Constructors and Destructors

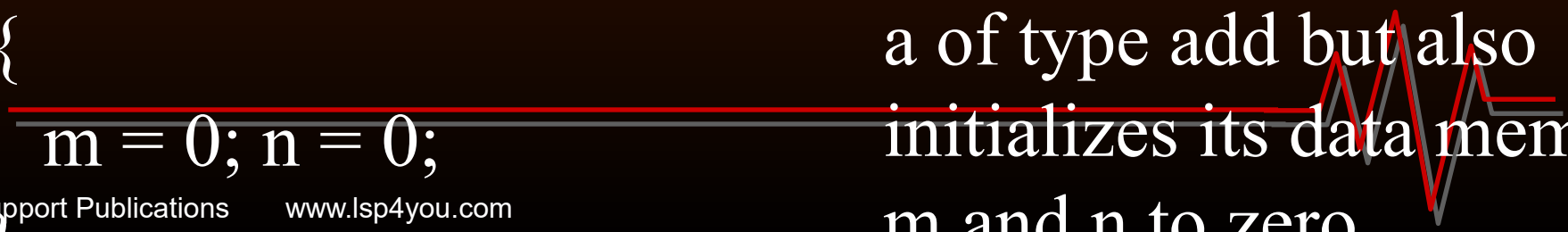
Constructors

- A constructor is a special member function whose task is to initialize the objects of its class.
- It is special because its name is same as the class name.
- The constructor is invoked whenever an object of its associated class is created.
- It is called constructor because it constructs the values of data members of the class.

Constructor - example

```
class add
{
      int m, n ;
   public :
      add (void) ;
      ------
};
add :: add (void)
{
   m = 0; n = 0;
}
```

- When a class contains a constructor, it is guaranteed that an object created by the class will be initialized automatically.
- add a ;
- Not only creates the object a of type add but also initializes its data members m and n to zero.



Constructors

continue …

- There is no need to write any statement to invoke the constructor function.
- If a ‘normal’ member function is defined for zero initialization, we would need to invoke this function for each of the objects separately.
- A constructor that accepts no parameters is called the default constructor.
- The default constructor for class A is A : : A ( )

Characteristics of Constructors

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and they cannot return values.

Characteristics of Constructors

continue …

- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ functions, Constructors can have default arguments.
- Constructors can not be virtual.

Characteristics of Constructors

continue …

- We can not refer to their addresses.
- An object with a constructor (or destructor) can not be used as a member of a union.
- They make ‘implicit calls’ to the operators ***new*** and ***delete*** when memory allocation is required.

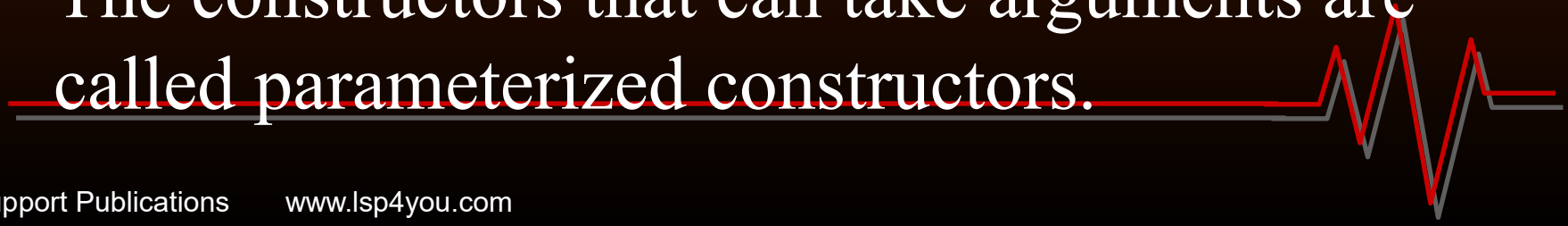
Constructors

continue …

- When a constructor is declared for a class initialization of the class objects becomes mandatory.

Parameterized Constructors

- It may be necessary to initialize the various data elements of different objects with different values when they are created.
- This is achieved by passing arguments to the constructor function when the objects are created.
- The constructors that can take arguments are called parameterized constructors.



