MEMORY CONSERVATION AND NAME SPACE POLLUTION

Prior to the invention of scope and block-structured dynamic storage allocation, computer languages used *static storage allocation* in which all program data is allocated memory at the beginning of the run. *Scope and block-structured dynamic storage allocation* were invented around 1960 as a means of conserving internal memory, which at that time was astronomically expensive. The goal was to save memory through its reuse – when execution passes out of the scope of the data, its memory is reallocated to some other use, analogous to a hotel room that is used by different people on different nights.

**Example**
Consider an application in which memory has been partitioned into three methods – *main* has the memory that always must be available whereas *one* and *two* divide the remaining memory that must be available for specific computations.

This table below shows that the savings in memory usage can be upwards of .

<table>
<thead>
<tr>
<th>Item</th>
<th>Storage Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>$m$</td>
</tr>
<tr>
<td>one</td>
<td>$n$</td>
</tr>
<tr>
<td>two</td>
<td>$t$</td>
</tr>
<tr>
<td>Total under static allocation</td>
<td>$m + n + t$</td>
</tr>
<tr>
<td>Total under block-structured allocation</td>
<td>$m + \max(n, t)$</td>
</tr>
</tbody>
</table>

---

1 In 1960, one megabyte of core memory for an IBM 1401 cost over $5 million.
Internal memory is no longer prohibitively expensive. Nowadays scope is used to control namespace pollution, which occurs when the programmer of one part of a large program uses identifiers that conflict with those in other parts of the same program. A typical commercial computer program contains millions of lines of code, hundreds of classes and is maintained by leagues of programmers. Without the control provided by the scope mechanism, these large programs would have thousands of conflicts between the identically-named identifiers introduced by you, your fellow programmers and the creators of the Java programming language.

**Example**

Pretend your organization has written and maintains a large application, one function of which is to determine the current outdoor temperature and report it through a dialog box. The code fragment below shows the overall organization of the application. The display message is stored in field `report` declared at line 42 and whose scope extends to line 980. Method `displayReport` displays it as shown above right.

```java
public class ReportTemperature {
    . . . // MANY LINES

    public static String report;  // string to display
    . . . // MANY LINES

    public static void displayReport( )
    {
        JOptionPane.showMessageDialog( null, report );
    }

    . . . // MANY LINES

    980   }
```
**Example, continued**

Your boss wants you to modify the application so that it displays the temperature in degrees Celsius as shown in the message box to the right.

You know nothing about the program and the boss has told you nothing but “keep your changes to the `displayReport` method.”

How do you go about this? Looking at `displayReport`, you can deduce that the displayed string is in the `String` object `report`. Therefore, not knowing where or in what variable the temperature is kept, you can extract it from the string, convert it to Celsius and append the result onto the end of the string. The rules of scope guarantee that any declarations you make are restricted to this method; thus, preventing your code from having unintentional side effects within the rest of the program.

The following modified method will do the job.

```java
public static void displayReport( )
{
    double temp; // temporary for Celsius calculation
    Scanner scan = new Scanner( report );
    String s = scan.findInLine( "[0-9]+" );
    temp = Double.parseDouble( s );
    temp = (5.0 / 9.0) * (temp - 32.0);
    report += "F (" + String.format("%.0f",temp) + "\u00B0C)";
    JOptionPane.showMessageDialog( null, report );
}
```