**Roll no. SC 55 - Shreyas Chavhan**

|  |
| --- |
| **International Institute of Information Technology, Pune - 411057  Department of Computer Engineering**  **Academic Year 2020-2021** |
|

**Assignment-I Class: S.E**

**Subject: Object oriented programming**

**Q. 1) Explain the important features/characteristics/properties of OOP**

**Ans -**

### The following are the major characteristics of Oop’s

### Objects:-

1) Objects are basic building blocks for designing programs.

2) An object is a collection of data members and associated member functions.

3) An object may represent a person, place or a table of data.

4) Each object is identified by a unique name. Each object must be a member of a particular class.

5) Example: chair, table, whiteboard are the objects of the class (class).

### Classes:-

1) The objects can be made user-defined data types with the help of a class.

2) A class is a collection of objects that have identical properties, common behavior and shared relationship.

3) Once the class is defined any number of objects of that class is created.

4) Classes are user-defined data types A class can hold both data and functions.

5) For example planets, sun and moon are the members of the solar system class.

### Data abstraction:-

1) Data abstraction refers to the process of representing essential features without including background details or explanations.

### Data encapsulation:-

1) The wrapping of data and function into a single unit is called data encapsulation.

2) Data encapsulation enables data hiding and information hiding.

### Inheritance:-

1) Inheritance is the process by which one object can acquire and the use of properties of another object.

2) The existing class is known as a base class or superclass.

3) The new class is known as a derived class or subclass.

4) The derived class shares some of the properties of the base class. Therefore a code from a base class can be reused by a derived class.

### Polymorphism:-

1) The ability of an operator and function to take.

2) Multiple forms are known as polymorphism.

3) The different types of polymorphism are operator

4) overloading and function overloading.

**Q.2) What is Constructor? What is a destructor? What are types of Constructors?**

**Ans -**

Constructors are special class functions which perform initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructors initialize values to object members after storage is allocated to the object.

Whereas, Destructor on the other hand is used to destroy the class object.

## Types of Constructors in C++

Constructors are of three types:

1. Default Constructor -

### Default Constructors

Default constructor is the constructor which doesn't take any argument. It has no parameter.

**Syntax:**

**class\_name(parameter1, parameter2, …….){**

**// constructor definition**

**}**

**A default constructor is so important for initialization of object members, that even if we do not define a constructor explicitly, the compiler will provide a default constructor implicitly.**

1. Parameterized Constructor -

These are the constructors with parameters. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

1. Copy Constructor -

These are special types of Constructors which takes an object as argument, and is used to copy values of data members of one object into another object.

**Q. 3) Explain with examples what are different data types supported by C++.**

**Ans -**

In C++, data types are declarations for variables. This determines the type and size of data associated with variables. For example,

int age = 13;

Here, age is a variable of type int. Meaning, the variable can only store integers of either 2 or 4 bytes.

C++ Fundamental Data Types

The table below shows the fundamental data types, their meaning, and their sizes (in bytes):

|  |  |  |
| --- | --- | --- |
| Data Type | Meaning | Size (in Bytes) |
| int | Integer | 2 or 4 |
| float | Floating-point | 4 |
| double | Double Floating-point | 8 |
| char | Character | 1 |
| wchar\_t | Wide Character | 2 |
| bool | Boolean | 1 |
| void | Empty | 0 |

1. C++ int

The int keyword is used to indicate integers.

Its size is usually 4 bytes. Meaning, it can store values from -2147483648 to 2147483647.

For example,

int salary = 85000;

2. C++ float and double

float and double are used to store floating-point numbers (decimals and exponentials).

The size of float is 4 bytes and the size of double is 8 bytes. Hence, double has two times the precision of float. To learn more, visit C++ float and double.

For example,

float area = 64.74;

double volume = 134.64534;

As mentioned above, these two data types are also used for exponentials. For example,

double distance = 45E12 // 45E12 is equal to 45\*10^12

3. C++ char

Keyword char is used for characters.

Its size is 1 byte.

Characters in C++ are enclosed inside single quotes ' '.

For example,

char test = 'h';

Note: In C++, an integer value is stored in a char variable rather than the character itself. To learn more, visit C++ characters.

4. C++ wchar\_t

Wide character wchar\_t is similar to the char data type, except its size is 2 bytes instead of 1.

It is used to represent characters that require more memory to represent them than a single char.

For example,

wchar\_t test = L'ם' // storing Hebrew character;

Notice the letter L before the quotation marks.

Note: There are also two other fixed-size character types char16\_t and char32\_t introduced in C++11.

5. C++ bool

The bool data type has one of two possible values: true or false.

Booleans are used in conditional statements and loops (which we will learn in later chapters).

For example,

bool cond = false;

6. C++ void

The void keyword indicates an absence of data. It means "nothing" or "no value".

We will use void when we learn about functions and pointers.

Note: We cannot declare variables of the void type.

