C++ Object and Class

Since C++ is an object-oriented language, program is designed using objects and classes in C++.

C++ Object

In C++, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.

In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.

Object is a runtime entity, it is created at runtime.

Object is an instance of a class. All the members of the class can be accessed through object.

Let's see an example to create object of student class using s1 as the reference variable.

1. Student s1;  //creating an object of Student

In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class.

C++ Class

In C++, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C++ class that has three fields only.

1. **class** Student
2. {
3. **public**:
4. **int** id;  //field or data member
5. **float** salary; //field or data member
6. String name;//field or data member
7. }

C++ Object and Class Example

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Student {
4. **public**:
5. **int** id;//data member (also instance variable)
6. string name;//data member(also instance variable)
7. };
8. **int** main() {
9. Student s1; //creating an object of Student
10. s1.id = 201;
11. s1.name = "Sonoo Jaiswal";
12. cout<<s1.id<<endl;
13. cout<<s1.name<<endl;
14. **return** 0;
15. }

Output:

201

SonooJaiswal

C++ Class Example: Initialize and Display data through method

Let's see another example of C++ class where we are initializing and displaying object through method.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Student {
4. **public**:
5. **int** id;//data member (also instance variable)
6. string name;//data member(also instance variable)
7. **void** insert(**int** i, string n)
8. {
9. id = i;
10. name = n;
11. }
12. **void** display()
13. {
14. cout<<id<<"  "<<name<<endl;
15. }
16. };
17. **int** main(**void**) {
18. Student s1; //creating an object of Student
19. Student s2; //creating an object of Student
20. s1.insert(201, "Sonoo");
21. s2.insert(202, "Nakul");
22. s1.display();
23. s2.display();
24. **return** 0;
25. }

Output:

201 Sonoo

202 Nakul

C++ Class Example: Store and Display Employee Information

Let's see another example of C++ class where we are storing and displaying employee information using method.

1. #include <iostream>
2. **using** **namespace** std;
3. **class** Employee {
4. **public**:
5. **int** id;//data member (also instance variable)
6. string name;//data member(also instance variable)
7. **float** salary;
8. **void** insert(**int** i, string n, **float** s)
9. {
10. id = i;
11. name = n;
12. salary = s;
13. }
14. **void** display()
15. {
16. cout<<id<<"  "<<name<<"  "<<salary<<endl;
17. }
18. };
19. **int** main(**void**) {
20. Employee e1; //creating an object of Employee
21. Employee e2; //creating an object of Employee
22. e1.insert(201, "Sonoo",990000);
23. e2.insert(202, "Nakul", 29000);
24. e1.display();
25. e2.display();
26. **return** 0;
27. }

Output:

201 Sonoo 990000

202 Nakul 29000

# Difference between Structure and Class in C++

In C++, the structure is the same as the class with some differences. Security is the most important thing for both structure and class. A structure is not safe because it could not hide its implementation details from the end-user, whereas a class is secure as it may hide its programming and design details. In this article, we are going to discuss the difference between a structure and class in [C++](https://www.javatpoint.com/cpp-tutorial). But before discussing the differences, we will know about the structure and class in C++.

## What is the structure in C++?

A structure is a **grouping** of variables of various [**data types**](https://www.javatpoint.com/cpp-data-types) referenced by the same name. A structure declaration serves as a template for creating an instance of the structure.

### Syntax:

The syntax of the structure is as follows:

1. Struct Structurename
2. {
3. Struct\_member1;
4. Struct\_member2;
5. Struct\_member3;
6. .
7. .
8. .
9. Struct\_memberN;
10. };

The **"struct"** keyword indicates to the compiler that a structure has been declared. The **"structurename"** defines the name of the structure. Since the structure declaration is treated as a statement, so it is often ended by a semicolon.

