**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.  
For example program 1 fails in compilation and program 2 works fine.

|  |
| --- |
| // Program 1  #include <stdio.h>    class Test {      int x; // x is private  };  int main()  {    Test t;    t.x = 20; // compiler error because x is private    getchar();    return 0;  } |
| // Program 2  #include <stdio.h>    struct Test {      int x; // x is public  };  int main()  {    Test t;    t.x = 20; // works fine because x is public    getchar();    return 0;  } |

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.  
For example program 3 fails in compilation and program 4 works fine.

|  |
| --- |
| // Program 3  #include <stdio.h>    class Base {  public:      int x;  };    class Derived : Base { }; // is equilalent to class Derived : private Base {}    int main()  {    Derived d;    d.x = 20; // compiler error becuase inheritance is private    getchar();    return 0;  } |
| // Program 4  #include <stdio.h>    class Base {  public:      int x;  };    struct Derived : Base { }; // is equilalent to struct Derived : public Base {}    int main()  {    Derived d;    d.x = 20; // works fine becuase inheritance is public    getchar();    return 0;  } |

# Difference between C structures and C++ structures

In C++, struct and class are exactly the same things, except for that struct defaults to public visibility and class defaults to private visibility.  
**Some important differences between the C and C++ structures:**

1. **Member functions inside structure**: Structures in C cannot have member functions inside structure but Structures in C++ can have member functions along with data members.
2. **Direct Initialization:** We cannot directly initialize structure data members in C but we can do it in C++.

|  |
| --- |
| // C program to demonstrate that direct  // member initialization is not possible in C  #include <stdio.h>    struct Record {      int x = 7;  };    // Driver Program  int main()  {      struct Record s;      printf("%d", s.x);      return 0;  }  /\* Output :  Compiler Error     6:8: error: expected ':', ', ', ';', '}' or    '\_\_attribute\_\_' before '=' token    int x = 7;          ^    In function 'main': \*/ |

1. **Using struct keyword:** In C, we need to use struct to declare a struct variable. In C++, struct is not necessary. For example, let there be a structure for Record. In C, we must use “struct Record” for Record variables. In C++, we need not use struct and using ‘Record‘ only would work.
2. **Static Members:** C structures cannot have static members but is allowed in C++.

|  |
| --- |
| // C program with structure static member  struct Record {      static int x;  };    // Driver program  int main()  {      return 0;  }  /\* 6:5: error: expected specifier-qualifier-list     before 'static'       static int x;       ^\*/ |

This will generate an error in C but no error in C++.

1. **Constructor creation in structure**: Structures in C cannot have constructor inside structure but Structures in C++ can have Constructor creation.

|  |
| --- |
| // C program to demonstrate that Constructor is not allowed  #include <stdio.h>    struct Student {      int roll;      Student(int x)      {          roll = x;      }  };    // Driver Program  int main()  {      struct Student s(2);      printf("%d", s.x);      return 0;  }  /\* Output in C :  Compiler Error     [Error] expected specifier-qualifier-list      before 'Student'     [Error] expected declaration specifiers or     '...' before numeric constant     [Error] 's' undeclared (first use     5555555555in this function)     In function 'main': \*/ |

**Output in C++:**

2

1. **sizeof operator:** This operator will generate **0** for an empty structure in C whereas **1** for an empty structure in C++.

|  |
| --- |
| // C program to illustrate empty structure  #include <stdio.h>    // empty structure  struct Record {  };    // Driver program  int main()  {      struct Record s;      printf("%d\n", sizeof(s));      return 0;  } |

Output in C:

0

Output in C++:

1

1. **Data Hiding:** C structures do not allow concept of Data hiding but is permitted in C++ as C++ is an object oriented language whereas C is not.
2. **Access Modifiers:** C structures do not have access modifiers as these modifiers are not supported by the language. C++ structures can have this concept as it is inbuilt in the language.

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

Predict the output of following program?

|  |
| --- |
| #include<iostream>  using namespace std;    class Empty {};    int main()  {    cout << sizeof(Empty);    return 0;  } |

Output:

1

Size of an empty class is not zero. It is 1 byte generally. **It is nonzero to ensure that the two different objects will have different addresses**. See the following example.

|  |
| --- |
| #include<iostream>  using namespace std;    class Empty { };    int main()  {      Empty a, b;        if (&a == &b)        cout << "impossible " << endl;      else        cout << "Fine " << endl;       return 0;  } |

Output:

Fine

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

|  |
| --- |
| #include<iostream>  using namespace std;    class Empty { };    int main()  {      Empty\* p1 = new Empty;      Empty\* p2 = new Empty;        if (p1 == p2)          cout << "impossible " << endl;      else          cout << "Fine " << endl;        return 0;  } |

Output:

Fine

Now guess the output of following program (This is tricky)

|  |
| --- |
| #include<iostream>  using namespace std;    class Empty { };    class Derived: Empty { int a; };    int main()  {      cout << sizeof(Derived);      return 0;  } |

Output (with GCC compiler. See [this](http://ideone.com/JFoX8)):

4

Note that the output is not greater than 4. 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. As an excercise, try the following program on your compiler.