**Q. 4) Define a class to represent a bank account .Include the following members : Data Members: Name of the depositor, Account Number, Type of Account, balance amount. Member Functions: To assign initial values, to deposit amounts. To withdraw an amount after checking the balance. To display name and balance. Write a program in C++.**

**Ans -**

|  |
| --- |
| #include <iostream>  using namespace std;  class bankAccount{  string depositor;  long account\_number;  string type\_of\_account;  float balance\_ammount;  public:  void initialize(int acc\_no, string name, string acc\_type, float balance){  depositor = name;  account\_number = acc\_no;  type\_of\_account = acc\_type;  balance\_ammount = balance;  }  void deposit();  void withdraw();  void display();  };  void bankAccount :: deposit(){  int deposit;  cout << "\nEnter Deposit Amount: ";  cin >> deposit;  balance\_ammount += deposit;  }  void bankAccount :: withdraw(){  int withdraw\_amount;  cout << "\nEnter Withdraw Amount = ";  cin >> withdraw\_amount;  if(withdraw\_amount > balance\_ammount)  cout << "\n Cannot Withdraw Amount";  balance\_ammount -= withdraw\_amount;  }  void bankAccount :: display(){  cout << "+++++++++++++++++++++++++++" << endl;  cout << "\nAccout No. : " << account\_number;  cout << "\nName : " << depositor;  cout << "\nAccount Type : " << type\_of\_account;  cout << "\nBalance : "<< balance\_ammount;  }  int main(){  int acc\_no;  string name, acc\_type;  float balance;  cout << "<-- Enter Details -->" << endl;  cout << "-----------------------" << endl;  cout << "Account No.: ";  cin >> acc\_no;  cout << "Name: ";  cin >> name;  cout << "Account Type: ";  cin >> acc\_type;  cout << "Balance: ";  cin >> balance;  bankAccount shreyas;  shreyas.initialize(acc\_no, name, acc\_type, balance);  shreyas.deposit();  shreyas.withdraw();  shreyas.display();  return 0;  } |

Q. 1) Create a class called employee that contains a name (an object of class string) and an employee number (type long). Include a member function called getdata() to get data from the user for insertion into the object, and another function called putdata() to display the data. Assume the name has no embedded blanks. Write a main() program to exercise this class. It should create an array of type employee, and then invite the user to input data for up to 100 employees. Finally, it should print out the data for all the employees

**Q.2) Explain with example What is static data members & static member functions**

Ans -

Static data members are class members that are declared using the static keyword. There is only one copy of the static data member in the class, even if there are many class objects. This is because all the objects share the static data member. The static data member is always initialized to zero when the first class object is created.

The syntax of the static data members is given as follows −

|  |
| --- |
| static data\_type data\_member\_name; |

In the above syntax, a static keyword is used. The data\_type is the C++ data type such as int, float etc. The data\_member\_name is the name provided to the data member.

A program that demonstrates the static data members in C++ is given as follows −

|  |
| --- |
| #include <iostream>  #include<string.h>  using namespace std;  class Student {  private:  int rollNo;  char name[10];  int marks;  public:  static int objectCount;  Student() {  objectCount++;  }  void getdata() {  cout << "Enter roll number: "<<endl;  cin >> rollNo;  cout << "Enter name: "<<endl;  cin >> name;  cout << "Enter marks: "<<endl;  cin >> marks;  }  void putdata() {  cout<<"Roll Number = "<< rollNo <<endl;  cout<<"Name = "<< name <<endl;  cout<<"Marks = "<< marks <<endl;  cout<<endl;  }  };  int Student::objectCount = 0;  int main(void) {  Student s1;  s1.getdata();  s1.putdata();  Student s2;  s2.getdata();  s2.putdata();  Student s3;  s3.getdata();  s3.putdata();  cout << "Total objects created = " << Student::objectCount << endl;  return 0;  } |

Q. 3) **Differentiate between pass-by value and pass-by reference with example. How pass-by reference is advantageous.**

Ans -

In C++ we can pass arguments into a function in different ways. These different ways are −

* Call by Value
* Call by Reference
* Call by Address

Sometimes the call by address is referred to as call by reference, but they are different in C++. In call by address, we use pointer variables to send the exact memory address, but in call by reference we pass the reference variable (alias of that variable). This feature is not present in C, there we have to pass the pointer to get that effect. In this section we will see what are the advantages of call by reference over call by value, and where to use them

## Call by Value

In call by value, the actual value that is passed as argument is not changed after performing some operation on it. When call by value is used, it creates a copy of that variable into the stack section in memory. When the value is changed, it changes the value of that copy, the actual value remains the same.

## Example Code

## 

|  |
| --- |
| #include<iostream>  using namespace std;  void my\_function(int x) {  x = 50;  cout << "Value of x from my\_function: " << x << endl;  }  main() {  int x = 10;  my\_function(x);  cout << "Value of x from main function: " << x;  } |

## Output

## 

|  |
| --- |
| Value of x from my\_function: 50  Value of x from main function: 10 |

## Call by Reference

In call by reference the actual value that is passed as argument is changed after performing some operation on it. When call by reference is used, it creates a copy of the reference of that variable into the stack section in memory. Is uses a reference to get the value. So when the value is changed using the reference it changes the value of the actual variable.

