**North South University - Spring 2023**

Course: CSE225L Assessment: HW1

Section: 06 NSU ID: 2211424642 Name: Joy Kumar Ghosh

// Task-1 Codes

|  |
| --- |
| #include <iostream>  #include <cmath>  using namespace std;  int main()  {  double radius, area, perimeter;  cout << "Enter radius of a circle: ";  cin >> radius;  area = M\_PI \* radius \* radius;  perimeter = 2.0 \* M\_PI \* radius;  printf("Area: %.2f\nPerimeter: %.2f\n", area, perimeter);  return 0;  } |
|  |
|  |

// Task-2 Codes

|  |
| --- |
| #include <iostream>  using namespace std;  int isPerfect(int x)  {  int sum = 0;  for(int i =1; i <= x/2; i++){  if(x % i == 0){  sum += i;  }  }  if(sum == x && x > 0)  return 1;  else  return 0;  } |
| int main()  {  int minimum, maximum;  cout << "Enter value of a: ";  cin >> minimum;  cout << "Enter value of b: ";  cin >> maximum;  cout << endl << "All Perfect Numbers between (" << minimum << "-" << maximum << "): " << endl;  for(int i = minimum; i <= maximum; i++){  if(isPerfect(i)){  cout << i << " ";  }  }  cout << endl;  return 0;  } |
|  |

// Task-3 Codes

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int i, j;  for(i = 1; i <= 5; i++){  for(j = 1; j <= i; j++){  if(i % 2 == 0){  cout << "+ ";  }  else{  cout << "\* ";  }  }  cout << endl;  }  return 0;  } |
|  |
|  |

// Task-4 Codes

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int numberOfCourses;  double totalCredit, totalGradePoint;  cout << "Enter total course number: ";  cin >> numberOfCourses;  double\* creditHours = new double[numberOfCourses];  double\* gradePoints = new double[numberOfCourses];  cout << endl;  for(int i = 0; i < numberOfCourses; i++){  cout << "For " << (i+1) << "-th Course:" << endl;  cout << "Enter Credit Hours: ";  cin >> creditHours[i];  totalCredit += creditHours[i];  cout << "Enter Grade Points(Max 4.00): ";  cin >> gradePoints[i];  totalGradePoint += (gradePoints[i] \* creditHours[i]);  }  printf("\n\nCGPA: %.2f\n\n", (totalGradePoint / totalCredit));  delete[] creditHours;  delete[] gradePoints;  return 0;  } |
|  |
|  |

// Task-5 Codes

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | Rectangle<T> | | - length: T  - width: T | | + Rectangle()  + Rectangle(l: T, w: T)  + setLength(l: T): void  + setWidth(w: T): void  + getLength(): T  + getWidth(): T  + print(): void  + getArea(): double  + getPerimeter(): T | |
| #include <iostream>  using namespace std;  // declaration file - Rectangle.h  template <class T>  class Rectangle {  private:  T length;  T width;  public:  Rectangle();  Rectangle(T, T);  void setLength(T l);  void setWidth(T w);  T getLength();  T getWidth();  void print();  double getArea();  T getPerimeter();  }; |
| // definition file - Rectangle.cpp  template <class T>  Rectangle<T>::Rectangle() {  length = 1;  width = 1;  }  template <class T>  Rectangle<T>::Rectangle(T l, T w) {  length = l;  width = w;  }  template <class T>  void Rectangle<T>::setLength(T l) {  length = l;  }  template <class T>  void Rectangle<T>::setWidth(T w) {  width = w;  }  template <class T>  T Rectangle<T>::getLength(){  return length;  }  template <class T>  T Rectangle<T>::getWidth(){  return width;  }  template <class T>  void Rectangle<T>::print() {  cout << "Rectangle:\nLength: " << length << "\nWidth: " << width << "\nArea: " << getArea() << "\nPerimeter: " << getPerimeter() << endl;  }  template <class T>  double Rectangle<T>::getArea() {  return (length \* width);  }  template <class T>  T Rectangle<T>::getPerimeter(){  return (2 \* (length + width));  } |
| int main() {  // create object 1  Rectangle<double> r1(4.2, 5.3);  // print object 1's info  r1.print();  cout << endl;  // create object 2  Rectangle<int> r2(4, 5);  // print object 2's info  r2.print();  cout << endl;  return 0;  } |
|  |
|  |

// Task-6 Codes

|  |
| --- |
| #include <iostream>  #include <cmath>  using namespace std;  // declaration file Complex.h  class Complex{  private:  double real;  double imaginary;  public:  Complex();  Complex(double, double);  Complex operator+(Complex);  Complex operator-(Complex);  Complex operator\*(Complex);  Complex operator/(Complex);  bool isEqual(Complex);  double getModulus();  void print();  }; |
| // definition file Complex.cpp  Complex::Complex(){  real = 0;  imaginary = 0;  }  Complex::Complex(double r, double i){  real = r;  imaginary = i;  }  Complex Complex::operator+(Complex a){  Complex t;  t.real = real + a.real;  t.imaginary = imaginary + a.imaginary;  return t;  }  Complex Complex::operator-(Complex a){  Complex t;  t.real = real - a.real;  t.imaginary = imaginary - a.imaginary;  return t;  }  Complex Complex::operator\*(Complex a){  Complex t;  t.real = (real \* a.real) - (imaginary \* a.imaginary);  t.imaginary = (real \* a.imaginary) + (imaginary \* a.real);  return t;  }  Complex Complex::operator/(Complex a){  Complex t;  t.real = (real \* a.real + imaginary \* a.imaginary) / (a.real \* a.real + a.imaginary \* a.imaginary);  t.imaginary = (imaginary \* a.real - real \* a.imaginary) / (a.real \* a.real + a.imaginary \* a.imaginary);  return t;  }  bool Complex::isEqual(Complex a){  return (real == a.real && imaginary == a.imaginary);  }  double Complex::getModulus(){  return sqrt(real \* real + imaginary \* imaginary);  }  void Complex::print(){  cout << "Real: " << real << endl;  cout << "Imaginary: " << imaginary << endl;  } |
| // definition file Complex.cpp  Complex::Complex(){  real = 0;  imaginary = 0;  }  Complex::Complex(double r, double i){  real = r;  imaginary = i;  }  Complex Complex::operator+(Complex a){  Complex t;  t.real = real + a.real;  t.imaginary = imaginary + a.imaginary;  return t;  }  Complex Complex::operator-(Complex a){  Complex t;  t.real = real - a.real;  t.imaginary = imaginary - a.imaginary;  return t;  }  Complex Complex::operator\*(Complex a){  Complex t;  t.real = (real \* a.real) - (imaginary \* a.imaginary);  t.imaginary = (real \* a.imaginary) + (imaginary \* a.real);  return t;  }  Complex Complex::operator/(Complex a){  Complex t;  t.real = (real \* a.real + imaginary \* a.imaginary) / (a.real \* a.real + a.imaginary \* a.imaginary);  t.imaginary = (imaginary \* a.real - real \* a.imaginary) / (a.real \* a.real + a.imaginary \* a.imaginary);  return t;  }  bool Complex::isEqual(Complex a){  return (real == a.real && imaginary == a.imaginary);  }  double Complex::getModulus(){  return sqrt(real \* real + imaginary \* imaginary);  }  void Complex::print(){  cout << "Real: " << real << endl;  cout << "Imaginary: " << imaginary << endl;  } |
|  |
|  |