HTML Tutorial

## What is Class in C++?

A class in C++ is similar to a C structure in that it consists of a list of **data members** and a set of operations performed on the class. In other words, a class is the **building block** of Object-Oriented programming. It is a user-defined object type with its own set of data members and member functions that can be accessed and used by creating a class instance. A C++ class is similar to an object's blueprint.

### Syntax:

The structure and the class are syntactically similar. The syntax of class in C++ is as follows:

1. **class** class\_name
2. {
3. // private data members and member functions.
4. Access specifier;
5. Data member;
6. Member functions (member list){ . . }
7. };

In this syntax, the class is a keyword to indicate the compiler that a class has been declared. OOP's main function is data hiding, which is achieved by having three access specifiers: **"public", "private"**, and **"safe"**. If no access specifier is specified in the class when declaring data members or member functions, they are all considered private by default.

The public access specifier allows others to access program functions or data. A member of that class may reach only the class's private members. During inheritance, the safe access specifier is used. If the access specifier is declared, it cannot be changed again in the program.

## Head-to-head comparison between the structure and class

Here, we are going to discuss a head-to-head comparison between the structure and class. Some of them are as follows:

|  |  |  |
| --- | --- | --- |
| **Features** | **Structure** | **Class** |
| Definition | A structure is a grouping of variables of various data types referenced by the same name. | In C++, a class is defined as a collection of related variables and functions contained within a single structure. |
| Basic | If no access specifier is specified, all members are set to 'public'. | If no access specifier is defined, all members are set to 'private'. |
| Declaration | structstructure\_name{  type struct\_member 1;  type struct\_member 2;  type struct\_member 3;  .  type struct\_memberN;  }; | class class\_name{  data member;  member function;  }; |
| Instance | Structure instance is called the 'structure variable'. | A class instance is called 'object'. |
| Inheritance | It does not support inheritance. | It supports inheritance. |
| Memory Allocated | Memory is allocated on the stack. | Memory is allocated on the heap. |
| Nature | Value Type | Reference Type |
| Purpose | Grouping of data | Data abstraction and further inheritance. |
| Usage | It is used for smaller amounts of data. | It is used for a huge amount of data. |
| Null values | Not possible | It may have null values. |
| Requires constructor and destructor | It may have only parameterized constructor. | It may have all the types of constructors and destructors. |

## Similarities

The following are similarities between the structure and class:

* Both class and structure may declare any of their members private.
* Both class and structure support inheritance mechanisms.
* Both class and structure are syntactically identical in C++.
* A class's or structure's name may be used as a stand-alone type.

## Conclusion

Structure in C has some limitations, such as the inability to hide data, the inability to treat 'struct' data as built-in types, and the lack of inheritance support. The C++ structure overcame these drawbacks.

The extended version of the structure in C++ is called a class. The programmer makes it easy to use the class to hold both the data and functions, whereas the structure only holds data.

# C++ Access Specifiers

The access restriction to the class members is specified by the labeled **public, private,** and **protected** sections within the class body. The keywords public, private, and protected are called access specifiers.

A class can have multiple public, protected, or private labeled sections. Each section remains in effect until either another section label or the closing right brace of the class body is seen. The default access for members and classes is private.

class Base {

public:

// public members go here

protected:

// protected members go here

private:

// private members go here

};

The public Members

A **public** member is accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function as shown in the following example −

[Live Demo](http://tpcg.io/eLDaBs)

#include<iostream>

usingnamespace std;

classLine{

public:

double length;

voidsetLength(doublelen);

doublegetLength(void);

};

// Member functions definitions

doubleLine::getLength(void){

return length ;

}

voidLine::setLength(doublelen){

length=len;

}

// Main function for the program

int main(){

Lineline;

// set line length

line.setLength(6.0);

cout<<"Length of line : "<<line.getLength()<<endl;

// set line length without member function

line.length=10.0;// OK: because length is public

cout<<"Length of line : "<<line.length<<endl;

return0;

}

When the above code is compiled and executed, it produces the following result −

Length of line : 6

Length of line : 10

The private Members

A **private** member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.

