# Final Exam

## EDAP05: Concepts of Programming Languages, HT 2021

### 2022-01-15

Anonymisation code: _____

| Question: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | **Sum** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Max Points: | 8 | 6 | 3 | 9 | 8 | 7 | 9 | 11 | 11 | 7 | 15 | 6 | 100 |
| Points Reached: | | | | | | | | | | | | | |

MYSTERY grammar (for reference):

$$\langle Program\rangle \longrightarrow \langle Block\rangle$$
$$| \quad \langle Block\rangle \; ';'$$
$$\langle Decls\rangle \longrightarrow \langle DeclList\rangle \mid \varepsilon$$
$$\langle DeclList\rangle \longrightarrow \langle Decl\rangle$$
$$| \quad \langle Decl\rangle \; ';' \; \langle DeclList\rangle$$
$$\langle Decl\rangle \longrightarrow \text{'VAR'} \; id \; \langle OptType\rangle$$
$$| \quad \text{'TYPE'} \; id \; '=' \; \langle Type\rangle$$
$$| \quad \langle ProcDecl\rangle$$
$$\langle OptType\rangle \longrightarrow \varepsilon \mid ':' \; \langle Type\rangle$$
$$\langle ProcDecl\rangle \longrightarrow \text{'PROC'} \; id \; '(' \; \langle Formals\rangle \; ')' \; \langle OptType\rangle \; '=' \; \langle Block\rangle$$
$$| \quad \text{'PROC'} \; id \; '(' \; \langle Formals\rangle \; ')' \; '=' \; \langle Block\rangle$$
$$\langle Formals\rangle \longrightarrow \langle FormalList\rangle \mid \varepsilon$$
$$\langle FormalList\rangle \longrightarrow \langle Formal\rangle$$
$$| \quad \langle FormalList\rangle \; ',' \; \langle Formal\rangle$$
$$\langle Formal\rangle \longrightarrow id \; ':' \; \langle Type\rangle$$
$$\langle Type\rangle \longrightarrow \text{'INT'}$$
$$| \quad \langle SubrTy\rangle$$
$$| \quad \langle ArrayTy\rangle$$
$$| \quad id$$
$$| \quad \langle ProcTy\rangle$$
$$\langle SubrTy\rangle \longrightarrow '[' \; number \; \text{'TO'} \; number \; ']'$$
$$\langle ArrayTy\rangle \longrightarrow \text{'ARRAY'} \; \langle SubrTy\rangle \; \text{'OF'} \; \langle Type\rangle$$
$$\langle ProcTy\rangle \longrightarrow \text{'PROC'} \; '(' \; \langle Formals\rangle \; ')' \; \langle OptType\rangle$$
$$\langle Block\rangle \longrightarrow \langle Decls\rangle \; \text{'BEGIN'} \; \langle Stmts\rangle \; \text{'END'}$$
$$\langle Stmts\rangle \longrightarrow \langle StmtList\rangle \mid \varepsilon$$
$$\langle StmtList\rangle \longrightarrow \langle Stmt\rangle$$
$$| \quad \langle StmtList\rangle \; ';' \; \langle Stmt\rangle$$
$$\langle Stmt\rangle \longrightarrow \langle Assignment\rangle$$
$$| \quad \langle Return\rangle$$
$$| \quad \langle Block\rangle$$
$$| \quad \langle Conditional\rangle$$
$$| \quad \langle Iteration\rangle$$
$$| \quad \langle Output\rangle$$
$$| \quad \langle Expr\rangle$$
$$\langle Assignment\rangle \longrightarrow \langle Expr\rangle \; ':=' \; \langle Expr\rangle$$
$$\langle Return\rangle \longrightarrow \text{'RETURN'} \; \langle Expr\rangle$$
$$\langle Conditional\rangle \longrightarrow \text{'IF'} \; \langle Expr\rangle \; \text{'THEN'} \; \langle StmtList\rangle \; \text{'ELSE'} \; \langle StmtList\rangle \; \text{'END'}$$
$$\langle Iteration\rangle \longrightarrow \text{'WHILE'} \; \langle Expr\rangle \; \text{'DO'} \; \langle StmtList\rangle \; \text{'END'}$$
$$\langle Output\rangle \longrightarrow \text{'PRINT'} \; \langle Expr\rangle$$
$$\langle Expr\rangle \longrightarrow \langle Operand\rangle$$
$$| \quad \langle Expr\rangle \; \langle Operator\rangle \; \langle Operand\rangle$$
$$\langle Operand\rangle \longrightarrow number$$
$$| \quad id$$
$$| \quad \langle Operand\rangle \; '[' \; \langle Expr\rangle \; ']'$$
$$| \quad \langle Operand\rangle \; '(' \; \langle Actuals\rangle \; ')'$$
$$| \quad '(' \; \langle Expr\rangle \; ')'$$
$$\langle Operator\rangle \longrightarrow '+' \mid '>' \mid '==' \mid \text{'AND'}$$
$$\langle Actuals\rangle \longrightarrow \langle ActualList\rangle \mid \varepsilon$$
$$\langle ActualList\rangle \longrightarrow \langle Expr\rangle$$
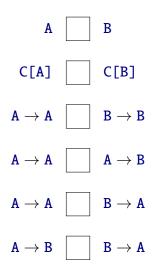$$| \quad \langle Actuals\rangle \; ',' \; \langle Expr\rangle$$

