University of Illinois at Urbana-Champaign
Second Midterm Exam, ECE 220 Honors Section

Thursday 5 April 2018

Name:
Net ID:

- **Be sure that your exam booklet has NINE pages.**
- **Write your name and Net ID on the first page.**
- **Do not tear the exam apart.**
- **This is a closed book exam. You may <u>not</u> use a calculator.**
- **You are allowed TWO handwritten 8.5×11-inch sheets of notes (both sides).**
- **Absolutely no interaction between students is allowed.**
- **Show all work, and clearly indicate any assumptions that you make.**
- **Challenge problems are marked with ***.**
- **Don't panic, and good luck!**

Problem 1	20 points	_____
Problem 2	15 points	_____
Problem 3	25 points	_____
Problem 4	15 points	_____
Problem 5	25 points	_____
Total	100 points	_____

Problem 1 (20 points): Short Answer Questions

1. **(5 points)** Write a simple recursive function **EST** in order to estimate the size of stack frame consumed by a single iteration of **EST**. The **abs** function returns the absolute value of its argument.

```
#include <stdio.h>
#include <stdlib.h>

int* firstAddr  = NULL;
int* secondAddr = NULL;

void EST (int arg)
{   // write your code here
    int a = arg;
    if (firstAddr != NULL){
        secondAddr = &a;
        return;}
    firstAddr = &a;
    EST (0);
}
int main ()
{
    EST (1);
    printf ("The stack frame size is %d bytes.\n",
            sizeof (*firstAddr) * abs (firstAddr - secondAddr));
    return 0;
}
```

if (1 == arg){
  firstAddr = &arg;
  EST(0);
} else {
  second = &arg;
}

2. **(5 points)** Write the output generated by the code below. ECAC

```
#include <stdint.h>
#include <stdio.h>

static char letters[6] = {'A', 'E', 'F', 'D', 'B', 'C'};

void mystery ()
{
    static int32_t X = 2;
    static int32_t Y;

    Y = 5;

    printf ("%c%c", letters[--X], letters[Y--]);
}

int main ()
{
    mystery ();
    mystery ();
    return 0;
}
```

x=2, Y=5

letters[1] letters[5]
x=1 Y=4
letters[0] letters[5]
x=0 Y=4

3. **(5 points)** Given below is the function prototype for the **`malloc`** function:

```
void* malloc (size_t size);
```

IN 30 WORDS OR FEWER, explain the meaning of the **`size`** parameter and the return value.

Parameter size means the number of bytes needed.
If the memory allocated successfully, return void pointer pointing to the start of allocated memory. If not, return NULL.

4. (5 points)*** What does the function `Func` below return? **ANSWER USING NO MORE THAN TEN WORDS.**

```
int Func (int a, int b) {
    if (b == 0) {
        return a;
    }
    return Func (a ^ b, (a & b) << 1);
}
```

It return at b

a = 5 b = 2
101 010
$a = 111_{(2)} = 7$ b = 0 return 7

a = 5 b = 4
101 100
$a = 001_{(2)} = 1$ $b = 1000_{(2)} = 8$ return 9
$a = 1001_{(2)} = 9$ b = 0

a = 5 b = -1
00101 11111
$a = 11010_{(2)} = -6$ $b = 01010_{(2)} = 10$
$a = 10000_{(2)} = -16$ $b = 10100_{(2)} = -12$
$a = 00100_{(2)} = 4$ $b = 00000_{(2)} = 0$

Problem 2 (15 points): Programming with Functions

A "brute force" approach is acceptable for these problems.

1. **(8 points)** For any two positive integers *M* and *N*, the **greatest common denominator of *M* and *N*** is the **largest number that evenly divides both *M* and *N***. Write the function `gcd` below to return the greatest common denominator of its two parameters, **`M`** and **`N`**. You must use the following function to determine whether a number divides another evenly:

   ```
   int32_t divides_evenly (int32_t divisor, int32_t number);
   ```

   The function **`divides_evenly`** returns 1 if **`divisor`** evenly divides **`number`** and 0 otherwise. You may assume that both **`M`** and **`N`** are between 1 and 2,147,483,647 (the largest 32-bit signed integer), and that **`M ≤ N`**.

   ```
   int32_t gcd (int32_t M, int32_t N) {
   ```

   int32_t answer, index;
   for (index = M; index >= 1; answer--) {
   if (divides_evenly (index , M) && divides_evenly(index,N)) {
   answer = index;
   }
   }
   return ~~answer~~ index;

   ```
   }
   ```

2. **(7 points)***** Two integers are **relatively prime** iff **the only factor they share is the number 1** (the only positive integer that divides both evenly is the number 1). Write the function **`rel_prime`** below as follows. Given parameters **`N`** and **`max`**, **`rel_prime`** must print out every integer between 1 and `max` (in order, including 1 and `max`) that is relatively prime to **`N`**. Your function must call the **`gcd`** function defined in **part 1** above. Each integer printed must be followed by a new line character (**`'\n'`**). Again, you may assume that both parameters are between 1 and 2,147,483,647.

