|  |  |
| --- | --- |
| **Ex. No. :- 9** | **Date:** DD/MM/YYYY |
| **Experiment Heading** | |
| **Aim:** Write a program to explain the concept of function overloading | |
| **Program:**  #include <iostream>  using namespace std;  void add(int a, int b)  {  cout << "sum = " << (a + b);  }  void add(double a, double b)  {  cout << endl << "sum = " << (a + b);  }  // Driver code  int main()  {  add(10, 2);  add(5.3, 6.2);  return 0;  } | |
| **Output:**  sum = 12  sum = 11.5 | |
| **Result:** The above experiment successfully completed. | |

|  |  |
| --- | --- |
| **Ex. No. :-10** | **Date:** DD/MM/YYYY |
| **Experiment Heading** | |
| **Aim:** Write a program to explain the concept of Constructor overloading | |
| **Program:**  #include <iostream>  using namespace std;  class construct  {  public:  float area;    // Constructor with no parameters  construct()  {  area = 0;  }    // Constructor with two parameters  construct(int a, int b)  {  area = a \* b;  }    void disp()  {  cout<< area<< endl;  }  };  int main()  {  construct o;  construct o2( 10, 20);    o.disp();  o2.disp();  return 1;  } | |
| **Output:**  0  200 | |
| **Result:** The above experiment successfully completed. | |

|  |  |
| --- | --- |
| **Ex. No.: 11** | **Date:** DD/MM/YYYY |
| **Experiment Heading** | |
| **Aim:** Write a program to explain the concept of operator overloading. | |
| **Program:**  #include<iostream>  using namespace std;  class Complex {  private:  int real, imag;  public:  Complex(int r = 0, int i = 0) {real = r; imag = i;}    // This is automatically called when '+' is used with  // between two Complex objects  Complex operator + (Complex const &obj) {  Complex res;  res.real = real + obj.real;  res.imag = imag + obj.imag;  return res;  }  void print() { cout << real << " + i" << imag << '\n'; }  };  int main()  {  Complex c1(10, 5), c2(2, 4);  Complex c3 = c1 + c2;  c3.print();  } | |
| **Output:**  12 + i9 | |
| **Result:** The above experiment successfully completed. | |

|  |  |
| --- | --- |
| **Ex. No.: 12** | **Date:** DD/MM/YYYY |
| **Experiment Heading** | |
| **Aim:**  Write a program for the concept of virtual functions**.** | |
| **Program:**  #include<iostream>  using namespace std;  class base {  public:  virtual void print()  {  cout << "print base class\n";  }  void show()  {  cout << "show base class\n";  }  };  class derived : public base {  public:  void print()  {  cout << "print derived class\n";  }  void show()  {  cout << "show derived class\n";  }  };  int main()  {  base \*bptr;  derived d;  bptr = &d;  // Virtual function, binded at runtime  bptr->print();  // Non-virtual function, binded at compile time  bptr->show();    return 0;  } | |
| **Output:**  print derived class  show base class | |
| **Result:** The above experiment successfully completed. | |

|  |  |
| --- | --- |
| **Ex. No.: 13** | **Date:** DD/MM/YYYY |
| **Experiment Heading** | |
| **Aim:**  Write a program to understand the concept of inheritance. | |
| **Program:**  #include<iostream>  using namespace std;  class A  {  public:  int a=10;  void DisplayA()  {  cout<<"DisplayA : a="<<a<<endl;  }  };  class B: public A  {  public:  int b=20;  void DisplayB()  {  cout<<"DisplayB : b="<<b<<endl;  }  };  main()  {  B b1;  b1.DisplayA();  b1.DisplayB();  } | |
| **Output:**  DisplayA : a=10  DisplayB : b=20 | |
| **Result:** The above experiment successfully completed. | |

