Name: _____ Precept (circle one): 1a 1b 2 3a 3b 4

COS 126 Fall 1996 **Nov. 21, 1996**
Second Midterm Examination

Write your name and indicate your precept number on all pages of this exam. We'll separate the pages during grading, so your name must appear on every page. Also, please sign the pledge:

I pledge on my honor that I have not violated the honor code during this examination.

_____

1. (5 pts) Give the 16-bit, two's-complement, binary representation for -100_{10}.

2. (5 pts) What is the output of the TOY program shown in the right margin?

```
10
10: B110
11: 8620
12: 4102
13: 0000
20: 9160
21: 4102
22: B201
23: 1226
24: 7200
```

3. (5 pts) Show the result of the Quicksort partitioning pass for the input `5 8 7 6 1 9 3 2 4`. Do just *one* partition, not the whole sort.

4. (5 pts) If Quicksort's partitioning process divides the input array in half, approximately how many recursive calls are made to sort N values?

5. (5 pts) Give a one-sentence description of the effect of the following recursive function.

```
int f(unsigned n) {
   if (n != 0) return f(n/2) + (n&1);
   return 0;
}
```

6. (5 pts) If an N^2 algorithm can process each input item in 1 microsecond, how many items can it process in 1 hour? That is, given 1 hour of time, what is N?

7. (5 pts) `struct word { char *str; int count; } *ptr` points to an array of `word` structures with `size` > 0 elements. The code below purports to double the size of this array:

```
size *= 2;
ptr = erealloc(ptr, size);
```

   This code is incorrect; fix the erroneous line above.

8. (5 pts) The code below prints the `n` counts and words in the `ptr` array described in the previous problem and deallocates the strings, then deallocates the array:

```
for ( ; n-- > 0; ptr++) { printf("%d\t%s\n", ptr->count, ptr->str); free(ptr->str); }
free(ptr);
```

   Is this code correct? If not, explain the bug in one sentence, and fix it.

Name: _____ Precept (circle one): 1a 1b 2 3a 3b 4

9. (5 pts) The code below prints the words in the input. `getword(char *word, int size)` reads the next word into `word[0..size-1]` and returns its length or `EOF`.

```
char *word;
while (getword(word, 100) != EOF) printf("%s\n", word);
```

Give a one-sentence description of the serious bug in this code.

10. (5 pts) `reverse(x, y, len)` copies `len` elements from `y` into `x` in reverse order. Here's the solution from the first midterm:

```
void reverse(int *x, int *y, int len) {
   int i, j = len - 1, t;
   for (i = 0; i < len; i++) x[i] = y[i];
   for (i = 0; i < j; i++) { t = x[i]; x[i] = x[j]; x[j--] = t; }
}
```

This version *still* has a serious bug when `reverse` is called with certain nonnull arguments. Give a call to `reverse` involving `a[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}` that doesn't work.

11. (10 pts) Write a TOY fragment that sets R3 to the sum of the integers in the linked list pointed to by R2. Nodes occupy two consecutive words: the first word holds an integer, and the second holds the address of the next node. The null pointer is represented by 0. For example, the TOY fragment in the margin is a 4-node list of the integers 1–4. For consistency, start your solution at address 01.

```
30:  1
31: 36
36:  2
37: 24
24:  3
25: 26
26:  4
27:  0
```

12. (10 pts) Write a code fragment that deallocates *all* of the nodes pointed to by `list`, declared below, and sets `list` to `NULL`. Be sure to handle the empty list and declare variables you use.

```
struct item { int info; struct item *link; } *list;
```

13. (10 pts) `treecount(tree)` returns the number of nodes in `tree`, which is a binary search tree. Fill in the body of `treecount` below.

```
struct node { int info; struct node *left, *right; };
int treecount(struct node *tree) {
```

14. (10 pts) `concat(s1, s2)` returns a dynamically allocated string that holds the concatenation of the strings `s1` and `s2`. For example, `concat("hello␣", "world")` returns `"hello␣world"`, where ␣ denotes a space. Fill in the body of `concat` below. You *may* call other functions.

```
char *concat(char *s1, char *s2) {
```

15. (10 pts) `itoa(n, str)` fills `str[0...]` with the decimal representation of `n` and returns `str`, *unless* `str` is null. When `str` is null, `itoa` allocates the space for the string, fills it with the decimal representation of `n`, and returns the string. Fill in the body of `itoa` below. You *may* call other functions. Assume that a nonnull `str` points to enough space to hold the result.

```
char *itoa(int n, char *str) {
```