   ```
   void rel_prime (int32_t N, int32_t max) {
   ```

   int32_t index;
   for (index = 1; index <= max; index ++){
   if (gcd (index, N) == 1){
   ~~printf("%d\n", index);~~
   }
   }
   return;

   ```
   }
   ```

Problem 3 (25 points): Pointers and Arrays

1. **(10 points)** Read the program below.

```
#include <stdio.h>

void mystery(int* a, int** b) {
      a = a + 2;
      *b = a;

      int i;
      for (i=0; i<5; i++) {
            a[i] = 2 * ((*b)[i]);
      }
}

int main() {
      int arr1[7] = {7, 6, 5, 4, 3, 2, 1};
      int arr2[5] = {-1, -2, -3, -4, -5};
      int* arr2_ptr = &(arr2[0]);

      mystery(arr1, &arr2_ptr);
      arr1[4] = 220;
      arr2_ptr[0] = 391;

      int i;
      printf("values of arr1: \n");
      for (i=0; i<7; i++) printf("%d ", arr1[i]);
      printf("\nvalues of arr2_ptr: \n");
      for (i=0; i<5; i++) printf("%d ", arr2_ptr[i]);
      printf("\n");

      return 0;
}
```

On the blanks below, write the expected output of the program above.

values of arr1:

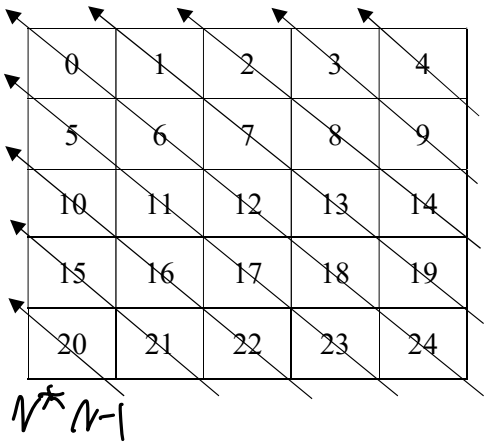
7 6 10 8 220 4 2

values of arr2_ptr:

391 -2 -3 -4 -5

Problem 3, continued:

2. **(15 points)** Implement **`diagonalPrint`**, which takes in an array of N × N size (elements are stored in the numerical order shown in the example below) and prints its values by traversing it in a diagonal order. The traversal for N = 5 is illustrated below:



num = N* N-1
for (index =0 ;

Starting at the bottom-left diagonal (index 20), move along the diagonals until you reach the top-right diagonal (index 4). In this case, for N = 5, the values of the array at indices 20, 21, 15, 22, … 2, 9, 3, 4 would be printed.

20 21 15 22 16 10 17 11 5 18 12 6 23 19 13 7 24 14 8 9 4 3 2

- Separate each output value with a single comma, and use no line feeds/newlines. You do not need to eliminate the trailing comma generated by the last element.
- Use at most EIGHT LINES of code (excluding curly braces). **Code after the first EIGHT LINES will be not graded.** It is possible to finish this problem using four lines of code.
- You must use a loop(s) in your code. Manual repetition will earn **ZERO** credit.
- No comments are needed.

N*(N-1)-1-(N+1)*index2-(N-2-index1)*N

N*(N-1)-1-(N-1)*N

```
void diagonalPrint (int N, int* arr) {
```

int index1, index2;
for (index1=0; index1 <= N-1; index1++){
for (index2=0; index2 <=index1; index2++){
printf("%d,", N*(N-1)+index1-(N+1)*index2);}}
for (index1=N-2; index1>=0; index1--){
for (index2=0; index2 <=index1; index2--){
printf("%d,", N*(N-1)-1-(N+1)*index2-(N-2-index1)*N;}}

```
}
```

Problem 4 (15 points): Error Taxonomy

A programmer wrote the function below to return the index of the first instance of a value **`N`** in an array of length **`len`**. The array is not necessarily sorted. If **`N`** is not found in the array, the function should return -1.

```
      /* Returns the index of the first instance of value "N" in
         the array "list" of length "len" */

1     int32_t bad_find (int32_t N, int32_t * list, int32_t len) {
2         for (i = 0; i < len; i++) {
3             if (list[i] > N) { break; }
4             if (list[i] == N) { return i; }
5         }
6         return -1;
7     }
```

