COS 217, Spring 2019
Midterm Exam

This exam consists of 4 questions, and you have 50 minutes – budget your time wisely. Do all of your work on these pages (using the back for scratch space), and give the answer in the space provided. Assume the ArmLab/Linux/C/gcc217 environment unless otherwise stated. This is a closed-book, closed-note exam, and “cheat sheets” are not allowed. Please place 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 Bob Dondero  4: TTh 1:30 Xiaoyan Li  
2: MW 3:30 Bob Dondero  5: TTh 3:30 James Heppenstall  
3: TTh 12:30 Seo Young Kyung  6: TTh 7:30 Josh 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 and Signature:

<table>
<thead>
<tr>
<th>Question</th>
<th>Total points</th>
<th>Points earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Arithmetic</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>2 Expressions</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>3 String Insert</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>4 Linked List</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. Abbreviated Arithmetic

(a) Complete the following subtraction problem on finite binary integers limited to 4 bits. (8 pts)

\[
\begin{array}{cccc}
1 & 0 & 1 & 0 \\
- & 0 & 1 & 0 \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 0 & 1 \\
\end{array}
\]

(b) What is the interpretation (in decimal) of the subtraction problem (operands and result) in (a), assuming the numbers are unsigned integers? Did (unsigned) overflow occur? (8 pts)

\[10 - 5 = 5. \text{ No unsigned overflow.}\]

(c) What is the interpretation (in decimal) of the subtraction problem (operands and result) in (a), assuming the numbers are two’s complement signed integers? Did (signed) overflow occur? (8 pts)

\[(-6) - 5 = 5. \text{ Signed overflow.}\]

(d) What is the largest number (in binary) with which 0101 could be replaced in the subtraction above that would not lead to unsigned overflow? (5 pts)

\[1010\]

(The subtraction would result in 0000, which is the smallest unsigned number.)

(e) What is the largest number (in binary) with which 0101 could be replaced in the subtraction above that would not lead to signed overflow? (5 pts)

\[0010\]

(The subtraction would result in 1000, which is the most-negative two’s complement number.)
2. Entertaining Expressions

Assume that each of the following expressions is evaluated with these variables defined:

```
unsigned i = 0, j = 1;
```

Write the answer to which each expression evaluates, and the values of \( i \) and \( j \) after evaluation. If the expression results in an error, write “error” and leave the other two columns blank. Assume that the expressions are evaluated independently: changes to \( i \) and/or \( j \) do not carry over. Give all answers in decimal. The C operator precedence table is provided below. (3 pts per expression)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result (or “error”):</th>
<th>New ( i ):</th>
<th>New ( j ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) ++i &amp; ++j</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(b) ++i &amp;&amp; ++j</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(c) i += j</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(d) i++ = j</td>
<td>error (i++ is not a variable and can’t be assigned to)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e) i++ == j</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(f) (i &lt;&lt; j) + (j &gt;&gt; i)</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(g) j &lt;&lt;= i+1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

C operators, in order of precedence, with their associativity

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ] () (function call) . -&gt; ++ -- (postfix)</td>
<td>Left-to-right</td>
<td>1</td>
</tr>
<tr>
<td>++ -- (prefix) &amp; (address-of) * (pointer dereference) + - (unary plus and minus) ~ sizeof</td>
<td>Right-to-left</td>
<td>2</td>
</tr>
<tr>
<td>() (cast)</td>
<td>Right-to-left</td>
<td>3</td>
</tr>
<tr>
<td>* / % (multiplication, division, remainder) + - (addition, subtraction)</td>
<td>Left-to-right</td>
<td>4</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>Left-to-right</td>
<td>5</td>
</tr>
<tr>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>Left-to-right</td>
<td>6</td>
</tr>
<tr>
<td>== !=</td>
<td>Left-to-right</td>
<td>7</td>
</tr>
<tr>
<td>&amp; (bitwise and) ^</td>
<td>Left-to-right</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Left-to-right</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>= /= %= += -= &lt;&lt;= &gt;&gt;= &amp;= ^=</td>
</tr>
<tr>
<td>,</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Incorrect Insertion

We are writing code that will insert a character into a string:

```c
/* Returns a new string consisting of string s with character c
inserted at position n. The caller owns the memory in which the new
string resides, and is responsible for freeing it. Returns NULL if
memory couldn’t be allocated, or if n is more than the length of s. */

char *strinsert(const char *s, char c, size_t n)
{
    /* Allocate space for a string with one more character */
    char *new_s = (char *) malloc(strlen(s) + 1);
    if (new_s == NULL)
        return NULL;

    /* Check if n is plausible */
    if (n > strlen(s))
        return NULL;

    /* Copy first part of s, then c, then rest of s, then null */
    memcpy(new_s, s, n);
    new_s[n] = c;
    memcpy(new_s + n + 1, s + n, strlen(s) - n);
    new_s[strlen(s) + 1] = '\0';

    return new_s;
}
```

Here is documentation for the standard library functions used above:

```c
void *malloc(size_t size);
    Allocates size bytes and returns a pointer to the allocated memory.
    Returns NULL on error.

size_t strlen(const char *s);
    Returns the length of the string pointed to by s, excluding the terminating null byte.

void *memcpy(void *dest, const void *src, size_t n);
    Copies n bytes from memory area src to memory area dest.
    Returns a pointer to the destination dest.
```
3. Incorrect Insertion (cont.)

(a) Unfortunately, the code above has two rather serious memory-management bugs. The first bug manifests itself if n is too large. *What is the bug? Describe how to rearrange the code to fix it.* (10 pts)

Memory for new_s is allocated but not freed by strinsert. Moreover, because strinsert returns NULL, there is no opportunity for the caller to free this memory.

To fix the bug, the second “if” statement must be moved before the malloc. This results in:

```c
char *new_s;

/* Check if n is plausible */
if (n > strlen(s))
    return NULL;

new_s = (char *) malloc(strlen(s) + 1);
if (new_s == NULL)
    return NULL;
```

Alternatively, “free(new_s);” could be added before returning after the second if statement.

(b) The second bug manifests itself if n is within range. *What is the bug? Rewrite one line of code so that it is fixed.* (10 pts)

Because strlen does not include the terminating null, s itself occupies strlen(s)+1 bytes. To add a character to it, we need one more byte.

The fixed code is:

```c
new_s = (char *) malloc(strlen(s) + 2);
```

(c) With the above two fixes, the code is correct, but it turns out that it’s inefficient. *How many times does it access each character* in the input string? (5 pts)

5 times: once for each of 4 calls to strlen, and once for the calls to memcpy.
4. Last in the List [King, p. 454]

Given the following definition:

```c
struct node {
    int key;
    struct node *next;
};
```

Write the following function: (20 pts)

```
/* Given a pointer to the first node in a linked list, returns a pointer to the last node whose key equals n. Returns NULL if n does not appear in the list. */

struct node *find_last(struct node *first, int n) {
    struct node *p, *match = NULL;
    for (p = first; p != NULL; p = p->next) {
        if (p->key == n) {
            match = p;
        }
    }
    return match;
}
```