Exam 1

March 2nd, 2005

Work alone. Do not use any notes or books. You have approximately 60 minutes to complete this exam.

Please write your answers on the exam. More paper is available if you need it. Please put your name at the top of the first page.

1 Output

What output is produced by the following program?

```
#include <stdio.h>

int
main(int argc, char **argv)
{
    int i;
    int j;

    for(i = 0; i < 10; i++) {
        if(i % 3 != 0) continue;
        for(j = i+1; j > 0; j /= 2) {
            printf("%d %d\n", i, j);
            if(j == 5) break;
        }
    }
}
```

Solution

```
0 1
3 4
3 2
3 1
6 7
6 3
6 1
9 10
9 5
```

2 Bugs (20 points)

The function `revdup` is intended to be a reversing version of `strdup`: given a null-terminated string `"abc"`, it returns a freshly-malloc'd null-terminated string `"cba"`. The file below compiles without errors with the command `gcc -Wall -ansi -pedantic -c revdup.c`. Nonetheless, it contains at least four errors that will prevent `revdup` from working as advertised. Identify as many of these errors as you can and provide a working version of this code.

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

/* return a newly-malloc'd copy of s */
/* in reverse order */
/* or 0 if malloc fails */
char *
revdup(const char *s)
{
    char *r;
    int i;
    int len;

    len = strlen(r);
    r = malloc(len);

    if(r == 0) return 0;

    for(i = 0; i < len; i++) {
        r[i] = s[len-i];
    }

    return r;
}
```

Solution

1. `strlen(r)` should be `strlen(s)`.
2. `malloc(len)` should be `malloc(len+1)` to leave room for the nul.
3. `r[i] = s[len-i];` should be `r[i] = s[len-i-1];`
4. Before returning `r`, there needs to be a line `r[len] = '\0';` to add a nul terminator to the string.

Here is the revised code:

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

/* return a newly-malloc'd copy of s */
/* in reverse order */
/* or 0 if malloc fails */
char *
revdup(const char *s)
{
    char *r;
    int i;
    int len;

    len = strlen(s);
    r = malloc(len+1);
    if(r == 0) return 0;

    for(i = 0; i < len; i++) {
        r[i] = s[len-i-1];
    }

    r[len] = '\0';

    return r;
}
```

3 Mediocrity

Suppose that you are asked to write a function that takes a set of n distinct `int`s, where n is odd, and returns an `int` x from the set such that exactly $(n-1)/2$ of the elements of the set are less than x. Before taking the assignment, you are allowed to specify what form the input to your function should take. Assuming that your goals are to minimize both programmer and CPU time, which one of the following data structures would be the **worst** way to organize the elements of the set and why? Assume in each case that n is also provided as an argument to your function.

1. As the elements of a sorted singly-linked list.
2. As the elements of an unsorted array.
3. As the keys of a hash table using open addressing.
4. As the elements of a sorted array.

Solution

The worst is as the keys of a hash table. Comparing costs for each approach:

1. Requires walking through $(n-1)/2$ linked list elements to find the median at cost $O(n)$, but the constant is small and the loop is about four lines of code.
2. Requires sorting (which takes $O(n \log n)$ time using `qsort`), but then median can be read directly out of the array. Code length is again pretty short, depending on how the sort is done, but the costs make this the second-worst option.
3. Basically equivalent to unsorted array with the added complexity of having to process and ignore empty hash table slots. The hash table functionality adds nothing that is useful for solving this problem.
4. Best choice: median computation is now just `return a[(n-1)/2];`.

So ordering by decreasing desirability we have $4 > 1 > 2 > 3$.

4 Census (20 points)

Write a function `census` that takes two null-terminated strings as arguments and returns a count of the number of characters in the first argument `haystack` that also appear at least once in the second argument `needles`. For example, `census("badcad", "abc")` should return 4 (all the characters except the two d's count), but `census("abracadabra", "q")` and `census("fish", "")` should both return 0. We have provided the function header for you.

Solution

The main tricky part is to make sure the a character that appears more than once in `needles` doesn't get overcounted. We can do this with a `break` in the right place.

```
int
census(const char *haystack, const char *needles)
{
    const char *hay;
    const char *needle;
    int count;

    count = 0;

    for(hay = haystack; *hay; hay++) {
        for(needle = needles; *needle; needle++) {
            if(*hay == *needle) {
                count++;
                break;
            }
        }
    }

    return count;
}
```