1. **(5 points)** The compiler reports a syntax error. Report the line number on which this error occurs and, **IN FIVE OR FEWER WORDS**, explain the error.

   line number: 2

   error: i not defined before using

2. **(5 points)** The programmer has also made an unfortunate algorithmic error. Give an array and a value for which the algorithm above returns -1 when it should successfully find an index.

   array: 1 2 3 5 4

   value: 4 ✓

3. **(5 points)** The programmer has failed to account for the case where the argument passed for **`list`** is **`NULL`**. This oversight is a specification ambiguity. Explain **IN TEN OR FEWER WORDS** what will happen if **`NULL`** is passed as the array. Assume that the syntax and semantic errors have been fixed. Also assume that **`len > 0`**.

   ~~The function will check an array starting at 0x0000~~

   Dereferences NULL

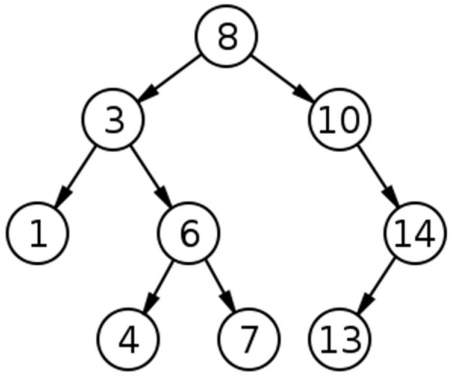
Problem 5 (25 points): Binary Search Trees

Professor Lumetta needs your trees!

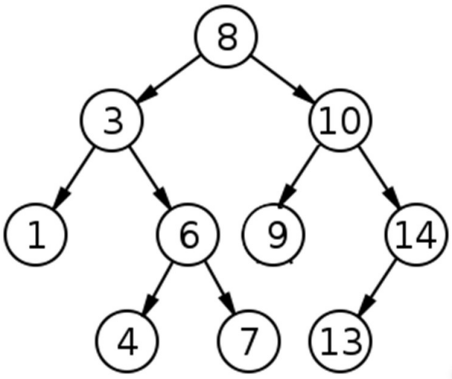
A BST (Binary Search Tree) is a data structure in which nodes are organized in the following way:

1. Each node contains a key (an integer in this problem).
2. All keys in a node's left subtree are less than the node's key.
3. All keys in a node's right subtree are greater than the node's key.
4. Duplicate keys are not allowed.

Given below is an example of a BST:



All elements in the left subtree of any element are less than the element, and all elements in the right subtree of any element are greater than the element.



The same tree after inserting 9 into the tree. Since 9 is greater than 8 but less than 10, it is placed in the right subtree of 8 and the left subtree of 10.

1. **(5 points) USING 30 OR FEWER WORDS**, explain why using a binary search tree to find a node with a particular key is usually faster than using a linked list for the same task.

Because using a binary search tree can avoid some unnecessary check by comparing the target value with keys in each node.

Problem 5, continued:

2. **(20 points)** The structure for a single node appears below. Your task is to write **`BST_insert`**, which inserts a new value into a binary search tree (the function appears below the node structure). Given a pointer to the pointer to the root node, **`rptr`**, and a new key **`value`**, the function must find the appropriate place for the new key, allocate and initialize a new node structure, and insert the new node at the proper place in the tree (following the rules described earlier). The function must return 1 on success, or 0 on failure.

Assume that the BST contains no duplicate keys, and that the new key is not already in the tree. Do NOT assume that the BST has any nodes in it when your function is called (**`*rptr`** may be **`NULL`**).

Only the first 25 lines of code will be graded. Only 15 lines are necessary to solve the problem.

If you need more space, please continue on the back of the previous page.

```
typedef struct BST_Node BST_Node;
struct BST_Node {
    int32_t   key;   // key value for this node
    BST_Node* left;  // left child
    BST_Node* right; // right child
};

int32_t BST_insert (BST_Node** rptr, int32_t value) {
        BST_Node* target = malloc (sizeof(BST_Node));
        if (target == NULL){
                return 0;}
        if (*rptr == NULL){
                        *rptr = target;
                        (*rptr)->key = value;
                        (*rptr)->left = (*rptr)->right = NULL;
                        return 1;
        }
        while(1){
            if (value > (*rptr)->key){
                    if ((*rptr)->right != NULL){
                            *rptr = right;
                    } else {(*rptr)->right = target;
                            break;}
                    continue;
            if (value < (*rptr)->key){
                    if ((*rptr)->left != NULL){
                            *rptr = left
                    } else{*(rptr)->left = target;
                            break;}
                    continue;
        (target)->key = value;
        (target)->left = (*rptr)->right = NULL;
        return 1;
}
```