|  |  |
| --- | --- |
| **Ex. No.: 14** | **Date:** DD/MM/YYYY |
| **Experiment Heading** | |
| **Aim:**  Programs to Overload Unary & Binary Operators as Member Function & Non Member Function, considered as Binary operator as non-member function | |
| **Program: As a member function and non member function**  #include<iostream> using namespace std; class num {     private:     int a,b,c;     public:     num(int j,int k,int m)     {         a=j;b=k;c=m;     }     void show(void);     void operator ++( ); }; void num::show() {     cout<<“\n a= “<<a<<“\n b= “<<b<<“\n c= “<<c; } void num::operator ++( ) {     ++a;     ++b;     ++c; } int main() {     num n(13,63,241);     n.show();     ++n;     n.show(); } | |
| **Output:**  a=14 b=64 c=242 | |
| **Result:** The above experiment successfully completed. | |

|  |  |
| --- | --- |
| **Ex. No.: 14** | **Date:** DD/MM/YYYY |
| **Experiment Heading** | |
| **Aim:**  Programs to Overload Unary & Binary Operators as Member Function & Non Member Function, considered as Binary operator as non-member function | |
| **Program:**  #include<iostream> using namespace std; class FBOP {     int x,y;     public:     FBOP(){   }     FBOP(int a,int b)     {         x=a;         y=b;     }     void show()     {         cout<<“x= “<<x<<“\t y= “<<y<<endl;     }     friend FBOP operator+(FBOP ob1,FBOP ob2); }; FBOP operator+(FBOP ob1,FBOP ob2) {     FBOP temp;     temp.x=ob1.x+ob2.x;     temp.y=ob1.y+ob2.y;     return temp; } int main() {     int a,b;     cout<<“Enter first object values:\n”;     cin>>a>>b;     FBOP ob1(a,b);     cout<<“Enter second object values:\n”;     cin>>a>>b;     FBOP ob2(a,b),ob3;     ob3=ob1+ob2;     cout<<“Values of first object are:\n”;     ob1.show();     cout<<“Values of second object are:\n”;     ob2.show();     cout<<“Addition of two objects is:\n”;     ob3.show(); } | |
| **Output:**  Enter first object values: 12           36 Enter second object values: 42           15 Values of first object are: x=12      y=36 Values of second object are: x=42      y=15 Addition of two objects is: x=54      y=51 | |
| **Result:** The above experiment successfully completed. | |

|  |  |
| --- | --- |
| **Ex. No.: 15** | **Date:** DD/MM/YYYY |
| **Experiment Heading** | |
| **Aim:**  Programs on Class Templates | |
| **Program:**  // C++ program to demonstrate the use of class templates  #include <iostream>  using namespace std;  // Class template  template <class T>  class Number {  private:  // Variable of type T  T num;  public:  Number(T n) : num(n) {} // constructor  T getNum() {  return num;  }  };  int main() {  // create object with int type  Number<int> numberInt(7);  // create object with double type  Number<double> numberDouble(7.7);  cout << "int Number = " << numberInt.getNum() << endl;  cout << "double Number = " << numberDouble.getNum() << endl;  return 0;  } | |
| **Output:**  Int number = 7  double number =7.7 | |
| **Result:** The above experiment successfully completed. | |

|  |  |
| --- | --- |
| **Ex. No.: 3** | **Date:** DD/MM/YYYY |
| **Experiment Heading** | |
| **Aim:**  Write program on Constructors & destructors. | |
| **Program:**  #include<iostream>  **using** **namespace** std;  **class** **class\_name**{    *// declaring private class data members*  **private**:  **int** a,b;    **public**:    *// declaring Constructor*  class\_name(**int** aa, **int** bb)  {  cout<<"Constructor is called"<<endl;  a = aa;  b = bb;    cout<<"Value of a: "<<a<<endl;  cout<<"Value of b: "<<b<<endl;  cout<<endl;  }    *// declaring destructor*  ~class\_name()  {  cout<<"Destructor is called"<<endl;  cout<<"Value of a: "<<a<<endl;  cout<<"Value of b: "<<b<<endl;  }    };  **int** **main**()  {  *// creating objects of class using parameterized constructor*  class\_name **obj**(5,6);    **return** 0;  } | |
| **Output:**  Constructor is called  Value of a: 5  Value of b: 6  Destructor is called  Value of a: 5  Value of b: 6 | |
| **Result:** The above experiment successfully completed. | |