Queen's University, Faculty of Arts and Science, School of Computing
CISC 223 Practice Exam, April 2024 (Instructor: Kai Salomaa)

INSTRUCTIONS

- **Aids allowed:** You may bring in one 8.5 × 11 inch sheet of paper containing notes, and use it during the exam. The sheet can be written/printed on both sides.
- This examination is THREE HOURS in length. Answer all 10 questions.
- **Answer each question in the space provided** (on the question paper). There are two extra pages at the end of the exam, if more space is needed. **Please write legibly.**

PLEASE NOTE: Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer exam questions as written.

STUDENT NUMBER: One digit in each square, please!

STUDENT NUMBER (written in words):

Problem 1	/4
Problem 2	/3
Problem 3	/3
Problem 4	/6
Problem 5	/6
Problem 6	/5
Problem 7	/6
Problem 8	/6
Problem 9	/5
Problem 10	/6
Total	/50

1. Consider the context-free grammar

$$S \longrightarrow 0S1 \mid 00S \mid S11 \mid \varepsilon$$

Here S is the start nonterminal, and 0, 1 are terminals.

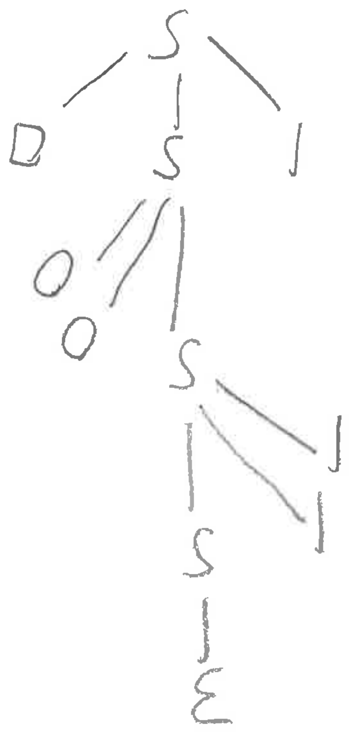
(i) Give a parse tree for the string 000111.

(ii) Give a parse tree for the string 001111.

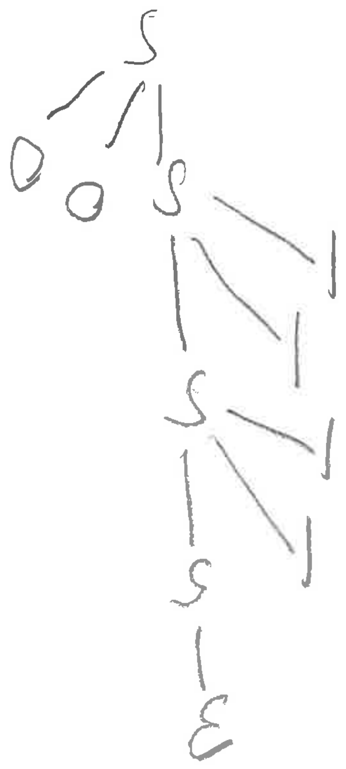
(iii) Is the above grammar ambiguous? Justify your answer.

(iv) Is the language generated by the grammar regular? Circle the correct answer: [✓ YES] NO (The answer to (iv) does not require an explanation.)

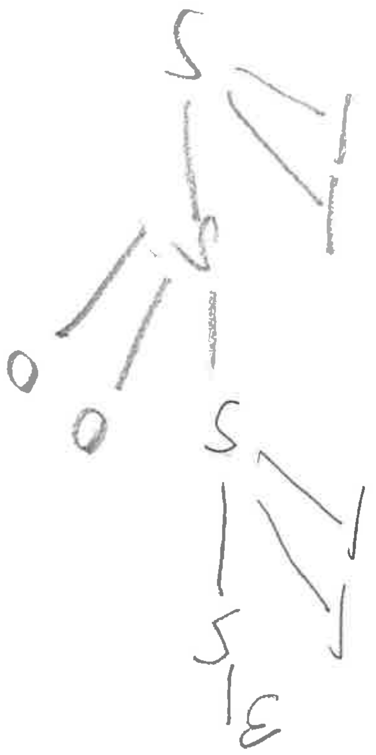
i)



