LUND INSTITUTE OF TECHNOLOGY Department of Computer Science

Solutions, C++ Programming Examination

2013–03–16

1. a)
```
bool is_odd(int x) {
    return x % 2 != 0;
}

bool is_odd_partitioned(const vector<int>& v) {
    return is_partitioned(v.begin(), v.end(), is_odd);
}
```

   With a lambda function (C++11):

```
bool is_odd_partitioned(const vector<int>& v) {
    return is_partitioned(v.begin(), v.end(), [](int x) { return x % 2 != 0; });
}
```

   b)
```
template <typename InputIt, typename UnaryPredicate>
bool is_partitioned(InputIt first, InputIt last, UnaryPredicate p) {
    while (first != last && p(*first)) {
        ++first;
    }
    while (first != last && !p(*first)) {
        ++first;
    }
    return first == last;
}
```

2. Private members in `Scheduler`:

```
struct CompareTimes {
    bool operator()(Event* e1, Event* e2) const {
        return e1->getTime() > e2->getTime();
    }
};
priority_queue<Event*, vector<Event*>, CompareTimes> q;
```

   `CompareTimes` is the comparison function (the objects should be stored in ascending time order). Member functions:

```
void Scheduler::insertEvent(Event* e) {
    q.push(e);
}

void Scheduler::actionLoop() {
    while (!q.empty()) {
        q.top()->action();
        delete q.top();
        q.pop();
    }
}
```

3. Definition of the iterator class:

```
class WordIterator {
public:
    WordIterator(const String& s, size_t p);
    bool operator!=(const WordIterator& wi) const;
    std::string operator*();
    WordIterator& operator++();
private:
    const String& my_s; // the string that we're iterating over
    size_t pos;         // position in the string
    void skip();        // skip blanks in the string
};
```

Additions to class `String`:

```
    friend class WordIterator;
public:
    typedef WordIterator word_iterator;
    WordIterator wi_begin() const { return WordIterator(*this, 0); }
    WordIterator wi_end() const { return WordIterator(*this, n); }
```

Member functions:

```
WordIterator::WordIterator(const String& s, size_t p) : my_s(s), pos(p) {
    skip();
}

bool WordIterator::operator!=(const WordIterator& wi) const {
    return &wi.my_s != &my_s || wi.pos != pos;
}

std::string WordIterator::operator*() {
    std::string s;
    while (pos < my_s.n && my_s.chars[pos] != ' ') {
        s += my_s.chars[pos];
        ++pos;
    }
    return s;
}

WordIterator& WordIterator::operator++() {
    skip();
    return *this;
}

void WordIterator::skip() {
    while (pos < my_s.n && my_s.chars[pos] == ' ') {
        ++pos;
    }
}
```

4.
```
void reverse_word(string& word) {
    reverse(word.begin(), word.end());
}

void tolower_word(string& word) {
    transform(word.begin(), word.end(), word.begin(), ::tolower);
}

int main() {
    ifstream infile("words.txt");
    vector<string> words((istream_iterator<string>(infile)),
            istream_iterator<string>());

    for_each(words.begin(), words.end(), tolower_word);
    for_each(words.begin(), words.end(), reverse_word);
    sort(words.begin(), words.end());
    vector<string>::iterator unique_end = unique(words.begin(), words.end());
    for_each(words.begin(), unique_end, reverse_word);

    ofstream outfile("backwords.txt");
    copy(words.begin(), unique_end, ostream_iterator<string>(outfile, "\n"));
}
```