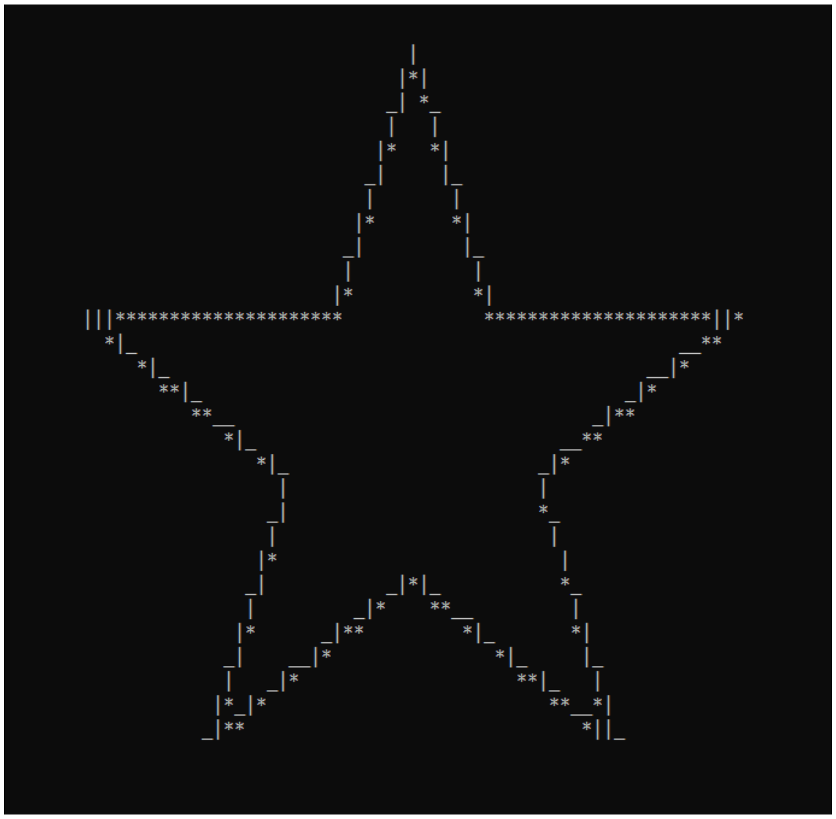
Electrical Engineering Case Study
Robot Kinematics



```
                              |
                             |*|
                            _| *_
                            |   |
                           |*   *|
                          _|     |_
                          |       |
                         |*       *|
                        _|         |_
                        |           |
                       |*           *|
|||*********************             *********************||*
  *|_                                                  __**
     *|_                                            __|*
       **|_                                       _|*
          **__                                 _|**
             *|_                            __**
                *|_                       _|*
                  |                       |
                 _|                       *_
                 |                         |
                |*                          |
               _|           _|*|_           *_
               |         _|*    **__         |
              |*      _|**         *|_       *|
             _|    __|*               *|_     |_
             |   _|*                    **|_   |
            |*_|*                          **__*|
           _|**                               *||_
```

Dhiyaa Al Jorf

da2863

ENGR-UH 1000 Computer Programming for Engineers
Assignment 4

Step 1 – Problem Identification & Statement:

The objective is to develop software to simulate the motion of a multi-degree of freedom kinematic chains that form the structure of a robotic system. The user must be given the freedom to construct as many robots as they want, each with as many segments as they specify, then manipulate them however they like.

Step 2 – Gathering Information & Input/Output Description:

Component Description:

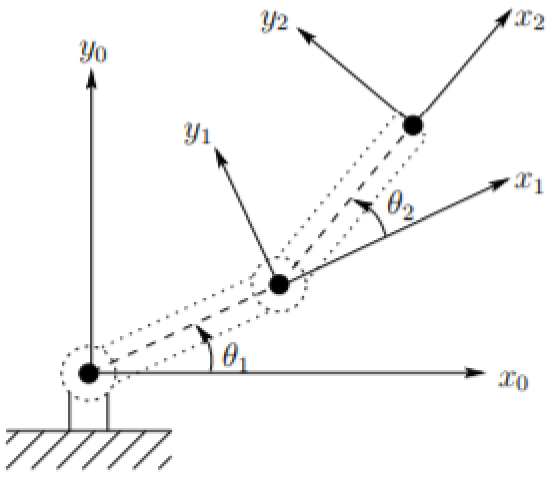


Figure 1. Example of robot chain with 2 segments

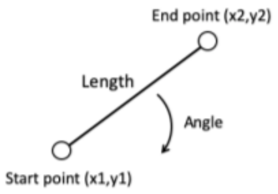


Figure 2. Parameters of a segment

The system has the following components:

1. Points: A point handles x and y coordinate information
2. Segments: A segment is a rigid member connecting two joints. As shown in Figure 2, a segment has the following parameters:
    a. A starting point (x1 and y1 coordinates)
    b. A length
    c. The angle: This angle is a directional angle from the x-axis with the counterclockwise direction as positive.
    d. An end point (x2 and y2 coordinates). The coordinates of the end point can be calculated given the start point coordinates, the length, and the angle.
3. Robots: A robot is comprised of a chain of segments that are connected via joints. The first segment is called the root as it is connected to the robot base/ground. A robot can have multiple segments, which is commonly known as the degrees of freedom. The end effector of the robot is the end point of the last. To determine the

position of the end effector, the position/orientation of all the segments must be determined – which is usually controlled through the angles of the segments.

Thus, the program will include three classes:

1. Point to handle the coordinates of a point in 2D space
    a. The class Point provides storage for a 2D point including the X and Y coordinates. The class should include functions to retrieve/update the coordinates.
2. Segment to store the parameters and associated functions about a specific segment
    a. The class Segment provides storage for the segment parameters, including the start point, the end point, the length (must be greater than 0), and the angle (in radians). The segment class must include a reference to a child segment to facilitate the creation of a chain of segments to form the robot system:
        i. The constructor function initializes the segment start point, length and angle. A local function must be developed (and called through the constructor) to automatically calculate the coordinates of the end point. The end point data must be updated every time any of the segment data are updated.
        ii. The getter/setter functions must be implemented for all the variables.
        iii. There should functions to display the segment information on the output screen.
3. Robot to store the information about the chain of segments making up the robot system.
    a. The class Robot provides storage for the chain of segments, including the root segment and the number of segments in the chain:
        i. The constructor function initializes a robot with no or one root segment. You may use multiple constructors as needed.
        ii. The Robot class must include functions to add and remove segments from the chain. Adding or removing segments is done at the end effector side (insert or delete segments at the end of the chain).
        iii. The Robot class must include a function to reset the chain to a default position. The default position is defined as moving all the segments to the positive x-axis with zero angles.
        iv. The Robot class must include functions to perform forward and inverse kinematics as described in the later two sections.
        v. The Robot class must include a function to print the current pose of the chain. You may use graphics libraries or asterisks to provide a visual representation of the current pose.

Forward Kinematics:

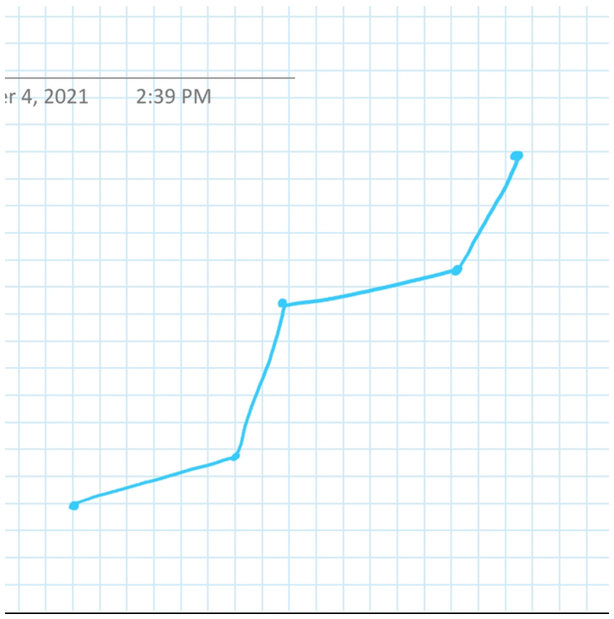


Figure 3. Forward Kinematics. As one segment rotates, all the segments after it rotate by the same amount. Therefore, the input angle by the user is the one between the last segment and the current segment. (The GIF should be playing properly)

The Robot class must include a function to perform the forward kinematics of the chain. The function receives an array of angles corresponding to the segments and returns the end effector position.

1. Note: The angles the user enters to rotate a segment are the directional angles starting at the ray extending from the previous segment to the segment whose angle is in question (As shown in Figure 1). That is because, in reality, the motor at the joint of the robot only knows its angle relative to its previous segment, i.e. whenever a segment rotates, all the ones after it rotate by the same amount as well as shown in Figure 3. These angles will be translated by the forward kinematics function into angles from the x-axis as described in the segment description.
2. To calculate the position of the end effector, the program will progressively calculate the endpoint of each segment until it reaches the last. The equations that will be used to determine the x and y coordinates are:
    a. $x = x_0 + l \times \cos(\theta)$
    b. $y = y_0 + l \times \sin(\theta)$

Inverse Kinematics (Additional Task):

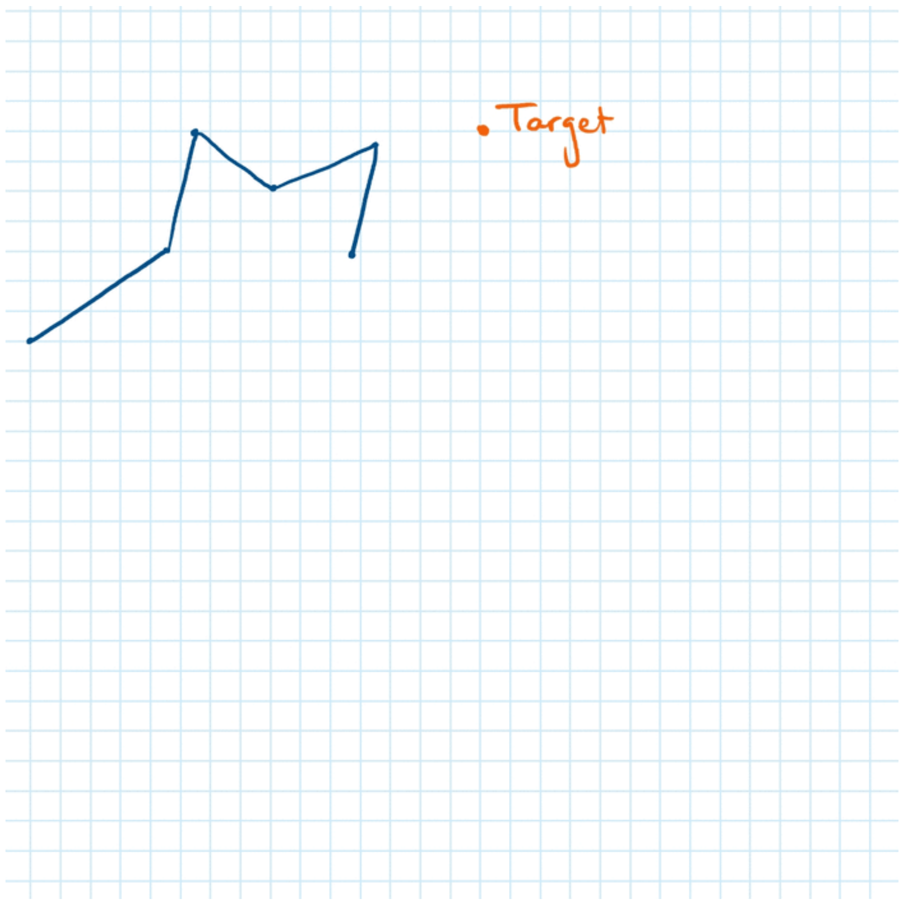


Figure 4. Inverse Kinematics in action (The GIF should be playing properly)

The inverse kinematics involves finding the set of angles to set the end effector to a target position. The method that will be used is described in figure 4 (It's an animated GIF). The method implemented in the program for performing inverse kinematics involves the iterative process of setting a target point, moving a segments endpoint to that target along the line connecting the startpoint and the target, setting the target as the startpoint of the segment that just moves, and repeating. By the end of the process, if the root segment is still in the correct position, the process was a success; otherwise, shift the robot back to its root position and repeat the process. If reaching the point is impossible, the robot will get stuck in a pattern of poses. Therefore, the program must put a cap for how many times this process will be repeated. A reasonable number for the time being is 100.[1][2]

[1] The Coding Train. *Coding Challenge #64.2: Inverse Kinematics*, Youtube, 17 Mar. 2017, https://www.youtube.com/watch?v=hbgDqyy8bIw.

[2] The Coding Train. *Coding Challenge #64.3: Inverse Kinematics - Fixed Point*, YouTube, 18 Mar. 2017, https://www.youtube.com/watch?v=RTc6i-7N3ms.

Displaying the robot:

To display a robot, there should be a class to rasterize a robot's segments and draw them on the output screen using a combination of the characters '*', '|', and '_'. It should use basic trigonometry to calculate which points should be filled and which should be left empty.

User input:

The source file will include an input management system allowing the user to create robots, add and remove segments, move the robot, and display the robot. The system will be as user friendly as possible to facilitate the whole process. It will also include the option for the program to read and display angles in degrees instead of radians since degrees are more intuitive for most users. The code however will still be operating using radians—the code will simply be converting when reading and writing in the degree mode. The menu system is detailed in figure 4.

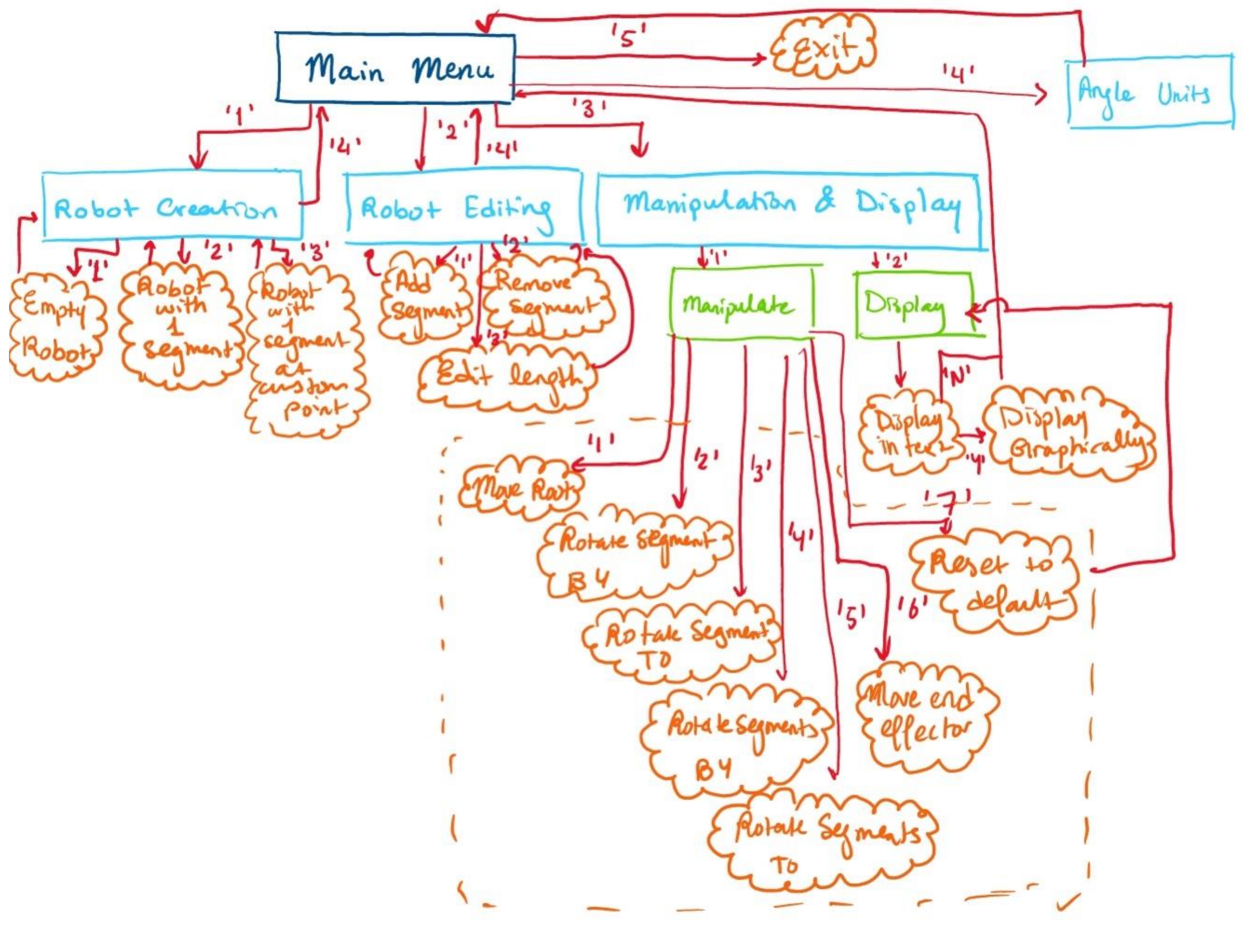
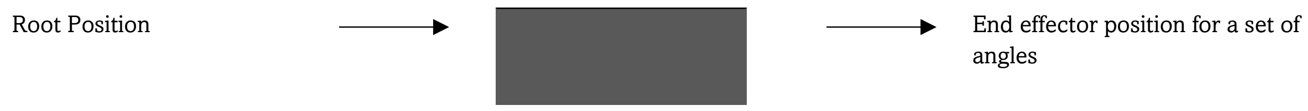
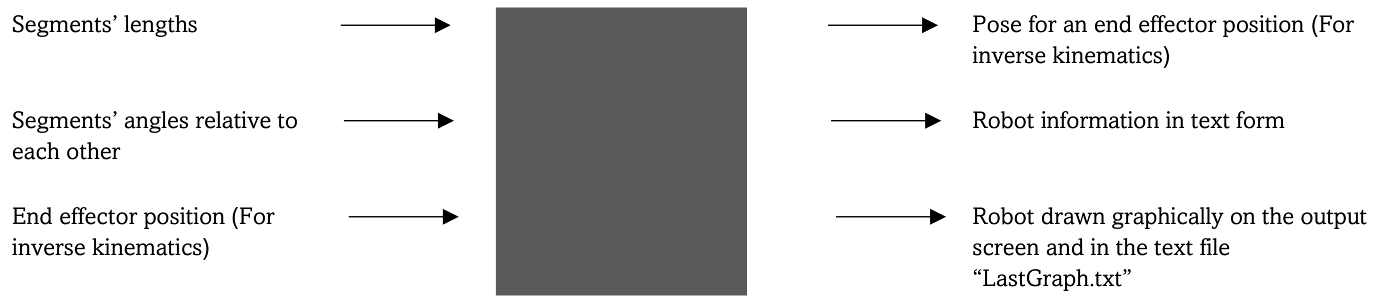


Figure 5. Menu description

I/O Description:

I/O Diagram




Segments’ lengths
Segments’ angles relative to each other
End effector position (For inverse kinematics)
Pose for an end effector position (For inverse kinematics)
Robot information in text form
Robot drawn graphically on the output screen and in the text file “LastGraph.txt”

Step 3 – Test Cases & Algorithms:

Test Cases:

Task 1 – Robot Creation:

Case 1: Creating an empty robot
Case 2: Creating a robot with 1 segment starting at the origin
Case 3: Creating a robot with 1 segment starting at a custom point

Task 2 – Adding, Removing, and Editing Length of Robot Segments:

Case 1: Adding segments to robot with segments
Case 2: Adding a segment to an empty robot
Case 3: Removing a segment from a robot with segments
Case 4: Removing a segment from an empty robot
Case 5: Changing the length of the root segment
Case 6: Changing the length of a segment in the middle of the chain
Case 7: Changing the length of the last segment

Task 3 – Forward Kinematics:

Case 1: Moving the robot's root to a new position
Case 2: Rotating a single segment BY an angle (relative to the previous segment)
Case 3: Rotating a single segment TO an angle (relative to the previous segment)
Case 4: Rotating multiple segments BY angles (relative to the previous segment)
Case 5: Rotating multiple segments TO angle (relative to the previous segment)

Task 4 – Resetting the Robot to its Default Position

Resetting the robot should make all the segments parallel to the x-axis with 0 angles

Task 5 – Inverse Kinematics:

Case 1: Point impossible to reach. Sometimes the point the user wants the end effector to reach is unreachable. In that case, the program should notify the user and return the robot to its initial position

Case 2: Point that is possible to reach. Since a point can be arrived at through multiple paths, there is no expected output per se. As long as the root segment stays in its correct place, all segments maintain their lengths, and the end effector reached its destination, this test case passes

Task 6 – Displaying Robot Information:

Throughout the process of verifying whether the other tasks work, the robot information will be displayed in text and/or graphically.

Bonus Case – Changing Angle Unit Mode:

Throughout the previous tests, the feature to choose what angle units the program should communicate with the user with will be showcased extensively.

Algorithm Design:

```
Point class
    Define DELTA as 0.000001
    Data Members:
        Double x
        Double y
    Function Members:
        Point() default constructor
            Assign 0 to x
            Assign 0 to y
        Point(xPos, yPos) non-default constructor
            Assign xPos to x
            Assign yPos to y
        setX(xPos)
            Assign xPos to x
        setY(yPos)
            Assign yPos to y
        getX()
            return x
        getY()
            return y
        printPointInfo()
            Print "(", x, ",", y, ")"
        distanceFrom(other)
            return $\sqrt{(x - other \to x)^2 + (y - other \to y)^2}$
        isEqualTo(other)
            Assign false to xIsEqual
            Assign false to yIsEqual
            If (|x-other->x| is less than or equal to DELTA)
                Assign true to xIsEqual
            If (|y-other->y| is less than or equal to DELTA)
                Assign true to yIsEqual
            Return yIsEqual AND xIsEqual


Segment class

    Define PI 3.14159265359
    Data Members:
        Point startPoint
        Point endPoint
        Double angle
        Double length
        Reference to a Segment child
    Function Members:
        Segment() default constructor
            Assign Point() to startPoint
            Assign 0 to angle
            Assign 1 to length
            Assign calculateEndPoint(startPoint, angle, length)
            Assign the null pointer to child
        Segment(start, theta, l) default constructor
            Assign start to startPoint
            Assign theta to angle
            Assign 1 to length
            Assign calculateEndPoint(startPoint, angle, length)
            Assign the null pointer to child
        setChild(childReference)
            Assign childReference to child
        setEnd(end)
```

```
            Assign end to endpoint
            Assign calculateStartPoint()
        moveStart(Point start)
            Assign start to startPoint
            Assign calculateEndPoint()
        changeLength(l)
            Assign l to length
            Assign calculateEndPoint()
        setAngle(theta)
            Assign theta to angle
            Assign calculateEndPoint()
        RotateBy(l)
            Increment angle by theta
            Assign calculateEndPoint()
        getChild()
            return child
        getStart()
            return startPoint
        getEnd()
            return endpoint
        getAngle()
            return angle
        getLength()
            return length
        calculateEndPoint()
            return Point(startPoint.getX() + (length*cos(angle)), startPoint.getY() + (length*sin(angle)))
        calculateStartPoint()
            return Point(endPoint.getX() - (length*cos(angle)), endPoint.getY() - (length*sin(angle)))
        printSegmentInfo(mode)
            Print "Start: "
            startPoint.printPointInfo()
            Print "End: "
            endpoint.printPointInfo()
            Print "; Length: ", length, "; Angle above x-axis: "
            If (mode)
                Print angle, " rad"
            Otherwise
                Print (angle * (180/PI)), " deg"
```

