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

COS 126 Fall 1996 **Jan. 20, 1997**
Final Examination

Write your name *legibly* 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.

_____

Correct answers to problems 1–12 are each worth 5 points, no answer is worth 0 points, and incorrect answers are worth −1 point. Be careful!

1. The 16-bit, two's-complement representation for -132_{10} is

   (a) 0084_{16}
   (b) $FF7B_{16}$
   (c) 177574_8
   (d) 177604_8
   (e) None of the above

2. The output of the TOY program shown to the right below is

   (a) 4102
   (b) 4602 0012
   (c) 0010 0012
   (d) 0012 0012
   (e) 4602 4102

```
10
10:  B110
11:  8620
12:  4602
13:  0000
20:  9160
21:  4102
22:  7600
```

3. If Quicksort uses 5, the *leftmost* value in the input `5 8 7 6 1 9 3 2 4`, as the pivot element, the result of just *one* partitioning step is

   (a) `5 8 7 6 1 9 3 2 4`
   (b) `2 3 1 4 7 9 8 5 6`
   (c) `5 4 2 6 1 9 3 7 8`
   (d) `1 4 2 3 5 9 6 7 8`
   (e) `5 4 7 6 1 9 3 2 8`

4. The function below computes Fibonaci numbers. How many recursive calls are made to compute `f(5)`, not counting the initial call to `f(5)`?

   (a) 14
   (b) 8
   (c) 7
   (d) 15
   (e) 18

```
int f(int n) {
   if (n < 2) return 1;
   return f(n-1) + f(n-2);
}
```

5. What does the recursive function shown to the right below return?

   (a) The number 0 bits in `n`.
   (b) The number 1 bits in `n`.
   (c) The sum of the contiguous 3-bit sequences in `n`.
   (d) The sum of the elements of `b`.
   (e) The sum of the elements of `b[`i`]` for each 3-bit sequence i in `n`.

```
int b[]={0, 1, 1, 2, 1, 2, 2, 3};
int f(unsigned n) {
   int k = b[n&7];
   if (n != 0) k += f(n>>3);
   return k;
}
```

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

6. `duplicate` returns 1 if there is a duplicate value in `x[0..N-1]`. The worst-case running time of `duplicate` is about

   (a) N^2
   (b) N^3
   (c) N
   (d) $N \lg N$
   (e) $\lg N$

```
int duplicate(int x[], int N) {
   int i;
   quicksort(x, 0, N-1);
   for (i = 0; i < N-2; i++)
      if (x[i] == x[i+1]) return 1;
   return 0;
}
```

7. Suppose a file system restricts data block numbers to 16 bits. The smallest data block size on a 1 GB (2^{30} bytes) disk is

   (a) 512 bytes (b) 65526 bytes (c) 8 KB (d) 16 KB (e) 32 KB

8. `struct word { char *str; int count; } *ptr` points to a dynamically allocated array of `word` structures. The code below prints `n` counts and words and deallocates the strings and structures.

```
for (i = 0; i < n; i++) {
   printf("%d\t%s\n", ptr[i].count, ptr[i].str);
   free(ptr[i].str);
   free(ptr[i]);
}
```

   This code is incorrect because

   (a) It does not deallocate the array.
   (b) `ptr[i]` does not point to a dynamically allocated structure.
   (c) `ptr[i].str` is not a dyamically allocated string.
   (d) `ptr[i].str` is deallocated twice.
   (e) All of the above.

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

```
char *word = emalloc(sizeof (char *));
while (getword(word, 200) != EOF) printf("%s\n", word);
```

   This code is incorrect because

   (a) The space pointed to by `word` is too small.
   (b) `word` is uninitialized.
   (c) The memory pointed to by `word` is for a character pointer, not for an array of characters.
   (d) `getword` can't change the memory pointed to by `word`.
   (e) `word` isn't an array of characters.

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

10. `reverse(x, y, len)` copies `len` elements from `y` into `x` in reverse order:

```
void reverse(int *x, int *y, int len) {
   int i;
   if (len > 0 && x >= y && x < y + len) {
      int *temp = emalloc(len*sizeof (int));
      for (i = 0; i < len; i++) temp[i] = y[i];
      reverse(x, temp, len);
      free(temp);
   } else
      for (i = 0; i < len; i++) x[i] = y[len-i-1];
}
```

Given `a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }`, which call illustrates the flaw in `reverse`.

(a) `reverse(a, a, 10)`
(b) `reverse(a + 1, a, 8)`
(c) `reverse(a + 4, a, 6)`
(d) `reverse(a + 8, a + 2, 2)`
(e) None of the above; the function is correct.

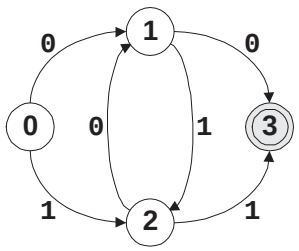
11. The *object code* shown to the right below is a TOY program that computes the sum of the integers from `M` to `N`, which are the values stored in the locations indicated. The instructions in this program that must be relocated by the linker are those at locations

(a) 00, 0D, 0E
(b) 01, 06, 0A
(c) 01, 06, 0A, 0D, 0E
(d) 01, 02, 03
(e) 00, 01, 06, 0A, 0D, 0E

```
00:  B001 =MAIN
01:  B10D
02:  9210
03:  9111
04:  2112
05:  B300
06:  610B
07:  1332
08:  1220
09:  2110
0A:  5006
0B:  4302
0C:  0000
0D:  00   =M
0E:  0A   =N
```

12. The regular expression that describes the language accepted by the FSA below is

(a) (0+1)((10)*0 + (01)*1)
(b) 0((10)*(0+11)) + 1((01)*(1+00))
(c) 0(10)*0 + 1(01)*1
(d) (0+1)((10*)+(01)*)(0+1)
(e) None of the above.



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

13. (10 pts) `listtoarray(list, last)` builds an $n+1$-element array that holds the n integers in the linked list `list` in elements 0 to $n-1$ and the value of `last` in element n, and it returns a pointer to the array. For example, if `list` holds `1`, `2`, `3`, `listtoarray(list, -1)` returns a pointer to the first element of the array `{ 1, 2, 3, -1 }`, and if `list` is empty, `listtoarray(list, -1)` returns a pointer to the one-element array `{ -1 }`. Fill in the body of `listtoarray` below.

```
struct item { int info; struct item *link; };
int *listtoarray(struct item *list, int last) {
```

14. (10 pts) `treefree(tree)` deallocates *all* the nodes in `tree`, which is a binary search tree. Fill in the body of `treefree` below.

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

15. (10 pts) `dup(n, s)` returns a dynamically allocated string that holds the concatenation of `n` copies of the nonnull string `s`. If `n`$\leq$0, `dup` returns the empty string. For example, the call `dup(3,"help␣")` returns `"help␣help␣help␣"`, where ␣ denotes a space, and `dup(0, "help␣")` returns `""`. Fill in the body of `dup` below. You *may* call other C library functions.

```
char *dup(int n, char *s) {
```

16. (10 pts) `itohex(n)` fills a dynamically allocated, null-terminated string with the hexadecimal representation of *all* 32 bits of `n` and returns that string. For example, `itohex(10)` returns `0000000A`. Fill in the body of `itohex` below. You *may* call other functions.
5 pt. Bonus: Make your function work even when ints are not 32 bits long.

```
char *itohex(int n) {
```