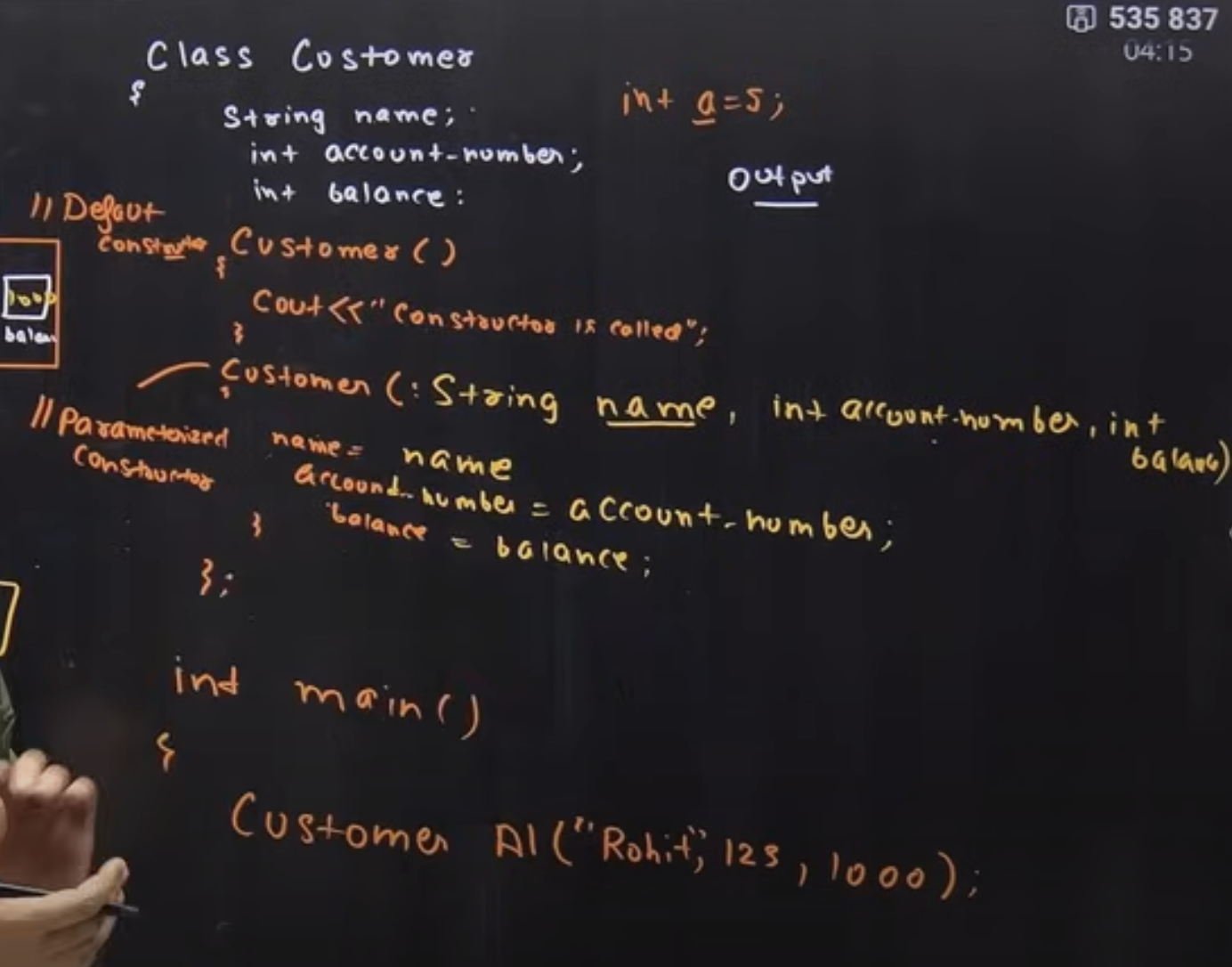
* Constructor  
  Default Constructor  
  Parameterised Constructor  
  Copy Constructor  
  This pointer  
  Constructor overloading  
  Incline constructor  
     
  **----------------------------------------------------**Constructor  
  A constructor is a special member function automatically called when an object is created. In C++, the constructor is automatically called when an object is created. It is a special class method because it does not have any return type. It has the same name as the class itself.  
     
