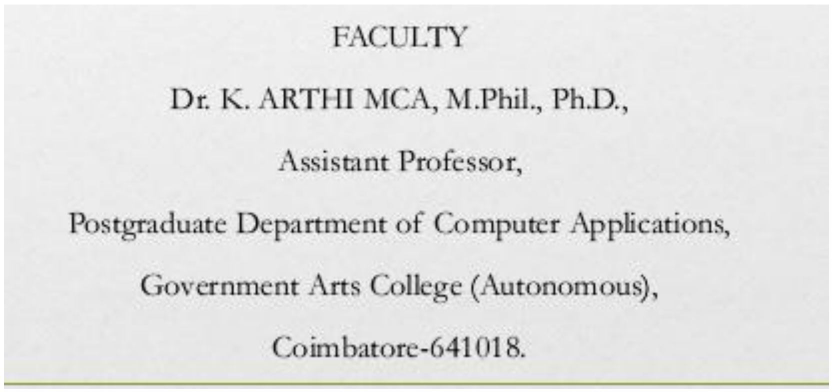
20MCA11C OBJECT ORIENTED PROGRAMMING WITH C++

UNIT II: Functions in C++:


FACULTY

Dr. K. ARTHI MCA, M.Phil., Ph.D.,

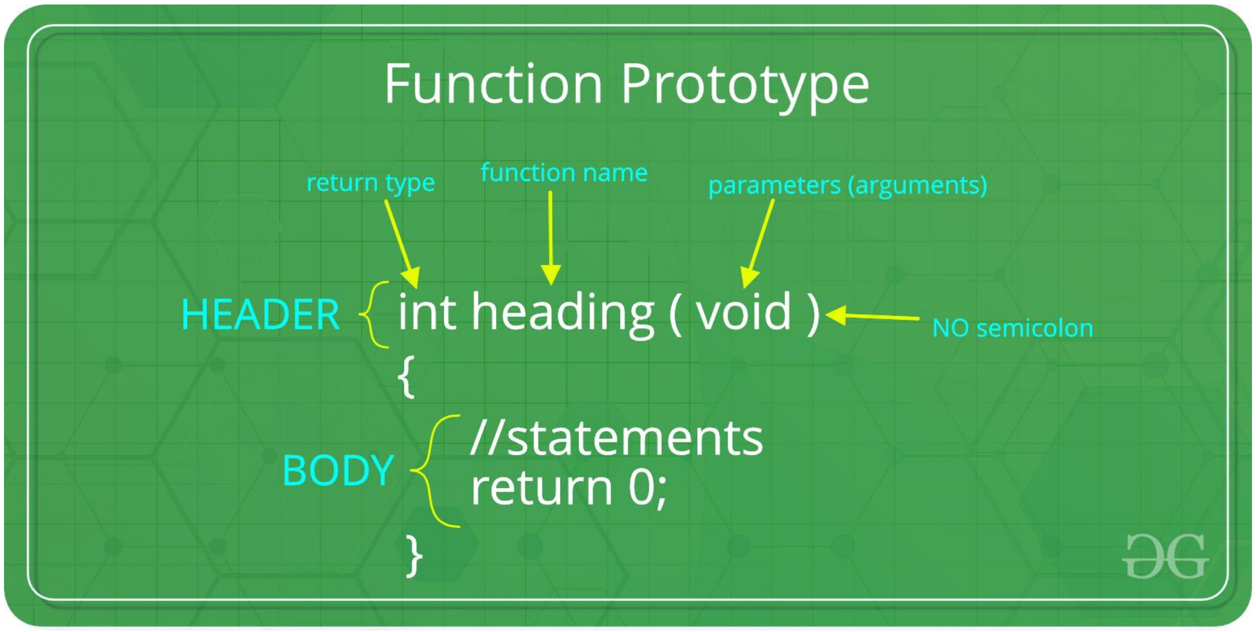
Assistant Professor,

Postgraduate Department of Computer Applications,

Government Arts College (Autonomous),

Coimbatore-641018.

Function Prototyping



Inline functions

C++ **inline** function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.

