Program4_1.c

```
/*
 * Program to estimate value of pi based on random guessing for user value N
 * Author: Jeet Chakrabarty
*/

#include <time.h>      //Makes random numbers more random
#include <stdio.h>     //Includes inputs and outpus
#include <stdlib.h>    //Includes use of RAND_MAX

int main (void){

	//Variables representing input, index, and sum of guesses inside circle
	int N, i, sum = 0;
	//Variables representing x and y coordinates and ratio landed inside
	double x, y, ratio;

	//Loop iterates through input reception until valid input is typed
	do {
		printf("What positive N would you like to calculate to?\n");
		scanf("%d", &N);      //Scans integer into N
	} while (N<=0);         //Condition for validity

	//Makes pseudo-random number generator based on time (more random)
	srand(time(NULL));

	//Loops through random value pairs N times
	for (i=0; i<N; i++){
		//Assigns //Assigns x random value between 0 and 1x random value between 0 and 1
		x = (double)rand()/(double)RAND_MAX;
		//Assigns y random value between 0 and 1
		y = (double)rand()/(double)RAND_MAX;

		//Increments sum if random value falls in circle
		if (x*x+y*y<=1){
			sum++;
		}
	}

	//Calculates ratio of dots landed inside to outside
	ratio = (double)sum/(double)N;

	//Outputs the approximated value of pi
	printf("The approximated value of pi is: %f\n", ratio*4);

	return 0;
}
```

Program4_1b.c

```
/*
 * Program to estimate value of pi based on random guessing
  * Author: Jeet Chakrabarty
*/

#include <time.h>      //Makes random numbers more random
#include <stdio.h>     //Includes inputs and outpus
#include <stdlib.h>    //Includes use of RAND_MAX
#include <math.h>      //Includes use of sqrt function

int main (void){
        //Variables for indices and sum of dots landed in circle
        int i, j, k, sum;
        //Values for N to loop through
        int values [] = {10, 100, 1000, 100000, 1000000, 10000000, 100000000};
        //Array to hold ten approximations per value
        double tenValues [10];
        //Array for x,y coordinates, ratio landed inside, mean, sum to calculate mean, and standard
deviation
        double x, y, ratio, mean, meanSum, stDev;


        //Makes pseudo-random number generator based on time (more random)
        srand(time(NULL));

        //Loops through designated values of N (10, 100, 1000, etc)
        for (i=0; i<sizeof(values)/sizeof(values[0]); i++){
                mean=0;         //Resets value of mean to 0
                meanSum=0;  //Resets value of sum of mean to 0

                //Loops through 10 values
                for (j=0; j<10; j++){
                        sum=0;          //Resets sum to 0

                        //Loops through N times
                        for (k=0; k<values[i]; k++){
                                //Assigns x and y coordinates random values
                                x = (double)rand()/(double)RAND_MAX;
                                y = (double)rand()/(double)RAND_MAX;

                                //Increments sum if point inside circle
                                if (x*x+y*y<=1){
                                        sum++;
                                }
                        }

                        //Calculates one pi approximation ratio
```

```
                tenValues[j] = (double)sum/(double)values[i];
                //Adds approximation to sum to calculate mean
                meanSum += tenValues[j]*4;
                //Prints approximation
                printf("%f \n", tenValues[j]*4);
            }

            //Calculates mean of 10 approximations
            mean=meanSum/10.0;
            //Resets sum to calculate mean to 0
            meanSum=0;

            //Loops through 10 values to calculate standard deviation
            for (j=0; j<10; j++){
                //Calculates value of each term
                meanSum+=(tenValues[j]*4-mean)*(tenValues[j]*4-mean);
            }
            //Calculates final standard deviation
            stDev=sqrt(meanSum/10);

            //Prints standards deviation
            printf("For %d, the mean is: %f and the std Dev is: %f \n", values[i], mean, stDev);
        }
        return 0;
}
```

Test Output for Program4_1.c:

```
obelix[105]% ./a.out
What positive N would you like to calculate to?
10000
The approximated value of pi is: 3.137600
obelix[106]% ./a.out
What positive N would you like to calculate to?
100000
The approximated value of pi is: 3.147400
obelix[107]% ./a.out
What positive N would you like to calculate to?
-58
```