Robot class

```
    Define ALLOWED 100
    Data Members:
        Reference to a Segment rootSegment
        Int numSegments
    Function Members:
        Robot() default constructor
            Assign the null pointer to rootSegment
            Assign 0 to numSegments
        Robot(length) default constructor
            Allocate enough memory in the heap for Segment(Point(), 0, length) and assign its address in
            rootSegment
            Assign 1 to numSegments
        Robot(length, root)
            Allocate enough memory in the heap for Segment(root, 0, length) and assign its address in
            rootSegment
            Assign 1 to numSegments
        getRootSegment()
            return rootSegment
        getNumSegments()
            return numSegments
        getSegment(index)
            If (index is greater that or equal numSegments)
                    Print "ERROR: There is no segment of this index", Newline
                    Return the null pointer
            Assign rootSegment to currentSegment
            Assign 0 to i
            Repeat While (i is less than index)
                    Assign currentSegment->getChild() to currentSegment
                    Increment i
            Return currentSegment
        addSegment(length)
            Increment numSegments
            If (rootSegment is equal to the null pointer)
                    Allocate enough memory in the heap for Segment(Point(), 0, length) and assign its
                    address in rootSegment
                    Print "Successfully added a segment", Newline
                    Return
            Assign rootSegment to currentSegment
            Repeat While(currentSegment->getChild() is not equal to the null pointer)
                    Assign currentSegment->getChild() to currentSegment
            currentSegment->setChild(address in the heap of Segment(
            currentSegment->getEnd(), currentSegment->getAngle(), length))
            Print "Successfully added a segment", Newline
        setSegment(length, angle, index)
            If (index is larger than numSegments)
                    Print "There is no segment stored at that index", Newline
                    Return
            Otherwise if (index is equal to numSegments)
                    addSegment(length)
                    return
            Assign the null pointer to previousSegment
            Assign rootSegment to currentSegment
            Assign 1 to i
            Repeat While (i is less than index)
                    Assign currentSegment to previousSegment
                    Assign currentSegment->getChild() to currentSegment
                    Increment i
            Assign currentSegment->getChild() to nextSegment
            Allocate enough memory in the heap for Point(
            currentSegment->getStart().getX(), currentSegment->getStart().getY()) and assign its address
            to oldStart
            Deallocate memory pointed to by currentSegment
            Allocate enough memory in the heap for Segment(content stored at oldPoint, angle, length)
            and assign its address to currentSegment
            If (previousSegment is not equal to the null pointer)
                    previousSegment->setChild(currentSegment)
            If (nexSegment is not equal to the null pointer)
                    currentSegment->setChild(nextSegment)
            Assign 0 to i
            Repeat While (i is less than numSegments)
                    getSegment(i)->moveStart(getSegment(i-1)->getEnd())
                    Increment i
        moveRoot(newPos)
            rootSegment->moveStart(newPos)
            Assign 1 to i
            Repeat While (i is less than numSegments)
                    getSegment(i)->moveStart(getSegment(i-1)->getEnd())
                    Increment i
        removeLastSegment()
            if (numSegments is not equal to 0)
                    Decrement numSegments
                    Assign the null pointer to previousSegment
                    Assign rootSegment to currentSegment
                    Repeat While (currentSegment->getChild() is not equal to the null pointer)
                            Assign currentSegment to previousSegment
                            Assign currentSegment->getChild() to currentSegment
                    If (previousSegment is not equal to the null pointer)
                            previousSegment->setChild(the null pointer)
                    If (numSegments is equal to 0)
                            Assign the null pointer to rootSegment
                    Print "Successfully removed the las segment", Newline
            Otherwise
                    PrintError "ERROR: You are trying to remove a segment from an empty robot . . .",
                    Newline
        resetRobot()
            If (numSegments is equal to -)
                    Print "Empty Robot: No actions performed . . .", Newline
            Otherwise
                    getSegment(0)->RotateBy(-getSegment(0)->getAngle())
                    Assign 1 to i
                    Repeat While (i is less than numSegments)
                            getSegment(i)->moveStart(getSegment(i-1)->getEnd())
                            getSegment(i)->RotateBy(-getSegment(0)->getAngle())
        forwardKinematics(angles, size)
            If (numSegments is less than size)
                    Print "Last ", (size-numSegments), " angles ignored.", Newline
                    Assign numSegments to size
            Assign 0 to i
            Repeat While (i is less than size)
                    getSegment(i)->RotateBy(angles[i])
                    Assign i+1 to j
                    Repeat While (j os less than numSegments)
                            getSegment{j}->moveStart(getSegment(j-1)->getEnd())
                            getSegment(j)->RotateBy(angles[i])
                            Increment j
                    Increment i
            Return getSegment(numSegments - 1)->getEnd()
        makeSegmentFollow(address of segment, address of target)
            Assign arctan(target->getY() – segment->getStart().getY()/target->getX()– segment-
            >getStart().getX()/) (in the correct quadrant based on whether x is positive or negative) to
            theta
            segment->setAngle(theta)
            segment->setEnd(content stored at target)
        inverseKinematics(address of endEffector)
            Assign 0 to maxLength
            Assign 0 to allowedIterations
            Assign 0 to i
            Repeat While (i is less than numSegments)
                Increment maxLength by getSegment(i)->getLength()
                Incement i
            If (Point().distanceFrom(endEffector) > maxLength)
                Return
            rootPosition(rootSegment->getStart().getX(),
            rootSegment->getStart().getY())
            Repeat
                Allocate memory in the heap for Point(endEffector->getX(). endEffector->getY())
                and assign its address to target
                Increment allowedIterations
                moveRoot(rootPosition)
                Assign numSegments – 1 to i
                Repeat While (i is greater than or equal to 0)
                    makeSegmentFollow(getSegment(i), target)
                    Save getSegment(i)->getStart() into the address stored in
                    target
                    Increment i
                Deallocate the memory stored in target
            While (NOT(rootSegment->getStart().isEqualTo(address of rootPosition)) AND
            allowedIterations < ALLOWED)
            moveRoot(rootPosition)
        printRobot(resolution)
            If (numSegments > 0)
                Create a DhiyaaGraphics object artist(resolution)
                artist.addRobotToCanvas(rootSegment, numSegment)
                artist.printToFile()
                artist.clearCanvas()
            Otherwise
                Print "Nothing to draw . . .", Newline, "The robot is empty . . .", Newline
        ~Robot() destructor
            Assign numSegments – 1 to i
            Repeat While (I is greater than or equal to 0)
                Delete getSegment(i)


DhiyaaGraphics class
    Define DEFAULT as 50
    Define RESOLUTION as 100
    Define PI as 3.14159265359
    Data Members:
        Address/Reference to a Segment rootSegment
        Int numSegments
        Int s
        Address storing addresses to Boolean arrays
        Point topCorner
        Point bottomCorner
    Function Members:
        DhiyaaGraphics() default constructor
            Assign DEFAUL * 2 to s
            Create a Boolean pointer array in the heap of size s and save its address in canvas
```

```
                Assign 0 to r
                Repeat While (r less than s)
                        Create a Boolean array in the heap of size s and save its address to canvas[r]
                        Increment r
                Assign Point() to topCorner and bottomCorner
                Assign the null pointer to rootSegment
                Assign 0 to numSegments
                clearCanvas()
        DhiyaaGraphics(width) Non-Default constructor
                Assign DEFAUL * 2 to width
                Create a Boolean pointer array in the heap of size s and save its address in canvas
                Assign 0 to r
                Repeat While (r less than s)
                        Create a Boolean array in the heap of size s and save its address to canvas[r]
                        Increment r
                Assign Point() to topCorner and bottomCorner
                Assign the null pointer to rootSegment
                Assign 0 to numSegments
                clearCanvas()
        getSegment(index)
                If (index is greater that or equal numSegments)
                        Print "ERROR: There is no segment of this index", Newline
                        Return the null pointer
                Assign rootSegment to currentSegment
                Assign 0 to i
                Repeat While (i is less than index)
                        Assign currentSegment->getChild() to currentSegment
                        Increment i
                Return currentSegment
        setUpEdges()
                Assign getSegment(0)->getEnd().getX() to maxAbs
                Assign 0 to i
                Repeat While (i is less than numSegments)
                        If (|getSegment(i)->getEnd().getX()| is greater than maxAbs)
                                Assign (|getSegment(i)->getEnd().getX()| to maxAbs
                        If (|getSegment(i)->getStart().getX()| is greater than maxAbs)
                                Assign (|getSegment(i)->getStart().getX()| to maxAbs
                        If (|getSegment(i)->getEnd().getY()| is greater than maxAbs)
                                Assign (|getSegment(i)->getEnd().getY()| to maxAbs
                        If (|getSegment(i)->getStart().getY()| is greater than maxAbs)
                                Assign (|getSegment(i)->getStart().getY()| to maxAbs
                        Increment i
                Assign maxAbs / 5 to padding
                Increment macAbs by padding
                Assign Point(maxAbs, maxAbs) to topCorner and bottomCorner
addRobotToCanvas(address of a Segment root, size)
        Assign root to rootSegment
        Assign size to numSegments
        setUpEdges
        Assign 0 to i
        Repeat While(i is less than numSegments)
                Assign getSegment(i)->getStart().getY() to yPrev
                Assign the difference between he x values of the start and endpoint of getSegment(i) /
                RESOLUTION to stepX
                If (stepX is equal to 0)
                Assign 0.000000000000000001 to stepX
            If (stepX is greater than 0)
                Assign getSegment(i)->getStart().getX() to j
                Repeat While (j is less than getSegment(i)->getEnd().getX())
                    Assign stepX * tan(getSegment(i)->getAngle()) + yPrev to yCurr
                    Assign yCurr to yPrev
                    Assign true to canvas[(s / 2 – (yCurr/ topCorner.getY()) * (s/2)) cast into
                    an int][( s / 2 – (j/ topCorner.getX()) * (s/2)) cast into an int]
                    Increment j by stepX
            Otherwise
                Assign getSegment(i)->getStart().getX() to j
                Repeat While (j is less than getSegment(i)->getEnd().getX())
                    Assign stepX * tan(getSegment(i)->getAngle()) + yPrev to yCurr
                    Assign yCurr to yPrev
                    Assign true to canvas[(s / 2 – (yCurr/ topCorner.getY()) * (s/2)) cast into
                    an int][( s / 2 – (getSegment(i)->getStart().getX() -  j/ topCorner.getX()) *
                    (s/2)) cast into an int]
                    Decrement j by stepX
clearCanvas()
    Assign 0 to r
    Repeat While (r is less than s)
        Assign 0 to c
        Repeat While (c is less than s)
            Assign false to canvas[r][c]
            Increment c
        Increment r
printToFile()
    Declare and output file stream and open “LastGraph.txt” into it in the output mode
    If (writer fails to open the file)
        Display error message and exit
    Assign topCorner.getY() / (s / 2) to step
    Assign -topCorner.getX() to x
    Repeat While (x is less than or equal to topCorner.getX())
        Print x, Space into writer and the output screen
        Increment x by step * (s/10)
    Print Newline to writer and the output screen
    Assign -topCorner.getX() to x
    Repeat While (x is less than or equal to topCorner.getX())
        Print ‘|’, Space into writer and the output screen
        Increment x by step * (s/10)
    Assign topCorener.getY() into y
    Print Newline to writer and the output screen
    Assign 0 to r
    Repeat While (r is less than s - 1)
        Assign 0 to c
        Repeat While (c is less than s)
            If (canvas[r][c] AND cavas[r+1][c])
                Print ‘|’ to writer and the output screen
            If (canvas[r][c])
                Print ‘*’ to writer and the output screen
            If (canvas[r+1][c])
                Print ‘_’ to writer and the output screen
            Otherwise
                Print ‘ ‘ to writer and the output screen
            Increment c
```

```
                    If (the remainder of the division of (r / 2) by (s /20) is equal to 0)
                              Print "--", y to writer and the output screen
                    Print Newline into writer and the output screen
                    Increment r by 2
          Close writer
~DhiyaaGraphics() Destructor
          Assign 0 to i
          Repeat While (i is less than s)
                    Deallocate the memory storing the array referred to by canvas[i]
                    Increment i
          Deallocate the memory storing the array referred to by canvas
```

Source File

```
Define SMALLSCREEN as 50
Define FULLSCREEN as 90
Define PI as 3.14159265359

options(reference to loop, reference to robots, reference to size, reference to mode)
          Display the main menu options to the user
          Read user input into choice
          If (choice is equal to '1')
                    createNewRobot(robots, size)
          Otherwise if (choice is equal to '2')
                    If (size > 0)
                              editExistingRobot(robots. size)
                    Otherwise
                              Print "Sorry, you haven't created any robots yet.", Newline, "Please choose option 1 before
                              editing your robot.", Newline
          Otherwise if (choice is equal to '3')
                    If (size is greater than 0)
                              moveAndDisplayRobot(robots, size, mode)
                    Otherwise
                              Print "Sorry, you haven't created any robots yet.", Newline, "Please choose option 1 before
                              manipulating/displaying  your robot.", Newline
          Otherwise if (choice is equal to '4')
                    changeAngleUnits(mode)
          Otherwise if (choice is equal to '5')
                    Assign false to loop
          Otherwise
                    Print "Invalid selection . . .", Newline

createNewRobot(Reference to robots, reference to size)
          Assign true to loop
          Print "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~", Newline
          Print "ROBOT CREATION MENU:", Newline
          Repeat While(loop)
                    Print the robot creation menu options to the user
                    Read user input into choice
          If (choice is equal to '1')
                    Increment size
                    Allocate memory in the heap for a Robot pointer array of size size and assign its
                    address to temp
                    Assign 0 to i
                    Repeat While (i is less than size – 1)
```

```
                Assign robots[i] to temp[i]
                Increment i
            Deallocate the array referred to by the address saved in robots
            Assign temp to robots
            Allocate memory in the heap for Robot() and assign its address in robots[size-1]
            Print "Robot ", size, " created.", Newline
        Otherwise If (choice is equal to '2')
            Increment size
            Allocate memory in the heap for a Robot pointer array of size size and assign its
            address to temp
            Assign 0 to i
            Repeat While (i is less than size – 1)
                Assign robots[i] to temp[i]
                Increment i
            Deallocate the array referred to by the address saved in robots
            Assign temp to robots
            Print "Enter the length of the segment: "
            Read user input into length
            Allocate memory in heap for Robot(length) and assign its address in robots[size-1]
            Print "Robot ", size, " with a segment of length ", length, " created", Newline
        Otherwise If (choice is equal to '3')
            Increment size
            Allocate memory in the heap for a Robot pointer array of size size and assign its
            address to temp
            Assign 0 to i
            Repeat While (i is less than size – 1)
                Assign robots[i] to temp[i]
                Increment i
            Deallocate the array referred to by the address saved in robots
            Assign temp to robots
            Print "Enter the length of the segment: "
            Read user input into length
            Print "Enter the x coordinate of the segment: "
            Read user input into x
            Print "Enter the y coordinate of the segment: "
            Read user input into y

            Allocate memory in heap for Robot(length, Point(x, y)) and assign its address in robots[size-1]
            Print "Robot ", size, " with a segment of length ", length, " created at "
            Point(x,y).printPointInfo()
            Print Newline
        Otherwise If (choice is equal to '4')
            Assign false to loop
        Otherwise
            Print "Invalid selection . . .", Newline

changeAngleUnits(reference to mode)
    Print "Current mode:"
    If (mode)
        Print "Radians", Newline
        Print "Would you like to switch to degrees? (Y/N)", Newline
    Otherwise
        Print "Degrees", Newline
        Print "Would you like to switch to radian? (Y/N)", Newline
    Read user input into choice
```

```
        If (the lower case version of choice is equal to ‘y’)
                Assign NOT(mode) to mode
editExistingRobot(Reference to robots, reference to size)
        Assign true to loop
        Print “~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~”, Newline
        Print “ROBOT EDITING MENU: “, Newline
        Print “Which robot would you like to edit? (Max robot ID ” =, size, “)”, Newline
        Read user input into robNumber
        If (robNumber > size)
                Print “Invalid selection . . .”, Newline
        Repeat While (loop)
                Print the robot editing menu options to the user
                Read user input into choice
                If (choice is equal to ‘1’)
                        Print “Enter length of segment to create: ”
                        Read user input into length
                        Robots[robNumber - 1]->addSegment(length)
                Otherwise if (choice is equal to ‘2’)
                        Robots[robNumber - 1]->removeLastSegment()
                Otherwise If (choice is equal to ‘3’)
                        If (robots[robNumber - 1]->getNumSegments() != 0)
                                Repeat While (loop)
                                        Print “Which segment would you like to change the length of? (Max
                                        Segment ID ” robots[robNumber - 1]->getNumSegments(), “)”, Newline
                                        Read user input into segNumber
                                        If (segNumber > robots[robNumber - 1]->getNumSegments())
                                                Print “Invalid selection . . .”, Newline
                                        Otherwise       Assign false to loop
                                Print “Enter the new length of the segment: ”
                                Read user input into length
                                robots[robNumber - 1]->setSegment(length,
                                robots[robNumber - 1]->getSegment(segNumber - 1)->getAngle(), (segNumber - 1))
                                Assign true to loop
                        Otherwise
                                Print “This robot is empty. There are no segments to modify”, Newline
                Otherwise if (choice is equal to ‘4’)
                        Assign false to loop
                Otherwise
                        Print “Invalid selection . . .”, Newline

moveAndDisplayRobot(reference to robots, reference to size, mode)
        Assign true to loop
        Print “~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~”, Newline
        Print “MANIPULATION AND DISPLAY MENU:”, Newline
        Print “Which robot would you like to deal with? (Max Robot ID ”, size, “)”, Newline
        Read user input into robNumber
        If (robNumber > size)
                Print “Invalid selection . . .”, Newline
        If (robots[robNumber - 1]->getNumSegments() is equal to 0)
                Print “This robot is empty, there’s nothing you can manipulate or display in it . . .”, Newline
        Print “What would you like to do with robot ”, robNumber, “?”, Newline
        Print “1 – Manipulate current position”, Newline
        Print “2 – Display robot info”, Newline
        Read user input into choice
        If (choice is not equal to ‘1’ ot ‘2’)
```

```
            Print “Invalid selection . . .”, Newline
            Return
    If (choice is equal to ‘1’)
        Create enough space in the heap for a double array of size robots[robNumber - 1]->getNumSegments()
        and save its address in angles
        Assign 0 to i
        Repeat While (i is less than robots[robNumber - 1]->getNumSegments())
            Assign 0 to angles[i]
        Print “~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~”, Newline
        Print “ROBOT MANIPULATION SUBMENU:”, Newline
        Print all manipulation submenu options to the user
        Read user input into choice
        If (choice is equal to ‘1’)
            Assign 0 to x and y
            Print “Enter x coordinate of new root positon: ”
            Read user input into x
            Print “Enter y coordinate of new root positon: ”
            Read user input into y
            Robots[robNumber - 1]-> moveRoot(Point(x, y))
        Otherwise if (choice is equal to ‘2’)
            Print “Which segment would you like to edit?”, Newline
            Repeat While (loop)
                Print “Which segment would you like to change the length of? (Max Segment ID ”
                robots[robNumber - 1]->getNumSegments(), “)”, Newline
                Read user input into segNumber
                If (segNumber > robots[robNumber - 1]->getNumSegments())
                    Print “Invalid selection . . .”, Newline
Otherwise                   Assign false to loop
            Ask the user for the angle
            Read user input to angles[segNumber - 1]
            Convert it to radians if the current mode is in degrees
            robots[robNumber – 1]->forwardKinematics(angles, segNumber)
        Otherwise if (choice is equal to ‘3’)
            Print “Which segment would you like to edit?”, Newline
            Repeat While (loop)
                Print “Which segment would you like to change the length of? (Max Segment ID ”
                robots[robNumber - 1]->getNumSegments(), “)”, Newline
                Read user input into segNumber
                If (segNumber > robots[robNumber - 1]->getNumSegments())
                    Print “Invalid selection . . .”, Newline
                Otherwise
                    Assign false to loop
            Ask the user for the angle
            Read user input to angles[segNumber - 1]
            Convert it to radians if the current mode is in degrees
            Decrement angles[segNumber - 1] by the current angle of that segment
            If (segnumber – 1 is greater than 0)
                Increment angles[segNumber - 1] by the angle of the previous segment
            robots[robNumber – 1]->forwardKinematics(angles, segNumber)
        Otherwise if (choice is equal to ‘4’)
            Assign 0 to i
            Repeat While (i is less than the number of segments in the robot)
                Print “Enter -999 to stop entering robots”
                Ask the user by what angle to rotate segment i + 1
                Read the user input into angles[i]
                If the user enters -999
                    Assign 0 to angles[i]
                    Break out of the loop
                Convert user input to radians if current mode is in deg
                Increment i
            robots[robNumber – 1]->forwardKinematics(angles, i)
        Otherwise if (choice is equal to '5')
            Assign 0 to i
            Repeat While (i is less than the number of segments in the robot)
                Print "Enter -999 to stop entering robots"
                Ask the user by what angle to rotate segment i + 1
                Read the user input into angles[i]
                If the user enters -999
                    Assign 0 to angles[i]
                    Break out of the loop
                Convert user input to radians if current mode is in deg
                Increment i
            robots[robNumber - 1]->resetRobot()
            robots[robNumber – 1]->forwardKinematics(angles, i)
        Otherwise if (choice is equal to '6')
            Ask the user for the x an y coordinates of the target point and save their input into x and y
            respectively
            Create the Point target(x,y)
            robots[robNumber - 1]->inverseKinematics(address of target)
            If the end effector of the robot reach the desired point
                Print success message
            Otherwise
                Print failure message
        Otherwise if (choice is equal to '7')
            robots[robNumber – 1]->resetRobot()
        Otherwise
            Print "Invalid selection . . .", Newline
            Deallocate the memory for the array referred to by angles
            Return
        Deallocate the memory for the array referred to by angles
        Ask whether the user would like to display the robot info
        If the user says yes
            Assign 2 to choice
    If (choice is equal to '2')
        Print the robot's information with the printRobotInfo() function
        Ask the user whether they would like to print the robot graphically
        If they say yes
            Ask whether they want it in full screen or not and print accordingly using the printRobot
            function

Main()
    Assign true to mode
    Asign 0 to size
    Allocate memory in the heap for a Robot pointer array of size size and assign its address to
    Robots
    Assign true to loop
    Print "Welcome to the robot factory . . .", Newline
    Repeat
        options(loop, robots, size, mode)
    While (loop)
```

```
Print “~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~”, Newline
Print “Thanks for using our services . . . Goodbye . . . ”, Newline
Assign 0 to i
Repeat While (i is less than size)
        Deallocate memory stored in the address robots[i]
        Increment i
Deallocate the memory used for the array starting at address robots
```

Step 4 – C++ Code:

Source code:
```
//***********************************************************//
//** Name: Dhiyaa Al Jorf                                  **//
//** Net ID: da2863                                        **//
//** Date Created: November 27, 2021                       **//
//** Assignment 4: Electrical Engineering Case Study:      **//
//** Robot Kinematics                                      **//
//** Program: Assignment3.cpp                              **//
//***********************************************************//

#include <cctype> // Used to allow user to enter both y or Y to say yes
// Including all the custom classes
#include "Robot.h"
#define SMALLSCREEN 50 // A good resolution for printing on the default size of the output window
#define FULLSCREEN 90 // A good resolution for printing on the maximized output window
#define PI 3.14159265359 // Value of pi to 11 decimal places. Used to convert from degrees to radians and
vice verca

using namespace std;

void options(bool& loop, Robot**& robots, int& size, bool& mode);
void createNewRobot(Robot**& robots, int& size);
void editExistingRobot(Robot**& robots, int& size);
void moveAndDisplayRobot(Robot**& robots, int& size, bool mode);
void changeAngleUnits(bool& mode);

int main() {
        bool mode = 1; // Default angle unit mode is 1 = radians
        int size = 0; // Represents the number of robots created
        Robot** robots = new Robot * [size]; // Dynamic array of robot pointers
        bool loop = true; // loop condition. true until user decides to exit program
        cout << "Welcome to the robot factory . . ." << endl;
        do {
                options(loop, robots, size, mode);
        } while (loop);
        cout <<
"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" <<
endl;
        cout << "Thanks for using our services . . . Goodbye . . ." << endl;
        // Deallocating all the memory allocated for the robots
        for (int i = 0; i < size; i++) {
                delete robots[i];
        }
        delete[] robots;
}

// MAIM MENU: This function displays the main menu and hadnles to which menu the user should go according
to their choice
void options(bool& loop, Robot**& robots, int& size, bool& mode) {
        char choice;
        cout <<
"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" <<
endl;
        cout << "MAIN MENU:" << endl;
        cout << "Enter number to select option:" << endl;
        cout << "1 - Create new robot" << endl;
        cout << "2 - Edit existing robot (Add, remove, or change length of segments)" << endl;
        cout << "3 - Manipulate positions and angles and/or display robot information" << endl;
        // Angle units mode toggler
        cout << "4 - Change angle units. Current mode: ";
        if (mode) {
                cout << "Radians" << endl;
        }
        else {
                cout << "Degrees" << endl;
        }
```

```
        cout << "5 - Exit program" << endl; // Set the condition for the do while in the main function to
false such that execution ends
        cin >> choice;
        // Switch case block to handle user input
        switch (choice) {
        case '1':
                createNewRobot(robots, size);
                break;
        // Option 2 and 3 are only available after at least one robot has been created
        case '2':
                if (size > 0)
                        editExistingRobot(robots, size);
                else {
                        cout << "Sorry, you haven't created any robots yet.\nPlease choose option 1
before editing your robot." << endl;
                }
                break;
        case '3':
                if (size > 0) {
                        moveAndDisplayRobot(robots, size, mode);
                }
                else {
                        cout << "Sorry, you haven't created any robots yet.\nPlease choose option 1
before manipulating/displaying your robot." << endl;
                }
                break;
        case '4':
                changeAngleUnits(mode);
                break;
        case '5':
                loop = false;
                break;
        default:
                cout << "Invalid selection . . ." << endl;
                break;
        }
}

// ROBOT CREATION MENU: This function allows the user to create custom robots using the constructors of
the Robot class
void createNewRobot(Robot**& robots, int& size) {
        bool loop = true;
        char choice;
        double length, x, y;
        cout <<
"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" <<
endl;
        cout << "ROBOT CREATION MENU:" << endl;
        while (loop) {
                cout << "Enter selection: " << endl;
                cout << "1 - Empty robot starting at the origin" << endl; // Default robot constructor
                cout << "2 - Create robot starting at the origin with 1 segment" << endl; // Non-default
robot constructor that asks for length only
                cout << "3 - Create robot at custom point with 1 segment" << endl; // Non-default robot
constructor that asks for length and root position
                cout << "4 - Back to main menu" << endl; // Back to main menu
                cin >> choice;
                // Switch case block to handle user input
                switch (choice) {
                case '1': {
                        // Expanding the robots array to have space for the robot we want to create
                        size++;
                        Robot** temp = new Robot * [size];
                        for (int i = 0; i < size - 1; i++) {
                                temp[i] = robots[i];
                        }
                        delete[] robots;
                        robots = temp;
                        robots[size - 1] = new Robot(); // Create the robot with the default constructor
```

```
                                cout << "Robot " << size << " created." << endl;
                                break;
                        }
                        case '2': {
                                // Expanding the robots array to have space for the robot we want to create
                                size++;
                                Robot** temp = new Robot * [size];
                                for (int i = 0; i < size - 1; i++) {
                                        temp[i] = robots[i];
                                }
                                delete[] robots;
                                robots = temp;
                                cout << "Enter length of segment: ";
                                cin >> length;
                                robots[size - 1] = new Robot(length); // Use the non-default constructor that
takes in the length only to create the new robot
                                cout << "Robot " << size << " with a segment of length " << length << " created"
<< endl;
                                break;
                        }
                        case '3': {
                                // Expanding the robots array to have space for the robot we want to create
                                size++;
                                Robot** temp = new Robot * [size];
                                for (int i = 0; i < size - 1; i++) {
                                        temp[i] = robots[i];
                                }
                                delete[] robots;
                                robots = temp;
                                cout << "Enter length of segment: ";
                                cin >> length;
                                cout << "Enter x position of root: ";
                                cin >> x;
                                cout << "Enter y position of root: ";
                                cin >> y;
                                robots[size - 1] = new Robot(length, Point(x, y)); // Use the non-default
constructor that takes in the length and starting position to create the new robot
                                cout << "Robot " << size << " with a segment of length " << length << " created
at ";
                                Point(x, y).printPointInfo();
                                cout << endl;
                                break;
                        }
                        case '4':
                                loop = false; // To exit this submenu
                                break;
                        default:
                                cout << "Invalid selection . . ." << endl;
                                break;
                        }
                }
}


// This function allows the user between angle units to give the user the flexibility in data input
void changeAngleUnits(bool& mode) {
        char choice;
        cout << "Current mode: ";
        if (mode) {
                cout << "Radians" << endl;
                cout << "Would you like to switch to degrees? (Y/N)" << endl;
        }
        else {
                cout << "Degrees" << endl;
                cout << "Would you like to switch to radians? (Y/N)" << endl;
        }
        cin >> choice;
        if (tolower(choice) == 'y')
                mode = !mode;
```

```
}

// ROBOT EDITING MENU: This function allows the user to add and remove segments from their robots
void editExistingRobot(Robot**& robots, int& size) {
	bool loop = true;
	int robNumber, segNumber;
	char choice;
	double length;
	cout <<
"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" <<
endl;
	cout << "ROBOT EDITING MENU:" << endl;
	// Displays the available robots
	cout << "Which robot would you like to edit? (Max robot ID " << size << "): " << endl;
	cin >> robNumber;
	if (robNumber > size) {
		cout << "Invalid selection . . ." << endl;
	}
	while (loop) {
		cout << "What would you like to do?" << endl;
		cout << "1 - Add segment to the end" << endl; // addSegment() function
		if (robots[robNumber - 1]->getNumSegments() == 0) {
			cout << "(This robot is empty. Adding a segment to it automatically adds it at
the origin)" << endl; // Notifies the user that an empty robot always sets the start point at the origin
		}
		cout << "2 - Remove segment from the end" << endl; // removeLastSegment() function
		cout << "3 - Edit segment length" << endl; // Makes use of the setSegment() function to
edit the length of a particular segment
		cout << "4 - Back to main menu" << endl;
		cin >> choice;
		// Switch-case block to handle user input
		switch (choice) {
		case '1':
			cout << "Enter length of segment to create: ";
			cin >> length;
			robots[robNumber - 1]->addSegment(length);
			break;
		case '2':
			robots[robNumber - 1]->removeLastSegment();
			break;
		case '3':
			if (robots[robNumber - 1]->getNumSegments() != 0) {
				while (loop) {
					cout << "Which segment would you like to change the length of?
(Max Segment ID " << robots[robNumber - 1]->getNumSegments() << ")" << endl;
					cin >> segNumber;
					if (segNumber > robots[robNumber - 1]->getNumSegments()) {
						cout << "Invalid selection" << endl;
					}
					else {
						loop = false;
					}
				}
				cout << "Enter the new length of the segment: ";
				cin >> length;
				robots[robNumber - 1]->setSegment(length, robots[robNumber - 1]-
>getSegment(segNumber - 1)->getAngle(), (segNumber - 1));
				loop = true;
			}
			else {
				cout << "This robot is empty. There are no segments to modify" << endl;
			}
			break;
		case '4':
			loop = false; // Leave the submenu
			break;
		default:
			cout << "Invalid selection . . ." << endl;
			break;
```

```
                }
        }
}


// MANIPULATION AND DISPLAY MENU: This function allows the user to perform forward kinematics on the
robot, move the robot's root position, perform inverse kinematics, display the robot's information, and/or
graphically display the robot
void moveAndDisplayRobot(Robot**& robots, int& size, bool mode) {
        bool loop = true;
        int robNumber, segNumber;
        char choice;
        cout <<
"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" <<
endl;
        cout << "MANIPULATION AND DISPLAY MENU:" << endl;
        cout << "Which robot would you like to work with? (Max robot ID " << size << "): " << endl;
        cin >> robNumber;
        if (robNumber > size) {
                cout << "Invalid selection . . ." << endl;
                return;
        }
        if (robots[robNumber - 1]->getNumSegments() == 0) {
                cout << "This robot is empty, there's nothing you can manipulate or display in it . . ."
<< endl;
                return;
        }
        cout << "What would you like to do with robot " << robNumber << "?" << endl;
        cout << "1 - Manipulate current postition" << endl;
        cout << "2 - Display robot info" << endl;
        cin >> choice;
        if (choice != '1' && choice != '2') {
                cout << "Invalid selection . . ." << endl;
                return;
        }
        if (choice == '1') {
                double* angles = new double[robots[robNumber - 1]->getNumSegments()];
                for (int i = 0; i < robots[robNumber - 1]->getNumSegments(); i++) {
                        angles[i] = 0;
                }
                cout << "~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~" << endl;
                cout << "ROBOT MANIPULATION SUB-MENU:" << endl;
                cout << "What would you like to do?" << endl;
                cout << "1 - Move root segment" << endl; // Uses the moveRoot() function
                cout << "2 - Rotate single segment BY an angle (angles measured from previous segment)" <<
endl; // Makes use of the forwardKinematics function
                cout << "3 - Rotate single segment TO an angle (angles measured from previous segment)" <<
endl; // Makes use of the forwardKinematics function
                cout << "4 - Rotate multiple segments BY set of angles (angles measured from previous
segment)" << endl; // Makes use of the forwardKinematics function
                cout << "5 - Rotate multiple segments TO set of angles (angles measured from previous
segment)" << endl; // Makes use of the forwardKinematics function
                cout << "6 - Move end effector to a point" << endl; // Uses the inverseKinematics()
function
                cout << "7 - Reset robot to default position" << endl; // Uses the resetPosition()
function
                cin >> choice;
                // Switch-case block to handle user selection
                switch (choice) {
                case '1': {
                        double x = 0, y = 0;
                        cout << "Enter x coordinate of new root position: ";
                        cin >> x;
                        cout << "Enter y coordinate of new root position: ";
                        cin >> y;
                        robots[robNumber - 1]->moveRoot(Point(x, y));
                        break;
                }
```

```
                case '2':
                        while (loop) {
                                cout << "Which segment would you like to edit? (Max Segment ID " <<
robots[robNumber - 1]->getNumSegments() << ")" << endl;
                                cin >> segNumber;
                                if (segNumber > robots[robNumber - 1]->getNumSegments()) {
                                        cout << "Invalid selection";
                                }
                                else {
                                        loop = false;
                                }
                        }
                        cout << "Enter angle to rotate the segment BY - Current Mode (";
                        if (mode) {
                                cout << "Radians):" << endl;
                        }
                        else {
                                cout << "Degrees):" << endl;
                        }
                        cin >> angles[segNumber - 1];
                        if (!mode) {
                                angles[segNumber - 1] *= (PI / 180); // Converts user input to radians if
the program is in degree mode
                        }
                        robots[robNumber - 1]->forwardsKinematics(angles, segNumber); // No need to alter
the angles array as the forward kinematics already rotates BY an angle
                        break;
                case '3':
                        while (loop) {
                                cout << "Which segment would you like to edit? (Max Segment ID " <<
robots[robNumber - 1]->getNumSegments() << ")" << endl;
                                cin >> segNumber;
                                if (segNumber > robots[robNumber - 1]->getNumSegments()) {
                                        cout << "Invalid selection";
                                }
                                else {
                                        loop = false;
                                }
                        }
                        cout << "Enter angle to rotate the segment TO: - Current Mode (";
                        if (mode) {
                                cout << "Radians):" << endl;
                        }
                        else {
                                cout << "Degrees):" << endl;
                        }
                        cin >> angles[segNumber - 1];
                        if (!mode) {
                                angles[segNumber - 1] *= (PI / 180); // Converts user input to radians if
the program is in degree mode
                        }
                        angles[segNumber - 1] -= robots[robNumber - 1]->getSegment(segNumber - 1)-
>getAngle(); // Calculates by how much each segment has to rotate to reach the specified angle
                        if (segNumber - 1 > 0) {
                                angles[segNumber - 1] += robots[robNumber - 1]->getSegment(segNumber -
2)->getAngle();
                        }
                        robots[robNumber - 1]->forwardsKinematics(angles, segNumber); // After altering
the angles array, the robot will now move TO the specified angles
                        break;
                case '4': {
                        int i = 0;
                        for (i = 0; i < robots[robNumber - 1]->getNumSegments(); i++) {
                                cout << "Enter -999 to stop entering angles (remaining angles will be
left unchanged)" << endl;
                                cout << "Rotate segment " << (i + 1) << " BY - Current Mode (";
                                if (mode) {
                                        cout << "Radians):" << endl;
                                }
```

```
					else {
						cout << "Degrees):" << endl;
					}
					cin >> angles[i];
					if (angles[i] == -999) {
						angles[i] = 0;
						break;
					}
					if (!mode) {
						angles[i] *= (PI / 180); // Converts user input to radians if the 
program is in degree mode
					}
				}
				robots[robNumber - 1]->forwardsKinematics(angles, i); // No need to alter the 
angles array as the forward kinematics already rotates BY an angle
				break;
			}
			case '5': {
				int i = 0;
				for (i = 0; i < robots[robNumber - 1]->getNumSegments(); i++) {
					cout << "Enter -999 to stop entering angles (remaining angles will be 
left unchanged)" << endl;
					cout << "Rotate segment " << (i + 1) << " TO - Current Mode (";
					if (mode) {
						cout << "Radians):" << endl;
					}
					else {
						cout << "Degrees):" << endl;
					}
					cin >> angles[i];
					if (angles[i] == -999) {
						angles[i] = 0;
						break;
					}
					if (!mode) {
						angles[i] *= (PI / 180); // Converts user input to radians if the 
program is in degree mode
					}
				}
				robots[robNumber - 1]->resetRobot(); // Reseting the robot makes any rotations BY 
the same as rotations TO
				robots[robNumber - 1]->forwardsKinematics(angles, i);
				break;
			}
			// Performs inverse kinematics
			case '6':
			{
				double x, y;
				cout << "Enter x coordinate of desired point: " << endl;
				cin >> x;
				cout << "Enter y coordinate of desired point: " << endl;
				cin >> y;
				Point target(x, y);
				robots[robNumber - 1]->inverseKinematics(&target, 0);
				bool success = robots[robNumber - 1]->getSegment(robots[robNumber - 1]-
>getNumSegments() - 1)->getEnd().isEqualTo(&target);
				// If the robot succeeds at reaching the specified point, print confirmation 
message. Otherwise, reset to default and notify the user that it's impossible
				if (success) {
					cout << "Succeeded at reaching the point" << endl;
				}
				else {
					robots[robNumber - 1]->resetRobot();
					cout << "Failed to reach the point . . . \nRobot returned to default 
position. . ." << endl;
				}
				break;
			}
			case '7':
				robots[robNumber - 1]->resetRobot();
				break;
			default:
				delete[] angles; // Deallocating memory used for the angles array
				cout << "Invalid selection . . ." << endl;
				return;
				break;
			}
			delete[] angles; // Deallocating memory used for the angles array
			cout << "Would you like to print robot info? (Y/N)" << endl;
			cin >> choice;
			if (tolower(choice) == 'y') {
				choice = '2';
			}
		}
		// Handles displaying the robot info both in text and graphically
		if (choice == '2') {
			cout << "~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~~" << endl;
			cout << "ROBOT DISPLAY SUB-MENU:" << endl;
			cout << "ROBOT " << robNumber << ":" << endl;
			robots[robNumber - 1]->printRobotInfo(mode);
			cout << "Would you like to display robot " << robNumber << "'s information graphically as 
well? (Y/N)" << endl;
			cin >> choice;
			if (tolower(choice) == 'y') {
				cout << "Would you like to display it in full screen mode? (Y/N)" << endl;
				cin >> choice;
				if (tolower(choice) == 'y')
					robots[robNumber - 1]->printRobot(FULLSCREEN);
				else
					robots[robNumber - 1]->printRobot(SMALLSCREEN);
			}
			return;
		}
}
```

Point Class:

```
#pragma once
#include <iostream>
#include <iomanip>
#include <cmath>
#define DELTA 0.000001

using namespace std;

// The Point class handles coordinate data and is used to create the start and endpoints of each segment
class Point {
private:
	double x;
	double y;
public:
	// Default constructor: Any non-initialized point is placed at the origin
	Point() {
		x = 0;
		y = 0;
	}
	// Non-default constructor: Creates a point at the specified coordinates
	Point(double xPos, double yPos) {
		x = xPos;
		y = yPos;
	}
	// Setter functions:
	void setX(double xPos) {
		this->x = xPos;
	}
	void setY(double yPos) {
		this->y = yPos;
	}
	// Getter functions:
	double getX() {
		return this->x;
	}
	double getY() {
		return this->y;
	}
	// The printPointInfo funtion prints the x and y coordinates in the format (x,y)
	void printPointInfo() {
		cout << '(' << setprecision(3) << fixed << x << ',' << setprecision(3) << fixed << y <<
')';
	}
	// Finds the euclidean distance between two pointsusing the pythagorean theorem
	double distanceFrom(Point* other) {
		return sqrt(pow(x - other->x, 2) + pow(y - other->y, 2));
	}
	// Checks if two points are the same within some margin of error due to the uncertainty of the
point data
	bool isEqualTo(Point* other) {
		bool xIsEqual = false;
		bool yIsEqual = false;
		if (fabs(x - other->x) <= DELTA) {
			xIsEqual = true;
		}
		if (fabs(y - other->y) <= DELTA) {
			yIsEqual = true;
		}
		return yIsEqual && xIsEqual;
	}
};
```

Segment Class:

```
#pragma once
// Including custom classes
#include "Point.h"
#define PI 3.14159265359 // Value of pi to 11 decimal places. Used to convert from degrees to radians and
vice verca

using namespace std;

// The Segment class handles segment information where each segment is desribed by a start point, a
length, an angle above the x-axis, and an endpoint
// Segments from the Segment class will be used as building blocks to assemenble robots
class Segment {
private:
	Point startPoint;
	Point endPoint;
	double angle;
	double length;
	// Reference to a child segment. If there is no child, it is set nullptr.
	Segment* child;
public:
	// Default Constructor: assigns the safe default values for the segment's instance variables
	Segment() {
		startPoint = Point();
		angle = 0;
		length = 1;
		endPoint = calculateEndPoint(startPoint, angle, length);
		child = nullptr;
	}
	// Non-Default Constructor: Creates a segment using the provided values for the start point,
length, and angle
	Segment(Point start, double theta, double l) {
		startPoint = start;
		angle = theta;
		// Length can't be less than or equal to 0; therefore, the program sets the value to the
safe default value of 1
		if (l <= 0) {
			cerr << "ERROR: Length of segment can't be non-positive. Segment length set to
default of 1" << endl;
```

```
				l = 1;
			}
			length = l;
			endPoint = calculateEndPoint(start, theta, l); // The user isn't given the option to set 
the end point since it is calculated using the other variables
			child = nullptr;
		}

		// Modified Setter Functions:
		void setChild(Segment* childReference) {
			child = childReference;
		}

		void setEnd(Point end) {
			this->endPoint = end;
			this->startPoint = calculateStartPoint(this->endPoint, this->angle, this->length); // Any 
time the start point is changed, the endpoint must be updated accordingly
		}

		void moveStart(Point start) {
			this->startPoint = start;
			this->endPoint = calculateEndPoint(this->startPoint, this->angle, this->length); // Any 
time the start point is changed, the endpoint must be updated accordingly
		}

		void changeLength(double l) {
			this->length = l;
			this->endPoint = calculateEndPoint(this->startPoint, this->angle, this->length); // Any 
time the length is changed, the endpoint must be updated accordingly
		}

		void setAngle(double theta) {
			this->angle = theta;
			this->endPoint = calculateEndPoint(this->startPoint, this->angle, this->length); // Any 
time the angle is changed, the endpoint must be updated accordingly
		}

		void RotateBy(double theta) {
			this->angle += theta;
			this->endPoint = calculateEndPoint(this->startPoint, this->angle, this->length); // Any 
time the angle is changed, the endpoint must be updated accordingly
		}

		// Getter Functions:
		Segment* getChild() {
			return child;
		}

		Point getStart() {
			return this->startPoint;
		}

		Point getEnd() {
			return this->endPoint;
		}

		double getAngle() {
			return this->angle;
		}

		double getLength() {
			return this->length;
		}

		// The calculateEndPoint function uses basic trigonometry to find the x and y coordinates of the 
endpoint
		Point calculateEndPoint(Point start, double theta, double l) {
			return Point(start.getX() + (l * cos(theta)), start.getY() + (l * sin(theta)));
		}
```

```
        // The calculateStartPoint function uses basic trigonometry to find the x and y coordinates of the
startpoint
        Point calculateStartPoint(Point end, double theta, double l) {
                return Point(end.getX() - (l * cos(theta)), end.getY() - (l * sin(theta)));
        }

        // The printSegmentInfo function prints the data related to segment. It makes use of the
printPointInfo function when displaying the start and end points of the segment.
        // The current units mode is also taken into account when displaying angles
        void printSegmentInfo(bool mode) {
                cout << "Start: ";
                startPoint.printPointInfo();
                cout << "; End: ";
                endPoint.printPointInfo();
                cout << "; Length: " << this->length << "; Angle above x-axis: ";
                // The below code displays the angle in the current angle mode
                if (mode) {
                        cout << this->angle << " rad";
                }
                else {
                        cout << this->angle * (180 / PI) << " deg";
                }
        }
};
```

Robot Class:

```
#pragma once
// Including custom classes
#include "DhiyaaGraphics.h"
#define DELTA 0.0001
#define ALLOWED 100

using namespace std;

// The Robot class links segments together and handles their interactions
class Robot {
private:
        Segment* rootSegment; // Pointer to the root segment of the robot
        int numSegments; // Used to keep track of the number of segments in a given robot
public:
        // Default constructor: Creates an empty robot at the origin
        Robot() {
                rootSegment = nullptr;
                numSegments = 0;
        }

        // Non-Default constructor: Creates a robot with a segment of specified length at the origin
        Robot(double length) {
                rootSegment = new Segment(Point(0,0), 0, length);
                numSegments = 1;
        }

        // Non-Default constructor: Creates a robot with a segment of specified length at the specified
point
        Robot(double length, Point root) {
                rootSegment = new Segment(root, 0, length);
                numSegments = 1;
        }

        // Getter Functions:
        Segment* getRootSegment() {
                return rootSegment;
        }

        int getNumSegments() {
                return numSegments;
        }

        // Functions to go through the linked list of segments
        Segment* getSegment(int index) {
```

```
		if (index >= numSegments) {
			cout << "ERROR: there is no segment of this index" << endl;
			return nullptr;
		}
		Segment* currentSegment = rootSegment;
		for (int i = 0; i < index; i++) {
			currentSegment = currentSegment->getChild();
		}
		return currentSegment;
	}

	// Function that handles the addition of segments at the end of the robot
	void addSegment(int length) {
		numSegments++;
		// If the robot is empty, create the root segment and set its adress to rootSegment then 
exit the function
		if (rootSegment == nullptr) {
			rootSegment = new Segment(Point(0, 0), 0, length);
			cout << "Successfully added a segment" << endl;
			return;
		}
		// Go to the end of the linked list (last segment), then create a child for the last 
segment using the user specified length
		Segment* currentSegment = rootSegment;
		while (currentSegment->getChild() != nullptr) {
			currentSegment = currentSegment->getChild();
		}
		currentSegment->setChild(new Segment(currentSegment->getEnd(), currentSegment->getAngle(), 
length));
		cout << "Successfully added a segment" << endl; // Confirmation message
	}

	// Useful for changing the length of a segment anywhere in the chain
	void setSegment(double length, double angle, int index) {
		// You can't set a segment that doesn't exist
		if (index > numSegments) {
			cout << "There is no segment stored at that index" << endl;
			return;
		}
		// If the index is numSegments, create a new segment at the end;
		else if (index == numSegments) {
			addSegment(length);
			return;
		}
		Segment* previousSegment = nullptr; // Used to edit the child of the previous segment
		Segment* currentSegment = rootSegment; // Used to edit the segment itself
		for (int i = 0; i < index; i++) {
			previousSegment = currentSegment; // Previous segment always lags 1 segment 
behind current segment
			currentSegment = currentSegment->getChild();
		}
		// Temporarilt holding all the old data of a segment that is relevant when it is deleted
		Segment* nextSegment = currentSegment->getChild();
		Point* oldStart = new Point(currentSegment->getStart().getX(), currentSegment-
>getStart().getY());
		// deleting the segment
		delete currentSegment;
		// Creating the robot
		currentSegment = new Segment(*oldStart, angle, length);
		// Linking the robot to the rest of the linked list
		if (previousSegment != nullptr) // This condition would be false if index == 0
			previousSegment->setChild(currentSegment);
		if (nextSegment != nullptr)
			currentSegment->setChild(nextSegment);
		// Correct for all the next segments
		for (int i = index + 1; i < numSegments; i++) {
		getSegment(i)->moveStart(getSegment(i - 1)->getEnd());
		}
	}
	// The moveRoot function moves the starting position of the robot to the specified new position 
while making sure the segments stay connected
	void moveRoot(Point newPos) {
		rootSegment->moveStart(newPos);
		// Propagating the change throughout the chain to maintain connectedness
		for (int i = 1; i < numSegments; i++) {
			getSegment(i)->moveStart(getSegment(i-1)->getEnd());
		}
	}

	// The following function removes a segment at the end effector if the robot is not empty
	void removeLastSegment() {
		if (numSegments != 0) {
			numSegments--;
			Segment* previousSegment= nullptr; // Keeping track of the previous segment
			Segment* currentSegment = rootSegment;
			while (currentSegment->getChild() != nullptr) {
				previousSegment = currentSegment; // previousSegment always lags behind 
currentSegment by 1
				currentSegment = currentSegment->getChild();
			}
			if (previousSegment != nullptr)
				previousSegment->setChild(nullptr);
			delete currentSegment;
			cout << "Successfully removed the last segment" << endl;
		}
		else {
			cerr << "ERROR: You are trying to remove a segment from an empty robot . . ." << 
endl;
		}
	}
	// The reset robot rotates everything back to 0 angles relative to the x-axis
	void resetRobot() {
		if (numSegments == 0) {
			cout << "Empty Robot: No actions performed . . ." << endl;
		}
		else {
			// Rotates by the opposite of its current angle to reset to 0
			getSegment(0)->RotateBy(-getSegment(0)->getAngle());
			for (int i = 1; i < numSegments; i++) {
				getSegment(i)->moveStart(getSegment(i - 1)->getEnd()); // Make sure the 
start of one segment is the endpoint of the first
				getSegment(i)->RotateBy(-getSegment(i)->getAngle());
			}
		}
	}
	// The forwardsKinematics function rotates the segments of a robot by a set of angles relative to 
the previous segments and returns the position of the end effector
	Point forwardsKinematics(double angles[], int size) {
		// To handle the cases where there are more angles than segments
		if (numSegments < size) {
			cout << "Last " << (size - numSegments) << " angles ignored." << endl;
			size = numSegments;
		}
		// Rotates each segment and the ones after it by a given angle. That is because the angles 
given to this function are relative the segment before it
		for (int i = 0; i < size; i++) {
			getSegment(i)->RotateBy(angles[i]);
			for (int j = i + 1; j < numSegments; j++) {
				getSegment(j)->moveStart(getSegment(j - 1)->getEnd()); // Make sure the 
start of one segment is the endpoint of the first
				getSegment(j)->RotateBy(angles[i]);
			}
		}
		// Returns the end point of the last segment
		return getSegment(numSegments - 1)->getEnd();
	}
	// Used in the inverseKinematics function. Moves a segment to a certain target
	void makeSegmentFollow(Segment* segment, Point* target) {
		double theta = atan2(target->getY() - segment->getStart().getY(), target->getX() -
segment->getStart().getX());
		segment->setAngle(theta);
		segment->setEnd(*target);
	}

	// Inverse kinematics function to move the end effector to a target point if possible
	void inverseKinematics(Point* endEffector, int allowedIterations) {
		double maxLength = 0;
		for (int i = 0; i < numSegments; i++) {
			maxLength += getSegment(i)->getLength();
		}
		if (Point().distanceFrom(endEffector) > maxLength) {
			return;
		}
		Point rootPosition(rootSegment->getStart().getX(), rootSegment->getStart().getY());
		do {
			Point* target = new Point(endEffector->getX(), endEffector->getY());
			allowedIterations++;
			//double maxLength = max;
			moveRoot(rootPosition);
			for (int i = numSegments - 1; i >= 0; i--) {
				makeSegmentFollow(getSegment(i), target);
				*target = getSegment(i)->getStart();
				//maxLength -= getSegment(i)->getLength();
			}
			delete target;
		} while (!(rootSegment->getStart().isEqualTo(&rootPosition)) && allowedIterations <
ALLOWED);
		moveRoot(rootPosition);

	}

	// The printRobotInfo function prints the information of all the segments contained in the robot.
It makes use of the printSegmentInfo function
	void printRobotInfo(bool mode) {
		for (int i = 0; i < numSegments; i++) {
			cout << "Segment " << (i + 1) << ": ";
			getSegment(i)->printSegmentInfo(mode);
			cout << endl;
		}
	}

	// The printRobot function makes use of the DhiyaaGraphics class to draw the robot onto the output
screen in a given resolution. It also prints the last robot into a file called "LastGraph.txt"
	void printRobot(int resolution) {
		// Only draws the robot if it isn't empty
		if (numSegments > 0) {
			DhiyaaGraphics artist(resolution); // Creates a canvas to draw onto with
specified resolution
			artist.clearCanvas(); // Clear the canvas before anything is drawn onto it
			artist.addRobotToCanvas(rootSegment, numSegments); // Adds the robot to the
canvas
			artist.printToFile(); // Prints the canvas on the output screen and a file called
"LastGraph.txt"
			artist.clearCanvas(); // Clears the canvas again (not a very necessary step as
the DhiyaaGraphics object is going to be destructed)
		}
		// If the robot is empty tell the user there's nothing to draw
		else {
			cout << "Nothing to draw . . .\nThe robot is empty . . ." << endl;
		}
	}

	// Deallocating the dynamic memory used for all the segments in the chain
	~Robot() {
```

```
                for (int i = numSegments-1; i >= 0; i--) { // Deleting in reverse because the only way to
access a segment is through the segment before it
                        delete getSegment(i);
                }
        }
};
```

