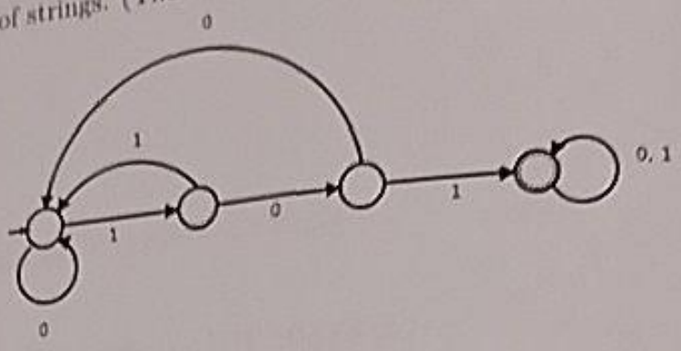
Question 1 *[30%]*

(i) Give a deterministic finite automaton for the set of binary strings that do not contain three consecutive 1s. (8%)

(ii) Describe in English the set of strings accepted by the automaton depicted below. Give a regular expression for the set of strings. (The accept state is the one on the right.)



(iii) Give a regular expression for the set of binary strings that do not contain three consecutive 1s (6%)

(iv) Construct a finite automaton equivalent to the following regular expression $(11(0|00))^*$. (7%)

Question 2 *[20%]*

(i) The following grammar captures the syntax of a subset of the programming language Pascal. (9%)

⟨program⟩ → **PROGRAM ID ;** ⟨compound-statement⟩ **.**
⟨compound-statement⟩ → **BEGIN** ⟨optional-statements⟩ **END**
⟨optional-statements⟩ → ⟨statement-list⟩
| ϵ
⟨statement-list⟩ → ⟨statement⟩ **;** ⟨statement-list⟩
| ϵ
⟨statement⟩ → ⟨variable⟩ **:=** ⟨expression⟩
| ⟨compound-statement⟩
| **IF** ⟨expression⟩ **THEN** ⟨statement⟩ **ELSE** ⟨statement⟩
| **WHILE** ⟨expression⟩ **DO** ⟨statement⟩
⟨variable⟩ → **ID**
⟨expression⟩ → **EXPR**

The non-terminals are enclosed within <>, while the terminals (reserved words PROGRAM, BEGIN, END, IF, THEN, ELSE, WHILE, DO; place-holders ID and EXPR and symbols ';', '.' and ':=') are shown in boldface.

Give a complete parse tree of the following source program. (Treat 1, n <= 10 and $n + 1$ as EXPRs.)

```
PROGRAM X;
BEGIN
n := 1;
WHILE n <= 10 DO
   n := n + 1
END.
```

(10

(*ii*) Modify the grammar to allow for some basic I/O using the READ and WRITE statements with the syntax illustrated below.

```
READ(x); READ(a, b, c); WRITE(y); WRITE(17); WRITE(d, e, 123)
```

The READ statement involves the reserved word "READ" followed by a comma separated list of one or more identifiers enclosed in parentheses. The WRITE statement involves the reserved word "WRITE" followed by a comma separated list of one or more variables or integer constants enclosed in parentheses.

(*10%*)

Question 3 *[35 %]*

(*i*) Write a brief summary of general principles of the recursive descent parsing technique. (*5%*)

(*ii*) List the FIRST sets of the non-terminals of the Pascal grammar shown above in Question 2 (i). (*10%*)

(*iii*) Sketch, in pseudocode or Java, a recursive descent parser for the Pascal subset presented in Question 2 (i) above. Pay particular attention in your description to the methods for productions `optional-statements`, `statement-list` and `statement`; you may summarize the other methods in prose.

You may assume that a lexical scanner is available that includes a method called nextToken to read through the source program one token at a time. State carefully any other assumption your description relies upon. (*20%*)

Question 4 *[15 %]*

(*i*) Write a brief note sketching how a parse/syntax tree might be used to facilitate a syntax-directed translation of a Pascal program into three-address code (TAC). You are not required to draw a tree, merely to describe in words how a tree might drive the translation process. (A summary of TAC are appended to this paper.) (*5%*)

(*ii*) Show a complete translation of the following Pascal fragment into TAC, indicating clearly the connection between the various source constructs and the corresponding TAC. You may assume that the Pascal if, while and assignment statements share the same semantics as their Java equivalents and that Pascal's BEGIN-END play the same role as Java's {} in grouping statements together. (*10%*)

```
IF n >= 1 THEN
   BEGIN
   i := 1;
   p := 0;
   c := 1;
   WHILE i < n DO
      BEGIN
      c := c + p;
      p := c - p;
      i := i + 1
      END
   END
ELSE
   BEGIN
   END
```