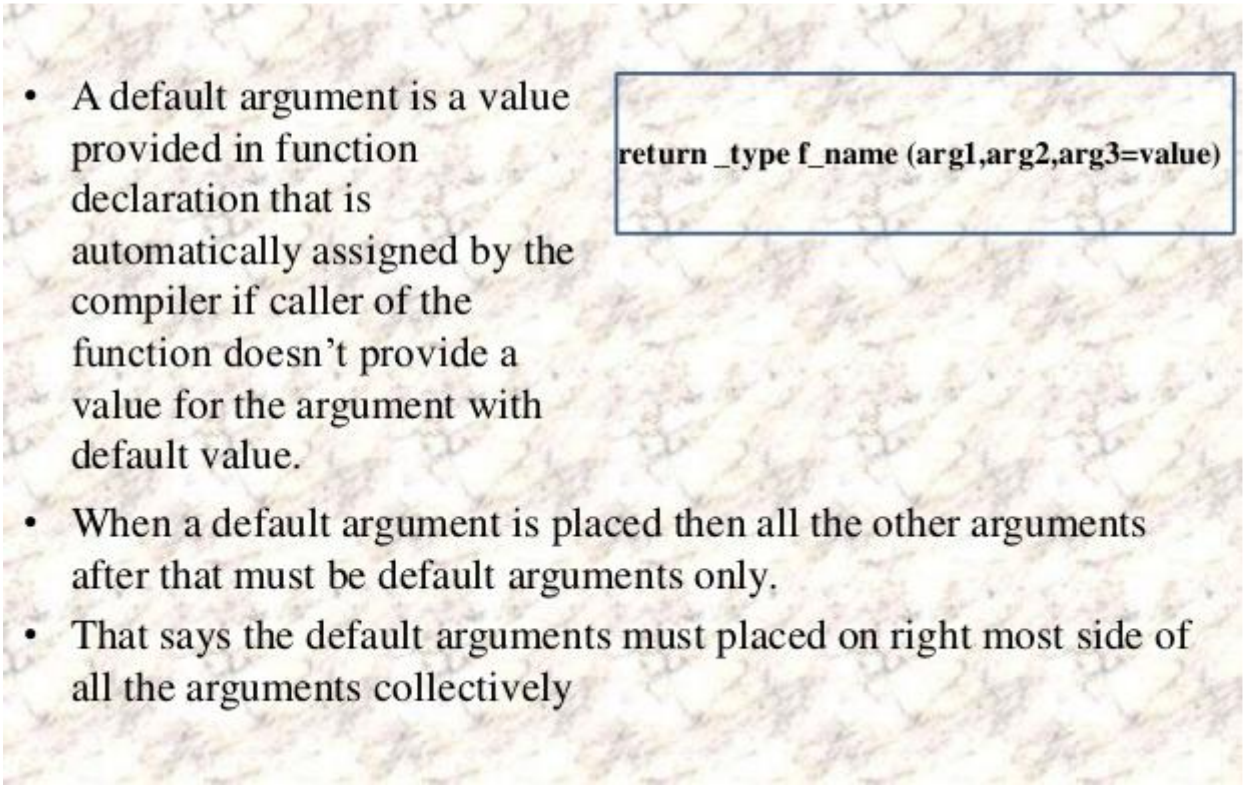
To inline a function, place the keyword **inline** before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

A function definition in a class definition is an inline function definition, even without the use of the **inline** specifier.

Following is an example, which makes use of inline function to return max of two numbers −

