CE305-6-SP

UNIVERSITY OF ESSEX

Undergraduate Examinations 2023

LANGUAGES AND COMPILERS

Time Allowed: **TWO** hours

This exam is Open Book (Restricted), students may bring/use the following in the exam:
Study Notes (up to 12 A4 pages)

Candidates are permitted to use:
Calculator – Casio FX-83GT PLUS/X or Casio FX-85GT PLUS/X only

The paper consists of **FOUR** questions.

Candidates must answer **ALL** questions.

The questions are of **NOT** of equal weight.

The percentages shown in brackets provide an indication of the proportion of the total marks for the **PAPER** which will be allocated.

Please do not leave your seat unless you are given permission by an invigilator.
Do not communicate in any way with any other candidate in the examination room.
Do not open the question paper until told to do so.
All answers must be written in the answer book(s) provided.
All rough work must be written in the answer book(s) provided. A line should be drawn through any rough work to indicate to the examiner that it is not part of the work to be marked.
At the end of the examination, remain seated until your answer book(s) have been collected and you have been told you may leave.
Your responses must be your own work. Procedures are in place to detect plagiarism.

<u>Question 1</u>

(a) What are the different stages of compilation? What is *parsing* and why is it important while compiling a program? [5%]

(b) Give a regular expression for the positive integer numbers, *digits*, making use of *digit* and *nonzerodigit* as defined below. Note: *digits* is non-zero and can be of any length. [5%]

nonzerodigit = (1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)
digit = (0 | *nonzerodigit*)

(c) Translate the regular expression given in your answer to (b) into a Non-deterministic Finite Automaton (NFA). [8%]

(d) Translate the Non-deterministic Finite Automaton given in your answer to (c) into a Deterministic Finite Automaton (DFA). [7%]

(e) In tokenising input that contains keyword and identifiers such as if, i and f what issue might arise? How can you resolve this issue? [5%]

Question 2

The syntax of a simple programming language includes the following context-free grammar rules:

$\mathrm{E} \rightarrow \mathrm{LIT(n)}$

$\mathrm{E} \rightarrow \mathrm{VAR(v)}$

$\mathrm{E} \rightarrow \mathrm{E\ E}\ +$

$\mathrm{E} \rightarrow \mathrm{E\ E}\ -$

$\mathrm{E} \rightarrow \mathrm{E\ E}\ *$

$\mathrm{B} \rightarrow \mathrm{E\ E}\ =$

$\mathrm{B} \rightarrow \mathrm{E\ E}\ >$

$\mathrm{B} \rightarrow \mathrm{B\ B\ \ AND}$

$\mathrm{B} \rightarrow \mathrm{B\ B\ \ OR}$

$\mathrm{S} \rightarrow \mathrm{E} -> \mathrm{VAR(v)}$

$\mathrm{S} \rightarrow \mathrm{B\ IF\ S\ ENDIF}$

$\mathrm{S} \rightarrow \mathrm{REPEAT\ S\ UNTIL\ B}$

where n is any number and v is any identifier.

(a) Translate the above grammar rules into EBNF notation. [5%]

(b) Parse the following string using the above grammar, showing your answer as a *Concrete Syntax Tree* (CST). [8%]

$\mathrm{REPEAT\ VAR(x)\ LIT(1)} - \mathrm{UNTIL\ VAR(x)\ LIT(1)}\ >$

(c) Draw the *Abstract Syntax Tree* (AST) that can be derived from the parse tree you gave in the answer to part (b) of this question. [8%]

(d) The above grammar can give rise to a potential problem for a simple top-down parser. Identify which rule(s) may cause problems for such a parser, and briefly sketch the nature of this problem. [4%]

Question 3

(a) Explain in detail how a stack-based symbol table could be used for type-checking when compiling the following example: [8%]

```
BEGIN
FLOAT n;
INT t;
STRING s;
...
BEGIN
INT s;
STRING t;
...
c := s;
d := t;
...
END
END
```

(b) Given that the final target language is often an untyped machine language, and that operations on different types of data may be translated into exactly the same instructions, what is the point of types and type-checking in high level languages? [5%]

(c) Discuss the distinction between *static* scoping and *dynamic* scoping. Your answer should include a comparison of the complexity of understanding the two kinds of scoping. [8%]

(d) In programming languages, can the same name be declared multiple times? Explain your answer, with examples. [4%]

Question 4

(a) Explain one key advantage that stack-based machines have over register-based machines, and one key advantage of register-based machines over stack-based machines, when used as intermediate targets for compiler-generated code. [5%]

(b) Consider the following rules for translating a high-level intermediate representation into *three-address* instructions:

```
I[NUM(n)]t    = t ← n
I[VAR(a)]t    = t ← a
I[E1 ADD E2]t = I[E1]t1
                I[E2]t2
                t ←  t1 + t2

I[E1 SUB E2]t = I[E1]t1
                I[E2]t2
                t ←  t1 - t2

T[VAR(a) ASSGN e] = T[e]t
                    a ←  t

T[IF b S] = I[b]t
            BRANCH ZERO t L
            T[S]
            LABEL L
```

Where **T** translates program statements, **I** translates expressions, t, t1, t2 are always fresh registers, and the t in I[EXP]t provides the name of the (new) register in which the result of translating EXP is to be stored.

Using these definitions of **I** and **T**, give translations of the following into three-address instructions.

(i) `VAR(a) ASSGN VAR(a) ADD NUM(4)` [5%]

(ii) `IF NUM(1) VAR(a) ASSGN NUM(2)` [6%]

(c) Consider (b) (ii) of the current question: `IF NUM(1) VAR(a) ASSGN NUM(2)`. How could the control flow of the generated program be optimised? [4%]

END OF PAPER CE305-6-SP