  A constructor initializes the class data members with garbage value if we don’t put any value to it explicitly.  
  The constructor must be placed in the public section of the class because we want the class to be instantiated anywhere. For every object in its lifetime constructor is called only once at the time of creation.  
     
  **Example:**   
  class class\_name  
  {   
  int data\_member1;   
  string data\_member2;   
  //creating constructor   
  public:   
  class\_name()  
  {   
  // initialize data members with garbage value   
  }  
  };  
     
  Here, the function class\_name() is a constructor of the class ‘class\_name’. Notice that the constructor
  1. has the same name as the class,
  2. does not have any return type, and
  3. it is public
* If we do not specify a constructor, the C++ compiler generates a default constructor for an object (which expects no parameters and has an empty body).  
     
  **Types of Constructors:**   
  There are three types of constructors in C++:
  1. Default constructor
  2. Parameterized Constructor
  3. Copy Constructor
* -------------------------------------------------------------------------------------  
  Default constructor:-   
     
  A constructor that doesn't take any argument or has no parameters is known as a default constructor. In the example above, class\_name() is a default constructor.  
     
  Syntax:  
     
  class class\_name{   
     
  int data\_member1;   
  string data\_member2;   
  //default constructor   
     
  public:   
  class\_name()  
  {   
  // initializing data members with their default values   
  data\_member1 = 69;   
  data\_member2 = "Coding Ninjas";   
  }  
  };  
     
  Here, the class\_name() constructor will be called when the object is created. This sets the data\_member1 variable of the object to 69 and the data\_member2 variable of the object to “Coding Ninjas”.  
     
  Note: If we have not defined a constructor in our class, the C++ compiler will automatically create a default constructor with an empty code and no parameters, which will initialize data members with garbage values.  
     
  When we write our constructor explicitly, the inbuilt constructor will not be available for us.  
   -------------------------------------------------------------------------------------  
  Parameterized Constructor:-  
     
  This is another type of Constructor with parameters. The parameterized constructor takes its arguments provided by the programmer. These arguments help initialize an object when it is created.  
     
  To create a parameterized constructor, simply add parameters to it the way you would to any other function. When defining the constructor’s body, use the parameters to initialize the object.   
     
  Using this Constructor, you can provide different values to data members of different objects by passing the appropriate values as arguments.  
     
  Syntax:  
     
  class class\_name {  
   int data\_member1;  
   string data\_member2;  
   // parameterized constructor  
  public:  
   class\_name(int num, string str) {  
   // initializing data members with values provided  
   data\_member1 = num;  
   data\_member2 = str;  
   }  
  };  
     
  Here, we have created a parameterized constructor class\_name() that has 2 parameters: int num and string str. The values contained in these parameters are used to initialize the member variables data\_member1 and data\_member2.  
   -------------------------------------------------------------------------------------  
  @this  
  Purpose: A pointer to the current object instance.  
  Usage: Used within member functions to refer to the instance of the object on which the function is called.  
  @this  
  This->prop is same as \*(this).prop  
    
  name=name; mae kon sa parameter name h ar kon sa class ka name h iss  
  Chiz ko distinguish krne ke liye this keyword use krte h   
  This->name mtlab ye name class ka name h   
     
  "This" obj ke address ko store kr ke rkta h   
     
  Example:  
  cpp  
  Copy code  
  class MyClass {  
  public:  
   int value;  
   void setValue(int value) {  
   this->value = value; // Refers to the member variable  
   }  
  };  
  int main() {  
   MyClass obj;  
   obj.setValue(5);  
   std::cout << "Value: " << obj.value << std::endl; // Output: Value: 5  
   return 0;  
  }  
  **this Pointer**   
  this pointer holds the address of the current object. In simple words, you can say that this pointer points to the current object of the class.   
     