## Example Code

## 

|  |
| --- |
| #include<iostream>  using namespace std;  void my\_function(int &x) {  x = 50;  cout << "Value of x from my\_function: " << x << endl;  }  main() {  int x = 10;  my\_function(x);  cout << "Value of x from main function: " << x;  } |

## Output

### 

|  |
| --- |
| Value of x from my\_function: 50  Value of x from main function: 50 |

### Where to use Call by reference?

* The call by reference is mainly used when we want to change the value of the passed argument into the invoker function.
* One function can return only one value. When we need more than one value from a function, we can pass them as an output argument in this manner.

**Q. 4) How function overloading is done in c++.What are its types.**

Ans -

Function overloading is a [C++ programming](https://beginnersbook.com/2017/08/c-plus-plus-tutorial-for-beginners/) feature that allows us to have more than one function having same name but different parameter list, when I say parameter list, it means the data type and sequence of the parameters, for example the parameters list of a function myfunc(int a, float b) is (int, float) which is different from the function myfunc(float a, int b) parameter list (float, int). Function overloading is a [compile-time polymorphism](https://beginnersbook.com/2017/08/cpp-polymorphism/).

Now that we know what a parameter list is, let's see the rules of overloading: we can have the following functions in the same scope.

sum(int num1, int num2)

sum(int num1, int num2, int num3)

sum(int num1, double num2)

The easiest way to remember this rule is that the parameters should qualify any one or more of the following conditions, they should have different **type**, **number** or **sequence** of parameters.

**For example:**

These two functions have different parameter **type**:

sum(int num1, int num2)

sum(double num1, double num2)

These two have different **number** of parameters:

sum(int num1, int num2)

sum(int num1, int num2, int num3)

These two have different **sequence** of parameters:

sum(int num1, double num2)

sum(double num1, int num2)

All of the above three cases are valid case of overloading. We can have any number of functions, just remember that the parameter list should be different. For example:

int sum(int, int)

double sum(int, int)

This is not allowed as the parameter list is same. Even though they have different return types, its not valid.

## Function overloading Example

Let's take an example to understand function overloading in C++.

#include <iostream>

using namespace std;

class Addition {

public:

int sum(int num1,int num2) {

return num1+num2;

}

int sum(int num1,int num2, int num3) {

return num1+num2+num3;

}

};

int main(void) {

Addition obj;

cout<<obj.sum(20, 15)<<endl;

cout<<obj.sum(81, 100, 10);

return 0;

}

**Output:**

35

191

## Function overloading Example 2

As I mentioned in the beginning of this guide that functions having different return types and same parameter list cannot be overloaded. However if the functions have different parameter lists then they can have same or different return types to be eligible for overloading. In short the return type of a function

does not play any role in function overloading. All that matters is the parameter list of function.

#include <iostream>

using namespace std;

class DemoClass {

public:

int demoFunction(int i) {

return i;

}

double demoFunction(double d) {

return d;

}

};

int main(void) {

DemoClass obj;

cout<<obj.demoFunction(100)<<endl;

cout<<obj.demoFunction(5005.516);

return 0;

}

**Output:**

100

5006.52

## Advantages of Function overloading

The main advantage of function overloading is to improve the **code readability** and allow **code reusability**. In the example 1, we have seen how we were able to have more than one function for the same task(addition) with different parameters, this allowed us to add two integer numbers as well as three integer numbers, if we wanted we could have some more functions with same name and four or five arguments.

Imagine if we didn’t have function overloading, we either have the limitation to add only two integers or we had to write different name functions for the same task addition, this would reduce the code readability and reusability.

**Q. 1) Explain friend function and friend class with example.**

**Ans -**

**As we know that a class cannot access the private members of other class. Similarly a class that doesn’t inherit another class cannot access its protected members.**

**Friend Class:**

**A friend class is a class that can access the private and protected members of a class in which it is declared as friend. This is needed when we want to allow a particular class to access the private and protected members of a class.**

## Function Class Example

**In this example we have two classes XYZ and ABC. The XYZ class has two private data members ch and num, this class declares ABC as friend class. This means that ABC can access the private members of XYZ, the same has been demonstrated in the example where the function disp() of ABC class accesses the private members num and ch. In this example we are** [**passing object as an argument to the function**](https://beginnersbook.com/2017/09/cpp-pass-and-return-object-from-a-function/)**.**

|  |
| --- |
| **#include <iostream>**  **using namespace std;**  **class XYZ {**  **private:**  **char ch='A';**  **int num = 11;**  **public:**  **/\* This statement would make class ABC**  **\* a friend class of XYZ, this means that**  **\* ABC can access the private and protected**  **\* members of XYZ class.**  **\*/**  **friend class ABC;**  **};**  **class ABC {**  **public:**  **void disp(XYZ obj){**  **cout<<obj.ch<<endl;**  **cout<<obj.num<<endl;**  **}**  **};**  **int main() {**  **ABC obj;**  **XYZ obj2;**  **obj.disp(obj2);**  **return 0;**  **}** |

