

---

## Reading

---

1. Read Douglas Crockford's "A Survey of the JavaScript Programming Language" at <http://javascript.crockford.com/survey.html>. If you want more information about JavaScript, you can look at other articles on Douglas Crockford's site at <http://javascript.crockford.com/> or consult the *JavaScript 1.5 Guide* at [http://developer.mozilla.org/en/docs/Core\\_JavaScript\\_1.5\\_Guide](http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide).
2. Read pages 57–63, the first part of section 4.2, Lambda Calculus, in the textbook (*Concepts in Programming Languages*).
3. Read Chapter 7 on Scope, Functions, and Storage Management.

---

## Problems

---

### 1. .... JavaScript in the Browser

JavaScript is most commonly used to manipulate web pages in a browser. In this problem, we will experiment with using JavaScript in the browser by typing commands into the browser's address bar. This problem has been tested with Internet Explorer 6 and Firefox 2. For parts that require a browser, please note the browser and version you used.

- (a) To instruct a web browser to visit a certain web page, we often type `http` URIs in the browser's address bar. For example, to visit the web site of this course, you can type

```
http://cs242.stanford.edu/
```

in the address bar. Although lesser known, it is also possible to instruct browsers to run JavaScript by typing `javascript` URIs in the address bar. For example, try typing the following URIs into your browser's address bar.

```
javascript:'What <strong>does</strong> this do?'  
javascript:alert('What about <strong>this</strong>?');
```

Can you explain how the browser processes `javascript` URIs?

- (b) Please navigate your browser to <http://cs242.stanford.edu/> and type the following in the address bar:

```
javascript:alert(document.getElementById('title').src);
```

What does this command display? From this, what can we conclude about the context in which this JavaScript is running?

- (c) Now, try typing the following in the address bar (on one line):

```
javascript:void(document.getElementById('title').src =  
    'http://www.google.com/intl/en_ALL/images/logo.gif');
```

What happened? Why is it necessary to call `void`?

- (d) The course web site contains a "cross-site scripting" vulnerability on a page located at:

```
http://www.stanford.edu/class/cs242/resources/xss.html
```

For example, try visiting the URI below:

```
http://www.stanford.edu/class/cs242/resources/xss.html?alert('Hello,%20World!')
```

Suppose you wanted to trick a fellow CS 242 student into believing that the course had a different collaboration policy. Can you craft a URI that causes the course web site to display an alternate collaboration policy? Assume that your victim will visit a web page that you set up, and click on a link you write to access the CS242 web page. (Hint: All HTML elements have a property called `innerHTML` that represents the HTML contained within the element.)

## 2. .... Symbolic Evaluation

The Algol-like program fragment

```
function f(x)
  return x+4
end;
function g(y)
  return 3-y
end;
f(g(1));
```

can be written as the following lambda expression:

$$\left( \underbrace{(\lambda f. \lambda g. f (g 1))}_{\text{main}} \underbrace{(\lambda x. x + 4)}_f \right) \underbrace{(\lambda y. 3 - y)}_g$$

Reduce the expression to a normal form in two different ways, as described below.

- (a) Reduce the expression by choosing, at each step, the reduction that eliminates a  $\lambda$  as far to the *left* as possible.
- (b) Reduce the expression by choosing, at each step, the reduction that eliminates a  $\lambda$  as far to the *right* as possible.

## 3. .... Javascript higher-order functions

Functions `map` and `reduce` are standard functions from traditional functional programming that achieved broader recognition as a result of Google's MapReduce method for processing and generating large data sets. While `map`, `reduce`, and a number of related functions are provided in many JavaScript implementations, `map` and `reduce` can also be defined relatively simply in JavaScript as follows:

```
function map (f, inarray) {
  var out = [];
  for(var i = 0; i < inarray.length; i++) {
    out.push( f(inarray[i]) )
  }
  return out;
}

function reduce (f, inarray) {
  if(inarray.length <= 1) return;
  if(inarray.length == 2) return f(inarray[0], inarray[1]);
  r = inarray[0];
  for(var li = 1; li < inarray.length; li++){
    r = f(r, inarray[li]);
  }
  return r;
}
```

Function `map(f, inarray)` returns an array constructed by applying `f` to every element if `inarray`. Function `reduce(f, inarray)` applies the function `f` of two arguments to elements in the list, from left to right, until it reduces the list to a single element. For example:

```
js> map( function(x){return x+1}, [1,2,3,4,5])
2,3,4,5,6
js> reduce( function(x,y){return x+y}, [1,2,3,4,5])
15
js> reduce( function(x,y){return x*y}, [1,2,3,4,5])
120
```

These functions can be combined in various ways.

