Agenda

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

Constructor

1
- It is a member function of a class which is used to initialize object.
- 2 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.
- 4 We can not call constructor on object, pointer or reference explicitly. It is designed to call implicitly.
- 5 constructor does not get called if we create pointer or reference.
- 6 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.
- 7 We can not declare constructor static, constant, volatile or virtual. We can declare constructor only inline. Inlining ctor. can improve performance by reducing function call overhead, but there is potential increase in code size.
- 8 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 - do not take any parameter
   - zero argument constructor.
   - user defined default 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)

C++ 98 & C++ 03 don't have constructor chaining
C++ 11 have ctor. delegation, to resuse body of existing ctor.

- ~~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

1 to initialize data member according to user requirement use ctor. body.
2 to initialize data member according to order of data member declaration use ctor. 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.
- 3 Except array we can initialize any member inside constructors member initializer list.
- 4 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 5 it is mandatory to initialize it using constructors member initializer list.

constant data member

```
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
    // }
}
```

```
class Point{
    int x;
    int y;
    const int num;
    public:
        Point(int value): y(value++), x(++value), num(++value){
            // x=4 y=4 num=6
        }
    void display(){
        cout << "x: "<<x<<" y: "<<y<<" num: "<<num<<endl;
    }
};
```

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()

Constant

- const is type qualifier.
- If we dont want modify value of the variable then we should use const keyword.
- constant variable is also called as read only variable.
- In C++, Initializing constant variable is mandatory.

```
const int num2; //Not OK : In C++
const int num3 = 10; //OK : In C++
```

- We can even make

1. Data Member as constant

```
- Once initialized, if we dont want to modify state of the data member
inside any member function of the class including constructor body then we
should declare data member constant.
- If we declare data member constant then it is mandatory to initialize it
using constructors member initializer list.

```CPP
    class Test
    {
    private:
        const int num1;
    public:
        Test( void ) : num1( 10 ) //OK
        {
            //this->num1 = 10; //Not OK
        }
    };
```
```

2. Member Function as constant

```
- We can not declare global function constant but we can declare member
function constant.
- If we dont want to modify state of current object inside member function
then we should declare member function constant.
- Non constant member function get this pointer like: `ClassName *const
this`
```

```
- Constant member function get this pointer like: `const ClassName *const
this;`
- We can not delclare following function constant:

  1. Global Function
  2. Static Member Function
  3. Constructor
  4. Destructor

- Since main function is a global function, we can not delcare it constant.
- We should declare read only function constant. e.g getter function,
printRecord function etc.
- In constant member function, if we want to modify state of non constant
data member then we should use mutable keyword.
```

3. Object as Constant

```
- If we dont want to modify state of the object then instead of declaring
data member constant, we should declare object constant.
- On non constant object, we can call constant as well as non constant
member function.
- On Constant object, we can call only constant member function of the
class.
```

Assignment

1. Complete Assignment-2
2. From demo11.cpp code complete the implementation of deposit and withdraw. Write a menu diven code where user will have functunality of
    - create account
    - display account details
    - deposit
    - withdraw
    - check balance