

# XJTLU Entrepreneur College (Taicang) Cover Sheet

Module code and Title	<b>DTS102TC Programming with C++</b>		
School Title	<b>School of Artificial Intelligence and Advanced Computing</b>		
Assignment Title	<b>Coursework 1 (Assignment)</b>		
Submission Deadline	<b>5 pm China time (UTC+8 Beijing) on Fri. 28th. Nov. 2025</b>		
Final Word Count	<b>NA</b>		
If you agree to let the university use your work anonymously for teaching and learning purposes, please type “yes” here.			<b>yes</b>

I certify that I have read and understood the University’s Policy for dealing with Plagiarism, Collusion and the Fabrication of Data (available on Learning Mall Online). With reference to this policy I certify that:

- My work does not contain any instances of plagiarism and/or collusion.
- My work does not contain any fabricated data.

**By uploading my assignment onto Learning Mall Online, I formally declare that all of the above information is true to the best of my knowledge and belief.**

Scoring – For Tutor Use	
<b>Student ID: 2468293</b>	<b>Student Name: Aibo Ge</b>

Stage of Marking	Marker Code	Learning Outcomes Achieved (F/P/M/D) (please modify as appropriate)			Final Score
		A	B	C	
1 <sup>st</sup> Marker – red pen					
Moderation – green pen	<b>IM Initials</b>	The original mark has been accepted by the moderator (please circle as appropriate):			Y / N
		Data entry and score calculation have been checked by another tutor (please circle):			Y
2 <sup>nd</sup> Marker if needed – green pen					
<b>For Academic Office Use</b>		<b>Possible Academic Infringement (please tick as appropriate)</b>			
<b>Date Received</b>	<b>Days late</b>	<b>Late Penalty</b>	<input type="checkbox"/> <b>Category A</b> <input type="checkbox"/> <b>Category B</b> <input type="checkbox"/> <b>Category C</b> <input type="checkbox"/> <b>Category D</b> <input type="checkbox"/> <b>Category E</b>		Total Academic Infringement Penalty (A,B, C, D, E, Please modify where necessary) _____

# DTS102TC Programming with C++

## Coursework 1 (Assignment)

### Question 1. Algebra: solve quadratic equations (5 marks)

```
1 #include <cmath>
2 #include <iostream>
3 using namespace std;
4
5 int main() {
6     // WRITE YOUR CODE HERE, DO NOT CHANGE THE TEMPLATE
7     double a, b, c;
8     double r1, r2, d;
9
10    cout << "Enter a, b, c: ";
11
12    cin >> a >> b >> c;
13
14    d = b * b - 4 * a * c;
15
16    if (d > 0) {
17        r1 = (-b + pow(d, 0.5)) / (2 * a);
18        r2 = (-b - pow(d, 0.5)) / (2 * a);
19        cout << "The roots are " << r1 << " and " << r2 << endl;
20    } else if (d == 0) {
21        r1 = -b / (2 * a);
22        cout << "The root is " << r1 << endl;
23    } else {
24        cout << "The equation has no real roots" << endl;
25    }
26
27    return 0;
28 }
```

```
(base) ~ C:\Source Code Template\git:(main) x cd "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/" && g++ 01-quadratic.cpp -o 01-quadratic && "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/"01-quadratic
Enter a, b, c: 1 -3 2
The roots are 2 and 1
(base) ~ C:\Source Code Template\git:(main) x cd "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/" && g++ 01-quadratic.cpp -o 01-quadratic && "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/"01-quadratic
Enter a, b, c: 1 -2 1
The root is 1
(base) ~ C:\Source Code Template\git:(main) x cd "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/" && g++ 01-quadratic.cpp -o 01-quadratic && "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/"01-quadratic
Enter a, b, c: 1 0 1
The equation has no real roots
```

The core functionality is implemented within the main function, which solves quadratic equations of the form  $ax^2 + bx + c = 0$ . The program first reads the coefficients  $a$ ,  $b$ , and  $c$ . The critical step is the calculation of the discriminant,  $d = b^2 - 4ac$ , which determines the nature of the roots.

The program uses a conditional structure (if-else if-else) to handle three

scenarios:

1. If  $d > 0$ , the equation has two distinct real roots. These are calculated using the quadratic formula  $(-b \pm \sqrt{d}) / 2a$ . The `pow(d, 0.5)` function is used to compute the square root.
2. If  $d = 0$ , the equation has exactly one real root, calculated as  $-b / 2a$ .
3. If  $d < 0$ , the equation has no real roots.

