Your full name:

Your UCSD ID number:

This exam is closed book and closed notes

Total number of points in this exam: 231 + 25 extra credit

This exam counts for 25% of your overall grade in the class

<table>
<thead>
<tr>
<th>Problem</th>
<th>Number of Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>15</td>
</tr>
<tr>
<td>Problem 2</td>
<td>30</td>
</tr>
<tr>
<td>Problem 3</td>
<td>45</td>
</tr>
<tr>
<td>Problem 4</td>
<td>16</td>
</tr>
<tr>
<td>Problem 5</td>
<td>35 + 25 extra</td>
</tr>
<tr>
<td>Problem 6</td>
<td>50</td>
</tr>
<tr>
<td>Problem 7</td>
<td>40</td>
</tr>
<tr>
<td>Total</td>
<td>231 + 25 extra</td>
</tr>
</tbody>
</table>
Problem 1 (History)  (15 points)

Circle the correct answer for each of the questions below:

a. Which language was the first to try to bring together programming for scientific application and for business applications?
   1. PL/I X
   2. ALGOL 60
   3. Ada

b. Why was separate compilation an important feature of early FORTRAN?
   1. it made it possible to detect syntax errors more precisely
   2. it made it possible to tolerate frequent machine downtimes X
   3. it allowed for better code reuse

c. What can be said of Smalltalk?
   1. it is faster than C++
   2. it is less object-oriented than Java
   3. it is more object-oriented than C++ X

d. What language introduced the concept of a “block structure”?
   1. Ada
   2. ALGOL 60 X
   3. SIMULA 67

e. What design criterion was used extensively in ALGOL 68?
   1. Readibility
   2. Safety
   3. Orthogonality X
Problem 2 (List Construction in Lisp)

a. After evaluating the Lisp expressions (setq l1 '(1 2 3)) and (setq l2 '(a b c)), what would the value of each of the following Lisp expressions be?

1. (5 points)  (cons l1 l2)
   (1 2 3) a b c

2. (5 points)  (cons (car l1) l2)
   (1 a b c)

3. (5 points)  (cons (car (cdr l1)) (cons (car (cdr l2)) NIL))
   (2 b)

b. After evaluating the Lisp expressions (setq l1 '(1 2 3)) and (setq l2 '(a b c)), give Lisp expressions to construct the following lists. Your expressions can only use car, cdr, cons, nil, l1 and l2.

1. (5 points)  (2 a)
   (cons (car (cdr l1)) (cons (car l2) nil))

2. (5 points)  ((a 1))
   (cons (cons (car l2) (cons (car l1) nil)) nil)

3. (5 points)  ((2) (a))
   (cons (cons (car (cdr l1)) nil) (cons (cons (car l2) nil) nil))
Problem 3 (Semantics)

a. (5 points) What is the use of static semantic? What does one use to define static semantic?
   Used to describe the legal form of programs in ways that a BNF grammar cannot. Attributed grammar.

b. (15 points) What are the three kinds of ”dynamic semantics” that we have seen in class. Name one good use of each.
   – Axiomatic semantics. Good for proving program correctness.
   – Denotational semantics. Good basis for an interpreter.
   – Operational semantics. Good for describing the semantics to a programmer.

c. (5 points) Compute the weakest precondition for the following statement and postcondition (assume that all variables are integers):

\[ z = 2 \times (c - 3) - 1 \quad \{ z > 1 \} \]

\{ c > 4 \} by the assignment axiom
c. (20 points) Compute the weakest precondition for the following statement and postcondition (assume that all variables are integers):

\[
\text{if (} a \neq 3 \text{) } a = 2a-1; \text{ else } a -= 2; \{ a > 0 \}
\]

By the assignment axiom:

\[
\{ a > 0 \} a = 2a - 1 \text{ (S1) } \{ a > 0 \}
\]

and:

\[
\{ a > 2 \} a -=2 \text{ (S2) } \{ a > 0 \}
\]

We need to find the weakest P such that:

\[
\frac{\{ P \text{ and } (a \neq 3) \}\text{S1}{a > 0}, \{ P \text{ and } (a = 3) \}\text{S2}{a > 0}}{\{ P \} \text{ if } (a \neq 3) \text{ then } \text{S1} \text{ else } \text{S2}{a > 0}}
\]

\[
P = \{ a > 0 \}
\]

is the weakest precondition because:

- \{ P \text{ and } (a \neq 3) \} \implies \{ P \}, \text{ which is the weakest precondition for } \text{S1}{a > 0}.
- \{ P \text{ and } (a = 3) \} = \{ a = 3 \}, \text{ which is a valid precondition for } \text{S2}{a > 0}.
Problem 4 (Storage Binding)  (16 points)

Consider the following fragment of C code:

```c
int a[4]={1,2,4,8};

int fun (int z) {
    static int foo=0;
    int *x;
    char *y = "hello!";
    int i;
    x = (int *) malloc(10*sizeof(int));
    for (i=0;i<10;i++)
        x[i] = foo++;
    return (z+2);
}

int main() {
    ...
}
```

For each memory reference below, indicate the area of memory it references (with an 'X' in the corresponding column):

<table>
<thead>
<tr>
<th>Reference</th>
<th>static area</th>
<th>heap</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[3]</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>foo</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>*y</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>a</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*x</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Problem 5 (Types)

a. (10 points)  Give two examples of languages that uses static type binding, and two examples of languages that uses dynamic type binding.

Static type binding: C, Java

Dynamic type binding: Lisp, Perl, Javascript

b. (15 points)  Give an example of a language construct in a statically typed language that requires dynamic type checking. Give an example of code that causes a type error undetectable by a compiler.

The C union

union {
    int i;
    char *s;
} x;

x.i = 1;
...
printf(x.s);

c. (10 points)  What are the trade-offs between name type compatibility and structure type compatibility? which type compatibility scheme do most language use?

Name compatibility is easy to implement but very restrictive, while structure compatibility is difficult to implement but more flexible. Most languages use a combination of both (e.g., declaration compatibility).
d. [EXTRA CREDIT] Multidimensional arrays can be stored in row major order, as in C, or in column major order, as in Fortran.

(i) (5 points) Why it is important for a programmer to know which way arrays are stored by the language (Circle the correct answer):

1. To limit the overhead of loop index computation
2. To increase data locality X
3. To save storage space

(ii) (20 points) In a way similar to what we’ve done in class, write the access function for 3-dimensional column-major arrays. Assume an array a, with dimension sizes m, n, and p, with address of the first element a[1,1,1], and with element size element_size.

\[
\text{location}(a[i,j,k]) = \text{address of } a[1,1,1] + ((k-1)*m*n + (j-1)*m + i-1) * \text{element_size}
\]
Problem 6 (Lisp Functions)

a. (15 points) Write a recursive LISP function called `getlast` that takes a list as input and returns the last element of the list. Do NOT use `setq`, `let`, `defvar`, or any other way to declare variables. Here is an example:

```
CL-USER(1): (getlast '(a b c d))
d
```

```
(defun getlast (l)
  (if (null (cdr l))
      (car l)
      (getlast (cdr l))))
```
b. (25 points) Write a recursive LISP function called `inserta` that takes a list as input and returns a new list in which symbol `a` has been inserted after every element of the original list. If the input list is empty, then `inserta` returns the empty list. Do NOT use `setq`, `let`, `defvar`, or any other way to declare variables. Here is an example:

```
(defun inserta (l)
  (cond ((null l) nil)
        (t (cons (car l) (cons 'a (inserta (cdr l)))))))
```

```
CL-USER(1): (inserta '(a b c d))
(a a b a c a d a)
```

c. (10 points) What does the following Common Lisp form evaluate to?

```
(mapcar #'(lambda (x) (list x (list x))) '(a b c))
```

```
(((a (a)) (b (b)) (c (c)))
```
Problem 7 (Expressions)
Consider the following program in C syntax:

```c
int fun (int *i) {
    *i += 5;
    return 2;
}

void main () {
    int x = 3;
    if ((x == 2) && (fun (&x)))
        x++;
    x = x + fun (&x);
    printf("%d\n",x);
}
```

What is the output of this program assuming:

1. (10 points) Left-to-right operand evaluation (for ALL operators), short-circuited boolean expressions
   5

2. (10 points) Left-to-right operand evaluation (for ALL operators), non-short-circuited boolean expressions
   10

3. (10 points) Right-to-left operand evaluation (for ALL operators), short-circuited boolean expressions
   15

4. (10 points) Right-to-left operand evaluation (for ALL operators), non-short-circuited boolean expressions
   15