1.

Test Cases:

i. N = 10

```
Enter a large number: 10

Pi = 3.6000000000
Pi = 3.6000000000
Pi = 3.6000000000
Pi = 2.8000000000
Pi = 3.6000000000
Pi = 2.8000000000
Pi = 2.8000000000
Pi = 1.6000000000
Pi = 3.2000000000
Pi = 2.8000000000

The mean is 3.0400000000

The sd is 0.5987
```

ii. N = 100

```
Enter a large number: 100

Pi = 2.9600000000
Pi = 3.2000000000
Pi = 3.1200000000
Pi = 3.2000000000
Pi = 3.0800000000
Pi = 3.3200000000
Pi = 3.0400000000
Pi = 3.0800000000
Pi = 3.3200000000
Pi = 3.2000000000

The mean is 3.1520000000

The sd is 0.1114
```

iii. N = 1000

```
Enter a large number: 1000

Pi = 3.2000000000
Pi = 3.2040000000
Pi = 3.1600000000
Pi = 3.0960000000
Pi = 3.1560000000
Pi = 3.0800000000
```

```
Pi = 3.1360000000
Pi = 3.0720000000
Pi = 3.0840000000
Pi = 3.1520000000

The mean is 3.1342000000

The sd is 0.0463
```

iv. N = 100000

```
Enter a large number: 100000

Pi = 3.1416000000
Pi = 3.1356800000
Pi = 3.1534400000
Pi = 3.1512400000
Pi = 3.1459600000
Pi = 3.1405200000
Pi = 3.1428800000
Pi = 3.1439600000
Pi = 3.1459600000
Pi = 3.1405200000

The mean is 3.1442680000

The sd is 0.0050
```

v. N = 1000000

```
Enter a large number: 1000000

Pi = 3.1408080000
Pi = 3.1425480000
Pi = 3.1409880000
Pi = 3.1398160000
Pi = 3.1398040000
Pi = 3.1406800000
Pi = 3.1428360000
Pi = 3.1396640000
Pi = 3.1398200000
Pi = 3.1383920000

The mean is 3.1405492000

The sd is 0.0013
```

vi. N = 10000000

```
Enter a large number: 10000000

Pi = 3.1411688000
```

Pi = 3.1409196000
Pi = 3.1420684000
Pi = 3.1420988000
Pi = 3.1419104000
Pi = 3.1431564000
Pi = 3.1418624000
Pi = 3.1420568000
Pi = 3.1409128000
Pi = 3.1408848000

