1) Consider the following program:

```
procedure Main is
  X, Y, Z : Integer;

procedure Sub1 is
  A, Y, Z : Integer;
  begin -- of Sub1
      ......
  end; -- of Sub1

procedure Sub2 is
  A, B, Z : Integer;
  begin -- of Sub2
      ......
  end; -- of Sub2

procedure Sub3 is
  A, X, W : Integer;
  begin -- of Sub3
      ......
  end;-- of Sub3

begin -- of Main
      ......
end; -- of Main
```

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last subprogram activated? Include with each visible variable the name of the unit where it is declared.

a) Main calls Sub1; Sub1 calls Sub2; Sub2 calls Sub3.
b) Main calls Sub1; Sub1 calls Sub3.
c) Main calls Sub2; Sub2 calls Sub3; Sub3 calls Sub1.
d) Main calls Sub3; Sub3 calls Sub1.
e) Main calls Sub1; Sub1 calls Sub3; Sub3 calls Sub2.
f) Main calls Sub3; Sub3 calls Sub2; Sub2 calls Sub1.

Soln
### Variable Where Declared

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>A, X, W</td>
<td>Sub3</td>
</tr>
<tr>
<td></td>
<td>B, Z</td>
<td>Sub2</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>Sub1</td>
</tr>
<tr>
<td>(b)</td>
<td>A, X, W</td>
<td>Sub3</td>
</tr>
<tr>
<td></td>
<td>Y, Z</td>
<td>Sub1</td>
</tr>
<tr>
<td>(c)</td>
<td>A, Y, Z</td>
<td>Sub1</td>
</tr>
<tr>
<td></td>
<td>X, W</td>
<td>Sub3</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>Sub2</td>
</tr>
<tr>
<td>(d)</td>
<td>A, Y, Z</td>
<td>Sub1</td>
</tr>
<tr>
<td></td>
<td>X, W</td>
<td>Sub3</td>
</tr>
<tr>
<td>(e)</td>
<td>A, B, Z</td>
<td>Sub2</td>
</tr>
<tr>
<td></td>
<td>X, W</td>
<td>Sub3</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>Sub1</td>
</tr>
<tr>
<td>(f)</td>
<td>A, Y, Z</td>
<td>Sub1</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>Sub2</td>
</tr>
<tr>
<td></td>
<td>X, W</td>
<td>Sub3</td>
</tr>
</tbody>
</table>

2) Compare the tombstone and lock-and-key methods of avoiding dangling pointers, from the point of view of safety and implementation cost.

**tombstone:** The pointer variable points to a Tomb which interns points to the storage; so

![Tombstone Diagram]

*P is really *(p)*

Now suppose

```
P
```

q = p

Now suppose delete P
Lock and Key:

Basic concept:
– Associate a lock with each heap object
– Associate a key with each pointer
• When dereferencing a pointer:
  – Lock and key must match
• When copying a pointer:
  – Copy the key as well as the pointer value
• When freeing memory:
  – Change the lock on the memory

```
new (my_ptr);
  my_ptr  135942    135942

ptr2 := my_ptr;
  my_ptr  135942    135942
  ptr2  135942

delete (my_ptr);
  my_ptr  135942    0
  (Potentially reused)
  ptr2  135942
```
3) What significant justification is there for the \(\rightarrow\) operator in C and C++?

The only justification for the \(\rightarrow\) operator in C and C++ is writability. It is slightly easier to write \(p \rightarrow q\) than \((p).q\).

4) What are all of the differences between the enumeration types of C++ and those of Java?

In JAVA:

An *enum type* is a type whose *fields* consist of a fixed set of constants. Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week. Because they are constants, the names of an enum type's fields are in uppercase letters.

In the Java programming language, you define an enum type by using the *enum* keyword. For example, you would specify a days-of-the-week enum type as:

```java
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

The words SUNDAY etc are constants and cannot be used for anything else (like PASCAL).

C++ they would just represent integers (e.g., SUNDAY = 0, etc)