This logic ensures all possible cases for real coefficients are handled correctly.

### Question 2. Geometry: area of a regular polygon (5 marks)

```
1 #include <cmath>
2 #include <iostream>
3 using namespace std;
4
5 int main() {
6     // WRITE YOUR CODE HERE, DO NOT CHANGE THE TEMPLATE
7
8     int n;
9     double s, area;
10
11    cout << "Enter the number of sides: ";
12    cin >> n;
13
14    cout << "Enter the length of a side: ";
15    cin >> s;
16
17    const double PI = 3.14159;
18    area = (n * s * s) / (4 * tan(PI / n));
19
20    cout << "The area of the polygon is " << area << endl;
21
22    return 0;
23 }
24
```

```
(base) ~ C:\Source Code Template\git:(main) x cd "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/" && g++ 02-polygon-area.cpp -o 02-polygon-area && "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/"02-polygon-area
Enter the number of sides: 5
Enter the length of a side: 6.5
The area of the polygon is 72.6903
```

The program calculates the area of a regular polygon given the number of sides ( $n$ ) and the side length ( $s$ ). The mathematical formula used is  $\text{Area} = (n * s^2) / (4 * \tan(\pi / n))$ .

The implementation defines  $\pi$  as a constant double for precision. It utilizes the `tan` function from the `<cmath>` library. The expression  $(n * s * s) / (4 * \tan(\pi / n))$  directly translates the mathematical formula into C++ code. The

result is stored in a double variable to accommodate floating-point values.

### Question 3. Count positive and negative numbers and compute the average of numbers (5 marks)

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     // WRITE YOUR CODE HERE, DO NOT CHANGE THE TEMPLATE
6
7     int sum;
8     int countPositive = 0;
9     int countNegative = 0;
10    int totalCount = 0;
11    int num = 0;
12
13    cout << "Enter an integer value, the input ends if it is 0: ";
14    cin >> num;
15
16    while (num != 0) {
17        if (num > 0) {
18            countPositive++;
19        }
20        else {
21            countNegative++;
22        }
23        sum += num;
24        totalCount++;
25        cin >> num;
26    }
27
28    cout << "The number of positives is " << countPositive << endl;
29    cout << "The number of negatives is " << countNegative << endl;
30    cout << "The total value is " << sum << endl;
31    cout << "The average value is " << (totalCount == 0 ? 0.0 : static_cast<double>(sum) / totalCount) << endl;
32
33    return 0;
34 }
```

```
(base) ➜ CW1 Source Code Template git:(main) x cd "/Users/albert/Documents/CW1/CW1 Source Code Template/" && g++ 03-counter.cpp -o 03-counter && ./03-counter
Enter an integer value, the input ends if it is 0: 34 26 52 48 31 48 83 32 0
The number of positives is 8
The number of negatives is 0
The total value is 354
The average value is 44.25
```

The program analyzes a stream of integer inputs ending with 0. It employs a while loop that continues execution as long as the input is not 0 (the sentinel value).

Inside the loop, conditional statements classify each number as positive or negative, incrementing the corresponding counters (countPositive, countNegative). Every non-zero

number is added to a running sum.

After the loop terminates, the average is calculated. A `static_cast<double>(sum)` is used to convert the integer sum to a floating-point number before division, ensuring the average is computed with decimal precision rather than performing integer division.

### Question 4. Binary to decimal (5 marks)

```
1 #include <iostream>
2 #include <string>
3
4 int solve_bin_to_dec(const std::string& binaryString) {
5     // WRITE YOUR CODE HERE, DO NOT CHANGE THE TEMPLATE
6
7     int decimal = 0;
8     int power = 1;
9     for (int i = binaryString.length() - 1; i >= 0; i--) {
10        if (binaryString[i] == '1') {
11            decimal += power;
12        }
13        power *= 2;
14    }
15    return decimal;
16
17 int main() {
18    std::cout << "Enter a binary number: ";
19    std::string binaryString;
20    std::cin >> binaryString;
21    std::cout << solve_bin_to_dec(binaryString) << std::endl;
22
23    return 0;
24 }
```