DhiyaaGraphics class:

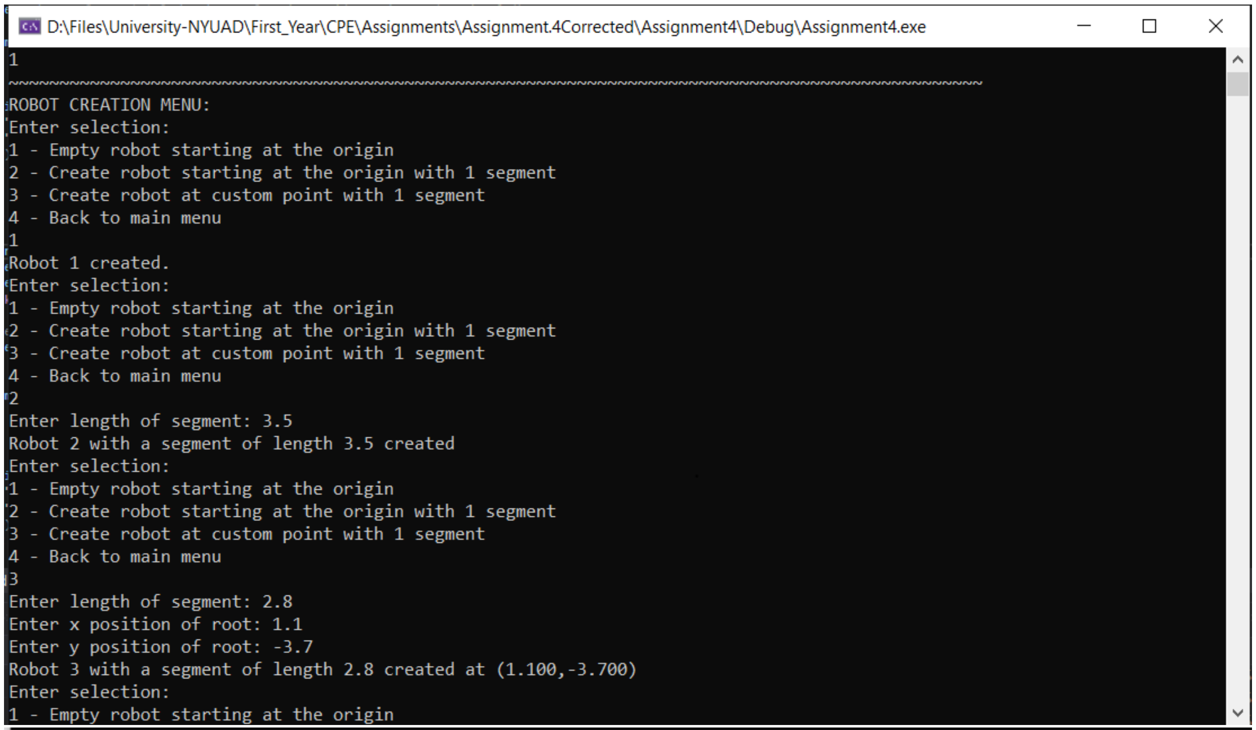
```
#pragma once
#include <fstream>
#include "Segment.h"
#define DEFAULT 50
#define RESOLUTION 100
#define PI 3.14159265359

using namespace std;

class DhiyaaGraphics {
private:
        // To navigate through a robot
        Segment* rootSegment;
        int numSegments;
        // Width of the canvas
        int s;
        // A boolean matrix representing pixels to draw into
        bool** canvas;
        // Points used to keep track of scaling of the axes
        Point topCorner;
        Point bottomCorner;
public:
        // Default Constructor: Creates an empty canvas of size 2 * DEFAULT
        DhiyaaGraphics() {
                s = 2 * DEFAULT;
                canvas = new bool* [s];
                for (int r = 0; r < s; r++) {
                        canvas[r] = new bool[s];
                }
                topCorner = bottomCorner = Point();
                rootSegment = nullptr;
                numSegments = 0;
                clearCanvas();
        }
        // Non-Default Constructor: Creates an empty canvas of size 2 * width
        DhiyaaGraphics(int width) {
                s = 2 * width;
                canvas = new bool* [s];
                for (int r = 0; r < s; r++) {
                        canvas[r] = new bool[s];
                }
                topCorner = bottomCorner = Point();
                rootSegment = nullptr;
                numSegments = 0;
                clearCanvas();
        }
        // This getSEgment class is the exact same as the one in the Robot class. The reason it was put
here was to avoid circular inclusion
        Segment* getSegment(int index) {
                if (index >= numSegments) {
                        cout << "ERROR: there is no segment of this index" << endl;
                        return nullptr;
                }
                Segment* currentSegment = rootSegment;
                for (int i = 0; i < index; i++) {
                        currentSegment = currentSegment->getChild();
                }
                return currentSegment;
        }
        // SetUpEdges finds the point farthest away from the origin and sets up the canvas's scale such
that it can fit the robot
```

```
        void setUpEdges() {
                double maxAbs = getSegment(0)->getEnd().getX();
                // Fincing the macimum distance from the origin
                for (int i = 0; i < numSegments; i++) {
                        if (abs(getSegment(i)->getEnd().getX()) > maxAbs) {
                                maxAbs = abs(getSegment(i)->getEnd().getX());
                        }
                        if (abs(getSegment(i)->getStart().getX()) > maxAbs) {
                                maxAbs = abs(getSegment(i)->getStart().getX());
                        }
                        if (abs(getSegment(i)->getEnd().getY()) > maxAbs) {
                                maxAbs = abs(getSegment(i)->getEnd().getY());
                        }
                        if (abs(getSegment(i)->getStart().getY()) > maxAbs) {
                                maxAbs = abs(getSegment(i)->getStart().getY());
                        }
                }
                // Adding padding for beauty
                double padding = maxAbs / 5;
                maxAbs += padding;
                topCorner = Point(maxAbs, maxAbs);
                bottomCorner = Point(-maxAbs, -maxAbs);
        }

        // This dunction receives the root segment of a robot and the number of segments in the robot to
add it to the
        void addRobotToCanvas(Segment* root, int size) {
                rootSegment = root;
                numSegments = size;
                setUpEdges(); // Setting up the scale
                for (int i = 0; i < numSegments; i++) {
                        double yPrev = getSegment(i)->getStart().getY();
                        double stepX = (getSegment(i)->getEnd().getX() - getSegment(i)-
>getStart().getX()) / (RESOLUTION); // The step size in x as we move through the segment
                        // For positive stepX
                        if (stepX == 0) // Any 90 degree angle will be slightly skewed for tan(angle) to
be defined
                                stepX = 0.000000000000000001;
                        if (stepX > 0) {
                                for (double j = getSegment(i)->getStart().getX(); j < getSegment(i)-
>getEnd().getX(); j += stepX) {
                                        double yCurr;
                                        yCurr = stepX * tan(getSegment(i)->getAngle()) + yPrev; // Basic
trigonometry
                                        yPrev = yCurr;
                                        canvas[(int)((double)s / 2 - (yCurr / topCorner.getY()) * (s /
2))][(int)((double)s / 2 + (j / topCorner.getX()) * (s / 2))] = true; // Filling uo the correspoding point
in the canvas
                                }
                        }
                        // For negative stepX
                        else {
                                for (double j = 0; j < getSegment(i)->getStart().getX() - getSegment(i)-
>getEnd().getX(); j -= stepX) {
                                        double yCurr;
                                        yCurr = stepX * tan(getSegment(i)->getAngle()) + yPrev; // Basic
trigonemety
                                        yPrev = yCurr;
                                        canvas[(int)((double)s / 2 - (yCurr / topCorner.getY()) * (s /
2))][(int)((double)s / 2 + ((getSegment(i)->getStart().getX() - j) / topCorner.getX()) * (s / 2))] = true;
// Filling uo the correspoding point in the canvas
                                }
                        }
                }
        }

        // Sets all the booleans in the matrix as false (0)
        void clearCanvas() {
                for (int r = 0; r < s; r++) {
```

```
                for (int c = 0; c < s; c++) {
                        canvas[r][c] = false;
                }
        }
}

// Draws axes and translates the boolean array into * _ and |
void printToFile() {
        ofstream writer("LastGraph.txt", ios::out);
        if (writer.fail()) {
                cerr << "ERROR: Failed to open file . . ." << endl;
                exit(-1);
        }
        double step = topCorner.getY() / (s / 2);
        for (double x = -topCorner.getX(); x <= topCorner.getX(); x += step * (s / 10)) {
                writer << left << fixed << setprecision(1) << setw(s / 16) << x;
        }
        writer << endl;
        cout << endl;
        for (double x = -topCorner.getX(); x <= topCorner.getX(); x += step * (s / 10)) {
                writer << setw(s / 10) << '|';
        }
        writer << endl;
        cout << endl;
        double y = topCorner.getY();
        for (int r = 0; r < s - 1; r += 2, y -= 2 * step) {
                for (int c = 0; c < s; c++) {
                        if (canvas[r][c] && canvas[r + 1][c]) {
                                writer << '|';
                        }
                        else if (canvas[r][c]) {
                                writer << '*';
                        }
                        else if (canvas[r + 1][c]) {
                                writer << '_';
                        }
                        else {
                                writer << ' ';
                        }
                }
                if ((r / 2) % (s / 20) == 0) {
                        writer << "  -- " << left << fixed << setprecision(1) << y;
                }
                writer << endl;
        }
        for (double x = -topCorner.getX(); x <= topCorner.getX(); x += step * (s / 10)) {
                cout << left << fixed << setprecision(1) << setw(s / 10) << x;
        }
        cout << endl;
        for (double x = -topCorner.getX(); x <= topCorner.getX(); x += step * (s / 10)) {
                cout << setw(s / 10) << '|';
        }
        cout << endl;
        y = topCorner.getY();
        int r;
        for (r = 0; r < s - 1; r += 2, y -= 2 * step) {
                for (int c = 0; c < s; c++) {
                        if (canvas[r][c] && canvas[r + 1][c]) {
                                cout << '|';
                        }
                        else if (canvas[r][c]) {
                                cout << '*';
                        }
                        else if (canvas[r + 1][c]) {
                                cout << '_';
                        }
                        else {
                                cout << ' ';
                        }
                        }
                        if ((r / 2) % (s / 20) == 0) {
                                cout << "  -- " << left << fixed << setprecision(1) << y;
                        }
                        cout << endl;
                }
                writer.close();
        }

        // Destructor: Deallocates the dynamically allocated memory for the canvas
        ~DhiyaaGraphics() {
                for (int i = 0; i < s; i++) {
                        delete[] canvas[i];
                }
                delete[] canvas;
        }
};
```

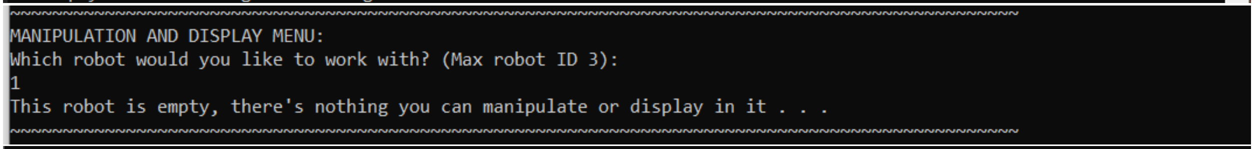
Step 5 – Software Testing & Verification:

Task 1 – Robot Creation:

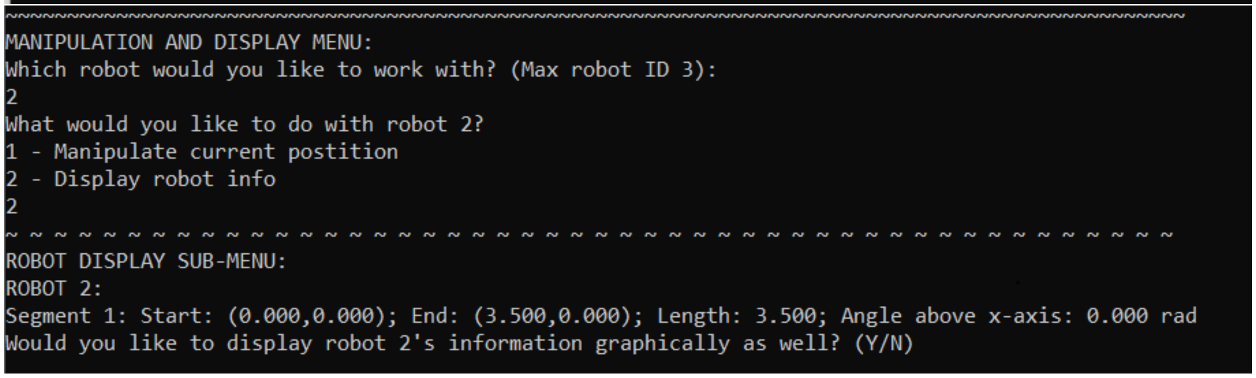


D:\Files\University-NYUAD\First_Year\CPE\Assignments\Assignment.4Corrected\Assignment4\Debug\Assignment4.exe

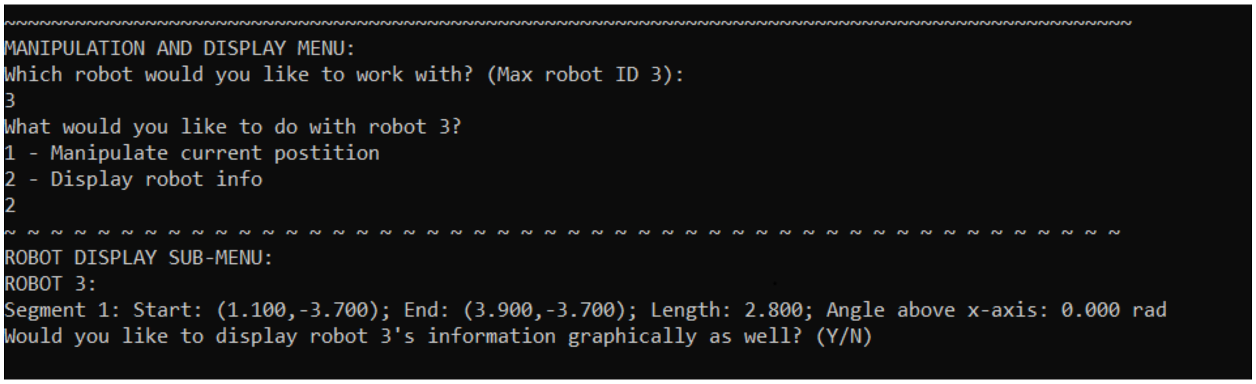
```
1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ROBOT CREATION MENU:
Enter selection:
1 - Empty robot starting at the origin
2 - Create robot starting at the origin with 1 segment
3 - Create robot at custom point with 1 segment
4 - Back to main menu
1
Robot 1 created.
Enter selection:
1 - Empty robot starting at the origin
2 - Create robot starting at the origin with 1 segment
3 - Create robot at custom point with 1 segment
4 - Back to main menu
2
Enter length of segment: 3.5
Robot 2 with a segment of length 3.5 created
Enter selection:
1 - Empty robot starting at the origin
2 - Create robot starting at the origin with 1 segment
3 - Create robot at custom point with 1 segment
4 - Back to main menu
3
Enter length of segment: 2.8
Enter x position of root: 1.1
Enter y position of root: -3.7
Robot 3 with a segment of length 2.8 created at (1.100,-3.700)
Enter selection:
1 - Empty robot starting at the origin
```



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MANIPULATION AND DISPLAY MENU:
Which robot would you like to work with? (Max robot ID 3):
1
This robot is empty, there's nothing you can manipulate or display in it . . .
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```



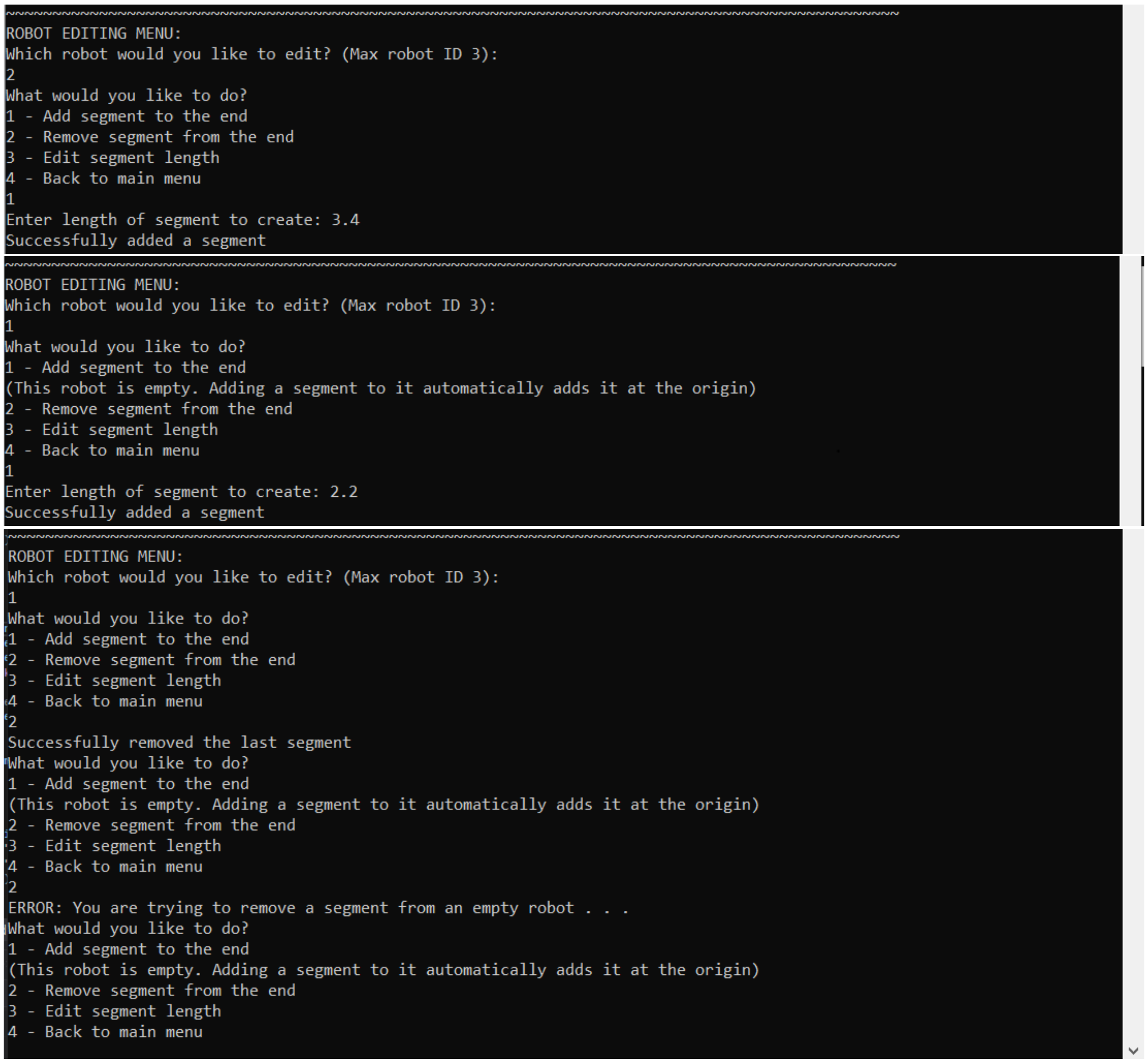
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MANIPULATION AND DISPLAY MENU:
Which robot would you like to work with? (Max robot ID 3):
2
What would you like to do with robot 2?
1 - Manipulate current postition
2 - Display robot info
2
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 2:
Segment 1: Start: (0.000,0.000); End: (3.500,0.000); Length: 3.500; Angle above x-axis: 0.000 rad
Would you like to display robot 2's information graphically as well? (Y/N)
```



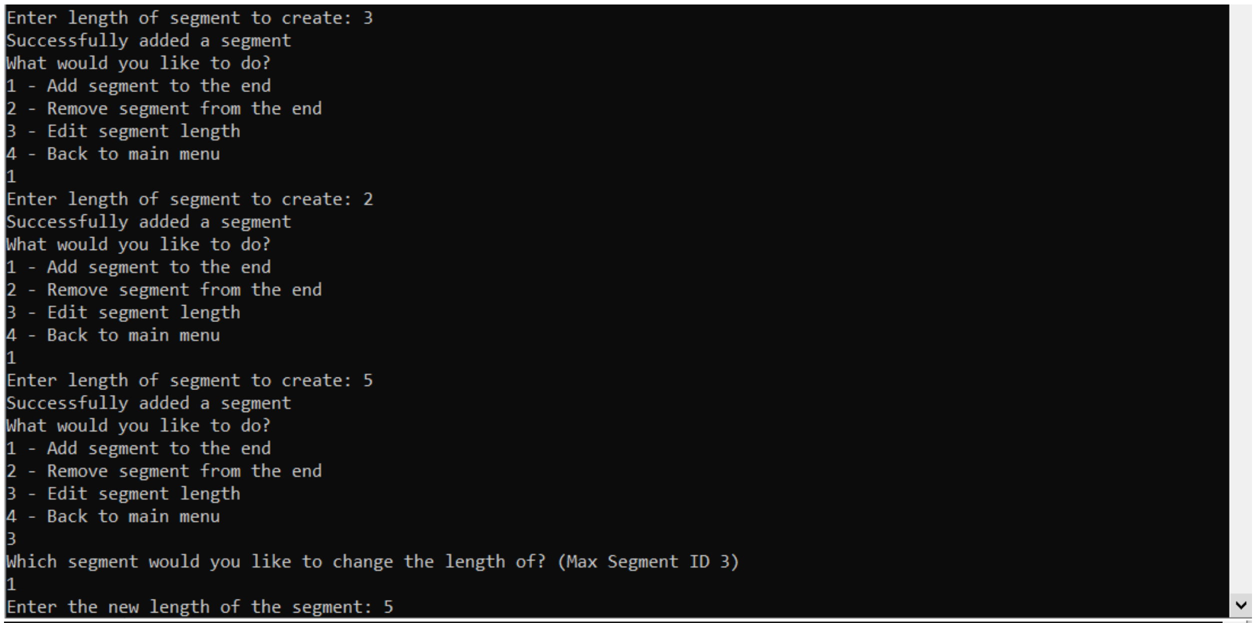
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MANIPULATION AND DISPLAY MENU:
Which robot would you like to work with? (Max robot ID 3):
3
What would you like to do with robot 3?
1 - Manipulate current postition
2 - Display robot info
2
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 3:
Segment 1: Start: (1.100,-3.700); End: (3.900,-3.700); Length: 2.800; Angle above x-axis: 0.000 rad
Would you like to display robot 3's information graphically as well? (Y/N)
```

All robots created as specified by the user => This task passes

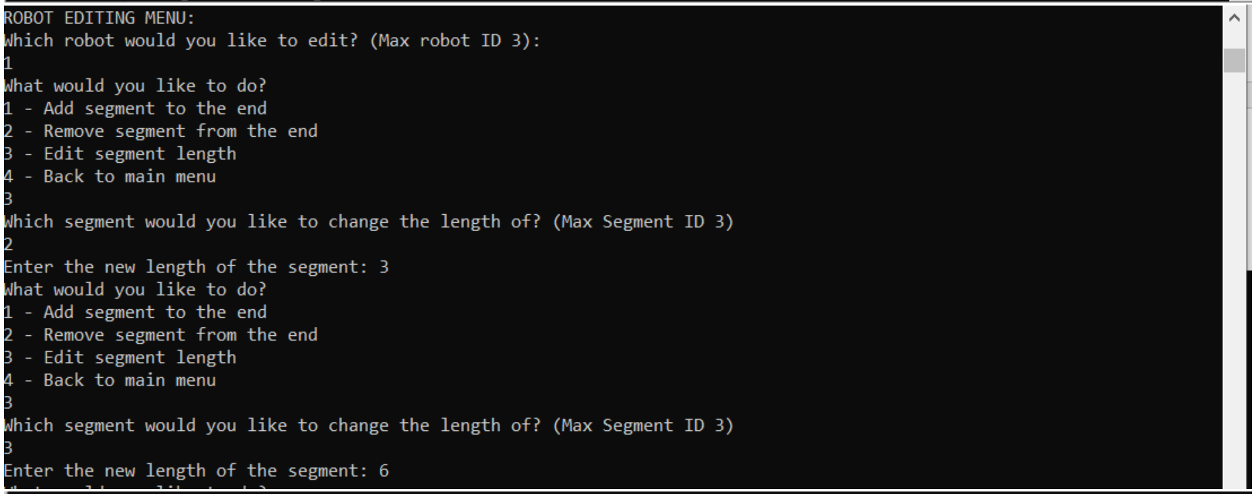
<u>Task 2 – Robot Editing:</u>



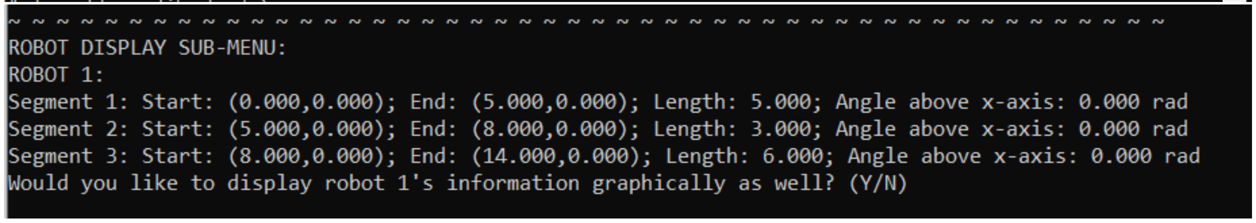
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ROBOT EDITING MENU:
Which robot would you like to edit? (Max robot ID 3):
2
What would you like to do?
1 - Add segment to the end
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
1
Enter length of segment to create: 3.4
Successfully added a segment
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ROBOT EDITING MENU:
Which robot would you like to edit? (Max robot ID 3):
1
What would you like to do?
1 - Add segment to the end
(This robot is empty. Adding a segment to it automatically adds it at the origin)
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
1
Enter length of segment to create: 2.2
Successfully added a segment
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ROBOT EDITING MENU:
Which robot would you like to edit? (Max robot ID 3):
1
What would you like to do?
1 - Add segment to the end
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
2
Successfully removed the last segment
What would you like to do?
1 - Add segment to the end
(This robot is empty. Adding a segment to it automatically adds it at the origin)
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
2
ERROR: You are trying to remove a segment from an empty robot . . .
What would you like to do?
1 - Add segment to the end
(This robot is empty. Adding a segment to it automatically adds it at the origin)
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
```



