**Classes and Objects**

# Class

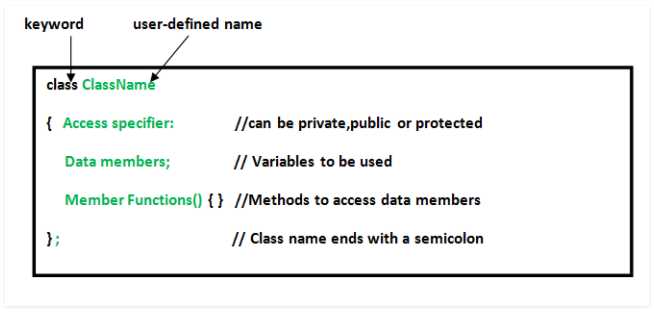
It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

An Object is an instance of a Class. When a class is defined, no memory is allocated\*\* but when it is instantiated (i.e. an object is created) memory is allocated.

\*\* only partly true (see [Memory allocation for objects](#_Memory_allocation_for))

# Defining Class

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



# Declaring 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, you need to create objects.

Syntax:

ClassName ObjectName;

# Accessing data members and member functions

The data members and member functions of class can be accessed using the dot(‘.’) operator with the object.

For example if the name of object is obj and you want to access the member function with the name printName() then you will have to write obj.printName().

# Accessing Data Members

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by Access modifiers in C++. There are three access modifiers : public, private and protected.

// C++ program to demonstrate accessing of data members

#include <bits/stdc++.h>

using namespace std;

class Geeks {

public: // Access specifier

string geekname; // Data Members

void printname() { // Member Functions()

cout << "Geekname is: " << geekname;

}

};

int main() {

Geeks obj1; // Declare an object of class geeks

obj1.geekname = "Abhi"; // accessing data member

obj1.printname(); // accessing member function

return 0;

}

Output:

Geekname is: Abhi

# Member Functions in Classes

There are 2 ways to define a member function:

1. Inside class definition
2. Outside class definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.

// C++ program to demonstrate function declaration outside class

#include <bits/stdc++.h>

using namespace std;

class Geeks {

public:

string geekname;

int id;

void printname(); // printname is not defined inside class defination

void printid() { // printid is defined inside class defination

cout << "Geek id is: " << id;

}

};

// Definition of printname using scope resolution operator ::

void Geeks::printname() {

cout << "Geekname is: " << geekname;

}

int main() {

Geeks obj1;

obj1.geekname = "xyz";

obj1.id=15;

obj1.printname(); cout << endl;

obj1.printid(); cout << endl;

return 0;

}

Output:

Geekname is: xyz

Geek id is: 15

Note that all the member functions defined inside the class definition are by default inline, but you can also make any non-class function inline by using keyword inline with them. Inline functions are actual functions, which are copied everywhere during compilation, like pre-processor macro, so the overhead of function calling is reduced.

Note: Declaring a friend function is a way to give private access to a non-member function.

# Const member functions in C++

A function becomes const when const keyword is used in function’s declaration.

The idea of const functions is not allow them to modify the object on which they are called.

It is recommended practice to make as many functions const as possible so that accidental changes to objects are avoided.

A const object can call only const member functions.

#include<iostream>

using namespace std;

class Test {

int value;

public:

Test(int v = 0) {value = v;} // constructor

// We get compiler error if we add a line like "value = 100;" in this function.

int constfun() const {

//value = 50; // 1\*

return value;

}

int fun() {

value = 30;

return value;

}

};

int main() {

Test obj(20);

const Test constobj(70);

cout << obj.constfun() << endl;

cout << obj.fun() << endl;

cout << endl;

cout << constobj.constfun() << endl;

//cout << constobj.fun() << endl; // 2\*

return 0;

}

Output:

20

30

70

// 1\*

// CE: assignment of member 'Test::value' in read-only object

// 2\*

// cppmain.cpp:33:23: error: passing 'const Test' as 'this' argument discards qualifiers [-fpermissive]

// cout << constobj.fun() << endl;

// ^

# Constructors

Constructors are special class members which are called by the compiler every time an object of that class is instantiated. Constructors have the same name as the class and may be defined inside or outside the class definition.

There are 3 types of constructors:

1. Default constructors
2. Parametrized constructors
3. Copy constructors

A Copy Constructor creates a new object, which is exact copy of the existing copy. The compiler provides a default Copy Constructor to all the classes.

Syntax:

class-name (class-name &){}

# Destructors

Destructor is another special member function that is called by the compiler when the scope of the object ends.