**Output:**

|  |
| --- |
| **A**  **11** |

**Friend Function:**

**Similar to friend class, this function can access the private and protected members of another class. A global function can also be declared as friend as shown in the example below:**

## Friend Function Example

|  |
| --- |
| **#include <iostream>**  **using namespace std;**  **class XYZ {**  **private:**  **int num=100;**  **char ch='Z';**  **public:**  **friend void disp(XYZ obj);**  **};**  **//Global Function**  **void disp(XYZ obj){**  **cout<<obj.num<<endl;**  **cout<<obj.ch<<endl;**  **}**  **int main() {**  **XYZ obj;**  **disp(obj);**  **return 0;**  **}** |

**Output:**

|  |
| --- |
| **100**  **Z** |

Q.2 Create a class called time that has separate int member data for hours, minutes, and seconds. One constructor should initialize this data to 0, and another should initialize it to fixed values. Another member function should display it, in 11:59:59 format. The final member function should add two objects of type time passed as arguments. A main() program should create two initialized time objects (should they be const?) and one that isn’t initialized. Then it should add the two initialized values together, leaving the result in the third time variable. Finally it should display the value of this third variable. Make appropriate member functions const.

Ans -

|  |
| --- |
| #include <iostream>  #include <conio.h>  using namespace std;  class time{  private:  int hours,minutes,seconds;  public:  time(){  hours = minutes = seconds = 0;  }  time(int h, int m, int s){  hours = h;  minutes = m;  seconds = s;  }  void showTime() const {  cout << hours << ':' << minutes << ':' << seconds;  }  void addTime(time x, time y){  seconds = x.seconds + y.seconds;  if(seconds>59){  seconds-=60;  minutes++;  }  minutes += x.minutes + y.minutes;  if(minutes>59){  minutes-=60;  hours++;  }  hours+=x.hours+y.hours;  }  };  int main(){  const time a(2,23,45), b(4,25,15);  time c;  c.addTime(a,b);  c.showTime();  return 0;  } |

Q. 3) What are the benefits of OOP

Ans -

1. **Simplicity:** software objects model real world objects, so the complexity is reduced and the program structure is very clear;

2. **Modularity:** each object forms a separate entity whose internal workings are decoupled from other parts of the system;

3. **Modifiability:** it is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods;

4. **Extensibility:** adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones;

5. **Maintainability:** objects can be maintained separately, making locating and fixing problems easier;

**6. Re-usability:** objects can be reused in different programs

Q. 4) Explain new, delete operator with example.

Ans -

Dynamic memory allocation in C/C++ refers to performing memory allocation manually by programmer. Dynamically allocated memory is allocated on **Heap** and non-static and local variables get memory allocated on **Stack**

**What are applications?**

* One use of dynamically allocated memory is to allocate memory of variable size which is not possible with compiler allocated memory except [variable length arrays](https://www.geeksforgeeks.org/variable-length-arrays-in-c-and-c/).
* The most important use is flexibility provided to programmers. We are free to allocate and deallocate memory whenever we need and whenever we don’t need anymore. There are many cases where this flexibility helps. Examples of such cases are [Linked List](https://www.geeksforgeeks.org/data-structures/linked-list/), [Tree](https://www.geeksforgeeks.org/binary-tree-2/), etc.

**How is it different from memory allocated to normal variables?**

For normal variables like “int a”, “char str[10]”, etc, memory is automatically allocated and deallocated. For dynamically allocated memory like “int \*p = new int[10]”, it is programmers responsibility to deallocate memory when no longer needed. If programmer doesn’t deallocate memory, it causes [memory leak](https://www.geeksforgeeks.org/what-is-memory-leak-how-can-we-avoid/) (memory is not deallocated until program terminates).

**How is memory allocated/deallocated in C++?**

C uses [malloc() and calloc()](https://www.geeksforgeeks.org/calloc-versus-malloc/) function to allocate memory dynamically at run time and uses free() function to free dynamically allocated memory. C++ supports these functions and also has two operators **new** and **delete** that perform the task of allocating and freeing the memory in a better and easier way.