Parameterized Constructors

continue …

```
class add
{
      int m, n ;
   public :
      add (int, int) ;
      ------
};
add : : add (int x, int y)
{
  m = x; n = y;
}
```

- When a constructor is parameterized, we must pass the initial values as arguments to the constructor function when an object is declared.
- Two ways Calling:
  - Explicit
    - add sum = add(2,3);
  - Implicit

Multiple Constructors in a Class

- C + + permits to use more than one constructors in a single class.
- Add( ) ; // No arguments
- Add (int, int) ; // Two arguments

Multiple Constructors in a Class

continue …

```
class add
{
     int m, n ;
   public :
     add ( ) {m = 0 ; n = 0
;}
     add (int a, int b)
          {m = a ; n = b ;}
     add (add & i)
          {m = i.m ; n =
i.n ;}
};
```

- The first constructor receives no arguments.
- The second constructor receives two integer arguments.
- The third constructor receives one add object as an argument.

Multiple Constructors in a Class

continue …

```
class add
{
     int m, n ;
   public :
     add ( ) {m = 0 ; n = 0 ;}
     add (int a, int b)
           {m = a ; n = b ;}
     add (add & i)
           {m = i.m ; n = i.n ;}
};
```

- Add a1;
  - Would automatically invoke the first constructor and set both m and n of a1 to zero.
- Add a2(10,20);
  - Would call the second constructor which will initialize the data members m and n of a2 to 10 and 20 respectively.

Multiple Constructors in a Class

continue …

```
class add
{
     int m, n ;
   public :
     add ( ) {m = 0 ; n = 0 ;}
     add (int a, int b)
           {m = a ; n = b ;}
     add (add & i)
           {m = i.m ; n = i.n ;}
};
```

- Add a3(a2);
  - Would invoke the third constructor which copies the values of a2 into a3.
  - This type of constructor is called the “copy constructor”.
- Construction Overloading
  - More than one constructor function is defined in a class.

Multiple Constructors in a Class

continue …

```
class complex
{
     float x, y ;
   public :
     complex ( ) {  }
     complex (float a)
        { x = y = a ; }
     complex (float r, float i)
        { x = r ; y = i }
     ------
};
```

- complex ( ) { }
  - This contains the empty body and does not do anything.
  - This is used to create objects without any initial values.

Multiple Constructors in a Class

continue …

- C + + compiler has an *implicit constructor* which creates objects, even though it was not defined in the class.
- This works well as long as we do not use any other constructor in the class.
- However, once we define a constructor, we must also define the “do-nothing” implicit constructor.

Constructors with Default Arguments

- It is possible to define constructors with default arguments.
- Consider complex (float real, float imag = 0);
  - The default value of the argument imag is zero.
  - complex C1 (5.0) assigns the value 5.0 to the real variable and 0.0 to imag.
  - complex C2(2.0,3.0) assigns the value 2.0 to real and 3.0 to imag.

Constructors with Default Arguments

continue …

- A : : A ( ) → Default constructor
- A : : A (int = 0) → Default argument constructor

- The default argument constructor can be called with either one argument or no arguments.
- When called with no arguments, it becomes a default constructor.

Dynamic Initialization of Objects

- Providing initial value to objects at run time.
- Advantage – We can provide various initialization formats, using overloaded constructors.

  This provides the flexibility of using different format of data at run time depending upon the situation.

Copy Constructor

- A copy constructor is used to declare and initialize an object from another object.

integer (integer & i) ;

integer I 2 (I 1) ; or integer I 2 = I 1 ;

The process of initializing through a copy constructor is known as ***copy initialization***.

Copy Constructor

continue …

The statement

I 2 = I 1;

will not invoke the copy constructor.

If I 1 and I 2 are objects, this statement is legal and assigns the values of I 1 to I 2, member-by-member.

Copy Constructor

continue …

- A reference variable has been used as an argument to the copy constructor.
- We cannot pass the argument by value to a copy constructor.

Dynamic Constructors

- The constructors can also be used to allocate memory while creating objects.
- This will enable the system to allocate the right amount of memory for each object when the objects are not of the same size.

Dynamic Constructors

continue …

- Allocation of memory to objects at the time of their construction is known as dynamic construction of objects.

- The memory is created with the help of the new operator.

Destructors

- A destructor is used to destroy the objects that have been created by a constructor.

- Like constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde.

eg: ~ integer ( ) { }

Destructors

continue …

- A destructor never takes any argument nor does it return any value.

- It will be invoked implicitly by the compiler upon exit from the program – or block or function as the case may be – to clean up storage that is no longer accessible.

Destructors

continue …

- It is a good practice to declare destructors in a program since it releases memory space for further use.

- Whenever ***new*** is used to allocate memory in the constructor, we should use ***delete*** to free that memory.

Thank You