LUND INSTITUTE OF TECHNOLOGY Department of Computer Science

Solutions, C++ Programming Examination

2015–08–28

1. a) Compilation error: the local variable `b` must be captured.
   b) Everything is fine: the *value* of the local variable `b` is captured (copied into the lambda).
   c) Undefined execution results: a *reference* to `b` is captured, but when the function exits `b` goes out of scope. So when `f2` and `f3` are called, the references are dangling.

2.
```
class Scheduler {
public:
    Scheduler(istream& in);
    void schedule();
    void print(ostream& out) const;
private:
    using row_type = bitset<80>;
    vector<row_type> rows;
    bool collides_with(int row_nbr) const;
};

Scheduler::Scheduler(istream& in) {
    string line;
    while (getline(in, line)) {
        reverse(line.begin(), line.end());
        rows.push_back(row_type(line, 0, line.size(), ' ', '1'));
    }
}

void Scheduler::schedule() {
    sort(rows.begin(), rows.end(), [](const row_type& r1, const row_type& r2)
            { return r1.count() > r2.count(); });
    for (vector<row_type>::size_type i = 1; i != rows.size(); ++i) {
        while (collides_with(i)) {
            rows[i] <<= 1;
        }
    }
}

void Scheduler::print(ostream& out) const {
    for (const auto& r : rows) {
        for (size_t j = 0; j != r.size(); ++j) {
            out << ((r[j] == 1) ? '1' : ' ');
        }
        out << endl;
    }
}

bool Scheduler::collides_with(int row_nbr) const {
    for (int i = 0; i != row_nbr; ++i) {
        if ((rows[i] & rows[row_nbr]).any()) {
            return true;
        }
    }
    return false;
}
```

3.
```
class EquivalenceClasses {
public:
    void join(int a, int b);
    int least(int n);
private:
    using set_type = bitset<256>;
    vector<set_type> sets;
    vector<set_type>::iterator find_set(int n) {
        return find_if(sets.begin(), sets.end(),
                [n](const set_type& s) { return s.test(n); });
    }
};

void EquivalenceClasses::join(int a, int b) {
    auto sa = find_set(a);
    auto sb = find_set(b);
    if (sa != sets.end() && sb != sets.end() && sa != sb) {
        *sa |= *sb;
        sets.erase(sb);
    } else if (sa != sets.end()) {
        sa->set(b);
    } else if (sb != sets.end()) {
        sb->set(a);
    } else {
        set_type s;
        s.set(a);
        s.set(b);
        sets.push_back(s);
    }
}

int EquivalenceClasses::least(int n) {
    auto it = find_set(n);
    if (it == sets.end()) {
        return n;
    } else {
        int i = 0;
        while (!it->test(i)) {
            ++i;
        }
        return i;
    }
}
```

4.
```
void label(Image& im) {
    EquivalenceClasses ec;
    int n = 1;
    for (int r = 0; r !=  im.height(); ++r) {
        for (int c = 0; c != im.width(); ++c) {
            int p1 = (c > 0) ? im.get(r, c - 1) : 255;
            int p2 = (r > 0) ? im.get(r - 1, c) : 255;
            if (im.get(r, c) == 0) {
                if (p1 < 255 && p2 < 255) {
                    im.set(r, c, p1);
                    ec.join(p1, p2);
                } else if (p1 < 255) {
                    im.set(r, c, p1);
                } else if (p2 < 255) {
                    im.set(r, c, p2);
                } else {
                    im.set(r, c, n);
                    ++n;
                }
            }
        }
        for (int r = 0; r !=  im.height(); ++r) {
            for (int c = 0; c != im.width(); ++c) {
                if (im.get(r, c) != 255) {
                    im.set(r, c, ec.least(im.get(r, c)));
                }
            }
        }
    }
}
```