**new operator**

The new operator denotes a request for memory allocation on the Free Store. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

**Syntax to use new operator**: To allocate memory of any data type, the syntax is:  
pointer-variable = **new** data-type;

Here, pointer-variable is the pointer of type data-type. Data-type could be any built-in data type including array or any user defined data types including structure and class.  
Example:

|  |
| --- |
| // Pointer initialized with NULL  // Then request memory for the variable  int \*p = NULL;  p = new int;  OR  // Combine declaration of pointer  // and their assignment  int \*p = new int; |

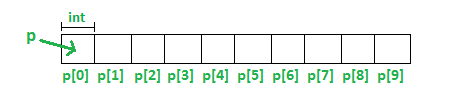
**Initialize memory:** We can also initialize the memory using new operator:  
pointer-variable = **new** data-type(value);

|  |
| --- |
| **Example:**  int \*p = new int(25);  float \*q = new float(75.25); |

**Allocate block of memory:** new operator is also used to allocate a block(an array) of memory of type *data-type*.  
pointer-variable = **new** data-type[size];

where size(a variable) specifies the number of elements in an array.

|  |
| --- |
| Example:  int \*p = new int[10] |

Dynamically allocates memory for 10 integers continuously of type int and returns pointer to the first element of the sequence, which is assigned to p(a pointer). p[0] refers to first element, p[1] refers to second element and so on.  


**Normal Array Declaration vs Using new**

There is a difference between declaring a normal array and allocating a block of memory using new. The most important difference is, normal arrays are deallocated by compiler (If array is local, then deallocated when function returns or completes). However, dynamically allocated arrays always remain there until either they are deallocated by programmer or program terminates.

**What if enough memory is not available during runtime?**

If enough memory is not available in the heap to allocate, the new request indicates failure by throwing an exception of type std::bad\_alloc, unless “nothrow” is used with the new operator, in which case it returns a NULL pointer (scroll to section “Exception handling of new operator” in [this](https://aticleworld.com/dynamic-memory-and-new-operator-c/) article). Therefore, it may be good idea to check for the pointer variable produced by new before using it program.

|  |
| --- |
| int \*p = new(nothrow) int;  if (!p)  {  cout << "Memory allocation failed\n"; |

}

**delete operator**

Since it is programmer’s responsibility to deallocate dynamically allocated memory, programmers are provided delete operator by C++ language.

**Syntax:**

|  |
| --- |
| // Release memory pointed by pointer-variable  **delete** pointer-variable; |

Here, pointer-variable is the pointer that points to the data object created by *new*.

Examples:

|  |
| --- |
| delete p;  delete q; |

To free the dynamically allocated array pointed by pointer-variable, use following form of *delete*:

|  |
| --- |
| // Release block of memory  // pointed by pointer-variable  delete[] pointer-variable;  Example:  // It will free the entire array  // pointed by p.  delete[] p; |

|  |
| --- |
| // C++ program to illustrate dynamic allocation  // and deallocation of memory using new and delete  #include <iostream>  using namespace std;    int main ()  {  // Pointer initialization to null  int\* p = NULL;    // Request memory for the variable  // using new operator  p = new(nothrow) int;  if (!p)  cout << "allocation of memory failed\n";  else  {  // Store value at allocated address  \*p = 29;  cout << "Value of p: " << \*p << endl;  }    // Request block of memory  // using new operator  float \*r = new float(75.25);    cout << "Value of r: " << \*r << endl;    // Request block of memory of size n  int n = 5;  int \*q = new(nothrow) int[n];    if (!q)  cout << "allocation of memory failed\n";  else  {  for (int i = 0; i < n; i++)  q[i] = i+1;    cout << "Value store in block of memory: ";  for (int i = 0; i < n; i++)  cout << q[i] << " ";  }    // freed the allocated memory  delete p;  delete r;    // freed the block of allocated memory  delete[] q;    return 0;  } |

Output:

Value of p: 29

Value of r: 75.25

Value store in block of memory: 1 2 3 4 5

Q. 1) Define class Number which has inline function mult() and cube() for calculating the multiplication of 2 double numbers given and cube of the integer number given.

|  |
| --- |
| class Number{  double num1, num2;  int n1, n2, n3;  public:  inline double mult(double a, double b){  num1 = a; num2 = b;  return num1 \* num2;  }  inline int cube(int c, int d, int e){  n1 = c; n2 = d; n3 = e;  return n1 \* n2 \* n3;  }  }; |

Q.2) Write a C++ program to compare two strings using operator overloading (==,>,<,!=)

|  |
| --- |
| #include<iostream>  #include<stdio.h>  #include<string.h>  using namespace std;    class String  {  char str[20];  public:    void getdata()  {  gets(str);    }    int operator ==(String s)  {  if(!strcmp(str,s.str))  return 1;    return 0;  }  };    int main()  {  String s1,s2;    cout<<"Enter first string :: ";  s1.getdata();  cout<<"\nEnter second string :: ";  s2.getdata();  if(s1==s2)  {  cout<<"\nStrigs are Equal\n";  }  else  {  cout<<"\nStrings are Not Equal\n";  }    return 0;  } |

