Solutions, C++ Programming Examination

2014–03–15

1. a) Objects of class `List` cannot be copied since the copying functions (the copy constructor and the copy assignment operator) are deleted.

   b) Add declarations of the functions `copy_list` and `delete_list` to the private part of the class (`delete_list` can also be used by the destructor). Implementation:

```
List::List(const List& rhs) {
    copy_list(rhs);
}

List& List::operator=(const List& rhs) {
    if (&rhs == this) {
        return *this;
    }
    delete_list();
    copy_list(rhs);
    return *this;
}

void List::copy_list(const List& rhs) {
    Node* n = rhs.first;
    if (n == nullptr) {
        first = nullptr;
        return;
    }
    first = new Node(n->value, nullptr);
    Node* current = first;
    n = n->next;
    while (n != nullptr) {
        current->next = new Node(n->value, nullptr);
        current = current->next;
        n = n->next;
    }
}

void List::delete_list() {
    Node* n = first;
    while (n != nullptr) {
        Node* next = n->next;
        delete n;
        n = next;
    }
}
```

   c) Examples of use:

```
    List list_1;
    List list_2(list_1); // uses copy constructor
    list_2 = list_1;     // uses copy assignment operator
```

2.
```
vector<int> digits(int nbr) {
    vector<int> res;
    do {
        int digit = nbr % 10;
        res.push_back(digit);
        nbr /= 10;
    } while (nbr != 0);
    return res;
}

bool is_happy(int nbr) {
    unordered_set<int> already_computed;
    while (nbr != 1 && already_computed.count(nbr) == 0) {
        already_computed.insert(nbr);
        vector<int> dig = digits(nbr);
        int sum = 0;
        for_each(dig.begin(), dig.end(), [&sum](int d) { sum += d * d; });
        nbr = sum;
    }
    return nbr == 1;
}

int main() {
    set<vector<int>> unique_happy;
    for (int nbr = 1; nbr <= 1000; nbr++) {
        if (is_happy(nbr)) {
            vector<int> dig = digits(nbr);
            auto it = remove(dig.begin(), dig.end(), 0);
            dig.erase(it, dig.end());
            sort(dig.begin(), dig.end());
            if (unique_happy.count(dig) == 0) {
                cout << nbr << " ";
                unique_happy.insert(dig);
            }
        }
    }
    cout << endl;
}
```

3. Add the following definitions to class `Memory`:

```
private:
    std::unordered_map<address, byte> m; // the memory map
    address sz; // size of the memory
```

Public member functions:

```
Memory::Memory(address size) : sz(size) {}

Memory::~Memory() {}

Memory::byte Memory::read(address addr) const {
    if (addr >= sz) { throw AddressingError(); }
    auto it = m.find(addr);
    return (it != m.end()) ? it->second : static_cast<byte>(rand());
}

void Memory::write(address addr, byte b) {
    if (addr >= sz) { throw AddressingError(); }
    m[addr] = b;
}
```

4. a)
```
int a[] = {1, 7, 5, 9, 3};
my_sort(begin(a), end(a));
```

b) The function cannot sort an empty range. Insert the following statement at the start of the function:

```
if (beg == end) {
    return;
}
```

c)
```
template <typename It, typename Pred>
void my_sort(It beg, It end, Pred compare) {
    if (beg == end) {
        return;
    }
    for (; beg != end - 1; ++beg) {
        It min = beg;
        for (It pos = beg + 1; pos != end; ++pos) {
            if (compare(*pos, *min)) {
                min = pos;
            }
        }
        std::swap(*beg, *min);
    }
}

template <typename It>
void my_sort(It beg, It end) {
    my_sort(beg, end, less<decltype(*beg)>());
}
```

d)
```
vector<Point> points = {Point(1, 3), Point(4, 4), Point(1, 2)};
my_sort(points.begin(), points.end(), [](const Point& p1, const Point& p2) {
        return p1.get_x() < p2.get_x() ||
               (!(p2.get_x() < p1.get_x()) && p1.get_y() < p2.get_y()); });
```

e) The parameters `beg` and `end` must be random access iterators (`end - 1` and `begin + 1`). Lists have bidirectional iterators, so `my_sort` cannot sort lists. Corrections:

```
...
It last = end;
--last;
for (; beg != last; ++beg) {
    ...
    It pos = beg;
    ++pos;
    for (; pos != end; ++pos) {
    ...
...
```