|  |
| --- |
| // Thanks to Venki for suggesting this code.  #include <iostream>  using namespace std;    class Empty  {};    class Derived1 : public Empty  {};    class Derived2 : virtual public Empty  {};    class Derived3 : public Empty  {      char c;  };    class Derived4 : virtual public Empty  {      char c;  };    class Dummy  {      char c;  };    int main()  {  cout << "sizeof(char) " << sizeof(char) << endl;  cout << "sizeof(int) " << sizeof(int) << endl;  cout << "sizeof(Empty) " << sizeof(Empty) << endl;      cout << "sizeof(Derived1) " << sizeof(Derived1) << endl;      cout << "sizeof(Derived2) " << sizeof(Derived2) << endl;      cout << "sizeof(Derived3) " << sizeof(Derived3) << endl;      cout << "sizeof(Derived4) " << sizeof(Derived4) << endl;      cout << "sizeof(Dummy) " << sizeof(Dummy) << endl;        return 0;  } |

Output:

sizeof(char) 1

sizeof(int) 4

sizeof(Empty) 1

sizeof(Derived1) 1

sizeof(Derived2) 8

sizeof(Derived3) 1

sizeof(Derived4) 16

sizeof(Dummy) 1

|  |
| --- |
| Question 5 |

Assume that an integer and a pointer each takes 4 bytes. Also, assume that there is no alignment in objects. Predict the output following program.

|  |  |
| --- | --- |
| #include<iostream>  using namespace std;    class Test  {      static int x;      int \*ptr;      int y;  };    int main()  {      Test t;      cout << sizeof(t) << " ";      cout << sizeof(Test \*);  } | |
| A | 12 4 |
| B | 12 12 |
| C | 8 4 |
| D | 8 8 |

Explanation:

For a compiler where pointers take 4 bytes, the statement "sizeof(Test \*)" returns 4 (size of the pointer ptr). The statement "sizeof(t)" returns 8. Since static is not associated with each object of the class, we get (8 not 12).

|  |
| --- |
| Question 6 |

Which of the following is true about the following program

|  |
| --- |
| #include <iostream>  class Test  {  public:      int i;      void get();  };  void Test::get()  {      std::cout << "Enter the value of i: ";      std::cin >> i;  }  Test t;  // Global object  int main()  {      Test t;  // local object      t.get();      std::cout << "value of i in local t: "<<t.i<<'n';      ::t.get();      std::cout << "value of i in global t: "<<::t.i<<'n';      return 0;  } |

Contributed by **Pravasi Meet**

|  |  |
| --- | --- |
| A | Compiler Error: Cannot have two objects with same class name |
| B | Compiler Error in Line "::t.get();" |
| C | Compiles and runs fine |

Explanation:

The above program compiles & runs fine. Like variables it is possible to create 2 objects having same name & in different scope.

|  |
| --- |
| Question 8 |

Which of the following is not correct for virtual function in C++ ?

|  |  |
| --- | --- |
| A | Must be declared in public section of class. |
| B | Virtual function can be static. |
| C | Virtual function should be accessed using pointers. |
| D | Virtual function is defined in base class. |

Explanation:

Virtual function is can't be static in C++. So, option (B) is correct.

|  |
| --- |
| Question 2 |

|  |  |
| --- | --- |
| #include<iostream>  using namespace std;  int x = 1;  void fun()  {      int x = 2;      {          int x = 3;          cout << ::x << endl;      }  }  int main()  {      fun();      return 0;  } | |
| A | 1 |
| B | 2 |
| C | 3 |
| D | 0 |

Explanation:

The value of ::x is 1. The scope resolution operator when used with a variable name, always refers to global variable.

|  |
| --- |
| Question 3 |

Predict the output of following C++ program

|  |  |
| --- | --- |
| #include<iostream>  using namespace std;    union A {    int a;    unsigned int b;    A() { a = 10; }    unsigned int getb() {return b;}  };    int main()  {      A obj;      cout << obj.getb();      return 0;  } | |
| A | Compiler Error: union can't have functions | |
| B | Compiler Error: can't access private members of A | |
| C | 10 | |
| D | garbage value | |

Explanation:

Like struct and class, union can have methods. Like struct and unlike class, members of union are public by default. Since data members of union share memory, the value of b becomes same as a.

|  |
| --- |
| Question 5 |

How can we make a C++ class such that objects of it can only be created using new operator? If user tries to create an object directly, the program produces compiler error.

|  |  |
| --- | --- |
| A | Not possible |
| B | By making destructor private |
| C | By making constructor private |
| D | By making both constructor and destructor private |

Explanation:

See the following example.

// Objects of test can only be created using new

class Test

{

private:

~Test() {}

friend void destructTest(Test\* );

};

// Only this function can destruct objects of Test

void destructTest(Test\* ptr)

{

delete ptr;

}

int main()

{

// create an object

Test \*ptr = new Test;

// destruct the object

destructTest (ptr);

return 0;

}

|  |
| --- |
| Question 7 |

|  |
| --- |
| #include<iostream>  using namespace std;    int x[100];  int main()  {      cout << x[99] << endl;  } |

*This question is contributed by Sudheendra Baliga*

|  |  |
| --- | --- |
| A | Unpredictable |
| B | Runtime error |
| C | 0 |
| D | 99 |

Explanation:

The correct answer is c. In C++ all the uninitialized global variables are initialized to 0.