Table of Contents

Progress bar program test oracle **2**

Summary 2

System environment 2

Output after running test_driver 3

CFG for progress bar program 4

Initial five test cases 5

CFG for progress bar program w/ initial path selection 6

Progress bar program path selection analysis **7**

Test cases with complete statement and branch coverage 7

Tower of Hanoi program test oracle **8**

Summary 8

System environment 8

Output after running tower_of_hanoi.c 9

CFG for tower of hanoi program 10

Program code **11**

progress_bar.h 11

test_driver.c 13

tower_of_hanoi.c 16

Progress bar program test oracle

Summary

This program is written in C. The code includes a header file `progress_bar.h` and a test driver file `test_driver.c`. The program contains 8 statements, 14 branches, and 10 paths. The test driver executes 14 test cases exhausting all 10 of the paths and repeating path 3 once and path 2 three times to check for potential logical errors.

System environment

Programming language: C

Laptop: Asus N550JK

Processor: Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz

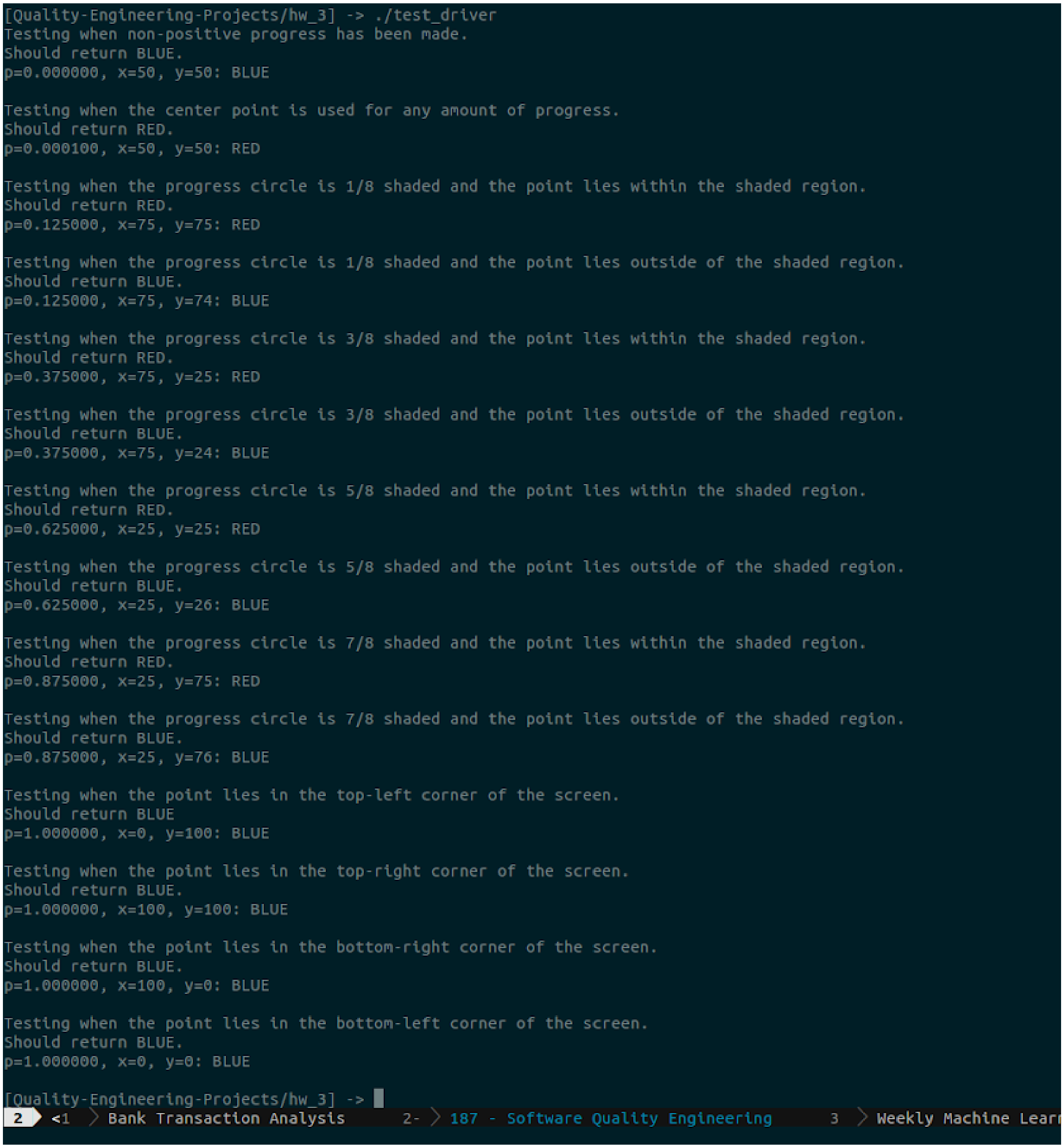
Processor speed: 3.4 GHz

Operating system: Ubuntu 14.04

Testing environment: GNU bash, version 4.3.11(1)-release (x86_64-pc-linux-gnu)