  There can be three main usages of this keyword in C++.
  1. It can be used to refer to a current class instance variable.
  2. It can be used to pass the current object as a parameter to another method.
  3. It can be used to declare indexers.
* Let’s take an example to understand this concept.  
     
  #include <bits/stdc++.h>  
  using namespace std;  
  class mobile  
  {  
   string model;  
   int year\_of\_manufacture;  
     
  public:  
   void set\_details(string model, int year\_of\_manufacture)  
   {  
   this->model = model;  
   this->year\_of\_manufacture = year\_of\_manufacture;  
   }  
     
   void print()  
   {  
   cout << this->model << endl;  
   cout << this->year\_of\_manufacture << endl;  
   }  
  };  
  int main()  
  {  
   mobile redmi;  
   redmi.set\_details("Note 7 Pro", 2019);  
   redmi.print();  
  }  
     
  Output:  
  Note 7 Pro2019  
     
  Here you can see that we have two data members model and year\_of\_manufacture. In member function set\_details(), we have two local variables with the same name as the data members’ names. Suppose you want to assign the local variable value to the data members. In that case, you won’t be able to do until unless you use this pointer because the compiler won’t know that you are referring to the object’s data members unless you use this pointer. This is one of example where you must use this pointer.  
     
  ---------------------------------------------------------------  
  Constructor Overloading  
     
  Constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.  
     
  As there is a concept of function overloading, similarly constructor overloading is applied when we overload a constructor more than a purpose.  
  The declaration is the same as the class name, but there is no return type as they are constructors.  
     
  The criteria to overload a constructor is to differ the number of arguments or the type of arguments. The corresponding constructor is called depending on the number and type of arguments passed.  
     
  class smartphone {   
   // Data Members (Properties)   
   string model;   
   int year\_of\_manufacture;   
   bool \_5g\_supported;   
  public:   
   // Constructor with 0 parameter   
   smartphone() {   
   model = "unknown";   
   year\_of\_manufacture = 0;   
   \_5g\_supported = false;   
   }   
   // Constructor with 2 parameters   
   smartphone(string model\_string, bool \_5g\_) {   
   model = model\_string;   
   \_5g\_supported = \_5g\_;   
   }   
   // Constructor with 3 parameters   
   smartphone(string model\_string, int manufacture, bool \_5g\_) {   
   // Initializing data members   
   model = model\_string;   
   year\_of\_manufacture = manufacture;   
   \_5g\_supported = \_5g\_;   
   }  
  };  
  int main() {   
   // Creating objects of smartphone class   
   // Using constructor with 0 parameters   
   smartphone unknown;   
   // Using constructor with 2 parameters   
   smartphone redmi("Note 7 Pro", false);  
   // Using constructor with 3 parameters   
   smartphone iphone("iphone 11", 2019, false);  
  }  
  -------------------------------------------------------------------  
  inline constructor  
  An **inline constructor** in C++ refers to defining a constructor directly within the class definition, which implicitly suggests that the compiler should try to expand the constructor "inline" to optimize performance by avoiding the overhead of a function call.  
     
  #include <iostream>  
  class Rectangle {  
   int width, height;  
     
  public:  
   // Inline constructor defined within the class  
   Rectangle(int w, int h) : width(w), height(h) {}  
   // Member function to calculate the area  
   int area() {  
   return width \* height;  
   }  
  };  
  int main() {  
   Rectangle rect(10, 5);  
   std::cout << "Area: " << rect.area() << std::endl;  
   return 0;  
  }  
  **Key Points:**
  1. **Inline Suggestion**: The constructor Rectangle(int w, int h) is defined within the class body, making it inline by default.
  2. **Optimization**: The inline keyword is a suggestion to the compiler, not a command. The compiler may ignore the request if inlining isn't beneficial (e.g., if the constructor is too complex).
  3. **Efficiency**: Inlining functions can improve efficiency by reducing the function call overhead, though it's typically beneficial for small functions like constructors.
* --------------------------------------------------  
  Copy Constructor:-  
     
  These are a particular type of constructor that takes an object as an argument and copies values of one object’s data members into another object. We pass the class object into another object of the same class in this constructor. As the name suggests, you Copy means to copy the values of one Object into another Object of Class. This is used for Copying the values of a class object into another object of a class, so we call them Copy constructor and for copying the values.  
     
