

CMPS 12B

Introduction to Data Structures

Quiz 4 Review Problems

1. The following C program includes a global variable called `time`. Since it is declared outside of all functions (on line 3), its scope is the entire file. Notice that `time` is incremented before each of the functions `a`, `b`, and `c` return. Show the state of the function call stack when `time=4` (i.e. at the instant `time` becomes equal to 4). Each stack frame should show the values of all function arguments and local variables, as well as the line to which execution will transfer when the function returns. If a local variable has not yet been assigned a value at `time=4`, indicate that fact by stating its value as `undef`. Assume that the terms in the right hand side of line 19 are evaluated from left to right, i.e. first call `a()` then call `b()`. Also determine the program output, and print it on the line below exactly as it would appear on the screen.

```
1.)  #include<stdio.h>
2.)  #include<stdlib.h>
3.)  int time;
4.)  int a(int x){
5.)      int i;
6.)      i = x*x;
7.)      time++;
8.)      return(i);
9.)  }
10.) int b(int y){
11.)     int j;
12.)     j = y+a(y);
13.)     time++;
14.)     return(j);
15.) }
17.) int c(int z){
18.)     int k;
19.)     k = a(z)+b(z); // first call a() then b()
20.)     time++;
21.)     return(k);
22.) }
23.) int main(void){
24.)     int q, r;
25.)     time = 0;
26.)     q = b(5);
27.)     r = c(2);
28.)     printf("q=%d, r=%d, time=%d\n", q, r, time);
29.)     return(EXIT_SUCCESS);
30.) }
```

Output:

State of the function call stack when `time=4`:

2. Consider the following C program.

```
#include<stdio.h>
#include<stdlib.h>

int main(void){
    int i, j;
    double x = 4.2, y;
    double * A = calloc(4, sizeof(double));
    double B[] = {1.2, 5.3, 2.1, 3.4};
    double *p, *q;

    p = malloc(sizeof(double));
    y = x+2;
    q = &y;
    *p = *q + 2.5;

    for(i=0; i<4; i++){
        j = 3-i;
        *(A+i) = B[j] + i;
    }
    printf("%f, %f, %f, %f\n", *A, *B, *p, *q);
    A = B;
    printf("%f, %f, %f, %f\n", *A, *(A+1), *(A+2), *(A+3) );
    return(EXIT_SUCCESS);
}
```

- a. Write the output of this program exactly as it would appear on the screen:

- b. List the pointer variables in this program, and for each one, state whether it points to stack memory or heap memory. If at some point in the program the pointer changes from stack to heap or heap to stack, note the point in the program where that happens.
- c. Does this program contain any memory leaks? If so, what alteration(s) would be needed to eliminate those leaks?

3. Write a C function called `search()` with the prototype below that takes as input a null terminated char array `S` (i.e. a string) and a single char `c`, and returns the leftmost index in `S` at which the target `c` appears, or returns -1 if no such index exists.

```
int search(char* S, char c){  
  
}
```

4. Write a C function called `diff()` with the prototype below that takes as input two null terminated char arrays (i.e. strings) `A` and `B` and returns the *difference* string consisting of all chars in `A` that are not in `B`. The returned chars should be stored in a null terminated char array in the same order they appeared in `A`, possibly with repetitions. The returned array should be allocated from heap memory to be the same length as `A` (although the null terminator might not appear at the end of this array.) You may use the `strlen()` function found in the library `string.h` to determine this length. You may also assume the existence of a C function called `search()` as described in the previous problem.

```
char* diff(char* A, char* B){  
  
}
```

5. For each of the Java programs in problems 6 and 7 of the quiz 1 review, do the following:
- Translate the program into C.
 - Perform a box trace of the recursive function (`getValue(3, 13, 5)` and `displayOctal(100)` respectively).
 - Perform a stack trace for the recursive function (`getValue(3, 13, 5)` and `displayOctal(100)` respectively).
6. Consider the C function below called `wasteTime()`. Your goal is to determine how much time `wasteTime()` wastes. The starred (*) lines below are to be considered basic operations, which do nothing but waste a multiple of some unspecified time unit. Determine the total amount $T(n)$ of time wasted on the input n . Find the asymptotic runtime of this algorithm, i.e. $T(n) = \Theta(\text{some simple function of } n)$.

```
void wasteTime(int n){  
    int i, j, k;  
    *   waste 2 units of time;  
    for(i=0; i<n; i++){  
    *       waste 5 units of time;  
        for(j=0; j<n j++){  
    *           waste 12 units of time;  
            for(k=0; k<n; k++){  
    *               waste 3 units of time;  
            }  
        }  
    }  
}
```

7. Write a C function called `CountComparisons()` that takes as input an `int` array `A`, and `int n` giving the length of `A`, and an `int i` specifying an index to `A`. The function will return an `int` giving the number of elements in `A` that are less than `A[i]`. Determine the number of comparisons performed by your function (in terms of the array length n).

```
int CountComparisons(int* A, int n, int i){  
    // your code goes here  
}
```