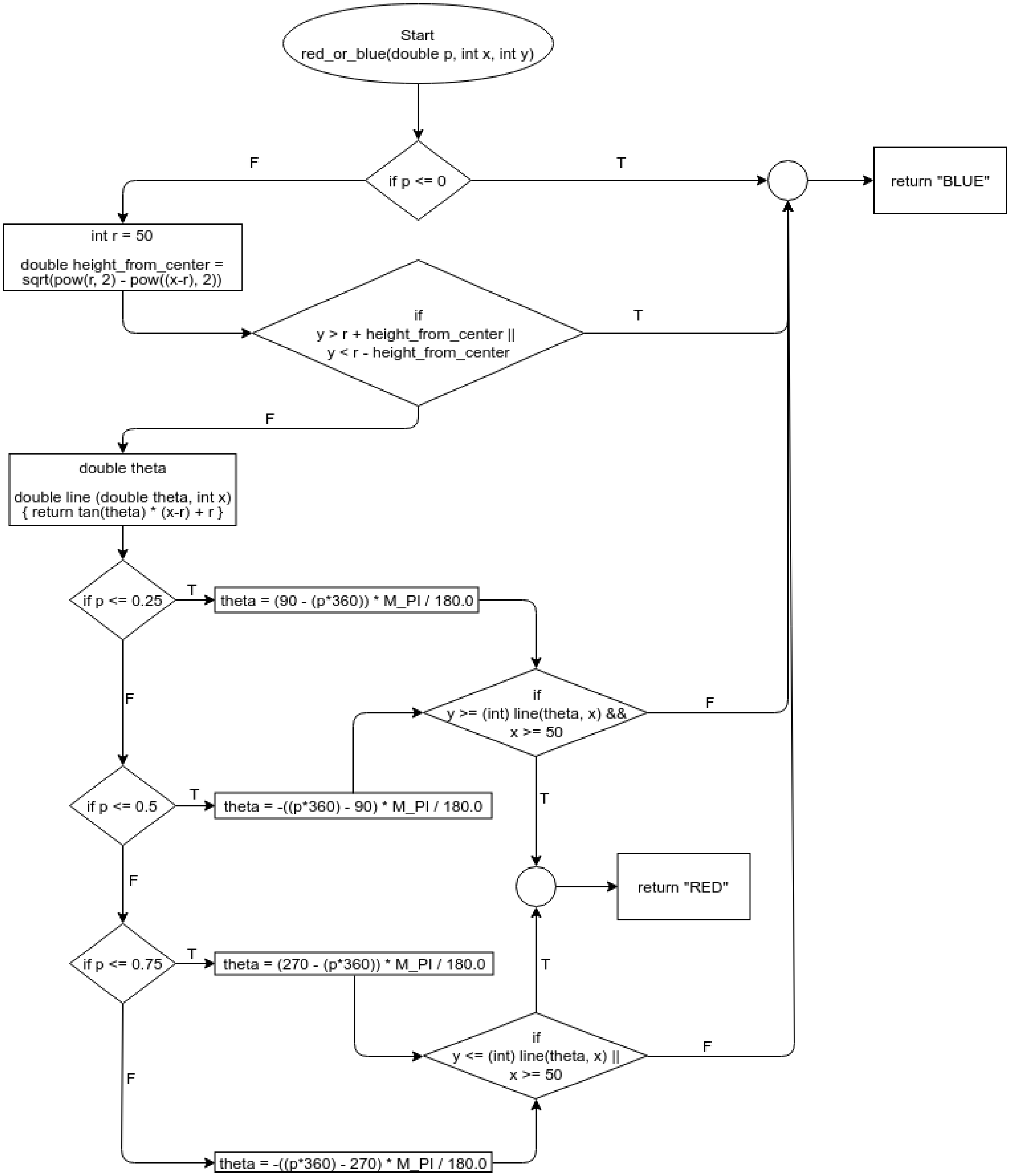
Text editor: Vim

Output after running test_driver



```
[Quality-Engineering-Projects/hw_3] -> ./test_driver
Testing when non-positive progress has been made.
Should return BLUE.
p=0.000000, x=50, y=50: BLUE

Testing when the center point is used for any amount of progress.
Should return RED.
p=0.000100, x=50, y=50: RED

Testing when the progress circle is 1/8 shaded and the point lies within the shaded region.
Should return RED.
p=0.125000, x=75, y=75: RED

Testing when the progress circle is 1/8 shaded and the point lies outside of the shaded region.
Should return BLUE.
p=0.125000, x=75, y=74: BLUE

Testing when the progress circle is 3/8 shaded and the point lies within the shaded region.
Should return RED.
p=0.375000, x=75, y=25: RED

Testing when the progress circle is 3/8 shaded and the point lies outside of the shaded region.
Should return BLUE.
p=0.375000, x=75, y=24: BLUE

Testing when the progress circle is 5/8 shaded and the point lies within the shaded region.
Should return RED.
p=0.625000, x=25, y=25: RED

Testing when the progress circle is 5/8 shaded and the point lies outside of the shaded region.
Should return BLUE.
p=0.625000, x=25, y=26: BLUE

Testing when the progress circle is 7/8 shaded and the point lies within the shaded region.
Should return RED.
p=0.875000, x=25, y=75: RED

Testing when the progress circle is 7/8 shaded and the point lies outside of the shaded region.
Should return BLUE.
p=0.875000, x=25, y=76: BLUE

Testing when the point lies in the top-left corner of the screen.
Should return BLUE
p=1.000000, x=0, y=100: BLUE

Testing when the point lies in the top-right corner of the screen.
Should return BLUE.
p=1.000000, x=100, y=100: BLUE

Testing when the point lies in the bottom-right corner of the screen.
Should return BLUE.
p=1.000000, x=100, y=0: BLUE

Testing when the point lies in the bottom-left corner of the screen.
Should return BLUE.
p=1.000000, x=0, y=0: BLUE

[Quality-Engineering-Projects/hw_3] ->
 2  <1  Bank Transaction Analysis      2-  187 - Software Quality Engineering      3  Weekly Machine Lear
```