```
(base) ➜ CW1 Source Code Template git:(main) x cd "/Users/albert/Documents/CW1/CW1 Source Code Template/" && g++ 04-bin2dec.cpp -o 04-bin2dec && ./04-bin2dec
Enter a binary number: 10001
17
(base) ➜ CW1 Source Code Template git:(main) x cd "/Users/albert/Documents/CW1/CW1 Source Code Template/" && g++ 04-bin2dec.cpp -o 04-bin2dec && ./04-bin2dec
Enter a binary number: 11011111101010010000000000
58631168
(base) ➜ CW1 Source Code Template git:(main) x
```

The function `solve_bin_to_dec` converts a binary string representation into a decimal integer. The algorithm iterates through the string from the last character (least significant bit) to the first (most significant bit).

A variable 'power' tracks the place value of the current bit (1, 2, 4, 8, ...). In

each iteration:

1. If the current character is '1', the current 'power' value is added to the total decimal result.
2. The 'power' is multiplied by 2 to prepare for the next bit position.

This approach efficiently converts the binary number in a single pass without the overhead of repeated calls to a power function.

### Question 5. Print distinct numbers (5 marks)

```

1 #include <iostream>
2
3 int main() {
4     // Write your code here. Do not change the template.
5
6     int numbers[10]; // Store distinct values
7     int size = 0;    // Indicate how many distinct values are in the array numbers
8
9     std::cout << "Enter ten integers: ";
10
11     int temp;
12     for (int i = 0; i < 10; i++) {
13         std::cin >> temp;
14         bool isDistinct = true;
15         for (int j = 0; j < size; j++) {
16             if (numbers[j] == temp) {
17                 isDistinct = false;
18                 break;
19             }
20         }
21         if (isDistinct) {
22             numbers[size] = temp;
23             size++;
24         }
25     }
26
27     std::cout << "The number of distinct numbers is " << size << std::endl;
28     std::cout << "The distinct numbers are: "; // Note: Sample output shows space after colon! No, sample shows "The distinct numbers are: 1 2 3 4 5"
29     for (int i = 0; i < size; i++) {
30         std::cout << " " << numbers[i];
31     }
32     std::cout << std::endl;
33
34     return 0;
35 }

```

```

(base) ~ % cd "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/" && g++ 06-distinct-numbers.cpp -std=c++11 -o 06-distinct-numbers
Enter ten integers: 2 7 1 3 0 9 5 9 4 2 6 1 5 2
The number of distinct numbers is 8
The distinct numbers are: 2 7 1 3 0 9 5 4

```

The program reads ten integers and filters out duplicates to display only distinct numbers. It utilizes an array 'numbers' to store the unique values identified so far and a counter 'size' to track the count of these unique values.

For each new input, a nested loop performs a linear search through the 'numbers' array. A boolean flag 'isDistinct' is used to indicate if the

number is already present. If the loop completes without finding the number, it is considered distinct, added to the array, and the size counter is incremented. This ensures that the output contains each number exactly once, preserving their original relative order.

## Question 6. Sum elements in each column (10 marks)

```

1 #include <iostream>
2
3 const int SIZE = 4;
4
5 double sumColumn(const double m[SIZE][SIZE], int rowSize, int columnIndex) {
6     // Write your code here. Do not change the template.
7
8     double sum = 0;
9     for (int i = 0; i < rowSize; i++) {
10         sum += m[i][columnIndex];
11     }
12     return sum;
13 }
14
15 int main() {
16     double matrix[SIZE][SIZE];
17
18     std::cout << "Enter a 3-by-4 matrix row by row " << std::endl;
19
20     for (int i = 0; i < 3; i++) {
21         for (int j = 0; j < SIZE; j++) {
22             std::cin >> matrix[i][j];
23         }
24     }
25
26     for (int j = 0; j < SIZE; j++) {
27         std::cout << "Sum of the elements at column " << j << " is " << sumColumn(matrix, 3, j) << std::endl;
28     }
29
30     return 0;
31 }

```

