Solutions, C++ Programming Examination

2010–03–11

1. a) Dynamic objects are created in `insert`, but they are never deleted. They must be deleted in the destructor:

```
~NameList() {
    for (list_type::iterator it = names.begin(); it != names.end(); ++it) {
        delete *it;
    }
}
```

   b) The parameter `nl` is called by value but the class doesn't have a copy constructor that copies what the pointers point to. Solution: 1) use call by reference `(const NameList& nl)`, or 2) write a copy constructor.

   c) In `printSorted` `*it` is printed, but `*it` is a pointer, not a string. Solution: `cout << **it`.

   d) The class `set` sorts according to the values of the set, and these are pointers. Solution: write a functor that specifies the sorting order, and use it when the set is defined:

```
struct StringPtrLess {
    bool operator()(const std::string* ps1, const std::string* ps2) const {
        return *ps1 < *ps2;
    }
};
typedef std::set<std::string*, StringPtrLess> list_type;
```

   e) The output operator must be overloaded:

```
ostream& operator<<(ostream& os, const NameList& nl) {
    for (NameList::list_type::const_iterator it = nl.names.begin();
            it != nl.names.end(); ++it) {
        os << **it << endl;
    }
    return os;
}
```

   The operator function must be specified as `friend` in `NameList`:

```
friend ostream& operator<<(ostream& os, const NameList& nl);
```

2.
```
void Index::build(const vector<docname>& doclist) {
    for (size_t i = 0; i != doclist.size(); ++i) {
        ifstream in(doclist[i].c_str());
        string word;
        while (in >> word) {
            index[word].insert(doclist[i]);
        }
    }
}
```

```
void Index::write(ostream& out) const {
    for (index_type::const_iterator it = index.begin(); it != index.end(); ++it) {
        out << it->first;
        const docset& ds = it->second;
        for (docset::const_iterator it2 = ds.begin(); it2 != ds.end(); ++it2) {
            out << " " << *it2;
        }
        out << endl;
    }
}

void Index::read(istream& in) {
    string line;
    while (getline(in, line)) {
        istringstream is(line);
        string word;
        is >> word;
        docset ds;
        docname doc;
        while (is >> doc) {
            ds.insert(doc);
        }
        index.insert(make_pair(word, ds));
    }
}

Index::docset Index::search(const vector<string>& wordlist) const {
    docset result;
    for (size_t i = 0; i != wordlist.size(); ++i) {
        index_type::const_iterator it = index.find(wordlist[i]);
        if (it != index.end()) {
            const docset& r = it->second;
            if (i == 0) {
                result = r;
            } else {
                docset tmp;
                set_intersection(r.begin(), r.end(), result.begin(), result.end(),
                        inserter(tmp, tmp.begin()));
                result.swap(tmp);
            }
        } else {
            return docset();
        }
    }
    return result;
}
```

3.
```
int main(int argc, char** argv) {
    if (argc != 2) {
        cerr << "Usage: parse filename" << endl;
        exit(1);
    }
    ifstream in(argv[1]);
    if (! in) {
        cerr << "Cannot open: " << argv[1] << endl;
        exit(1);
    }
    XMLParser parser(in);
    try {
        parser.parse();
        cout << "Wellformed XML" << endl;
    } catch (parse_error) {
        cout << "Syntax error" << endl;
    }
}

void XMLParser::parse() throw(parse_error) {
    stack<Tag> tags;
    int nbr_nodes = 0;
    Tag tag = get_tag();
    while (! in.eof()) {
        if (tag.kind == Tag::open) {
            if (tags.empty()) {
                ++nbr_nodes;
            }
            tags.push(tag);
        } else {
            if (tags.empty()) {
                throw parse_error();
            }
            Tag compare = tags.top();
            if (compare.text != tag.text) {
                throw parse_error();
            }
            tags.pop();
        }
        tag = get_tag();
    }
    if (! tags.empty()) {
        throw parse_error();
    }
    if (nbr_nodes != 1) {
        throw parse_error();
    }
}
```