BINARY SOLUTIONS

MAT 275
Radius Calculation on Polar Coordinate System in C++

contact@thebinarysolutionsllc.com

Introduction

This program calculates the radius corresponding to different angles on a polar coordinate system. It demonstrates the usage of trigonometric functions in C++ to convert polar coordinates to Cartesian coordinates and then calculates the radius from these coordinates.

Problem Statement

Given a set of angles (θ) and an initial radius value (r), the program computes the radius corresponding to each angle on the polar coordinate system.

Solution Steps

- Define a vector to store the angles (θ).
- Initialize the initial radius value (r).
- Define vectors to store the x and y coordinates.
- Calculate the x and y coordinates for each angle using the trigonometric functions cosine and sine.
- Compute the radius from the x and y coordinates using the Pythagorean theorem.
- Display the calculated radius values.

Pseudo Code

- Include necessary header files (iostream, cmath, vector).
- Begin main function.
  - Define angles:
    - Create a vector called 'theta' to store angles in radians, initialized with specific values: 0, π/4, π/2, 3π/4, π, and 5π/4.
  - Define initial radius:
    - Declare and initialize a variable 'r' of type double with the initial radius value (2 in this case).
  - Define vectors to store x and y coordinates:
    - Create two vectors 'x' and 'y' of type double to store x and y coordinates, respectively, with sizes equal to the size of the 'theta' vector.
  - Calculate x and y coordinates:
    - For each angle 'theta[i]' in the 'theta' vector:
      - Calculate x coordinate: x[i] = r * cos(theta[i])
      - Calculate y coordinate: y[i] = r * sin(theta[i])

Pseudo Code

- Calculate radius from x and y coordinates:
  - Create a vector 'radius' of type double to store the calculated radius values.
  - For each pair of x and y coordinates (x[i], y[i]):
    - Calculate radius: radius[i] = sqrt(x[i]^2 + y[i]^2)
- Display radius:
  - Output the label "radius:" to the console.
  - For each radius value in the 'radius' vector:
    - Output the radius value followed by a space.
  - Output a newline character to move to the next line.
- End main function.

C++ Code

```
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;
int main() {
vector<double> theta = {0, M_PI / 4, M_PI / 2, 3 * M_PI / 4, M_PI, 5 * M_PI / 4};
double r = 2;
vector<double> x(theta.size());
   vector<double> y(theta.size());
for (size_t i = 0; i < theta.size(); ++i) {
      x[i] = r * cos(theta[i]);
      y[i] = r * sin(theta[i]);
   }
vector<double> radius(theta.size());
   for (size_t i = 0; i < theta.size(); ++i) {
      radius[i] = sqrt(pow(x[i], 2) + pow(y[i], 2));
   }
   // Display radius
   cout << "radius:" << endl;
   for (size_t i = 0; i < radius.size(); ++i) {
      cout << radius[i] << " ";
   }
   cout << endl;
   return 0;
}
```

contact@thebinarysolutionsllc.com

Code Explanation

- **#include <iostream>#include <cmath>#include <vector>using namespace std;**

These lines include the necessary header files: ‘<iostream>’ for input/output stream functionality, ‘<cmath>’ for mathematical functions, and ‘<vector>’ for using vectors in C++.

- **int main() {**

This line marks the beginning of the ‘main’ function, which serves as the entry point of the program

- **vector<double> theta = {0, M_PI / 4, M_PI / 2, 3 * M_PI / 4, M_PI, 5 * M_PI / 4};**

This line defines a vector ‘theta’ of double-precision floating-point numbers and initializes it with a series of angles in radians.

- **double r = 2;**

This line defines a double-precision floating-point variable ‘r’ and initializes it with the initial value of the radius.

- **vector<double> x(theta.size()); vector<double> y(theta.size());**

These lines define two vectors ‘x’ and ‘y’ of double-precision floating-point numbers to store the x and y coordinates, respectively.

contact@thebinarysolutionsllc.com

Code Explanation

- **for (size_t i = 0; i < theta.size(); ++i) { x[i] = r * cos(theta[i]); y[i] = r * sin(theta[i]); }**

This loop calculates the ‘x’ and ‘y’ coordinates for each angle using the trigonometric functions ‘cos’ and ‘sin’, respectively, and stores them in the x and y vectors.

