

1.

- a. Describe the main differences between lexical analysis and syntax analysis.

[10 marks]

- b. The abstract syntax of the language SIMP, studied in the course, is defined by the grammar:

$$\begin{aligned}
 C &::= \text{skip} \mid l := E \mid C; C \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \text{ do } C \\
 E &::= !l \mid n \mid E \text{ op } E \\
 \text{op} &::= + \mid - \mid * \mid / \\
 B &::= \text{True} \mid \text{False} \mid E \text{ bop } E \mid \neg B \mid B \wedge B \\
 \text{bop} &::= > \mid < \mid =
 \end{aligned}$$

Draw the abstract syntax tree of the program:

$$\begin{aligned}
 &y := 0; \\
 &\text{if } !x = !y \text{ then } (x := !x + 1; y := !y - 1) \\
 &\text{else } (x := !x - 1; y := !y + 1)
 \end{aligned}$$

(You are allowed – but not required – to draw the abstract syntax tree with pen and paper and submit the answer to this item in the second PDF file to complement the first with your typed answers as per instructions on the cover of this exam)

[10 marks]

- c. Give the structural induction principle for Boolean expressions in SIMP.

[10 marks]

- d. Let C_1 , C_2 and C_3 be arbitrary SIMP commands.

Using the abstract machine semantics of SIMP, show that the programs $C_1; (C_2; C_3)$ and $(C_1; C_2); C_3$ are equivalent, that is, they produce the same results when executed in the same state. Therefore the brackets are not needed.

[10 marks]

- e. The new Boolean operator *is-positive* applies to an integer expression and indicates whether the integer is positive or not: it produces *True* if the integer is greater or equal to zero, and *False* otherwise.

Give a formal semantics for expressions of the form

$$is\text{-}positive(E).$$

More precisely, extend the big-step semantics of SIMP by providing the necessary axioms or rules to evaluate expressions of the form *is-positive*(E).

(You are allowed – but not required – to write the new axiom(s) and rule(s) with pen and paper and submit the answer to this item in the second PDF file to complement the first with your typed answers as per instructions on the cover of this exam)

[10 marks]

2.

- a. i. Define the concept of a *typed* variable. Differentiate *static* from *dynamic* typing in programming languages.

[10 marks]

- ii. Give two advantages of the use of types in a language and one disadvantage.

[6 marks]

- b. i. Distinguish the *call-by-value* and *call-by-name* evaluation strategies, citing one advantage and one disadvantage for each strategy.

[6 marks]

- ii. Consider the SFUN program below.

$$\begin{aligned} \text{calc}(x) &= \text{calc}(x + 1) \\ \text{sum}(x, y) &= \text{if } y = 0 \text{ then } 0 \text{ else } x + y \end{aligned}$$

Evaluate the expression

$$\text{sum}(\text{calc}(0), 0)$$

using the *call-by-value* strategy and the *call-by-name* strategy. Explain your answer.

[8 marks]

- c. i. Give two different ways through which a *polymorphic* type system may be defined for a programming language.

[4 marks]

- ii. What do you understand by a type *constructor*? Give two examples of type constructors.

[4 marks]

iii. Find a polymorphic type for the functions below. State your reasoning.

1. `zero x = 0`

2. `comp f g x = f (g x)`

3. The function `func` defined as

```
func (Leaf l) g = g l
```

```
func (Branch l r) g = (func l g) ++ (func r g)
```

where

```
data Tree a = Leaf a | Branch (Tree a) (Tree a)
```

and `++` is list concatenation.

[12 marks]

3.

a. Consider the following recursive Haskell function:

```
mysum :: Int -> Int
```

```
mysum x = if x==1 then 0 else 2*(x-1) + mysum (x-1)
```

- i. Describe in your own words what the function `mysum x` computes and illustrate your answer by computing the value of `mysum 5`.

[4 marks]

- ii. Prove by induction that `mysum x` returns the value $x^2 - x$.

[6 marks]

b. i. Explain the concepts of *unifier* and *most general unifier*. Illustrate the difference between the two concepts, by giving two unifiers σ and ρ that would solve the unification problem

$$\mathcal{U} = \{X = a, f(Y) = f(Z)\},$$

such that σ is *not* a most general unifier of $\mathcal{U}$, but ρ is.

[10 marks]

- ii. Indicate which of the following expressions unify with the functional term

```
transform(X, a(b,c), g(W, [H | T]))
```

If the expression unifies with the term, give a most general unifier that unifies them. If it does not unify, explain why.

1. `[XH | XT]`
2. `transform(f(X), a(b,Y), g(c, [1]))`
3. `transform(b, a(b,c), g(W, [a,b,c]))`
4. `transform(r(b,QQ,E), fb(G,H), G)`
5. `transform(X1, X2, g(234, []))`

[10 marks]

c. Consider the following PROLOG program.

```
compute(1,0).  
compute(X,Y) :- X > 1,  
                Z is X-1,  
                compute(Z,Z1),  
                Y is (2*Z)+Z1.
```

i. Give the answers PROLOG would produce to the following queries.

1. `compute(X,0).`
2. `compute(2,0).`
3. `compute(3,Y).`
4. `compute(0,Y).`
5. `compute(8,Y).`

[10 marks]

ii. Draw the complete SLD-resolution tree for the query

`:- compute(3,Y).`

using the program above and the computation rule that always selects the leftmost literal to be resolved upon.

Rename the variables as necessary and indicate the variable bindings of all resolution steps.

(You are allowed – but not required – to draw the SLD-resolution tree with pen and paper and submit the answer to this item in the second PDF file to complement the first with your typed answers as per instructions on the cover of this exam)

[10 marks]