

COSC2P05 Concepts of Programming Languages

Homework 3

Due: 14:00 pm EST, March 3, 2022 on Sakai

1. Explain why the concept of dynamic type binding is in fact closely tied to that of implicit heap-dynamic variables; you may use the following Javascript code as the basis of your explanation (it might also help you see why!):

```
var message;  
...  
message = "I mean to say this";  
...  
message = "nvm I mean to say THIS!"
```

2. When the following scope rules are applied, what is the output of the code?

- a. Static scope

- b. Dynamic scope

```
public class J {  
    int y = 1;  
    void fun1() {  
        int y = 2;  
        fun2();  
    }  
    void fun2() {  
        println("y = " + y)  
    }  
    public static void main(String[] args) {  
        new A().fun1();  
    }  
}
```

3. Given the Java program below, please
- list all static, stack-dynamic, explicit heap-dynamic, and implicit heap-dynamic variables.
 - At each marked computation step, show the contents of memory in terms of what variable is stored in static memory, system stack, and the heap during execution. Note that in the recursive and while-loop statements, you must show the contents in memory at **each** iteration or recursive step when the computation step is reached.
 - What is the output of this program, when run, in display? In one sentence, describe the purpose of this program (i.e., what does it do?)

```
public static void main ( String args[] ) {new Example();
};

public Example( ) {

    private ASCIIIDisplayer out;

    out = new ASCIIIDisplayer();
    dateTest();
}

public void dateTest ( ) {

    String message;

    public static final int YEAR = 2022;

    Date j;

    BinaryOutputFile fileOut;

    BinaryDataFile fileIn;

    j = new JulianDate(91);
    message = "The month is:" + j.monthInYearWhile();
    System.out.println(message);

    message = "The month is:" +
j.monthInYearRecursive(1);

    System.out.println(message);
}
```

→ computation step 1

→ computation step 6

→ computation step 2

→ computation step 3

```

public class JulianDate{

    public final int [] daysToMonth
    = {0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335};

    private int days;

    public JulianDate (int dayNum ) {

        days = dayNum;

    }

    public int monthInYearWhile(){

        int result;          // month in year

        result = 0;

        while ( days > daysToMonth[result] ) {

            result = result + 1;

            → computation step 4

        };

        return result;

    };

    public int monthInYearRecursive (int index) {

        int result;

        if (days <= daysToMonth[index]) {

            result = 1;

            return result;

        } else {

            → computation step 5

            return 1 + monthInYearRecursive(1 + index);

        }

    }

}

```

4. Given the following sample of a C program (it's not complete):

```
void sub1(void) {  
    int b, c, g;  
    ...  
}  
  
void sub2(void) {  
    ...  
}  
  
void sub3(void) {  
    int a, d, e;  
    ...  
}  
  
void main () {  
    int a, b, c, f;  
    ...  
}
```

For the calling sequences below, and assuming dynamic scoping, show the variables that are visible during the execution of the last subprogram activated
e.g., if the sequence is: main calls sub2; sub2 calls sub3; sub3 calls sub1

Answer is: (b, c, g : sub1; a, d, e: sub3, f: main)

- a) main calls sub3; sub3 calls sub1; sub1 calls sub2.
- b) sub3 calls main; main calls sub2; sub2 calls sub1.

5. Using C, write a small program to compare the efficiency of various storage binding methods. Write three functions in C:

- 1) One function, e.g., `arrayStatic(int 100)` declares an array of size 100 statically. In C, this
- 2) The second function, e.g., `arrayStack(int 100)` declares an array of size 100 on the stack.
- 3) The third function, e.g., `arrayHeap(int 100)` declares the same array of size 100 using the heap.

Call each of the functions a large number of times, say 100,000, and output the time required by each. Now, as with A1, find a value for the size of the array (perhaps larger than 100 in order to see a difference in the time of execution). Using multiple increasing values of this size (use at least 10 different values), graph the time it takes the functions to execute against the size of the array, n . Explain the results. Name your C program that has these three methods `arrayDeclarations.c`.

Note: A lot of the code you wrote in A1 would be helpful, especially that with the `clock()` function!

What to submit on Sakai:

1. A single pdf (e.g., `assignment3.pdf`) of your answers to all of the questions above (ideally typed, but you can also paste scans of handwritten answers on the pdf).
2. Your `arrayDeclarations.c` file.