Assignment 1
CPSC 331

Guransh Mangat, 30061719
Daniel Contreras, 10080311
Steven Ferguson, 30037518

Problem 1. Suppose that the function $f(n) = n$ is a bound function. (**References**: Tutorial Exercise 3)

Proof. Since n is an integer input, this is an *integer-valued* function.

When the recursive function is applied in *line 9*, the value of n is reduced by at least 1 $((n-1), (n-3)\ and\ (n-4))$.

n is a non negative integer, which means $n \geq 0$. Thus it is only possible that $f(n) = n \leq 0$ when n = 0. In that case, the test at *line 1* passes, the execution continues at *line 1* and the execution ends without having to call the recursive function repeatedly.

Hence, the function $f(n) = n$ is a bound function. ■

Problem 2. Prove that the smacG algorithm correctly solves the "MacGonagall Mystery Computation" problem. (**References**: Lecture Notes 2)

Proof. This theorem will be proved by induction on n. The strong form of mathematical induction will be used, and base cases of $n = 0, 1, 2, 3$ will each be considered in the basis.

Basis: Suppose, first $n = 0$:
During the execution of algorithm smacG on input n, the test at line 1 succeeds and the execution continues at line 2. This causes the execution to terminate with the value of $M_0 = 1$ returned at output - as required for this case

Suppose, next, $n = 1$:
During the execution of the algorithm smacG, the test at line 1 fails and the execution continues with test at line 3.
This test succeeds, so the execution continues at line 4. This causes the execution of algorithm to terminate with value of $M_1 = 0$ returned as output- as required by this case

Suppose, $n = 2$:
During the execution of the algorithm smacG, the test at line 1 and line 3 fails, and the execution continues with test at line 5.
This test succeeds, so the execution continues with line 6. This causes the execution of the algorithm to terminate with the value of $M_2 = 5$ returned as output - as require by this case.

Suppose, n = 3:
During the execution of the algorithm smacG, the test at line 1, line 3 and line 5 fails, and the execution continues with the test at line 7.
This test succeeds, so the execution continues with line 8. This causes the execution of the algorithm to terminate with the value of $M_3 = 8$ returned as output - as required by this case.

Inductive Step: Let k be an integer such that $k \geq 3$.

Inductive Hypothesis: Suppose n is a non-negative integer such that $0 \leq n \leq k$. If the algorithm smacG is executed given n as input then this execution of the algorithm eventually terminates, with the n^{th} MacGonagall number, M_n returned as output.

Inductive Claim: If the algorithm smacG is executed given $n = k + 1$ as input then this execution of the algorithm eventually terminates, with the $k + 1^{st}$ MacGonagall number $M_{k+1} = M_n$ returned as output.

Suppose the algorithm smacG is executed with $n = k + 1$ given as input. Since k is an integer such that $k \geq 3$, n is an integer such that $n \geq 4$.
Since $n \geq 4$, the test at line 1, line 3, line 5 and line 7 fails and the algorithm continues at line 8.
Line 9 includes a recursive execution of this algorithm with the input $2 \times (n - 1)$. Since $n = k + 1 \geq 4, 0 \leq n - 1 = k \leq k$, and it follows by the **inductive hypothesis** that this recursive execution of the algorithm eventually ends with $M_{n-1} = M_k$ returned as output
Line 9 also includes a recursive execution of this algorithm with the input $-2 \times (n-3)$. Since $n = k + 1 \geq 4, 0 \leq n - 3 = k - 2 \leq k$, and it follows the **inductive hypothesis** that this recursive execution of the algorithm eventually ends with $M_{n-3} = M_{k-2}$ returned as output.
Line 9 also includes a recursive execution of algorithm with the input $n - 4$. Since $n = k + 1 \geq 4, 0 \leq n - 4 = k - 3 \leq 4$, and it follows by the **inductive hypothesis** that this recursive function of the algorithm eventually ends with $M_{n-4} = M_{k-3}$ returned as output.
Once the execution of algorithm ends, since, $k + 1 \geq 4$, it follows the definition of M_{k+1}, that the value returned as output is:

$$2 \times M_{n-1} - 2 \times M_{n-3} + M_{n-4} = 2 \times M_k - 2 \times M_{k-2} + M_{k-3} = M_{k+1}$$