```
Enter length of segment to create: 3
Successfully added a segment
What would you like to do?
1 - Add segment to the end
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
1
Enter length of segment to create: 2
Successfully added a segment
What would you like to do?
1 - Add segment to the end
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
1
Enter length of segment to create: 5
Successfully added a segment
What would you like to do?
1 - Add segment to the end
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
3
Which segment would you like to change the length of? (Max Segment ID 3)
1
Enter the new length of the segment: 5
```



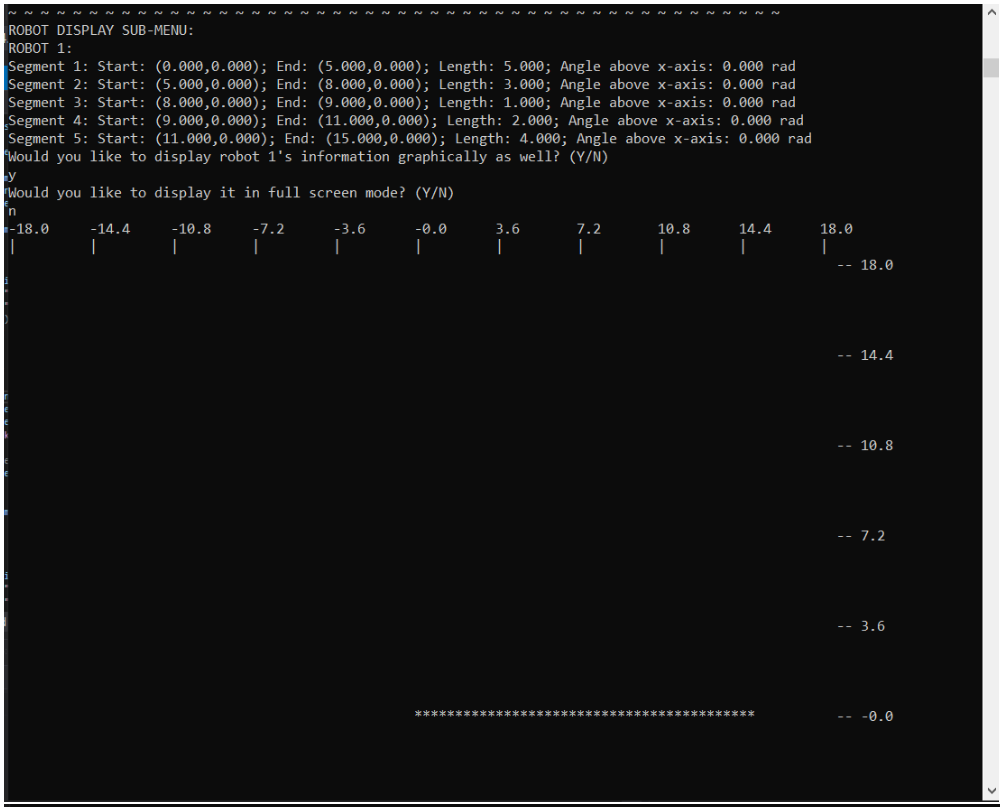
```
ROBOT EDITING MENU:
Which robot would you like to edit? (Max robot ID 3):
1
What would you like to do?
1 - Add segment to the end
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
3
Which segment would you like to change the length of? (Max Segment ID 3)
2
Enter the new length of the segment: 3
What would you like to do?
1 - Add segment to the end
2 - Remove segment from the end
3 - Edit segment length
4 - Back to main menu
3
Which segment would you like to change the length of? (Max Segment ID 3)
3
Enter the new length of the segment: 6
```



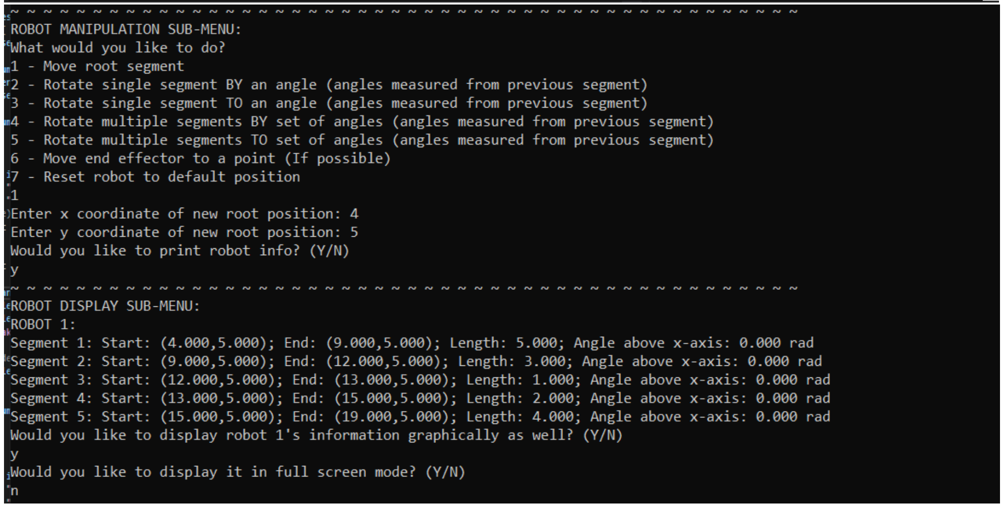
```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (0.000,0.000); End: (5.000,0.000); Length: 5.000; Angle above x-axis: 0.000 rad
Segment 2: Start: (5.000,0.000); End: (8.000,0.000); Length: 3.000; Angle above x-axis: 0.000 rad
Segment 3: Start: (8.000,0.000); End: (14.000,0.000); Length: 6.000; Angle above x-axis: 0.000 rad
Would you like to display robot 1's information graphically as well? (Y/N)
```

All the cases worked as expected with no errors => This task passes

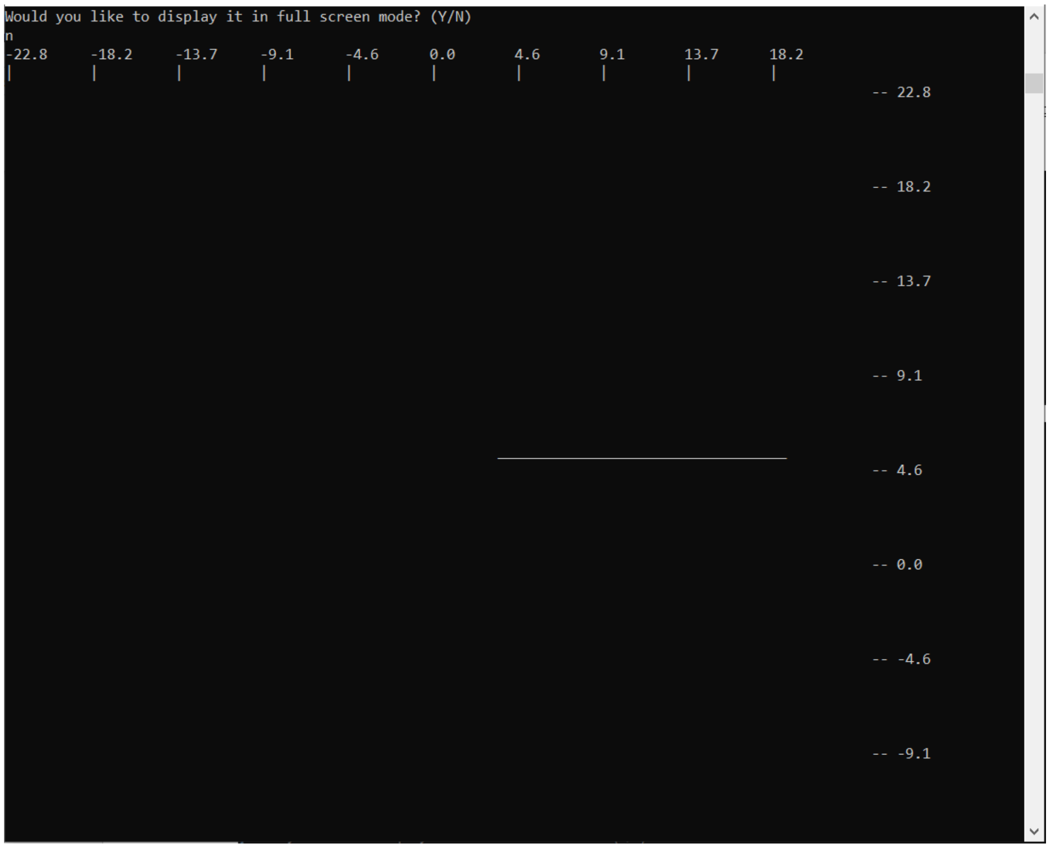
<u>Task 3 – Forward Kinematics</u>



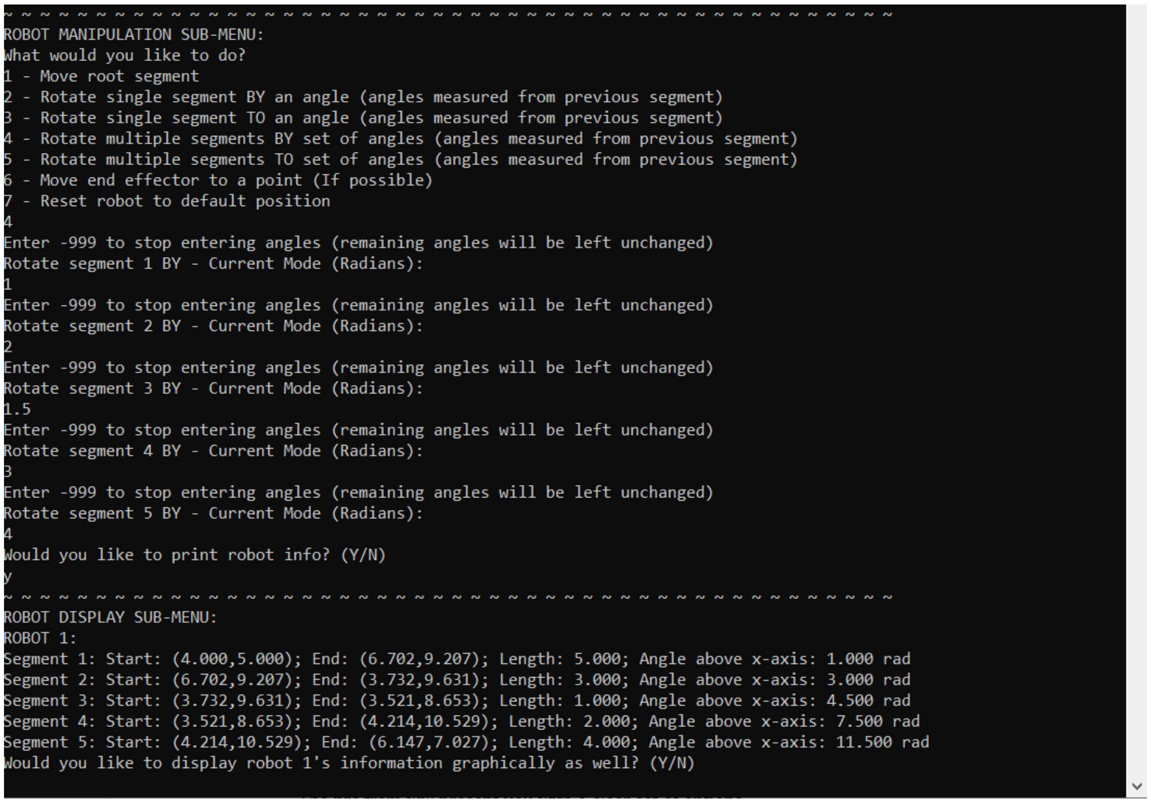
```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (0.000,0.000); End: (5.000,0.000); Length: 5.000; Angle above x-axis: 0.000 rad
Segment 2: Start: (5.000,0.000); End: (8.000,0.000); Length: 3.000; Angle above x-axis: 0.000 rad
Segment 3: Start: (8.000,0.000); End: (9.000,0.000); Length: 1.000; Angle above x-axis: 0.000 rad
Segment 4: Start: (9.000,0.000); End: (11.000,0.000); Length: 2.000; Angle above x-axis: 0.000 rad
Segment 5: Start: (11.000,0.000); End: (15.000,0.000); Length: 4.000; Angle above x-axis: 0.000 rad
Would you like to display robot 1's information graphically as well? (Y/N)
y
Would you like to display it in full screen mode? (Y/N)
n
-18.0     -14.4     -10.8     -7.2      -3.6      -0.0      3.6       7.2       10.8      14.4      18.0
|         |         |         |         |         |         |         |         |         |         |
                                                                                                      -- 18.0




                                                                                                      -- 14.4




                                                                                                      -- 10.8




                                                                                                      -- 7.2




                                                                                                      -- 3.6




                                                  ******************************************          -- -0.0
```



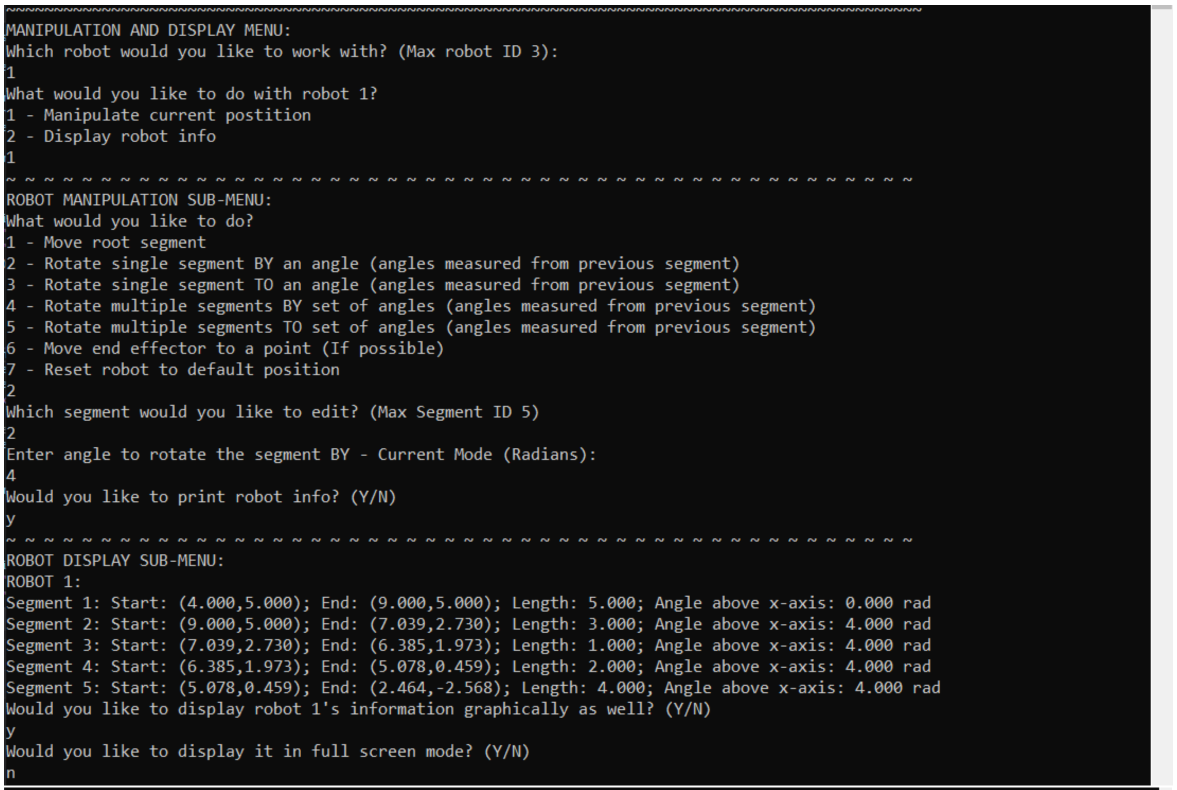
```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
1
Enter x coordinate of new root position: 4
Enter y coordinate of new root position: 5
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (4.000,5.000); End: (9.000,5.000); Length: 5.000; Angle above x-axis: 0.000 rad
Segment 2: Start: (9.000,5.000); End: (12.000,5.000); Length: 3.000; Angle above x-axis: 0.000 rad
Segment 3: Start: (12.000,5.000); End: (13.000,5.000); Length: 1.000; Angle above x-axis: 0.000 rad
Segment 4: Start: (13.000,5.000); End: (15.000,5.000); Length: 2.000; Angle above x-axis: 0.000 rad
Segment 5: Start: (15.000,5.000); End: (19.000,5.000); Length: 4.000; Angle above x-axis: 0.000 rad
Would you like to display robot 1's information graphically as well? (Y/N)
y
Would you like to display it in full screen mode? (Y/N)
n
```



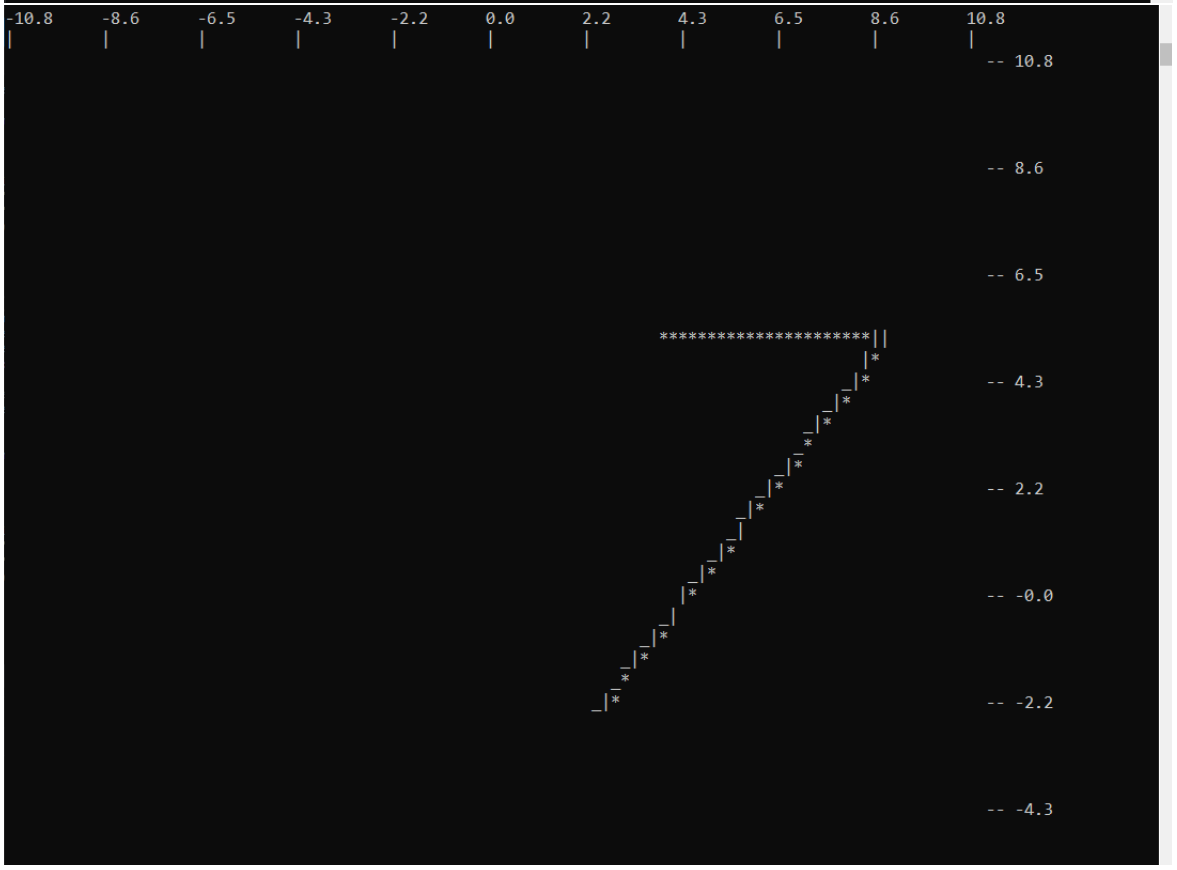
```
Would you like to display it in full screen mode? (Y/N)
n
-22.8     -18.2     -13.7     -9.1      -4.6      0.0       4.6       9.1       13.7      18.2
|         |         |         |         |         |         |         |         |         |
                                                                                                    -- 22.8