- (a) Show how to use `map` and `reduce` to compute the sum of the first five squares, in one line. (The sum of the first three squares is  $1^2 + 2^2 + 3^2$ .)
- (b) Show how to use `map` and `reduce` to count the number of positive numbers in an array of numbers.
- (c) Show how to use `map` and/or `reduce` to “flatten” an array of arrays of numbers, such as `[[1,2],[3,4],[5,6],[7,8,9]]`, to an array of numbers.

#### 4. .... Activation Records for Inline Blocks

You are helping a friend debug a C program. The debugger `gdb`, for example, lets you set breakpoints in the program, so the program stops at certain points. When the program stops at a breakpoint, you can examine the values of variables. If you want, you can compile the programs given in this problem and run them under a debugger yourself. However, you should be able to figure out the answers to the questions by thinking about how activation records work.

- (a) Your friend comes to you with the following program, which is supposed to calculate the absolute value of a number given by the user.

```
1:     int main()
2:     {
3:         int num, absVal;
4:
5:         printf(``Absolute Value\n``);
6:         printf(``Please enter a number:``);
7:         scanf(``%d``, &num);
8:
9:         if (num >= 0)
10:        {
11:            int absVal = num;
12:        }
13:        else
14:        {
15:            int absVal = -num;
16:        }
17:
18:        printf(``The absolute value of %d is %d.\n\n``, num, absVal);
19:
20:        return 0;
21:    }
```

Your friend complains that this program just doesn't work and shows you some sample output:

```
cardinal:~> gcc -g test.c
cardinal:~> ./a.out
Absolute Value
Please enter a number: 5
The absolute value of 5 is 4.
```

```
cardinal:~> ./a.out
Absolute Value
Please enter a number: -10
The absolute value of -10 is 4.
```

**Being a careful student, your friend has also used the debugger to try to track down the problem. Your friend sets a breakpoints at line 11 and at line 18 and looks at the address of absval. Here is the debugger output:**

```
cardinal:~> gdb a.out
(gdb) break test.c:11
(gdb) break test.c:18
(gdb) run
Starting program:
Absolute Value
Please enter a number: 5

Breakpoint 1, main (argc=1, argv=0xffffb04) at test.c:11
11      int absVal = num;
(gdb) print &absVal
$1 = (int *) 0xffffbfa84
(gdb) c
Continuing.
```

```
Breakpoint 2, main (argc=1, argv=0xffffb04) at test.c:18
18      printf(`The absolute value of %d is %d.\n\n',num absVal);
(gdb) print &absVal
$2 = (int *) 0xffffbfa88
```

**Your friend swears that the computer must be broken, since it is changing the address of a program variable. Using the information provided by the debugger, explain to your friend what the problem is.**

- (b) **Your explanation must not have been that good, because your friend does not believe you. Your friend brings you another program:**

```
1:      int main()
2:      {
3:          int num;
4:
5:          printf(`Absolute Value\n');
6:          printf(`Please enter a number:');
7:          scanf(`%d',&num);
8:
9:          if (num >= 0)
10:         {
11:             int absVal = num;
12:         }
13:         else
14:         {
15:             int absVal = -num;
16:         }
17:
18:         {
19:             int absVal;
```

```

20:         printf(`The absolute value of %d is %d.\n\n`, num, absVal);
21:     }
22: }

```

This program works. Explain why.

(c) Imagine that line 17 of the program in part (b) were split into three lines:

```

17a:     {
17b:     ...
17c:     }

```

Write a single line of code to replace the ... that would guarantee this program would NEVER be right. You may not declare functions or use the word “absVal” in this line of code.

(d) Explain why the change you made in part (c) breaks the program in part (b).

