Solutions, C++ Programming Examination

2013–08–24

1. a) `operator*` must return a reference, `int&`.

   b)
```
template <typename T>
class Ptr {
public:
    Ptr(T* p) : curr(p) {}
    T& operator*() const { return *curr; }
    bool operator!=(const Ptr& p) const {
        return curr != p.curr;
    }
    Ptr& operator++() {
        ++curr;
        return *this;
    }
private:
    T* curr;
};
```

2.
```
class Accumulator {
    friend ostream& operator<<(ostream& s, const Accumulator& a);
public:
    Accumulator() : sum(0) {}
    Accumulator& operator+=(int nbr) {
        history.push(nbr);
        sum += nbr;
        return *this;
    }
    void undo() {
        if (!history.empty()) {
            sum -= history.top();
            history.pop();
        }
    }
    void commit() {
        history.clear();
    }
    void rollback() {
        while (!history.empty()) {
            undo();
        }
    }
private:
    int sum;
    stack<int> history;
};

ostream& operator<<(ostream& s, const Accumulator& a) {
    return s << a.sum;
}
```

3.
```
class RefHandler {
public:
    RefHandler(const string& fn) : filename(fn) {}
    void print();
private:
    string filename;
    map<string, int> labels;
    void collect_labels();
    string line;
    string find_command(string& arg) const;
    void replace_command(const string& command, const string& rep);
};

string RefHandler::find_command(string& arg) const {
    auto start_pos = line.find("\\");
    if (start_pos == string::npos) {
        return "";
    }
    ++start_pos;
    auto end_pos = line.find("{", start_pos);
    string command = line.substr(start_pos, end_pos - start_pos);
    start_pos = end_pos + 1;
    end_pos = line.find("}", start_pos);
    arg = line.substr(start_pos, end_pos - start_pos);
    return command;
}

void RefHandler::collect_labels() {
    ifstream in(filename);
    int section_counter = 0;
    int line_nbr = 0;
    while (getline(in, line)) {
        ++line_nbr;
        string arg;
        string command = find_command(arg);
        if (command == "section") {
            ++section_counter;
        } else if (command == "label") {
            if (labels.find(arg) != labels.end()) {
                cerr << "Duplicate label '" << arg << "' at input line "
                        << line_nbr << endl;
            } else {
                labels.insert({arg, section_counter});
            }
        }
    }
}

void RefHandler::replace_command(const string& command, const string& rep) {
    auto start_pos = line.find("\\" + command);
    auto end_pos = line.find("}", start_pos + command.length() + 2);
    line.replace(start_pos, end_pos - start_pos + 1, rep);
}

void RefHandler::print() {
    collect_labels();
    ifstream in(filename);
    ofstream out(filename + ".res");
    int section_counter = 0;
    int line_nbr = 0;
    while (getline(in, line)) {
        ++line_nbr;
        string arg;
        string command = find_command(arg);
        if (command == "section") {
            ++section_counter;
            replace_command("section", to_string(section_counter) + " " + arg);
        } else if (command == "label") {
            replace_command("label", "");
        } else if (command == "ref") {
            auto it = labels.find(arg);
            string ref;
            if (it != labels.end()) {
                ref = to_string(it->second);
            } else {
                ref = "??";
                cerr << "Undefined reference '" << arg << "' at input line "
                        << line_nbr << endl;
            }
            replace_command("ref", ref);
        }
        out << line << endl;
    }
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        cerr << "Usage: ref file" << endl;
        exit(1);
    }
    RefHandler rh(argv[1]);
    rh.print();
}
```