# Solutions, C++ Programming Examination

**2015–05–08**

1. The time-consuming part of the program is the copying of the image objects. Objects are copied 1) when they are inserted into the vector, 2) when the vector's capacity is exceeded and the vector's internal storage must be re-allocated.

   a) Insert the statement `images.reserve(1000);` after the creation of the vector. No re-allocation will be necessary.

   b) Add a move constructor to the class:

   ```
   Image(Image&& rhs) noexcept : w(rhs.w), h(rhs.h), pixels(rhs.pixels) {
       rhs.pixels = nullptr;
   }
   ```

   The temporary object which is returned from `readImage` will be moved into the vector, not copied.

2. 
```
template <typename InputIterator>
pair<typename iterator_traits<InputIterator>::value_type, size_t>
max_count(InputIterator first, InputIterator last) {
    using value_t = typename iterator_traits<InputIterator>::value_type;
    if (first == last) {
        return make_pair(value_t(), 0);
    }
    value_t max = *first;
    size_t count = 1;
    ++first;
    while (first != last) {
        value_t val = *first;
        if (max < val) {
            max = val;
            count = 1;
        } else if (val < max) {
            // nothing
        } else {
            ++count;
        }
        ++first;
    }
    return make_pair(max, count);
}
```

3.
```
/* bi.h */

class BI {
    friend std::ostream& operator<<(std::ostream& os, const BI& b);
public:
    BI(unsigned int nbr = 0);
    BI(const std::string& s);

    BI& operator+=(const BI& rhs);
private:
    using digit_type = unsigned char;
    std::vector<digit_type> digits;
};

std::istream& operator>>(std::istream& is, BI& b);
std::ostream& operator<<(std::ostream& os, const BI& b);

BI operator+(const BI& lhs, const BI& rhs);

/* bi.cc */

using namespace std;

BI::BI(unsigned int nbr) : BI(to_string(nbr)) {}

BI::BI(const string& s) {
    string pattern = "^(0*)(\\d+)$"; // start, optional zeros, digits, end
    regex r(pattern);                // (just to practice regexps)
    smatch results;
    if (!regex_search(s, results, r)) {
        throw illegal_number();
    }
    string match = results.str(2);
    for_each(match.rbegin(), match.rend(),
             [this](char ch) { digits.push_back(ch - '0'); });
}

BI& BI::operator+=(const BI& rhs) {
    vector<digit_type> result;
    digit_type carry = 0;
    string::size_type pos = 0;
    while (pos < digits.size() || pos < rhs.digits.size()) {
        digit_type dig1 = (pos < digits.size()) ? digits[pos] : 0;
        digit_type dig2 = (pos < rhs.digits.size()) ? rhs.digits[pos] : 0;
        digit_type sum = dig1 + dig2 + carry;
        result.push_back(sum % 10);
        carry = sum / 10;
        ++pos;
    }
    if (carry == 1) {
        result.push_back(carry);
    }
    digits = result;
    return *this;
}

istream& operator>>(istream& is, BI& b) {
    string s;
    is >> s;
    b = BI(s);
    return is;
}
```

```
ostream& operator<<(ostream& os, const BI& b) {
    copy(b.digits.rbegin(), b.digits.rend(), ostream_iterator<int>(cout));
    return os;
}

BI operator+(const BI& lhs, const BI& rhs) {
    BI sum(lhs);
    sum += rhs;
    return sum;
}
```

4. Under `public` in `BI` (the copy operations are necessary, move operations would be a bonus):

```
    BI(const BI& rhs);             // copy constructor
    BI& operator=(const BI& rhs); // copy assignment operator
    ~BI();                         // destructor
```

   Under `private`, remove the vector and insert:

```
    digit_type* digits; // array of digits
    size_t n;           // number of digits
```

5. 
```
class MorseCode {
public:
    MorseCode();
    string decode(const string& code) const;
private:
    map<string, char> table;
};

MorseCode::MorseCode() {
    ifstream def("morse.def");
    char sym;
    string code;
    while (def >> sym >> code) {
        table.insert(make_pair(code, sym));
    }
    def.close();
}

string MorseCode::decode(const string& code) const {
    istringstream iss(code);
    string s;
    string result;
    while (iss >> s) {
        auto it = table.find(s);
        result += (it != table.end()) ? it->second : '?';
    }
    return result;
}
```