Q. 3) Explain polymorphism with examples.

Ans -

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Consider the following example where a base class has been derived by other two classes −

|  |
| --- |
| #include <iostream>  using namespace std;    class Shape {  protected:  int width, height;    public:  Shape( int a = 0, int b = 0){  width = a;  height = b;  }  int area() {  cout << "Parent class area :" <<endl;  return 0;  }  };  class Rectangle: public Shape {  public:  Rectangle( int a = 0, int b = 0):Shape(a, b) { }    int area () {  cout << "Rectangle class area :" <<endl;  return (width \* height);  }  };  class Triangle: public Shape {  public:  Triangle( int a = 0, int b = 0):Shape(a, b) { }    int area () {  cout << "Triangle class area :" <<endl;  return (width \* height / 2);  }  };  // Main function for the program  int main() {  Shape \*shape;  Rectangle rec(10,7);  Triangle tri(10,5);  // store the address of Rectangle  shape = &rec;    // call rectangle area.  shape->area();  // store the address of Triangle  shape = &tri;    // call triangle area.  shape->area();    return 0;  } |

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

Parent class area :

Parent class area :

The reason for the incorrect output is that the call of the function area() is being set once by the compiler as the version defined in the base class. This is called static resolution of the function call, or static linkage - the function call is fixed before the program is executed. This is also sometimes called early binding because the area() function is set during the compilation of the program.

But now, let's make a slight modification in our program and precede the declaration of area() in the Shape class with the keyword virtual so that it looks like this −

|  |
| --- |
| class Shape {  protected:  int width, height;    public:  Shape( int a = 0, int b = 0) {  width = a;  height = b;  }  virtual int area() {  cout << "Parent class area :" <<endl;  return 0;  }  }; |

After this slight modification, when the previous example code is compiled and executed, it produces the following result −

Rectangle class area

Triangle class area

This time, the compiler looks at the contents of the pointer instead of it's type. Hence, since addresses of objects of tri and rec classes are stored in \*shape the respective area() function is called.

As you can see, each of the child classes has a separate implementation for the function area(). This is how polymorphism is generally used. You have different classes with a function of the same name, and even the same parameters, but with different implementations.

## Virtual Function

A virtual function is a function in a base class that is declared using the keyword virtual. Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function.

What we do want is the selection of the function to be called at any given point in the program to be based on the kind of object for which it is called. This sort of operation is referred to as dynamic linkage, or late binding.

## Pure Virtual Functions

It is possible that you want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class.

We can change the virtual function area() in the base class to the following −

|  |
| --- |
| class Shape {  protected:  int width, height;  public:  Shape(int a = 0, int b = 0) {  width = a;  height = b;  }    // pure virtual function  virtual int area() = 0;  }; |

The = 0 tells the compiler that the function has no body and above virtual function will be called pure virtual function.

Q. 4) Explain Operator Overloading with example (unary & binary operator overloading)

Ans -

Operator Overloading:

C++ provides a special function to change the current functionality of some operators within its class which is often called operator overloading. Operator Overloading is the method by which we can change the function of some specific operators to do some different task.

This can be done by declaring the function, its syntax is,

|  |
| --- |
| Return\_Type classname :: operator op(Argument list)  {  Function Body  } |

In the above syntax Return\_Type is a value type to be returned to another object, operator op is the function where the operator is a keyword and op is the operator to be overloaded.

Operator function must be either non-static (member function) or friend function.

Operator Overloading can be done by using three approaches, they are

Overloading unary operator.

Overloading binary operator.

Overloading binary operator using a friend function.

Below are some criteria/rules to define the operator function:

In case of a non-static function, the binary operator should have only one argument and unary should not have an argument.

In the case of a friend function, the binary operator should have only two argument and unary should have only one argument.

All the class member object should be public if operator overloading is implemented.

Operators that cannot be overloaded are . .\* :: ?:

Operator cannot be used to overload when declaring that function as friend function = () [] ->.

Refer this, for more rules of Operator Overloading

Overloading Unary Operator: Let us consider to overload (-) unary operator. In unary operator function, no arguments should be passed. It works only with one class objects. It is a overloading of an operator operating on a single operand.

Example:

Assume that class Distance takes two member object i.e. feet and inches, create a function by which Distance object should decrement the value of feet and inches by 1 (having single operand of Distance Type).

|  |
| --- |
| // C++ program to show unary operator overloading  #include <iostream>    using namespace std;    class Distance {  public:    // Member Object  int feet, inch;    // Constructor to initialize the object's value  Distance(int f, int i)  {  this->feet = f;  this->inch = i;  }    // Overloading(-) operator to perform decrement  // operation of Distance object  void operator-()  {  feet--;  inch--;  cout << "\nFeet & Inches(Decrement): " << feet << "'" << inch;  }  };    // Driver Code  int main()  {  // Declare and Initialize the constructor  Distance d1(8, 9);    // Use (-) unary operator by single operand  -d1;  return 0;  } |

**Output:**

Feet & Inches(Decrement): 7'8

In the above program, it shows that no argument is passed and no return\_type value is returned, because unary operator works on a single operand. (-) operator change the functionality to its member function.

Note: d2 = -d1 will not work, because operator-() does not return any value.

Overloading Binary Operator: In binary operator overloading function, there should be one argument to be passed. It is overloading of an operator operating on two operands.

