CSc 103 Midterm Exam

Instructions

Read each question carefully. Calculators, books, phones, computers, notes, etc. are not permitted. All you need is something to write with. Scratch paper is provided at the front of the room. You'll have 60 minutes for the exam. **NOTE:** Don't use library functions that trivialize problems. See page 6 for a list of functions that are OK to use (or if you need a reminder on something), and be sure to ask before using anything that isn't on the list.

Name: ______________________________

Problem	Score
1 (10 points)	
2 (10 points)	
3 (10 points)	
4 (extra credit) (5 points)	
Total: 30 points	

1. (10 points) Write a small program (a `main()` function) that reads integers from standard input, and prints the smallest integer that is divisible by 3. For example, if the input is `5 9 2 6 7`, then the answer would be 6, since it is smallest among `9 6`, the only inputs that were divisible by 3.

   You can assume that all the standard `#includes` are there. Just write the `main()` function. (**NOTE:** You don't even need vectors for this.)

```
int main()
{   int x; // input
    int s = INT_MAX; // smallest thing seen so far, which
                                               is div. by 3
    while (cin >> x) {
        if(x %3 ==0 && x < s)
              s = x;
    }
    cout << s;
    return 0;
}
```

2. (10 points) Write a function that evaluates a polynomial f at a given input x. **NOTE:** we will describe the polynomial f by a list of its roots, which assume are all real. That is, the polynomial $f(x)$ has the following form, where r_i are the roots of f:

$$f(x) = (x - r_0)(x - r_1)\dots(x - r_{n-1}).$$

For example, if the input roots are $2, -3, 4$ and $x = 1$, then the output should be $12 = (1 - 2)(1 + 3)(1 - 4)$.

Here is a start, where vector `r` contains the roots:

```
int polyEval(const vector<int>& r, int x) {
    int p = 1;
    for(size_t i = 0; i < r.size(); i++)
        p *= (x - r[i]);
    return p;
}
```

3. (10 points) Write a program that reads integers in the range 0...49 from standard input, and then prints a histogram of how many of each integer were less than 10, how many were between 10 and 19, 20 and 29, and so on. You can ignore any inputs not in the range 0...49. Here is an example: if the program name is histo, then executing

```
$ echo 11 31 43 2 10 3 19 17 23 37 47 42 | ./histo
```

should produce output like this:

```
0: ##
1: ####
2: #
3: ##
4: ###
```

C[0] = # of things from 0... 9
C[1] = # things from 10... 19
⋮

You can assume that all the standard #includes are there. Just write the main() function. (Hint: use an array or a vector.)

```
int x; // input
vector<size_t> C; // C[i] ≡ # values seen in
                  //   [10i, 10(i+1))
C.resize(5,0);
while (cin >> x) {
        C[x/10]++;
 }
// Now just print the result:
for (size_t i=0; i<C.size(); i++) {
    cout << i << ": ";
    for (size_t j=0; j < C[i]; j++)
            cout << "#";
    cout << "\n";
    }
    return 0;
}
```

4. (5 points) Write a function that takes a vector of integers V and a target integer t, and returns a boolean value indicating whether or not there are two distinct elements of V such that $V[i] + V[j] = t$. **Important:** your algorithm should NOT perform a brute force search. If n is the size of V, it must take fewer than cn^2 steps asymptotically, for any value of c.

   Hint: begin by taking for granted a sorting algorithm that works in fewer than cn^2 steps. If you solve the problem using this mystery sorting algorithm, that will get you 4/5 points. Here's a prototype for such a sort function:

```
void sort(vector<int>& V);
```

Say V already sorted. E.g.

Say $t = 18$

$\underline{\cancel{1}}$ $\underline{3}$ 7 11 13 $\underline{15}$

✓

Cheat Sheet / Function Reference

~~Linked List Structure~~

```
struct node {
  int data;
  node* next;
  node(int d=0, node* n=NULL) : data(d),next(n) {}
};
```

Vectors

Use any of the following `vector` functions:

- `push_back(x)` (adds `x` to the end of the vector)
- `pop_back()` (removes whatever is at the end of the vector)
- `size()` (returns the number of elements)
- `resize(n)` and `resize(n,x)` (forces vector to have size `n`; if a second parameter `x` is given, any new values will be set to copies of `x`)
- `clear()` (remove all elements)
- `back()` (this gives the last element; `V.back()` is the same as `V[V.size()-1]`)

Strings

The `string` type actually supports **all** of the above vector functions, but you can also use the following:

- `x + y` (gives a new string that is the concatenation of strings `x` and `y`)
- `x += y` (appends string `y` to the end of `x`)
- `length()` (returns the number of characters; synonym for `size()`)

Constants

If you need the smallest thing that can be stored in an integer, it is `INT_MIN`; the largest is `INT_MAX`.

General

Also remember that you can access the elements of vectors and strings using the square brackets, e.g., `V[i]`, and that making an assignment between vectors or strings does what you expect (left hand side becomes a copy of the right hand side). You can also check vectors and strings for equality with `==` or other comparison operators like `<,>` (doesn't always make sense for vectors, though). And if you know about constructors, you can use those as well, but at best they will just simplify your code a bit (non-default constructors will never be essential). Also, if you need to read all integers from `stdin`, remember you can just do this:

```
int x;
while (cin >> x) {
        /* do something with x... */
}
```