C++ Constructors

By: Dr. Shweta Jain

Head Coordinator, Data Science

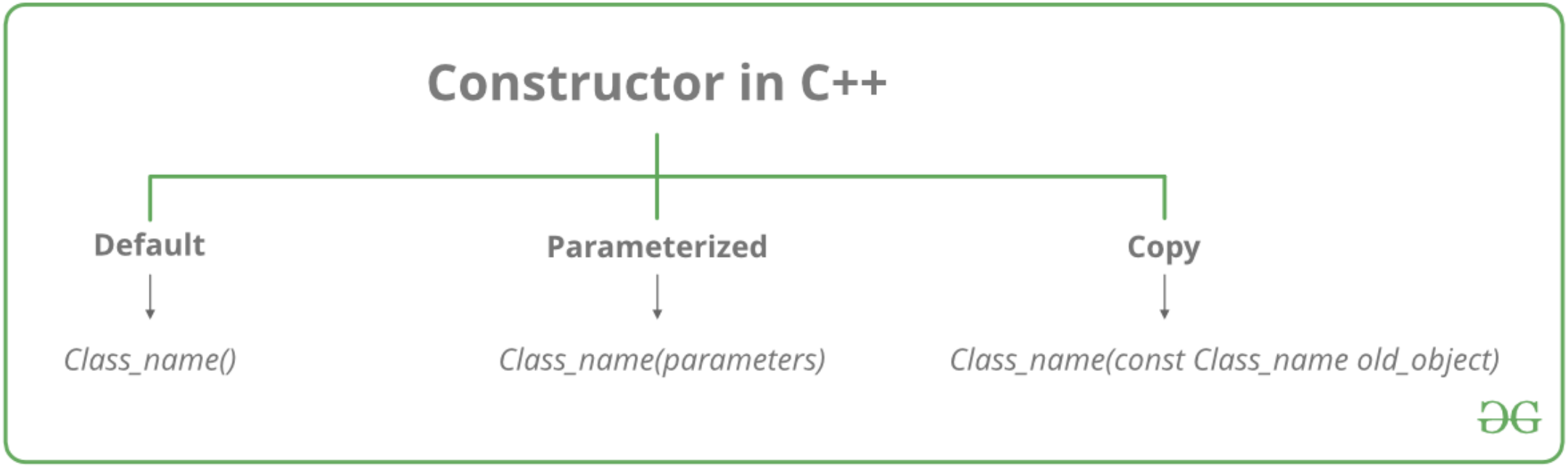
C++ Constructors

- A constructor is a special type of member function that is called automatically when an object is created.

- In C++, a constructor has the same name as that of the class and it does not have a return type. For example:

```
class  Wall {
  public:
    // create a constructor
    Wall() {
      // code
    }
};
```

C++ Constructors



C++ Default Constructor

- A constructor with no parameters is known as a default constructor. In the example above, Wall() is a default constructor.

```
// C++ program to demonstrate the use of default constructor

#include <iostream>
using namespace std;

// declare a class
class  Wall {
  private:
    double length;
public:
    // default constructor to initialize variable
    Wall() {
      length = 5.5;
      cout << "Creating a wall." << endl;
      cout << "Length = " << length << endl;
    }
};

int main() {
  Wall wall1;
  return 0;
}
```

Output:

Creating a Wall
Length = 5.5

C++ Parameterized Constructor

- In C++, a constructor with parameters is known as a parameterized constructor. This is the preferred method to initialize member data.

```
// C++ program to calculate the area of a wall

#include <iostream>
using namespace std;

// declare a class
class Wall {
  private:
    double length;
    double height;

  public:
    // parameterized constructor to initialize variables
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }
    double calculateArea() {
      return length * height;
    }
};

int main() {
  // create object and initialize data members
  Wall wall1(10.5, 8.6);
  Wall wall2(8.5, 6.3);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;
}
```

Output

```
Area of Wall 1: 90.3
Area of Wall 2: 53.55
```

C++ Copy Constructor

- The copy constructor in C++ is used to copy data of one object to another.

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

// declare a class
class Wall {
  private:
    double length;
    double height;

  public:
    // initialize variables with parameterized constructor
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }

    // copy constructor with a Wall object as parameter
    // copies data of the obj parameter
    Wall(Wall &obj) {
      length = obj.length;
      height = obj.height;
    }
    double calculateArea() {
        return length * height;
    }
};

int main() {
  // create an object of Wall class
  Wall wall1(10.5, 8.6);

  // copy contents of wall1 to wall2
  Wall wall2 = wall1;

  // print areas of wall1 and wall2
  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;
}
```

Output:

```
Area of Wall 1: 90.3
Area of Wall 2: 90.3
```

Destructors (C++)

- A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

- A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

- Note: C++ destructor cannot have parameters.

Syntax:

~constructor-name();

Properties of Destructor:

- Destructor function is automatically invoked when the objects are destroyed.
- It cannot be declared static or const.
- The destructor does not have arguments.
- It has no return type not even void.
- An object of a class with a Destructor cannot become a member of the union.
- A destructor should be declared in the public section of the class.
- The programmer cannot access the address of destructor.

When is destructor called?

A destructor function is called automatically when the object goes out of scope:

(1) the function ends

(2) the program ends

(3) a block containing local variables ends

(4) a delete operator is called

How are destructors different from a normal member function?

Destructors have same name as the class preceded by a tilde (~)

Destructors don’t take any argument and don’t return anything

C++ Constructor and Destructor Example

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
       Employee()
       {
          cout<<"Constructor Invoked"<<endl;
       }
```

```
~Employee()
       {
          cout<<"Destructor Invoked"<<endl;
       }
};

int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    return 0;
}
```

OUTPUT:

```
Constructor Invoked
Constructor Invoked
Destructor Invoked
Destructor Invoked
```

When do we need to write a user-defined destructor?

- If we do not write our own destructor in class, **compiler creates a default destructor** for us.
- The default destructor works fine unless we have dynamically allocated memory or **pointer** in class.
- When a class contains a pointer to memory allocated in class, we should write a destructor to release memory before the class instance is destroyed. This must be done to avoid memory leak.