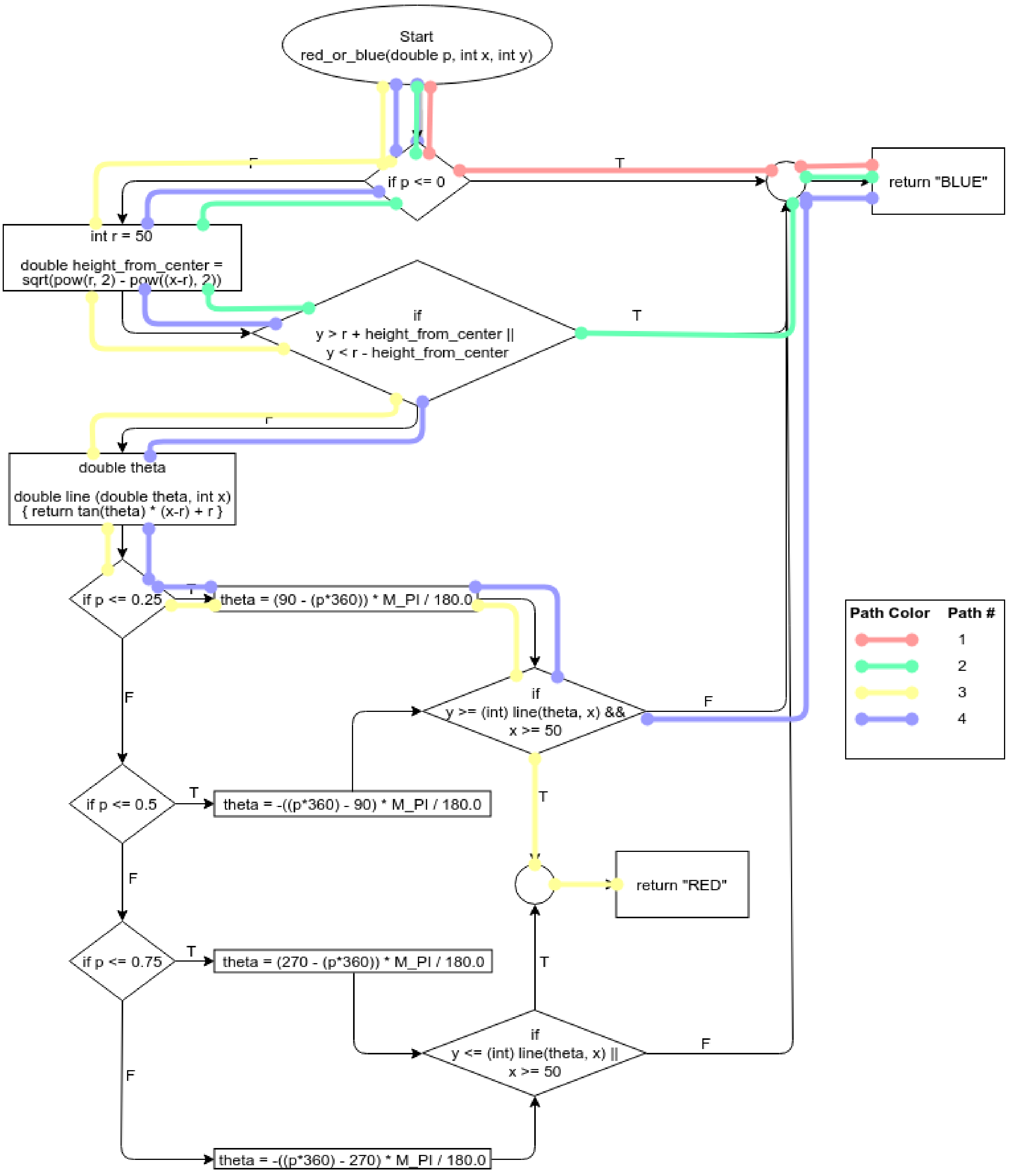
CFG for progress bar program


Start
red_or_blue(double p, int x, int y)
if p <= 0
F
T
return "BLUE"
int r = 50
double height_from_center =
sqrt(pow(r, 2) - pow((x-r), 2))
if
y > r + height_from_center ||
y < r - height_from_center
T
F
double theta
double line (double theta, int x)
{ return tan(theta) * (x-r) + r }
if p <= 0.25
T
theta = (90 - (p*360)) * M_PI / 180.0
F
if
y >= (int) line(theta, x) &&
x >= 50
F
if p <= 0.5
T
theta = -((p*360) - 90) * M_PI / 180.0
T
F
return "RED"
if p <= 0.75
T
theta = (270 - (p*360)) * M_PI / 180.0
T
if
y <= (int) line(theta, x) ||
x >= 50
F
F
theta = -((p*360) - 270) * M_PI / 180.0

