1. Answer the following concisely.

a. **Describe a C construct whose storage can be statically allocated.**
   The storage for a global variable of type integer can be statically allocated.

b. **Describe a Java construct whose storage must be dynamically allocated.**
   The storage for the local variables of a recursive method must be dynamically allocated.

c. **How are parameters passed in C?**
   Parameters in C are passed by value.

d. **How are parameters passed in Java?**
   Parameters in Java are passed by value. (However, for Java objects the value is essentially a reference to the object.)

e. **Describe a C construct that makes the language not strongly typed.**
   In C a union can contain elements of different types. If the programmer does not keep track of which type is currently stored in a union, a value can be stored in a union as one type and extracted as another type.

f. **Describe a C construct that makes the language not type-safe.**
   Because of pointer arithmetic, a pointer can be made to point to an object of a different type than intended. Also, unions as described in (e) can cause operators to be applied to unintended types.

g. **Give an SDD that shows the difference between a synthesized and inherited attribute.**
   In the following SDD for a while-statement

   \[
   S \rightarrow \text{while (C) S1} \quad L1 = \text{new}; L2 = \text{new}; \\
   S1.next = L1; \\
   C.false = S.next; C.true = L2; \\
   S.code = \text{label} || L1 || C.code || \text{label} || L2 || S1.code
   \]

   the attributes S.next, C.true, and C.false are inherited; the attributes S.code and C.code are synthesized.

h. **Show program fragments to illustrate the difference between name equivalence and structural equivalence.**
   Consider the following two structure declarations:
struct TreeNode {                  struct Node {
    int value;                         int value;
    struct TreeNode *left;             struct Node *left;
    struct TreeNode *right;            struct Node *right;
}                                  }

Under name equivalence the two structures are not equivalent, but under structural
equivalence they would be.

i. Why are activation records used to implement C?
   C is a recursive language and activation records facilitate the implementation of
   recursive functions.

j. What is short-circuit evaluation?
   Short-circuit evaluation of an expression is skipping evaluating portions of
   the expression that cannot affect its value. For example, in the boolean
   expression

   \[(a >= 0) \&\& (b < 0)\]

   if \(a < 0\), then evaluation of the second relational can be skipped.

2. Add rules to the SDD in Fig. 6.36 (ALSU, p. 402) for a do-while
   statement:
   \[S \rightarrow \text{do } S \text{ while } B\]
   Show the code your SDD generates for the program

   do
       do
           assign1
           while a < b
       while c < d

   The most straightforward rules that can be added to Fig. 6.36:

   \[S \rightarrow \text{do } S1 \text{ while } B\] \quad B.true = newlabel();
   \quad B.false = newlabel();
   \quad S1.next = S.next;
   \quad S.code = label(B.true)
   \quad || S1.code
   \quad || B.code
   \quad || label(B.false)

   Assuming the code generated for a boolean expression B of the form \(x < y\) is

   \[\text{if } x < y \text{ goto } B.true \]
   \quad \text{goto } B.false\]
the code generated for the above program is

L1:
L3: assign1.code
   if a<b goto L3
   goto L4
L4: if c<d goto L1
   goto L2
L2:

This SDD generates superfluous goto’s which can be eliminated by a more refined SDD.

3. Consider the following program written in a hypothetical statically scoped language that allows nested functions. In this program, main calls f which calls g2 which calls h which calls g1.

```plaintext
function main() {
    int i;
    function f() {
        int a, b, c;
        function g1() {
            int a, d;
            a = b + c;               // point 1
        }; // end of g1
        function g2(int x) {
            int b, e;
            function h() {
                int c, e;
                g1();
                e = b + a;             // point 2
            }; // end of h
            h();
            a = d + e;               // point 3
        }; // end of g2
        g2();
    }; //end of f
    // execution of main begins here
    f();
}; // end of main
```

a. Suppose we have activation records with the following fields:
local variables, parameters, control link, access link, return address.
If function p is nested immediately within function q, then the access link in any AR for p points to the most recent AR for q. Show the activation records on the run-time stack when execution first arrives at point 1 in the program above.
b. To which declarations are the references to variables a, b, c at position 1?

At position 1, a references the local declaration in g1, and b and c reference the declarations in f.

c. To which declarations are the references to variables a, b, e at position 2?