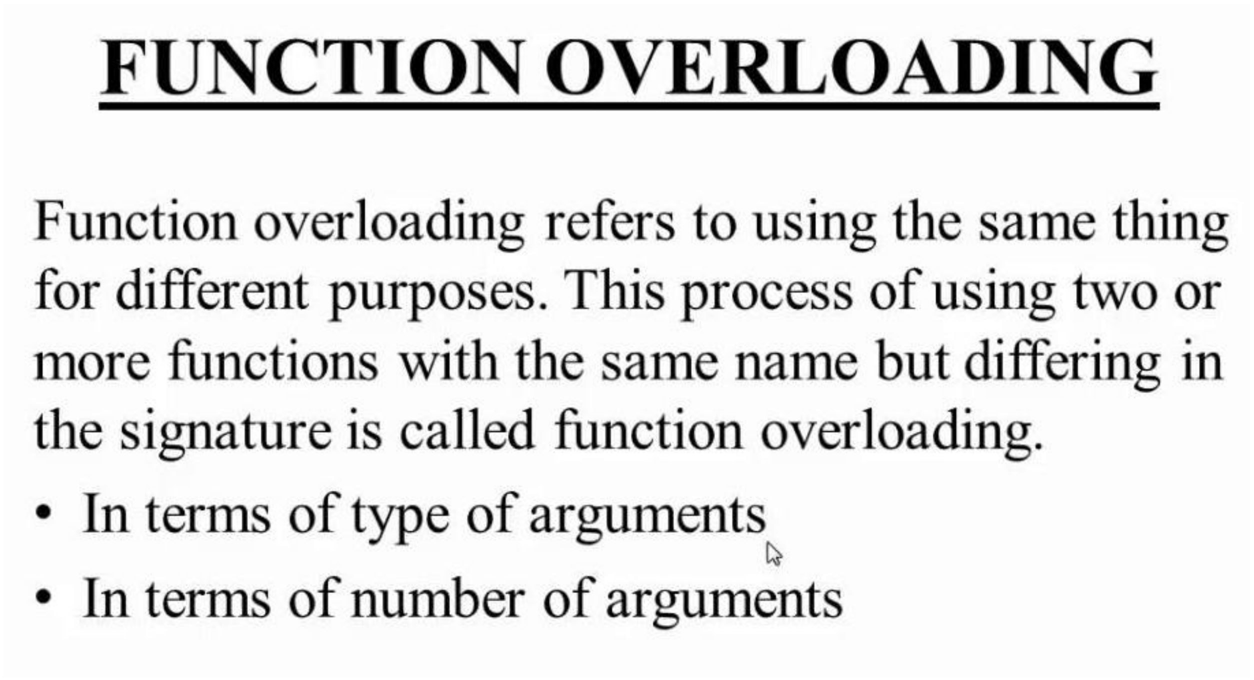
```
#include <iostream>

using namespace std;

inline int Max(int x, int y) {
   return (x > y)? x : y;
}

// Main function for the program
int main() {
  cout << "Max (20,10): " << Max(20,10) << endl;
  cout << "Max (0,200): " << Max(0,200) << endl;
  cout << "Max (100,1010): " << Max(100,1010) << endl;

  return 0;
}
```

DEFAULT ARGUMENTS



- A default argument is a value provided in function declaration that is automatically assigned by the compiler if caller of the function doesn't provide a value for the argument with default value.
- When a default argument is placed then all the other arguments after that must be default arguments only.
- That says the default arguments must placed on right most side of all the arguments collectively

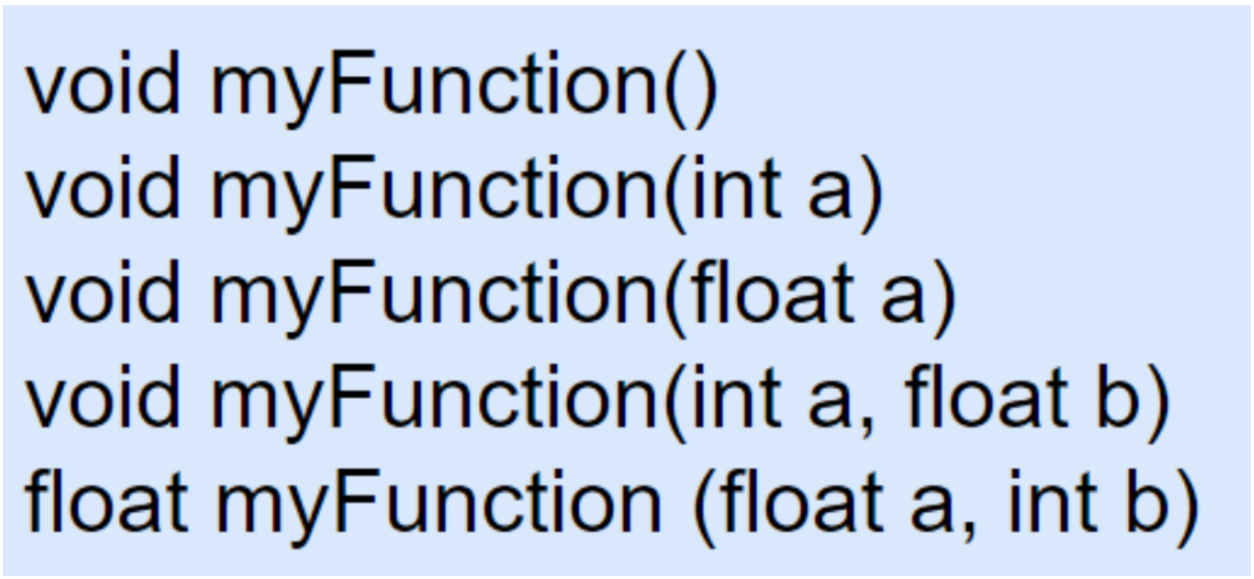
```
return _type f_name (arg1,arg2,arg3=value)
```