Initial five test cases

Inputs				Coverage			Reason
p	**x**	**y**	**Expected Output**	**Statement # (8)**	**Branch # (14)**	**Path # (10)**	
0	50	50	BLUE	7	1	1	No progress has been made
1	0	100	BLUE	1,7	2, 3	2	Point lies in corner of screen
0.0001	50	50	RED	1, 2, 3, 8	2, 4, 5, 11	3	Point is the center of the screen
0.125	75	75	RED	1, 2, 3, 8	2, 4, 5, 11	3	Point is inside the 1st quadrant boundary line
0.125	75	74	BLUE	1, 2, 3, 7	2, 4, 5, 12	4	Point outside the 1st quadrant boundary line
Coverage Percentage				0.625	0.5	0.4	

CFG for progress bar program w/ initial path selection


Start
red_or_blue(double p, int x, int y)
if p <= 0
T
F
return "BLUE"
int r = 50
double height_from_center =
sqrt(pow(r, 2) - pow((x-r), 2))
if
y > r + height_from_center ||
y < r - height_from_center
T
F
double theta
double line (double theta, int x)
{ return tan(theta) * (x-r) + r }
if p <= 0.25
T
theta = (90 - (p*360)) * M_PI / 180.0
F
if
y >= (int) line(theta, x) &&
x >= 50
F
T
if p <= 0.5
T
theta = -((p*360) - 90) * M_PI / 180.0
F
return "RED"
if p <= 0.75
T
theta = (270 - (p*360)) * M_PI / 180.0
T
if
y <= (int) line(theta, x) ||
x >= 50
F
F
theta = -((p*360) - 270) * M_PI / 180.0
Path Color
Path #
1
2
3
4

Progress bar program path selection analysis

Statement coverage:

- 4 paths are necessary to achieve complete statement coverage
- there are 8 paths that can be chosen from to select 4 such paths
- 2 possible selections of 4 paths from these 8 fail to test the “return RED” and “return BLUE” statements, respectively
- there are `{8 choose 4} - 2 = 68` path selections that result in complete statement coverage

Branch coverage:

- 6 paths are necessary to achieve complete branch coverage
- 2 of these paths must be path 1 and path 2 as they, together, exclusively test branches 1 and 3
- the remaining 4 paths can be selected from 8 test cases; it’s the same as when considering statement coverage except there are now 6 possible path selections that exclude one or two branches
- there are `{8 choose 4} - 6 = 64` path selections that result in complete branch coverage

Test cases with complete statement and branch coverage

Inputs				**Coverage**			**Reason**
p	**x**	**y**	**Expected Output**	**Statement # (8)**	**Branch # (14)**	**Path # (10)**	
0	50	50	BLUE	7	1	1	No progress has been made
1	0	100	BLUE	1,7	2, 3	2	Point lies in corner of screen
0.125	75	75	RED	1, 2, 3, 8	2, 4, 5, 11	3	Progress is <= ¼ and point is red
0.375	75	24	BLUE	1, 2, 4, 7	2, 4, 6, 7, 12	6	Progress is <= ½ and point is blue
0.625	25	25	RED	1, 2, 5, 8	2, 4, 6, 8, 9, 13	7	Progress is <= ¾ and point is red
0.875	25	76	BLUE	1, 2, 6, 7	2, 4, 6, 8, 10, 14	10	Progress is <= 1 and point is blue
Coverage Percentage				1	1	0.6	

Tower of Hanoi program test oracle

Summary

This program is written in C and the program solves the Tower-of-Hanoi Problem using Recursion. It contains a main method and a tower method. The tower method is used recursively to solve the problem for the sequence of moves involved in the Tower of Hanoi.

