This exam has 5 questions, some with several parts. You have 50 minutes, so budget your time. Do all of your work on these pages and give the answer in the space provided. Assume the arm\lab/Linux/C/gcc217 environment unless otherwise stated. This is a closed-book, closed-note exam, and “cheat sheets” are not allowed. Please place all items that you will not need out of view in your bag or under your working space at this time. Electronic devices such as cell phones, PDAs, laptops, MP3 players, iPods, etc. may not be used during this exam.

Name: ___________________________ NetID: ___________________________

Precept (circle one): 1: MW 1:30 X. Li 2: MW 3:30 X. Li 3: MW 7:30 A. Mizrahi
4: TTh 12:30 D. Gabai 5: TTh 1:30 D. Gabai 6: TTh 3:30 J. Zhang

This examination is administered under the Princeton University Honor Code. Students should sit one seat apart from each other and refrain from talking to other students during the exam. All suspected violations of the Honor Code must be reported to honor@princeton.edu.

Write out and sign the Honor Code pledge before turning in the test:

“I pledge my honor that I have not violated the Honor Code during this examination.”

Pledge: ___________________________

Signature: ___________________________

<table>
<thead>
<tr>
<th>Question # and Theme</th>
<th>Available points</th>
<th>Points earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 You get the forms, I'll prepare them...</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1 What would you say you do here?</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>2 I can't describe it. It was just constant.</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3 Would you mind giving me some pointers here?</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4 This is a completely different dynamic.</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>70</td>
<td></td>
</tr>
</tbody>
</table>
0. You get the forms, I'll prepare them... (1 point)

Fill out the identifying information on the front sheet correctly and in its entirety: name, NetID, section. Write out the honor pledge. When you have completed the exam, sign the honor pledge.

1. What would you say you do here? (28 points)

(1a – 5 points) What is printed to stdout by each of the following printf calls? You should assume that these are within the main function of a file that begins with #include <stdio.h>. Circle your final answer for each call.

```c
printf("%d", 0x21 + 7 + 0217);

printf("%d", ~1);

printf("%d", ~1);

printf("%d", ~(255 < 8));

printf("%d", 2 ^ -2);
```

(1b – 10 points) Compare and contrast the given literals by filling in the table below. For type give the appropriate C type (e.g. double). For value, if the literal’s value is 0, write ZERO; if the literal’s value is strictly between 0 and 128, write SMALL; if it is at least 128, write BIG.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>‘0’</th>
<th>‘\0’</th>
<th>“0”</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(1c – 5 points) Consider the following program:

```c
#include <stdio.h>
enum mottoWords {
    IN, THE,
    SERVICE = 1746,
    OF = 1896,
    HUMANITY
};
int main(void) {
    printf("%d %d %d %d %d\n", IN, THE, SERVICE, OF, HUMANITY);
    return 0;
}
```

What values are printed to standard output? You can write GARBAGE to represent an uninitialized value that is printed. Answer in the space below the code box.

(1d – 8 points) Consider the following program:

```c
#include <stdio.h>
int main(void) {
    char c;
    scanf("%c", &c);
    switch(c) {
    case 'a':
        printf("do ");
        break;
    case 'b':
        printf("re ");
        break;
    case 'c':
        printf("mi ");
        break;
    case 'd':
        printf("fa ");
        break;
    default:
        printf("so ");
        break;
    }
    printf("\n");
    return 0;
}
```

Recall that default is triggered if the switch expression matches none of the case options. What is printed to standard output for each of the five inputs below? Answer beside each corresponding input.

a
b
c
d
e
2. I can't describe it. It was just constant. (8 points)

Consider the following program, which has 8 lines for which gcc217 will emit a warning for discarding a const qualifier. Indicate any 4 of these questionable lines by circling the corresponding line number in the list to the right of the program box. (Do not circle more than 4.)

```c
1 int* ii(int* bar) {
2    return bar;
3 }
4
5 int* ici(const int* bar) {
6    return bar;
7 }
8
9 const int* cii(int* bar) {
10    return bar;
11 }
12
13 const int* cici(const int* bar) {
14    return bar;
15 }
16
17 int main(void) {
18    int i;
19    int* pi;
20    const int* cpi;
21    int* r;
22    const int* c;
23    i = 5;
24    pi = &i;
25    cpi = &i;
26    r = ii(pi);
27    r = ii(cpi);
28    c = ii(pi);
29    c = ii(cpi);
30    r = ici(pi);
31    r = ici(cpi);
32    c = ici(pi);
33    c = ici(cpi);
34    r = cii(pi);
35    r = cii(cpi);
36    c = cii(pi);
37    c = cii(cpi);
38    r = cici(pi);
39    r = cici(cpi);
40    c = cici(pi);
41    c = cici(cpi);
42    r = cici(pi);
43    r = cici(cpi);
44    c = cici(pi);
45    c = cici(cpi);
46
47    return 0;
48 }
```
3. Would you mind giving me some pointers here? (15 points)

