Solutions and Evaluation Criteria for Exam #2 (02/11/2015)

Professors of PRO1

November 27, 2015

1 Turn #1

1.1 X36272: Negatives followed by positives

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> positives;
    bool first = true;

    // read the n values, printing the negatives on-the-go
    // and storing positives to print them later
    for (int i = 0; i < n; ++i) {
      int x;
      cin >> x;
      if (x < 0) {
        // negative elements are immediately printed to cout
        if (first) {
          first = false;
        } else {
          cout << ' ';
        }
        cout << x;
      } else {
        // positive elements are stored into the vector v
        positives.push_back(x);
      }
```

```
    }

    // now we print the positive elements, there
    // is at least 1 negative, so we write always a blank
    for (int i = 0; i < int(positives.size()); ++i)
      cout << ' ' << positives[i];
    cout << endl;
}
```

1.2 X12059: Above a line

```
#include <iostream>
#include <vector>
using namespace std;

struct point2d {
    int x, y;
};

// returns how many points (given in the vector v) are above the
// straight line y = a x + b
int above_line(const vector<point2d>& v, int a, int b) {
  int n = v.size(); int above = 0;
  for (int i = 0; i < n; ++i)
    if (v[i].y > a*v[i].x + b)
      ++above;
  return above;
}

// reads n points from cin (n pairs xi, yi) and returns them
// in a vector<point2d>
vector<point2d> read_points(int n) {
  vector<point2d> v(n);
  for (int i = 0; i < n; ++i)
    cin >> v[i].x >> v[i].y;
  return v;
}

int main() {
  int n;
  while (cin >> n) {
    vector<point2d> v = read_points(n);
    int a, b;
    cin >> a >> b;
    cout << above_line(v, a, b) << " points lie above ";
    cout << "the line y = " << a << "x" ;
```

```
    if (b>0) cout << " + " << b << endl;
    else if (b<0) cout << " - " << -b << endl;
    else cout << endl;
  }
}
```

1.3 X30814: Substitution cipher

The statement of this problem encouraged the students to define and use functions, like `is_letter`, `index`, ... in the solution given below. Failing to do so will translate into a lower pre-score; a solution that does not define any auxiliary function should have a pre-score at most 9. Using `tolower`, `toupper`, ... defined in the STL will be not penalized, even though it is coincidental that they can be used because of the inclusion of `string`; to use them one should actually have the directive `#include <locale>` or `#include <cctype>`.

On the other hand, any reasonable correct solution to this problem follows from the application of a treat-all scheme, where each character from the input that is a letter is used to index the apropiate spot in the string `key` to produce the encoding. Any solution significantly deviating from this "pattern" will be considered seriously erroneous, even if passing all tests, and the penalty factor will be ≤ 0.5.

Summary

- Not defining and using functions $\Longrightarrow$ pre-score ≤ 9
- Wrong application of the treat-all scheme, no indexing or inappropriate indexing the string `key` $\Longrightarrow$ penalty ≤ 0.5

```
#include <iostream>
#include <string>
using namespace std;

// returns the index of the letter c, e.g.
// index('A') = index('a') = 0, index('B') = index('b') = 1, ...
int index(char c) {
  if ('A' <= c and c <= 'Z')
    return int(c) - int('A');
  else // c is a lowercase letter
    return int(c) - int('a');
}

// returns true iff c is a upper- or lowercase letter
bool is_letter(char c) {
  return ('A' <= c and c <= 'Z') or
         ('a' <= c and c <= 'z');
}
```

```
// see the statement of the problem
void encipher(const string& key) {
  char c;
  while (cin >> c)
    if (is_letter(c))
      cout << key[index(c)];
  cout << endl;
}

int main() {
  string key;
  cin >> key;
  encipher(key);
}
```

1.4 X76713: Symmetric difference

An efficient solution to this problem must exploit the fact that the given vectos are in ascending order. Using binary search to determine for each $x \in A$ if it belongs (or not) to B, and similarly, for elements in B may lead to a correct and reasonably efficient solution, although the pre-score will be lower (≤ 9). Failing to take advantadge of the ordering of the vector should get the lowest possible pre-score.

In addition, if the solution consists of a loop that treats the two vectors in parallel with a first loop, but it does not process the rest of A or B (whichever vector retains unprocessed elements) after the first loop ends, it is a serious error and the penalty factor will be ≤ 0.6. The same penalty applies if the solution does not correctly process the case where one of the sets is empty. Using additional memory to store the elements of $A \triangle B$ is unnecessary and it will be considered a minor error.

Summary

- Not exploiting order of the vectors $\Longrightarrow$ pre-score $= 7$
- Not traversing the vectors in parallel ("vector fusion") $\Longrightarrow$ pre-score ≤ 9
- Not processing correctly the case where one set is empty, or not processing correctly the remaining unprocessed elements after the first loop $\Longrightarrow$ penalty factor ≤ 0.6
- Additional useless vector to store $A \triangle B \Longrightarrow$ penalty factor ≤ 0.9