                                                                                                    -- 18.2


                                                                                                    -- 13.7


                                                                                                    -- 9.1


                                                                                                    -- 4.6


                                                                                                    -- 0.0


                                                                                                    -- -4.6


                                                                                                    -- -9.1
```

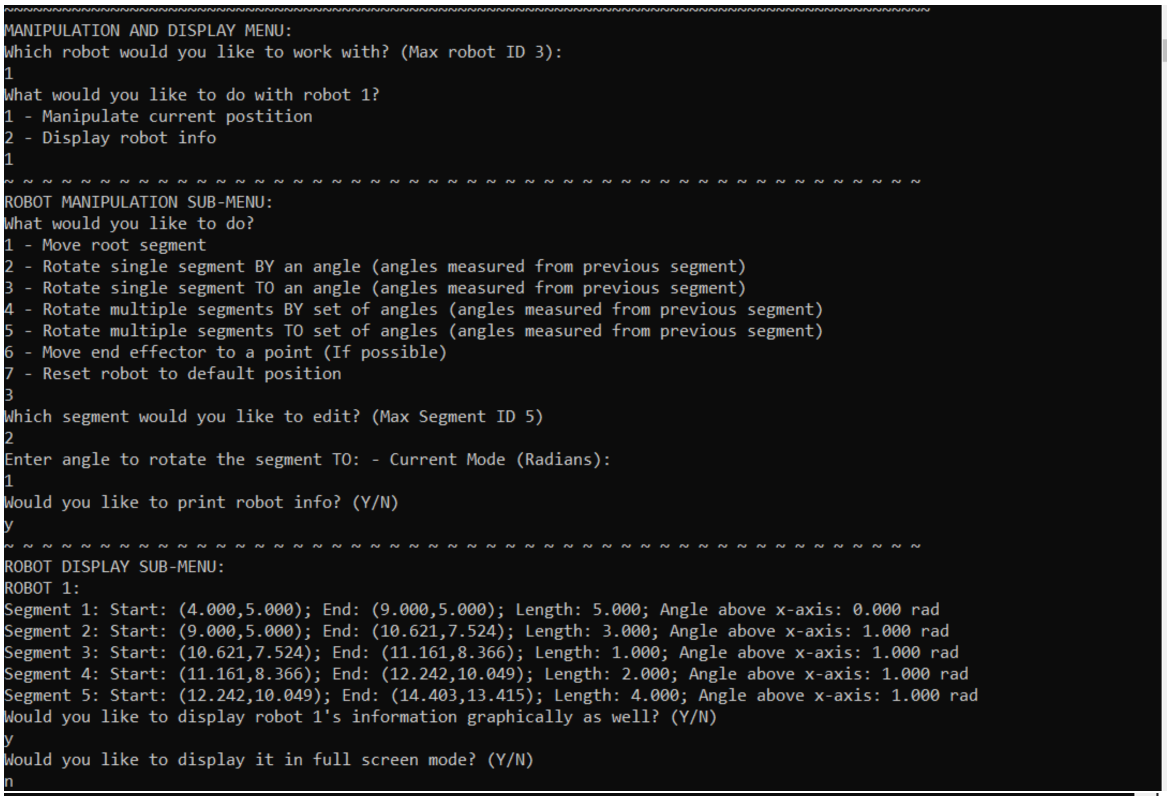


```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
4
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 1 BY - Current Mode (Radians):
1
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 2 BY - Current Mode (Radians):
2
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 3 BY - Current Mode (Radians):
1.5
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 4 BY - Current Mode (Radians):
3
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 5 BY - Current Mode (Radians):
4
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (4.000,5.000); End: (6.702,9.207); Length: 5.000; Angle above x-axis: 1.000 rad
Segment 2: Start: (6.702,9.207); End: (3.732,9.631); Length: 3.000; Angle above x-axis: 3.000 rad
Segment 3: Start: (3.732,9.631); End: (3.521,8.653); Length: 1.000; Angle above x-axis: 4.500 rad
Segment 4: Start: (3.521,8.653); End: (4.214,10.529); Length: 2.000; Angle above x-axis: 7.500 rad
Segment 5: Start: (4.214,10.529); End: (6.147,7.027); Length: 4.000; Angle above x-axis: 11.500 rad
Would you like to display robot 1's information graphically as well? (Y/N)
```

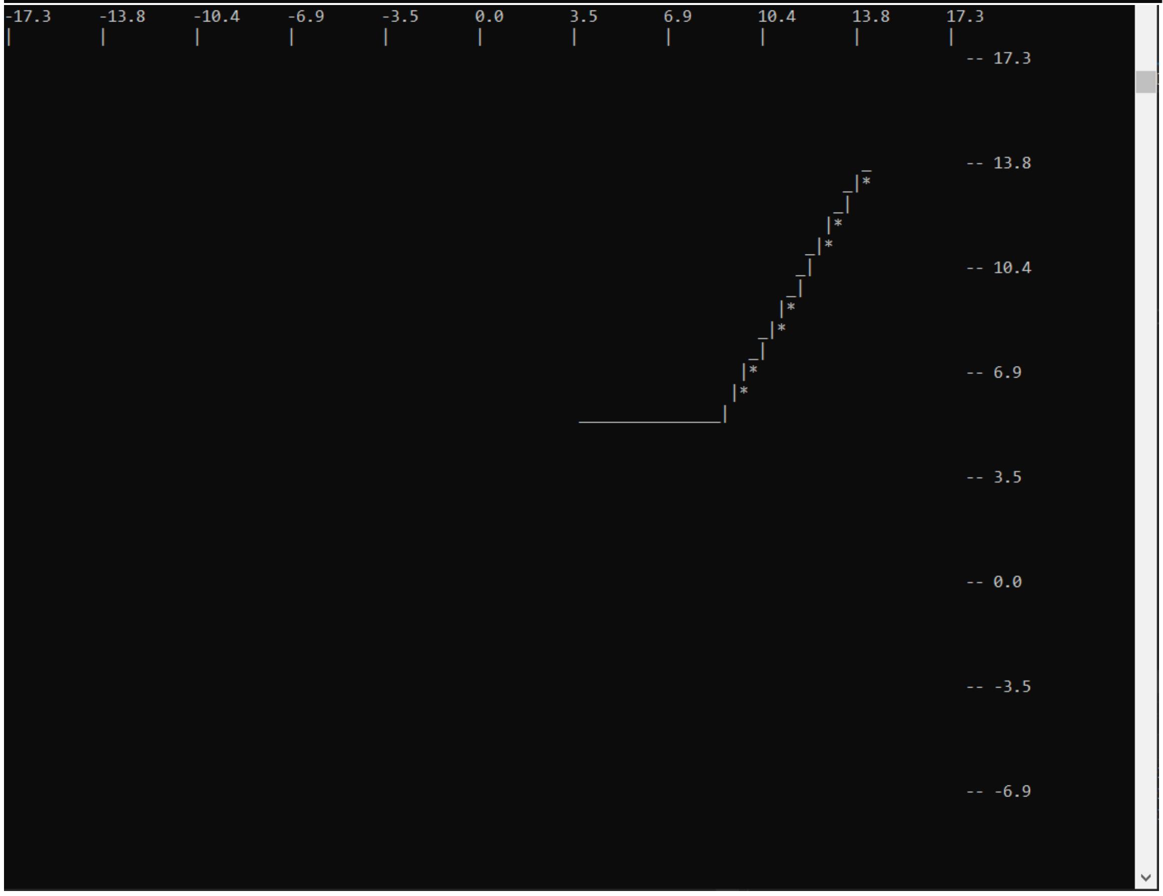


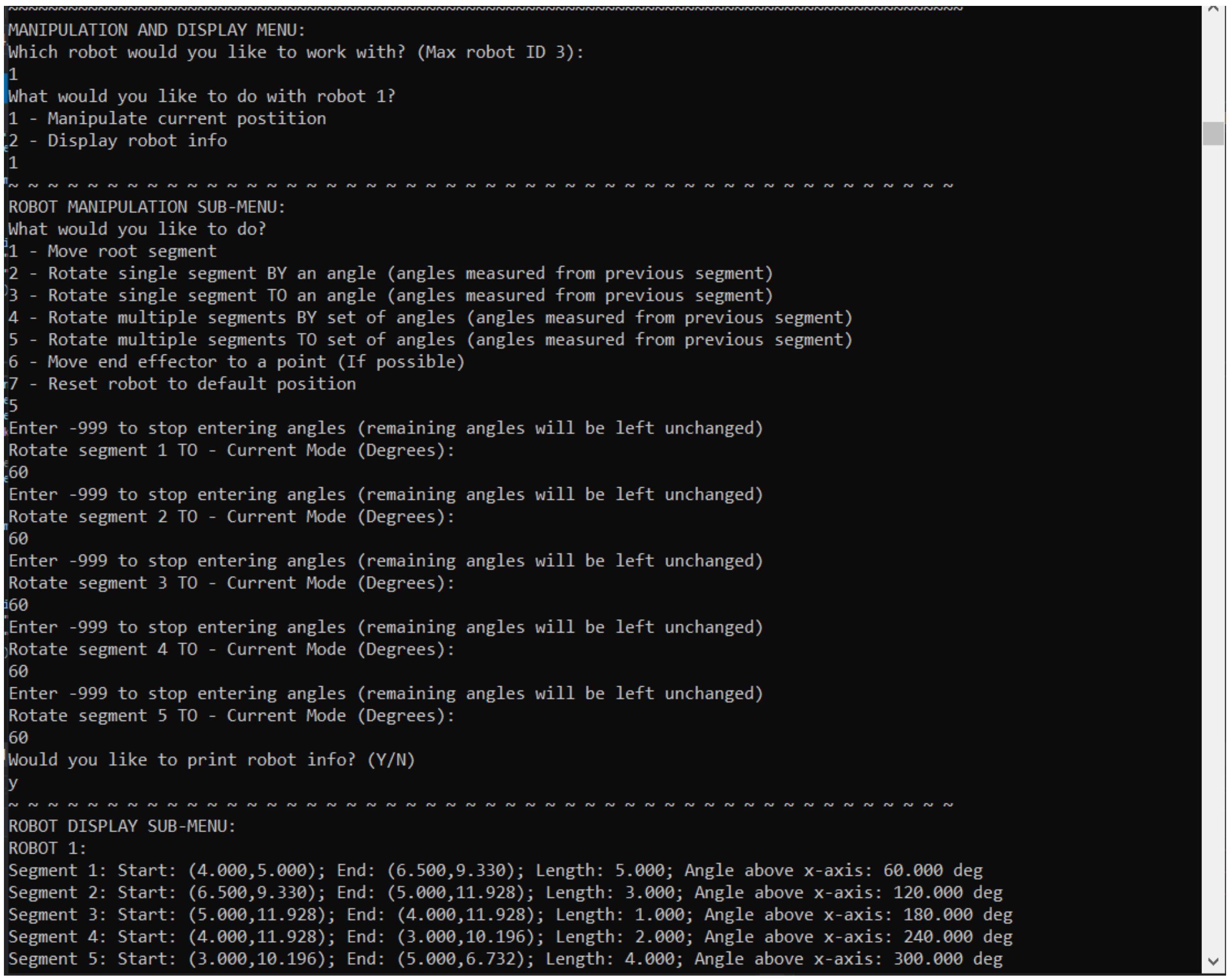
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MANIPULATION AND DISPLAY MENU:
Which robot would you like to work with? (Max robot ID 3):
1
What would you like to do with robot 1?
1 - Manipulate current postition
2 - Display robot info
1
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
2
Which segment would you like to edit? (Max Segment ID 5)
2
Enter angle to rotate the segment BY - Current Mode (Radians):
4
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (4.000,5.000); End: (9.000,5.000); Length: 5.000; Angle above x-axis: 0.000 rad
Segment 2: Start: (9.000,5.000); End: (7.039,2.730); Length: 3.000; Angle above x-axis: 4.000 rad
Segment 3: Start: (7.039,2.730); End: (6.385,1.973); Length: 1.000; Angle above x-axis: 4.000 rad
Segment 4: Start: (6.385,1.973); End: (5.078,0.459); Length: 2.000; Angle above x-axis: 4.000 rad
Segment 5: Start: (5.078,0.459); End: (2.464,-2.568); Length: 4.000; Angle above x-axis: 4.000 rad
Would you like to display robot 1's information graphically as well? (Y/N)
y
Would you like to display it in full screen mode? (Y/N)
n
```





```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MANIPULATION AND DISPLAY MENU:
Which robot would you like to work with? (Max robot ID 3):
1
What would you like to do with robot 1?
1 - Manipulate current postition
2 - Display robot info
1
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
3
Which segment would you like to edit? (Max Segment ID 5)
2
Enter angle to rotate the segment TO: - Current Mode (Radians):
1
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (4.000,5.000); End: (9.000,5.000); Length: 5.000; Angle above x-axis: 0.000 rad
Segment 2: Start: (9.000,5.000); End: (10.621,7.524); Length: 3.000; Angle above x-axis: 1.000 rad
Segment 3: Start: (10.621,7.524); End: (11.161,8.366); Length: 1.000; Angle above x-axis: 1.000 rad
Segment 4: Start: (11.161,8.366); End: (12.242,10.049); Length: 2.000; Angle above x-axis: 1.000 rad
Segment 5: Start: (12.242,10.049); End: (14.403,13.415); Length: 4.000; Angle above x-axis: 1.000 rad
Would you like to display robot 1's information graphically as well? (Y/N)
y
Would you like to display it in full screen mode? (Y/N)
n
```

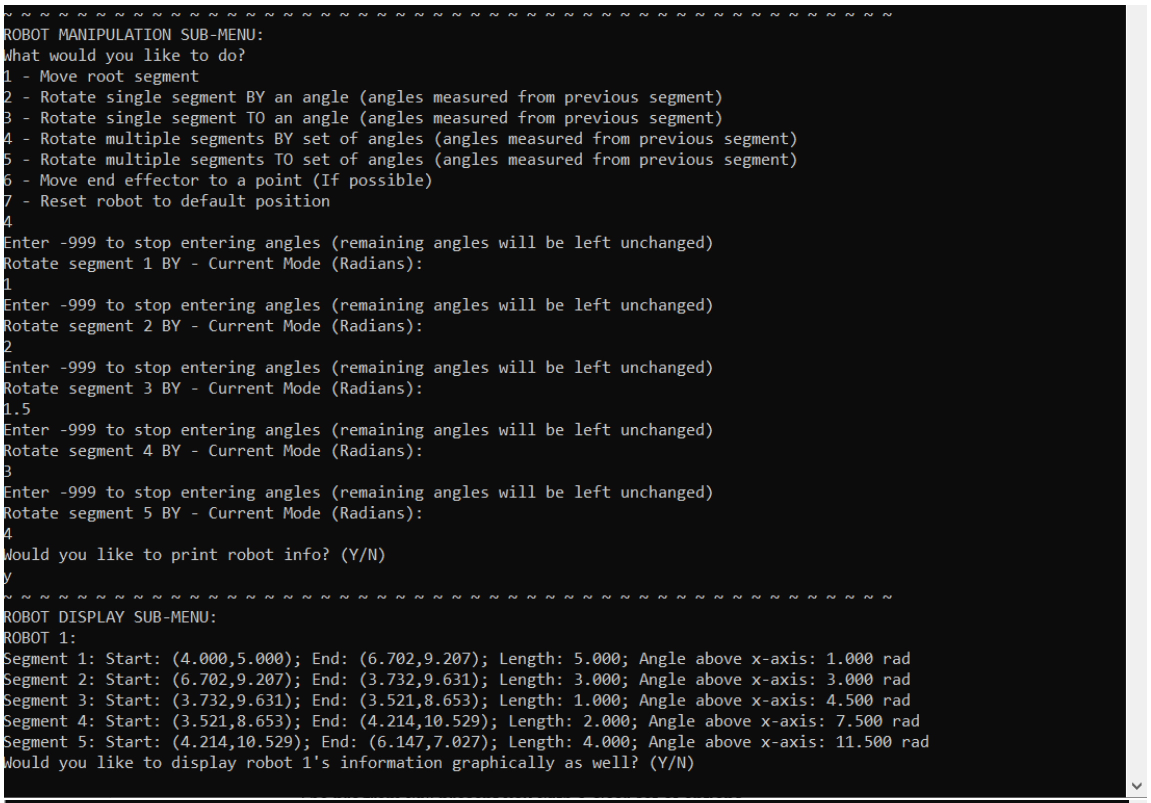




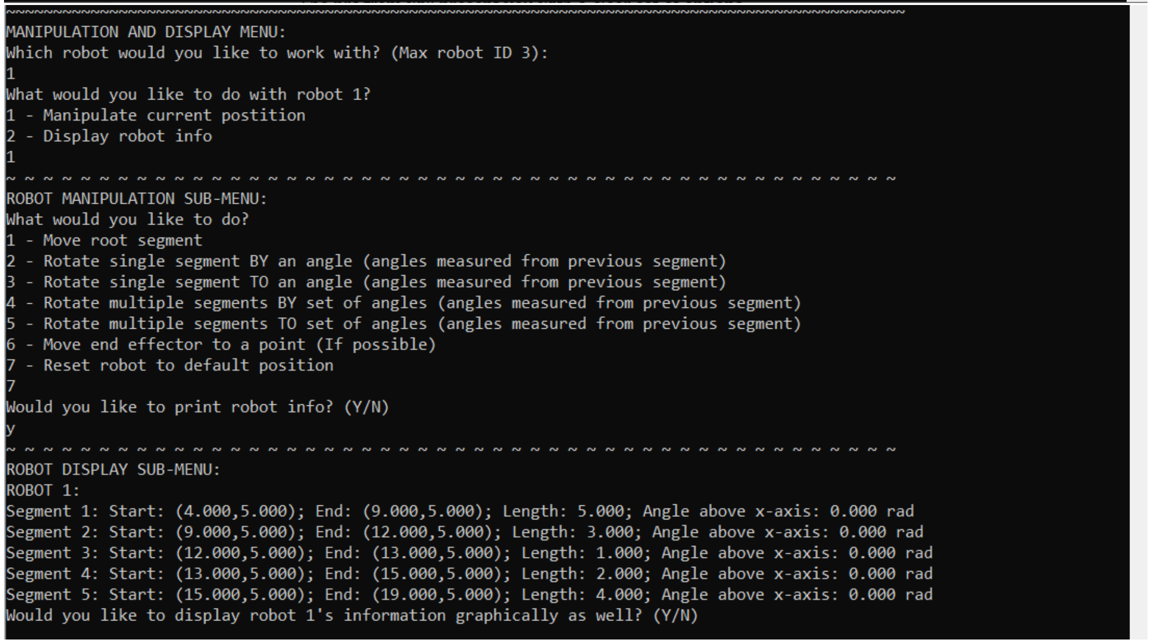
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MANIPULATION AND DISPLAY MENU:
Which robot would you like to work with? (Max robot ID 3):
1
What would you like to do with robot 1?
1 - Manipulate current postition
2 - Display robot info
1
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
5
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 1 TO - Current Mode (Degrees):
60
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 2 TO - Current Mode (Degrees):
60
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 3 TO - Current Mode (Degrees):
60
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 4 TO - Current Mode (Degrees):
60
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 5 TO - Current Mode (Degrees):
60
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (4.000,5.000); End: (6.500,9.330); Length: 5.000; Angle above x-axis: 60.000 deg
Segment 2: Start: (6.500,9.330); End: (5.000,11.928); Length: 3.000; Angle above x-axis: 120.000 deg
Segment 3: Start: (5.000,11.928); End: (4.000,11.928); Length: 1.000; Angle above x-axis: 180.000 deg
Segment 4: Start: (4.000,11.928); End: (3.000,10.196); Length: 2.000; Angle above x-axis: 240.000 deg
Segment 5: Start: (3.000,10.196); End: (5.000,6.732); Length: 4.000; Angle above x-axis: 300.000 deg
```

All the cases went as expected => This Task passes

<u>Task 4 – Resetting Robot to Default Position</u>



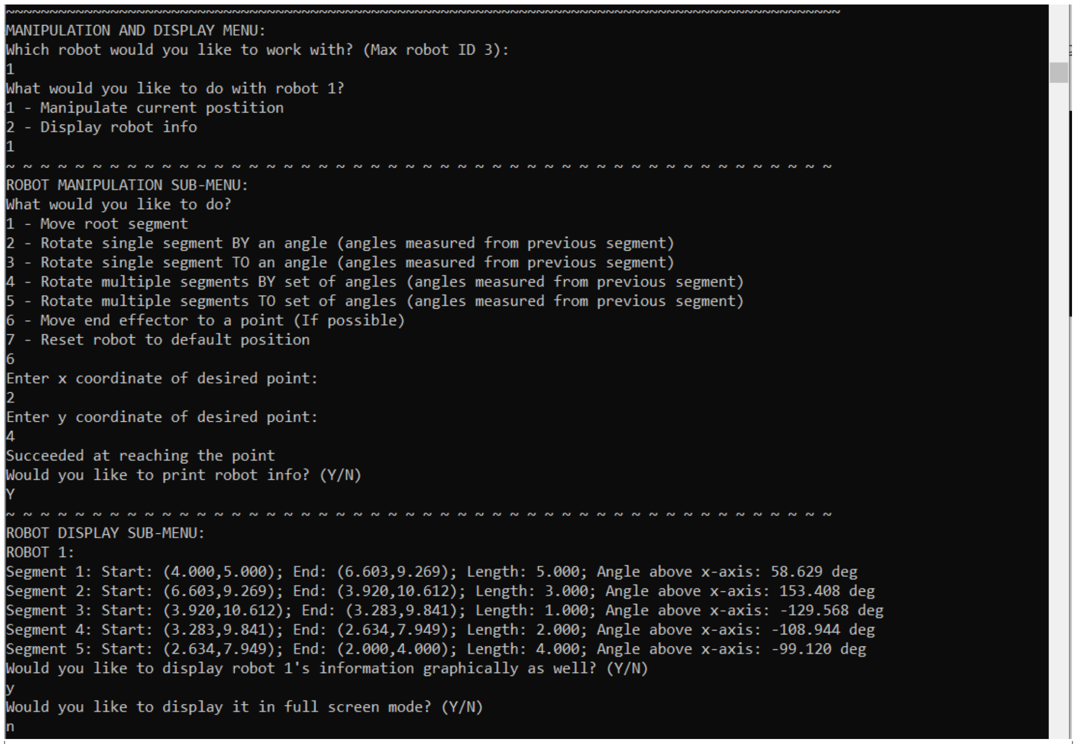
```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
4
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 1 BY - Current Mode (Radians):
1
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 2 BY - Current Mode (Radians):
2
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 3 BY - Current Mode (Radians):
1.5
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 4 BY - Current Mode (Radians):
3
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 5 BY - Current Mode (Radians):
4
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (4.000,5.000); End: (6.702,9.207); Length: 5.000; Angle above x-axis: 1.000 rad
Segment 2: Start: (6.702,9.207); End: (3.732,9.631); Length: 3.000; Angle above x-axis: 3.000 rad
Segment 3: Start: (3.732,9.631); End: (3.521,8.653); Length: 1.000; Angle above x-axis: 4.500 rad
Segment 4: Start: (3.521,8.653); End: (4.214,10.529); Length: 2.000; Angle above x-axis: 7.500 rad
Segment 5: Start: (4.214,10.529); End: (6.147,7.027); Length: 4.000; Angle above x-axis: 11.500 rad
Would you like to display robot 1's information graphically as well? (Y/N)
```



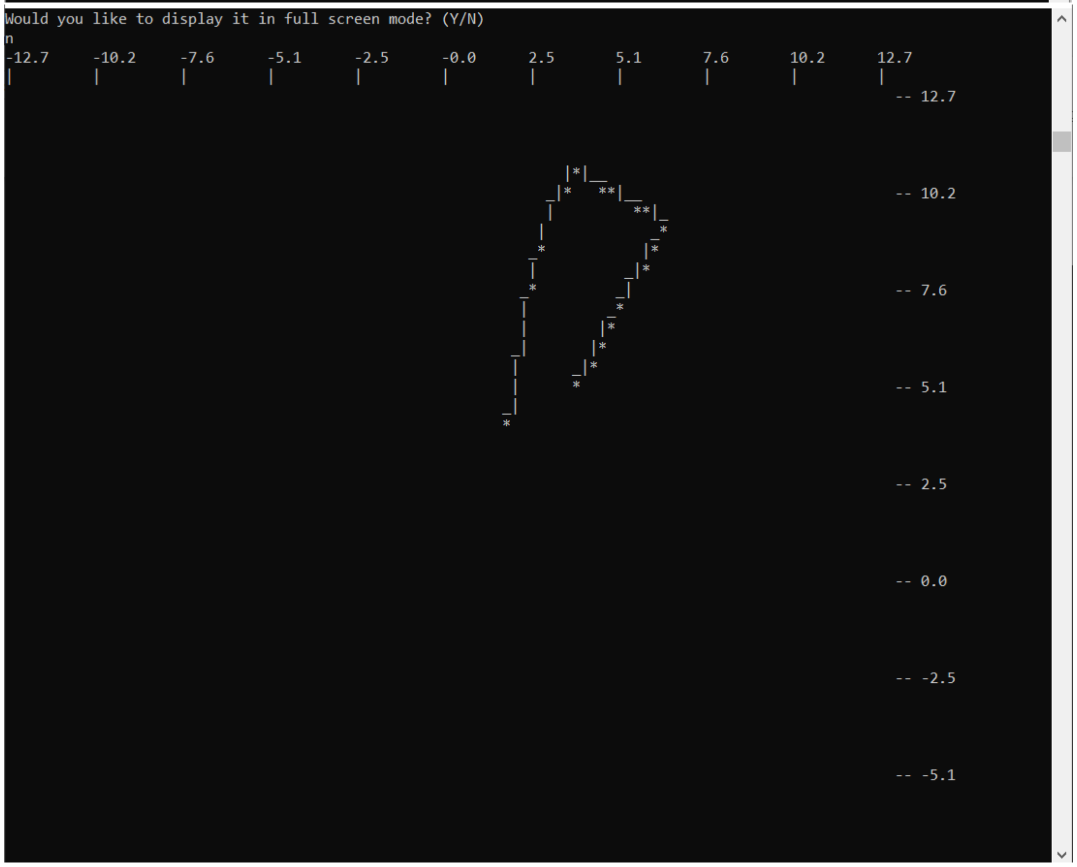
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MANIPULATION AND DISPLAY MENU:
Which robot would you like to work with? (Max robot ID 3):
1
What would you like to do with robot 1?
1 - Manipulate current postition
2 - Display robot info
1
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
7
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (4.000,5.000); End: (9.000,5.000); Length: 5.000; Angle above x-axis: 0.000 rad
Segment 2: Start: (9.000,5.000); End: (12.000,5.000); Length: 3.000; Angle above x-axis: 0.000 rad
Segment 3: Start: (12.000,5.000); End: (13.000,5.000); Length: 1.000; Angle above x-axis: 0.000 rad
Segment 4: Start: (13.000,5.000); End: (15.000,5.000); Length: 2.000; Angle above x-axis: 0.000 rad
Segment 5: Start: (15.000,5.000); End: (19.000,5.000); Length: 4.000; Angle above x-axis: 0.000 rad
Would you like to display robot 1's information graphically as well? (Y/N)
```

Worked as expected => This test case passes

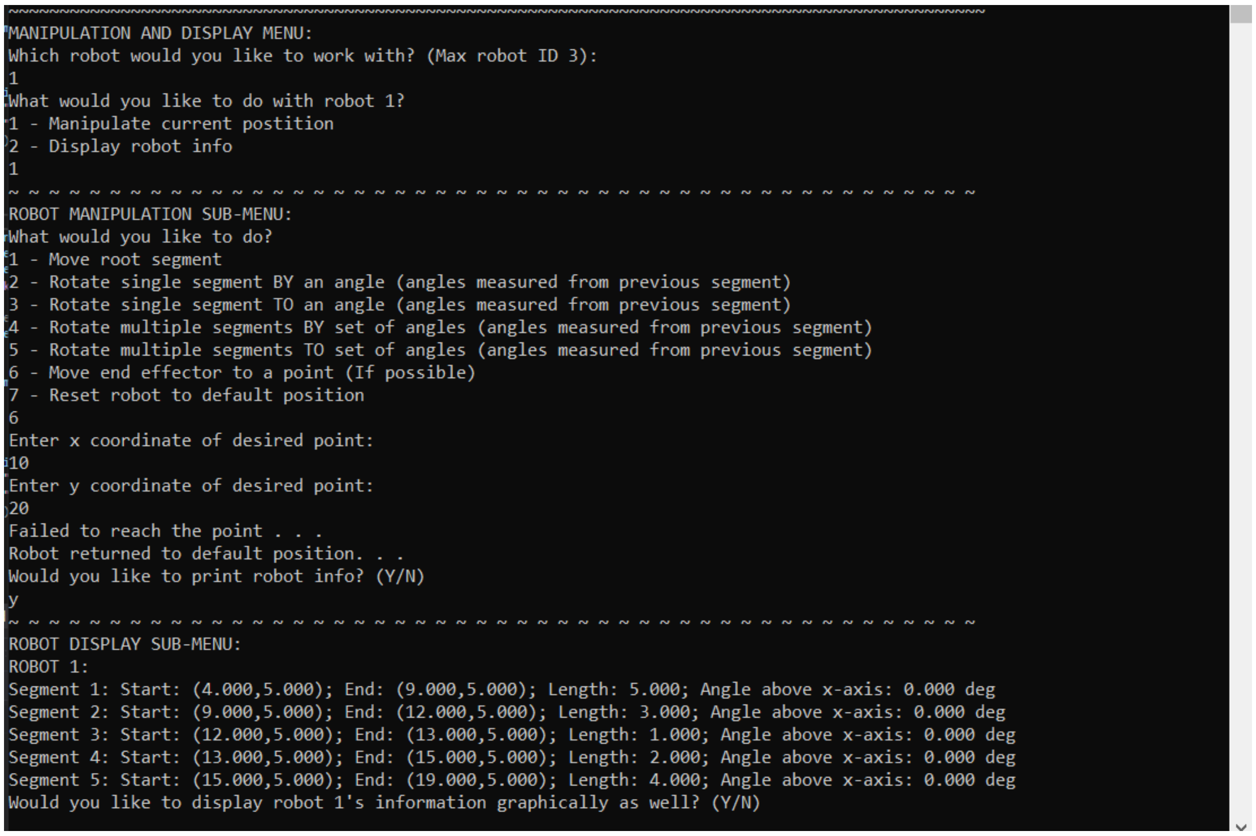
<u>Task 5 – Inverse Kinematics</u>



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MANIPULATION AND DISPLAY MENU:
Which robot would you like to work with? (Max robot ID 3):
1
What would you like to do with robot 1?
1 - Manipulate current postition
2 - Display robot info
1
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
6
Enter x coordinate of desired point:
2
Enter y coordinate of desired point:
4
Succeeded at reaching the point
Would you like to print robot info? (Y/N)
Y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (4.000,5.000); End: (6.603,9.269); Length: 5.000; Angle above x-axis: 58.629 deg
Segment 2: Start: (6.603,9.269); End: (3.920,10.612); Length: 3.000; Angle above x-axis: 153.408 deg
Segment 3: Start: (3.920,10.612); End: (3.283,9.841); Length: 1.000; Angle above x-axis: -129.568 deg
Segment 4: Start: (3.283,9.841); End: (2.634,7.949); Length: 2.000; Angle above x-axis: -108.944 deg
Segment 5: Start: (2.634,7.949); End: (2.000,4.000); Length: 4.000; Angle above x-axis: -99.120 deg
Would you like to display robot 1's information graphically as well? (Y/N)
y
Would you like to display it in full screen mode? (Y/N)
n
```