```

(base) ~ % cd "/Users/albert/Documents/CodeProjects/DTS102/CW1 Source Code Template/" && g++ 06-sum-column.cpp -std=c++11 -o 06-sum-column
Enter a 3-by-4 matrix row by row:
72 69 75 53
75 1 13 2
22 50 92 17
Sum of the elements at column 0 is 169
Sum of the elements at column 1 is 120
Sum of the elements at column 2 is 180
Sum of the elements at column 3 is 72

```

The function sumColumn computes the sum of elements in a specific column of a matrix. It accepts a 2D array, the number of rows, and the target column index as parameters.

The logic involves a single for-loop that iterates through the rows (from 0 to rowSize - 1) while keeping the column index fixed. In each iteration, the value at the current row and specified column is added to an accumulator variable 'sum'. This effectively traverses the matrix vertically. The main function demonstrates this by calling sumColumn for each column index (0 to 3) of a 3x4 matrix.

## Question 7. The Rectangle class (10 marks)

```

1 // Design a class named Rectangle to represent a rectangle. The class contains:
2 //
3 // 1. Two double data fields named width and height that specify the width and height of the rectangle.
4 // 2. A no-arg constructor that creates a rectangle with width 1 and height 1.
5 // 3. A constructor that creates a rectangle with the specified width and height.
6 // 4. The accessor and mutator functions for all the data fields.
7 // 5. A function named getArea() that returns the area of this rectangle.
8 // 6. A function named getPerimeter() that returns the perimeter.
9
10 #include <iostream>
11 using namespace std;
12
13 class Rectangle {
14 public:
15     Rectangle();
16     Rectangle(double width, double height);
17     double getWidth() const;
18     void setWidth(double width);
19     double getHeight() const;
20     void setHeight(double height);
21     double getArea() const;
22     double getPerimeter() const;
23 private:
24     double width, height;
25 };
26
27 int main() {
28     Rectangle r1;
29     Rectangle r2(2, 3);
30     cout << "Area of r1: " << r1.getArea() << endl;
31     cout << "Area of r2: " << r2.getArea() << endl;
32     return 0;
33 }

```

```

1 #include "Rectangle.h"
2
3 Rectangle::Rectangle() {
4     width = 1;
5     height = 1;
6 }
7
8 Rectangle::Rectangle(double width, double height) {
9     this->width = width;
10    this->height = height;
11 }
12
13 double Rectangle::getWidth() const {
14     return width;
15 }
16
17 void Rectangle::setWidth(double width) {
18     this->width = width;
19 }
20
21 double Rectangle::getHeight() const {
22     return height;
23 }
24
25 void Rectangle::setHeight(double height) {
26     this->height = height;
27 }
28
29 double Rectangle::getArea() const {
30     return width * height;
31 }
32
33 double Rectangle::getPerimeter() const {
34     return 2 * (width + height);
35 }

```

**Test Summary**

- 1. A no-arg constructor that creates a rectangle with width 1 and height 1. (0/0)
- 2. A constructor that creates a rectangle with the specified width and height. (0/0)
- 3. The accessor and mutator functions for all the data fields. (0/0)
- 4. A function named getArea() that returns the area of this rectangle. (0/0)
- 5. A function named getPerimeter() that returns the perimeter of this rectangle. (0/0)

**Assignment Results**

Question 7: 100%

Question 8: 100%

Question 9: 100%

Question 10: 100%

Question 11: 100%

Question 12: 100%

Question 13: 100%

Question 14: 100%

Question 15: 100%

Question 16: 100%

Question 17: 100%

Question 18: 100%

Question 19: 100%

Question 20: 100%

Question 21: 100%

Question 22: 100%

Question 23: 100%

Question 24: 100%

Question 25: 100%

Question 26: 100%

Question 27: 100%

Question 28: 100%

Question 29: 100%

Question 30: 100%

Question 31: 100%

Question 32: 100%

Question 33: 100%

Question 34: 100%

Question 35: 100%

Question 36: 100%

Question 37: 100%

Question 38: 100%

Question 39: 100%

Question 40: 100%

Question 41: 100%

Question 42: 100%

Question 43: 100%

Question 44: 100%

Question 45: 100%

Question 46: 100%

Question 47: 100%

Question 48: 100%

Question 49: 100%

Question 50: 100%

Question 51: 100%

Question 52: 100%

Question 53: 100%

Question 54: 100%

Question 55: 100%

Question 56: 100%

Question 57: 100%

Question 58: 100%

Question 59: 100%

Question 60: 100%

Question 61: 100%

Question 62: 100%

Question 63: 100%

Question 64: 100%

Question 65: 100%

Question 66: 100%

Question 67: 100%

Question 68: 100%

Question 69: 100%

Question 70: 100%

Question 71: 100%

Question 72: 100%

Question 73: 100%

Question 74: 100%

Question 75: 100%

Question 76: 100%

Question 77: 100%

Question 78: 100%

Question 79: 100%

Question 80: 100%

Question 81: 100%

Question 82: 100%

Question 83: 100%

Question 84: 100%

Question 85: 100%

Question 86: 100%

Question 87: 100%

Question 88: 100%

Question 89: 100%

Question 90: 100%

Question 91: 100%

Question 92: 100%

Question 93: 100%

Question 94: 100%

Question 95: 100%

Question 96: 100%

Question 97: 100%

Question 98: 100%

Question 99: 100%

Question 100: 100%

- The Rectangle class encapsulates the properties and behaviors of a geometric rectangle.*
- 1. Data Encapsulation: The width and height are stored as private double variables, ensuring they are modified only through defined interfaces.*
  - 2. Constructors: A no-argument constructor initializes a default rectangle (1x1), while a parameterized constructor allows creating a rectangle with specific dimensions.*
  - 3. Accessors: Public methods getWidth and getHeight provide read access to the private data.*
  - 4. Computation: The getArea method returns the product of width and height, and getPerimeter returns 2 \* (width + height). This design adheres to object-oriented principles by bundling data with the methods that operate on it.*

## Question 8. Geometry: The Rectangle2D class (15 marks)