Test Output for Program4_1b.c:

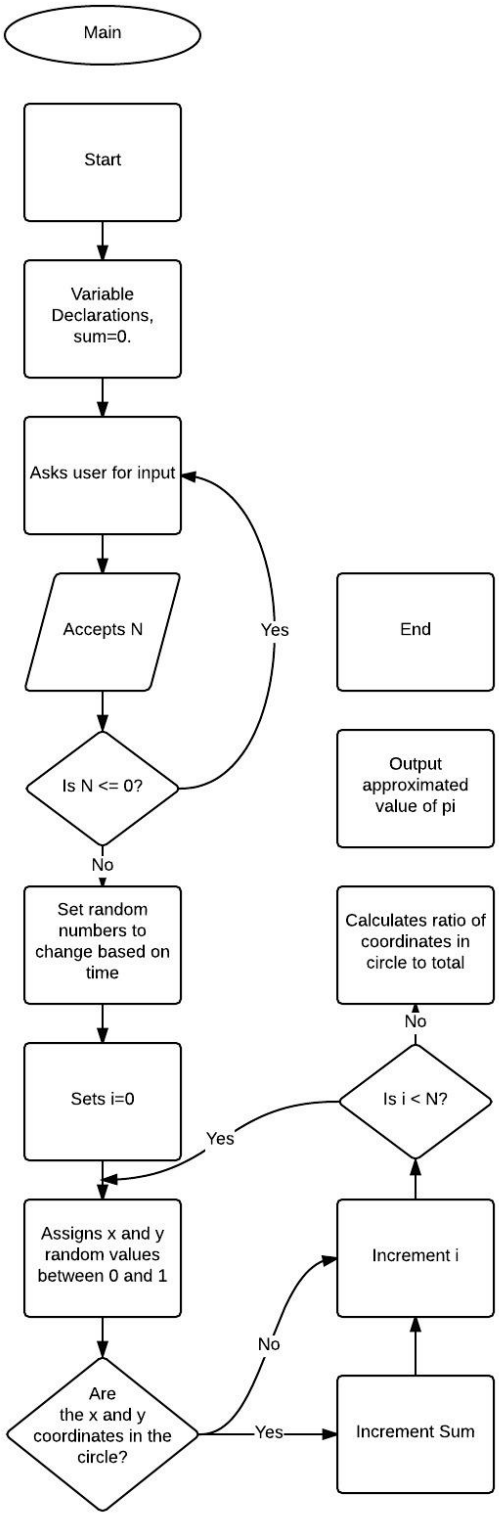
```
obelix[12]% ./a.out
2.800000
3.200000
3.600000
3.600000
3.200000
2.800000
3.200000
3.600000
4.000000
3.200000
For 10, the mean is: 3.320000 and the std Dev is: 0.360000
3.200000
2.840000
2.760000
3.120000
3.240000
3.320000
3.160000
3.160000
3.120000
3.200000
For 100, the mean is: 3.112000 and the std Dev is: 0.166661
3.140000
3.140000
3.128000
3.096000
3.136000
3.128000
3.100000
3.176000
3.196000
```

3.256000
For 1000, the mean is: 3.149600 and the std Dev is: 0.045614
3.137440
3.142880
3.140720
3.147120
3.133960
3.149320
3.140920
3.151400
3.141280
3.143160
For 100000, the mean is: 3.142820 and the std Dev is: 0.005023
3.141824
3.140152
3.142744
3.141260
3.140040
3.143848
3.141324
3.142760
3.141088
3.142428
For 1000000, the mean is: 3.141747 and the std Dev is: 0.001149
3.141753
3.141792
3.140996
3.141654
3.141126
3.141101
3.142073
3.141312
3.141607
3.141186
For 10000000, the mean is: 3.141460 and the std Dev is: 0.000344
3.141537
3.141553
3.141510
3.141342
3.141466
3.141497
3.141415
3.141654
3.141471
3.141862
For 100000000, the mean is: 3.141531 and the std Dev is: 0.