as required to establish the **inductive claim.**

This results now follows by induction on n. ■

Problem 3.

```
package cpsc331.assignment1;
/**
 *
 * A class for the recursive computation of the MacGonagall Mystery sequence
 *
 * @author Steven Ferguson (30037518)
 * @author Daniel Contreras (10080311)
 * @author Guransh Mangat (30061719)
 * @version 1.0
 *
 */

public class SMacGonagall {
```

```
/**
 *
 * The main method for this class.
 *
 * @param args A single non-negative integer n.
 *
 */

public static void main(String[] args) {

    if (args.length != 1) {
        System.out.println("Fiddlesticks! One integer input is required.");
        return;
    }

    try {
        int n = Integer.parseInt(args[0]);
        System.out.println(smacG(n));
    } catch (NumberFormatException e) {
        System.out.println("Fiddlesticks! One integer input is required.");
    } catch (IllegalArgumentException e) {
        System.out.println("Fiddlesticks! The integer input cannot be negative.");
    }

}

/**
 *
 *  Computes the nth MacGonagall Number M_n, throwing
 *  an IllegalArgumentException if n is negative.
 *
 * Precondition:
 * An integer n given as input.
 *
 * Postcondition:
 * The nth MacGonagall Number, M_n is returned as output, and
 * an IllegalArgumentException is thrown if n < 0.
 *
 * @param n the MacGonagall Number to be computed
 * @return the nth MacGonagall Number, M_n
 * @throws IllegalArgumentException if the input is negative
 *
 */

public static int smacG(int n) throws IllegalArgumentException {

    // Assertion: n is an integer such that n >= 0
```

```
        if(n < 0) {
            throw new IllegalArgumentException();
        }

        switch (n) {
            case 0:
                return 1;
            case 1:
                return 0;
            case 2:
                return 5;
            case 3:
                return 8;
            default:
                return 2 * smacG(n - 1) - 2 * smacG(n - 3) + smacG(n - 4);
        }

        // Assertion:
        // 1. n is an integer such that n >= 0
        // 2. The nth MacGonagall number, M_n is returned as output


    }
}
```

Problem 4. Let $T_{smacG}(n)$ be the number of steps included in the execution of the algorithm, *smacG*, shown in Figure 1 on input n, for a non-negative integer n assuming that the uniform cost criterion is used to define this and the only steps counted are the numbered steps shown in Figure 1.
Give a recurrence for $T_{smacG}(n)$.

Proof. In order to determine the number of steps included in the execution of $T_{smacG}(n)$, we will define the running time of each step in the algorithm to be *one*.

Using the uniform cost criterion to define $T_{smacG}(n)$:

The algorithm executes 2 steps (at lines 1 and 2) if it is executed when $n = 0$.
The algorithm executes 3 steps (at lines 1, 3, 4) if it is executed when $n = 1$.
The algorithm executes 4 steps (at lines 1, 3, 5, 6) if it is executed when $n = 2$.
The algorithm executes 5 steps (at lines 1, 3, 5, 7, 8) if it is executed when $n = 3$.

$$T_{smacG}(n) = \begin{cases} 2 & if n = 0, \\ 3 & if n = 1, \\ 4 & if n = 2, \\ 5 & if n = 3, \\ T_{smacG}(n-1) + T_{smacG}(n-3) + T_{smacG}(n-4) + 5 & if n \geq 4. \end{cases}$$

We can use this recurrence to show the following:

$$T_{smacG}(0) = 2 \qquad \text{(by definition of } T_{smacG}(n)\text{)}$$

$$T_{smacG}(1) = 3 \qquad \text{(by definition of } T_{smacG}(n)\text{)}$$

$$T_{smacG}(2) = 4 \qquad \text{(by definition of } T_{smacG}(n)\text{)}$$

$$T_{smacG}(3) = 5 \qquad \text{(by definition of } T_{smacG}(n)\text{)}$$

$$\begin{aligned} T_{smacG}(4) &= T_{smacG}(4-1) + T_{smacG}(4-3) + T_{smacG}(4-4) + 5 \\ &= T_{smacG}(3) + T_{smacG}(1) + T_{smacG}(0) + 5 \\ &= 5 + 3 + 2 + 5 \\ &= 15. \end{aligned}$$

$$\begin{aligned} T_{smacG}(5) &= T_{smacG}(5-1) + T_{smacG}(5-3) + T_{smacG}(5-4) + 5 \\ &= T_{smacG}(4) + T_{smacG}(2) + T_{smacG}(1) + 5 \\ &= 15 + 4 + 3 + 5 \\ &= 27. \end{aligned}$$

