

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page. The exam has 4 questions with equal weight. Some questions have multiple parts. Point totals are given in the margin.

Name: _____ **Answer Key** _____

1. _____ **25**

2. _____ **25**

3. _____ **25**

4. _____ **25**

Total _____ **100**

No notes or books may be used on the exam. If you have any questions, please raise your hand and I will try to answer them.

25 1. Consider the following grammar:

$$E \rightarrow E_1 + T$$

$$| T$$

$$T \rightarrow T_1 * F$$

$$| F$$

$$F \rightarrow \mathbf{id}$$

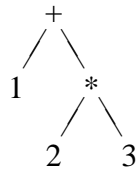
$$| \mathbf{real}$$

$$| (E) \quad \{ F.\text{node} = E.\text{node} \}$$

Finish writing the actions to build the *abstract* syntax tree for this grammar. Assume you have *only* the following functions available:

- Node *mkleaf (char *label);
- Node *mknode (char *label, Node *left, Node *right);

You do *not* need to provide attribute type definitions for your grammar. You may assume that **id** and **real** have an attribute type of string (char *) and that all nonterminals have an attribute named *node*. As an example, the abstract syntax tree for $1 + 2 * 3$ is:



Answer:

$$E \rightarrow E_1 + T \quad \{ E.\text{node} = \text{mknode} ("+", E_1.\text{node}, T.\text{node}) \}$$

$$| T \quad \{ E.\text{node} = T.\text{node} \}$$

$$T \rightarrow T_1 * F \quad \{ T.\text{node} = \text{mknode} ("*", T_1.\text{node}, F.\text{node}) \}$$

$$| F \quad \{ T.\text{node} = F.\text{node} \}$$

$$F \rightarrow \mathbf{id} \quad \{ F.\text{node} = \text{mkleaf} (\mathbf{id}) \}$$

$$| \mathbf{real} \quad \{ F.\text{node} = \text{mkleaf} (\mathbf{real}) \}$$

$$| (E) \quad \{ F.\text{node} = E.\text{node} \}$$

- 25 2. Consider the following standard arithmetic operations on points (represented as a pair of real numbers) and scalars (represented as a real number):

$$\begin{aligned} \text{dot product: } (a, b) \times (c, d) &= a \times c + b \times d \\ \text{addition: } (a, b) + (c, d) &= (a + c, b + d) \\ \text{scaling: } (a, b) \times x &= (a \times x, b \times x) \\ \text{translation: } (a, b) + x &= (a + x, b + x) \end{aligned}$$

For example, the dot product of two points is a scalar. Finish writing the actions for the following grammar to *type check* expressions involving points and scalars:

$$\begin{array}{l} E \rightarrow E_1 + T \\ \quad | T \end{array}$$

$$\begin{array}{l} T \rightarrow T_1 * F \\ \quad | F \end{array}$$

$$\begin{array}{l} F \rightarrow \mathbf{real} \quad \{ F.type = \mathbf{REAL} \} \\ \quad | (\mathbf{real}, \mathbf{real}) \quad \{ F.type = \mathbf{POINT} \} \end{array}$$

Your actions should allow standard addition and multiplication on reals as well as the operations defined above. You may assume that all nonterminals have an attribute named *type*, which can have either the value REAL or POINT. Note that you are writing actions to type check the expressions, *not* to evaluate them.

Answer:

$$\begin{array}{l} E \rightarrow E_1 + T \quad \{ \text{if } E_1.type = \mathbf{POINT} \text{ or } T.type = \mathbf{POINT} \text{ then } E.type = \mathbf{POINT} \text{ else } E.type = \mathbf{REAL} \} \\ \quad | T \quad \{ E.type = T.type \} \end{array}$$

$$\begin{array}{l} T \rightarrow T_1 * F \quad \{ \text{if } T_1.type = F.type \text{ then } T.type = \mathbf{REAL} \text{ else } T.type = \mathbf{POINT} \} \\ \quad | F \quad \{ T.type = F.type \} \end{array}$$

$$\begin{array}{l} F \rightarrow \mathbf{real} \quad \{ F.type = \mathbf{REAL} \} \\ \quad | (\mathbf{real}, \mathbf{real}) \quad \{ F.type = \mathbf{POINT} \} \end{array}$$

5 3. (a) Give type signatures for the following declarations:

- i. `type t1 = array [1..10] of integer;`
- ii. `type t2 = array [1..2] of array [3..4] of real;`
- iii. `function f (x : array [1..10] of integer; y : real) : integer;`
- iv. `function g (x : real; y : integer) : real;`
- v. `function h (function i (x : integer) : real): real;`

Answer:

- i. `array(1..10,integer)`
- ii. `array(1..2,array(3..4,real))`
- iii. `array(1..10,integer) × real → integer`
- iv. `real × integer → real`
- v. `(integer → real) → real`

5 (b) Which of the following variables have equivalent types using *strict* name equivalence?

```
type t1 = array [1..10] of integer;
type t2 = integer;
var a, b : t1;
var x, y: array [1..10] of integer;
var i : integer;
var j : t2;
```

Answer:

- a and b
- x and y

5 (c) Which of the previous variables have equivalent types using structural equivalence?

Answer:

- a, b, x, and y
- i and j

- 5 (d) Write type signatures for each of the following (possibly overloaded) Pascal operators (assume that the usual coercions are performed and consider only the types CHAR, REAL, INTEGER, and BOOLEAN in your answer):

- i. *
- ii. /
- iii. and
- iv. not
- v. <

Answer:

- i. $\text{integer} \times \text{integer} \rightarrow \text{integer}$
 $\text{real} \times \text{real} \rightarrow \text{real}$
- ii. $\text{real} \times \text{real} \rightarrow \text{real}$
- iii. $\text{boolean} \times \text{boolean} \rightarrow \text{boolean}$
- iv. $\text{boolean} \rightarrow \text{boolean}$
- v. $\text{integer} \times \text{integer} \rightarrow \text{boolean}$
 $\text{real} \times \text{real} \rightarrow \text{boolean}$
 $\text{char} \times \text{char} \rightarrow \text{boolean}$
 $\text{boolean} \times \text{boolean} \rightarrow \text{boolean}$

- 5 (e) Write type signatures for the following *polymorphic* list operations:

- i. get: removes and returns the object on front of specified list
- ii. concat: return the concatenation of two specified lists
- iii. first: returns object on front of specified list
- iv. empty: indicates whether specified list is empty
- v. member: indicates whether a specified member is in a specified list

You may assume that you have a type called `list`. For example, an operation that returns the size of a list would have a signature of $\text{list} \rightarrow \text{integer}$.

Answer:

- i. $\text{list} \rightarrow \alpha$
- ii. $\text{list} \times \text{list} \rightarrow \text{list}$
- iii. $\text{list} \rightarrow \alpha$
- iv. $\text{list} \rightarrow \text{boolean}$
- v. $\text{list} \times \alpha \rightarrow \text{boolean}$

4. Briefly (in a single sentence) answer the following questions:

5

(a) How does your compiler refer to a variable allocated using static allocation?

Answer: by its name

5

(b) How does your compiler refer to a variable allocated using stack allocation?

Answer: by its negative offset from the frame pointer

5

(c) Under what circumstances would a called function *not* need to save the return register?

Answer: if it calls no function

5

(d) Under what circumstances would a called function *not* need to allocate a stack frame?

Answer: if it uses no local variables nor any callee-saved registers (e.g., \$fp)

5

(e) Why does static allocation not work for recursive functions?

Answer: each recursive call overwrites the values of the previous recursive call