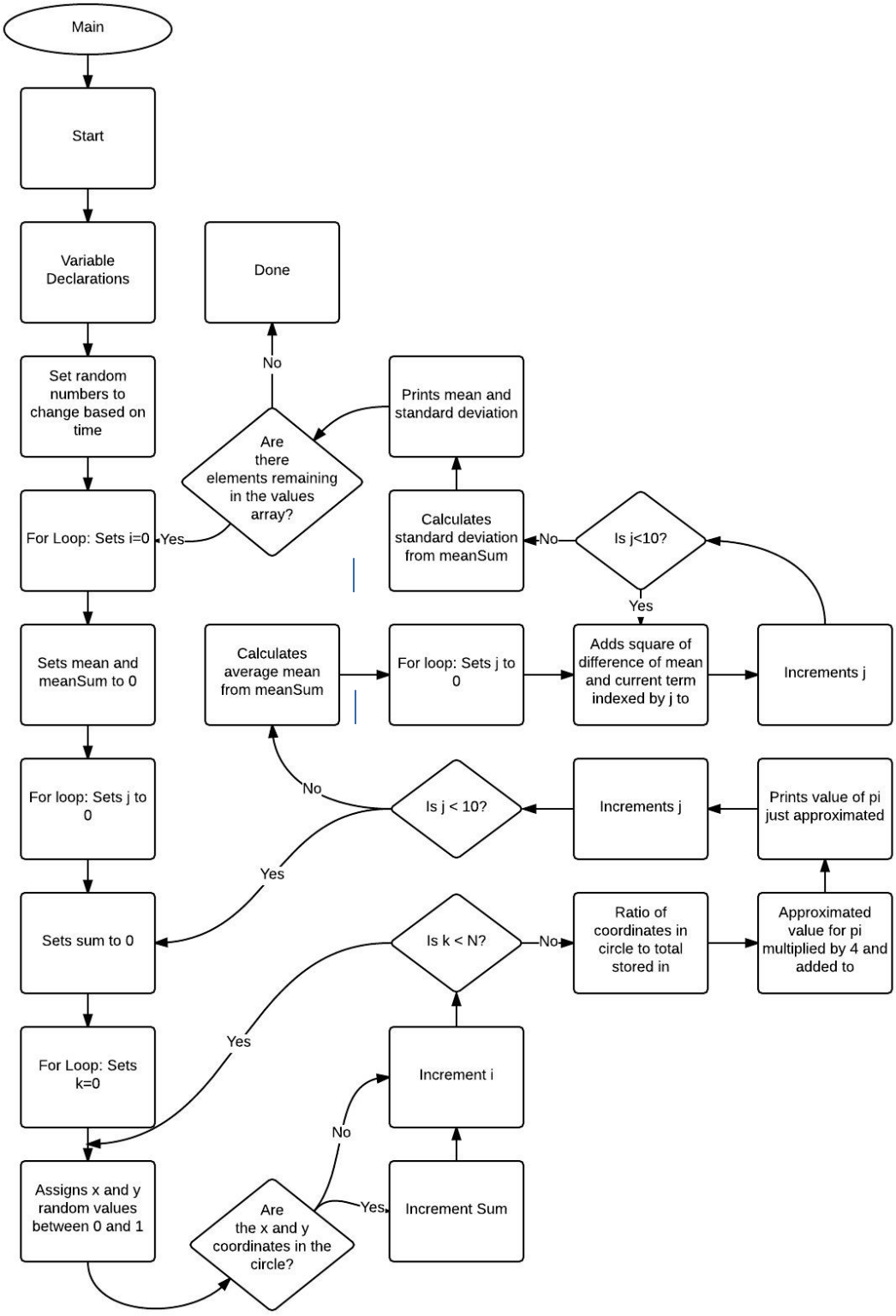
Discussion for Program 1:

As the value of N increased, the mean became progressively better at estimating the value of pi and the standard deviation became smaller (meaning it was more accurate). This confirms basic statistic theory asserting that a calculated mean becomes more accurate (and the standard deviation and confidence interval smaller) the larger the sample size becomes.