# Question 1 (8 Points)

In the table below you see pairs of types with a box ☐ in between. Write an $X$ in the box if neither type is a subtype of the other, or draw a $<:$ or $:>$ (suitably) to indicate that one is a subtype of the other.

Use the same assumptions as in class, i.e., that (1) we are using an imperative language (updates are allowed), that (2) the type system enforces strong typing and (3) the type system permits any type to be a subtype of another if and only if doing so will not require dynamic checks.

(a) (4 Points) Fill in as indicated above:

`[5 TO 8]` ☐ `[0 TO 8]`

`[5 TO 8]` ☐ `[5 TO 7]`

`[5 TO 8]` ☐ `[3 TO 7]`

`ARRAY [0 TO 10] OF [0 TO 10]` ☐ `ARRAY [5 TO 10] OF [0 TO 10]`

`ARRAY [0 TO 10] OF [5 TO 10]` ☐ `ARRAY [0 TO 10] OF [0 TO 10]`

(b) (4 Points) Continue filling in. For the following, assume that `A` is a *supertype* of `B`, and that the type `C[X]` is *contravariant* in type parameter `X`.

`A` ☐ `B`

`C[A]` ☐ `C[B]`

`A → A` ☐ `B → B`

`A → A` ☐ `A → B`

`A → A` ☐ `B → A`

`A → B` ☐ `B → A`

# Question 2  (6 Points)

Consider the following MYSTERY program.

```
1 PROC g(v : INT) =
2   VAR z : INT;
3   PROC R() =
4     BEGIN
5       z := 0        // Assignment-A
6     END;
7   PROC f(i : INT): INT =
8     VAR z : INT
9     BEGIN
10      R();
11      z := z + i;  // Assignment-B
12      RETURN z
13    END
14  BEGIN
15    RETURN f(v);
16  END
17 BEGIN
18   PRINT g(0);
19 END
```

(a) (4 Points) Assuming static or dynamic scoping, which z (line 2 or line 8) do Assignment-A and Assignment-B update? Fill in.

| Scoping | Assignment | Line |
|---------|------------|------|
| **Static** | Assignment-A | |
| **Static** | Assignment-B | |
| **Dynamic** | Assignment-A | |
| **Dynamic** | Assignment-B | |

(b) (2 Points) Assuming static scoping, what are the scopes of i and of the z in line 8? Give line numbers or mark in the code above.

