Exam 2 Practice Problems

CS 340 → Practice Problems → Exam 2 Practice Problems

Note: when considering a type as a set of values $S$, the cardinality of the type, denoted $S\#$, is the number of values belonging to the type.

(1) Consider the following C program:

```c
#include <stdio.h>

int sumDigits(int n) {
    if (n < 10) {
        return n;
    } else {
        return (n % 10) + sumDigits(n / 10);
    }
}

int main(void) {
    int x = 90125;
    int result = sumDigits(x);
    // (HERE)
    printf("%d\n", result);
    return 0;
}
```

(a) Show the stack of activation records in the program at the point where the maximum number of simultaneous activation records exist when the program runs.

(b) List all bindings of names in the environment at the point labeled (HERE). For each binding, list the identifier, type, and entity to which the identifier is bound. (For example, the name "x" is bound to a local variable whose type is `int`.)

(2) Consider the following program written in a C-like programming language:

```c
int f(int x) {
    x = 42;
}

int main(void) {
    int q = 17;
    f(q);
    printf("%d\n", q);
}
```

(a) If the program prints 17, what is the default parameter-passing mechanism? `by-value`

(b) If the program prints 42, what is the default parameter passing mechanism? `by-reference` or `copy-out`

(3) Consider the following C declarations:

```c
enum Component { RED, GREEN, BLUE };
```
(a) Using set notation, specify the set of values that belong to the types `Component` and `Color`.

(b) Using set notation, specify the set of values that make up the types of the `myColors` array and the `intensity` function. (Assume that `intensity` is a function procedure, i.e., it has no side effects.)

(c) Assume that `#double` is \(2^{64}\). What is the cardinality of the `Color` type? \(\#\text{Color} = 2^{64} \times 2^{64} \times 2^{64} = 2^{192}\)

(4) Consider the following enumerated types:

```c
enum S { A, B };  
enum T { X, Y, Z };  
```

(a) Cross out the values below which are not members of the mapping type \(S \rightarrow T\):

- \{(A, X), (A, Y)\}
- \{(A, X), (B, Y)\}
- \{(X, A), (Y, B)\}
- \{(X, A), (Y, A), (Z, A)\}
- \{(X, A), (Y, B), (Z, B)\}
- \{(A, X), (B, X)\}

(b) What is the cardinality of the type \(S \rightarrow T\)? \(\#S = 2\) \(\#(S \rightarrow T) = \#T = 3\)

(5) Consider the following C++ program:

```c
void merge(int dest[], int lh[], int lnum, int rh[], int rnum) {  
 ...no function calls in merge...  
}

void mergesort_work(int arr[], int temp[], int n) {  
if (n < 2) { return; }  
mergesort_work(arr, temp, n/2);  
mergesort_work(arr + n/2, temp + n/2, n-(n/2));  
merge(temp, arr, n/2, arr + n/2, n-(n/2));  
for (int i = 0; i < n; i++) {  
arr[i] = temp[i];  
}
}

void mergesort(int arr[], int n) {  
int temp[n];  
mergesort_work(arr, temp, n);
}

int main(void) {  
int arr[] = {59, 8, 37, 73, 85, 61, 52, 39, 15, 80};  
mergesort(arr, 10);
```
(a) What is the maximum number of activation records that will exist when the program runs? Explain briefly.

(b) Recall that one of the values stored in an activation is the return address, which is the address of the machine instruction to which the called procedure will return. How many distinct return address values are possible in activation records for the mergesort_work function? Explain briefly. (Because it is called in 3 different places: once in mergesort, twice in mergesort_work.)

(6) Consider the following context-free grammar with the start symbol E, nonterminal symbols E IDA L, and terminal symbols ( ) a b 1 2 . . .

Assume that you are writing a recursive descent parser for this grammar. Assume you have a lexer supporting hasMoreTokens(), peek(), and get() operations. Use detailed pseudo-code to show how the following parse functions would be implemented. You don't have to build the parse tree, but do show all lexer operations and calls to parse functions.

(a) parseE()

(b) parseL()

(7) Write a Scheme function called select-members which takes two parameters:

- a list
- a function

The select-members function should apply the function to each member of the list. The function will return a true (#t) or false (#f) value. Each list member for which the function returns true should be a member of the list returned by select-members.

Example:

```scheme
guile> (define select-members
  (lambda (l fn)
    (if (null? l)
        l
        (cons (car l) (select-members (cdr l) fn))))

guile> (select-members ' (1 2 3 4 5 6 7 8 9) odd?)
#f
12/03/2009 09:31 AM
```

3 of 4
(8) Write a tail-recursive version of the select-members function described in problem (6). Hint: use the following top-level function

```scheme
(define select-members
  (lambda (lst fn)
    (select-members-work lst fn '()))
)
```

and the recursive worker function

```scheme
(define select-members-work
  (lambda (lst fn accum)
    ...
  )
)
```

Note: tail-recursive list processing functions typically return values in the reverse of their original order. You may use the built-in reverse function to ensure that the values are returned in their original order.

```scheme
(define select-members
  (lambda (lst fn)
    (reverse (select-members-work lst fn '()))
  )
)

(define select-members-work
  (lambda (lst fn accum)
    (if (null? lst)
        accum
        (if (fn (car lst))
            (cons (car lst) accum)
            (select-members-work (cdr lst) fn
                                   (select-members-work (cdr lst) fn accum))
        )
  )
)
```