ISU Programming Assessment, Mar 15, 2019

Name: ______________________ CS class______________________

Put all answers in boxes. Nothing you write outside the boxes will be counted. Did you bring an eraser?

1. **Write** a program that gets an integer, `n,` from the user and then prints `n` repetitions of the pattern: some `A`'s then some `B`'s. The first pattern, has two `A`'s and one `B`. Each new pattern has one more `A` and two more `B`'s than the pattern before it. **Example:** if `n=3,` then the program will print `AABAAABBBAAAABBBBB`

```
int main(int argc, char *argv[]) {
```

```
    return 0;
}
```

2. **Get input a character at a time.** Write a program that finds two numbers: the total number of characters in the input and the total number of upper case letters in the range `M`–`Z`. Note `M` and `Z` are included in the range. Your program should print only final value of both numbers.

```
int main(int argc, char *argv[]) {
```

```
    return 0;
}
```

3. Write the function `fallCount` that is passed the address of the first node of the list. The function counts the number of times a number is greater than the next number in the list. It returns this count.

```
  typedef struct NODE {
    int data;
    struct  NODE *next;
  }   node_t;
```

```
int  smallTotal(node_t *curr) {

    }
```

4. A BST is constructed in the usual way using the node definition below. The **cost** of a path from a node to a leaf is the sum of the data in all the nodes in the path (includes the start node and the leaf). **Write** the function

```
        int cheapestPath( bst_node_t *curr)
```

that is passed the address of the head node of the BST. It returns the cost of the cheapest path.

```
    typedef struct  BST_NODE_T   {
     int   data;
     struct   BST_NODE_T   *left,   *right;
    }   bst_node_t;
```

5. Write the function

```
        int Corr00(int n)
```

where `n` is a 32–bit int. Displayed as bits `n` is $b_{31}...b_{16}b_{15}...b_0$. The lower half is $b_{15}...b_0$. Each bit b_i in the lower half corresponds to bit b_{i+16} in the upper half. The function counts and returns the number of times the corresponding bits are both 0's.