CS_218

COMPILERS

(*Attempt 2 questions out of 3*)

Question 1

(a) What differences might you expect between compilers designed for the following purposes:

(i) a compiler used in an introductory programming course.

(ii) a compiler that targets the embedded processor used in a mobile phone.

[5 marks]

(b) Explain and give one example of each of the following types of *syntax* error:

(i) a lexical error.

(ii) a grammatical error.

[4 marks]

(c) Explain and give one example of each of the following types of *semantic* error:

(i) a type error.

(ii) a logical error.

(iii) a run-time error.

[6 marks]

(d) Briefly describe the effect of each of the major compilation phases (lexical analyser, syntax analyser, semantic analyser, intermediate code generator, code optimiser, code generator) when compiling the assignment statement

```
x := y * z + 3 + 2
```

where `x`, `y` and `z` are real number variables. State any assumptions that you make.

[10 marks]

Question 2

Consider the following context-free grammar for variable declarations in a Pascal-like language:

```
VarDec   ⟶ var DecList
DecList  ⟶ Dec ; DecList
DecList  ⟶ Dec ;
Dec      ⟶ IdList :  IdType
IdList   ⟶ IdList , id
IdList   ⟶ id
IdType   ⟶ integer
IdType   ⟶ boolean
```

where the token `id` represents any arbitrary single letter.

(a) Derive a leftmost derivation for the string:

```
var x,y :  integer; z :  boolean;
```

and draw the corresponding parse tree.

[8 marks]

We wish to construct a top-down predictive parser for the above grammar, but we must first transform it into an LL(1) grammar by eliminating left recursions and carrying out left factoring.

(b) What do the two occurrences of the letter "L" and the number "1" stand for in "LL(1)"?

[3 marks]

(c) Define the terms *left recursion*, *immediate left recursion*, and *left factoring*.

[4 marks]

(d) Transform the above grammar into LL(1) form by eliminating left recursions and applying left factoring wherever needed. Describe each step of your transformation.

[10 marks]

Question 3

(a) Draw state transition diagrams for DFAs which recognise the following languages.

(i) The set of nonempty strings of a's over the alphabet $\{a, b\}$; that is, $\{a, aa, aaa, \ldots\}$.

(ii) The set of strings of alternating a's and b's over the alphabet $\{a, b\}$ which contain the same number of a's as b's, and not beginning with a b; that is, $\{\varepsilon, ab, abab, \ldots\}$.

(iii) The set of binary numerals over the alphabet $\{0, 1\}$; that is, $\{0, 1, 10, 11, 100, \ldots\}$. (Note that a numeral must not start with a 0 unless it is 0.)

[5 marks]

(b) Give regular expressions for each of the three langages from part (a) above.

[5 marks]

(c) Give a context-free grammar for the language of mathematical expressions that can be made from the variables x and y, the infix binary operators $+$ and $*$, and the parentheses $($ and $)$. For example, the expressions $x + x * y$ and $(y + x) * (x * (y))$ are in the language.

For full marks, your grammar must be unambiguous, and must enforce the following rules of precedence and associativity:

- the $*$ operator has higher precedence than the $+$ operator;
- the $+$ operator is left associative;
- the $*$ operator is right associative.

That is, it should not be possible to derive a parse tree in which these rules are violated.

[12 marks]

(d) Explain, briefly and informally, why the language from part (c) above could not be represented by a DFA or a regular expression.

[3 marks]