THE CO-OPERATIVE UNIVERSITY OF KENYA

END OF SEMESTER EXAMINATION DECEMBER -2022

EXAMINATION FOR THE DEGREE OF BACHELOR OF SCIENCE IN COMPUTER SCIENCE
(YR III SEM I)

UNIT CODE: BCSC 3120

UNIT TITLE: COMPILER CONSTRUCTION AND DESIGN

DATE: FRIDAY, 16TH DECEMBER, 2022

TIME: 11:30 AM – 1:30 PM

INSTRUCTIONS:

- **Answer question ONE (compulsory) and any other TWO questions**

QUESTION ONE [30 MARKS]

(a) Define the following terms: [4 Marks]

(i) Compiler

(ii) Interpreter

(iii) Assembler

(iv) Parser

(b) Name four different types of compilers, and provide a real-world example for each that you name [6 Marks]

(c) Explain the difference between:

i) the *syntax* of a language and its *semantics.* [4 Marks]

ii) a context-free grammar and a context-sensitive grammar [4 Marks]

(d) Why do semantic routines need to be placed at the end of productions in LR grammars? [2 Marks]

(e) Write a regular expression for all strings of **as** and **bs** which contains the substring **abba**. [2 Marks]

(f) Write an algorithm for non-recursive predictive parsing. [4 Marks]

(g) "A good choice of compiler implementation language is important". Using a suitable programming language, explain in support of above statement. [4 Marks]

QUESTION TWO [20 MARKS]

a) "A *finite language* is a language with a finite number of strings. For example, the language with only the string *a*, *ba*, and *bba* is finite, while the languages in parts 1,2, and 3 above are not finite".

i) Do you agree that all finite languages are regular? [1 Mark]

ii) If so, explain why, and if not, give an example of a finite language which is not regular. [2 Marks]

b) Below are the following three separate grammars:

$A \rightarrow bAc \mid \in$ (1)

$A \rightarrow bAb \mid b$ (2)

$A \rightarrow bAb \mid c$ (3)

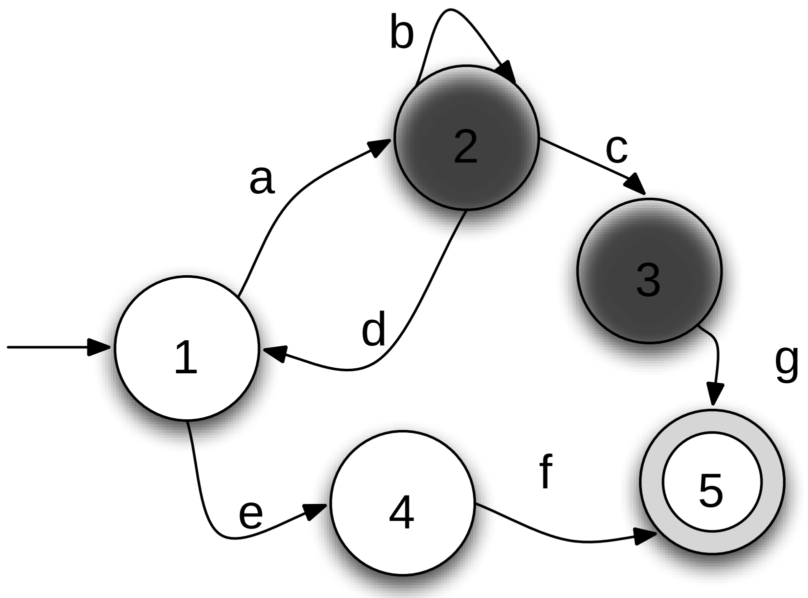
Symbol A is the start symbol and the only non-terminal, and b and c are terminals.

For the all the three grammars:

i) Calculate the First-set and Follow-set of A [4 Marks]

ii) After extending the grammar with a new start symbol and production $A' \rightarrow A$, draw the LR (0) – DFA. [6 Marks]

c) Convert the following NFA to a regular expression: [7 Marks]



QUESTION THREE **[20 MARKS]**

a) Using suitable examples in each phase, explain the phases of compiler design. [10 Marks]

b) Write a computer program to implement the design of a lexical analyzer able to recognize tokens defined by a given grammar. [10 Marks]

QUESTION FOUR **[20 MARKS]**

Consider the program of 3-address intermediate code given below. Study it and use it to answer the questions that follows.

a) Indicate where new *basic blocks start*. For each basic block, give the line number such that the instruction in the line is the first one of that block. [4 Marks]

```
a := input
b := input
t1 := a + b //line 3
t2 := a * 2
c := t1 + t2
if a < c goto 8
t2 := a + b
b := 25
c := b + c
d :=  a – b
if 12 = 0 goto 17
d := a + b
t1 := b – c
c := d – t1
if c < d goto 3
c : a + b
output c   //line 17
output d
```

b) The developer who is responsible for generating the intermediate TA-code assures that temporary variables in the generated code are *dead* at the end of each basic block as well as dead at the beginning of the program, even if the same temporary variable may well be used in different basic blocks. Formulate a general rule to *check* locally in a basic block whether or not the above claim is honored or violated in a given program. Assume that all variables are dead after the last instruction. [4 Marks]

c) Use the rule formulated in the previous sub-problem on the TA-code given, to check if the condition is met or not. The temporary variables are called t_1, t_2 etc. in the code. [4 Marks]

d) Draw the control flow graph of the problem and find the values for *inLive* and *outLive* for each basic block. Consider the temporaries as ordinary variables. [8 Marks]

QUESTION FIVE **[20 MARKS]**

(a) Consider the following grammar.

$$S \quad \rightarrow \quad S\ a \mid b \mid \text{error}\ a$$

(i) Show the DFA for recognizing viable prefixes of this grammar. Use LR(0) items, and treat error as a terminal. [6 Marks]

(ii) Tools such as bison use error productions in the following way. When a parsing error is encountered (i.e., the machine cannot shift, reduce, or accept):

First: pop and discard elements of the stack one at a time until a stack configuration is reached where the error terminal of an error production can be shifted on to the stack.

Second: discard tokens of the input one at a time until one is found that can be shifted on to the stack.

Third: resume normal execution of the parser.

Show the sequence of stack configurations of an SLR(1) parser for the grammar above on the following input. [8 Marks]

(b) Discuss the code generator design issues. [6 Marks]