

# Classes

---

ENG. LAMYA GABER ALI

# Why Classes

---

- In data structure, we can create a new datatype to collect related variables in one place.
- But still there are some problems in structure, that we can't define functions in data structure.
- For example, if we need to compute an area of rectangle, circle and other shapes. By using data structure, we can only collect the attributes of each shape in one structure (data). But we have to define separate functions to compute each operation of each shape.
- So with data structure, we can only collect data not operations in one place.
- For this reason, we have to use object oriented programming (classes).

```
#include <iostream>
using namespace std;
struct rect{
    float l, w;
};
void area_rect(rect r){
    cout << "area of rectangle = " << r.l*r.w << endl;
}
struct circle{
    const float PI = 3.1415;
    float r;
};
void area_circle(circle c){
    cout << "area of circle = " << c.PI * c.r * c.r << endl;
}
int main(){
    rect r1, r2;
    r1.l = 2;
    r1.w = 3;
    area_rect(r1);

    r2.l = 10;
    r2.w = 20;
    area_rect(r2);

    circle c;
    c.r = 12;
    area_circle(c);
}
```

# Classes

---

- A *class* is an expanded concept of a data structure: instead of holding only data, it can hold both data and functions.
- In data structure, all members are public by default. But in classes, all members are private by default.
- So there are three different types of access specifier:
  - public
  - private
  - protected

# Public and Private members

```
#include <iostream>
using namespace std;

class Rect{
    int l, w;
public:
    void setvalues(int x, int y){
        l = x;
        w = y;
    }
    void area(){
        cout << "area of rectangle = " << l*w << endl;
    }
};

void main(){
    Rect r;
    r.
    {
        area
        setvalues
    }
    public : void Rect:area()
    File: Source.cpp
```

# Inline function

---

- Inline methods can be defined inside a class so there is no need for prototype.
- Also, you can define methods outside class. So you have to define a prototype inside its class. And you have to use a colon (::) with the name of the class as well.

```
#include <iostream>
using namespace std;

class Rect{
public:
    float l, w;
    void area();
};

void Rect::area(){
    cout << "area of rectangle = " << l*w << endl;
}

class Circle{
    float pi = 3.1415;
public:
    float r;
    void area();
};

void Circle::area(){
    cout << "area of circle = " << pi*r*r << endl;
}

void main(){
    Rect r;
    r.l = 2;
    r.w = 3;
    r.area();

    Circle c;
    c.r = 4.5;
    c.area();
}
```

```

#include <iostream>
using namespace std;

void Mul(int x, int y){
    cout << "x*y = " << x*y << endl;
}

void Div(int x, int y){
    cout << "x/y = " << x / y << endl;
}

void main(){
    int x, y;
    x = 4;
    y = 5;
    Mul(x, y);
    Div(x, y);
}

```

Program without class

```

#include <iostream>
using namespace std;

class Operations{
public:
    int x, y;
    void Mul(){
        cout << "x*y = " << x*y << endl;
    }
    void Div();
};

void Operations::Div(){
    cout << "x/y = " << x / y << endl;
}

void main(){
    Operations O;
    O.x = 4;
    O.y = 5;
    O.Mul();
    O.Div();
}

```

Program with class

# Constructor

---

- In order to initialize data within a class simplicity, you can create a constructor.
- Constructor is a special function which is automatically called whenever you create a new object of this class to initiate private data in the class.
- Constructor must have the same name of the class and cannot return any type of values; not even void.

```
#include <iostream>
using namespace std;

class Rect{
public:
    float l, w;
    void area();
};

void Rect::area(){
    cout << "area of rectangle = " << l*w << endl;
}

void main(){
    Rect r;
    r.l = 2;
    r.w = 3;
    r.area();
}
```

Using  
Constructor →

```
#include <iostream>
using namespace std;

class Rect{
    float l, w;
public:
    // define a constructor
    Rect(float length, float width){
        l = length;
        w = width;
    }
    void area();
};

void Rect::area(){
    cout << "area of rectangle = " << l*w << endl;
}

void main(){
    Rect r(2,3);
    r.area();
}
```

# Destructor

---

- Destructor fulfills the opposite functionality.
- If an object allocates dynamic memory, you have to delete memory that object was allocated by creating destructor.
- Destructor is a special function in a class which automatically called whenever you create an object from this class.
- It must have the same name as the name of class and it preceded with tilde (~).

```
#include <iostream>
using namespace std;

class Rect{
    float *x = new float[10];
    float l, w;
public:
    // define a constructor
    Rect(float length, float width){
        l = length;
        w = width;
    }
    ~Rect(){
        delete x;
    }
    void area();
};

void Rect::area(){
    cout << "area of rectangle = " << l*w << endl;
}

void main(){
    Rect r(2,3);
    r.area();
}
```

# Overloading

---

- If you need to create two objects of one class but each one has different number of parameters, this is called “Overloading”.
- you can create different constructors in the same class with the same name of this class but with different types of parameters.
- An example, If you need to calculate the area of a circle by two ways. So you need to create two different objects. Each one has its own parameters.
- So when an object is created, the constructor matches the arguments declared by the object is executed.

```
#include <iostream>
using namespace std;

class Rect{
    float l,w;
public:
    // define a constructor
    Rect(float length,float width){
        l = length;
        w = width;
    }
    Rect(){
        l = 4;
        w = 2;
    }
    void area();
};

void Rect::area(){
    cout << "area of rectangle = " << l*w << endl;
}

void main(){
    Rect r(2,3);
    r.area();

    Rect r1;
    r1.area();
}
```

# Pointers to classes

---

- As class is a new data type, so you can use pointers to classes like any datatypes.
- you can refer to a member in class by using (->).
- Note: you can use pointers with the same two ways in data structure.

```
#include <iostream>
using namespace std;

class Rect{

public:
    float l, w;
    void area();
};

void Rect::area(){
    cout << "area of rectangle = " << l*w << endl;
}

void main(){
    Rect r;
    r.l = 10;
    r.w = 2;
    r.area();
    // you can enter data by using * with .
    Rect *r1 = &r;
    (*r1).l = 10;
    (*r1).w = 2;
    (*r1).area();
    // Also you can enter data directly by using ->
    Rect *r2 = &r;
    r2->l = 10;
    r2->w = 2;
    r2->area();
}
```

```
#include <iostream>
#include <string>
using namespace std;

struct person{
    string name;
    int id;
    int father_id;
};

int main(){
    person X;
    person *p = &X;

    // you can enter data by using * with .
    (*p).name = "Ahmed";
    (*p).id = 12345;
    (*p).father_id = 6789;
    cout << "Name: " << (*p).name << endl;
    cout << "ID : " << (*p).id << endl << "Father_ID : " << (*p).father_id;

    // Also you can enter data directly by using ->
    p->name = "Ahmed";
    p->id = 12345;
    p->father_id = 6789;
    cout << "\nName: " << p->name << endl;
    cout << "ID : " << p->id << endl << "Father_ID : " << p->father_id << endl;
}
```

Thank you