## 5. .... Parameter Passing

Consider the following procedure, written in an Algol/Pascal-like notation:

```

proc power(x, y, z:int)
begin
    z := 1
    while y > 0 do
        z := z*x
        y := y-1
    end
end

```

The code that makes up the body of `power` is intended to calculate  $x^y$  and place the result in `z`. However, depending on the actual parameters, `power` may not behave correctly for certain combinations of parameter-passing methods. For simplicity, we only consider call-by-value and call-by-reference.

- (a) Assume `a` and `c` are assignable integer variables with distinct L-values. Which parameter-passing methods make  $c = a^a$  after a call `power(a, a, c)`. You may assume that the R-values of `a` and `c` are non-negative integers.
- (b) Suppose that `a` and `c` are formal parameters to some procedure `P`, and that the expression `power(a, a, c)` above is evaluated inside the body of `P`. If `a` and `c` are passed to `P` by reference, and become aliases, then what parameter passing method(s) will make  $c = a^a$  after a call `power(a, a, c)`? If, after the call,  $c = a^a$ , does that mean that `power` actually performed the correct calculation?

## 6. .... Function Calls and Memory Management

This question asks about memory management in the evaluation of the following code. Assume that the final `{, }` on the last line of code creates a separate scope, even though this is not the standard semantics of JavaScript. (As discussed in class, a block creating a new scope can be written in JavaScript using an anonymous function.)

```

var x = 5;
function f(y) {return (x+y)-2};
fun g(h){var x = 7; return h(x)};
{var x = 10; g(f)};

```

- (c) Fill in the missing information in the following depiction of the run-time stack after the call to `h` inside the body of `g`. Remember that function values are represented by closures, and that a closure is a pair consisting of an environment (pointer to an activation record) and compiled code.

In this drawing, a bullet ( $\bullet$ ) indicates that a pointer should be drawn from this slot to the appropriate closure or compiled code. Since the pointers to activation records cross and could become difficult to read, each activation record is numbered at the far left. In each activation record, place the number of the activation record of the statically enclosing scope in the slot labeled “access link.” The first two are done for you. Also use activation record numbers for the environment pointer part of each closure pair. Write the values of local variables and function parameters directly in the activation records.

<i>Activation Records</i>	<i>Closures</i>	<i>Compiled Code</i>												
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center; width: 10%;">(1)</td> <td style="width: 60%;">access link</td> <td style="width: 30%; text-align: center;">( 0 )</td> </tr> <tr> <td></td> <td>x</td> <td></td> </tr> </table>	(1)	access link	( 0 )		x									
(1)	access link	( 0 )												
	x													
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">(2)</td> <td>access link</td> <td style="text-align: center;">( 1 )</td> </tr> <tr> <td></td> <td>f</td> <td style="text-align: center;"><math>\bullet</math></td> </tr> </table>	(2)	access link	( 1 )		f	$\bullet$								
(2)	access link	( 1 )												
	f	$\bullet$												
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">(3)</td> <td>access link</td> <td style="text-align: center;">( )</td> </tr> <tr> <td></td> <td>g</td> <td style="text-align: center;"><math>\bullet</math></td> </tr> </table>	(3)	access link	( )		g	$\bullet$	$\langle ( ), \bullet \rangle$	code for f						
(3)	access link	( )												
	g	$\bullet$												
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">(4)</td> <td>access link</td> <td style="text-align: center;">( )</td> </tr> <tr> <td></td> <td>x</td> <td></td> </tr> </table>	(4)	access link	( )		x		$\langle ( ), \bullet \rangle$							
(4)	access link	( )												
	x													
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">(5)</td> <td>g(f)</td> <td>access link</td> <td style="text-align: center;">( )</td> </tr> <tr> <td></td> <td></td> <td>h</td> <td style="text-align: center;"><math>\bullet</math></td> </tr> <tr> <td></td> <td></td> <td>x</td> <td></td> </tr> </table>	(5)	g(f)	access link	( )			h	$\bullet$			x			code for g     ... ..
(5)	g(f)	access link	( )											
		h	$\bullet$											
		x												
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">(6)</td> <td>h(x)</td> <td>access link</td> <td style="text-align: center;">( )</td> </tr> <tr> <td></td> <td></td> <td>y</td> <td></td> </tr> </table>	(6)	h(x)	access link	( )			y							
(6)	h(x)	access link	( )											
		y												

- (b) What is the value of the call `g(f)`? Why?