CS-251, Fall 2017
Written Homework 1
Due Sunday Jan 28, by 11:59PM

Submission will be done using gradescope (you will scan and upload your written homework). Details of the gradescope submission process will be posted to Piazza.

- **Your writeup must be neat and clear**
- **There are 6 problems, some with multiple parts; clearly label your answers.**

Each Problem will be scored out of 20 points (for a total of 120 points).

PROBLEM 1: The function **has_dups** to the right determines if a given array of n elements has any duplicate elements: if at least one value appears two or more times, **true** is returned (it "has duplicates"); otherwise it returns **false** (all elements are distinct: it does not "have duplicates").

Take a few minutes to understand the logic of the function and why it works.

```
bool has_dups(int a[], int n){
int i, j;

  for(i=0; i<n; i++) {
     for(j=i+1; j<n; j++) {
        if(a[i] == a[j])
           return true;
       }
  }
  return false;
}
```

Your job: write a linked-list version of ***exactly*** the same algorithm. A linked list is a sequence of elements just like an array after all -- i.e., a given linked list either has duplicates or it does not.

Use the struct and function prototype below.

```
struct NODE {
  int val;
  NODE *next;
};

bool has_dups(NODE *lst) {

}
```

You will submit a scanned hardcopy (hand-written or printed). Of course, you are free to try out your solution in a real program.

PROBLEM 2: Below is a (trivial) C function which returns the square of its parameter (a non-negative integer):

```
unsigned int square(unsigned int n) {

    return n*n;
}
```

Your job: write a function which also returns n^2 but with the following constraints:

- You cannot use the multiplication operator ‘*’
- You cannot use the division operator ‘/’
- You cannot have any loops
- You cannot add any additional parameters to the function
- Your function must be self-contained: no helper functions!
- You cannot use any globals
- You cannot use any static variables
- You cannot use any "bit twiddling" operations -- no shifts, etc.

However, …

- You *can* use recursion
- You *can* use the ‘+’ and ‘-’ operators.

You will submit a scanned hardcopy (hand-written or printed) or pdf via gradescope. Of course, you are free to try out your solution in a real program.

Addendum: derivation required!

You must explain the logic of your solution! (Explain how you derived it).

Just giving a correct C++ function is not sufficient and will not receive many points (possibly zero!)

PROBLEM 3: Below is a C++ function which is supposed to take an integer array a[] of length and create a “clone” of a (an array of the same length with the same contents) and return the clone.

This attempt is faulty!!!

```
int clone_array(int a[], int n) {
int b[n];
int i;

  for(i=0; i<n; i++) {
      b[i] = a[i];
  }
  return b;
}
```

3.A: identify and describe the errors in this attempt to the best of your ability. Hint: one of the issues relates to the return type (but this is not the only issue). Describe a scenario in which things might go haywire even if the return type issues is corrected.

3.B: if this was an exam question worth 10 points, how much partial credit would you give if you were the grader?

3.C: Give a correct version!

PROBLEM 4: Consider the C++ function below:

```
void fubar(unsigned int n) {
int i, j;

  for(i=0; i<n; i++){
    cout <<"tick" << endl;
  }
  for(i=0; i<n; i++) {
    for(j=0; j<n; j++) {
        cout <<"tick" << endl;
    }
  }
}
```

4.A: Complete the following table indicating how many "ticks" are printed for various parameters n.

> Unenforceable rule: derive your answers "by hand" -- not simply by writing a program calling the function.

n	**number of ticks printed when fubar(n) is called**
0	
1	
2	
3	
4	

4.B: Derive a closed-form expressing the number of ticks as a function of n -- i.e., complete the following:

> *"For all* $n \geq 0$*, calling fubar(n) results in* ______________ *ticks being printed"*
>
> Give a brief justification of your answer; you do not need a formal proof.

PROBLEM 5: Consider the recursive C function below:

```
void foo(unsigned int n) {

    cout << "tick" << endl;
    if(n > 0) {
        foo(n-1);
        foo(n-1);
    }
}
```

5.A: Complete the following table indicating how many “ticks” are printed for various parameters n.

Unenforceable rule: derive your answers “by hand” -- not simply by writing a program calling the function.

n	**number of ticks printed when foo(n) is called**
0	
1	
2	
3	
4	

5.B: Derive a conjecture expressing the number of ticks as a function of n -- i.e., complete the following:

“Conjecture: for all $n \geq 0$, calling foo(n) results in _____________ ticks being printed”

5.C: Prove your conjecture from part B (hint: Induction!)

PROBLEM 6: In the puzzle game sudoku we have a 9x9 grid which must be populated with integers in {1..9}. In a correct solution each row, column must contain each value in {1..9} exactly once (there are also 9 3x3 sub-grids that must obey the same rule).

We want a function which takes an integer array of length 9 representing a sudoku row and determines if it is "ok" or not according to the rule above; it should return true or false accordingly.

Below is an *attempt* at solving this problem.

```
// array row[] is assumed to be of length at least 9
bool sudoku_row_ok(int row[]) {
int sum=0;
int i;

  for(i=0; i<9; i++) {
    if(row[i] < 1 || row[i] > 9)
        return false;            // out of range
    sum += row[i];
  }
  if(sum == 45)     // notice:  1+2+3+4+5+6+7+8+9 = 45
    return true;
  else
    return false;
}
```

2.A: The above attempt is faulty! Give and briefly explain in your own words a counter-example showing that it is faulty.

2.B: Write a correct version of the function. You may not rearrange the elements in the given array. Your solution just has to be correct -- if it seems inefficient, don't worry about it (at least for the purposes of this homework).