■

Problem 5. Suppose that T_{smacG} is a function of the non-negative integers such that, for every integer $n \geq 0$,

$$T_{smacG}(n) = \begin{cases} 2 & n = 0 \\ 3 & n = 1 \\ 4 & n = 2 \\ 5 & n = 3 \\ T_{smacG}(n-1) + T_{smacG}(n-3) + T_{smacG}(n-4) + 5 & n \geq 4 \end{cases}$$

then $T_{smacG}(n) \geq \left(\frac{3}{2}\right)^n$ for every non-negative integer n and therefore the number of steps executed by the *smacG* algorithm is at least exponential in its input.

Proof. This will be established by induction on n. The strong form of mathematical induction will be used and the cases $n = 0$, $n = 1$, $n = 2$ and $n = 3$ will be considered in the basis.

Basis. If $n = 0$, then $T_{smacG}(n) = T_{smacG}(0) = 2$, and

$$\begin{aligned} T_{smacG}(n) &\geq \left(\frac{3}{2}\right)^n \\ T_{smacG}(0) &\geq \left(\frac{3}{2}\right)^0 \\ 2 &\geq 1 \end{aligned}$$

as required. If $n = 1$, then $T_{smacG}(n) = T_{smacG}(1) = 3$, and

$$\begin{aligned} T_{smacG}(n) &\geq \left(\frac{3}{2}\right)^n \\ T_{smacG}(1) &\geq \left(\frac{3}{2}\right)^1 \\ 3 &\geq \frac{3}{2} \end{aligned}$$

as required. If $n = 2$, then $T_{smacG}(n) = T_{smacG}(2) = 4$, and

$$\begin{aligned} T_{smacG}(n) &\geq \left(\frac{3}{2}\right)^n \\ T_{smacG}(2) &\geq \left(\frac{3}{2}\right)^2 \\ 4 &\geq \frac{9}{4} \end{aligned}$$

as required. If $n = 3$, then $T_{smacG}(n) = T_{smacG}(3) = 5$, and

$$\begin{aligned} T_{smacG}(n) &\geq \left(\frac{3}{2}\right)^n \\ T_{smacG}(3) &\geq \left(\frac{3}{2}\right)^3 \\ 5 &\geq \frac{27}{8} \end{aligned}$$

as required.

Inductive Step. Let $k \geq 4$ be an integer. It is necessary and sufficient to use the following

Inductive Hypothesis: $T_{smacG}\left(m\right) \geq \left(\frac{3}{2}\right)^m$ for every integer m such that $0 \leq m \leq k$.

to prove the following

Inductive Claim: $T_{smacG}\left(k+1\right) \geq \left(\frac{3}{2}\right)^{k+1}$

Now, since $k+1 > 4$,

$$\begin{aligned}
T_{smacG}(k+1) &= T_{smacG}(k) + T_{smacG}(k-2) + T_{smacG}(k-3) + 5 \\
&\geq \left(\frac{3}{2}\right)^{k} + \left(\frac{3}{2}\right)^{k-2} + \left(\frac{3}{2}\right)^{k-3} + 5 && \text{(by inductive hypothesis)} \\
&= \left(\frac{3}{2}\right)^{k+1} + \left(\left(\frac{3}{2}\right)^{-1} + \left(\frac{3}{2}\right)^{-3} + \left(\frac{3}{2}\right)^{-4}\right) + 5 \\
&= \left(\frac{3}{2}\right)^{k+1}\left(\frac{94}{81}\right) + 5 \\
&\geq \left(\frac{3}{2}\right)^{k+1}\left(\frac{94}{81}\right) \\
&\geq \left(\frac{3}{2}\right)^{k+1} && \left(\text{because } \frac{94}{81} > 1\right)
\end{aligned}$$

thus establishing the inductive claim, as needed. Therefore, by principle of induction $T_{smacG}\left(n\right) \geq \left(\frac{3}{2}\right)^n$ for all non-negative integers n.

■

Problem 6. State a loop invariant for the while loop at lines $15-17$ of this algorithm. (**References**: Lecture 3 and Tutorial Exercise 4)

1. n is an integer input such that $n \geq 4$
2. i is an integer variable such that $4 \leq i \leq n$
3. M is an integer array with length $n+1$
4. $M_j = fmacG_j$ for every integer j such that $0 \leq j \leq i$

Problem 7. Prove that your answer for the previous question really is a loop invariant for the while loop in this algorithm.