# Question 3 (3 Points)

Consider the following MYSTERY program. Assume that all `INT` variables are initialised to `0` by default.

```
1 PROC h(i : INT): INT =
2   VAR y : INT
3   BEGIN
4     y := y + i;
5     RETURN y
6   END
7 BEGIN
8   PRINT h(0);
9   PRINT h(1);
10  PRINT h(2)
11 END
```

When run, the program prints the following:

```
0
1
3
```

(a) (3 Points) What storage binding does y (line 2) use? Explain.

# Question 4 (9 Points)

You are using a programming language that has a built-in array type, array<T>, where T can be any type in the language. You now want to write a subroutine concat that concatenates two arrays. Below is a snippet in Java syntax that illustrates the idea:

```java
// in Java: assume a static method for the subroutine
static SomeType_0 concat(SomeType_1 array1, SomeType_2 array2) {
    SomeType_3 mergedArray =
        new SomeType_4[array1.length + array2.length];

    // Copy array1 into mergedArray
    ...
    // Copy array2 into mergedArray
    ...

    return mergedArray;
}
```

Assume that we want the type of concat to be as general as possible, and that the language is strongly statically typed. In the following, you can use Java, Scala, SML, Rust, or C++ syntax, or explain informally. If you cannot express what you want to express in the syntax that you picked, use English.

(a) (2 Points) Assume that the language supports parametric polymorphism, but not subtype polymorphism. Specify the most general type for concat that you can.

(b) (2 Points) Consider the type that you gave in (a). In a language like Java with subtype polymorphism, are there any uses of concat that would be type-safe from the perspective of strong typing but that are forbidden by your type? Explain or give an example.

(c) (4 Points) Assume that the language supports subtype polymorphism and bounded parametric polymorphism. Specify the most general type for `concat` that you can.

(d) (1 Point) Are there any uses of `concat` in this language that would be type-safe from the perspective of strong typing but that are forbidden by your type? Explain or give an example.

# Question 5  (8 Points)

What is the relation between variance and parameter passing modes, if any?

Consider the following MYSTERY program:

```
1 PROC g(f : PROC(x : [0 TO 10]): INT) : INT = //...
```

Here, f : [1 TO 10] → INT, and f takes a parameter x : [1 TO 10].

In the following sub-questions, *explain why the alternative(s) do not preserve strong typing*. You can give an example. Remember that you can reference your answers to other sub-questions to simplify your answer.

(a) (2 Points)  Assume that we pass x via *pass-by-value*. What is the most general variance that we can give to the type of x, relative to the type of f? Explain.

(b) (3 Points)  (**Synthesis**) Assume that we pass x via *pass-by-result*. What is the most general variance that we can give to the type of x, relative to the type of f? Explain.

(c) (3 Points)  (**Synthesis**) Assume that we pass x via *pass-by-name*. What is the most general variance that we can give to the type of x, relative to the type of f? Explain.

# Question 6  (7 Points)

(a) (4 Points)  Which of the following is part of the syntax, static semantics, and dynamic semantics? Add check-marks as appropriate.

| | Syntax | Static Semantics | Dynamic Semantics |
|---|---|---|---|
| Type Inference | | | |
| Type Checking | | | |
| Operator associativity | | | |
| Closures | | | |

(b) (3 Points)  (**Synthesis**) Assume that you are given a language that is implemented in a pure interpreter (i.e., not a hybrid implementation). Is it possible that the language uses static typing? Explain your reasoning.

# Question 7 (9 Points)

We are adding a new type to Java. This type, `valset<T>`, is polymorphic over `T`, and represents sets that contain `T` values as elements.

Values of type `valset<T>` cannot be modified: any operation on them instead creates a new value, analogously to integers.

Assume that in the Java syntax, $\langle expr \rangle$ is the Java non-terminal for arbitrary expressions. We extend $\langle expr \rangle$ as follows:

$$\langle expr \rangle \quad \longrightarrow \quad \text{`[['} \langle exprlist \rangle \text{`]]'}$$

$$\begin{aligned}\langle exprlist \rangle \quad \longrightarrow \quad & \langle expr \rangle \\ | \quad & \langle expr \rangle \text{`,'} \langle exprlist \rangle\end{aligned}$$

We also overload the operator +. Let `s1, s2 : valset<T>` and `e : T`, then:

    a. `s1 + e` evaluates to a set that contains the elements of `s1` and the element `e`.

    b. `s1 + s2` evaluates to the union of the two sets `s1` and `s2`.

