CSC 3320 : System-Level Programming
Fall 2024: HW3 Key

1. i) (3 points) Ans: `a = 12.3, b = 45, c = 0.6`

   Explanation: The `scanf` function scans for a floating point data first, it first encounters 12.3 and assigns it to `a`. Then the function looks for an integer and takes 45 as an integer (i.e. stops at .6) and assigns to `b`. The remaining part of the number, i.e. .6, is assigned to `c` as it is a float-type variable. The other input part (i.e. 128) will remain unread and will be available for the next `scanf` statement.

   ii) (3 points) Ans: `Output: 4 11 6`

   Explanation: The value of `i` is incremented later and the value of `j` is incremented first and then used in this expression. So, 10 - 6 = 4 is printed for the first statement. In the second `printf` statement, incremented values for both `i` and `j` are printed.

   iii) (3 points) Ans: `Output: 6, 15`

   Explanation: The value of `i` is decremented first and then used in this expression.

2. (5 points) Ans: `Output: one two`

   Explanation: When a case is satisfied (in this case `case 1:`), it continues the following statements without checking further case conditions. So, it will continue executing printing statements. So, to stop after the current case (where it is satisfied), we must use `break` statement in each of the cases as follows:

```
i=1;
switch (i % 3) {
    case 0: printf("zero"); break;
    case 1: printf("one"); break;
    case 2: printf("two"); break;
}
```

3. (5 points) Ans:

   Without conditional operator: `(i > j)- (i < j)`

   With conditional operator: `i>j? 1: (i<j ? -1 : 0)`

4. (5 points) Ans: Yes, the `if` statement is syntactically correct. But the result of comparison of constant 10 with boolean expression (i.e. `n >= 1`) is always true. That is, when `n=0`, the comparison `n >= 1 <= 10` reduces to `false <= 10` and the condition will be `true` and the print operation will happen.

5. (4 points) Ans: 5 4 3 2

   Explanation: The value of `j` is 1 less than `i`. The comma operator in the condition section causes to evaluate both `i > 0` and `j > 0`. But only the condition for `j` i.e. `j > 0` is used for this loop to continue. When `i = 1` and `j = 0`, the loop exits without printing the value of `i`.

6. (5 points) Ans: `Output: 147`

   Explanation: This loop sums up all the integers between 1 and 20 except those integers that are divisible by 3 `i.e. 3, 6, 9, 12, 15, and 18`. So, the result will be 147, because `210 - (3+6+9+12+15+18)=147`.

7. (3 points) Ans: We can rewrite the loop as follows:

```
for (; m > 0; m /= 2 )
    ;
```

   Explanation: `n` is merely a counter for iteration and `n` is not related to `m`. The body changes only the value of `m` and the loop exits only based on the value of `m (i.e. if m > 0)` . So, we can replace the update part `i.e. n++` with `m /= 2`. As `n` does not have any other connection to this loop's function, we can also omit the initialization part `i.e., n = 0`.

8. (4 points) Ans: The error is that the function only checks the first element of the array. If the first element is zero, it returns `true`. But if it is `false`, it returns `false`, which is not correct. Because, other elements of the array might have zeros. We can only return `false` after checking every element of the array as non-zero. But we can return `true` earlier if we get 0 at any point while traversing the array elements. To fix the issue, we have to rewrite the function as follows:

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

    for (i = 0; i < n; i++)
        if (a[i] == 0)
            return true;

    return false;
}
```

9. (5 points) Ans: `Output: i=5, j=10`
   No exchange will happen. Because it was a call by value.

   To swap two variables without using temporary variables, we can modify the function like this:

```
void swap(int a, int b)
{
    a = a + b;
    b = a - b;
    a = a - b;
}
```

10. (5 points) Ans:

    Explanation: It will print the binary representation of a number n using `putchar` function and recursion.

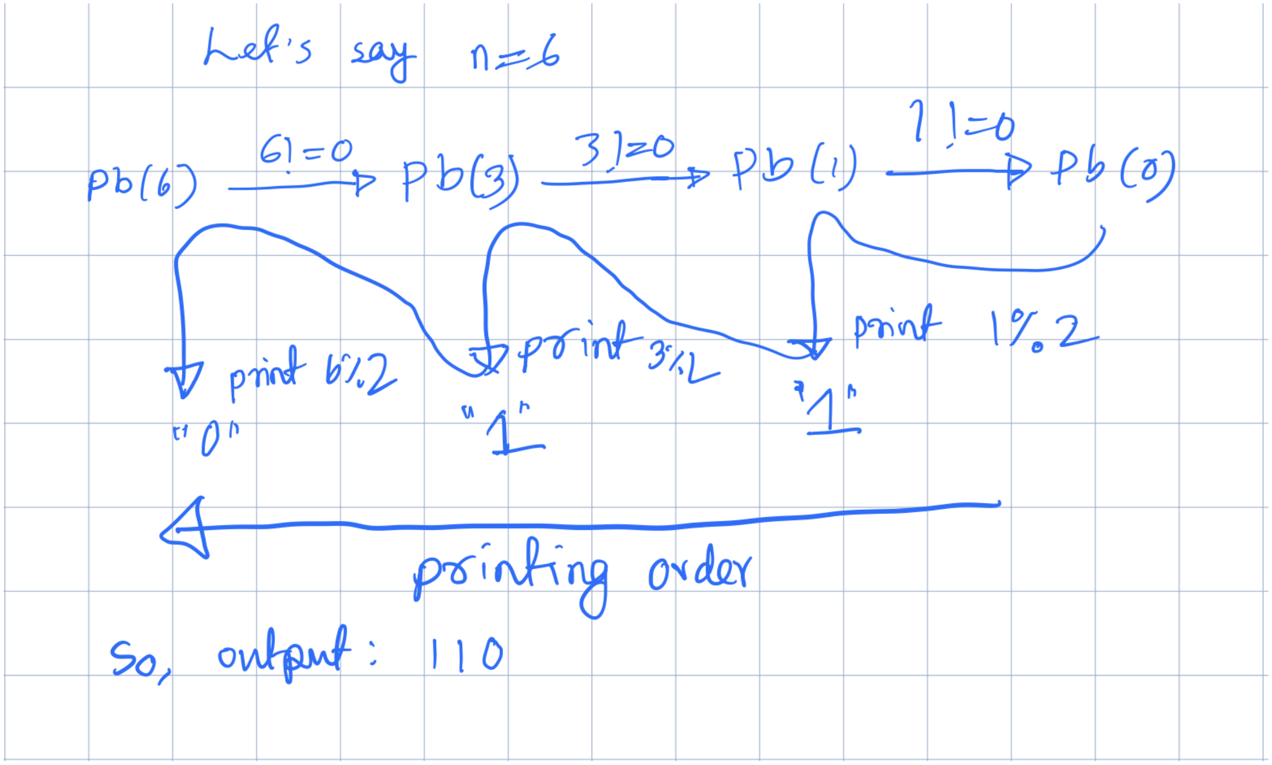

Figure 1: An example tracing with n=6.

11. (2 points (bonus)) i) (3 points (bonus))

Solution: One of the many possible solution is as follows:

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

int fibonacci_fun(int a[], int n)
{   static int count = 0;

    // Memoized approach

    if(a[n] != -1)
      return a[n];

    // count function call
    count += 1;

    if(n==0 || n==1)
      a[n] = n;
    else
      a[n] = fibonacci_fun(a, n-1) + fibonacci_fun(a, n-2);

   // check total counts
   if (n==39)
     printf("\ncurrent n=%d, count=%d\n\n", n, count);

    return a[n] ;
}
int main()
{
   int fib_numbers[40];
   fib_numbers[0] = 0;
   fib_numbers[1] = 1;
   int i=2;
   for(int i=2; i<40; i++)
   {
       fib_numbers[i] = fib_numbers[i-1] + fib_numbers[i-2];
   }

   printf("Fibonacci numbers using iterative approach:\n");

   for(i=0; i<40; i++)
      printf("%d ", fib_numbers[i]);

   printf("\n");

   int rec_fib_numbers[40];

   // set all values as -1 (sentinel value)
   memset(rec_fib_numbers, -1, sizeof(rec_fib_numbers));

   // call function

   fibonacci_fun(rec_fib_numbers, 39);

   printf("Fibonacci numbers using recursive approach:\n");

   for(i=0; i<40; i++)
      printf("%d ", rec_fib_numbers[i]);

   printf("\n");

   return 0;
}
```

Question:	1	2	3	4	5	6	7	8	9	10	11	Total
Points:	9	5	5	5	4	5	3	4	5	5	0	50
Bonus Points:	0	0	0	0	0	0	0	0	0	0	5	5
Score:												