System environment

Programming language: C

Laptop: Macbook Pro

Processor: 2.8 GHz Intel Core i5

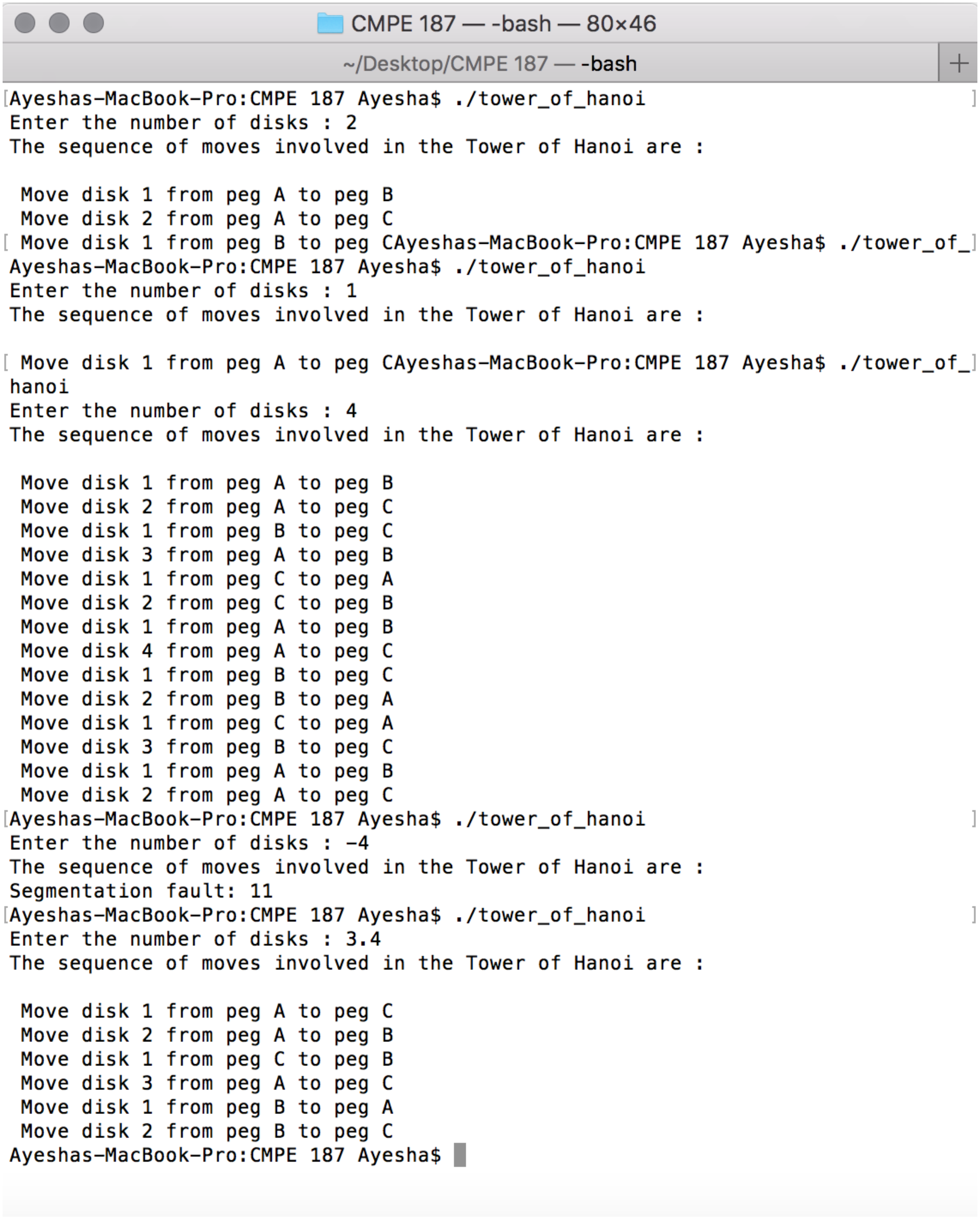
Processor speed: 2.8 GHz

Operating system: High Sierra

Testing environment: Mac OS Terminal

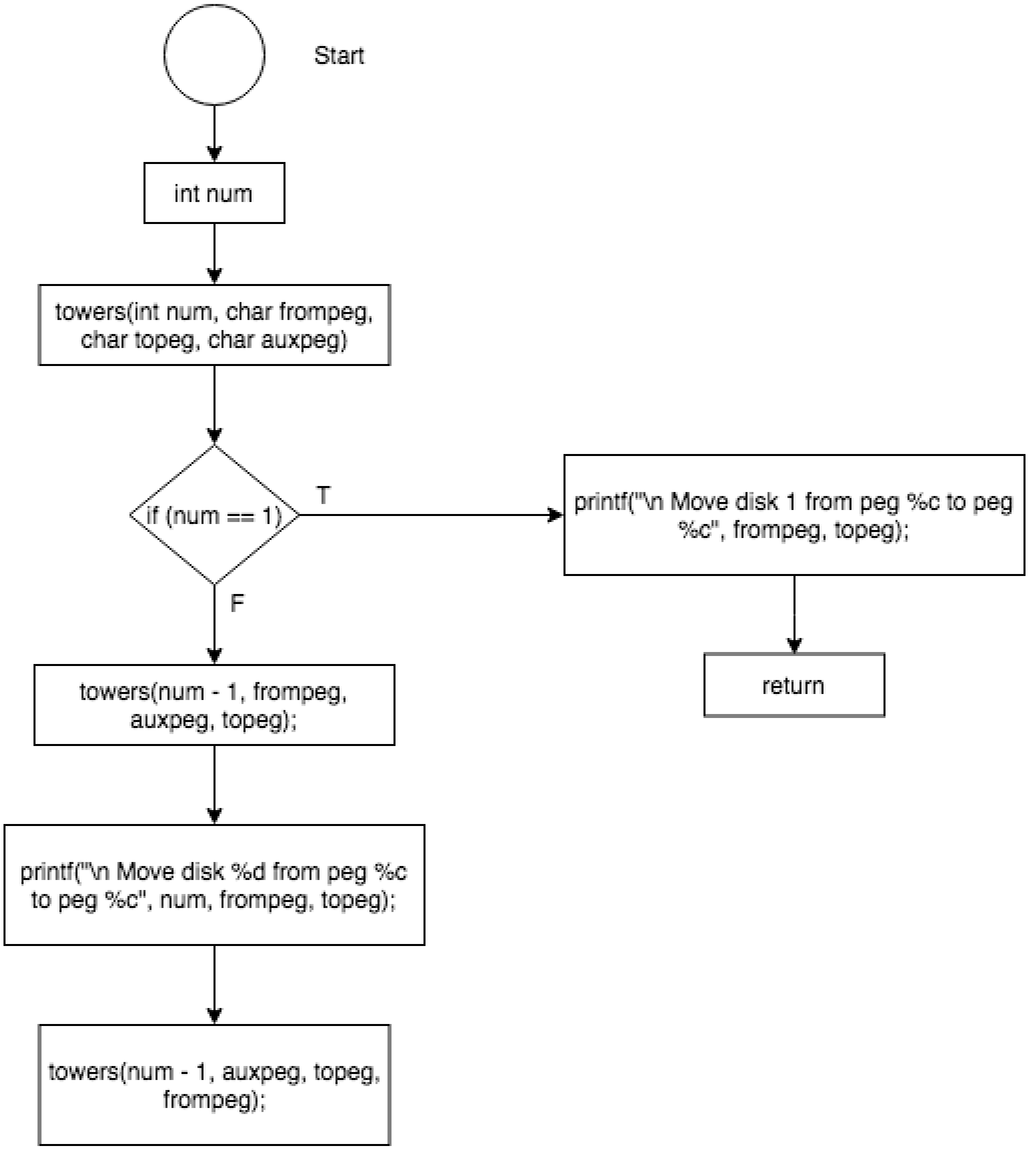
Text editor: Atom 3.0

Output after running tower_of_hanoi.c



```
[Ayeshas-MacBook-Pro:CMPE 187 Ayesha$ ./tower_of_hanoi                          ]
Enter the number of disks : 2
The sequence of moves involved in the Tower of Hanoi are :

 Move disk 1 from peg A to peg B
 Move disk 2 from peg A to peg C
[ Move disk 1 from peg B to peg CAyeshas-MacBook-Pro:CMPE 187 Ayesha$ ./tower_of_]
Ayeshas-MacBook-Pro:CMPE 187 Ayesha$ ./tower_of_hanoi
Enter the number of disks : 1
The sequence of moves involved in the Tower of Hanoi are :

[ Move disk 1 from peg A to peg CAyeshas-MacBook-Pro:CMPE 187 Ayesha$ ./tower_of_]
hanoi
Enter the number of disks : 4
The sequence of moves involved in the Tower of Hanoi are :

 Move disk 1 from peg A to peg B
 Move disk 2 from peg A to peg C
 Move disk 1 from peg B to peg C
 Move disk 3 from peg A to peg B
 Move disk 1 from peg C to peg A
 Move disk 2 from peg C to peg B
 Move disk 1 from peg A to peg B
 Move disk 4 from peg A to peg C
 Move disk 1 from peg B to peg C
 Move disk 2 from peg B to peg A
 Move disk 1 from peg C to peg A
 Move disk 3 from peg B to peg C
 Move disk 1 from peg A to peg B
 Move disk 2 from peg A to peg C
[Ayeshas-MacBook-Pro:CMPE 187 Ayesha$ ./tower_of_hanoi                          ]
Enter the number of disks : -4
The sequence of moves involved in the Tower of Hanoi are :
Segmentation fault: 11
[Ayeshas-MacBook-Pro:CMPE 187 Ayesha$ ./tower_of_hanoi                          ]
Enter the number of disks : 3.4
The sequence of moves involved in the Tower of Hanoi are :

 Move disk 1 from peg A to peg C
 Move disk 2 from peg A to peg B
 Move disk 1 from peg C to peg B
 Move disk 3 from peg A to peg C
 Move disk 1 from peg B to peg A
 Move disk 2 from peg B to peg C
Ayeshas-MacBook-Pro:CMPE 187 Ayesha$ 
```