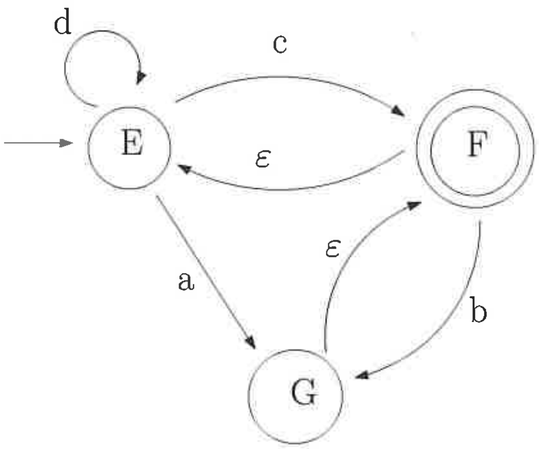
ii)

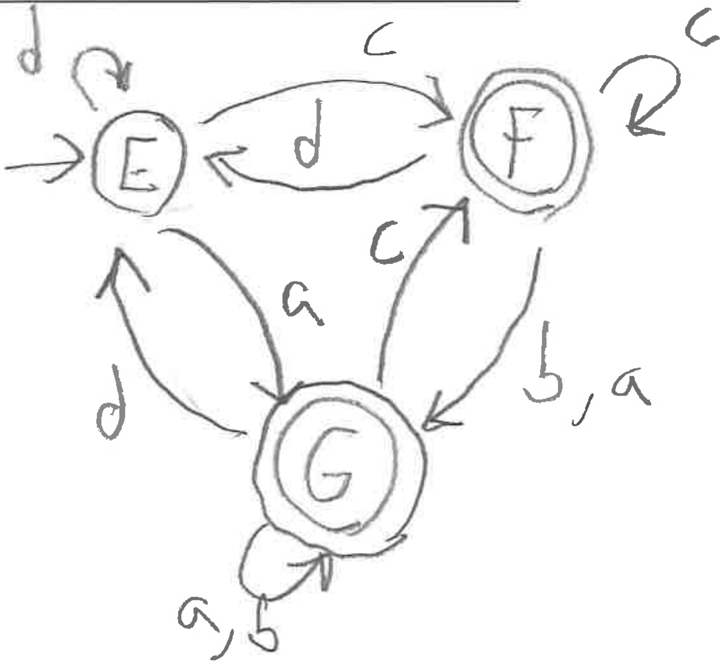


iii) Yes, 001111 has also parse tree

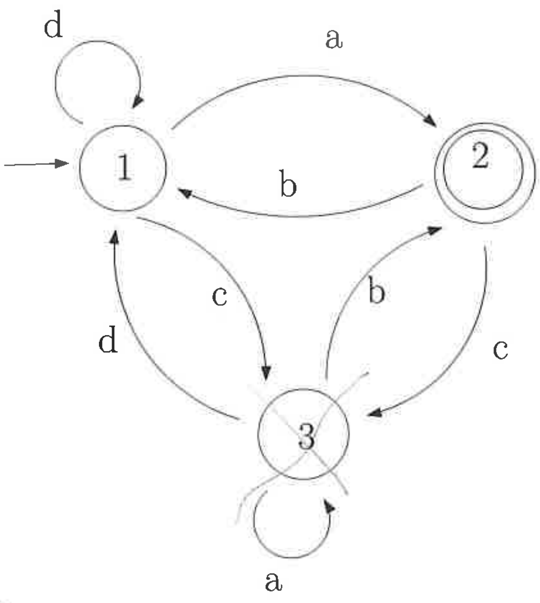


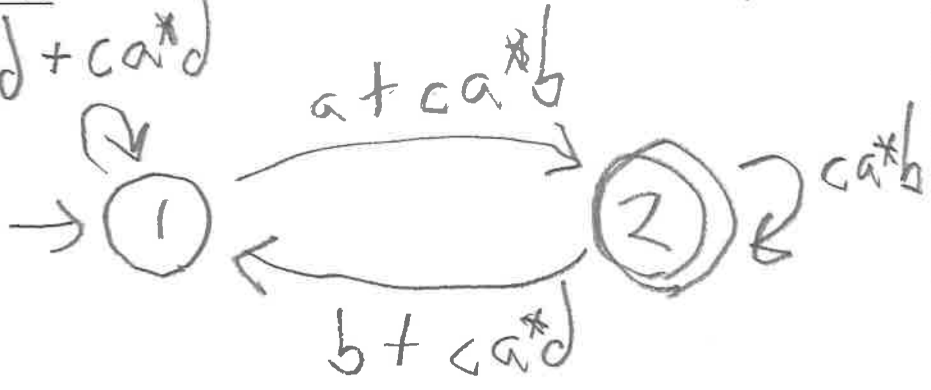
2. Using the **systematic method described in the course**, convert the below state diagram with ε-transitions into an equivalent state diagram without ε-transitions. The alphabet is $\Sigma = \{a, b, c, d\}$.





