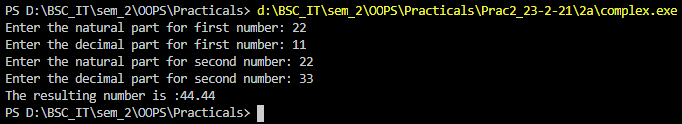
Practical 2: Using friend functions.

1. Write a friend function for adding the two complex numbers, using a single class.

Code:

|  |
| --- |
| // Write a friend function for adding the two complex numbers, using a single class  #include<iostream>  class complexNumber  {  private:      int natural;      int decimal;      friend complexNumber addNum(complexNumber i, complexNumber j);  public:      complexNumber(){          natural = 0;          decimal = 0;      }      void format(complexNumber X)      {          std::cout << "The resulting number is :" << X.natural<<"."<<X.decimal;      }      complexNumber(int x, int y);      ~complexNumber();  };  int main()  {      int i, j;      std::cout << "Enter the natural part for first number: ";      std::cin >> i;      std::cout << "Enter the decimal part for first number: ";      std::cin >> j;      complexNumber Num1(i, j);      std::cout << "Enter the natural part for second number: ";      std::cin >> i;      std::cout << "Enter the decimal part for second number: ";      std::cin >> j;      complexNumber Num2(i, j);      complexNumber result;      result.format(addNum(Num1, Num2));  }  complexNumber addNum(complexNumber i, complexNumber j)  {      complexNumber Ans;      Ans.natural = i.natural + j.natural;      Ans.decimal = i.decimal + j.decimal;      return Ans;  }  complexNumber::complexNumber(int x, int y)  {      natural = x;      decimal = y;  }  complexNumber::~complexNumber()  {  } |

Output:

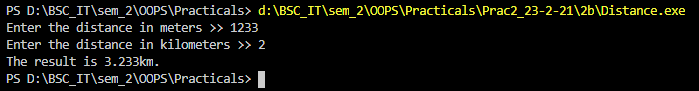


1. Write a friend function for adding the two different distances and display its sum, using two classes.

Code:

|  |
| --- |
| // Write a friend function for adding the two different distances and display its sum, using two classes.  #include<iostream>  class kiloMeters;  class meters  {  private:      float values;      friend kiloMeters addDist(meters, kiloMeters);  public:      meters(int x);      ~meters();  };  class kiloMeters  {  private:      float value;      friend kiloMeters addDist(meters, kiloMeters);  public:      kiloMeters()      {          value = 0;      }      void disp(kiloMeters X)      {          std::cout << "The result is "<<X.value<<"km.";      }      kiloMeters(int x);      ~kiloMeters();  };  int main()  {      float i;      std::cout << "Enter the distance in meters >> ";      std::cin >> i;      meters var1(i);      std::cout << "Enter the distance in kilometers >> ";      std::cin >> i;      kiloMeters var2(i);      kiloMeters result;      result.disp(addDist(var1, var2));  }  kiloMeters::kiloMeters(int x)  {      value = x;  }  kiloMeters::~kiloMeters()  {  }  meters::meters(int x)  {      values = x;  }  meters::~meters()  {  }  kiloMeters addDist(meters x, kiloMeters y)  {      meters temp = x;      float meterCon =  temp.values / 1000;      kiloMeters sum;      sum.value = y.value + meterCon;      return sum;  } |

Output:

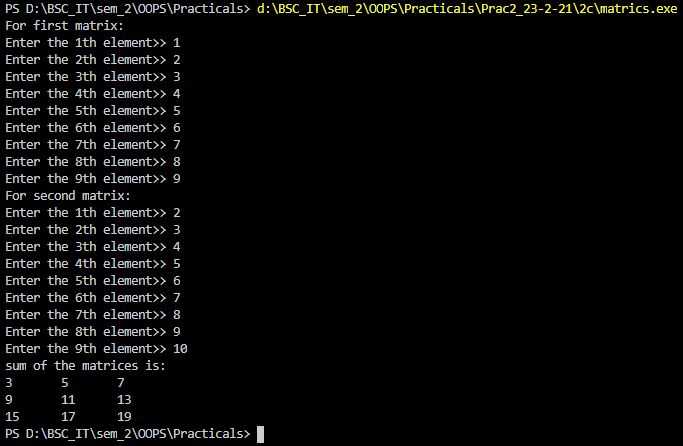


1. Write a friend function for adding the two matrices from two different classes and display its sum.

Code:

|  |
| --- |
| // Write a friend function for adding the two matrixA from two different classes and display its sum  #include<iostream>  class matrixX32;  class matrixX3  {  private:      int matrixA[3][3];  public:      friend matrixX32 add32x3(matrixX3 X, matrixX32 Y);      matrixX3();      void buildmatrix();      ~matrixX3(){}  };  class matrixX32  {  private:      int matrix4[3][3];  public:      friend matrixX32 add32x3(matrixX3 X, matrixX32 Y);      matrixX32();      void buildmatrix();      void disp();      ~matrixX32(){}  };  int main(void)  {      std::cout << "For first matrix: \n";      matrixX3 first;      first.buildmatrix();      std::cout << "For second matrix: \n";      matrixX32 second;      second.buildmatrix();      matrixX32 sum;      sum = add32x3(first, second);      std::cout << "sum of the matrices is: \n";      sum.disp();  }  matrixX3::matrixX3()  {      for(int i = 0; i < 3; i++)      {          for(int j = 0; j < 3; j++)          {              matrixA[i][j] = 0;          }      }  }  matrixX32::matrixX32()  {      for(int i = 0; i < 3; i++)      {          for(int j = 0; j < 3; j++)          {              matrix4[i][j] = 0;          }      }  }  void matrixX32::disp()  {      for(int i = 0; i < 3; i++)      {          for(int j = 0; j < 3; j++)          {              std::cout << matrix4[i][j] <<"\t";          }          std::cout << std::endl;      }  }  void matrixX3::buildmatrix()  {      int input = 0, count = 0;      for(int i = 0; i < 3; i++)      {          for(int j = 0; j < 3; j++)          {              std::cout << "Enter the " << count + 1 << "th element>> ";              std::cin >> input;              matrixA[i][j] = input;              count++;          }      }  }  void matrixX32::buildmatrix()  {      int input = 0, count = 0;      for(int i = 0; i < 3; i++)      {          for(int j = 0; j < 3; j++)          {              std::cout << "Enter the " << count + 1 << "th element>> ";              std::cin >> input;              matrix4[i][j] = input;              count++;          }      }  }  matrixX32 add32x3(matrixX3 X, matrixX32 Y)  {      matrixX32 result;      for(int i = 0; i < 3; i++)      {          for(int j = 0; j < 3; j++)          {              result.matrix4[i][j] = X.matrixA[i][j] + Y.matrix4[i][j];          }      }      return result;  } |

Output:



Write-up:

