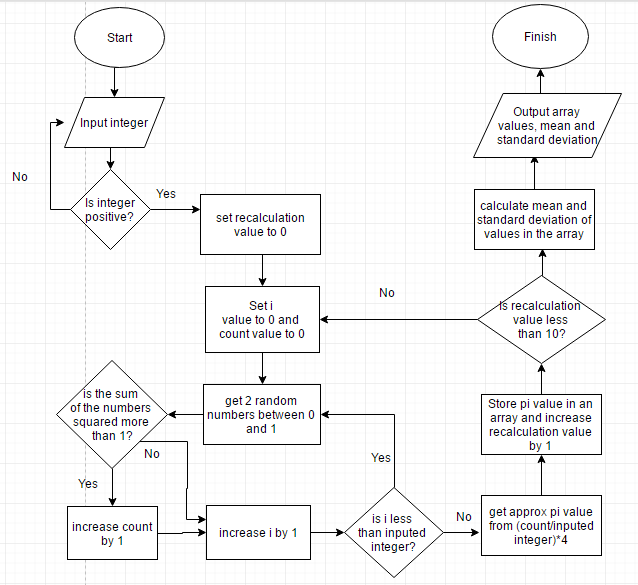
**Program 1**Flowchart:



Testcase results:

> gcc -lm Q1.c  
> a.out  
Enter integer: 0  
Error, re-enter integer: -10  
Error, re-enter integer: 10  
N = 10:  
Approx pi values:  
2.800000  
3.200000  
3.600000  
3.600000  
4.000000  
2.400000  
3.200000  
3.200000  
3.200000  
3.200000  
Mean: 3.240000  
Standard Deviation: 0.417612  
> a.out  
Enter integer: 100  
N = 100:  
Approx pi values:  
3.200000  
3.200000  
3.280000  
3.080000  
2.760000  
3.360000  
3.080000  
3.200000  
3.040000  
3.320000  
Mean: 3.152000  
Standard Deviation: 0.164730  
> a.out  
Enter integer: 1000  
N = 1000:  
Approx pi values:  
3.176000  
3.092000  
3.100000  
3.156000  
3.192000  
3.184000  
3.168000  
3.244000  
3.148000  
3.236000  
Mean: 3.169600  
Standard Deviation: 0.047132  
> a.out  
Enter integer: 100000  
N = 100000:  
Approx pi values:  
3.139800  
3.141760  
3.139560  
3.140680  
3.148920  
3.125920  
3.144120  
3.146080  
3.145280  
3.143320  
Mean: 3.141544  
Standard Deviation: 0.005921  
> a.out  
Enter integer: 10000000  
N = 10000000:  
Approx pi values:  
3.140682  
3.140770  
3.141714  
3.141170  
3.141426  
3.141150  
3.142012  
3.142025  
3.141565  
3.141374  
Mean: 3.141389  
Standard Deviation: 0.000438

Discussion

As N gets larger the mean gets closer to the actual value of pi. This is because for each approximated pi value, there are more coordinates taken and so the ratio is a more accurate representation of the area of the quadrant. As well having more coordinates makes decreases the likelihood for each approximated pi value to be different from one another so the standard deviation is smaller for larger N values. In other words, larger sample sizes increases accuracy and decreases standard deviation.

C Code

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

//compile using gcc -lm Q1.c

int main(void){

srand((unsigned) time(NULL));

double x, y, ratio, approxPi, sumPi = 0, mean, sumSD = 0, sd, count = 0;

int N, validN = 0, i = 0, j = 0, k = 0;

double ratios[10];

printf("Enter integer: ");

scanf("%d", &N);

//loop for positive integer

while (N<=0){

printf("Error, re-enter integer: ");

scanf("%d", &N);

}

printf("N = %d:\n", N);

printf("Approx pi values:\n");

//calculate pi 10 times

for (j; j<10; j++){

count = 0;

i = 0;

//count number of coordinates inside quadrant

for (i; i<N; i++){

x = (float)rand()/RAND\_MAX;

y = (float)rand()/RAND\_MAX;

if ( (x\*x + y\*y) <= 1 ){

count += 1;

}

}

ratio = count/N;

approxPi = ratio\*4;

//store approximate pi values in array for standard deviation

//calculation

ratios[j] = approxPi;

sumPi += approxPi;

printf("%f\n", approxPi);

}

mean = sumPi/10;

//calculate the numerator value of the standard deviation

for (k; k<10; k++){

sumSD += (ratios[k]-mean)\*(ratios[k]-mean);

}

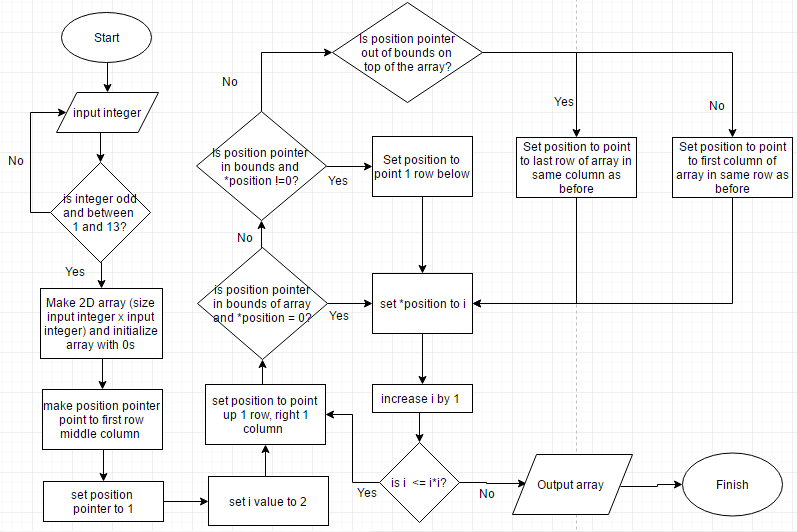
sd = sqrt(sumSD/10);

printf("Mean: %f\n", mean);

printf("Standard Deviation: %f\n", sd);

}

**Program 2**Flowchart



Testcase results:

> gcc Q2.c  
> a.out  
Enter size: 0  
Error, re-enter size: -10  
Error, re-enter size: 6  
Error, re-enter size: 15  
Error, re-enter size: 1  
1  
> a.out  
Enter size: 3  
8 1 6  
3 5 7  
4 9 2  
> a.out  
Enter size: 5  
17 24 1 8 15  
23 5 7 14 16  
4 6 13 20 22  
10 12 19 21 3  
11 18 25 2 9  
> a.out  
Enter size: 11  
68 81 94 107 120 1 14 27 40 53 66  
80 93 106 119 11 13 26 39 52 65 67  
92 105 118 10 12 25 38 51 64 77 79  
104 117 9 22 24 37 50 63 76 78 91  
116 8 21 23 36 49 62 75 88 90 103  
7 20 33 35 48 61 74 87 89 102 115  
19 32 34 47 60 73 86 99 101 114 6  
31 44 46 59 72 85 98 100 113 5 18  
43 45 58 71 84 97 110 112 4 17 30  
55 57 70 83 96 109 111 3 16 29 42  
56 69 82 95 108 121 2 15 28 41 54  
> a.out  
Enter size: 13  
93 108 123 138 153 168 1 16 31 46 61 76 91  
107 122 137 152 167 13 15 30 45 60 75 90 92  
121 136 151 166 12 14 29 44 59 74 89 104 106  
135 150 165 11 26 28 43 58 73 88 103 105 120  
149 164 10 25 27 42 57 72 87 102 117 119 134  
163 9 24 39 41 56 71 86 101 116 118 133 148  
8 23 38 40 55 70 85 100 115 130 132 147 162  
22 37 52 54 69 84 99 114 129 131 146 161 7  
36 51 53 68 83 98 113 128 143 145 160 6 21  
50 65 67 82 97 112 127 142 144 159 5 20 35  
64 66 81 96 111 126 141 156 158 4 19 34 49  
78 80 95 110 125 140 155 157 3 18 33 48 63  
79 94 109 124 139 154 169 2 17 32 47 62 77

C Code

#include <stdio.h>

#include <stdlib.h>

int main(void){

int size, mid, number = 2, i = 1;

printf("Enter size: ");

scanf("%d", &size);

//loop for positive odd integer

while ( (size%2 == 0) || (size <1) || (size> 13) ){

printf("Error, re-enter size: ");

scanf("%d", &size);

}

int magic[size][size], \*limit = magic, \*p = magic, \*m = magic, \*a;

limit = limit + (size\*size);

mid = size/2;

//initialize array to be filled with 0s

for(m = magic; m<limit; m++){

\*m = 0;

}

p = p + mid;

\*p = 1;

//insert numbers into the array

for (number; number <= (size\*size); number++){

//pointer moving up 1 row right 1 column (assuming no wrapping occurs)

a = p - size + 1;

//check if wrapping occurred from changing pointer by checking if

// a and p are in the same row

int d = 1, sameRow = 0;

for (d; d<size; d++){

m = a - d\*size;

if (m == magic){

sameRow = 1;

}

}

//fill directly below p if p is the top right corner

m = magic;

m = m + size -1;

if (p == m){

p = p + size;

\*p = number;

} else {

//fill \*a if moving up 1 row right 1 column doesn't

//move out of bounds, wrap around and a is not already filled

if ((a < limit) && (magic <= a) && (\*a == 0) && (sameRow ==0)){

p = a;

\*p = number;

} else {

//fill directly below a if moving up 1 row 1 column stays

//in bounds, doesn't wrap around but \*a is filled

if ((a < limit) && (magic <= a) && (\*a != 0) &&

(sameRow ==0)){

a = p + size;

//make sure moving down is in bounds and not filled

if ((a < limit) && (magic <= a) && (\*a == 0) &&

(sameRow ==0)){

p = a;

\*p = number;

}

} else {

//fill on first column up one row if wrapping

//to the same row occurred

if (sameRow==1){

a = a - size;

p = a;

\*p = number;

//fill on same column bottom row if a went out

//of bounds of the array

} else {

a = p + (size-1)\*size + 1;

p = a;

\*p = number;

}

}

}

}

}

//print array

m = magic;

for (m; m<limit; m++){

printf("%-6d", \*m);

if (i%size == 0){

printf("\n");

}

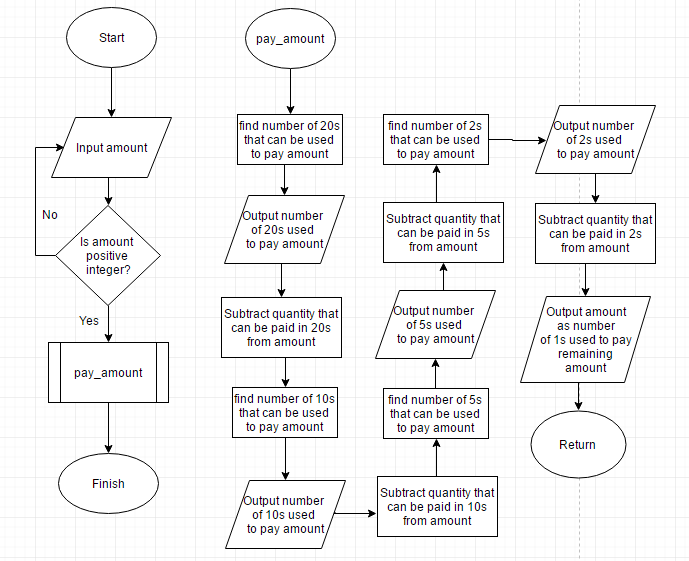
i++;

}

}

**Program 3**

Flowchart



Testcase results

> gcc Q3.c  
Q3.c:52:2: warning: no newline at end of file  
> a.out  
Enter amount: -10  
Error, re-enter amount: 0  
Error, re-enter amount: 118  
# of 20's: 5  
# of 10's: 1  
# of 5's: 1  
# of 2's: 1  
# of 1's: 1  
> a.out  
Enter amount: 81  
# of 20's: 4  
# of 10's: 0  
# of 5's: 0  
# of 2's: 0  
# of 1's: 1  
> a.out  
Enter amount: 12  
# of 20's: 0  
# of 10's: 1  
# of 5's: 0  
# of 2's: 1  
# of 1's: 0  
> a.out  
Enter amount: 1  
# of 20's: 0  
# of 10's: 0  
# of 5's: 0  
# of 2's: 0  
# of 1's: 1

C code

#include <stdio.h>

void pay\_amount(int dollars, int \*twenties, int \*tens, int \*fives, int \*toonies, int \*lonnie);

int main(void){

int amount, quantityStart = 0, \*\_20, \*\_10, \*\_5, \*\_2, \*\_1;

//quantityStart represents all bill/coins needed, adjusted accordingly to first be the # of 20s needed, then 10s,then 5s then 2s then 1s as seen in function pay\_amount()

\_1 = \_2 = \_5 = \_10 = \_20 = &quantityStart;

printf("Enter amount: ");

scanf("%d", &amount);

//loop for positive integer

while (amount<=0){

printf("Error, re-enter amount: ");

scanf("%d", &amount);

}

//calculate minimum #coins/bills needed

pay\_amount(amount, \_20, \_10, \_5, \_2, \_1);

}

void pay\_amount(int amount, int \*\_20, int \*\_10, int \*\_5, int \*\_2, int \*\_1){

//find # of 20s needed, subtract the amount paid with 20s from amount

\*\_20 = amount/20;

printf("# of 20's: %d\n", \*\_20);

amount = amount - (\*\_20)\*20;

//find # of 10s needed, subtract the amount paid with 10s from amount

\*\_10 = amount/10;

printf("# of 10's: %d\n", \*\_10);

amount = amount - (\*\_10)\*10;

//find # of 5s needed, subtract the amount paid with 5s from amount

\*\_5 = amount/5;

printf("# of 5's: %d\n", \*\_5);

amount = amount - (\*\_5)\*5;

//find # of 2s needed, subtract the amount paid with 2s from amount

\*\_2 = amount/2;

printf("# of 2's: %d\n", \*\_2);

amount = amount - (\*\_2)\*2;

//the rest of the amount must be paid with in 1s

\*\_1 = amount;

printf("# of 1's: %d\n", \*\_1);

}