3. In this question $\Sigma = \{a, b, c, d\}$. Using the systematic method described in the course convert the below state diagram into an equivalent **regular expression**. Your answer should include the intermediate step(s) used in the construction.





Regex:

$$(d+ca^*d)^*(a+ca^*b)\left[ca^*b + (b+ca^*d)(d+ca^*d)^*(a+ca^*b)\right]^*$$

4. (i) (2 marks) Using left-factoring and/or elimination of left-recursion give grammars equivalent to the below two grammars where the immediate problems preventing use of recursive-descent parsing have been removed. Capital letters denote variables and the set of terminals is $\{a, b, c, d\}$.

(a) $S \rightarrow bcbSb \mid bcaSa \mid bcda \mid caa$

$S \rightarrow bcS' \mid caa$

$S' \rightarrow bSb \mid aSa \mid da$

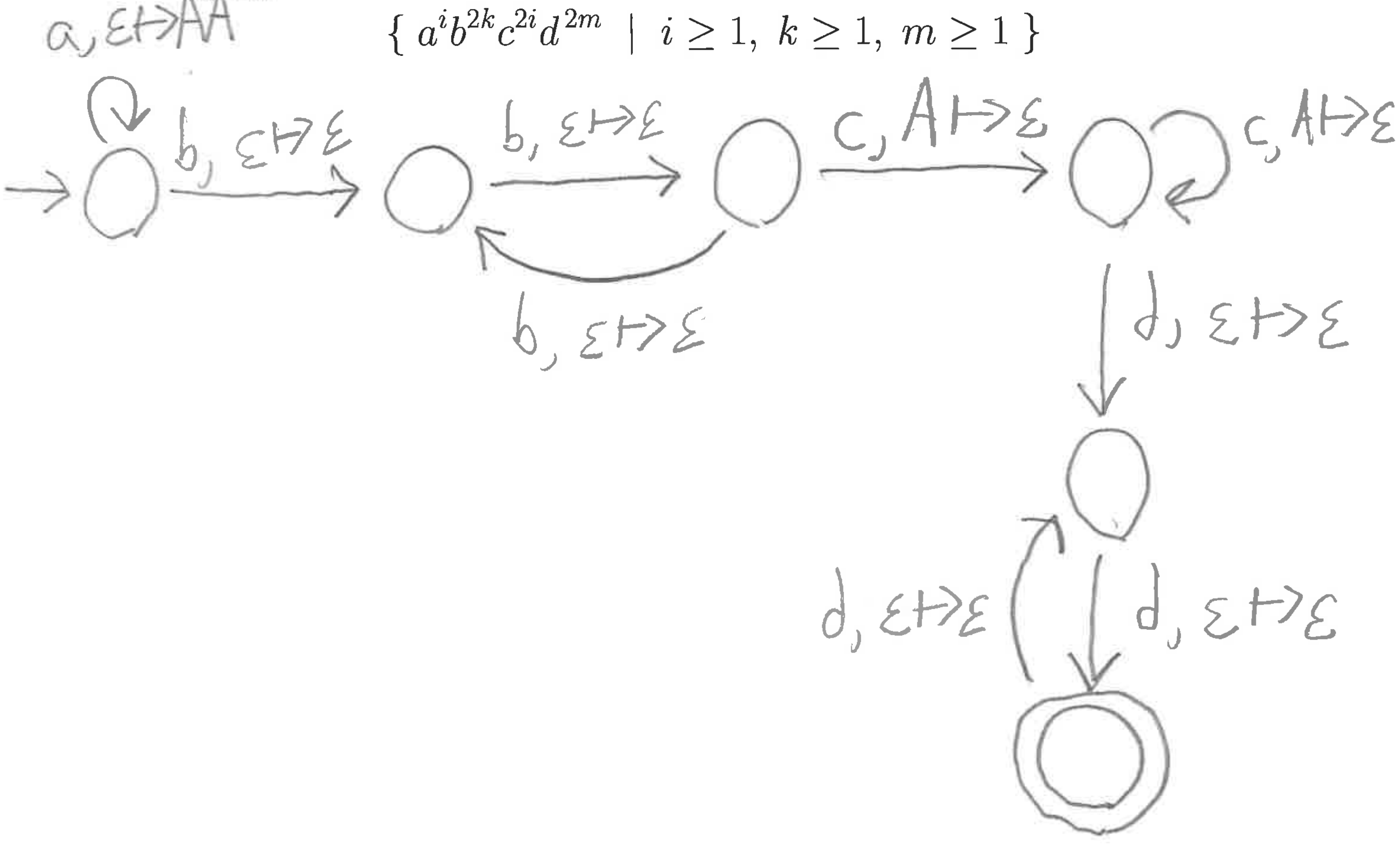
(b) $S \rightarrow Sbc \mid cda \mid Sad \mid Sdb \mid \varepsilon$