Flow Chart for Program4_1.c:



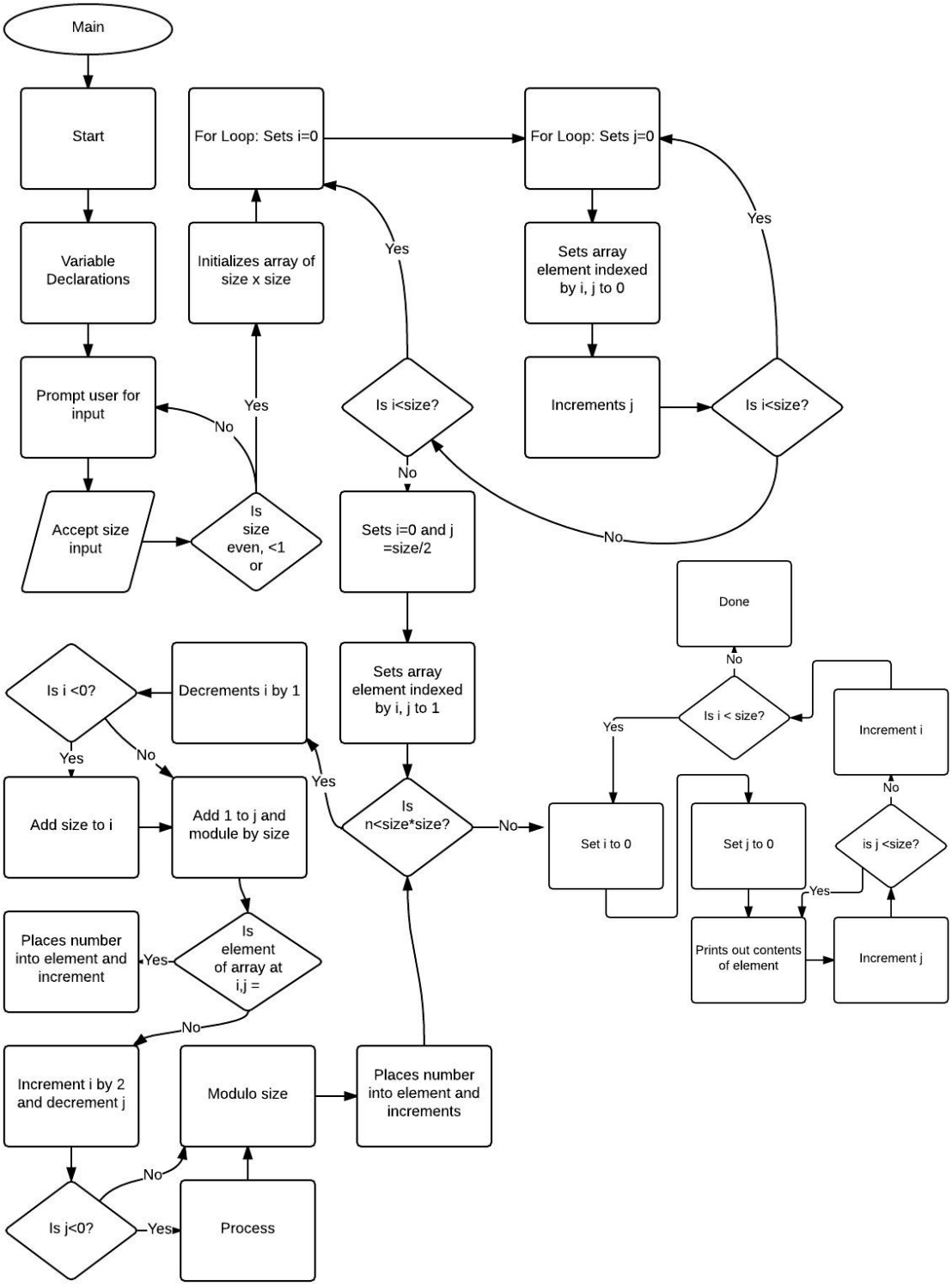
Program4_1b.c



Program4_2.c:

```
/*
 *Program for creating and and outputting magic square representing squares
 *Author: Jeet Chakrabarty
*/

#include <stdio.h>     //Includes inputs and outpus

int main (void){

        //Variables for size of magic square, and indices of 2D array
        int size, i=0, j, n=2;

        //Input loop does not continue program until valid input
        do {
                printf("Enter (odd and positive integer) size of magic square: ");
                scanf("%d", &size); //Accepts size input
        }
        while (size%2==0 && size<1 && size>99); //checks if even and valid

        //Initializes 2d square array of size inputted by user
        int array [size][size];

        for (i=0; i<size; i++)  //Sets all values in array to 0
        {
                for (j=0; j<size; j++)
                {
                        array[i][j]=0;
                }
        }
        i=0;
        j=size/2; //Sets j equivalent to the position in the middle of the array

        array[i][j]=1;  //Sets top-middle element

        //Loop to input square numbers into square
        while(n<=size*size){

                i=i-1;  //Moves up one row
                if(i<0) //Checks if index out of bounds
                        i=i+size;       //Moves to bottom of square

                j=(j+1)%size;  //Moves forward one column while wrapping

                if (array[i][j]==0)       //Checks is element is occupied
                        array[i][j]=n++;        //Places number into element then increments
                else{   //Executes if element is empty
                        i=i+2;  //Moves down 2 rows (counters move-up from before)
```

```
                        j--;       //Counters move from before
                        if(j<0) //Checks if index out of bounds
                                j=j+size;
                        i%=size;
                        array[i][j]=n++;         //Places number into element then increments

                }
        }

        //Loop to output contents of magic square
        for(i=0; i<size; i++){  //Iteratres through rows
