LUND INSTITUTE OF TECHNOLOGY Department of Computer Science

Solutions, C++ Programming Examination

2013–04–06

1. a) Member in `IntSet`: `set<int> numbers`. Definition of member functions:

```
IntSet::IntSet() {}

bool IntSet::contains(int nbr) const {
    return numbers.find(nbr) != numbers.end();
}

void IntSet::insert(int nbr) {
    numbers.insert(nbr);
}
```

b) Members in `IntSet`, definition of member functions:

```
    struct Element {
        Element(int m, int n) : min(m), max(n) {}
        int min;
        int max;
    };
    list<Element> li;
    static const int MIN = numeric_limits<int>::min();
    static const int MAX = numeric_limits<int>::max();

IntSet::IntSet() {
    li.push_back(Element(MIN, MIN));
    li.push_back(Element(MAX, MAX));
}

bool IntSet::contains(int nbr) const {
    auto it = find_if(li.begin(), li.end(),
            [nbr](const Element& e) { return e.max >= nbr; });
    return it->min <= nbr;
}

void IntSet::insert(int nbr) {
    auto it = find_if(li.begin(), li.end(),
            [nbr](const Element& e) { return e.max >= nbr; });
    auto prev = it;
    --prev;
    if (prev->max == nbr - 1 && it->min == nbr + 1) {
        it->min = prev->min;
        li.erase(prev);
    } else if (it->min == nbr + 1) {
        it->min = nbr;
    } else if (prev->max == nbr - 1) {
        prev->max = nbr;
    } else if (prev->max < nbr && it->min > nbr) {
        li.insert(it, Element(nbr, nbr));
    }
}
```

2.
```
ostream& operator<<(ostream& os, const IntSet &rhs) {
    auto first = rhs.li.begin();
    ++first;
    auto last = rhs.li.end();
    --last;
    for (auto e = first; e != last; ++e) {
        for (int x = e->min; x <= e->max; ++x) {
            os << x << " ";
        }
    }
    return os;
}
```

The function must be a friend of `IntSet`.

3.
```
int main() {
    ifstream in("dict.txt");
    map<string, int> counts; // sorted letters -> number of occurrences
    multimap<string, string> words; // sorted letters -> original word

    string s;
    while (in >> s) {
        string temp = s;
        sort(temp.begin(), temp.end());
        ++counts[temp];
        words.insert(make_pair(temp, s));
    }

    int max_count = -1;
    vector<string> max_strings;
    for (auto c : counts) {
        if (c.second > max_count) {
            max_count = c.second;
            max_strings.clear();
            max_strings.push_back(c.first);
        } else if (c.second == max_count) {
            max_strings.push_back(c.first);
        }
    }

    cout << "The largest anagram groups:" << endl;
    for (auto s : max_strings) {
        auto range = words.equal_range(s);
        for (auto it = range.first; it != range.second; ++it) {
            cout << it->second << " ";
        }
        cout << endl;
    }
}
```

4. a) Operator<< must be defined for `Car` objects:

```
ostream& operator<<(ostream& os, const Car& c) {
    return os << c.getNbr() << " " << c.getOwner()->getName();
}
```

b) Delete the `Person` objects immediately before the end of the function (note that this solution doesn't work if a person owns more than one car):

```
for (auto& c : v) {
    delete c.getOwner();
}
```

c) Calls to sort on license numbers and on owner names:

```
sort(v.begin(), v.end(), [](const Car& c1, const Car& c2) {
    return c1.getNbr() < c2.getNbr(); });

sort(v.begin(), v.end(), [](const Car& c1, const Car& c2) {
    return c1.getOwner()->getName() < c2.getOwner()->getName(); });
```

A note about the solutions: I'm trying to get used to C++11, so I've used C++11 features in many places (`auto`, lambdas, ...). C++98 solutions are naturally still ok.