initialize data members declared by compiler(e.g v-ptr).
   - If compiler do not declare any data member implicitly then it doesnt generate default constructor.
   - We can write multiple constructor's inside class. It is called constructor overloading.

```
 Point()
{
    cout << "Inside Parameterless Ctor" << endl;
    x = 1;
    y = 1;
}
// constructor overloading
Point(int value)
{
    x = value;
    y = value;
}
// constructor overloading
Point(int x, int y)
{
    cout << "Inside Parameterized Ctor" << endl;
    this->x = x;
    this->y = y;
}
```

Constructor delegation(C++ 11)

- In C++98 and C++ 03, we can not call constructor from another constructor. In other words C++ do not support constructor chaining.
- In C++ 11 we can call constructor from another constructor. It is called constructor delegation. Its main purpose is to reuse body of existing constructor.

```
Point() : Point(1, 1)
{
    cout << "Inside Parameterless Ctor" << endl;
}

Point(int value) : Point(value, value)
{
```

```
        cout << "Inside single Parameterized Ctor" << endl;
    }

    Point(int x, int y)
    {
        cout << "Inside Parameterized Ctor" << endl;
        this->x = x;
        this->y = y;
    }
```

Constructor's member initializer list

- If we want to initialize data members according to users requirement then we should use constructor body.
- If we want to initialize data member according to order of data member declaration then we should use constructors member initializer list.
- Except array we can initialize any member inside constructors member initializer list.
- If we provide constructor member initializer list as well Constructor body then compiler first execute constructor member initializer list.
- In case of modular approach, constructors member initializer list must appear in definition part(.cpp).
- If we declare data member constant then it is mandatory to initialize it using constructors member initializer list.

```
class Point
{
    int x;
    int y;
    const int num;

public:
    // ctor members initializer list initialize data member according to order of
data member declaration in class
    // here x will get initialized first then y and then num
    Point(int value) : y(value), x(++value), num(value) // x= 3, y = 3, num = 3
    {
    }

    // Point(int value)
    // {
    //     this->y = value;   // y = 2
    //     this->x = ++value; // x = 3
    //     this->num = value; // NOT OK
    // }
}
```

Destructor

- It is a member function of a class which is used to release the resources.
- Due to following reasons, it is considered as special function of the class
    1. Its name is same as class name and always preceds with tild operator(~)
    2. It doesnt have return type or doesn't take parameter.
    3. It is designed to call implicitly.
- We can declare destructor as a inline and virtual only.
- Destructor calling sequence is exactly opposite of constructor calling sequence.
- We can not call constructor on object, pointer or reference explicitly. It is designed to call implicitly.
- Destructor is designed to call implicitly but we can call it explicitly.
- If we do not define destructor inside class then compiler generates default destructor for the class.
- Default destructor do not deallocate resources allocated by the programmer. If we want to deallocate it then we should define destructor inside class.

Mutator

- A member function of a class, which is used to modify state of the object is called mutator function.
- It is also called as modifier function or setter function
- e.g setReal() and setImag()

Inspector

- A member function of class, which is used to read state of the object is called inspector function.
- It is also called selector function of getter function.
- e.g getReal() and getImag()

Facilitator

- Member function of a class which allows us to perform operations on Console/file/database is called faciliator function.
- e.g acceptData() and printData()