                for(j=0; j<size; j++){  //Iterates through columns
                        printf("%2d\t", array[i][j]);    //Prints contents
                }
        printf("\n");
        }
}
```


Main
Start
Variable Declarations
Prompt user for input
Accept size input
Is size even, <1 or
No
Yes
Initializes array of size x size
For Loop: Sets i=0
Yes
Is i<size?
No
Sets i=0 and j =size/2
Sets array element indexed by i, j to 1
Is n<size*size?
Yes
No
Is i <0?
Yes
No
Decrements i by 1
Add size to i
Add 1 to j and module by size
Is element of array at i,j =
Yes
No
Places number into element and increment
Increment i by 2 and decrement j
Is j<0?
No
Yes
Modulo size
Process
Places number into element and increments
For Loop: Sets j=0
Sets array element indexed by i, j to 0
Increments j
Is i<size?
Yes
No
Done
Is i < size?
Yes
No
Set i to 0
Set j to 0
Prints out contents of element
is j <size?
Yes
No
Increment i
Increment j

Program4_2 Test Cases:

```
obelix[117]% ./a.out
Enter (odd and positive integer) size of magic square: -29
Enter (odd and positive integer) size of magic square: 9999999
Enter (odd and positive integer) size of magic square: 2
Enter (odd and positive integer) size of magic square: 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
obelix[118]% ./a.out
Enter (odd and positive integer) size of magic square: 9
47      58      69      80      1       12      23      34      45
57      68      79      9       11      22      33      44      46
67      78      8       10      21      32      43      54      56
77      7       18      20      31      42      53      55      66
 6      17      19      30      41      52      63      65      76
16      27      29      40      51      62      64      75      5
26      28      39      50      61      72      74      4       15
36      38      49      60      71      73      3       14      25
37      48      59      70      81      2       13      24      35
obelix[119]% ./a.out
Enter (odd and positive integer) size of magic square: 3
 8      1       6
 3      5       7
 4      9       2
obelix[120]% ./a.out
Enter (odd and positive integer) 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
```