$S \rightarrow cdaS' \mid S'$

$S' \rightarrow bcS' \mid adS' \mid dbS' \mid \varepsilon$

(ii) (4 marks) Give a **deterministic pushdown automaton** that recognizes the language

$$\{\, a^i b^{2k} c^{2i} d^{2m} \mid i \geq 1,\ k \geq 1,\ m \geq 1 \,\}$$



5. Verify the validity of the following correctness statements **by adding all the intermediate assertions** (that is, give the proof tableau). All variables are of type `int`. Clearly state any mathematical facts and inference rules used.

   (i)
   ```
   ASSERT( y == 0 && z <= -2 )
   z = z - x;
   y = y - z;
   x = y - x;
   ASSERT( x >= 2 && y + z < 5 )
   ```
   Solution:
   ```
   ASSERT( y == 0 && z <= -2 )
   // y==0 implies y < 5
   // (y==0 && z<=-2) implies y-z >= 2
   ASSERT( y - z + x - x >= 2 && y < 5 )
   z = z - x;
   ASSERT( y-z-x >= 2 && y-z+z < 5 )
   y = y - z;
   ASSERT( y-x >= 2 && y+z < 5 )
   x = y - x;
   ASSERT( x >= 2 && y + z < 5 )
   ```

   (ii)
   ```
   ASSERT( x >= 5 || ( x <= 0 && y == 2 ) )
   z = x - y;
   y = y + z;
   x = y - z;
   ASSERT( x == 2 || y > 3 )
   ```
   Solution:
   ```
   ASSERT( x >= 5 || ( x <= 0 && y == 2 ) )
   // x >= 5 implies x > 3
   //  x <= 0 && y == 2 implies y == 2
   ASSERT( y == 2 || y + x - y > 3 )
   z = x - y;
   ASSERT( y + z - z == 2 || y + z > 3 )
   y = y + z;
   ASSERT( y-z == 2 || y > 3 )
   x = y - z;
   ASSERT( x == 2 || y > 3 )
   ```

6. For each of the following sets of strings over the alphabet $\Sigma = \{0,1\}$, give a **regular expression** that denotes it. Below "or" always means "inclusive or".

   (i) All strings over $\{0,1\}$ that have a substring 000 or have a prefix 111.
   Regular expression:

   $(0+1)^*000(0+1)^* + 111(0+1)^*$

   (ii) All strings over $\{0,1\}$ that have prefix 011 and have suffix 110. Note that the prefix and suffix may overlap.
   Regular expression:

   $011(0+1)^*110 + 0110 + 01110$

   (iii) All strings over $\{0,1\}$ that have an odd number of occurrences of the symbol 1 (and any number of 0's).
   Regular expression:

   $0^*1(0+10^*1)^*$

   (iv) All strings over $\{0,1\}$ that do not have 000 as a substring.
   Regular expression:

   $(1+01+001)^* \cdot (0+00+\varepsilon)$

   (v) All strings over $\{0,1\}$ of even length that have 001 as a substring.
   Regular expression:

   $(00+01+10+11)^*001(0+1)(00+01+10+11)^*$
   $+(00+01+10+11)^*(0+1)001(00+01+10+11)^*$

7. Are the following languages A and B context-free or not context-free?

   - If a language is context-free, give a context-free grammar that generates it.
   - If a language is not context-free, prove that it is not context-free.