CFG for tower of hanoi program


Start
int num
towers(int num, char frompeg, char topeg, char auxpeg)
if (num == 1)
T
printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
return
F
towers(num - 1, frompeg, auxpeg, topeg);
printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
towers(num - 1, auxpeg, topeg, frompeg);

```c
/*
 * This is a program that, when provided with P (progress), X (x-coordinate),
 * and Y (y-coordinate), determines whether or not the pixel at (X, Y) is BLUE
 * (not shaded) or RED (shaded). The shading represents the fullness of a
 * "progress bar" that is in the shape of a circle. This circle shades in
 * clockwise order going through quandrants 1, 2, 3, then 4 of the screen in
 * that order.
 *
 * The function RED_OR_BLUE completes this task by first checking the trivial
 * cases (no progress, any progress with the (X, Y) point being in the center,
 * (X, Y) point lying in one of the screen's corners). Once the trivial cases
 * are enumerated, the variable boundary line of the shaded region is modeled
 * as the hypotenuse of a triangle located in one of the four quandrants
 * (depending on the input P). Once it is determined which quandrant the
 * boundary line exists in, we can solve for the y-coordinate of the hypotenuse
 * at the input point X and compare this with the input point Y to see if the
 * point (X, Y) lies inside or outside of the shaded region.
 *
 *
 * Compile: `gcc test_driver.c -lm -o test_driver`
 * Run: `./test_driver`
 */

#include <stdio.h>
#include <math.h>

char * red_or_blue(double p, int x, int y) {
    /*
     * Given the percentage of the circle shaded P and the coordinates for the
     * point in question (X, Y), return whether or not the point lies inside
     * the shaded sector of the circle.
     */

    if (p <= 0) {
        /*
         * If no progress has been made, then every pixel is trivially blue.
         */

        return "BLUE";
    }

    /*
     * Radius of the circle. Also used as the center pixel's (x,y) value when
     * calculating HEIGHT_FROM_CENTER.
     */
    int r = 50;

    // Apply Pythagorean theorem to solve for the missing side at point X.
    double height_from_center = sqrt(pow(r, 2) - pow((x-r), 2));

    if ((y > r + height_from_center) || (y < r - height_from_center)) {
        /*
         * Given the radius of the circle, we can calculate where the "corners"
         * of the screen are. If our (X,Y) point lies in one of these
         * regions, we can trivially determine that the pixel corresponding to
         * the point (X, Y) is blue.
         */

        return "BLUE";
    }

    /*
     * At this point we know that our point (X, Y) does not lie in one of the
     * screen's "corners" and we know that at least some portion of the
     * progress circle is shaded.
     *
     * We separate the circle into quandrants so we can apply Pythagoras'
     * theorem and exploit the rules of right triangles. For each triangle, we
     * calculate the equation for its hypotenuse and solve for the y at the
     * given X value. Then we check if the given point Y is above or below this
     * point y allowing us to conclude for certain whether or not the given
     * point (X, Y) is within the shaded sector.
     */

    double theta;
    double line(double theta, int x) {
        return tan(theta)*(x-50) + 50;
    }

    if (p <= 0.25) {
        /*
         * The boundary line lies in the first quandrant.
         */

        theta = (90 - (p*360)) * M_PI / 180.0;
        if (y >= (int) line(theta, x) && x >= 50)
            return "RED";
    } else if (p <= 0.5) {
        /*
         * The boundary line lies in the second quandrant.
         */

        theta = -((p*360) - 90) * M_PI / 180.0;
        if (y >= (int) line(theta, x) && x >= 50)
            return "RED";
    } else if (p <= 0.75) {
        /*
         * The boundary line lies in the third quandrant.
         */

        theta = (270 - (p*360)) * M_PI / 180.0;
        if (y <= (int) line(theta, x) || x >= 50)
            return "RED";
    } else {
        /*
         * The boundary line lies in the fourth quandrant.
         */

        theta = -((p*360) - 270) * M_PI / 180.0;
        if (y <= (int) line(theta, x) || x >= 50)
            return "RED";
    }

    return "BLUE";
}
```

```
/*
 * This is the test driver for our "progress bar". It tests to ensure that:
 *      - BLUE is returned trivially when no progress has been made
 *      - RED is returned trivially when any progress has been made and the
 *        center point is used
 *      - BLUE is returned trivially when the point lies in any of the screen's
 *        corners
 *      - RED and BLUE are returned when the point lies inside or outside,
 *        respectively, of the shaded region when progress is 1/8, 3/8, 5/8, 7/8
 *        complete.
 *
 * Compile: `gcc test_driver.c -lm -o test_driver`
 * Run: `./test_driver`
 */

#include "progress_bar.h"

int main() {
    char * color;
    double p;
    int x, y;

    /*
     * Testing when non-positive progress has been made.
     * Should return BLUE.
     */
    p = 0, x = 50, y = 50;
    color = red_or_blue(p, x, y);
    printf("Testing when non-positive progress has been made.\nShould return BLU
E.\np=%f, x=%d, y=%d: %s\n\n", p, x, y, color);

    /*
     * Testing when the center point is used for any amount of progress.
     * Should return RED.
     */