```

1 // Design a class named Rectangle2D to represent a 2D rectangle. The class contains:
2 //
3 // 1. Two double data fields named x and y that specify the center of the rectangle.
4 // 2. Two double data fields named width and height that specify the width and height of the rectangle.
5 // 3. A no-arg constructor that creates a default rectangle with (0, 0) for the center and 1 for both width and height.
6 // 4. A constructor that creates a rectangle with the specified x, y, width, and height.
7 // 5. A constant function getArea() that returns the area of the rectangle.
8 // 6. A constant function getPerimeter() that returns the perimeter of the rectangle.
9 // 7. A constant function contains(double x, double y) that returns true if the specified point (x, y) is inside this rectangle.
10 // 8. A constant function contains(const Rectangle2D &r) that returns true if the specified rectangle is inside this rectangle.
11 // 9. A constant function overlaps(const Rectangle2D &r) that returns true if the specified rectangle overlaps with this rectangle.
12
13 #include <iostream>
14 using namespace std;
15
16 class Rectangle2D {
17 public:
18     Rectangle2D();
19     Rectangle2D(double x, double y, double width, double height);
20     double getArea() const;
21     double getPerimeter() const;
22     bool contains(double x, double y) const;
23     bool contains(const Rectangle2D &r) const;
24     bool overlaps(const Rectangle2D &r) const;
25 };
26
27 int main() {
28     Rectangle2D r1;
29     Rectangle2D r2(2, 3, 4, 6);
30     cout << "Area of r1: " << r1.getArea() << endl;
31     cout << "Area of r2: " << r2.getArea() << endl;
32     cout << "Perimeter of r1: " << r1.getPerimeter() << endl;
33     cout << "Perimeter of r2: " << r2.getPerimeter() << endl;
34     cout << "r1 contains (0, 0): " << r1.contains(0, 0) << endl;
35     cout << "r1 contains (2, 3): " << r1.contains(2, 3) << endl;
36     cout << "r1 contains r2: " << r1.contains(r2) << endl;
37     cout << "r2 contains r1: " << r2.contains(r1) << endl;
38     cout << "r1 overlaps r2: " << r1.overlaps(r2) << endl;
39     cout << "r2 overlaps r1: " << r2.overlaps(r1) << endl;
40     return 0;
41 }

```