(3a – 5 points) Here is an excerpt of Assignment 3’s interface:

```c
/* SymTable_contains returns 1 (TRUE) if oSymTable contains a binding whose key is pcKey, and 0 (FALSE) otherwise. */
int SymTable_contains(SymTable_T oSymTable, const char *pcKey);

/* SymTable_get returns the value of the binding within oSymTable whose key is pcKey, or NULL if there is no binding with that key. */
void* SymTable_get(SymTable_T oSymTable, const char *pcKey);
```

Is it correct (setting aside the lack of assert validation) to implement `SymTable_contains` as:

```c
t
```

If it is correct, then write “YES” and identify (using the terminology from our *programming in the large* lectures or in your own words) the software engineering principle this represents. If it is not correct, then write “NO” and identify a bug that could result from this implementation.

(3b – 5 points) Consider the following program, which results in a compilation error:

```c
#include <stdio.h>
int main(void) {
    int i = 'a';
    int* pi = &i;
    void* pv = &i;
    printf("%d\n", *pi);
    printf("%d\n", *pv);
    return 0;
}
```

Which line has the compilation error, and what is wrong with that line?

Rewrite the line substantially unchanged except for a fix that will satisfy the compiler:
(3c – 5 points) A fellow COS 217 student has learned about pointers and now wants to use them everywhere! Your friend wrote the following program to read two numbers from stdin and print their sum to stdout:

```c
#include <stdio.h>
int* getInt()
{
    int x;
    scanf("%d", &x);
    return &x;
}
int main(void)
{
    int* p = getInt();
    int* q = getInt();
    printf("%d + %d = %d\n", *p, *q, *p + *q);
    return 0;
}
```

Your friend tried two favorite numbers, 76 and 141, but found that the program didn’t work correctly. Worse, adding an fprintf that prints to stderr on line 12 resulted in the program printing a completely different (but still wrong) answer!

What was the most likely incorrect output from the program (with inputs 76 and 141) before the fprintf was added?

What is the bug in the program?
(Hint: think about why adding the call to fprintf could change the output from the program.)
4. This is a completely different dynamic. (18 points)

(4a – 6 points) Consider the following program to build an arbitrarily long string from repeated reads from standard input. You may assume that each string provided on stdin will fit in buf, that memory allocation never returns NULL, and that size_t will always be sufficient to store the capacity of str.

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

enum { LIMIT = 100, K = 500 };

int main(void) {
    char buf[LIMIT];
    size_t length = 0;
    size_t capacity = 0;
    char* str = calloc(capacity+1,sizeof(char));

    while(scanf("%s", buf) == 1) {
        length += strlen(buf);
        if(length >= capacity) {
            capacity = length + (long) K;
            realloc(str, capacity+1);
        }
        strcat(str,buf);
    }

    printf("%lu: %s\n", length, str);
    free(str);
    return 0;
}
```

This program passes all our simple tests, passes all our boundary tests, passes all our statement tests, and even passes all our path tests. But it fails a stress test!

In the space below, identify the bug and describe the precise condition that will result in failure. Then, add a simple correction to the program printed above that will eliminate the bug so as to produce a working program under all conditions.
(4b – 12 points) In Assignment 2, you created two implementations of a string library that mirrored the string.h interface. As a reminder, here were the declarations from your str.h:

```c
size_t Str_getLength(const char* pcSrc);
char* Str_copy(char* pcDest, const char* pcSrc);
char* Str_concat(char* pcDest, const char* pcSrc);
int Str_compare(const char* pcFirst, const char* pcSecond);
char* Str_search(const char* pcHaystack, const char* pcNeedle);
char* strdup(const char* pcSrc);
```

The standard library also contains a function strdup, with the following signature:

```c
char* strdup(const char* pcSrc);
```

This function does the exact same thing as strcpy, except that instead of copying into a space pointed to by a parameter (for which the client must have allocated the space), strdup allocates memory dynamically to store the copy of the string.

Expand your string library by implementing Str_duplicate as the equivalent of strdup. Above your implementation, write a function comment that accords to the COS 217 standards for establishing a contract with clients who will call your function.

Your function should match the general requirements for your Assignment 2 implementation – you may call any of the other functions from your library, but nothing from string.h; your function must not be grossly inefficient: it must not traverse a (potentially long) string more times than necessary; etc. We have given the pointer version of strdup’s signature above, but you may choose to use either pointer or array syntax in your implementation.
(This page intentionally left blank. You may use it for scratch work or to complete a problem for which you found insufficient space earlier in the exam. In the latter case, please clearly label which problem you are continuing.)
Because some of you will have been thinking about this: 
*Shawshank Redemption, Office Space, Interstellar, There’s Something About Mary, Gone Girl.*