CS 5 Gold Midterm – Practice!
4 questions – 100 points

Exam Outline

**Question #1**: Recursion == 25 points
**Question #2**: Circuits == 25 points
**Question #3**: Hmmm… == 25 points
**Question #4**: Loops et al. == 25 points

**Optional Extra Credit**: up to +4.2 points

You may use any python functions or capabilities that have been presented thus far in the course (or used/written in labs or hwks). Here are some of the functions you are welcome to use without rewriting:

- `removeOne(e,L)`
- `removeAll(e,L)`
- `abs(x)`
- `removeUpto(e,L)`
- `count(e,L)`
- `min(L)`
- `ind(e,L)`
- `frontNum(L)`
- `max(L)`
- `binToNum(binstr)`
- `numToBin(n)`
- `sort(L)`

and there are certainly others….

Some problems, however, require recursion, while others involve the use of loops. List comprehensions may be used either way.

If you’re unsure of how to solve a problem with code, partial credit is available for an English description of a solution and/or explaining why/how you feel unsure. Some questions ask specifically for an English response (with no code). Other notes:

- All of Hmmm's instructions appear on the final page: feel free to tear it off
- You do **not** need to write docstrings for any Python functions.
- You do **not** need to hand in your page of notes – perhaps save it for the final, which allows for two such pages…
Question 1.

(5 points) **Part A.** You’re the first hire at a new start-up, PalindromeCity, or PalCity. PalCity is revolutionizing the measurement of “string reversibility.” At PalCity, a string is *partially reversible* if the first character of the string equals the final character of the string.

They are eager for you to implement this idea in Python, in a function named `checkends(s)`. `checkends(s)` takes a string `s` as input and returns a Boolean as output.

Specifically, if `s` is the empty string, `checkends(s)` should return `True`. If `s` is a string with only one character, it should also return `True`, since in that case the initial character is *necessarily* the final character! Also, it should return `True` if the initial character of `s` is equal to the final character of `s`.

On the other hand, if the string `s` has two or more characters, but the initial character is *different* from the final character, then `checkends(s)` should return `False`.

Here are four examples of input and output for `checkends`:

```
In : checkends('')
Out: True

In : checkends('e')
Out: True

In : checkends('racecars')
Out: False

In : checkends('racecar')
Out: True
```

Below, implement this `checkends` function (no docstring needed):

```python
def checkends( s ):
(10 points) **Part B.** PalCity’s core innovation is `palScore`, their measurement of the “reversibility” or “palindromicity” of a string. Their (not-yet-written) function `palScore(s)` will take in a string `s` and return an integer. The way it should work is that, if the input string `s` is empty, its `palScore` is 0. If the input string `s` has one character, then `palScore` should return 1.

For larger strings, if a string’s first character is the same as to its final character, it gets a `palScore` of 1, plus the `palScore` of what’s left *after getting rid of its first and final characters*. On the other hand, if a string’s first and final characters are different, then those first and final characters are dropped, and `palScore` is computed on everything that remains.

To keep to their software-design-and-reuse guidelines, PalindromeCity has asked you to use the `checkends` function from part A. In addition, they ask that you write this `palScore` function *without using loops*. Recursion and/or list comprehensions are OK. (Even if you didn’t write `checkends` in part A, you can and should use it in your `palScore` function.)

Here are four input/output examples of `palScore`:

```
In  : palScore('')
Out : 0

In  : palScore('e')
Out : 1

In  : palScore('fluff')
Out : 2

In  : palScore('racecar')
Out : 4
```

Using `checkends`, implement this `palScore` function below *(no docstring is needed)*:

```python
def palScore(s):
```
(10 points) **Part C.** PalCity’s business plan is based on helping its clients *maximize* the `palScore` of their prose, their text, or any other strings. As a result, PalCity has asked for a function `bestPal(L)`, which takes in a list `L`. `L` will *always* be a list of one or more strings. From there, `bestPal(L)` should return the string in `L` with the highest `palScore`.

Here are two input/output examples of `bestPal`:

```
In : bestPal(["", 'e', 'fluff', 'racecar'])
Out: 'racecar'

In : bestPal(['e', 'fluff', 'racecars', ''])
Out: 'fluff'
```

Again, in keeping with the software style of PalCity’s rapidly-expanding codebase, you’re asked to write this function *without using loops*. Both recursion and/or list comprehensions are OK. You *should* use the `palScore` function from part B (even if you didn’t write it there).

