Question 1 /30%/

(i) The C programming language defines the lexical syntax of floating constants of an integer part, a decimal point, a fraction of the constant consists of an integer part. The integer part is a fraction of the constant consists of an integer part. The C programming language defines the textern part, a decimal point, a fraction follows. A floating constant consists of an integer exponent. The integer and fraction follows. A floating constant consists of an integer exponent. The C programming constant consists of in integer exponent. The integer and fraction follows. A floating constant consists of in integer exponent. The integer and fraction part, an E and an optionally signed integer exponent. Either the integer part or the fraction part, an E and an optionally sequence of digits. part, an E and an optionally signed integer the integer part or the fraction parts both consist of a sequence of digits. Either the integer part or the E and the expension that may be missing; either the decimal point or the E and the expension that part, an E and the sequence of digits. Established part of the fraction parts both consist of a sequence of digits. Established part (not both) may be missing: either the decimal point or the E and the exponent part (not both) may be missing. Give a regular expression that captures this parts both countries and the exponent part (not both) may be missing. Give a regular expression that captures this set of (not both) may be missing. (n) Give a deterministic finite automaton for the language of Part (i). (10%)

(ii) Give a deterministic finite (iii) The characters /* introduce a C comment, which terminates with the characters (iii) The characters /* introduce a C comment, which terminates with the characters (iii) The characters /* introduce a C comment, which terminates with the characters (iii) The characters /* introduce a C comment, which terminates with the characters (iii) The characters /* introduce a C comment, which terminates with the characters (iii) The characters /* introduce a C comment, which terminates with the characters (iii) The characters /* introduce a C comment, which terminates with the characters (iii) The characters /* introduce a C comment, which terminates with the characters (iii) The characte

*/. Comments do not nest, so this string is not legal:

/* Not /* a */ comment.*/

Give a nondeterministic finite automaton that accepts the set of strings that are valid comments according to this definition.

Question 2 [30%]

(i) Consider the following simplified grammar for the Tiny programming language.

```
\langle program \rangle \rightarrow \langle stmtseq \rangle
\langle \text{stmtseq} \rangle \rightarrow \langle \text{statement} \rangle \langle \text{stmtseq2} \rangle
\langle stmtseq2 \rangle \rightarrow ; \langle stmtseq \rangle \mid \epsilon
\langle statement \rangle \rightarrow \langle ifstmt \rangle \mid \langle repeatstmt \rangle \mid \langle assignstmt \rangle
(ifstmt) → if (exp) then (stmtseq) end
(repeatstmt ) → repeat (stmtseq) until (exp)
(assignstmt) \rightarrow id := (exp)
(exp) \rightarrow num
```

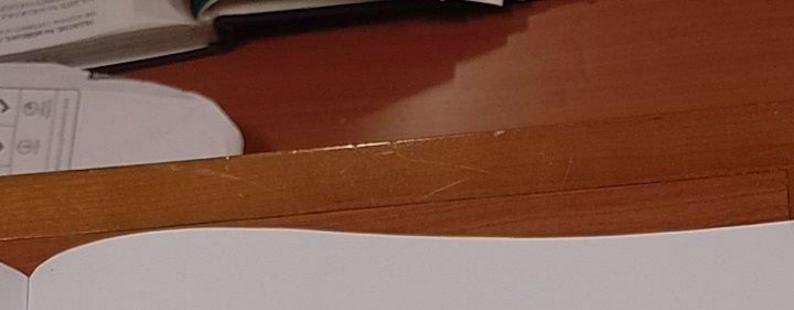
Give a complete parse tree for the following Tiny program.

x := 17;if 1 then y := 23 end; z := 29

(7%) (ii) Give a context-free grammar for the language consisting of strings of the form 0"1" over the alphabet {0, 1} i.e strings containing some number of zeros followed by the same number of ones. (3%)

(iii) Give a context-free grammar for the language consisting of strings over the alphabet $\{0,1\}$ that contain precisely the same number of zeros and ones. (14%)

(iv) Draw a complete parse tree for each of the following strings: (a) 000111. (b) 010101 and (c) 00111100 (6%)



Question 3 [30 %]

- (i) Compute the FIRST and FOLLOW sets for the nonterminals of the Tiny grammar given above. (14 marks)
- (ii) Complete the parse table for this language for the stack-based, table-driven LL(1) parsing algorithm. (10 marks)
- (iii) Write a succinct description of the stack-based, table-driven LL(1) parsing algorithm.
 (6%)

Question 4 [10 %]

(i) Suppose we wish to augment the Tiny language with a simple for-loop with the syntax shown below.

```
\begin{array}{ll} \text{for } v := s \text{ to f do} \\ \text{begin} \\ \text{loop body} \\ \text{end} \end{array} \qquad \begin{array}{ll} \text{total } := 0; \\ \text{for } i := 1 \text{ to } 100 \text{ do} \\ \text{begin} \\ \text{total } := \text{total } + i \end{array}
```

In the template shown above left, v is an integer variable (the loop variable), s and f are integer expressions specifying the first and last values for the loop variable and "loop body" is a placeholder for any sequence of statements bracketed by the keywords begin and end. The intended interpretation is that the loop body should be executed with variable v taking on each value in the sequence from s to f in turn inclusive.

Show how the grammar of Question 2 may be modified to incorporate this feature.

(5%)

(ii) Give a general template for the translation of any for loop into three-address code. Use the example given above right to illustrate the approach. (5%)