HW1 discussion.

PROBLEM 2: recursive squaring

```
unsigned int square(unsigned int n) {
unsigned int n_minus1_sq;

    if(n==0)
           return  0;

    n_minus1_sq = square(n-1);

    return n_minus1_sq + n+n-1;

}
```

Let's assume this works.
n_minus1_sq == $(n-1)^2$

OUR JOB:
RETURN n^2.

Some algebra:

$(n-1)^2 + x = n^2$

$(n^2 - 2n + 1) + x = n^2$

$x - 2n + 1 = 0$

$x = 2n - 1$.

$n^2 = (n-1)^2 + 2n - 1$

$= (n-1)^2 + n + n - 1$

Known from recursion

PROBLEM 5.

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

Let $t(n)$ be # ticks printed by foo(n).

$$t(0) = 1$$

$$t(n) = 1 + t(n-1) + t(n-1) = 1 + 2t(n-1)$$

FOR $n \geq 1$

n	t(n)	2^n
0	1	1
1	3	2
2	7	4
3	15	8
4	31	16
5	63	32

Claim: $t(n) = 2^{n+1} - 1$ FOR ALL $n \geq 0$

PROOF: BASIS: Show $t(0) = 2^{0+1} - 1$.

Note $t(0) = 1$

$2^{0+1} - 1 = 2^1 - 1 = 2 - 1$

$= 1$

LHS

RHS

I.H. Assume for some $k \geq 0$, that

(a) all

(b) some

$$t(k) = 2^{k+1} - 1.$$

PROVE: $t(k+1) = 2^{(k+1)+1} - 1 = 2^{k+2} - 1.$

PROOF: $t(k+1) = 1 + 2t(k)$ by RR., $k \geq 0$, $k+1 \geq 1$.

$= 1 + 2\left(2^{k+1} - 1\right)$ By I.H.

$= 1 + 2 \times 2^{k+1} - 2$ Alg.

$= (1 - 2) + 2^{k+2}$ Alg/rear. ~

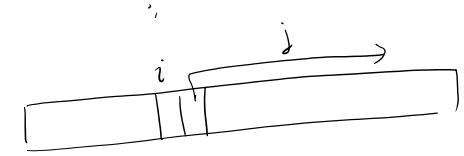
$= 2^{k+2} - 1$ ✓

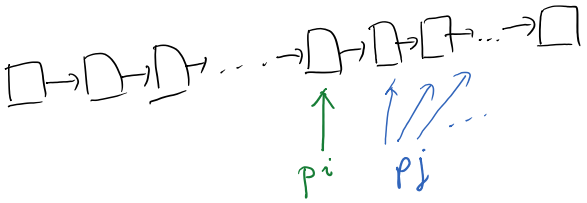
```
bool has_dups(int a[], int n){
int i, j;

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

Algorithm:

for each element x in sequence
  scan elements y after x
    if (y == x) return true






```
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;
}
```

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

bool has_dups(NODE *lst){
NODE *pi, *pj;

 for(pi=lst; pi!=NULL; pi=pi->next) {
    for(pj=pi->next; pj!=NULL; pj=pj->next) {
      if(pi->val == pj->val)
         return true;
   }
 }
 return false;
}
```

Worst-Case Runtime of has_dups

(a) constant

(b) quadratic

e.g. / when all distinct.

(c) linear

Best-case?

(a) constant

(b) linear

(c) quadratic

eg: if a[0] == a[1]

```
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;
}
```

sukoku

Sum == 45 is a NECESSARY CONDITION,

but NOT A ***SUFFICIENT*** CONDITION!

consider {5, 5, 5, 5, 5, 5, 5, 5, 5}

```
// 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;
}
```

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

```
int i;
 for(i=0; i<9; i++) {
   if(row[i] < 1 || row[i] > 9)
       return false;            // out of range
 }
 return !has_dups(row, 9);
}
```

This C program demonstrates how returning pointers to stack-allocated data (local variables) can result in unexpected results

```
typedef struct {
        int a;
        int b;
} PAIR;

// Returns a pointer to an initialized PAIR
PAIR * create_pair(){
     PAIR p;

     p.a = 0;
     p.b = 0;

     return &p;
}

int foo() {
     int x, y, z, p;

     x = 99;
     y = 22;
     z = 33;
     p = 111;
     return 256;
}

// prints a pair
void print_pair(PAIR *pp) {
     printf("(%d, %d)\n", pp->a, pp->b);
}

int main() {
  PAIR * pp;
  // these are used for part II
  PAIR p1;
  PAIR *pptr1, *pptr2;
  int a_before, b_before;
  int a_after, b_after;


  /* PART I */
  pp = create_pair();

  a_before = pp->a;
  b_before = pp->b;
  printf(" JUST SAVED pp->a AND pp->b BEFORE CALL TO foo()\n");
  foo();

  /*** PRINT *pp AFTER CALL TO foo() ****/

  // read a and b fields into local vars and print them
  a_after = pp->a;
  b_after = pp->b;
  printf(" PRESS RETURN TO SEE WHAT HAPPENS");
  fgetc(stdin);
  printf("values before call to foo():\n");
  printf("  pp->a before: %i\n", a_before);
  printf("  pp->b before: %i\n", b_before);

  printf(" PRESS RETURN TO SEE WHAT HAPPENS AFTER foo()");
```

```
$ ./a.out
 JUST SAVED pp->a AND pp->b BEFORE CALL TO foo()
 PRESS RETURN TO SEE WHAT HAPPENS
values before call to foo():
  pp->a before: 0
  pp->b before: 0
 PRESS RETURN TO SEE WHAT HAPPENS AFTER foo()
values AFTER call to foo():
  pp->a AFTER: 99
  pp->b AFTER: 22
 PRESS RETURN TO CONTINUE...
```

```
  fgetc(stdin);
  printf("values AFTER call to foo():\n");
  printf("  pp->a AFTER: %i\n", a_after);
  printf("  pp->b AFTER: %i\n", b_after);
  printf(" PRESS RETURN TO CONTINUE...");
  fgetc(stdin);

  return 0;
}
```

```
1 // n set above
2 int sum = 0;
3 for(i=1; i<=n; i=i+1)
4 sum = 2*sum +  1;
```