Let’s take the same example of class Distance, but this time, add two distance objects.

|  |
| --- |
| // C++ program to show binary operator overloading  #include <iostream>    using namespace std;    class Distance {  public:  // Member Object  int feet, inch;  // No Parameter Constructor  Distance()  {  this->feet = 0;  this->inch = 0;  }    // Constructor to initialize the object's value  // Parametrized Constructor  Distance(int f, int i)  {  this->feet = f;  this->inch = i;  }    // Overloading (+) operator to perform addition of  // two distance object  Distance operator+(Distance& d2) // Call by reference  {  // Create an object to return  Distance d3;    // Perform addition of feet and inches  d3.feet = this->feet + d2.feet;  d3.inch = this->inch + d2.inch;    // Return the resulting object  return d3;  }  };    // Driver Code  int main()  {  // Declaring and Initializing first object  Distance d1(8, 9);    // Declaring and Initializing second object  Distance d2(10, 2);    // Declaring third object  Distance d3;    // Use overloaded operator  d3 = d1 + d2;    // Display the result  cout << "\nTotal Feet & Inches: " << d3.feet << "'" << d3.inch;  return 0;  } |

Output:

Total Feet & Inches: 18'11

Here in the above program,

See Line no. 26, Distance operator+(Distance &d2), here return type of function is distance and it uses call by references to pass an argument.

See Line no. 49, d3 = d1 + d2; here, d1 calls the operator function of its class object and takes d2 as a parameter, by which operator function return object and the result will reflect in the d3 object.

Pictorial View of working of Binary Operator:

operator

Overloading Binary Operator using a Friend function: In this approach, the operator overloading function must precede with friend keyword, and declare a function class scope. Keeping in mind, friend operator function takes two parameters in a binary operator, varies one parameter in a unary operator. All the working and implementation would same as binary operator function except this function will be implemented outside of the class scope.

Let’s take the same example using the friend function.

|  |
| --- |
| // C++ program to show binary operator overloading  #include <iostream>    using namespace std;    class Distance {  public:    // Member Object  int feet, inch;    // No Parameter Constructor  Distance()  {  this->feet = 0;  this->inch = 0;  }    // Constructor to initialize the object's value  // Parametrized Constructor  Distance(int f, int i)  {  this->feet = f;  this->inch = i;  }    // Declaring friend function using friend keyword  friend Distance operator+(Distance&, Distance&);  };    // Implementing friend function with two parameters  Distance operator+(Distance& d1, Distance& d2) // Call by reference  {  // Create an object to return  Distance d3;    // Perform addition of feet and inches  d3.feet = d1.feet + d2.feet;  d3.inch = d1.inch + d2.inch;    // Return the resulting object  return d3;  }    // Driver Code  int main()  {  // Declaring and Initializing first object  Distance d1(8, 9);    // Declaring and Initializing second object  Distance d2(10, 2);    // Declaring third object  Distance d3;    // Use overloaded operator  d3 = d1 + d2;    // Display the result  cout << "\nTotal Feet & Inches: " << d3.feet << "'" << d3.inch;  return 0;  } |

Output:

Total Feet & Inches: 18'11

Here in the above program, operator function is implemented outside of class scope by declaring that function as the friend function.

In these ways, an operator can be overloaded to perform certain tasks by changing the functionality of operators.

Q. 1) How data conversion is supported by C++. Explain with examples.

Ans -

A type cast is basically a conversion from one type to another. There are two types of type conversion:

1. **Implicit Type Conversion** Also known as ‘automatic type conversion’.
   * Done by the compiler on its own, without any external trigger from the user.
   * Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.

All the data types of the variables are upgraded to the data type of the variable with largest data type.  
bool -> char -> short int -> int ->

unsigned int -> long -> unsigned ->

long long -> float -> double -> long double

* + It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

1. **Example of Type Implicit Conversion:**

|  |
| --- |
| // An example of implicit conversion    #include <iostream>  using namespace std;    int main()  {  int x = 10; // integer x  char y = 'a'; // character c    // y implicitly converted to int. ASCII  // value of 'a' is 97  x = x + y;    // x is implicitly converted to float  float z = x + 1.0;    cout << "x = " << x << endl  << "y = " << y << endl  << "z = " << z << endl;    return 0;  } |

**Output:**x = 107

y = a

z = 108

1. **Explicit Type Conversion**: This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.  
   In C++, it can be done by two ways:
   * **Converting by assignment:** This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.  
     **Syntax:**(type) expression  
     where *type* indicates the data type to which the final result is converted.  
     **Example:**

|  |
| --- |
| // C++ program to demonstrate  // explicit type casting    #include <iostream>  using namespace std;    int main()  {  double x = 1.2;    // Explicit conversion from double to int  int sum = (int)x + 1;    cout << "Sum = " << sum;    return 0;  } |

**Output:**Sum = 2

* + **Conversion using Cast operator:** A Cast operator is an **unary operator** which forces one data type to be converted into another data type.  
    C++ supports four types of casting:
    1. [Static Cast](https://www.geeksforgeeks.org/static_cast-in-c-type-casting-operators/)
    2. Dynamic Cast
    3. [Const Cast](https://www.geeksforgeeks.org/casting-operators-in-c-set-1-const_cast/)
    4. [Reinterpret Cast](https://www.geeksforgeeks.org/reinterpret_cast-in-cpp/)
       1. **Example:**

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  float f = 3.5;    // using cast operator  int b = static\_cast<int>(f);    cout << b;  } |

**Output:**3

**Advantages of Type Conversion:**

* This is done to take advantage of certain features of type hierarchies or type representations.
* It helps to compute expressions containing variables of different data types.

