LUND INSTITUTE OF TECHNOLOGY Department of Computer Science

Solutions, C++ Programming Examination

2015–03–21

1. Code to remove negative numbers from a vector v:

```
v.erase(remove_if(v.begin(), v.end(), [](int x) { return x < 0; }), v.end());
```

Implementation of `remove_if`:

```
template <typename ForwardIterator, typename UnaryPredicate>
ForwardIterator remove_if(ForwardIterator first, ForwardIterator last,
                          UnaryPredicate pred) {
    first = find_if(first, last, pred);
    if (first != last) {
        ForwardIterator it = first;
        while (++it != last) {
            if (!pred(*it)) {
                *first = *it;
                ++first;
            }
        }
    }
    return first;
}
```

2.
```
void make_swearword(string& s) {
    string vowels = "aeiouyåääöAEIOUYÅÄÖ";
    set<char> vowel_set(vowels.begin(), vowels.end());
    string rep = "@$!*#&%";
    default_random_engine e(time(0));
    uniform_int_distribution<unsigned> u(0, rep.length() - 1);

    for_each(s.begin(),s.end(),
             [&](char& c) { if (vowel_set.count(c) == 1) { c = rep[u(e)]; }});
}
```

3.
```
class BulgarianSolitaire {
    friend ostream& operator<<(ostream& os, const BulgarianSolitaire& bs);
public:
    BulgarianSolitaire(int n);
    bool atGoal() const;
    void move();
private:
    vector<int> piles;
};
```

... fortsätter nästa sida

```
BulgarianSolitaire::BulgarianSolitaire(int n) {
    default_random_engine e(time(0));
    uniform_int_distribution<unsigned> u1(1, n);
    piles = vector<int>(u1(e), 1);
    uniform_int_distribution<unsigned> u2(0, piles.size() - 1);
    for (int i = piles.size(); i != n; ++i) {
        piles[u2(e)]++;
    }
}

bool BulgarianSolitaire::atGoal() const {
    vector<int> sequence(piles.size());
    iota(sequence.begin(), sequence.end(), 1);
    return is_permutation(piles.begin(), piles.end(), sequence.begin());
}

void BulgarianSolitaire::move() {
    for_each(piles.begin(), piles.end(), [](int& x) { --x; });
    piles.push_back(piles.size());
    auto end_notempty = remove(piles.begin(), piles.end(), 0);
    piles.erase(end_notempty, piles.end());
}

ostream& operator<<(ostream& os, const BulgarianSolitaire& bs) {
    copy(bs.piles.begin(), bs.piles.end(), ostream_iterator<int>(os, " "));
    return os;
}
```

4.
```
template <typename Container>
class sort_insert_iterator
    : public iterator<output_iterator_tag, void, void, void, void> {
public:
    explicit sort_insert_iterator(Container& x) : cont(&x) {}

    sort_insert_iterator& operator=(const typename Container::value_type& value) {
        auto it = cont->begin();
        while (it != cont->end() && *it < value) {
            ++it;
        }
        cont->insert(it, value);
        return *this;
    }

    sort_insert_iterator& operator*() { return *this; }

    sort_insert_iterator& operator++() { return *this; }
private:
    Container* cont;
};

template <typename Container>
sort_insert_iterator<Container> sort_inserter(Container& c) {
    return sort_insert_iterator<Container>(c);
}
```