By default all the members of a class would be private, for example in the following class **width** is a private member, which means until you label a member, it will be assumed a private member −

class Box {

double width;

public:

double length;

voidsetWidth( double wid );

doublegetWidth( void );

};

Practically, we define data in private section and related functions in public section so that they can be called from outside of the class as shown in the following program.

[Live Demo](http://tpcg.io/OJaPdZ)

#include<iostream>

usingnamespace std;

classBox{

public:

double length;

voidsetWidth(doublewid);

doublegetWidth(void);

private:

double width;

};

// Member functions definitions

doubleBox::getWidth(void){

return width ;

}

voidBox::setWidth(doublewid){

width=wid;

}

// Main function for the program

int main(){

Boxbox;

// set box length without member function

box.length=10.0;// OK: because length is public

cout<<"Length of box : "<<box.length<<endl;

// set box width without member function

// box.width = 10.0; // Error: because width is private

box.setWidth(10.0);// Use member function to set it.

cout<<"Width of box : "<<box.getWidth()<<endl;

return0;

}

When the above code is compiled and executed, it produces the following result −

Length of box : 10

Width of box : 10

The protected Members

A **protected** member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

You will learn derived classes and inheritance in next chapter. For now you can check following example where I have derived one child class **SmallBox** from a parent class **Box**.

Following example is similar to above example and here **width** member will be accessible by any member function of its derived class SmallBox.

[Live Demo](http://tpcg.io/wVR5ie)

#include<iostream>

usingnamespace std;

classBox{

protected:

double width;

};

classSmallBox:Box{// SmallBox is the derived class.

public:

voidsetSmallWidth(doublewid);

doublegetSmallWidth(void);

};

// Member functions of child class

doubleSmallBox::getSmallWidth(void){

return width ;

}

voidSmallBox::setSmallWidth(doublewid){

width=wid;

}

// Main function for the program

int main(){

SmallBox box;

// set box width using member function

box.setSmallWidth(5.0);

cout<<"Width of box : "<<box.getSmallWidth()<<endl;

return0;

}

When the above code is compiled and executed, it produces the following result −

Width of box : 5

# C++ Scope resolution operator

The scope resolution operator ( :: ) is used for several reasons. For example: If the global variable name is same as local variable name, the scope resolution operator will be used to call the global variable. It is also used to define a function outside the class and used to access the static variables of class.

Here an example of scope resolution operator in C++ language,

## Example

 Live Demo

#include<iostream>

usingnamespace std;

char a ='m';

static int b =50;

int main(){

   char a ='s';

   cout<<"The static variable : "<<::b;

   cout<<"\nThe local variable : "<< a;

   cout<<"\nThe global variable : "<<::a;

   return0;

}

## Output

Here is the output

The static variable : 50

The local variable : s

The global variable : m

**Member functions**can be defined within the class definition or separately using **scope resolution operator, :** −. Defining a member function within the class definition declares the function **inline**, even if you do not use the inline specifier. So either you can define **Volume()** function as below −

class Box {

public:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

doublegetVolume(void) {

return length \* breadth \* height;

}

};

If you like, you can define the same function outside the class using the **scope resolution operator** (::) as follows −

double Box::getVolume(void) {

return length \* breadth \* height;

}

Here, only important point is that you would have to use class name just before :: operator. A member function will be called using a dot operator (**.**) on a object where it will manipulate data related to that object only as follows −

Box myBox; // Create an object

myBox.getVolume(); // Call member function for the object

Let us put above concepts to set and get the value of different class members in a class −

Live Demo

#include<iostream>

usingnamespace std;

classBox{

public:

double length;// Length of a box

double breadth;// Breadth of a box

double height;// Height of a box

// Member functions declaration

doublegetVolume(void);

voidsetLength(doublelen);

voidsetBreadth(doublebre);

voidsetHeight(doublehei);

};

// Member functions definitions

doubleBox::getVolume(void){

return length \* breadth \* height;