FUNCTION OVERLOADING

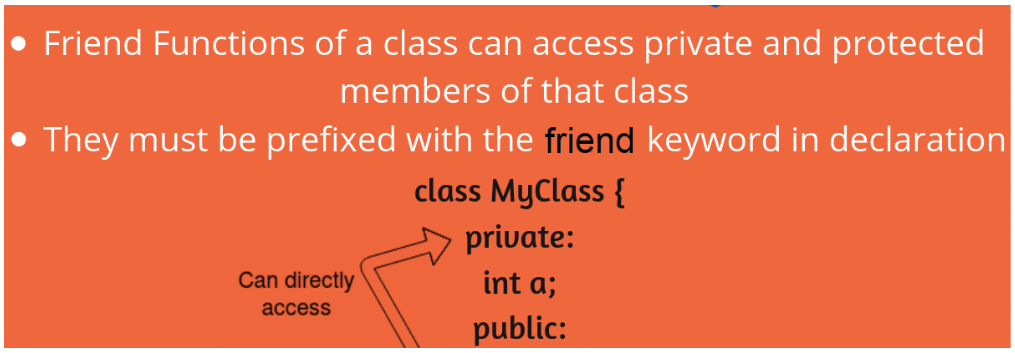
Function overloading refers to using the same thing for different purposes. This process of using two or more functions with the same name but differing in the signature is called function overloading.

- In terms of type of arguments
- In terms of number of arguments



```
void myFunction()
void myFunction(int a)
void myFunction(float a)
void myFunction(int a, float b)
float myFunction (float a, int b)
```

friend function



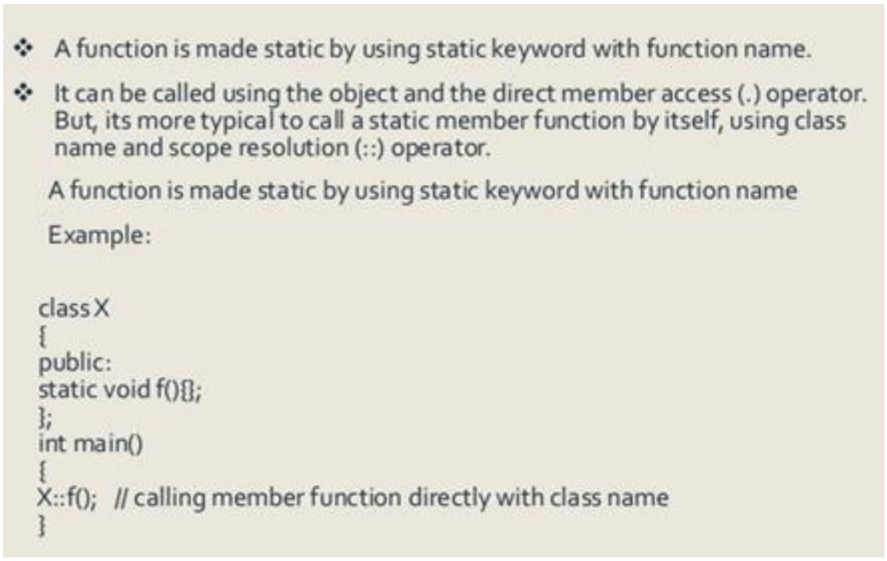
- Friend Functions of a class can access private and protected members of that class
- They must be prefixed with the friend keyword in declaration

```
class MyClass {
  private:
    int a;
  public:
```

Can directly access

- It is not in scope of class.
- It cannot be called using object of that class.
- It can be invoked like a normal function.
- It should use a dot operator for accessing members.
- It can be public or private.
- It has objects as arguments.
- Perhaps the most common use of friend functions is **overloading << and >>** for I/O.