For brevity, we omit other useful operators that such a set should have.

We define the semantics of set constructions with the following natural semantics rules:

$$\frac{\text{[[ } \ell \text{ ]]} \Downarrow v_\ell \quad e \Downarrow v_e}{\text{[[ } e, \ell \text{ ]]} \Downarrow v_\ell \cup \{v_e\}} \; (\textit{literal-singleton}) \qquad\qquad \frac{e \Downarrow v_e}{\text{[[ } e \text{ ]]} \Downarrow \{v_e\}} \; (\textit{literal-multi})$$

To define the semantics of the overloaded + operator, we have the choice between two options:

---

**Option A**:

$$\frac{\text{[[ } \ell \text{ ]]} \Downarrow v_\ell \quad e \neq \text{[[ } \ell' \text{ ]] for any } \ell' \quad e \Downarrow v_e}{\text{[[ } \ell \text{ ]] } + \; e \Downarrow v_\ell \cup \{v_e\}} \; (\textit{add-element-A})$$

$$\frac{\ell_1 \Downarrow v_{\ell_1} \quad \ell_2 \Downarrow v_{\ell_2}}{\text{[[ } \ell_1 \text{ ]] } + \text{ [[ } \ell_2 \text{ ]]} \Downarrow v_{\ell_1} \cup v_{\ell_2}} \; (\textit{union-A})$$

---

**Option B**:

$$\frac{s : \text{valset<T>} \quad s \Downarrow v_s \quad e : \text{T} \quad e \Downarrow v_e}{s \; + \; e \Downarrow v_s \cup \{v_e\}} \; (\textit{add-element-B})$$

$$\frac{s_1 : \text{valset<T>} \quad s_2 : \text{valset<T>} \quad s_1 \Downarrow v_{s_1} \quad s_2 \Downarrow v_{s_2}}{s_1 \; + \; s_2 \Downarrow v_{s_1} \cup v_{s_2}} \; (\textit{union-B})$$

---

The following questions explicitly provide any Java-specific knowledge you need.

(Questions on the next page.)

(a) (6 Points) Do the natural semantics of **Option A** and **Option B** result in different behaviour in practice? Give an example to support your claim.

(b) (3 Points) (**Synthesis**) Assume that we add boxing/unboxing coercions to Java that can box `valset`<T> into the object type `ValueSet`<T> and unbox accordingly. Thus, $e$ : `valset`<T> whenever $e$ : `ValueSet`<T>.

Review the two semantic options **Option A** and **Option B**. Which of them, if any, are affected by the boxing/unboxing coercions?

Hint: Java uses subtyping, and all object types have `Object` as a supertype.

# Question 8   (11 Points)

We define the language L1 via the non-terminal ⟨*expr*⟩ in the following grammar:

$$
\begin{aligned}
\langle expr \rangle &\longrightarrow \langle con \rangle \\
&\mid \quad \text{`X'} \\
&\mid \quad \langle con \rangle \; \text{`@'} \; \langle expr \rangle \\
\langle con \rangle &\longrightarrow \langle lit \rangle \\
&\mid \quad \langle lit \rangle \; \text{`+'} \; \langle con \rangle \\
&\mid \quad \text{`?'} \; \langle expr \rangle \langle proc \rangle \\
\langle proc \rangle &\longrightarrow \text{`A'} \langle con \rangle \\
&\mid \quad \text{`STOP'} \\
\langle lit \rangle &\longrightarrow nat
\end{aligned}
$$

where *nat* describes the natural numbers ($\mathbb{N}$).