// C++ program to demonstrate constructors

#include <bits/stdc++.h>

using namespace std;

class Geeks {

public:

int id;

//Default Constructor

Geeks() {

cout << "Default Constructor called" << endl;

id=-1;

}

//Parametrized Constructor

Geeks(int x) {

cout << "Parametrized Constructor called" << endl;

id=x;

}

~Geeks() {

cout << "Destructor called for id: " << id <<endl;

}

};

int main() {

int i = 0;

Geeks obj1; // obj1 will call Default Constructor

cout << "Geek id is: " <<obj1.id << endl;

Geeks obj2(21); // obj1 will call Parametrized Constructor

cout << "Geek id is: " <<obj2.id << endl;

cout << endl;

while ( i < 5 ) {

Geeks obj3;

obj3.id=i;

i++;

} // Scope for obj3 ends here

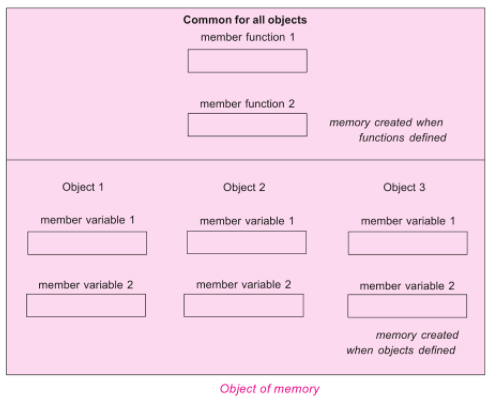
return 0;

} // Scope for obj1, obj2 ends here

# Memory allocation for objects

When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. This statement is only partially true.

Actually, the member functions are created and placed in the memory space only once when they are defined in class specification. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created. Only space for member variable is allocated separately for each object because member variable holds different value for different objects.



# Size of an empty class

**Why is the size of an empty class not zero in C++?**

Size of an empty class is not 0. It is 1 byte generally.

It is nonzero to ensure that the two different objects will have different addresses.

For the same reason (different objects should have different addresses), “new” always returns pointers to distinct objects.

#include<iostream>

using namespace std;

class Empty { };

class Derived: Empty { int a; };

int main() {

Empty a, b;

cout << "size of Empty : "<< sizeof(Empty) << endl;

cout << "size of Derived : "<< sizeof(Derived) << endl;

if (&a == &b)

cout << "impossible " << endl;

else

cout << "Fine " << endl;

return 0;

}

Output:

size of Empty : 1

size of Derived : 4

Fine

size of Derived is 4 because

There is an interesting rule that says that an empty base class need not be represented by a separate byte.

So compilers are free to make optimization in case of empty base classes.

# Class object of Self type

**Can a C++ class have an object of self type?**

A class declaration cannot have a non-static object of self type.

**A class declaration can contain static object of self type and pointer to self type.**

If a non-static object is member then declaration of class is incomplete and compiler has no way to find out size of the objects of the class.

**Static variables do not contribute to the size of objects. So no problem in calculating size with static variables of self type.**

For a compiler, all pointers have a fixed size irrespective of the data type they are pointing to, so no problem with this also.

#include<iostream>

using namespace std;

class Test {

static Test self;

Test \*ptr;

//Test obj; // error: field 'obj' has incomplete type 'Test'

};

int main() {

Test T;

return 0;

}

# Structure vs class in C++

In C++, a structure is same as class except the following differences:

1. Members of a class are private by default and members of struct are public by default.
2. When deriving a struct from a class/struct, default access-specifier for a base class/struct is public. And when deriving a class, default access specifier is private.

# Local Classes in C++

facts about local classes.

1. A local class type name can only be used in the enclosing function.
2. All the methods of Local classes must be defined inside the class only.
3. Member methods of local class can only access static and enum variables of the enclosing function. Non-static variables of the enclosing function are not accessible inside local classes.
4. Local classes can access global types, variables and functions. Also, local classes can access other local classes of same function. Global variables should be used with the scope operator (::).
5. Local class cannot use automatic local variables.

Restrictions:

1. A Local class cannot contain static data members. It may contain static functions though.
2. Enclosing function cannot access the private members of a local class. Although, we can achieve this by declaring the enclosing function as a friend.

# Nested Classes in C++

A nested class is a class which is declared in another enclosing class.

A nested class is a member and as such has the same access rights as any other member.

The members of an enclosing class have no special access to members of a nested class; the usual access rules shall be obeyed.

A nested object is created in two stages:

1. The member objects are created using their respective constructors
2. Then , the other members are created

It means, constructor of all the member objects should be called before its own constructor body is executed.

#include<iostream>

using namespace std;

class Enclosing {

int x;

class Nested {

int y;

void NestedFun(Enclosing \*e) {

cout<<e->x; // works fine: nested class can access private members of Enclosing class

}

}; // declaration Nested class ends here

void EnclosingFun(Nested \*n) {

// cout<<n->y; // error: 'int Enclosing::Nested::y' is private

// error: within this context

}

}; // declaration Enclosing class ends here

int main() {

return 0;

}

# END OF FILE