```

1 #include "Rectangle2D.h"
2
3 Rectangle2D::Rectangle2D() {
4     x = 0;
5     y = 0;
6     width = 1;
7     height = 1;
8 }
9
10 Rectangle2D::Rectangle2D(double x, double y, double width, double height) {
11     this->x = x;
12     this->y = y;
13     this->width = width;
14     this->height = height;
15 }
16
17 double Rectangle2D::getArea() const {
18     return width * height;
19 }
20
21 double Rectangle2D::getPerimeter() const {
22     return 2 * (width + height);
23 }
24
25 bool Rectangle2D::contains(double x, double y) const {
26     return (x >= this->x - this->width / 2 && x <= this->x + this->width / 2 && y >= this->y - this->height / 2 && y <= this->y + this->height / 2);
27 }
28
29 bool Rectangle2D::contains(const Rectangle2D &r) const {
30     return (x >= r.x - r.width / 2 && x <= r.x + r.width / 2 && y >= r.y - r.height / 2 && y <= r.y + r.height / 2 && r.width <= this->width && r.height <= this->height);
31 }
32
33 bool Rectangle2D::overlaps(const Rectangle2D &r) const {
34     return !this->contains(r) && (x < r.x + r.width / 2 && x >= r.x - r.width / 2 || y < r.y + r.height / 2 && y >= r.y - r.height / 2);
35 }

```

**Test Summary**

- 1. The double data fields with width and height with constant get functions and set functions. (0/0)
- 2. A no-arg constructor that creates a default rectangle with (0, 0) for the center and 1 for both width and height. (0/0)
- 3. A constructor that creates a rectangle with the specified x, y, width and height. (0/0)
- 4. A constant function getArea() that returns the area of the rectangle. (0/0)
- 5. A constant function getPerimeter() that returns the perimeter of the rectangle. (0/0)
- 6. A constant function contains(double x, double y) that returns true if the specified point (x, y) is inside this rectangle. (0/0)
- 7. A constant function contains(const Rectangle2D &r) that returns true if the specified rectangle is inside this rectangle. (0/0)
- 8. A constant function overlaps(const Rectangle2D &r) that returns true if the specified rectangle overlaps with this rectangle. (0/0)

**Assignment Results**

Question 7: 100%

Question 8: 100%

Question 9: 100%

Question 10: 100%

Question 11: 100%

Question 12: 100%

Question 13: 100%

Question 14: 100%

Question 15: 100%

Question 16: 100%

Question 17: 100%

Question 18: 100%

Question 19: 100%

Question 20: 100%

Question 21: 100%

Question 22: 100%

Question 23: 100%

Question 24: 100%

Question 25: 100%

Question 26: 100%

Question 27: 100%

Question 28: 100%

Question 29: 100%

Question 30: 100%

Question 31: 100%

Question 32: 100%

Question 33: 100%

Question 34: 100%

Question 35: 100%

Question 36: 100%

Question 37: 100%

Question 38: 100%

Question 39: 100%

Question 40: 100%

Question 41: 100%

Question 42: 100%

Question 43: 100%

Question 44: 100%

Question 45: 100%

Question 46: 100%

Question 47: 100%

Question 48: 100%

Question 49: 100%

Question 50: 100%

Question 51: 100%

Question 52: 100%

Question 53: 100%

Question 54: 100%

Question 55: 100%

Question 56: 100%

Question 57: 100%

Question 58: 100%

Question 59: 100%

Question 60: 100%

Question 61: 100%

Question 62: 100%

Question 63: 100%

Question 64: 100%

Question 65: 100%

Question 66: 100%

Question 67: 100%

Question 68: 100%

Question 69: 100%

Question 70: 100%

Question 71: 100%

Question 72: 100%

Question 73: 100%

Question 74: 100%

Question 75: 100%

Question 76: 100%

Question 77: 100%

Question 78: 100%

Question 79: 100%

Question 80: 100%

Question 81: 100%

Question 82: 100%

Question 83: 100%

Question 84: 100%

Question 85: 100%

Question 86: 100%

Question 87: 100%

Question 88: 100%

Question 89: 100%

Question 90: 100%

Question 91: 100%

Question 92: 100%

Question 93: 100%

Question 94: 100%

Question 95: 100%

Question 96: 100%

Question 97: 100%

Question 98: 100%

Question 99: 100%

Question 100: 100%

- The Rectangle2D class extends the rectangle concept to a 2D coordinate space, adding (x, y) coordinates for the center.*
- 1. Point Containment: The contains(double x, double y) method determines if a point lies inside the rectangle. It checks if the point's coordinates fall within the ranges [center\_x - width/2, center\_x + width/2] and [center\_y - height/2, center\_y + height/2].*
  - 2. Rectangle Containment: The contains(const Rectangle2D &r) method checks if another rectangle 'r' is entirely within the current rectangle. This requires that all boundaries of 'r' are within the boundaries of the current rectangle.*
  - 3. Overlap Detection: The overlaps(const Rectangle2D &r) method determines if two rectangles intersect. It uses the logic that two rectangles overlap unless they are completely separated (e.g., one is entirely to the left, right, above, or below the other).*