}

voidBox::setLength(doublelen){

length=len;

}

voidBox::setBreadth(doublebre){

breadth=bre;

}

voidBox::setHeight(doublehei){

height=hei;

}

// Main function for the program

int main(){

BoxBox1;// Declare Box1 of type Box

BoxBox2;// Declare Box2 of type Box

double volume =0.0;// Store the volume of a box here

// box 1 specification

Box1.setLength(6.0);

Box1.setBreadth(7.0);

Box1.setHeight(5.0);

// box 2 specification

Box2.setLength(12.0);

Box2.setBreadth(13.0);

Box2.setHeight(10.0);

// volume of box 1

volume=Box1.getVolume();

cout<<"Volume of Box1 : "<< volume <<endl;

// volume of box 2

volume=Box2.getVolume();

cout<<"Volume of Box2 : "<< volume <<endl;

return0;

}

When the above code is compiled and executed, it produces the following result −

Volume of Box1 : 210

Volume of Box2 : 1560

**Union**

Union is a user-defined datatype. All the members of union share same memory location. Size of union is decided by the size of largest member of union. If you want to use same memory location for two or more members, union is the best for that.

Unions are similar to structures. Union variables are created in same manner as structure variables. The keyword “union” is used to define unions in C language.

Here is the syntax of unions in C language,

unionunion\_name {

   member definition;

} union\_variables;

Here,

* **union\_name** − Any name given to the union.
* **member definition** − Set of member variables.
* **union\_variable** − This is the object of union.

Here is an example of unions in C language,

## Example

 Live Demo

#include<stdio.h>

#include<string.h>

unionData{

   inti;

   float f;

} data, data1;

int main(){

   printf("Memory size occupied by data : %d\t%d",sizeof(data),sizeof(data1));

   return0;

}

## Output

Here is the output

Memory size occupied by data : 4 4

Difference between Structure and Union

Let's summarize the above discussed topic about the Struct and Union in the form of a table that highlight the differences between structure and union:

|  |  |
| --- | --- |
| **Struct** | **Union** |
| The struct keyword is used to define a structure. | The union keyword is used to define union. |
| When the variables are declared in a structure, the compiler allocates memory to each variables member. The size of a structure is equal or greater to the sum of the sizes of each data member. | When the variable is declared in the union, the compiler allocates memory to the largest size variable member. The size of a union is equal to the size of its largest data member size. |
| Each variable member occupied a unique memory space. | Variables members share the memory space of the largest size variable. |
| Changing the value of a member will not affect other variables members. | Changing the value of one member will also affect other variables members. |
| Each variable member will be assessed at a time. | Only one variable member will be assessed at a time. |
| We can initialize multiple variables of a structure at a time. | In union, only the first data member can be initialized. |
| All variable members store some value at any point in the program. | Exactly only one data member stores a value at any particular instance in the program. |
| The structure allows initializing multiple variable members at once. | Union allows initializing only one variable member at once. |
| It is used to store different data type values. | It is used for storing one at a time from different data type values. |
| It allows accessing and retrieving any data member at a time. | It allows accessing and retrieving any one data member at a time. |

C++ Enumeration

Enum in C++ is a data type that contains fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The C++ enum constants are static and final implicitly.

C++ Enums can be thought of as classes that have fixed set of constants.

Points to remember for C++ Enum

* enum improves type safety
* enum can be easily used in switch
* enum can be traversed
* enum can have fields, constructors and methods
* enum may implement many interfaces but cannot extend any class because it internally extends Enum class

C++ Enumeration Example

Let's see the simple example of enum data type used in C++ program.

13. PHP Unit testing tutorial and pages list | Automated testing with PHPUnit | CMS OOP tutorial MVC

1. #include <iostream>
2. **using** **namespace** std;
3. **enum** week { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };
4. **int** main()
5. {
6. week day;
7. day = Friday;
8. cout << "Day: " << day+1<<endl;
9. **return** 0;
10. }

Output:

Day: 5