The mean is 3.1417154000

The sd is 0.0007

vii. N = 100000000

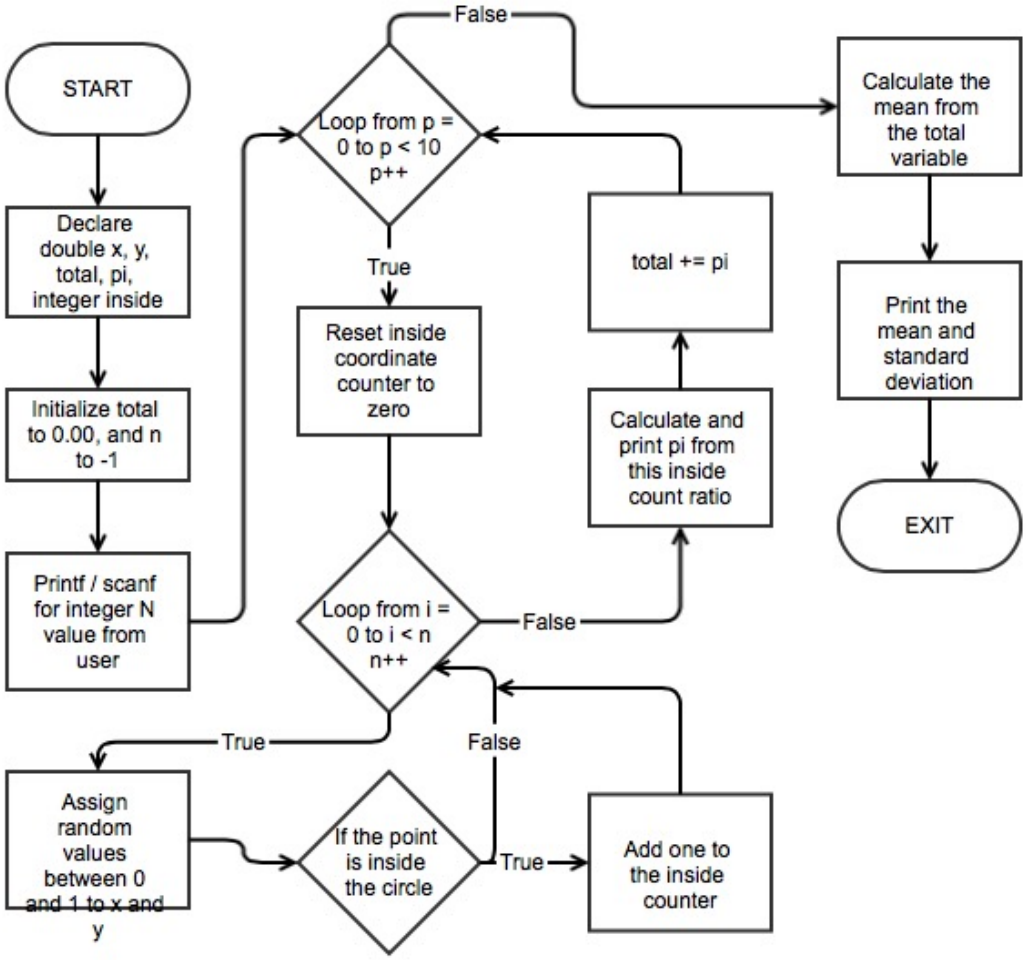
Enter a large number: 100000000

Pi = 3.1417918400
Pi = 3.1414920800
Pi = 3.1415872000
Pi = 3.1413118400
Pi = 3.1414663200
Pi = 3.1415348000
Pi = 3.1413266000
Pi = 3.1417587600
Pi = 3.1412110800
Pi = 3.1416888000

The mean is 3.1415281920

The sd is 0.0002

Based on my observations, as the value of N increases, the calculated values of pi become more accurate. This is shown by the mean converging on the actual value of pi and the standard deviation approaching zero.


START
Declare double x, y, total, pi, integer inside
Initialize total to 0.00, and n to -1
Printf / scanf for integer N value from user
Loop from p = 0 to p < 10 p++
False
True
Reset inside coordinate counter to zero
Loop from i = 0 to i < n n++
False
True
Assign random values between 0 and 1 to x and y
If the point is inside the circle
False
True
Add one to the inside counter
Calculate and print pi from this inside count ratio
total += pi
Calculate the mean from the total variable
Print the mean and standard deviation
EXIT

2.

Test Cases:

i. Test for even input and for numbers that are not between 1 and 99

```
Enter the size of magic square: 100
Please enter a valid odd integer.
Enter the size of magic square: 151
Please enter a valid odd integer.
Enter the size of magic square: 8
Please enter a valid odd integer.
Enter the size of magic square:
```

ii. Test for a negative input

```
Enter the size of magic square: -1
Please enter a valid odd integer.
Enter the size of magic square:
```

iii. Test for magic square 1 x 1

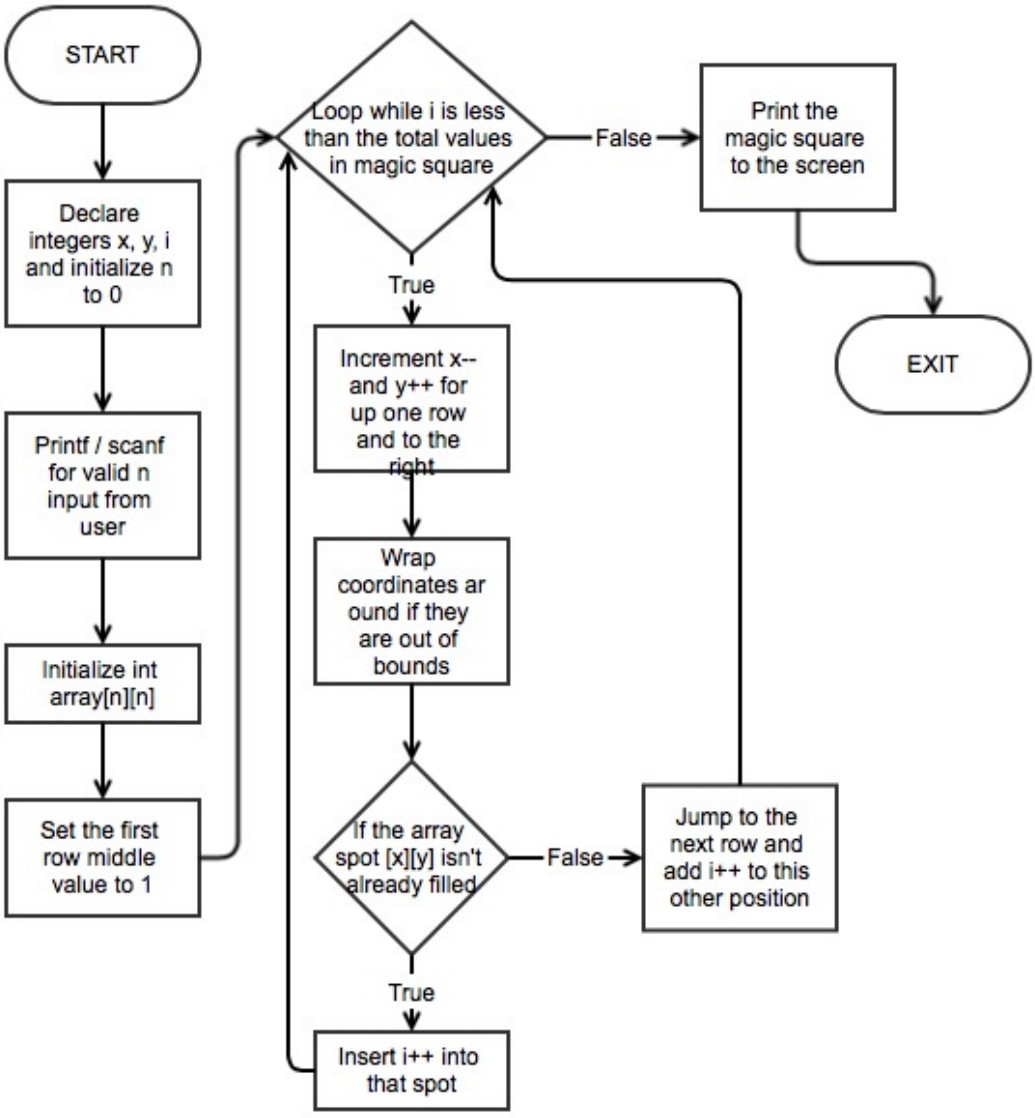
```
Enter the size of magic square: 1

1
```

iv. Test for a large valid number

```
Enter the size of magic square: 7

30      39      48      1       10      19      28
38      47      7       9       18      27      29
46      6       8       17      26      35      37
5       14      16      25      34      36      45
13      15      24      33      42      44      4
21      23      32      41      43      3       12
22      31      40      49      2       11      20
```


START
Declare integers x, y, i and initialize n to 0
Printf / scanf for valid n input from user
Initialize int array[n][n]
Set the first row middle value to 1
Loop while i is less than the total values in magic square
False
Print the magic square to the screen
EXIT
True
Increment x-- and y++ for up one row and to the right
Wrap coordinates ar ound if they are out of bounds
If the array spot [x][y] isn't already filled
False
Jump to the next row and add i++ to this other position
True
Insert i++ into that spot

3.

Test Cases:

i. Test case for one loonie $1

```
Enter the amount of dollars: $1

The smallest number of bills and coins to make up $1 is:
0 x $20's
0 x $10's
0 x $5's
0 x $2's
1 x $1's
```

ii. Test case for one toonie $2

```
Enter the amount of dollars: $2

The smallest number of bills and coins to make up $2 is:
0 x $20's
0 x $10's
0 x $5's
1 x $2's
0 x $1's
```

iii. Test case for one five $5

```
Enter the amount of dollars: $5

The smallest number of bills and coins to make up $5 is:
0 x $20's
0 x $10's
1 x $5's
0 x $2's
0 x $1's
```

iv. Test case for one ten $10

```
Enter the amount of dollars: $10

The smallest number of bills and coins to make up $10 is:
0 x $20's
1 x $10's
0 x $5's
0 x $2's
0 x $1's
```

v. Test case for one twenty $20

```
Enter the amount of dollars: $20

The smallest number of bills and coins to make up $20 is:
1 x $20's
0 x $10's
0 x $5's
0 x $2's
0 x $1's
```

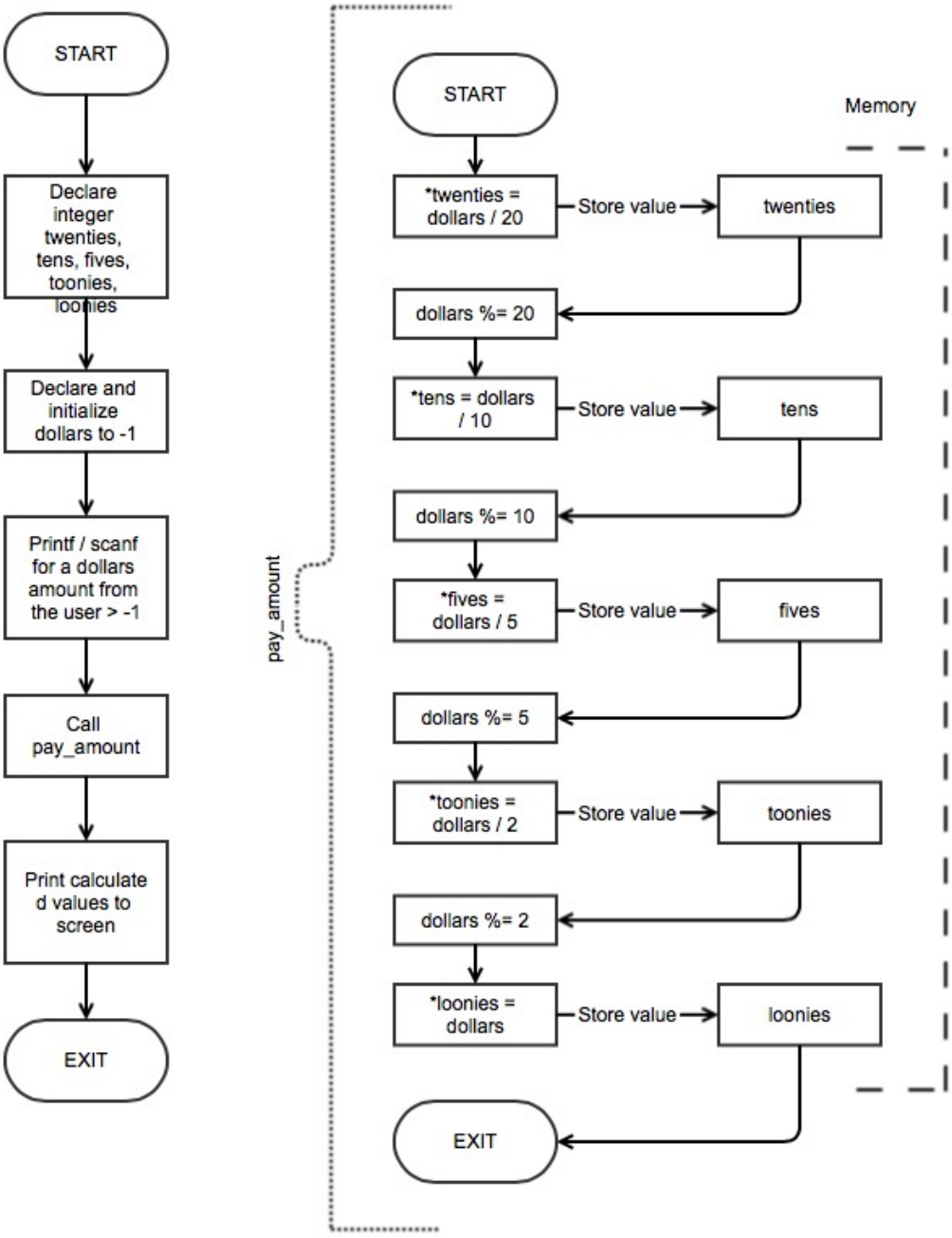
vi. Test case for negative input

```
Enter the amount of dollars: $-1
Please enter a positive amount of dollars.
Enter the amount of dollars: $
```

vii. Test case for a large amount $458

```
Enter the amount of dollars: $458

The smallest number of bills and coins to make up $458 is:
22 x $20's
1 x $10's
1 x $5's
1 x $2's
1 x $1's
```


START
Declare integer twenties, tens, fives, toonies, loonies
Declare and initialize dollars to -1
Printf / scanf for a dollars amount from the user > -1
Call pay_amount
Print calculated values to screen
EXIT
pay_amount
START
Memory
*twenties = dollars / 20
Store value
twenties
dollars %= 20
*tens = dollars / 10
Store value
tens
dollars %= 10
*fives = dollars / 5
Store value
fives
dollars %= 5
*toonies = dollars / 2
Store value
toonies
dollars %= 2
*loonies = dollars
Store value
loonies
EXIT