MTH 4300, E. Fink

Midterm 1

All code should be written in C++. Unless otherwise specified, you may (and I generally will):

All code should be written in C++. Unless otherwise specified, you may (and I generally will):

- assume that the user of any code you write will be cooperative with the input they supply;
- omit `std::` and the return value of `main()`;
- assume that all necessary libraries have been `#include`d;
- omit `main()` entirely for problems that ask ONLY for function definitions;
- not concern yourself with having optimal solutions (within reason);
- not worry yourself about prompt messages for user input (I sometimes give descriptive prompts to clarify problems);
- not recopy code I have provided, or which you have written in other parts of problems.

Partial credit *will* be given, so do your best if you encounter a difficult question. PLEASE BOX YOUR ANSWER if it is not otherwise clear!

1. Write a program which asks the user to enter in a string; you may assume that this string contains only letters. Your program should then take this string, and print all the odd letters on one line, and all the even letters on another line. ("Odd letters" means the first letter, the third letter, the fifth letter, etc., and "even letters" is described likewise.) *For example*, a sample run might look like:

   ```
   Enter a word: shoulder
   Odd letters: sole
   Even letters: hudr
   ```

   where here, `shoulder` is entered by the user, and all else is produced by the program.

2. What would the following code display when run?

```
char fn(char p[], char &q, char r) {
   p[0] = 'b';
   q = 'c';
   r = 'k';
   cout << "1)" << p[0] << p[1] << q << r << endl;
   return r;
}

int main() {
   char x[2] = {'w', 'o'};
   char y = 'r';
   char z = 'd';
   x[1] = fn(x, y, z);
   cout << "2)" << x[0] << x[1] << y << z << endl;
}
```

3. Write a program which asks the user to enter a positive integer N. Your program should open a file named `nums.txt`. The program should write `1` on the first line of this file; `1 2` on the second line; `1 2 3` on the third line; and so on, leading up to a line containing the first N positive integers, separated by spaces.

4. Consider the following code.

```
int main() {
   int x[2] = {12, 34}, y = 567;
   int *a = x, *b = &y;
   *b = *a;
   b = a + 1;
   *a = *b + 1;
   cout << x << endl;
   cout << x[0] << " " << x[1] << " " << y << " " << &(x[1]) << " " << a << endl;
}
```

If the first `cout` prints out `0x458`, what five values would print (most likely) from the second `cout`?

5. Write a **recursive** function called `all_in`. This function should receive an array of `char`s named `lets`, two `int`s named `beg` and `end`, and a `string` named `word`. The function should return `true` if every element of `lets` with index $\geq$ `beg` and $\leq$ `end` appears in `word`. (The function is only required to work when `beg` and `end` are valid indices for `lets` with `beg` $\leq$ `end`.)

For *example*, if `x` contains {`'a', 'b', 'c', 'd'`}, then `all_in(x, 0, 3, "fxaybz")` returns `false` (since `c` and `d` don't appear in `"fxaybz"`), but `all_in(x, 0, 1, "fxaybz")` returns `true` (since `a` and `b` both appear in `"fxaybz"`).

For full credit, do not use any NESTED LOOPS in your function – use recursion. (You may use a single loop. Hint: think carefully about whether or not you want to loop through the LIST or through the STRING.)

6. a. Ten people are playing a game of Scattergories. The category is “Animals.” Each player has to list as many examples of animals as they can think of.

   Write code which allows the user to store all players’ answers in one jagged 2D array named `all_entries`. Specifically, for each of the ten players, there should be a request to input how many animals the player can think of. Then, a “row” in the array should be created with the given length, and the user should be allowed to enter and store the names of that many animals.

   You should write your code in such a way that, after it was completed, if I were to hypothetically write

   ```
   cout << all_entries[7][1];
   ```

   then the *eighth player’s second animal* would be printed out. **For full credit: allocate EXACTLY as much memory as you will need, and only use language features we’ve discussed in class** (i.e. do NOT use `vectors`).

   b. Now suppose that the game has a second round. To avoid memory leaks, you should deallocate all the memory you’ve dynamically allocated in part a. Write code that accomplishes that.

7. I take the R train to Union Square, and then wait for the next arriving N train. I wish to have a program that will help me predict when this next N train will show up. To this end, I create the following code:

```
int main() {
   int r_train[] = {1045, 1120, 1250, 1325, 1508, 1700}; // CONTENTS MAY CHANGE
   int r_len = 6;        //  = LENGTH OF r_train

   int n_train = {1100, 1150, 1220, 1335, 1400, 1420, 1800}; // CONTENTS MAY CHANGE
   int n_len = 7;         //  = LENGTH OF n_train

   int next_train[] = {-1, -1, -1, -1, -1, -1}; // SAME LENGTH AS r_train
}
```

The first array represents the times when an R train arrives at Union Square; the next array represents the times when the N train arrives. You may assume that both arrays are sorted in increasing order.

The last array, which *will be the same length as `r_train`*, is intended to contain the time of the next arriving N train after a given R train. For example, with the given numbers, `next_train` should be {1100, 1150, 1335, 1335, 1800, 1800}, which are the earliest times in `n_train` after 1045, 1120, 1250, 1325, 1508, and 1700 respectively.

Complete `main()` so that, at its completion, `next_train[i]` contains the time of the next N train arriving after `r_train[i]`, for all `i`. If there is no later N train than `r_train[i]`, the value of `next_train[i]` can remain `-1`.

For full credit, YOUR CODE SHOULD NOT USE ANY NESTED LOOPS. Instead, you should WRITE AND CALL A FUNCTION. (Also, your code should continue to work if the lengths or values of `r_train` and `n_train` were changed. However, the variables `r_len` and `n_len` will be equal to the lengths of the arrays, and `next_train` will be the same length as `r_train`.)