You have **50 minutes** to complete this exam.

This is a **closed-book** exam.

Do not share anything with other students. Do not talk to other students. Do not look other students’ exams. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.

If you believe there is an error or an ambiguous question, you may seek clarification from the instructor. Please speak quietly or write your question out.

Including this cover sheet and rules at the end, there are 8 pages to the exam, including one blank page for workspace. Please verify that you have all 8 pages.

Please write your name and NetID in the spaces above, and also in the provided space at the top of every sheet.

Show your work. Partial credit will be given for incomplete answers.

If you finish with time remaining, check your work!
<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>
Problem 1. (20 points)

Consider the following BNF grammar:

\[
\begin{align*}
O & ::= \langle \text{ident} \rangle | \langle \# \rangle \text{ pieces of } \langle \text{ident} \rangle \\
I & ::= \text{take } O \text{ from } O | \text{put } O \text{ in } O | \text{do } I \langle \# \rangle \text{ times}
\end{align*}
\]

(a) (10 points) Write OCaml datatypes `ast_O` and `ast_I` that encode the abstract syntax trees of `O` and `I` respectively. You may represent `⟨ ident ⟩` with the `string` type and `⟨ # ⟩` with the `int` type.

Solution:

```ocaml
type ast_O = Ident of string | Pieces of int * string

type ast_I = Take of ast_O * ast_O | Put of ast_O * ast_O | Do of ast_I * int
```

(b) (10 points) Write the instance of the `ast_I` type corresponding to the AST for the term `do (take 3 pieces of paper from box) 5 times`

If you prefer, you can instead draw the AST for the term.

Solution:

```ocaml
Do (Take (Pieces (3, Ident "paper"), Ident "box"), 5)
```

or

```
\[
\text{do-times} \quad 5 \\
\text{take-from} \\
\text{pieces-of} \quad \text{box} \\
\text{3} \\
\text{paper}
\]
```
Problem 2. (25 points)
The typing rules for a simple imperative language are given in Appendix [A] Write a proof tree for the judgment

\[ \Gamma \vdash x := \text{if false then 5 else } x : \text{ok} \]

given that \( \Gamma(x) = \text{int} \).

### Solution:

<table>
<thead>
<tr>
<th>( (\Gamma(x) = \text{int}) )</th>
<th>( (\text{false is a boolean}) )</th>
<th>( (5 \text{ is a number}) )</th>
<th>( (\Gamma(x) = \text{int}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash \text{false} : \text{bool} )</td>
<td>( \Gamma \vdash 5 : \text{int} )</td>
<td>( \Gamma \vdash x : \text{int} )</td>
<td></td>
</tr>
<tr>
<td>( \Gamma \vdash \text{if false then 5 else } x : \text{int} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \Gamma \vdash x := \text{if false then 5 else } x : \text{ok} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 3. (15 points)
Suppose you were writing a type-checking function `typecheck_exp : context -> exp -> typ -> bool` that takes a type context `gamma`, an expression `e`, and a type `t` that is either `Tint` or `Tbool`, and returns `true` if `gamma ⊢ e : t` and `false` otherwise. Fill in the skeleton below by translating the typing rule for the if-then-else expression into OCaml code.

```
let rec typecheck_exp (gamma : context) (e : exp) (t : typ) : bool =
  match e with
  | If (e, e1, e2) ->
```

**Solution:**

```
let rec typecheck_exp (gamma : context) (e : exp) (t : typ) : bool =
  match e with
  | If (e, e1, e2) ->
    typecheck_exp gamma e Tbool && typecheck_exp gamma e1 t && typecheck_exp gamma e2 t
```

Problem 4. (15 points)
The operational semantics rules for a simple imperative language are shown in Section [B]. Write a proof tree for the next step that takes.

```
(if true then y := 5 else skip, {y = 3})
```

**Solution:**

```
(\(true, \{y = 3\}\) ⊨ true)
```

```
(if true then y := 5 else skip, \{y = 3\}) \rightarrow (y := 5, \{y = 3\})
```
Problem 5. (25 points)
Write small-step operational semantics rules for the construct

\[
\text{switch } e \{ \text{ case 0: } c_0; \text{ case 1: } c_1; \text{ default: } c_2 \}
\]

executes \(c_0\) if the value of \(e\) is 0, \(c_1\) if the value of \(e\) is 1, and \(c_2\) if the value of \(e\) is anything else.