Proof. We will use Loop Theorem 1 to prove this loop invariant. (**References**: Lecture 3 and Tutorial Exercise 4)

1. The loop test at *line 15* has no side effects.

2. Consider the execution of the algorithm with the precondition for the "MacGonagall Mystery Computation" satisfied. There is nothing to be proved if the while loop at *line 15* is never reached, so if tests at *line 1, 3, 5, 7* fail, then *lines 9-15* are executed. By the inspection of code, we can say that the loop is not executed more than once. So it suffices us to consider the first execution of the loop.
   The precondition for the "MacGonagall Mystery Computation" states that n needs to be a non-negative integer i.e. $n \geq 0$ when the execution of the algorithm begins and by inspection, the value of n never changes. Furthermore, $n \neq 1, 2, 3$ as tests at *line 1, 3, 5 and 7* pass and the loop is never reached. Thus, $n \geq 4$ which establishes part 1 of the loop invariant when the execution of loop begins. One can see by the inspection of the code, i is an integer variable with a value of 4 when the execution of the loop begins, since $n \geq 4$, this establishes part 2 of the loop invariant when the execution of the loop begins. Since M is an *integer-array* variable whose length is set to be $n + 1$. Finally, since *lines 10, 11, 12, and 13* have been executed (and M is not changed at step 14), $M[0] = fmacG_0, M[1] = fmacG_1, M[2] = fmacG_2, and M[3] = fmacG_3$. Since $i = 4, M[j] = fmacG_j$ for every integer j such that $0 \leq i \leq j$. Thus, the fourth part of the assertion before the execution of the loop body, as required for the proof of the claim.

3. Consider an execution of the body of this loop that begins with the proposed *loop invariant* satisfied. It follows by *part 1* of the loop invariant that n is initially and integer variable such that $n \geq 4$ and since, the value of n is not changed during the executions of *line 16, 17*, this is also true when the execution of this loop ends. It follows by *part 2* of the loop invariant that i is an integer variable such that $4 \leq i \leq n$. Since, the test at *line 15* was checked and passed, $i < n$ so that $4 \leq i \leq n - 1$ at this point. When the value of n is never changed, the value of i is increased by one during the execution of *line 17*, so $5 \leq i \leq n$ at the end of the execution of this loop body and *part 2* of the loop invariant is satisfied at the end of this execution of the loop body as well. It follows *part 3* that M is an integer valued array with the length $n+1$ at the beginning of the execution of this loop. Since, n is not being changed during the execution loop, the value of M remains unchanged once the loop has been executed re-establishing *part 3*.From *part 4*, since, $i \geq 4$ and $i + 1 \geq 5$, $M[i] = M[i + 1]$ immediately after the execution of *line 16*, which means $M[i+1] = 2\times M[i]-2\times M[i-2]+M[i-3]$. However, since i is being incremented at *line 17*, $M[i] = 2 \times M[i - 1] - 2 \times M[i - 3] + M[i - 4]$. Since, $M_j = fmacG_j$ and $0 \leq j \leq i$, *part 4* of this loop invariant is satisfies again at the end of the execution as required.

It follows from "Loop Theorem # 1" that this is a loop invariant for this loop, as claimed.

■

Problem 8. Use this to prove that this algorithm is **partially correct**.

Proof. By inspection of the function signature for $fmacG(n)$, this algorithm has no undocumented side effects - that is it does not access or change inputs or global data, and does not create outputs unless documented in the "MacGongall Mystery Computation" problem.

It now suffices to show that if the $fmacG(n)$ is executed when the precondition for the "MacGongall Mystery Computation" is satisfied, then either:

(a) The execution of the algorithm halts, with the post-condition for the "MacGongall Mystery Computation" problem being satisfied when this occurs, or alternatively

(b) The algorithm runs forever, whereby the algorithm never halts at all

Now, consider an execution of this algorithm where the precondition of this problem is initially satisfied. That is, such that n is an integer input where $n \geq 0$.

- Suppose that $n = 0$. In this case, the test at line 1 passes, and the execution of the algorithm halts after reaching line 2, where $M_0 = M_n = 1$ is returned, as required to satisfy our condition established in (a).
- Suppose that $n = 1$. In this case, the test at line 1 fails, and the execution of the algorithm proceeds and passes the test at line 3, where the execution of the algorithm halts after the execution of line 4, and $M_1 = M_n = 0$ is returned, as required to satisfy our condition established in (a).
- Suppose that $n = 2$. In this case, the test at line 1 and 3 fail, and the execution of the algorithm proceeds and passes the test at line 5, where the execution of the algorithm halts after the execution of line 6, and $M_2 = M_n = 5$ is returned, as required to satisfy our condition established in (a).
- Suppose that $n = 3$. In this case, the test at line 1, 3, and 5 fail, and the execution of the algorithm proceeds and passes the test at line 7, where the execution of the algorithm halts after the execution of line 8, and $M_3 = M_n = 8$ is returned, as required to satisfy our condition established in (a).
- The only other case is when $n \geq 4$, such that the tests at lines 1, 3, 5, and 7 fail, and the steps at $9 \rightarrow 14$ are executed, which proceeds to the while loop that is later executed. Should the while loop run indefinitely, then the execution of the algorithm never halts, and the condition established in (b) is satisfied.

  If the algorithm does not halt during loop body execution, then the loop invariant is satisfied at this point of execution, such that n is an input integer where $n \geq 4$, and i is an integer variable such that $4 \leq i \leq n$ as established in (1) and (3) of the loop invariant. Since the algorithm halts, then the test at line 15 has been evaluated and thus failed, this means $i \leq n$ as well, that is $i = n$ after the execution of the loop body.

It can now be said for part 4 of the loop invariant that $M[j] = fmacG_j = M_n$, such that the nth MacGonagall has been returned as output following the return statement at line 18 being executed, whereby condition (a) has been satisfied.

■

Problem 9. The function $f(n, i) = n - (i - 1)$ is a bound function for the while loop in this algorithm. (**References**: L05 and T05)

Proof. Consider the above function f. We can show that f is a bound function by first inspecting to see if the function is an integer-valued total function of the inputs, variables and global data; as specified by a bound function's definition for a while loop. Then, since n is an integer input and i is an integer variabled, we can establish that f is indeed an integer-valued function.

Now, we must also show that the bound function satisfies condition a of its definition. That is, when the body of the while loop is executed, we note that the value of i is increased by one (at step 17) and the value of n is not changed. Thus, the value of f is decreased by at least one.

Lastly, we must show that the bound function also satisifies condition b of its definition. Then, we note that if the value of f is less than zero, then $n - (i - 1) < 0$, so that $i > n$ and the loop test (at line 15) fails, causing the execution of the loop to halt.

It then follows by definition of a bound function for a while loop, that f is indeed a bound function for the while loop in the f_{macG} algorithm, as claimed.

■

Problem 10. Suppose the $fmacG(n)$ algorithm is executed so that the precondition for the MacGonagall Mystery Computation problem is satisfied. The precondition of this problem assumes that is a non-negative integer before execution of the algorithm begins (i.e. $n \geq 0$.) (**References**: Lecture 3 and Tutorial Exercise 4) When n = 0, the test at line 1 passes, and the algorithm's execution halts after reaching line 2.

When n = 1, the test at line 1 fails, and the algorithm's execution passes the test at line 3, where its execution halts after the line 4.

When n = 2, the test at line 1 and 3 fail, and the algorithm's execution proceeds and passes the test at line 5, where execution halts after reaching line 6.

When n = 3, the test at line 1, 3, and 5 fail, and the algorithm's exeuction proceeds and passes the test at line 7, where execution halts after reaching line 8.

Otherwise, if $n \geq 4$, the test at lines 1, 3, 5, and 7 fail, and the steps at lines 9 $\rightarrow$ 14 execute, whereby the loop-body is then executed.

- At line 15, the loop-test compares integer input n and a counter variable i, therefore it has no side effects, and the loop-body halts after each execution.
- Since the body of the while loop is simply two assignment operations, every execution of the loop body halts as well.

- As mentioned in question (9), this while loop has a bound function.

It can now be proved using "Loop Theorem #2" that the execution of the while loop halts. The execution of this algorithm halts after the execution of the return statement at line 18, which establishes the claim as required. It now follows in question (11) that this algorithm is provably correct.

Problem 11. Using what has been proved so far, complete a proof that the fmacG algorithm correctly solves the MacGonagall Mystery Computation" problem.

