Comp 524—Spring 2009

Exercise 4 Solutions

Due at 3:30 PM (on paper), March 26, 2009.

1. (20 pts) Ex. 7.19, p. 401. Hint: Due to alignment constraints, there is padding between array elements. Show your work.

Each member of the 2D array is a struct containing an int and a char. The int is 4 Bytes long and the char is 1 Byte long but we must pad with an additional 3 Bytes for alignment. Thus, each element is 8 Bytes.

To reach element A[3][7], we pass through three full rows and seven elements of one partial row. Since each row consists of ten elements, the space requirement for a row is 10 * 8B = 80B. Thus the address for A[3][7] can be found as follows:

Array start address:	1000B
+ preceding rows space:	+ (3 * 80B)
+ preceding elements space:	+ (7 * 8B)
---	---
Total:	1296B

2. (10 pts) Ex. 7.28, parts a and b, p.403.

We can regard objects as useless at some point in the code if they are never accessed after that point. These objects will not reclaimed by the garbage collector as long as there is a chain of references from the stack that can be followed to reference them.

In order to determine that such objects are useless, the compiler and/or run-time system would have to confirm that they are not used in paths of program execution yet to be taken. While this is possible for some (simple) programs, in the general case this is not possible. The problem is depends on the halting problem: It is undecidable whether a program will even halt on a finite input. Thus it is also not possible to determine which paths it will take nor which variables will be used. Intuitively, consider the difficulty of the problem when user input directs the path taken by the program.

3. (20 pts) Ex. 6.14, p. 300. Note: The figure is on p. 281.

We are asked to modify the code to implement in-order rather than pre-order enumeration. Here is one possible solution:

```
public class TreeNode<T> implements Iterable<T> {

      TreeNode<T> left;
```

```
	TreeNode<T> right;
	T val;

	public Iterator<T> iterator() {
		return new TreeIterator(this);
	}

	private class TreeIterator implements Iterator<T> {
		private Stack<TreeNode<T>> s = new Stack<TreeNode<T>>();

		TreeIterator(TreeNode<T> n) {
			TreeNode<T> currNode = n;

			// leftmost node is first in enumeration
			while (currNode != null) {
				s.push(currNode);
				currNode = currNode.left;
			}
		}

		public boolean hasNext() {
			return !s.empty();
		}

		public T next() {
			if (!hasNext()) {
				throw new NoSuchElementException();
			}

			TreeNode<T> n = s.pop();
			TreeNode<T> currNode = n;

			if (currNode.right != null) {
				currNode = currNode.right;

				// proceed to leftmost node in right subtree
				do {
					s.push(currNode);
					currNode = currNode.left;
				} while (currNode != null);
			}
			return n.val;
		}

		public void remove() {
			throw new UnsupportedOperationException();
		}
	}

}
```

4. (20 pts) Ex. 6.22, p. 301. Write your alternative in Java. Contrast it with Rubin's code in a well-written paragraph. Which is has fewer LOC (lines of code)? Which is more intuitive?

Examples of possible solutions are given in the paper aptly titled, ""GOTO Considered Harmful" Considered Harmful' Considered Harmful?""
http://www.ecn.purdue.edu/ParaMount/papers/acm_may87.pdf

Djikstra penned his own response later since he didn't feel that the paper above expressed his thinking on the matter. You can read that very formal paper here:
http://www.cs.utexas.edu/users/EWD/transcriptions/EWD10xx/EWD1009.html

I would add from my own point of view that LOC, though widely used, is an imperfect measure of code size. It is most useful when the code to be compared uses a standard coding style (especially important in free-form languages) and as an asymptotic measure of code size, i.e. O(1000) LOC.

Many people argue that the structured solution is more intuitive. I tend to agree. The impact of Djikstra's original paper and its argument can be felt in the lack of a goto statement in Java.

5. (30 pts) Write a well-written argument about a programming language construct that you consider harmful. In what situations does it lead to logic errors or poor code quality? Are there viable alternatives to that construct in the language? If so, do they cover all cases? If not, what additions or modifications would you make to a language to replace the offending construct? Give examples to illustrate your points. (Put some effort into this if you want full credit!)

Here are some of the things COMP524 students consider harmful:

The ternary operator
Operator overloading
Nested comments
Block comments
Public data members
Multiple inheritance
String object
Java serialization
Automatic type coercion
Java switch statement

I gave five points extra credit for a handful of responses that were exceptionally thorough.