Static Data Members



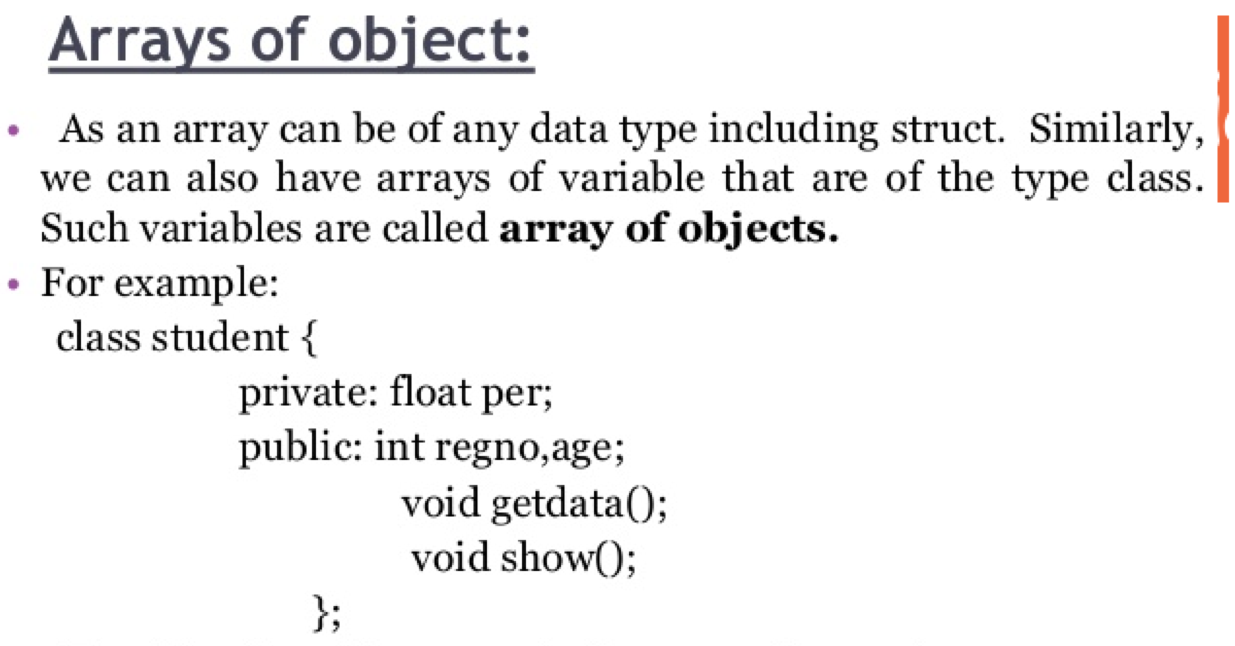
- A function is made static by using static keyword with function name.
- It can be called using the object and the direct member access (.) operator. But, its more typical to call a static member function by itself, using class name and scope resolution (::) operator.

A function is made static by using static keyword with function name

Example:

```
class X
{
public:
static void f(){};
};
int main()
{
X::f();   // calling member function directly with class name
}
```

Array of Objects



Arrays of object:

- As an array can be of any data type including struct. Similarly, we can also have arrays of variable that are of the type class. Such variables are called **array of objects.**
- For example:

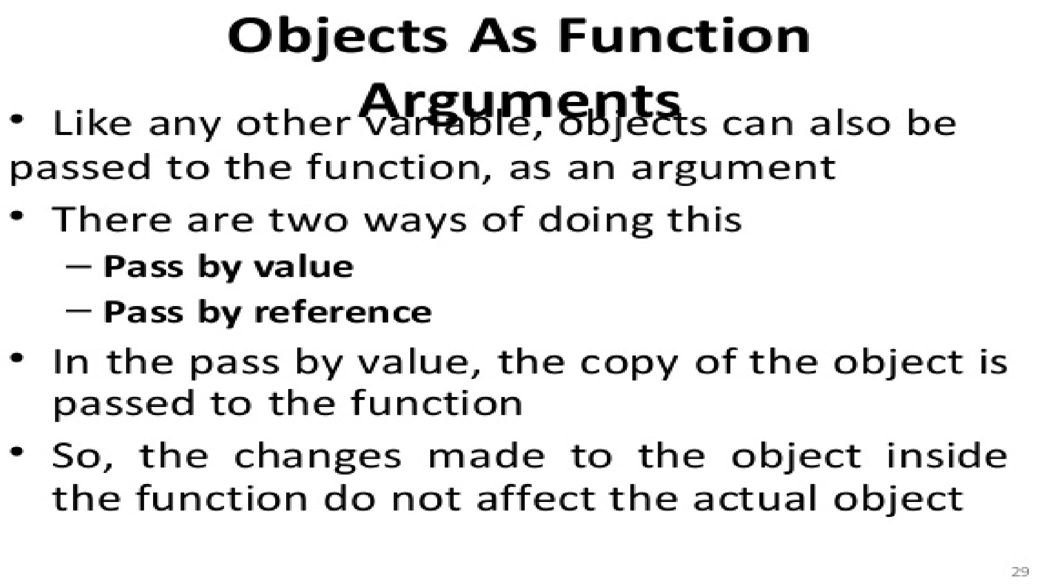
```
class student {
            private: float per;
            public: int regno,age;
                   void getdata();
                    void show();
               };
```

void main()