Proof. According to the notes from Lecture 3, Claim #7 "If an algorithm, for a given computational problem, is both partially correct and terminates, whenever it is executed when its problems precondition initially satisfied, then this algorithm is correct".

The proof from **Problem 8** proves the partial correctness of the "MacGongall Mystery Computation" problem and the proof from **Problem 10** proves the termination of the "MacGongall Mystery Computation" problem.
Since, the algorithm is both partially correct and terminates, whenever it is executed it's precondition satisfied, it will be correct.

■

Problem 12. An upper bound function for $T_{fmacG}(n)$ in its closed form can be found by the following methods. (**References**: L05 and T06)

Proof. We will begin by considering the steps executed before the while loop. Namely, in the cases when $n = 0$, $n = 1$, $n = 2$, and $n = 3$.

If the algorithm is executed with input $n = 0$, then the test at line 1 is checked and passes. The execution then continues and ends with the return statement at line 2. Thus, the number of steps in this case is 2.

Next, suppose that the algorithm is executed with input $n = 1$, then the test at line 1 is checked and fails, then the test at line 3 is checked and passes and ends with the execution of the return statement at line 4. Thus, the number of steps in this case is 3.

Next, suppose that the algorithm is executed with input $n = 2$, then the tests at lines 1 and 3 are checked an all fail. The execution then continues to line 5 where the condition is checked and passes and ends with the execution of the return statement at line 6. Hence, 4 steps are are executed in this case.

Next, suppose that the algorithm is executed with input $n = 3$, then the tests at lines 1, 3, and 5 are checked an all fail. The execution then continues to line 7 where the condition is checked and passes and ends with the execution of the return statement at line 8. Hence, 5 steps are are executed in this case.

Suppose instead that $n \geq 4$. The tests at lines 1, 3, 5 and 7 are checked and all fail. The algorithm then continues and executes lines 9-14 before the while loop. Hence, 10 steps are executed before the while loop begins.

Given the while loops bound function, the initial value for this loop is $n-3$, that is, there are at most $n-3$ executions of the body of the while loop and at most $n-2$ executions of the loop test.

The loop body consists of two statements, at lines 16-17. There are no loops, tests or method calls, so every execution of the loop body includes two steps: For $4 \leq j \leq n-3$, the cost of the j^{th} execution of the loop body is

$$T_{body}(j) = 2$$

if there is actually a j^{th} execution of the body of the loop.

Similarly, the loop test consists of a single statement which does not call any other methods - so, for $1 \leq j \leq n-2$

$$T_{test}(j) = 1$$

if there is actually a j^{th} execution of the body of the loop.

It now follows that the total number of steps included in this execution of the loop is at most,

$$\begin{aligned}\sum_{j=1}^{n-3} T_{body}(j) + \sum_{j=1}^{n-2} T_{test}(j) &= \sum_{j=1}^{n-3} 2 + \sum_{j=1}^{n-2} 1 \\ &= 2(n-3) + (n-2) \\ &= 2n - 6 + n - 2 \\ &= 3n - 8\end{aligned}$$

There is only one more statement executed after this, at line 18, and the cost of this is 1.

Since $10 + (3n-8) + 1 = 3n + 3$, it now follows that the number of steps included in an execution of this algorithm on a non-negative integer input n is,

$$T_{fmacG}(n) = \begin{cases} 2 & n = 0 \\ 3 & n = 1 \\ 4 & n = 2 \\ 5 & n = 3 \\ 3n+3 & n \geq 4 \end{cases}$$