- **vector<double> radius(theta.size()); for (size_t i = 0; i < theta.size(); ++i) { radius[i] = sqrt(pow(x[i], 2) + pow(y[i], 2)); }**

This loop calculates the radius for each pair of x and y coordinates using the Euclidean distance formula and stores the result in the ‘radius’ vector.

- **cout << "radius:" << endl; for (size_t i = 0; i < radius.size(); ++i) { cout << radius[i] << " "; } cout << endl;**

This loop outputs the calculated radius values to the standard output (typically the console), separated by spaces.

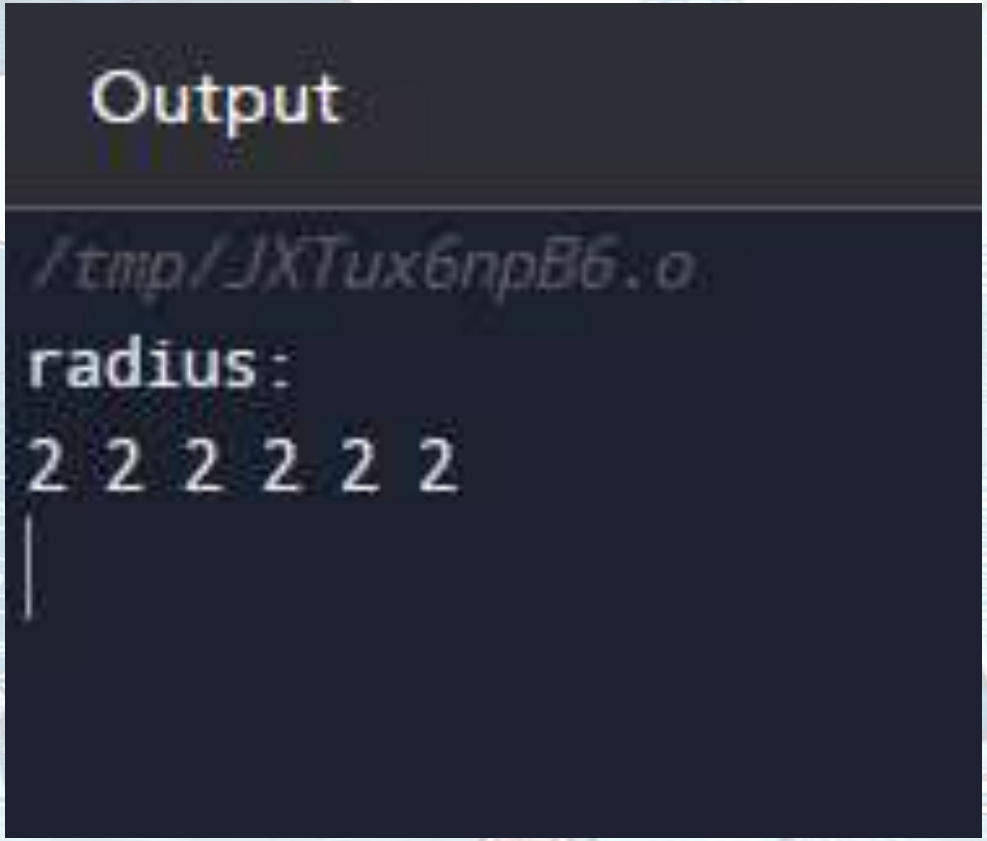
- **return 0;}**

This line indicates the end of the ‘main’ function and returns an integer value of ‘0’ to the operating system, typically indicating successful execution.

contact@thebinarysolutionsllc.com

Final Answer

The final output is the radius vector containing the calculated ‘radius’ values from Cartesian coordinates.



Additional Comments/Tips

- Ensure the correctness of the provided angles and initial radius value to obtain accurate results.
- Consider handling edge cases, such as angles covering a full circle or negative radius values.

Conclusion

This program showcases the transformation of polar coordinates to Cartesian coordinates and the subsequent calculation of the radius on a polar coordinate system. Understanding these computations is essential in various fields, including physics, engineering, and computer graphics.