  We have to pass the object’s name whose values we want to copy, and when we are using or passing an object to a constructor, we must use the & ampersand or address operator.  
     
     
  #include <iostream>  
  using namespace std;  
  class Example {  
  public:  
   int x;  
   // Default constructor  
   Example(int a) : x(a) {}  
   // Copy constructor  
   Example(const Example &obj) {  
   x = obj.x;  
   cout << "Copy constructor called" << endl;  
   }  
  };  
  int main() {  
   Example obj1(10); // Calls default constructor  
   Example obj2 = obj1; // Calls copy constructor  
   cout << "obj2.x = " << obj2.x << endl;  
   return 0;  
  }  
  Output  
  Copy constructor called  
  obj2.x = 10  
  **Why Pass by Reference in the Copy Constructor?**Passing by reference is necessary in the copy constructor for several reasons:
  1. **Avoid Infinite Recursion**: If the argument is passed by value (i.e., ClassName old\_obj), it would call the copy constructor again to pass old\_obj to the copy constructor, leading to infinite recursion and a stack overflow. Passing by reference prevents this because it passes a reference to the original object, avoiding another copy constructor call.
  2. **Efficiency**: Passing objects by reference avoids copying the entire object, which can be inefficient, especially for large objects. By passing a reference, you directly use the original object, making it faster and less resource-intensive.
  3. **Const Reference**: The const keyword ensures that the original object won't be modified inside the copy constructor, maintaining the integrity of the original object. This is essential because the purpose of a copy constructor is to make a copy, not modify the original.

Syntax:  
   
class class\_name {  
 int data\_member1;  
 string data\_member2;  
 // copy constructor  
public:  
 class\_name(class\_name& obj) {  
 // copies data of the obj parameter  
 data\_member1 = obj.data\_member1;  
 data\_member2 = obj.data\_member2;  
 }  
};  
In this program,  
 we have used a copy constructor to copy the contents of one object of  
 the class ‘class\_name’ to another.The code of the copy constructor is  
 : class\_name(class\_name& obj) {  
 // copies data of the obj parameter  
 data\_member1 = obj.data\_member1;  
 data\_member2 = obj.data\_member2;  
}  
   
If we don’t define our own copy constructor, the C++ compiler creates a default copy constructor for each class which does a memberwise copy between objects.  
   
Example using smartphone class:  
   
class smartphone {  
 // Data Members(Properties)  
 string model;  
 int year\_of\_manufacture;  
 bool \_5g\_supported;  
public:  
 // Default constructor  
 smartphone() {  
 model = "unknown";  
 year\_of\_manufacture = 0;  
 \_5g\_supported = false;  
 }  
 // Parameterized constructor  
 smartphone(string model\_string, int manufacture, bool \_5g\_) {  
 // Initialising data members  
 model = model\_string;  
 year\_of\_manufacture = manufacture;  
 \_5g\_supported = \_5g\_;  
 }  
 // Copy constructor  
 smartphone(smartphone& obj) {  
 // Copies data of the obj parameter  
 model = obj.model;  
 year\_of\_manufacture = obj.year\_of\_manufacture;  
 \_5g\_supported = obj.\_5g\_supported;  
 }  
};  
int main() {  
 // Creating objects of smartphone class  
 // Using default constructor  
 smartphone unknown;  
 // Using parameterized constructor  
 smartphone iphone("iphone 11", 2019, false);  
 // Using copy constructor  
 smartphone iphone\_2(iphone);  
}  
Value Assignment with Assignment Operator  
int main() 
Customer Al; 
Customer A2("Rohit", 
23, løøø); 
Customer "Mohit" , 25) 
Al. display(); 
A2.disptay( ) ; 
A3. display(); 
Customer A4(A3); 
A4. display( ) ; 
Customer A5; 
1 
A5.disptay(); 
// file open   
--------------------------------------------------