(a) (3 Points)  For each of the following token sequences, check-mark whether they are productions of the L1 grammar:

|  | **Yes** | **No** |
|---|---|---|
| ? 1 + 2 STOP @ X |  |  |
| 1 @ 2 @ 3 + X |  |  |
| ? 1 A X @ 2 |  |  |

(b) (4 Points)  L1 has two binary operators, '@' and '+'. What is the *associativity* of '+'? Explain by giving a parse tree.

(c) (4 Points)  L1 has two binary operators, '@' and '+'. What is the *precedence* of '+'? Explain by giving a parse tree.

# Question 9 (11 Points)

Consider the following custom-defined SML datatype, which describes a list that can contain both `int` and `string` elements:

```
datatype polylist = END
                  | INT of int * polylist
                  | STR of string * polylist
```

With this type, we can e.g. list the values `1`, `"a"`, `2`, `"b"` in order. We would write this list `pl0` as follows:

```
val pl0 = INT(1, STR("a", INT(2, STR("b", END))))
```

(a) (5 Points) Write an SML function `rmstr : polylist → polylist` that removes all STR elements from a `polylist`, unless that element is the very last element in the list. For example, `rmstr(pl0)` should yield `INT(1, INT(2, STR("b", END)))`.

```
1 (* repeated from the previous page *)
2 datatype polylist = END
3                    | INT of int * polylist
4                    | STR of string * polylist
```

(b) (1 Point) Given our definition of `polylist`, what is the type of `INT`?

(c) (5 Points) (**Synthesis**) Consider the following SML function, `org`:

```
1 val org : polylist -> polylist =
2   let fun proc (intl, strl) =
3       let fun mkint (i) (ir) = intl(INT (i, ir))
4           fun mkstr (s) (sr) = strl(STR (s, sr))
5           fun sub (END)        = intl(strl(END))
6             | sub (INT (i, r)) = proc (mkint i, strl) (r)
7             | sub (STR (s, r)) = proc (intl, mkstr s) (r)
8       in sub
9       end
10     fun id (x:polylist) = x
11   in proc (id, id)
12   end
```

SML uses type inference to find the most general type for each function and variable. Using your knowledge of SML, determine the types of the following functions and variables:

    id

  intl

  strl

 mkint

 mkstr

   sub

  proc

(d) (**Optional**: 3 points (bonus)) What does the function `org` from (b) compute?

# Question 10   (7 Points)

(a) Specify a *generic abstract datatype* for arrays in a language of your choice, with operations at least for:

    a. reading array element

    b. updating array elements

    c. determining the array length

The abstract datatype does not have to match the array operations already provided by the language that you have selected. Specify which language you used.

You may use language features that your language of choice does not provide, if you explain them, and you may deviate from the language syntax as long as your meaning is clear.

(b) In what sense is your specification an abstract datatype?

(c) In what sense is your specification a *generic* abstract datatype?

# Question 11   (15 Points)

Explain the differences between various concepts that we have discussed throughout the course.

(a)  (3 Points)  What are the differences between pointers and references?

(b)  (4 Points)  What is the difference between reference equality vs structural equality? Explain with an example.

(c) (4 Points) What is the difference between an enumeration type and an algebraic datatype? Explain with an example.

(d) (4 Points) What is the difference between overloading and overriding? Explain with one example for each.

# Question 12 (6 Points)
**(Synthesis)**

Consider the following MYSTERY program:

```
1   PROC P(a : INT, b : INT) : INT =
2   BEGIN
3     IF a
4     THEN RETURN b
5     ELSE RETURN a
6     END
7   END;
8   PROC Q(x : INT): INT =
9   BEGIN
10    PRINT x;
11    RETURN x
12  END
13 BEGIN
14   P(Q(1) AND Q(0), Q(0) AND Q(1))
15 END
```

Upon running the program, you observe the following output:

```
0
0
```

Assume static scoping, stack-dynamic storage binding, and that we are not using pass-by-result.

(a) Given this output, what can we say about the semantics of the **AND** operator? Explain your answer.

(b) Given this output, what can we say about subroutine parameter evaluation order? Explain your answer.