   (i) $A = \{\, a^i b^k c^\ell d^m \mid i = k + m,\ i \geq 0,\ k \geq 0,\ \ell \geq 0,\ m \geq 0 \,\}$

   (ii) $B = \{\, d^i c^{2k} b^i a^m \mid i \geq k \geq 0,\ m \geq 0 \,\}$

i) Grammar for A:

$$S \rightarrow aSd \mid XY$$
$$X \rightarrow aXb \mid \varepsilon$$
$$Y \rightarrow cY \mid \varepsilon$$

ii) We show that B is not context-free. Assume B is context-free and let p be the constant given by PL. Choose $s = d^p c^{2p} b^p$. According to PL we can write $s = uvwxy$ where the parts u, v, w, x, y satisfy conditions of PL.

1. ~~If~~ v or x contains more than one type of symbol, then $uv^2wx^2y \notin d^*c^*b^*a^*$ and is not in B.

2. In the following the v and x each contain at most one type of symbol.

2a) Assume v or x contains d's or b's. Since $|vwx| \leq p$, vx cannot contain both d's and b's and uv^2wx^2y does not have same numbers of d's and b's.

2b) The remaining possibility is the vx has only c's. Now the number of c's in uv^2wx^2y is more than two times the number of d's. Again $uv^2wx^2y \notin B$.

8. For each question, **circle one answer**. If you circle more than one answer, it will be considered a wrong answer. If in doubt, it is to your advantage to make a guess.

   (i) Let L be the language consisting of all strings over the alphabet $\{b, c\}$ having an equal number of b's and c's. The language L is denoted by the regular expression:

   (a) $(b+c)^* + (bb+cc)^*$

   (b) $(bc + cb + bbcc + bcbc + bccb + cbbc + cbcb + ccbb)^*$

   (c) $((bc+cb)^* + (bbcc + bcbc + bccb + cbbc + cbcb + ccbb)^*)^*$

   d [✓ (d) None of the above.]

   (ii) Consider a context-free grammar that has productions $S \to \alpha \mid \beta$ where α derives the empty string. Which of the below conditions always prevents the use of predictive recursive-descent parsing with this grammar:

   (a) $\text{FIRST}(S) \cap \text{FIRST}(\alpha) \neq \emptyset$

   (b) $\text{FOLLOW}(S) \cap \text{FIRST}(\alpha) \neq \emptyset$

   (c) $\text{FIRST}(S) \cap \text{FIRST}(\beta) \neq \emptyset$

   d [✓ (d) $\text{FOLLOW}(S) \cap \text{FIRST}(\beta) \neq \emptyset$]

   (e) None of the above.

   (iii) Consider the correctness statement: `P { x = 3x - 5; } x >= 0`
   where `x` has type integer. The correctness statement is valid when `P` is the assertion:

   a [✓ (a) `false`]

   (b) `x >= 1`

   (c) `x <= 1`

   (d) None of the above.

   (iv) Consider an inference rule for correctness statements: $\frac{Q_1\{C\}P_1 \qquad Q_2\{C\}P_2}{Q_1 \,\|\, Q_2\ \{C\}\ P_1 \,\|\, P_2}$. This inference rule is:

   a [✓ (a) Generally valid.]

   (b) Valid only if C has no assignments to variables used in the assertions P_1 and P_2.

   (c) Valid only if C does not interfere with variables used in the assertions Q_1 and Q_2. .

   (d) None of the above.

   (v) The Church-Turing thesis states that

   (a) The halting problem cannot be solved using programming languages (such as C) but it can be solved using Turing machines.

   (b) Turing machines cannot solve the halting problem.

   (c) Functional programming languages can implement certain functions that cannot be computed by Turing machines.

   d [✓ (d) None of the above.]

   (vi) Let A be a solvable algorithmic problem. The following is true:

   (a) A is not reducible to any unsolvable algorithmic problem.

   b [✓ (b) A is reducible to all unsolvable algorithmic problems.]

   (c) A is reducible to some, but not to all, unsolvable algorithmic problems.

   (d) None of the above.

9. (i) (2 marks) What should the pre-condition `P` be in each of the following correctness statements for the statement to be an instance of Hoare's axiom scheme? All variables are of type `int`.

