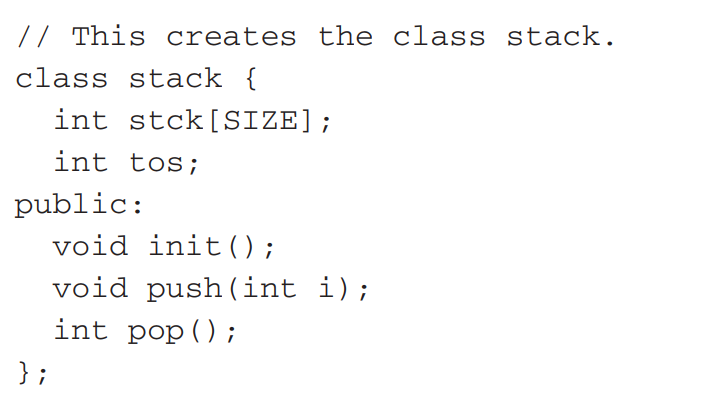
C++ Classes

In C++, to create an object, you first must define its general form by using the keyword class. A class is similar syntactically to a structure. Here is an example. The following class defines a type called stack, which will be used to create a stack:



A class may contain private as well as public parts. By default, all items defined in a class are private. For example, the variables stck and tos are private. This means that they cannot be accessed by any function that is not a member of the class.

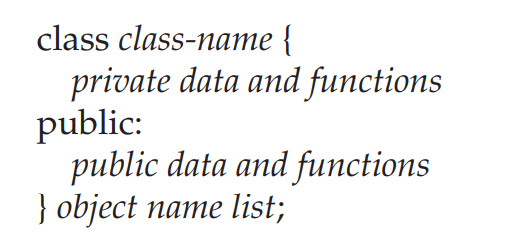
a class public (that is, accessible to other parts of your program), you must declare them after the public keyword. All variables or functions defined after public can be accessed by all other functions in the program.

The functions init( ), push( ), and pop( ) are called member functions because they are part of the class stack. The variables stck and tos are called member variables (or data members). Only member functions have access to the private members of their class. Thus, only init( ), push( ), and pop( ) may access stck and tos.

Once you have defined a class, you can create an object of that type by using the class name. In essence, the class name becomes a new data type specifier. For example, this creates an object called mystack of type stack:

stack mystack;

When you declare an object of a class, you are creating an instance of that class. In this case, mystack is an instance of stack.



**C++ Classes and Objects**

we will learn about objects and classes and how to use them in C++ with the help of examples.

Suppose, we need to store the length, breadth, and height of a rectangular room and calculate its area and volume.

To handle this task, we can create three variables, say, length, breadth, and height along with the functions calculateArea() and calculateVolume().

However, in C++, rather than creating separate variables and functions, we can also wrap these related data and functions in a single place (by creating **objects**). This programming paradigm is known as object-oriented programming.  
  
But before we can create **objects** and use them in C++, we first need to learn about **classes**.

**C++ Class**

A class is a blueprint for the object.  
  
We can think of a class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

**Create a Class**

A class is defined in C++ using keyword class followed by the name of the class.

The body of the class is defined inside the curly brackets and terminated by a semicolon at the end.

class className {

// data

// functions

};

For example,

class Room {

public:

double length;

double breadth;

double height;

double calculateArea(){

return length \* breadth;

}

double calculateVolume(){

return length \* breadth \* height;

}

};

Here, we defined a class named Room.

The variables length, breadth, and height declared inside the class are known as **data members**. And, the functions calculateArea() and calculateVolume() are known as **member functions** of a class.

**C++ Objects**

When a class is defined, only the specification for the object is defined; no memory or storage is allocated.

To use the data and access functions defined in the class, we need to create objects.

**Syntax to Define Object in C++**

className objectVariableName;

We can create objects of Room class (defined in the above example) as follows:

// sample function

void sampleFunction() {

// create objects

Room room1, room2;

}

int main(){

// create objects

Room room3, room4;

}

Here, two objects room1 and room2 of the Room class are created in sampleFunction(). Similarly, the objects room3 and room4 are created in main().