At position 2, a references the declaration in f, b references the declaration in g2, and e references the local declaration in h.

d. To which declarations are the references to variables a, d, e at position 3?

At position 3, a references the declaration in f, and e references the declaration in g2. The reference to d is an error because there is no visible scope containing a declaration for d at this point.

4. Exercise 8.4.1: Matrix-multiplication program

a) Three address statements for Fig. 8.10:

1) i = 0
2) j = 0
3) t1 = n * i
4) t2 = t1 + j
5) t3 = 8 * t2
6) c[t3] = 0.0
7) j = j + 1
8) if j < n goto (4)
9) i = i + 1
10) if i < n goto (2)
11) i = 0
12) j = 0
13) k = 0
14) t4 = n * i
15) t5 = t4 + j
16) t6 = 8 * t5
17) t7 = c[t6]
18) t8 = t4 + k
19) t9 = 8 * t8
20) t10 = a[t9]
21) t11 = n * k
22) t12 = t11 + j
23) t13 = 8 * t12
24) t14 = b[t13]
25) t15 = t10 * t14
26) t16 = t7 + t15
27) c[t6] = t14
28) k = k + 1
29) if k < n goto (14)
30) j = j + 1
31) if j < n goto (13)
32) i = i + 1
33) if i < n goto (12)

b) Construct the flow graph.
ENTRY

B1

i = 0

B2

j = 0

B3

t1 = n * i

t2 = t1 + j

t3 = 8 * t2

c[t3] = 0.0

j = j + 1

if j < n goto B3

B4

i = i + 1

if i < n goto B2

B5

i = 0

B6

j = 0

B7

k = 0

t4 = n * i

B8

t5 = t4 + j

t6 = 8 * t5

t7 = c[t6]

t8 = t4 + k

t9 = 8 * t8

t10 = a[t9]

t11 = n * k

t12 = t11 + j

t13 = 8 * t12

t14 = b[t13]

t15 = t10 * t14

t16 = t7 + t15

c[t6] = t14

k = k + 1

if k < n goto B8

B9

j = j + 1

if j < n goto B7

B10

i = i + 1

if i < n goto B7

EXIT
c) **Identify the loops.**

There are five loops:

{ B3 }

{ B2, B3, B4 }

{ B8 }

{ B7, B8, B9 }

{ B6, B7, B8, B9, B10 }

5. For the flow graph of Fig. 9.10, compute the *def*, *use*, *IN*, and *OUT* sets for live variable analysis.

<table>
<thead>
<tr>
<th>def</th>
<th>use</th>
</tr>
</thead>
<tbody>
<tr>
<td>B₁</td>
<td>{a,b}</td>
</tr>
<tr>
<td>B₂</td>
<td>{c,d}</td>
</tr>
<tr>
<td>B₃</td>
<td>{d}</td>
</tr>
<tr>
<td>B₄</td>
<td>{d,e}</td>
</tr>
<tr>
<td>B₅</td>
<td>{b,e}</td>
</tr>
<tr>
<td>B₆</td>
<td>{a}</td>
</tr>
</tbody>
</table>

Applying Algorithm 9.14 using the order B₆, B₅, B₄, B₃, B₂, B₁, we get the following solution in two iterations:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>B₁</td>
<td>-</td>
<td>abe</td>
<td>e</td>
<td>abe</td>
<td>e</td>
</tr>
<tr>
<td>B₂</td>
<td>-</td>
<td>abcde</td>
<td>abe</td>
<td>abcde</td>
<td>abe</td>
</tr>
<tr>
<td>B₃</td>
<td>-</td>
<td>abcde</td>
<td>abcde</td>
<td>abcde</td>
<td>abcde</td>
</tr>
<tr>
<td>B₄</td>
<td>-</td>
<td>-</td>
<td>abe</td>
<td>abcde</td>
<td>abce</td>
</tr>
<tr>
<td>B₅</td>
<td>-</td>
<td>bd</td>
<td>abcd</td>
<td>abde</td>
<td>abcd</td>
</tr>
<tr>
<td>B₆</td>
<td>-</td>
<td>-</td>
<td>bd</td>
<td>-</td>
<td>bd</td>
</tr>
</tbody>
</table>