Q.2) Describe the mechanism of accessing data members and member functions in the following cases: (i)inside the main program(ii)Inside a member function of the same class(iii)Inside a member function of another class.

to access class A members inside another class member function you can:

1. instantiate class A instance inside class B member function

2. make desired class A members static, so you need not to provide class A object to access this members.

class A {

public: // ... stativ void do\_stuff() {}

};

class B {

//.... void do complicated stuff()

{/\*...\*/ A::do\_stuff();}

};

Accessing a data member depends solely on the access control of that data member. If its public, then the data member can be easily accessed using the direct member access (.) operator with the object of that class.

If, the data member is defined as private or protected, then we cannot access the data variables directly. Then we will have to create special public member functions to access, use or initialize the private and protected data members. These member functions are also called Accessors and Mutator methods or getter and setter functions.

he mechanism of accessing data members and member functions in the following cases is done by 1.Using object and dot membership operator.

2.accessing a local variable of a function.

Example for how to access data members and member functions inside a member function of another class.

#include<iostream.h>

class a

{

public:

int x;

void display()

{

cout<<"sample \n";

x=111;

}

};

class b

{

public:

void display()

{

a s; cout<<" Member function \n"; s.

display();

cout<<" x = "<<example<<"\n";

}

};

void main()

{

b b1; // b1 is a object of class b.

b1.display();

}

Q. 3) How many arguments are required in the definition of an overloaded unary operator?

Ans -

n C++, the answer is either zero, one, or two.

If you overload a unary operator as a member function, you need zero explicit arguments, because the implicit “this” argument is used:

1. class MyClass {
2. int operator!() const { ... }
3. }

If you overload a unary operator as a non-member function, you need one argument:

1. int operator~( const MyClass &my ) { ... }

But, there’s a special case. If you want to define postfix increment, it needs to be distinguished from prefix increment, but both are operator++. So, a dummy int argument is used:

1. MyClass operator++( MClass &my, int ) { ... }

Q. 4) When is friend function compulsory. Explain with example.

Declaring a function as a friend of a class is required only if that function is not a member of the class *and* it needs access to one or more of the class’s private members.

This situation frequently comes up when overloading the inserter << and extractor >> operators for a class, to allow easy output of an object to a stream, or input of an object from a stream. Because the object appears on the right side of these operators, the overloaded operator functions cannot be class member functions; they must be functions defined outside the class.

Q. 1) Define two classes Polar and Rectangular to represent points in the polar and rectangular systems. Use conversion routines to convert from one system to the other

Q.2) A friend function cannot be used to overload the assignment operator =.Explain why?

Because if you do not declare it as a class member compiler will make one up for you and it will introduce ambiguity.

Assignment(=) operator is a special operator that will be provided by the constructor to the class when the programmer has not provided(overloaded) as member of the class.(like copy constructor).

When programmer is overloading = operator using friend function, two = operations will exists:

1) compiler is providing = operator

2) programmer is providing(overloading) = operator by friend function.

Then simply ambiguity will be created and the compiler will give an error. Its compilation error.

Q. 3) What is function .what is function prototype

In [computer programming](https://en.wikipedia.org/wiki/Computer_programming), a **function prototype** or **function interface** is a [declaration](https://en.wikipedia.org/wiki/Declaration_(computer_programming)) of a [function](https://en.wikipedia.org/wiki/Function_(programming)) that specifies the function’s name and [type signature](https://en.wikipedia.org/wiki/Type_signature) ([arity](https://en.wikipedia.org/wiki/Arity), [data types](https://en.wikipedia.org/wiki/Data_type) of [parameters](https://en.wikipedia.org/wiki/Parameter_(computer_programming)), and [return type](https://en.wikipedia.org/wiki/Return_type)), but omits the function body.

Q. 4) What is a friend function. Explain with example

**Friend Function:**

**Similar to friend class, this function can access the private and protected members of another class. A global function can also be declared as friend as shown in the example below:**

## Friend Function Example

|  |
| --- |
| **#include <iostream>**  **using namespace std;**  **class XYZ {**  **private:**  **int num=100;**  **char ch='Z';**  **public:**  **friend void disp(XYZ obj);**  **};**  **//Global Function**  **void disp(XYZ obj){**  **cout<<obj.num<<endl;**  **cout<<obj.ch<<endl;**  **}**  **int main() {**  **XYZ obj;**  **disp(obj);**  **return 0;**  **}** |

**Output:**

|  |
| --- |
| **100**  **Z** |