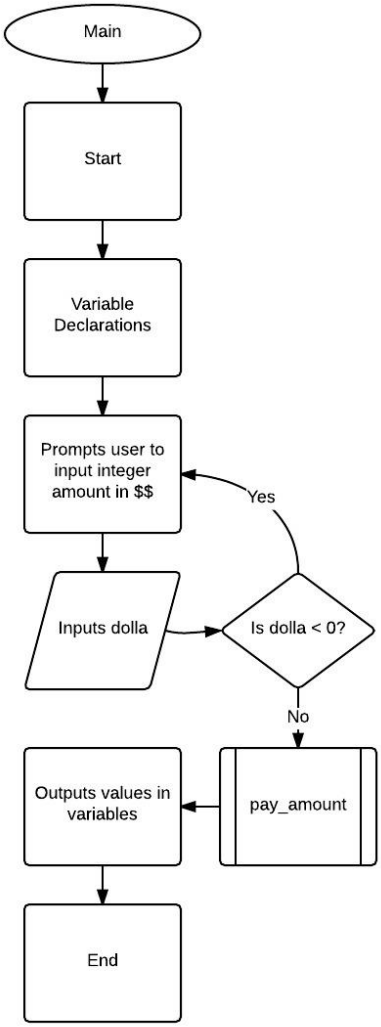
Program4_3.c

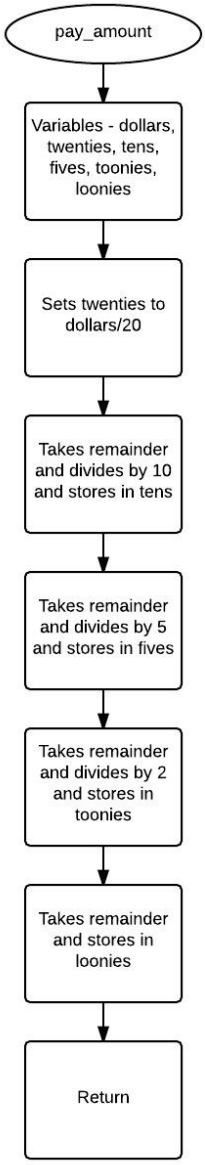
```
/*
 * Program for creating and and outputting magic square representing squares
 * Author: Jeet Chakrabarty
*/

#include <stdio.h>      //Includes inputs and outpus

/*
 * Function to calculate smallest number number of bills/coins equivalent
 * to some integer amount
 * @param dollars represents dollar amount to be calculated
 * @param *twenties, *tens, *fives, *toonies, *loonies represents pointers
 * to bill/coin amounts
*/
void pay_amount(int dollars, int *twenties, int *tens, int *fives,
                                    int *toonies, int *loonies){

        *twenties=dollars/20; //Calculates number of $20 bills
        dollars%=20;  //Changes dollar to remaining amount
        *tens=dollars/10;       //As above, except with $10 bills
        dollars%=10;
        *fives=dollars/5;
        dollars%=5;
        *toonies=dollars/2;
        dollars%=2;
        *loonies=dollars;

}

/*
 * Main program to ask user for $ and output minimum equivalent bills
*/
int main (void){

        int dolla;        //Variable representing user's dollar input
        int tw, te, fi, to, lo;     //Pointers representing change

        //Input loop does not continue program until valid input
        do {
                printf("Enter integer $ amount: ");
                scanf("%d", &dolla); //Accepts size input
        }
        while (dolla<0); //Condition repreats loop if not positive

        //Calls pay_amount function to calculate change
        pay_amount(dolla, &tw, &te, &fi, &to, &lo);
```

```
        //Prints out values dollars split into
        printf("You should get: %d twentie(s), %d ten(s), %d five(s), %d toonie(s), and %d loonie(s).
\n", tw, te, fi, to, lo);
}
```

Program4_3 Test Cases:

```
obelix[123]% ./a.out
Enter integer $ amount: -1
Enter integer $ amount: 39
You should get: 1 twentie(s), 1 ten(s), 1 five(s), 2 toonie(s), and 0 loonie(s).
obelix[124]% ./a.out
Enter integer $ amount: 38
You should get: 1 twentie(s), 1 ten(s), 1 five(s), 1 toonie(s), and 1 loonie(s).
obelix[125]% ./a.out
Enter integer $ amount: 37
You should get: 1 twentie(s), 1 ten(s), 1 five(s), 1 toonie(s), and 0 loonie(s).
obelix[129]% ./a.out
Enter integer $ amount: 9999
You should get: 499 twentie(s), 1 ten(s), 1 five(s), 2 toonie(s), and 0 loonie(s).
obelix[130]% ./a.out
Enter integer $ amount: 123456789
You should get: 6172839 twentie(s), 0 ten(s), 1 five(s), 2 toonie(s), and 0 loon
```


Main
Start
Variable Declarations
Prompts user to input integer amount in $$
Inputs dolla
Is dolla < 0?
Yes
No
pay_amount
Outputs values in variables
End


pay_amount
Variables - dollars, twenties, tens, fives, toonies, loonies
Sets twenties to dollars/20
Takes remainder and divides by 10 and stores in tens
Takes remainder and divides by 5 and stores in fives
Takes remainder and divides by 2 and stores in toonies
Takes remainder and stores in loonies
Return