(a) `P { z = x + y; } Exists(x = 0; x < z) 2*x + z >= 2*y + t`

Exists(a=0, a<x+y) 2*a + x+y >= 2*y+t

(b) `P { z = x+y; } Exists(x=0; x < 50) ForAll(y=0; y < z) x+2*w >= 2*y+z`

Exists(a=0, a<50) ForAll(b=0, b<x+y) a+2*w >= 2*b + x+y

(ii) (3 marks) Use the array-component assignment axiom (two times) to find the most general sufficient pre-condition `P` for the following code fragment:

```
ASSERT(P) /*determine what is P*/
A[j] = 3;
A[k] = x+2;
ASSERT( A[m] > A[k] )
```

Above `A` is an array of integers, `x` is an integer variable and we assume that all the subscripts are within the range of subscripts for `A`.

Write the assertion `P` first using the notation from the array-component assignment axiom, and then rewrite `P` in a logically equivalent form that does not contain any notation $(A \mid I \mapsto E)$. Show your work.

Solution:

```
ASSERT( ((A | j +--> 3 ) | k +--> x+2 )[m] > x+2 )
A[j] = 3;
ASSERT( A | k +--> x+2 )[m] > x + 2 )
ASSERT( A | k +--> x+2 )[m] > ( A | k +--> x+2 )[k] ) //right side is x+2
A[k] = x+2;
ASSERT( A[m] > A[k] )

Rewrite the pre-condition:
m == k: x+2 > x+2, or, false
m != k: m == j: 3 > x+2
        m != j: A[m] > x+2

Precondition is
(m != k && m==j && 1 > x) || ( m != k && m != j && A[m] > x+2 )
```

10. Assume a declarative interface where `n` and `max` are constant integers. Also `A` is an array of integers and we know that the entries in the segment `A[0:max]` are defined. Consider the following (partial) correctness statement:

```
ASSERT( 1 <= n < max )
{ int i; i = 1;
  A[0] = 1;
  while( i < n ) { A[i] = A[i-1] + 4*i + 3;
                   i = i+1;
                 } //end while
}
ASSERT(ForAll(k = 0; k < n) A[k] == 2*k*k + 5*k + 1)
```

Choose a loop invariant and give a **complete proof tableau** by adding all the intermediate assertions. Be sure to **clearly indicate what is your loop invariant.** Also state any mathematical facts used. Does the loop terminate? Explain your answer.

invariant I: 1 <= i <= n &&
ForAll(k = 0, k<i) A[k] == 2*k*k + 5*k + 1

Question 10 solution

```
invariant I: 1 <= i <= n && ForAll(k = 0; k < i) A[k] == 2*k*k + 5*k + 1

ASSERT( 1 <= n < max )
{ int i; // variable declaration does not affect reasoning
ASSERT( 1 <= n && 1 == 2*0*0 + 5*0 + 1 )
// with k == 0 the ForAll yields 1 == 1, i.e., true
ASSERT( 1 <= 1 <= n && ForAll(k = 0; k < 1) (A | 0 +-> 1)[k] == 2*k*k+5*k+1 )
  i = 1;
ASSERT( 1 <= i <= n && ForAll(k = 0; k < i) (A | 0 +-> 1)[k] == 2*k*k+5*k+1 )
  A[0] = 1;
ASSERT(I)
  while( i < n ) { ASSERT( I && i < n )
// i < n implies i+1 <= n when i, n are integers
// I implies the below ForAll with values k = 0, ..., i - 1
// According to invariant A[i-1] == 2*(i-1)*(i-1) + 5*(i-1) + 1
// == 2*i*i + i - 2. Thus with k==i the equality in below ForAll is
// A[i-1] + 4*i + 3 == 2*i*i + i - 2 + 4*i + 3 == 2*i*i +5*i + 1
ASSERT( 1 <= i+1 <= n && ForAll(k = 0; k < i+1)
                (A | i +-> A[i-1] + 4*i +3)[k] == 2*k*k + 5*k + 1 )
        A[i] = A[i-1] + 4*i + 3;
ASSERT( 1 <= i+1 <= n && ForAll(k = 0; k < i+1) A[k] == 2*k*k + 5*k + 1 )
        i = i+1;
ASSERT(I)
                  } //end while
}
ASSERT( I && i >= n )
// i <= n && i >= n implies i == n
// with i == n, the invariant yields the postcondition
ASSERT(ForAll(k = 0; k < n) A[k] == 2*k*k + 5*k + 1)
```

$2i^2 - 4i + 2 + 5i - 5 + 1$

Termination: According to invariant $1 \le i \le n$.
The loop terminates because each iteration increments i by one.