```
#include <iostream>
#include <vector>
using namespace std;
```

```
typedef vector<int> IntSet;

// returns the size of the symmetric difference of the sets A and B
// represented in the two given IntSet's
int size_symm_difference(const IntSet& A, const IntSet& B) {
  int m = A.size(); int n = B.size();
  int i = 0, j = 0, ssd = 0;
  // ssd = size of symm. diff. so far, that is
  // the size of the symm. diff. of A[0..i-1] and B[0..j-1]
  while (i < m and j < n) {
    if (A[i] < B[j]) {
      ++i; ++ssd;
    } else if (A[i] > B[j]) {
      ++j; ++ssd;
    } else { // A[i] == B[j]
      ++i; ++j;
    }
  }
  // at most one of these loops is executed, not both
  while (i < m) {
    ++i; ++ssd;
  }
  while (j < n) {
    ++j; ++ssd;
  }
  return ssd;
}

// reads a set of N integers from cin and returns them in an IntSet
IntSet read_set_integers(int N) {
  IntSet v(N);
  for (int i = 0; i < N; ++i)
    cin >> v[i];
  return v;
}

int main() {
  int m, n;
  while (cin >> m) {
    IntSet A = read_set_integers(m);
    cin >> n;
    IntSet B = read_set_integers(n);
    cout << size_symm_difference(A, B) << endl;
  }
}
```

2 Turn #2

2.1 X62402: Second subsequence followed by first subsequence

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  int n, k;
  cin >> n >> k;
  // read the n elements in a vector
  vector<int> v(n);
  for (int i = 0; i < n; ++i)
     cin >> v[i];

  // output the last k elements
  cout << v[n-k];
  for (int i = n-k+1; i < n; ++i)
     cout << ' ' << v[i];

  // output the first n-k elements
  for (int i = 0; i < n-k; ++i)
     cout << ' ' << v[i];
  cout << endl;
}
```

2.2 X12751: Inside a circle

```
#include <iostream>
#include <vector>
using namespace std;

struct point2d {
    int x, y;
};

// returns true iff the point p=(x,y) is inside the circle
// of radius r entered at the origin
bool point_inside_circle(const point2d& p, int r) {
  return p.x*p.x + p.y*p.y < r*r;
}

// returns how many points (given in the vector v) are inside
// the circle of radius r > 0 centered at the origin
```

```
int inside_circle(const vector<point2d>& v, int r) {
  int n = v.size(); int inside = 0;
  for (int i = 0; i < n; ++i)
    if (point_inside_circle(v[i], r))
      ++inside;
  return inside;
}

// reads n points from cin (n pairs xi, yi) and returns them
// in a vector<point2d>
vector<point2d> read_points(int n) {
  vector<point2d> v(n);
  for (int i = 0; i < n; ++i)
    cin >> v[i].x >> v[i].y;
  return v;
}

int main() {
  int n;
  while (cin >> n) {
    vector<point2d> v = read_points(n);
    int r;
    cin >> r;
    cout << inside_circle(v, r) << " points lie inside ";
    cout << "the circle  x^2 + y^2 = " << r*r << endl;
  }
}
```

2.3 X61261: Morse code

The statement of this problem encouraged the students to define and use functions, like `is_letter`, `index`, ... in the solution given below. Failing to do so will translate into a lower pre-score; a solution that does not define any auxiliary function should have a pre-score at most 9. Using `tolower`, `toupper`, ... defined in the STL will be not penalized, even though it is coincidental that they can be used because of the inclusion of `string`; to use them one should actually have the directive `#include <locale>` or `#include <cctype>`.

On the other hand, any reasonable correct solution to this problem follows from the application of a treat-all scheme, where each character from the input that is a letter is used to index the apropiate spot in the vector `MC` to produce the encoding. Any solution significantly deviating from this "pattern" will be considered seriously erroneous, even if passing all tests, and the penalty factor will be ≤ 0.5.

Summary

- Not defining and using functions $\Longrightarrow$ pre-score ≤ 9

- Wrong application of the treat-all scheme, no indexing or inappropriate indexing the vector `MC` $\Longrightarrow$ penalty ≤ 0.5

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

// fills the MC vector with appropriate values for the Morse code
void fill_Morse_code(vector<string>& MC)  {
  MC = vector<string>(int('Z') - int('A') + 1);
  MC[int('A') - int('A')] = ".-";   MC[int('B') - int('A')] = "-...";
  MC[int('C') - int('A')] = "-.-."; MC[int('D') - int('A')] = "-..";
  MC[int('E') - int('A')] = ".";    MC[int('F') - int('A')] = "..-.";
  MC[int('G') - int('A')] = "--.";  MC[int('H') - int('A')] = "....";
  MC[int('I') - int('A')] = "..";   MC[int('J') - int('A')] = ".---";
  MC[int('K') - int('A')] = "-.-";  MC[int('L') - int('A')] = ".-..";
  MC[int('M') - int('A')] = "--";   MC[int('N') - int('A')] = "-.";
  MC[int('O') - int('A')] = "---";  MC[int('P') - int('A')] = ".--.";
  MC[int('Q') - int('A' )] = "--.-";MC[int('R') - int('A')] = ".-.";
  MC[int('S') - int('A')] = "...";  MC[int('T') - int('A')] = "-";
  MC[int('U') - int('A')] = "..-";  MC[int('V') - int('A')] = "...-";
  MC[int('W') - int('A')] = ".--";  MC[int('X') - int('A')] = "-..-";
  MC[int('Y') - int('A')] = "-.--"; MC[int('Z') - int('A')] = "--..";
}

