|  | **National University of Computer and Emerging Sciences (Lahore)** | | | |
| --- | --- | --- | --- | --- |
| **Course:** | **OOP** | **Code:** | **CS217** |
| **Section:** | **BSCS-2B** | **Semester:** | **Spring 2024** |
| **Time:** | **40 minutes** | **TotalMarks:** | **20** |
| **Date:** |  | **ID:** | **Quiz-2A** |
| **Name:** |  | **Roll no:** |  |

**Question:**

**Implement a class named ComplexNumberArray which represents an array of complex numbers. A complex number is a number that can be represented in the form of a + bi, where a and b are real numbers and i is the solution to the equation x^2 = −1.**

1. **Define private data members real and imaginary to represent the real and imaginary parts of the complex number, respectively. Both of these data members are supposed to be dynamically allocated arrays of type <*double>*. The class should also have a data member int size which stores the size of arrays of real and imaginary numbers.Include a static data member totalCount to keep track of the total number of ComplexNumberArray objects created. [2]**
2. **Implement a default constructor that initializes both real and imaginary parts to NULL and size to 0: ComplexNumberArray(); [1]**
3. **Implement a default parameterized constructor that takes two arrays as arguments and initializes the real and imaginary parts accordingly. Default values for the pointers should be NULL and 0 for size. [1]**
4. **Implement getters for all the data members. [.5 each]**
5. **Implement a destructor: ~ComplexNumberArray(); [1]**
6. **Implement a copy constructor for deep copy that performs a shallow copy of the ComplexNumbers object: ComplexNumberArray(const ComplexNumberArray& obj); [2]**
7. **Implement an assignment operator overload that performs a deep copy of the ComplexNumbers object. [2]**
8. **Overload the \* operator within the ComplexNumberArray class, which takes another ComplexNumberArray object as an argument and returns a new ComplexNumberArray object representing the element-wise product of the two arrays. The product of two complex numbers (a + bi) and (c + di) is calculated as (ac - bd) + (ad + bc)i for each corresponding pair of numbers in the arrays. Show error in case the size of both ComplexNumberArrays is not equal. [2]**
9. **Overload the -= operator to add another ComplexNumbers object to the current object and update the current object (calling object). Show error in case the size of both ComplexNumberArrays is not equal. [2]**
10. **Overload the [] operator to access individual complex numbers in the arrays. This operator should return a pair of the real and complex numbers as an ordered pair: double\* operator[](int index) const; [2]**
11. **Add a method called magnitude() that calculates and returns the magnitude of each complex number in the array. The magnitude of a complex number a + bi is given by the square root of (a^2 + b^2): double\* magnitude() const; [2]**
12. **Implement a member function print() to print the complex number in the format “a+bi”: void print() const; [1]**

**class ComplexNumberArray {**

**private:**

**double\* real;**

**double\* imaginary;**

**int size;**

**static int totalCount;**

**public:**

**// Constructors**

**ComplexNumberArray();**

**ComplexNumberArray(double\* realArr, double\* imagArr, int s = 0);**

**ComplexNumberArray(const ComplexNumberArray& obj);**

**// Destructor**

**~ComplexNumberArray();**

**// Assignment operator**

**ComplexNumberArray& operator=(const ComplexNumberArray& obj);**

**// Overloading operators**

**ComplexNumberArray operator\*(const ComplexNumberArray& obj) const;**

**ComplexNumberArray& operator-=(const ComplexNumberArray& obj);**

**double\* operator[](int index) const;**

**// Method to return the magnitude of each complex number in the array**

**double\* magnitude() const;**

**// Getter methods**

**int getSize() const;**

**static int getTotalCount();**

**// Member function to print the complex number**

**void print() const;**

**};**

**// Default constructor**

**ComplexNumberArray::ComplexNumberArray() : real(nullptr), imaginary(nullptr), size(0) {**

**totalCount++;**

**}**

**// Parameterized constructor**

**ComplexNumberArray::ComplexNumberArray(double\* realArr, double\* imagArr, int s) : size(s) {**

**real = new double[size];**

**imaginary = new double[size];**

**for (int i = 0; i < size; ++i) {**

**real[i] = realArr[i];**

**imaginary[i] = imagArr[i];**

**}**

**totalCount++;**

**}**

**// Copy constructor**

**ComplexNumberArray::ComplexNumberArray(const ComplexNumberArray& obj) : size(obj.size) {**

**real = new double[size];**

**imaginary = new double[size];**

**for (int i = 0; i < size; ++i) {**

**real[i] = obj.real[i];**

**imaginary[i] = obj.imaginary[i];**

**}**

**totalCount++;**

**}**

**// Destructor**

**ComplexNumberArray::~ComplexNumberArray() {**

**delete[] real;**

**delete[] imaginary;**

**totalCount--;**

**}**

**// Assignment operator**

**ComplexNumberArray& ComplexNumberArray::operator=(const ComplexNumberArray& obj) {**

**if (this == &obj)**

**return \*this;**

**delete[] real;**

**delete[] imaginary;**

**size = obj.size;**

**real = new double[size];**

**imaginary = new double[size];**

**for (int i = 0; i < size; ++i) {**

**real[i] = obj.real[i];**

**imaginary[i] = obj.imaginary[i];**

**}**

**return \*this;**

**}**

**// Overloading the \* operator**

**ComplexNumberArray ComplexNumberArray::operator\*(const ComplexNumberArray& obj) const {**

**if (size != obj.size) {**

**throw std::invalid\_argument("Error: Array sizes are not equal for element-wise multiplication.");**

**}**

**ComplexNumberArray result;**

**result.size = size;**

**result.real = new double[size];**

**result.imaginary = new double[size];**

**for (int i = 0; i < size; ++i) {**

**result.real[i] = (real[i] \* obj.real[i]) - (imaginary[i] \* obj.imaginary[i]);**

**result.imaginary[i] = (real[i] \* obj.imaginary[i]) + (imaginary[i] \* obj.real[i]);**

**}**

**return result;**

**}**

**// Overloading the -= operator**

**ComplexNumberArray& ComplexNumberArray::operator-=(const ComplexNumberArray& obj) {**

**if (size != obj.size) {**

**throw std::invalid\_argument("Error: Array sizes are not equal for subtraction.");**

**}**

**for (int i = 0; i < size; ++i) {**

**real[i] -= obj.real[i];**

**imaginary[i] -= obj.imaginary[i];**

**}**

**return \*this;**

**}**

**// Overloading the [] operator**

**double\* ComplexNumberArray::operator[](int index) const {**

**if (index < 0 || index >= size) {**

**throw std::out\_of\_range("Error: Index out of range.");**

**}**

**double\* num = new double[2];**

**num[0] = real[index];**

**num[1] = imaginary[index];**

**return num;**

**}**

**// Method to return the magnitude of each complex number in the array**

**double\* ComplexNumberArray::magnitude() const {**

**double\* magnitudes = new double[size];**

**for (int i = 0; i < size; ++i) {**

**magnitudes[i] = std::sqrt(real[i] \* real[i] + imaginary[i] \* imaginary[i]);**

**}**

**return magnitudes;**

**}**

**// Getter methods**

**int ComplexNumberArray::getSize() const {**

**return size;**

**}**

**int ComplexNumberArray::getTotalCount() {**

**return totalCount;**

**}**

**// Member function to print the complex number**

**void ComplexNumberArray::print() const {**

**for (int i = 0; i < size; ++i) {**

**std::cout << real[i] << " + " << imaginary[i] << "i" << std::endl;**

**}**

**}**

**// Static data member initialization**

**int ComplexNumberArray::totalCount = 0;**