As we can see, we can create objects of a class in any function of the program. We can also create objects of a class within the class itself, or in other classes.

Also, we can create as many objects as we want from a single class.

**C++ Access Data Members and Member Functions**

We can access the data members and member functions of a class by using a . (dot) operator. For example,

room2.calculateArea();

This will call the calculateArea() function inside the Room class for object room2.

Similarly, the data members can be accessed as:

room1.length = 5.5;

In this case, it initializes the length variable of room1 to 5.5.

**Example 1: Object and Class in C++ Programming**

// Program to illustrate the working of

// objects and class in C++ Programming

#include <iostream>

using namespace std;

// create a class

class Room {

public:

double length;

double breadth;

double height;

double calculateArea() {

return length \* breadth;

}

double calculateVolume() {

return length \* breadth \* height;

}

};

int main() {

// create object of Room class

Room room1;

// assign values to data members

room1.length = 42.5;

room1.breadth = 30.8;

room1.height = 19.2;

// calculate and display the area and volume of the room

cout << "Area of Room = " << room1.calculateArea() << endl;

cout << "Volume of Room = " << room1.calculateVolume() << endl;

return 0;

}

**Output**

Area of Room = 1309

Volume of Room = 25132.8

In this program, we have used the Room class and its object room1 to calculate the area and volume of a room.

In main(), we assigned the values of length, breadth, and height with the code:

room1.length = 42.5;

room1.breadth = 30.8;

room1.height = 19.2;

We then called the functions calculateArea() and calculateVolume() to perform the necessary calculations.

Note the use of the keyword public in the program. This means the members are public and can be accessed anywhere from the program.

As per our needs, we can also create private members using the private keyword. The private members of a class can only be accessed from within the class. For example,

class Test {

private:

int a;

void function1() { }

public:

int b;

void function2() { }

}

Here, a and function1() are private. Thus they cannot be accessed from outside the class.

On the other hand, b and function2() are accessible from everywhere in the program.

**Example 2: Using public and private in C++ Class**

// Program to illustrate the working of

// public and private in C++ Class

#include <iostream>

using namespace std;

class Room {

private:

double length;

double breadth;

double height;

public:

// function to initialize private variables

void initData(double len, double brth, double hgt) {

length = len;

breadth = brth;

height = hgt;

}

double calculateArea() {

return length \* breadth;

}

double calculateVolume() {

return length \* breadth \* height;

}

};

int main() {

// create object of Room class

Room room1;

// pass the values of private variables as arguments

room1.initData(42.5, 30.8, 19.2);

cout << "Area of Room = " << room1.calculateArea() << endl;

cout << "Volume of Room = " << room1.calculateVolume() << endl;

return 0;

}

**Output**

Area of Room = 1309

Volume of Room = 25132.8

The above example is nearly identical to the first example, except that the class variables are now private.

Since the variables are now private, we cannot access them directly from main(). Hence, using the following code would be invalid:

// invalid code

obj.length = 42.5;

obj.breadth = 30.8;

obj.height = 19.2;

Instead, we use the public function initData() to initialize the private variables via the function parameters double len, double brth, and double hgt.

Stack:

#include <iostream>

using namespace std;

#define SIZE 100

// This creates the class stack.

class stack {

int stck[SIZE];

int tos;

public:

void init();

void push(int i);

int pop();

};

void stack::init()

{

tos = 0;

}

void stack::push(int i)

{

if(tos==SIZE) {

cout << "Stack is full.\n";

return;

}

stck[tos] = i;

tos++;

}

int stack::pop()

{

if(tos==0)

{

cout << "Stack underflow.\n";

return 0;

}

tos--;

return stck[tos];

}

int main()

{

stack stack1, stack2; // create two stack objects

stack1.init();

stack2.init();

stack1.push(1);

stack2.push(2);

stack1.push(3);

stack2.push(4);

cout << stack1.pop() << " ";

cout << stack1.pop() << " ";

cout << stack2.pop() << " ";

cout << stack2.pop() << "\n";

return 0;

}