2. Advantage: “flexibility”; any variable can be used for any type values.

Disadvantage: “poor reliability” due to the ease with which type errors can be made, coupled with the impossibility of type checking detecting them.

5. Implicit heap-dynamic variables acquire types only when assigned values, which must be at runtime. Therefore, these variables are always dynamically bound to types.

8.

(a) i. Sub1
   ii. Sub1
   iii. Main

(b) i. Sub1
   ii. Sub1
   iii. Sub1

9. Static scoping: $x$ is 5.
   Dynamic scoping: $x$ is 10
10. **Variable** | **Where Declared**

**In Sub1:**
- A Sub1
- Y Sub1
- Z Sub1
- X Main

**In Sub2:**
- A Sub2
- B Sub2
- Z Sub2
- Y Sub1
- X Main

**In Sub3:**
- A Sub3
- X Sub3
- W Sub3
- Y Main
- Z Main
11. In Sub1:
   A  Sub1
   Y  Sub1
   Z  Sub1
   X  Main

In Sub2:
   A  Sub2
   X  Sub2
   W  Sub2
   Y  Main
   Z  Main

In Sub3:
   A  Sub3
   B  Sub3
   Z  Sub3
   X  Sub2
   W  Sub2
   Y  Main
12. Point 1:  
  a  1  
  b  2  
  c  2  
  d  2  

Point 2:  
  a  1  
  b  2  
  c  3  
  d  3  
  e  3  

Point 3: same as Point 1

Point 4:  
  a  1  
  b  1  
  c  1  

13. Variable  Where Declared
(a) d, e, f  fun3
   c        fun2
   b        fun1
a main
(b) d, e, f fun3
   b, c fun1
   a main
(c) b, c, d fun1
   e, f fun3
   a main
(d) b, c, d fun1
   e, f fun3
   a main
(e) c, d, e fun2
   f fun3
   b fun1
   a main
(f) b, c, d fun1
   e fun2
   f fun3
   a main
<table>
<thead>
<tr>
<th></th>
<th>Variable</th>
<th>Where Declared</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>
1. Boolean variables stored as single bits are very space efficient, but on most computers access to them is slower than if they were stored as bytes.

2. Integer values stored in decimal waste storage in binary memory computers, simply as a result of the fact that it takes four binary bits to store a single decimal digit, but those four bits are capable of storing 16 different values. Therefore, the ability to store six out of every 16 possible values is wasted. Numeric values can be stored efficiently on binary memory computers only in number bases that are multiples of 2. If humans had developed hands with a number of fingers that was a power of 2, these kinds of problems would not occur.

7. The main justification for the \(-\rightarrow\) operator in C and C++ is writability. It is slightly easier to write \(p \rightarrow q\) than \((p) \cdot q\).
Also, $(\ast p).q$ is less readable than $p \rightarrow q$ which illustrates the direction from $p$ to $q$ (In order to access $q$ one should pass from $p$)

8. Advantage: shows that unions are different from records.

Disadvantages:

1) it requires an additional reserved word and \\
2) unions are often separately defined but included in records, thereby complicating the program that uses them.

9. Let the subscript ranges of the three dimensions be named $\min(1), \min(2), \min(3), \max(1), \max(2), \text{and } \max(3)$.

Let the sizes of the subscript ranges be $\text{size}(1), \text{size}(2), \text{and } \text{size}(3)$. Assume the element size is 1.

Row Major Order:

$$\text{location}(a[i,j,k]) = (\text{address of } a[\min(1), \min(2), \min(3)])$$
Column Major Order:

\[
\text{location}(a[i,j,k]) = (\text{address of } a[\min(1),\min(2),\min(3)]) + ((k-\min(3)) \times \text{size}(1) + (j-\min(2)) \times \text{size}(2) + (i-\min(1))}
\]

14. Implicit heap storage recovery eliminates the creation of dangling pointers through explicit deallocation operations, such as \textit{delete}.

The disadvantage of implicit heap storage recovery is the execution time cost of doing the recovery, often when it is not even necessary (there is no shortage of heap storage).