Hint: Think about which existing construct is most similar to switch, and look at its rules.

Solution:

\[
\begin{array}{c}
(e, \sigma) \downarrow 0 \\
\quad \text{(switch } e \{ \text{ case 0: } c_0; \text{ case 1: } c_1; \text{ default: } c_2 \}, \sigma) \rightarrow (c_0, \sigma) \\
\end{array}
\]

\[
\begin{array}{c}
(e, \sigma) \downarrow 1 \\
\quad \text{(switch } e \{ \text{ case 0: } c_0; \text{ case 1: } c_1; \text{ default: } c_2 \}, \sigma) \rightarrow (c_1, \sigma) \\
\end{array}
\]

\[
\begin{array}{c}
(e, \sigma) \downarrow v \quad v \neq 0 \text{ and } v \neq 1 \\
\quad \text{(switch } e \{ \text{ case 0: } c_0; \text{ case 1: } c_1; \text{ default: } c_2 \}, \sigma) \rightarrow (c_2, \sigma) \\
\end{array}
\]
Typing Rules for a Simple Imperative Language

\[
\frac{(n \text{ is a number})}{\Gamma \vdash n : \text{int}} \quad \frac{(b \text{ is a boolean})}{\Gamma \vdash b : \text{bool}} \quad \frac{(\Gamma(x) = \tau)}{\Gamma \vdash x : \tau}
\]

\[
\frac{\Gamma \vdash e_1 : \text{int}}{\Gamma \vdash e_1 \oplus e_2 : \text{int}} \quad \text{where } \oplus \text{ is an arithmetic operator}
\]

\[
\frac{\Gamma \vdash e_1 : \text{bool}}{\Gamma \vdash e_1 \otimes e_2 : \text{bool}} \quad \frac{\Gamma \vdash e_2 : \text{bool}}{\Gamma \vdash e_1 = e_2 : \tau}
\]

\[
\frac{(\Gamma(x) = \tau)}{\Gamma \vdash x := e : \text{ok}} \quad \frac{\Gamma \vdash c : \text{ok}}{\Gamma \vdash \text{while } e \text{ do } c : \text{ok}}
\]

Operational Semantics for a Simple Imperative Language

\[
\frac{(n, \sigma) \Downarrow n}{(b, \sigma) \Downarrow b} \quad \frac{(\sigma(x) = v)}{(x, \sigma) \Downarrow v}
\]

\[
\frac{(e_1, \sigma) \Downarrow v_1 \quad (e_2, \sigma) \Downarrow v_2 \quad (v_1 \oplus v_2 = v)}{(e_1 \oplus e_2, \sigma) \Downarrow v} \quad \text{where } \oplus \text{ is an arithmetic or boolean operator}
\]

\[
\frac{(e, \sigma) \Downarrow \text{true} \quad (e_1, \sigma) \Downarrow v}{(\text{if } e \text{ then } e_1 \text{ else } e_2, \sigma) \Downarrow v}
\]

\[
\frac{(e, \sigma) \Downarrow \text{false}}{(\text{if } e \text{ then } e_1 \text{ else } e_2, \sigma) \Downarrow v}
\]

\[
\frac{(x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto v])}{(c_1 ; c_2, \sigma) \rightarrow (c_2, \sigma)}
\]

\[
\frac{(c_1, \sigma) \rightarrow (c'_1, \sigma')}{(c_1 ; c_2, \sigma) \rightarrow (c'_1 ; c_2, \sigma')}
\]

\[
\frac{(e, \sigma) \Downarrow \text{true}}{(\text{if } e \text{ then } c_1 \text{ else } c_2, \sigma) \rightarrow (c_1, \sigma)}
\]

\[
\frac{(e, \sigma) \Downarrow \text{false}}{(\text{if } e \text{ then } c_1 \text{ else } c_2, \sigma) \rightarrow (c_2, \sigma)}
\]

\[
\frac{(\text{while } e \text{ do } c, \sigma) \rightarrow (\text{if } e \text{ then } (c ; \text{while } e \text{ do } c) \text{ else skip}, \sigma)}{\text{skip} ; c_2, \sigma) \rightarrow (c_2, \sigma)}
\]
C Scratch Space