```

1 #include "Rectangle2D.h"
2
3 #include <stdexcept>
4 #include <iostream>
5 #include <string>
6
7 const int SIZE = 2;
8
9 Rectangle2D::Rectangle2D(const double xcenter[SIZE], int n) {
10     if (n < 0) return Rectangle2D(0, 0, 0, 0);
11
12     double xmid = points[0][0], ymid = points[0][1];
13     double xend = points[1][0], yend = points[1][1];
14
15     for (int i = 1; i < n; i++) {
16         if ((points[i][0] < xmid) || (xmid < points[i][0]))
17             xmid = points[i][0];
18         if ((points[i][1] < ymid) || (ymid < points[i][1]))
19             ymid = points[i][1];
20     }
21
22     double width = xend - xmid;
23     double height = yend - ymid;
24     double centerx = xmid + width / 2;
25     double centery = ymid + height / 2;
26
27     return Rectangle2D(centerx, centery, width, height);
28 }
29
30 Rectangle2D::Rectangle2D(const double points[SIZE][2], int n) {
31     if (n < 0) return new Rectangle2D(0, 0, 0, 0);
32
33     double xmid = points[0][0], ymid = points[0][1];
34     double xend = points[1][0], yend = points[1][1];
35
36     for (int i = 1; i < n; i++) {
37         if ((points[i][0] < xmid) || (xmid < points[i][0]))
38             xmid = points[i][0];
39         if ((points[i][1] < ymid) || (ymid < points[i][1]))
40             ymid = points[i][1];
41     }
42
43     double width = xend - xmid;
44     double height = yend - ymid;
45     double centerx = xmid + width / 2;
46     double centery = ymid + height / 2;
47
48     return new Rectangle2D(centerx, centery, width, height);
49 }
50
51 int main() {
52     double points[SIZE];
53     std::cout << "Enter four points: ";
54     for (int i = 0; i < SIZE; i++)
55         points[i] = point(i);
56
57     Rectangle2D* boundingRectangle = getBoudingRectangle(points, 3);
58     std::cout << "The bounding rectangle is: " << boundingRectangle->getx() << ", " << boundingRectangle->gety()
59     << ", " << boundingRectangle->getWidth() << ", " << height: " << boundingRectangle->getHeight()
60     << std::endl;
61
62     Rectangle2D* boundingRectangle2 = getBoudingRectangle(points, 3);
63     std::cout << "The bounding rectangle is: " << boundingRectangle2->getx() << ", " << boundingRectangle2->gety()
64     << ", " << boundingRectangle2->getWidth() << ", " << height: " << boundingRectangle2->getHeight()
65     << std::endl;
66
67     return 0;
68 }
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

1 #include "Rectangle2D.h"
2
3 Rectangle2D::Rectangle2D() {
4     x = 0;
5     y = 0;
6     width = 1;
7     height = 1;
8 }
9
10 Rectangle2D::Rectangle2D(double x, double y, double width, double height) {
11     this->x = x;
12     this->y = y;
13     this->width = width;
14     this->height = height;
15 }
16
17 double Rectangle2D::getx() const { return x; }
18 void Rectangle2D::setx(double x) { this->x = x; }
19 double Rectangle2D::gety() const { return y; }
20 void Rectangle2D::sety(double y) { this->y = y; }
21 double Rectangle2D::getWidth() const { return width; }
22 void Rectangle2D::setWidth(double width) { this->width = width; }
23 double Rectangle2D::getHeight() const { return height; }
24 void Rectangle2D::setHeight(double height) { this->height = height; }
25
26 double Rectangle2D::getArea() const {
27     return width * height;
28 }
29
30 double Rectangle2D::getPerimeter() const {
31     return 2 * (width + height);
32 }
33
34 bool Rectangle2D::contains(double x, double y) const {
35     return x >= (this->x - width / 2) && x <= (this->x + width / 2) &&
36         y >= (this->y - height / 2) && y <= (this->y + height / 2);
37 }
38
39 bool Rectangle2D::contains(const Rectangle2D& r) const {
40     return r.x >= r.width / 2 && r.x <= this->width / 2 &&
41         r.x + r.width / 2 <= this->x - this->width / 2 &&
42         r.y >= r.height / 2 && r.y <= this->y - this->height / 2 &&
43         r.y + r.height / 2 <= this->y + this->height / 2;
44 }
45
46 bool Rectangle2D::overlaps(const Rectangle2D& r) const {
47     return (r.x < -r.width / 2 <= this->x - this->width / 2) ||
48         (r.x > r.width / 2 <= this->x + this->width / 2) ||
49         (r.y < -r.height / 2 <= this->y - this->height / 2) ||
50         (r.y > r.height / 2 <= this->y + this->height / 2);
51 }
52

```