```
Would you like to display it in full screen mode? (Y/N)
n
-12.7     -10.2     -7.6      -5.1      -2.5      -0.0      2.5       5.1       7.6       10.2      12.7
|         |         |         |         |         |         |         |         |         |         |
                                                                                                      -- 12.7



                                                                |*|__
                                                              _|*   **|__                             -- 10.2
                                                              |         **|_
                                                             |            _*
                                                            _*           |*
                                                            |          _|*
                                                           _*         _|                              -- 7.6
                                                           |         _*
                                                           |        |*
                                                          _|       |*
                                                          |      _|*
                                                          |      *                                    -- 5.1
                                                         _|
                                                         *


                                                                                                      -- 2.5




                                                                                                      -- 0.0




                                                                                                      -- -2.5




                                                                                                      -- -5.1
```



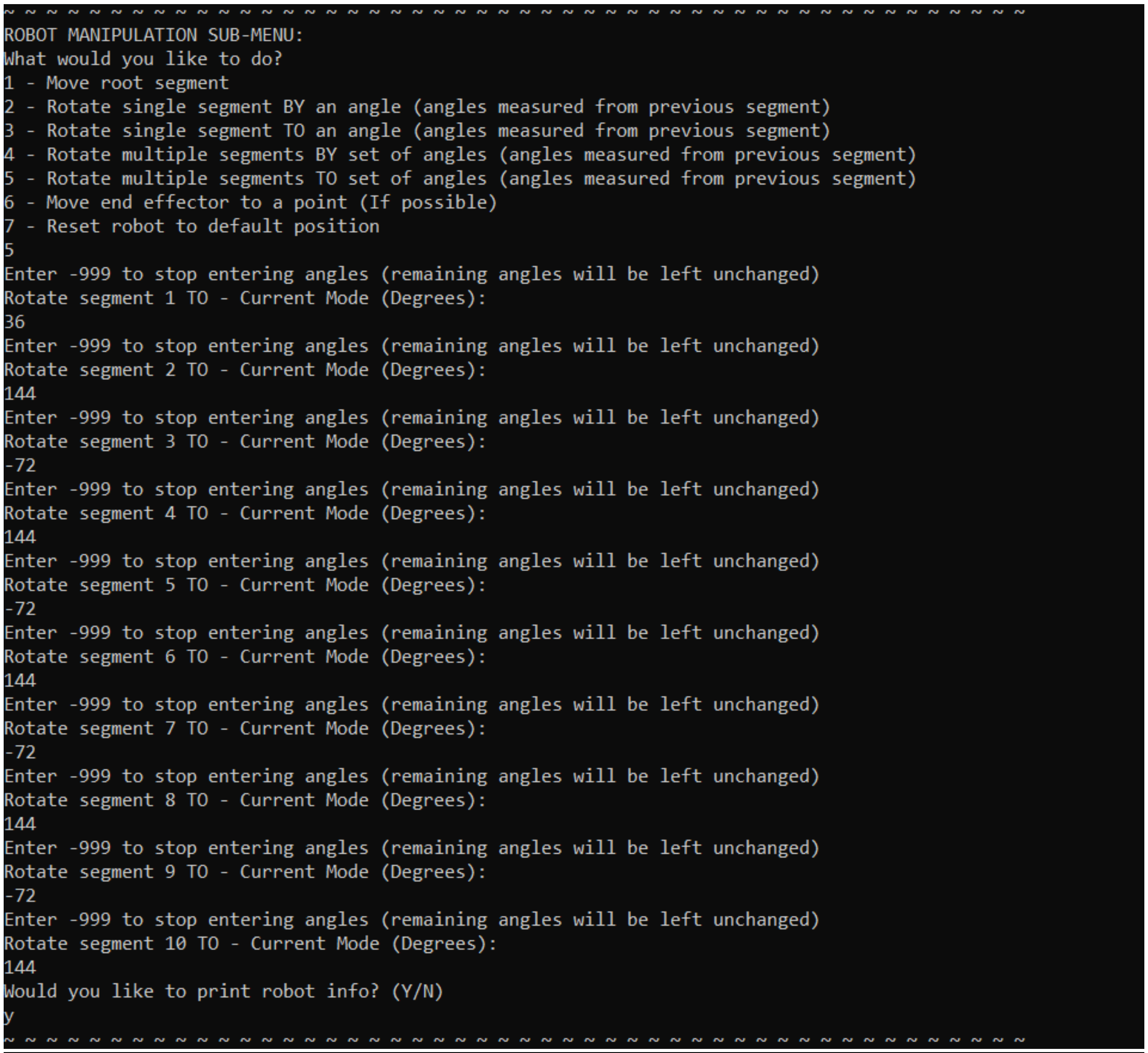
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MANIPULATION AND DISPLAY MENU:
Which robot would you like to work with? (Max robot ID 3):
1
What would you like to do with robot 1?
1 - Manipulate current postition
2 - Display robot info
1
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
6
Enter x coordinate of desired point:
10
Enter y coordinate of desired point:
20
Failed to reach the point . . .
Robot returned to default position. . .
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 1:
Segment 1: Start: (4.000,5.000); End: (9.000,5.000); Length: 5.000; Angle above x-axis: 0.000 deg
Segment 2: Start: (9.000,5.000); End: (12.000,5.000); Length: 3.000; Angle above x-axis: 0.000 deg
Segment 3: Start: (12.000,5.000); End: (13.000,5.000); Length: 1.000; Angle above x-axis: 0.000 deg
Segment 4: Start: (13.000,5.000); End: (15.000,5.000); Length: 2.000; Angle above x-axis: 0.000 deg
Segment 5: Start: (15.000,5.000); End: (19.000,5.000); Length: 4.000; Angle above x-axis: 0.000 deg
Would you like to display robot 1's information graphically as well? (Y/N)
```

Everything worked as expected => This task passes

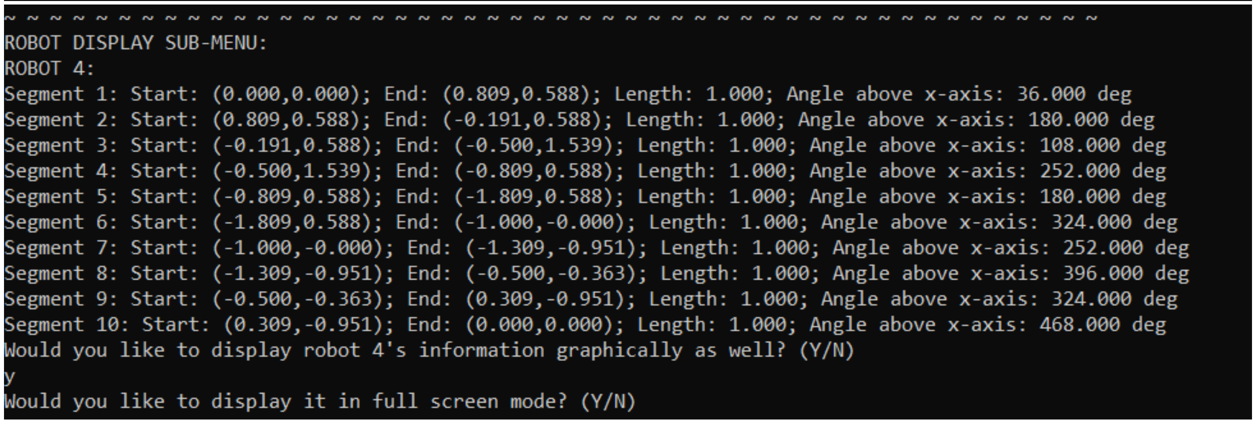
Task 6 – Displaying Robot Information:

While the previous test cases show that displaying the robot works, here are a few interesting looking robots just for the fun of it:

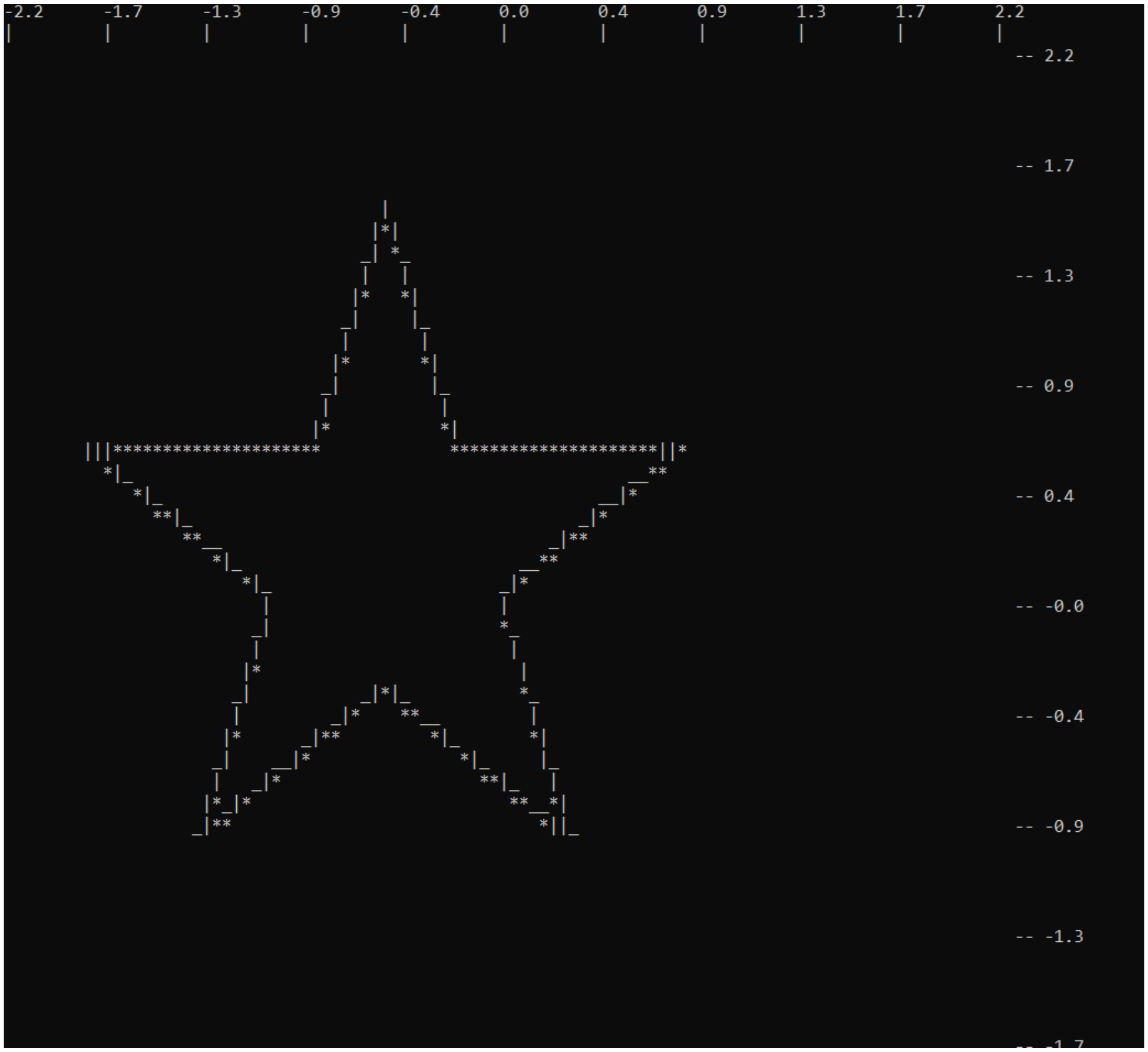
Stars:



```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
5
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 1 TO - Current Mode (Degrees):
36
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 2 TO - Current Mode (Degrees):
144
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 3 TO - Current Mode (Degrees):
-72
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 4 TO - Current Mode (Degrees):
144
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 5 TO - Current Mode (Degrees):
-72
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 6 TO - Current Mode (Degrees):
144
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 7 TO - Current Mode (Degrees):
-72
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 8 TO - Current Mode (Degrees):
144
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 9 TO - Current Mode (Degrees):
-72
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 10 TO - Current Mode (Degrees):
144
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
```

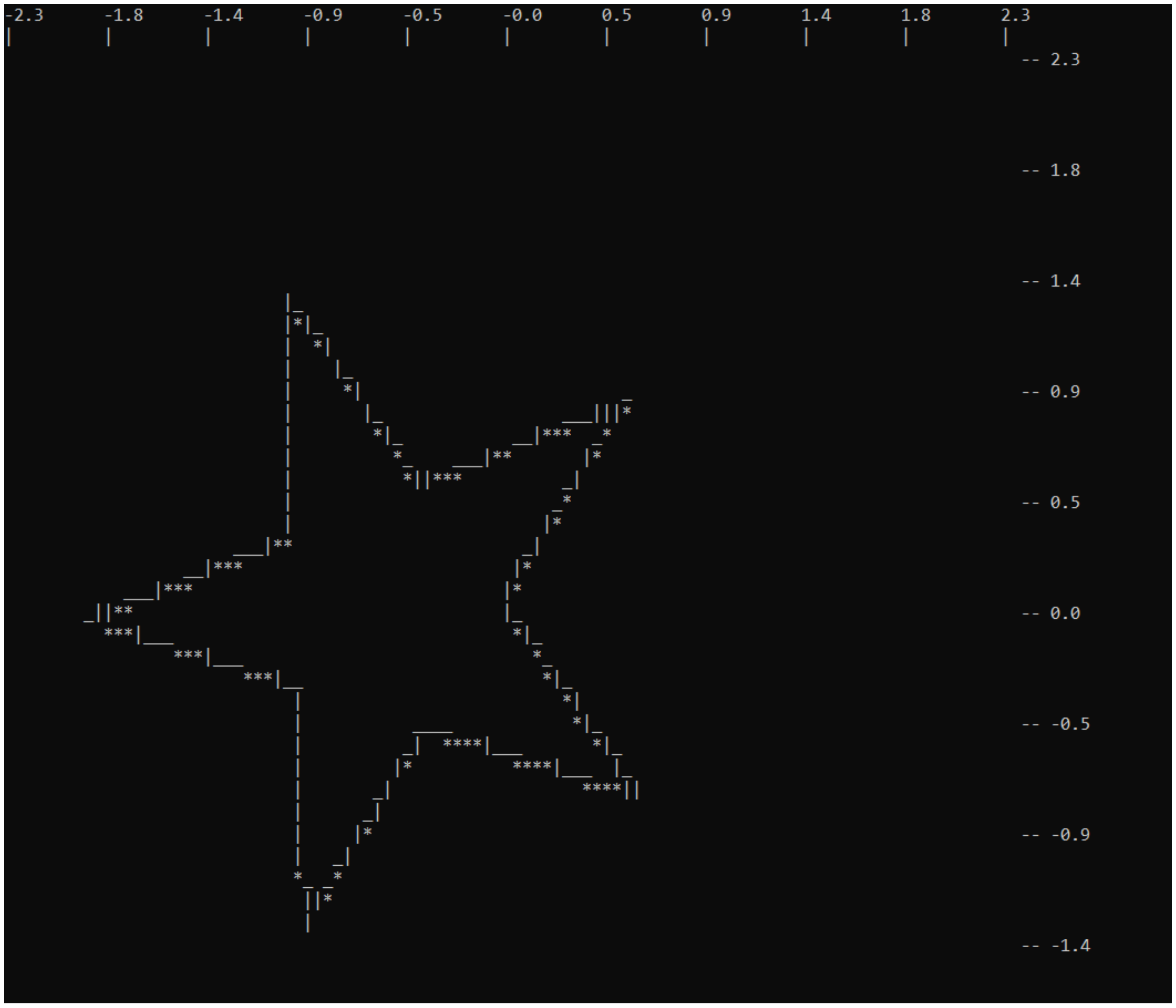


```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 4:
Segment 1: Start: (0.000,0.000); End: (0.809,0.588); Length: 1.000; Angle above x-axis: 36.000 deg
Segment 2: Start: (0.809,0.588); End: (-0.191,0.588); Length: 1.000; Angle above x-axis: 180.000 deg
Segment 3: Start: (-0.191,0.588); End: (-0.500,1.539); Length: 1.000; Angle above x-axis: 108.000 deg
Segment 4: Start: (-0.500,1.539); End: (-0.809,0.588); Length: 1.000; Angle above x-axis: 252.000 deg
Segment 5: Start: (-0.809,0.588); End: (-1.809,0.588); Length: 1.000; Angle above x-axis: 180.000 deg
Segment 6: Start: (-1.809,0.588); End: (-1.000,-0.000); Length: 1.000; Angle above x-axis: 324.000 deg
Segment 7: Start: (-1.000,-0.000); End: (-1.309,-0.951); Length: 1.000; Angle above x-axis: 252.000 deg
Segment 8: Start: (-1.309,-0.951); End: (-0.500,-0.363); Length: 1.000; Angle above x-axis: 396.000 deg
Segment 9: Start: (-0.500,-0.363); End: (0.309,-0.951); Length: 1.000; Angle above x-axis: 324.000 deg
Segment 10: Start: (0.309,-0.951); End: (0.000,0.000); Length: 1.000; Angle above x-axis: 468.000 deg
Would you like to display robot 4's information graphically as well? (Y/N)
y
Would you like to display it in full screen mode? (Y/N)
```

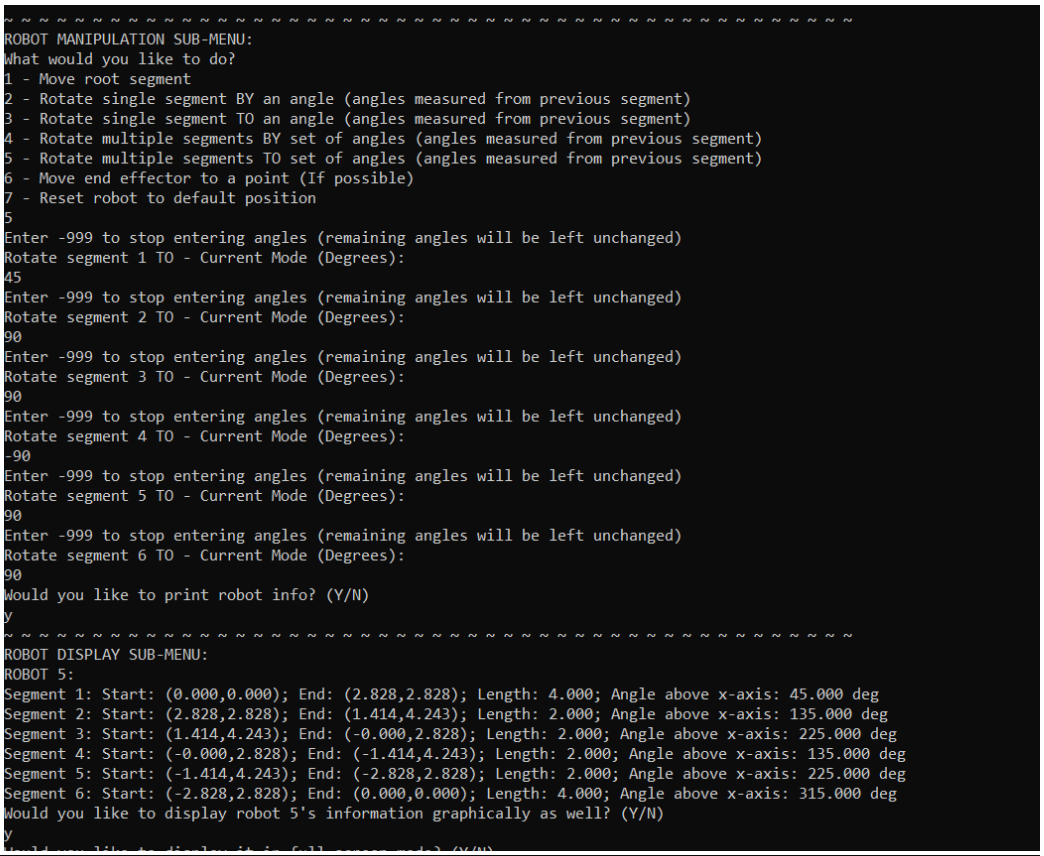


```
-2.2      -1.7      -1.3      -0.9      -0.4      0.0       0.4       0.9       1.3       1.7       2.2
|         |         |         |         |         |         |         |         |         |         |
                                                                                                      -- 2.2




                                                                                                      -- 1.7

                                      |
                                     |*|
                                    _| *_
                                    |   |                                                             -- 1.3
                                   |*   *|
                                  _|     |_
                                  |       |
                                 |*       *|
                                _|         |_                                                         -- 0.9
                                |           |
                               |*           *|
        |||*********************             *********************||*
          *|_                                                  __**
             *|_                                            __|*                                      -- 0.4
               **|_                                       _|*
                  **__                                 _|**
                     *|_                            __**
                        *|_                       _|*
                          |                       |                                                   -- -0.0
                         _|                       *_
                         |                         |
                        |*                          |
                       _|           _|*|_           *_
                       |         _|*    **__         |                                                -- -0.4
                      |*      _|**         *|_       *|
                     _|    __|*               *|_     |_
                     |   _|*                    **|_   |
                    |*_|*                          **__*|
                   _|**                               *||_                                            -- -0.9




                                                                                                      -- -1.3




                                                                                                      -- -1.7
```

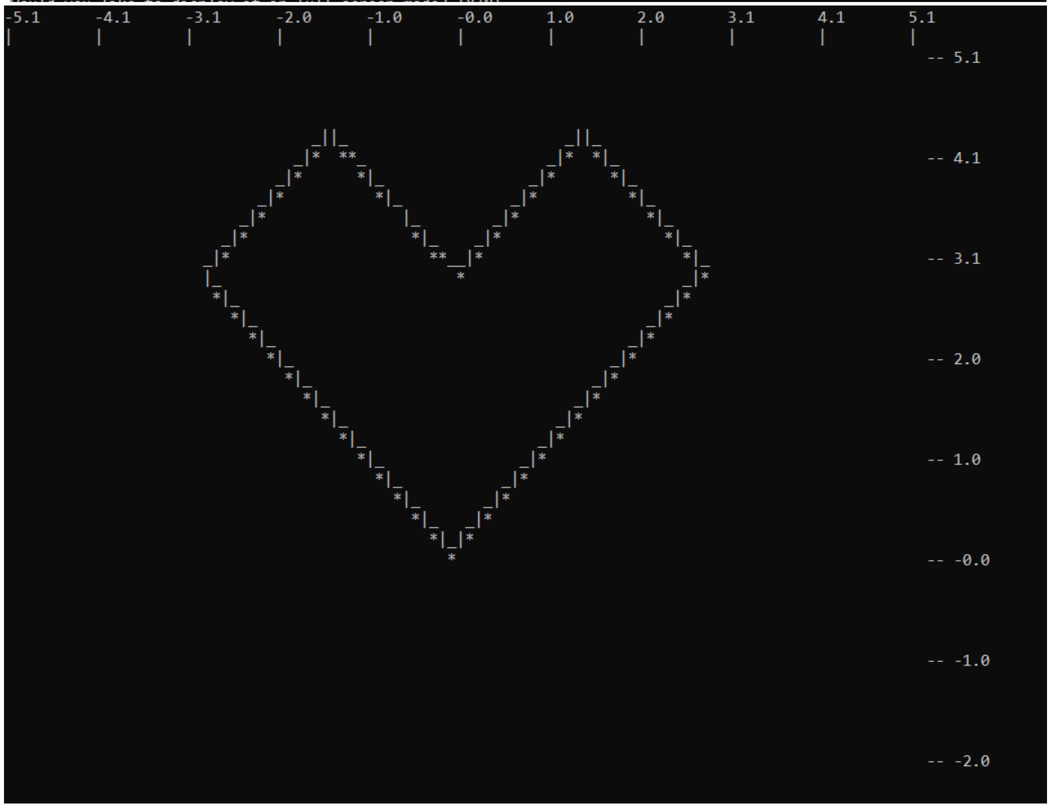
Rotating the first segment:



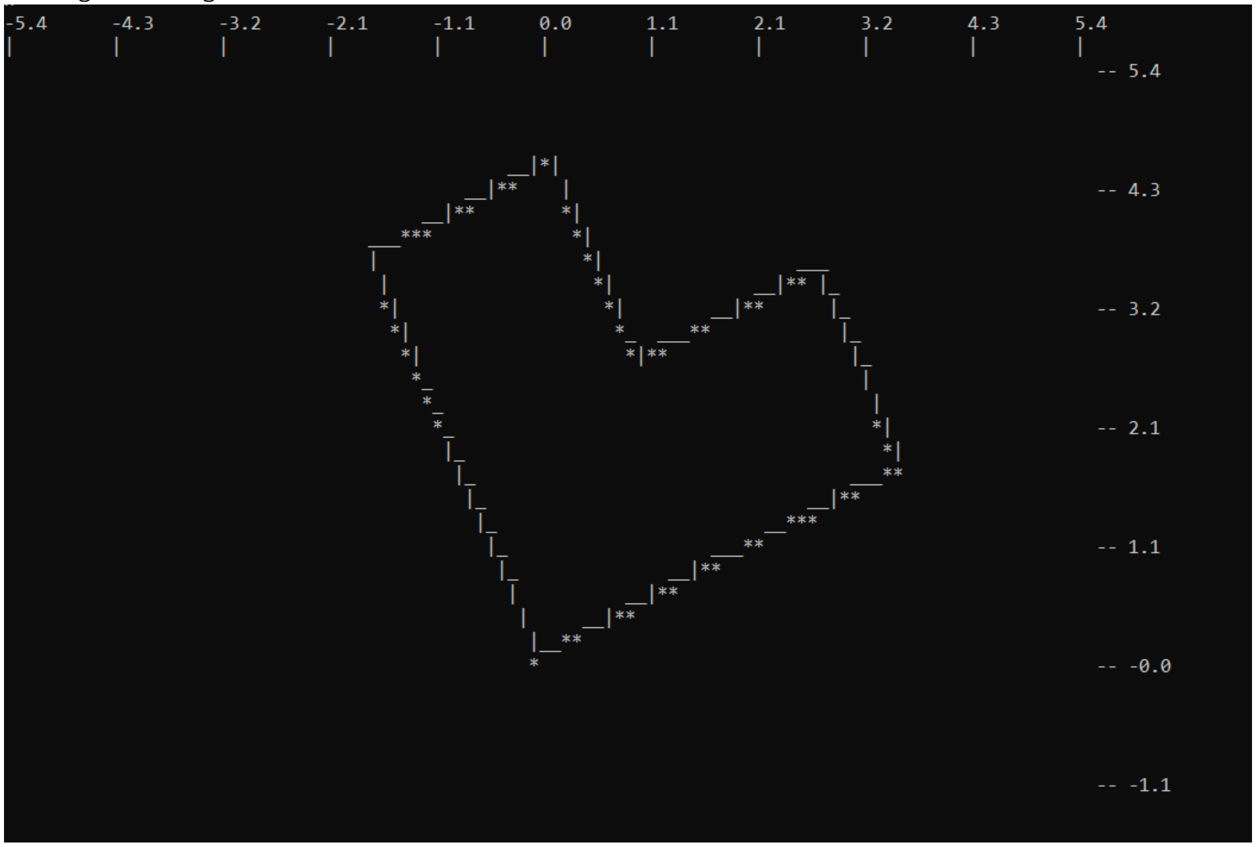
Hearts:



```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
5
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 1 TO - Current Mode (Degrees):
45
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 2 TO - Current Mode (Degrees):
90
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 3 TO - Current Mode (Degrees):
90
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 4 TO - Current Mode (Degrees):
-90
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 5 TO - Current Mode (Degrees):
90
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 6 TO - Current Mode (Degrees):
90
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 5:
Segment 1: Start: (0.000,0.000); End: (2.828,2.828); Length: 4.000; Angle above x-axis: 45.000 deg
Segment 2: Start: (2.828,2.828); End: (1.414,4.243); Length: 2.000; Angle above x-axis: 135.000 deg
Segment 3: Start: (1.414,4.243); End: (-0.000,2.828); Length: 2.000; Angle above x-axis: 225.000 deg
Segment 4: Start: (-0.000,2.828); End: (-1.414,4.243); Length: 2.000; Angle above x-axis: 135.000 deg
Segment 5: Start: (-1.414,4.243); End: (-2.828,2.828); Length: 2.000; Angle above x-axis: 225.000 deg
Segment 6: Start: (-2.828,2.828); End: (0.000,0.000); Length: 4.000; Angle above x-axis: 315.000 deg
Would you like to display robot 5's information graphically as well? (Y/N)
y
```

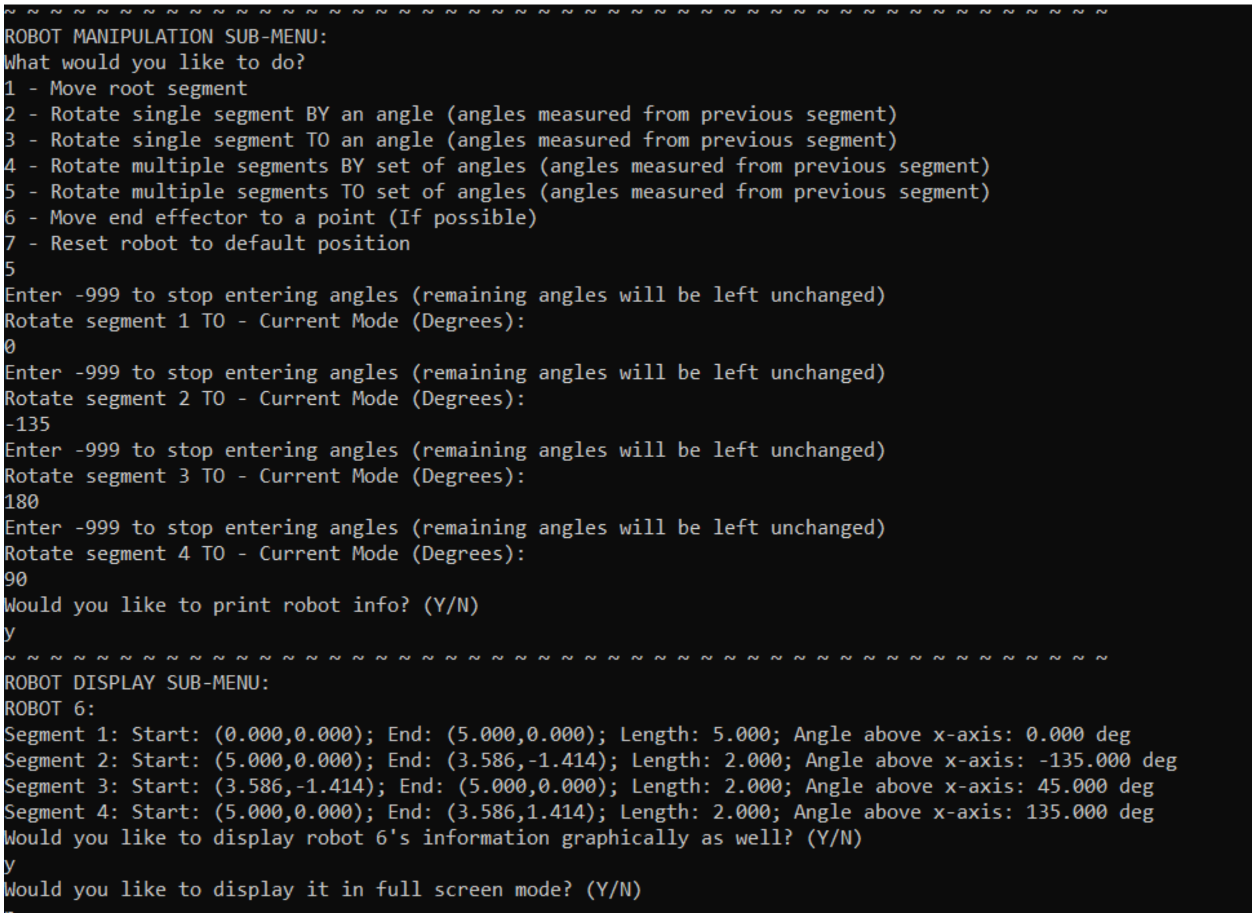


Rotating the first segment:

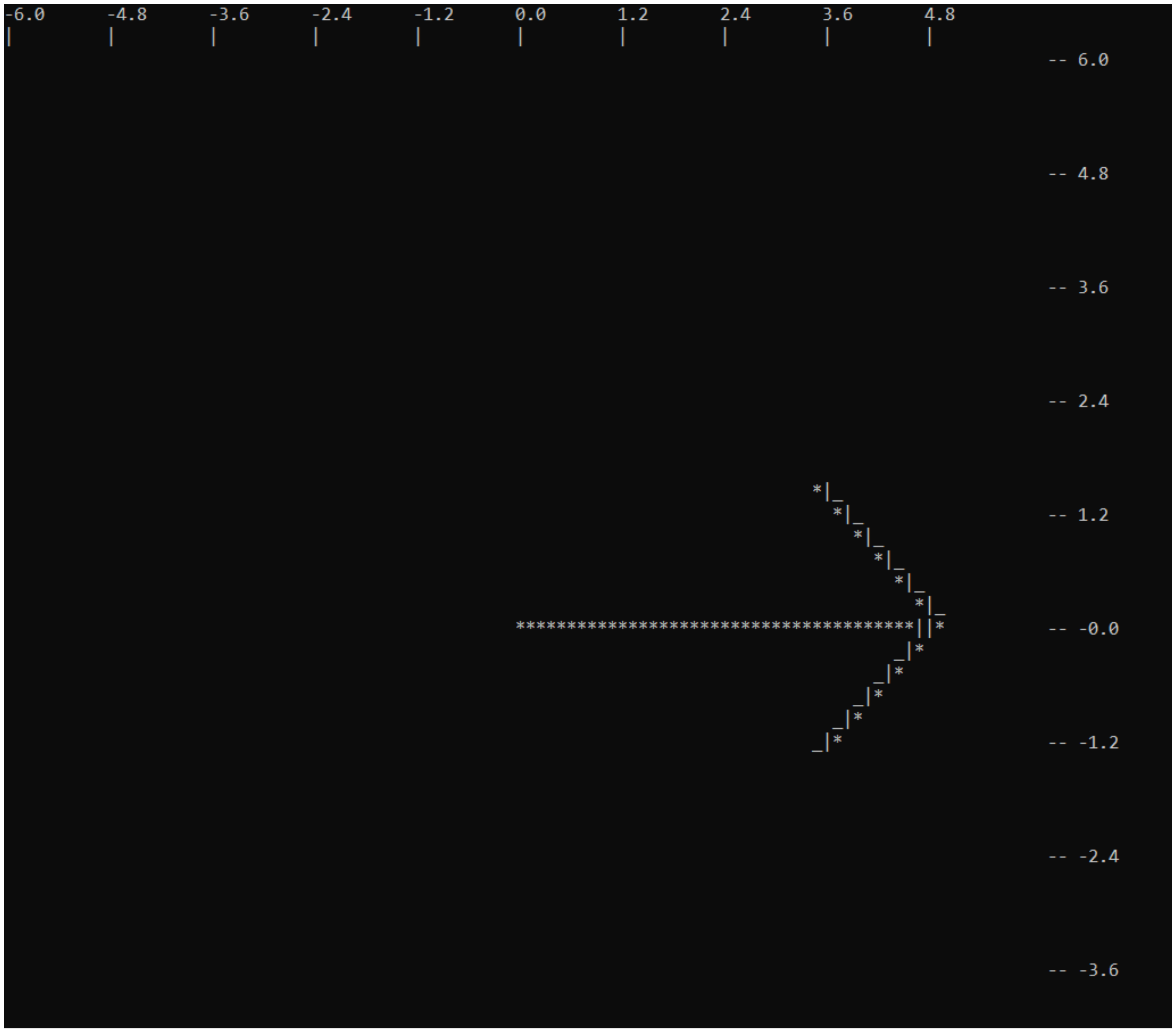


```
-5.4      -4.3      -3.2      -2.1      -1.1      0.0       1.1       2.1       3.2       4.3       5.4
|         |         |         |         |         |         |         |         |         |         |
                                                                                                      -- 5.4



                                               __|*|
                                           __|**    |                                                 -- 4.3
                                       __|**        *|
                                  ___***             *|
                                  |                   *|                  ___
                                   |                   *|             __|** |_
                                   *|                   *|        __|**      |_                       -- 3.2
                                    *|                   *_  ___**            |_
                                     *|                   *|**                 |_
                                      *_                                        |
                                       *_                                        |
                                        *_                                       *|                   -- 2.1
                                         |_                                       *|
                                          |_                                   ___**
                                           |_                              __|**
                                            |_                         __***
                                             |_                   ___**                               -- 1.1
                                              |_              __|**
                                               |          __|**
                                                |     __|**
                                                 |__**
                                                 *                                                    -- -0.0




                                                                                                      -- -1.1
```

Arrows:

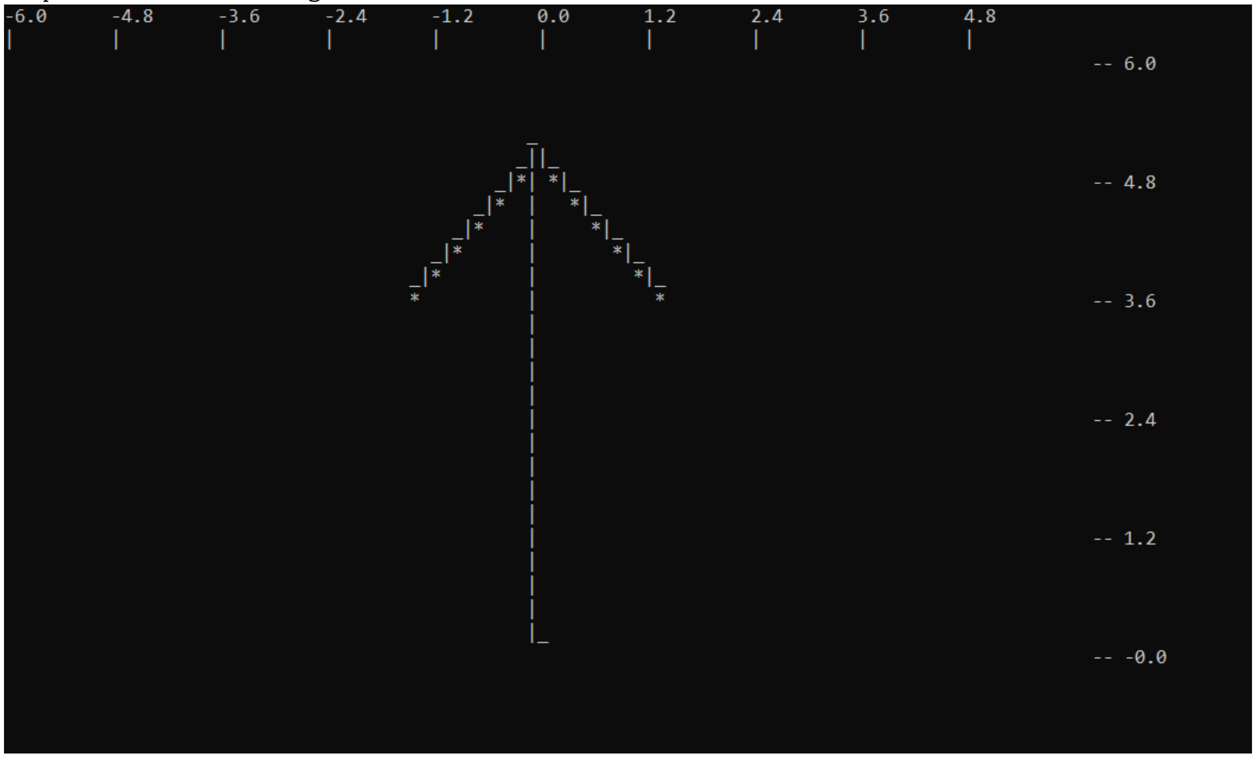


```
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT MANIPULATION SUB-MENU:
What would you like to do?
1 - Move root segment
2 - Rotate single segment BY an angle (angles measured from previous segment)
3 - Rotate single segment TO an angle (angles measured from previous segment)
4 - Rotate multiple segments BY set of angles (angles measured from previous segment)
5 - Rotate multiple segments TO set of angles (angles measured from previous segment)
6 - Move end effector to a point (If possible)
7 - Reset robot to default position
5
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 1 TO - Current Mode (Degrees):
0
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 2 TO - Current Mode (Degrees):
-135
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 3 TO - Current Mode (Degrees):
180
Enter -999 to stop entering angles (remaining angles will be left unchanged)
Rotate segment 4 TO - Current Mode (Degrees):
90
Would you like to print robot info? (Y/N)
y
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
ROBOT DISPLAY SUB-MENU:
ROBOT 6:
Segment 1: Start: (0.000,0.000); End: (5.000,0.000); Length: 5.000; Angle above x-axis: 0.000 deg
Segment 2: Start: (5.000,0.000); End: (3.586,-1.414); Length: 2.000; Angle above x-axis: -135.000 deg
Segment 3: Start: (3.586,-1.414); End: (5.000,0.000); Length: 2.000; Angle above x-axis: 45.000 deg
Segment 4: Start: (5.000,0.000); End: (3.586,1.414); Length: 2.000; Angle above x-axis: 135.000 deg
Would you like to display robot 6's information graphically as well? (Y/N)
y
Would you like to display it in full screen mode? (Y/N)
```

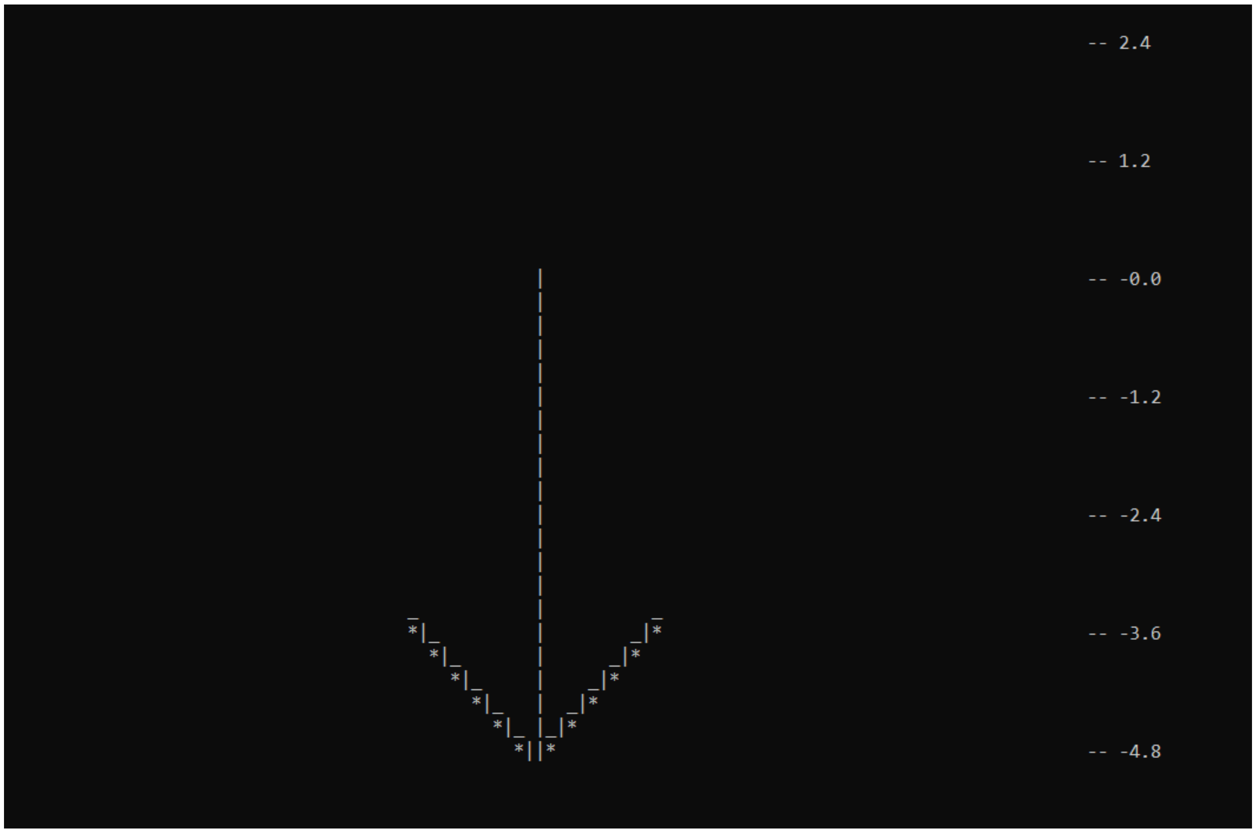


```
-6.0      -4.8      -3.6      -2.4      -1.2      0.0       1.2       2.4       3.6       4.8
|         |         |         |         |         |         |         |         |         |
                                                                                                      -- 6.0




                                                                                                      -- 4.8




                                                                                                      -- 3.6




                                                                                                      -- 2.4



                                                                               *|_
                                                                                 *|_                  -- 1.2
                                                                                   *|_
                                                                                     *|_
                                                                                       *|_
                                                                                         *|_
                                                  ***************************************||*          -- -0.0
                                                                                       _|*
                                                                                     _|*
                                                                                   _|*
                                                                                 _|*
                                                                               _|*                    -- -1.2




                                                                                                      -- -2.4




                                                                                                      -- -3.6
```

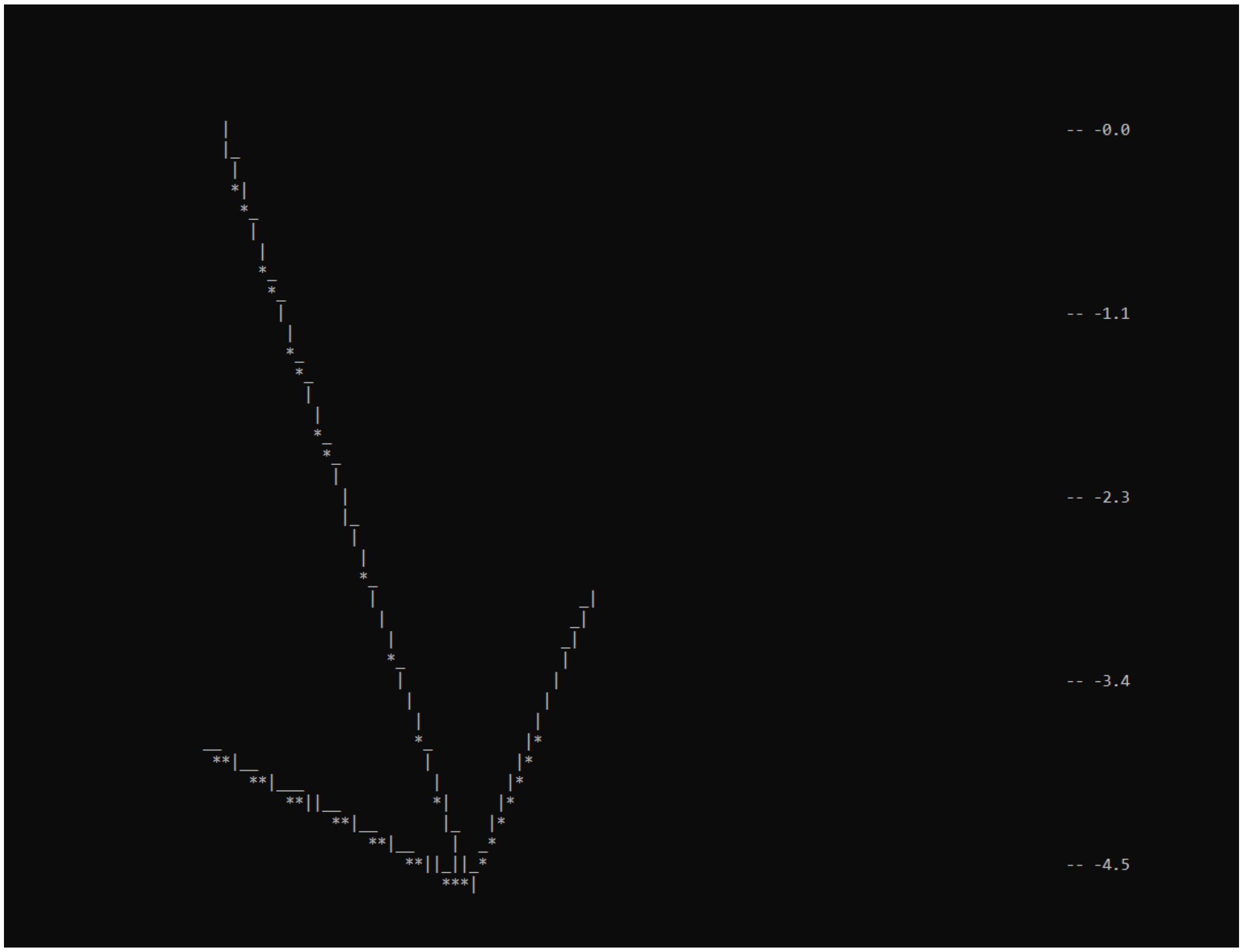
Simple rotations of the first segment:



```
-6.0        -4.8        -3.6        -2.4        -1.2        0.0         1.2         2.4         3.6         4.8
|           |           |           |           |           |           |           |           |           |
                                                                                                                          -- 6.0


                                                           _
                                                          _||_
                                                        _|*| *|_                                                          -- 4.8
                                                      _|*  |   *|_
                                                    _|*    |     *|_
                                                  _|*      |       *|_
                                                _|*        |         *|_
                                                *          |           *                                                  -- 3.6
                                                           |
                                                           |
                                                           |
                                                           |
                                                           |                                                              -- 2.4
                                                           |
                                                           |
                                                           |
                                                           |
                                                           |                                                              -- 1.2
                                                           |
                                                           |
                                                           |
                                                           |_
                                                                                                                          -- -0.0
```



```
                                                                                                                          -- 2.4




                                                                                                                          -- 1.2




                                                            |                                                             -- -0.0
                                                            |
                                                            |
                                                            |
                                                            |
                                                            |                                                             -- -1.2
                                                            |
                                                            |
                                                            |
                                                            |
                                                            |                                                             -- -2.4
                                                            |
                                                            |
                                                            |
                                                _           |          _
                                                *|_         |        _|*                                                  -- -3.6
                                                  *|_       |      _|*
                                                    *|_     |    _|*
                                                      *|_   |  _|*
                                                        *|_ |_|*
                                                          *||*                                                            -- -4.8
```



User Guide:

- Make sure all the header files (DhiyaaGraphics.h, Point.h, Segment.h, Robot.h) are in the same folder as Assignment4.cpp
- Compile and run Assignment4.cpp
- The program will present you with a clear set of options.
- When you're done with the program, make sure to quit by choosing exit (option 5 in the main menu).