■

Problem 13.

```
package cpsc331.assignment1;

/**
 *
 * A class for the loop-based computation of the MacGonagall Mystery sequence
 *
 * @author Steven Ferguson (30037518)
 * @author Daniel Contreras (10080311)
 * @author Guransh Mangat (30061719)
 * @version 1.0
 *
 */

public class FMacGonagall {

    /**
     *
     * The main method for this class.
     *
     * @param args A single non-negative integer n.
     *
     */

    public static void main(String[] args) {

        if (args.length != 1) {
            System.out.println("Fiddlesticks! One integer input is required.");
            return;
        }

        try {
            int n = Integer.parseInt(args[0]);
            System.out.println(fmacG(n));
        } catch (NumberFormatException e) {
            System.out.println("Fiddlesticks! One integer input is required.");
        } catch (IllegalArgumentException e) {
            System.out.println("Fiddlesticks! The integer input cannot be negative.");
        }

    }

    /**
     *
     *  Computes the nth MacGonagall Number M_n, throwing
     *  an IllegalArgumentException if n is negative.
     *
     * Precondition:
     * An integer n given as input.
     *
     * Postcondition:
     * The nth MacGonagall Number, M_n is returned as output, and
     * an IllegalArgumentException is thrown if n < 0.
     *
     * @param n the MacGonagall Number to be computed
     * @return the nth MacGonagall Number, M_n
     * @throws IllegalArgumentException if the input is negative
     *
     */

    public static int fmacG(int n) throws IllegalArgumentException {

        // Assertion: n is an integer such that n >= 0

        if (n < 0) {
            throw new IllegalArgumentException();
        }

        switch (n) {
            case 0:
                return 1;
            case 1:
                return 0;
            case 2:
                return 5;
            case 3:
                return 8;
            default:

                int[] M = new int[n + 1];
                M[0] = 1;
                M[1] = 0;
                M[2] = 5;
                M[3] = 8;
                int i = 4;

                // Loop Invariant:
                // 1. n is an integer such that n >= 4
                // 2. i is an integer variable such that 4 <= i <= n
                // 3. M is an integer array with length n + 1
                // 4. M_j = fmacG_j for every integer j such that 0 <= j <= i
                //
                // Bound Function: n - (i - 1)
```

```
                while (i <= n) {
                    M[i] = 2 * M[i - 1] - 2 * M[i - 3] + M[i - 4];
                    i += 1;
                }

                return M[n];
        }

    }
}
```

Problem 14. Suppose that M is a function of the non-negative integers such that, for every integer $n \geq 0$,

$$M(n) = \begin{cases} 1 & i = 0 \\ 0 & i = 1 \\ 5 & i = 2 \\ 8 & i = 3 \\ 2M_{i-1} - 2M_{i-3} + M_{i-4} & i \geq 4 \end{cases}$$

Then, the closed form formula is $M(n) = n^2 + (-1)^n$.

Proof. This will be established by induction on n. The strong form of mathematical induction will be used and the cases $n = 0$, $n = 1$, $n = 2$, and $n = 3$ will be considered in the basis.

Basis. If $n = 0$, then $M(n) = M(0) = 1$, and

$$\begin{aligned} M(n) &= n^2 + (-1)^n \\ M(0) &= 0^2 + (-1)^0 \\ &= 1 \end{aligned}$$

If $n = 1$, then $M(n) = M(1) = 0$, and

$$\begin{aligned} M(n) &= n^2 + (-1)^n \\ M(1) &= 1^2 + (-1)^1 \\ &= 0 \end{aligned}$$

If $n = 2$, then $M(n) = M(2) = 5$, and

$$\begin{aligned} M(n) &= n^2 + (-1)^n \\ M(2) &= 2^2 + (-1)^2 \\ &= 5 \end{aligned}$$

If $n = 3$, then $M(n) = M(3) = 8$, and

$$\begin{aligned} M(n) &= n^2 + (-1)^n \\ M(3) &= 3^2 + (-1)^3 \\ &= 8 \end{aligned}$$

Inductive Step. Let $k \geq 4$ be an integer. It is necessary and sufficient to use the following

Inductive Hypothesis: $M_i = i^2 + (-1)^i$ for every integer i such that $0 \leq i \leq k$.

to prove the following

Inductive Claim: $M_{k+1} = (k+1)^2 + (-1)^{k+1}$

Now,

$$\begin{aligned} M_{k+1} &= 2M_k - 2M_{k-2} + M_{k-3} \\ &= 2(k^2 + (-1)^k) - 2((k-2)^2 + (-1)^{K-2}) + (k-3)^2 + (-1)^{k-3} && \text{(from IH)} \\ &= 2k^2 + 2(-1)^k - 2(k-2)^2 - 2(-1)^{k-2} + (k-3)^2 + (-1)^{k-3} \\ &= \big(k^2 + 2k + 1\big) + (-1)^k[2 - 2(-1)^{-2} + (-1)^{-3}] \\ &= (k+1)^2 + (-1)^{k+1} \end{aligned}$$

thus establishing the inductive claim, as required. Thefore, by principle of induction, $M(n) = n^2 + (-1)^n$ for all non-negative integers n.

■