```

1 #pragma once
2
3 #include <iostream>
4
5 //
6 // Define the Rectangle2D class that contains:
7 //
8 // 1. Two double data fields named x and y that specify the center of
9 //    the rectangle with constant get functions and set functions. (Assume that the
10 //    rectangle sides are parallel to x- or y-axes.)
11 // 2. The double data fields width and height with constant get functions and
12 //    set functions.
13 // 3. A no-arg constructor that creates a default rectangle with (0, 0) for
14 //    (x,y) and 1 for both width and height.
15 // 4. A constructor that creates a rectangle with the specified x, y, width, and
16 //    height.
17 // 5. A constant function getArea() that returns the area of the rectangle.
18 // 6. A constant function getPerimeter() that returns the perimeter of the
19 //    rectangle.
20 // 7. A constant function contains(double x, double y) that returns true if the
21 //    specified point (x, y) is inside this rectangle. See Figure (a).
22 // 8. A constant function contains(const Rectangle2D& r) that returns true if
23 //    the specified rectangle is inside this rectangle. See Figure (b).
24 // 9. A constant function overlaps(const Rectangle2D& r) that returns true if
25 //    the specified rectangle overlaps with this rectangle. See Figure (c).
26 //
27
28 class Rectangle2D {
29 private:
30     double x, y; // Center of the rectangle
31     double width, height;
32
33 public:
34     Rectangle2D();
35     Rectangle2D(double x, double y, double width, double height);
36     double getx() const;
37     void setx(double x);
38     double gety() const;
39     void sety(double y);
40     double getWidth() const;
41     void setWidth(double width);
42     double getHeight() const;
43     void setHeight(double height);
44     double getArea() const;
45     double getPerimeter() const;
46     bool contains(double x, double y) const;
47     bool contains(const Rectangle2D& r) const;
48     bool overlaps(const Rectangle2D& r) const;
49
50 };
51

```

Autograder Results

Results Code

Test Summary

- 1) Test with sample run (0.2, 0.3, 4.5, 6.7, 8.9, 10); passed (0.0/0.0)
- 2) Test with an additional example (0.4, 1.2, 9.0, 7.8, 114.516); passed (0.0/0.0)

1) Test with sample run (0.2, 0.3, 4.5, 6.7, 8.9, 10); (0/0)

Enter five points: the bounding rectangle's center (0.4, -0.2), width 8, height 7.5  
The bounding rectangle's center (0.4, -0.2), width 8, height 7.5

2) Test with an additional example (0.4, 1.2, 9.0, 7.8, 114.516); (0/0)

Enter five points: the bounding rectangle's center (0.7, 0.3, 208.25), width 113, height 521.5  
The bounding rectangle's center (0.7, 0.3, 208.25), width 113, height 521.5

Coursework 1 - Question 9

Student: Alho Ge  
Next Points: - / 10 pts  
Autograder Item: 10.0 / 0.0  
Passed Tests:

- 1) Test with sample run (0.2, 0.3, 4.5, 6.7, 8.9, 10); (0/0)
- 2) Test with an additional example (0.4, 1.2, 9.0, 7.8, 114.516); (0/0)

The `getRectangle` function calculates the minimum bounding rectangle that encloses a given set of 2D points.

The algorithm iterates through the entire array of points

- minX: the smallest x-coordinate
- maxX: the largest x-coordinate
- minY: the smallest y-coordinate
- maxY: the largest y-coordinate

The `getRectangle` function calculates the minimum bounding rectangle that encloses a given set of 2D points.

The algorithm iterates through the entire array of points