    p = 0.0001, x = 50, y = 50;
    color = red_or_blue(p, x, y);
    printf("Testing when the center point is used for any amount of progress.\nS
hould return RED.\np=%f, x=%d, y=%d: %s\n\n", p, x, y, color);

    /*
     * Testing when the progress circle is 1/8 shaded and the point lies
     * within the shaded region.
     * Should return RED.
     */
    p = 0.125, x = 75, y = 75;
    color = red_or_blue(p, x, y);
    printf("Testing when the progress circle is 1/8 shaded and the point lies wi
thin the shaded region.\nShould return RED.\np=%f, x=%d, y=%d: %s\n\n", p, x, y,
 color);

    /*
     * Testing when the progress circle is 1/8 shaded and the point lies
     * outside of the shaded region.
     * Should return BLUE.
     */
    p = 0.125, x = 75, y = 74;
    color = red_or_blue(p, x, y);
    printf("Testing when the progress circle is 1/8 shaded and the point lies ou
tside of the shaded region.\nShould return BLUE.\np=%f, x=%d, y=%d: %s\n\n", p,
x, y, color);

    /*
     * Testing when the progress circle is 3/8 shaded and the point lies
     * within the shaded region.
     * Should return RED.
     */
    p = 0.375, x = 75, y = 25;
    color = red_or_blue(p, x, y);
    printf("Testing when the progress circle is 3/8 shaded and the point lies wi
thin the shaded region.\nShould return RED.\np=%f, x=%d, y=%d: %s\n\n", p, x, y,
 color);

    /*
     * Testing when the progress circle is 3/8 shaded and the point lies
     * outside of the shaded region.
     * Should return BLUE.
     */
    p = 0.375, x = 75, y = 24;
    color = red_or_blue(p, x, y);
    printf("Testing when the progress circle is 3/8 shaded and the point lies ou
tside of the shaded region.\nShould return BLUE.\np=%f, x=%d, y=%d: %s\n\n", p,
x, y, color);

    /*
     * Testing when the progress circle is 5/8 shaded and the point lies
     * within the shaded region.
     * Should return RED.
     */
    p = 0.625, x = 25, y = 25;
    color = red_or_blue(p, x, y);
    printf("Testing when the progress circle is 5/8 shaded and the point lies wi
thin the shaded region.\nShould return RED.\np=%f, x=%d, y=%d: %s\n\n", p, x, y,
 color);

    /*
     * Testing when the progress circle is 5/8 shaded and the point lies
     * outside of the shaded region.
     * Should return BLUE.
     */
    p = 0.625, x = 25, y = 26;
    color = red_or_blue(p, x, y);
    printf("Testing when the progress circle is 5/8 shaded and the point lies ou
tside of the shaded region.\nShould return BLUE.\np=%f, x=%d, y=%d: %s\n\n", p,
x, y, color);

    /*
     * Testing when the progress circle is 7/8 shaded and the point lies
     * within the shaded region.
     * Should return RED.
     */
    p = 0.875, x = 25, y = 75;
    color = red_or_blue(p, x, y);
    printf("Testing when the progress circle is 7/8 shaded and the point lies wi
thin the shaded region.\nShould return RED.\np=%f, x=%d, y=%d: %s\n\n", p, x, y,
 color);

    /*
     * Testing when the progress circle is 7/8 shaded and the point lies
     * outside of the shaded region.
     * Should return BLUE.
     */
    p = 0.875, x = 25, y = 76;
    color = red_or_blue(p, x, y);
    printf("Testing when the progress circle is 7/8 shaded and the point lies ou
tside of the shaded region.\nShould return BLUE.\np=%f, x=%d, y=%d: %s\n\n", p,
x, y, color);

    /*
     * Testing when the point lies in the top-left corner of the screen.
     * Should return BLUE.
     */
    p = 1, x = 0, y = 100;
    color = red_or_blue(p, x, y);
    printf("Testing when the point lies in the top-left corner of the screen.\nS
hould return BLUE\np=%f, x=%d, y=%d: %s\n\n", p, x, y, color);

    /*
     * Testing when the point lies in the top-right corner of the screen.
     * Should return BLUE.
     */
    p = 1, x = 100, y = 100;
    color = red_or_blue(p, x, y);
    printf("Testing when the point lies in the top-right corner of the screen.\n
Should return BLUE.\np=%f, x=%d, y=%d: %s\n\n", p, x, y, color);

    /*
     * Testing when the point lies in the bottom-right corner of the
     * screen.
     * Should return BLUE.
     */
    p = 1, x = 100, y = 0;
    color = red_or_blue(p, x, y);
    printf("Testing when the point lies in the bottom-right corner of the screen
.\nShould return BLUE.\np=%f, x=%d, y=%d: %s\n\n", p, x, y, color);

    /*
     * Testing when the point lies in the bottom-left corner of the
     * screen.
     * Should return BLUE.
     */
    p = 1, x = 0, y = 0;
    color = red_or_blue(p, x, y);
    printf("Testing when the point lies in the bottom-left corner of the screen.
\nShould return BLUE.\np=%f, x=%d, y=%d: %s\n\n", p, x, y, color);

    return 0;
}
```

```c
/*
 * Title: C Program to Solve Tower-of-Hanoi Problem using Recursion
 * Author: Manish Bhojasia
 * Availability: http://www.sanfoundry.com/c-program-tower-of-hanoi-using-recurs
ion/
 *
 * Compile: `gcc -o tower_of_hanoi tower_of_hanoi.c`
 * Run: `./tower_of_hanoi`
 */

#include <stdio.h>

void towers(int, char, char, char);

int main() {
    int num;

    printf("Enter the number of disks : ");
    scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi are :\n");
    towers(num, 'A', 'C', 'B');

    return 0;
}

void towers(int num, char frompeg, char topeg, char auxpeg) {

    if (num == 1) {
        printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
        return;
    }

    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);
}
```