// returns the index of the letter c, e.g.
// index('A') = index('a') = 0, index('B') = index('b') = 1, ...
int index(char c) {
  if ('A' <= c and c <= 'Z')
    return int(c) - int('A');
  else // c is a lowercase letter
    return int(c) - int('a');
}

// returns true iff c is a upper- or lowercase letter
bool is_letter(char c) {
  return ('A' <= c and c <= 'Z') or
         ('a' <= c and c <= 'z');
}


// see the statement of the problem
void encode_text_in_Morse(const vector<string>& MC) {
  char c;
  bool first = true;
```

```
  while (cin >> c)
    if (is_letter(c)) {
      if (first) {
        first = false;
      } else {
        cout << ' ';
      }
      cout << MC[index(c)];
    }
  cout << endl;
}

int main() {
  vector<string> MC;
  fill_Morse_code(MC);
  encode_text_in_Morse(MC);
}
```

2.4 X49511: Jaccard index

An efficient solution to this problem must exploit the fact that the given vectos are in ascending order. Using binary search to determine for each $x \in A$ if it belongs (or not) to B, and similarly, for elements in B may lead to a correct and reasonably efficient solution, although the pre-score will be lower (≤ 9). Failing to take advantadge of the ordering of the vector should get the lowest possible pre-score.

In addition, if the solution consists of a loop that treats the two vectors in parallel with a first loop, but it does not process the rest of A or B (whichever vector retains unprocessed elements) after the first loop ends, it is a serious error and the penalty factor will be ≤ 0.6. The same penalty applies if the solution does not correctly process the case where one of the sets is empty (the case were both are empty is excluded). Using additional memory to store the elements of $A \cup B$ or $A \cap B$ is unnecessary and it will be considered a minor error.

Similarly, to compute the index we need only to determine the size of the intersection or of the union, since the other can easily be deduced from the other: $|A \cup B| = |A| + |B| - |A \cap B|$. A solution computing directly and explicitly both sizes will be considered a minor error. If these two errors occur then the penalty factor will be ≤ 0.8.

Summary

- Not exploiting order of the vectors $\Longrightarrow$ pre-score $= 7$
- Not traversing the vectors in parallel ("vector fusion") $\Longrightarrow$ pre-score ≤ 9
- Not processing correctly the case where one set is empty ≤ 0.6

- Additional useless vector to store $A \cap B$ or $A \cup B \Longrightarrow$ penalty factor ≤ 0.9
- Computing **both** the size of the intersection and of the union $\Longrightarrow$ penalty factor ≤ 0.9
- No explicit conversion to double to compute the index $\Longrightarrow$ penalty factor ≤ 0.9
- Two minor errors from those mentioned above $\Longrightarrow$ penalty factor ≤ 0.8

```
#include <iostream>
#include <vector>
using namespace std;

typedef vector<int> IntSet;

// returns the size of the intersection of the sets A and B
// represented in the two given IntSet's
int size_intersection(const IntSet& A, const IntSet& B) {
  int m = A.size(); int n = B.size();
  int i = 0, j = 0, sz_int= 0;
  // sz_int = size of intersection so far, that is
  // the size of the intersection of A[0..i-1] and B[0..j-1]
  while (i < m and j < n) {
    if (A[i] < B[j]) {
      ++i;
    } else if (A[i] > B[j]) {
      ++j;
    } else { // A[i] == B[j]
      ++i; ++j; ++sz_int;
    }
  }
  return sz_int;
}

// returns the Jaccard index of the sets A and B, |A U B| > 0
double jaccard_index(const IntSet& A, const IntSet& B) {
  int sz_int = size_intersection(A, B);
  // we do not need to compute explicitly
  // the size of the union, since
  // |A U B | = |A| + |B| - |A intersection B|
  return double(sz_int) / double(A.size()+B.size()-sz_int);
}

// reads a set of N integers from cin and returns them in an IntSet
IntSet read_set_integers(int N) {
  IntSet v(N);
```

```
  for (int i = 0; i < N; ++i)
    cin >> v[i];
  return v;
}

int main() {
  int m, n;
  cout.setf(ios::fixed); cout.precision(3);
  while (cin >> m) {
    IntSet A = read_set_integers(m);
    cin >> n;
    IntSet B = read_set_integers(n);
    cout << jaccard_index(A, B) << endl;
  }
}
```