{

student stu[100];------- represents array of 100 objects

}

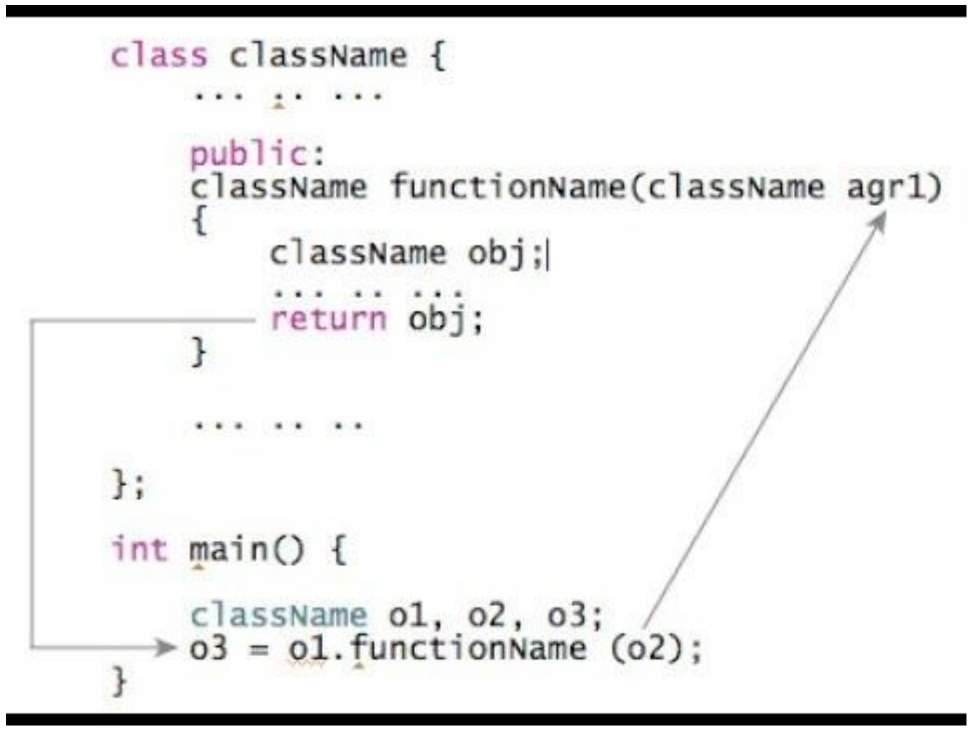


Objects As Function Arguments

- Like any other variable, objects can also be passed to the function, as an argument
- There are two ways of doing this
  - **Pass by value**
  - **Pass by reference**
- In the pass by value, the copy of the object is passed to the function
- So, the changes made to the object inside the function do not affect the actual object

29

Example



```
class className {
    ... .. ...

    public:
    className functionName(className agr1)
    {
        className obj;
        ... .. ...
        return obj;
    }

    ... .. ..

};

int main() {

    className o1, o2, o3;
    o3 = o1.functionName (o2);
}
```

class

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows −

```
class Box {
  public:
     double length;   // Length of a box
     double breadth;  // Breadth of a box
     double height;   // Height of a box
};
```

The keyword **public** determines the access attributes of the members of the class that follows it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as **private** or **protected** which we will discuss in a sub-section.

Define C++ Objects

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box −

```
Box Box1;          // Declare Box1 of type Box
Box Box2;          // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

Accessing the Data Members

The public data members of objects of a class can be accessed using the direct member access operator (.).

POINTER TO MEMBER OPERATOR

1): : * to declare a pointer through of member of class

2)* * To express a member using object name to the member

3) ⟶ ' To access a member using a pointer to the object and a pointer to that member.

THANK YOU

This content is taken from the text books and reference books prescribed in the syllabus.