You must write **Java** or **C++** on the first page of your answers!! You may answer the questions using either Java or C++, but you must use the same language for each question on the exam.

1. **(2 pts)** Write a function named `multiples` that takes an integer `n` as an argument. The function outputs a table of the multiples of `n`, formatted into 3 rows of 10 columns. For example, `multiples(7)` would display the following:

   7  14  21  28  35  42  49  56  63  70
   77  84  91  98  105  112  119  126  133  140
   147  154  161  168  175  182  189  196  203  210

   (the values do not need to line up in columns, but they should be separated by at least one space and there must be exactly 10 values on each line).

The next question uses the following class, which implements a linked list:

```java
// Java
class List {
    private class Node {
        int value;
        Node next;
    }
    private Node head;
    public List() {head = null;}
    public void addToFront(int x) {...}
    public void print() {...}
}
```

2. **(2 pts)** Define the public member functions `addToFront(x)` which adds the value `x` to the front of the linked list, and `print()` which displays the contents of the list to the screen (all on one line, values separated by one space).

3. **(2 pts)** Write a recursive function `containsOnly` that takes three parameters: an array, its size, and a target value. The array contains integers, and the target value is an integer. The function should return true if every value in the array is equal to the target value, otherwise it should return false. Note: your function should return true if the size is 0.

   Do not use loops, extra parameters, or global or static variables
4. (4 pts) A Hash Table stores a finite set of values of a given type in an array. It uses a hash function that takes a value of the given type as an argument and computes from it a position in the array. The problem is that multiple values may have the same hash value, but only one can be stored at a given position.

One solution is linear probing. When trying to insert a value x, if another value is already in the position computed by the hash function (say position p), then x is placed at position p+1, unless that location is already full, in which case position p+2 is checked. This process continues until an empty position is found (and if the end of the table is reached, it goes back to position 0).

When trying to determine if a value x is already in the Hash Table (the find operation), you must also use linear probing if a value not equal to x is found in the position computed by the hash function.

The following simple HashTable class stores a set of non-negative integers. The hash function is given, and the capacity can be anything less than or equal to 10000 (the capacity represents the maximum number of elements the table can contain). For simplicity, you may assume the hash table is never full.

```cpp
// C++
class HashTable {
private:
    int array[10000];
    int capacity;
    int hash (int key) {
        return key % capacity;
    }
public:
    HashTable (int c);
    void insert (int x);
    bool find(int x);
};
```

```java
// Java
public class HashTable {
    private int array[];
    private int capacity;
    private int hash (int key) {
        return key % capacity;
    }
    public HashTable (int c) {...}
    public void insert (int x) {...}
    public boolean find(int x) {...}
};
```

Implement the three public functions:
- a 1-argument constructor that creates an empty HashTable, given a capacity. All values in the table should be set to -1 to indicate that they are empty.
- insert(x) to put integer x in the table. Use the hash function and linear probing to find its proper position. Do not add an element that is already in the HashTable.
- find(x) to return true if x has been inserted into the table, false otherwise. Use the hash function and linear probing to determine the result.