```python
def bestPal( L ):
```
**Question #2** (15 points) After their success with string-scoring, PalCity decides to expand into the integer-scoring marketplace. Having consulting with their extensive userbase, PaCity has concluded that an integer should be a “Pal” if it is a divisor of 42. Anticipating huge market demand for their \textsc{isPal} technology, they seek to implement it in hardware, rather than software – and you’ve been tasked to design a three-bit circuit that prototypes \textsc{isPal}.

Specifically, they’ve asked you to complete a three-bit truth table for the \textsc{isPal} function (below). The input bits are named \textit{x y} and \textit{z}, from left-to-right. When complete, the truth table should include all \textit{eight} combinations of three-bit values, running from 000 (which is zero) up to 111 (which is seven).

The output, named \textsc{isPal}, should be a 1 if the input bits \textit{xyz} form a binary number whose value is a divisor of 42: the divisors of 42 are 1, 2, 3, 6, and 7. For example, 010 forms the binary representation of 2, which is a divisor of 42, so the output is 1. On the other hand, the output should be a 0 if the input bits represent a binary number that does not divide 42: the non-divisors are 0, 4, and 5.

The first three lines of the truth table (for 0, 1, and 2) are already filled out.

After completing the truth table, sketch a circuit that directly implements, via minterm-expansion, the \textsc{isPal} function - using only AND, OR, and NOT gates. ANDs and ORs may have as many inputs as you like.

At right is a block-diagram of \textsc{isPal}. Below, draw the circuit itself.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>x y z</td>
<td>\textsc{isPal}</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1</td>
</tr>
</tbody>
</table>

[you’ll need 8 input lines total]

(10 points) Your PalCity colleagues point out that, in binary, doubling a value is the same as shifting it to the left by one bit. With this in mind, they’ve designed the circuit below, \textsc{twoPal}, which uses \textsc{isPal}. They want \textsc{twoPal} to output a 1 if the four input bits \textit{xyzw} form a binary number that is \textit{two times a divisor of 42}. But, if \textit{xyzw} form a binary number that is \textit{not} two-times a divisor of 42, then \textsc{twoPal} should output a 0.

Something is wrong with \textsc{twoPal}! In a phrase or sentence, \textit{what’s wrong}? How could \textsc{twoPal} be fixed to behave as wanted? Feel free to change the diagram, too.
**Question #3**  (25 points) Below is a Hmmm assembly-language program that is *supposed* to input five integers and then write out *how far away their sum is from 42*.

That is, the program should write out the value of  (42 minus the *sum* of those five integers).

For example, if the input values were 10, 5, 20, 5, 0, then the program should stop taking inputs and write 2, because those five inputs sum to 40, and 42 minus 40 yields 2.

As a second example, if the input values were 25, 0, 5, 1, 1, then the program should stop taking inputs and write 10, because those five inputs sum to 32, and 42-32 is 10.

However, the program below *does not work* correctly. That is, it does not work as described above.

Your first task is to indicate what the program, as currently written, writes as an output when given the inputs of 10, 5, 20, 5, and 0.

Then, by indicating changes *only in lines 4-7*, fix this program so that it works as intended. If you wish, you may squeeze new line(s) into that span, but new lines are not necessary. Don’t bother to renumber lines. Also, feel free to refer to the complete Hmmm reference on the last exam page.

There are no errors – and no changes should be made – on the lines with comments.

```
00 setn r4 42       # r4 holds 42 for now
01 setn r3 0        # r3 will accumulate inputs
02 setn r2 5        # r2 holds 5 for now
03 read r1          # r1 is the current input
04 add r3 r1 r1     # r1 is the current input
05 addn r2 -1       # r2 holds 5 for now
06 jeqzn r1 8       # subtract r4 = r4 - r3
07 jumpn 2
08 sub r4 r4 r3     # print what's now in r4
09 write r4         # stop.
10 halt              # stop.
```

What does the program *currently* output when given inputs of 10, 5, 20, 5, and 0? ___________
Question #4

Three-eyed aliens, it turns out, love Python! In fact, they love it so much, they find themselves writing programs without even worrying about what the programs do! They've written the function below, named `mystery`: your task is to determine what outputs would be returned for several inputs and vice-versa. First, look over the function:

```python
def mystery( L ):
    
    """
    This function inputs a list, L
    L will always contain only numbers!
    """
    result = 0
    for i in range( len(L) ):
        if L[i] == L[i-1]:
            result *= 2
        elif L[i] == i:
            result += 1
        if L[i] == 42:
            result = 42
    return result
```

**Part (A)** What value is returned for the input list \( L = [1,2,3,42] \)

**Part (B)** What value is returned for the input list \( L = [0,1,6,7] \)

**Part (C)** What value is returned for the input list \( L = [0,0,2,2] \)

**Part (D)** What value is returned for the input list \( L = [42,42,5,5] \)
# Table of Hmmm instructions

(feel free to tear off – no need to hand in)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System instructions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>halt</td>
<td>Stop!</td>
<td></td>
</tr>
<tr>
<td>read rX</td>
<td>Place user input in register rX</td>
<td></td>
</tr>
<tr>
<td>write rX</td>
<td>Print contents of register rX</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td>Do nothing</td>
<td></td>
</tr>
</tbody>
</table>

| **Setting register data** | | |
| setn rX N | Set register rX equal to the integer N (-128 to +127) | |
| addn rX N | Add integer N (-128 to 127) to register rX | |
| copy rX rY | Set rX = rY | mov |

| **Arithmetic** | | |
| add rX rY rZ | Set rX = rY + rZ | |
| sub rX rY rZ | Set rX = rY - rZ | |
| neg rX rY | Set rX = -rY | |
| mul rX rY rZ | Set rX = rY * rZ | |
| div rX rY rZ | Set rX = rY / rZ (integer division; no remainder) | |
| mod rX rY rZ | Set rX = rY % rZ (returns the remainder of integer division) | |

| **Jumps!** | | |
| jumpn N | Set program counter to address N | |
| jumppr rX | Set program counter to address in rX | jump |
| jeqsn rX N | If rX == 0, then jump to line N | jeqz |
| jnezn rX N | If rX != 0, then jump to line N | jnez |
| jgtzn rX N | If rX > 0, then jump to line N | jgtz |
| jltzn rX N | If rX < 0, then jump to line N | jltz |
| calln rX N | Copy the next address into rX and then jump to mem. addr. N | call |

| **Interacting with memory (RAM)** | | |
| pushr rX rY | Store contents of register rX onto stack pointed to by reg. rY | |
| popr rX rY | Load contents of register rX from stack pointed to by reg. rY | |
| loadn rX N | Load register rX with the contents of memory address N | |
| storn rX N | Store contents of register rX into memory address N | |
| loadr rX rY | Load register rX with data from the address location held in reg. rY | loadi, load |
| storer rX rY | Store contents of register rX into memory address held in reg. rY | storei, store |