LUND INSTITUTE OF TECHNOLOGY Department of Computer Science

Solutions, C++ Programming Examination

2017–06–05

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. The result of ”Covfefe count: ” is a pointer of type `char *` which points to the location in memory where the compiler have placed the text "Covfefe count: ". The addition results in a new pointer that points to a position in memory that is located 34567890 bytes (assuming a char is 8 bits) away in memory. In our case that means that the program attempts to access memory located outside the memory spece allocated to the program, which in turn results in an error message.

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

   c) 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;
```

4.
```
template<typename ForwardIterator>
bool is_strict_monotonic(ForwardIterator first, ForwardIterator last) {
    bool inc_mono{true};
    bool dec_mono{true};
    ForwardIterator next{first};
    ++next;

    while ((inc_mono || dec_mono) && next!=last) {
        if (!(*first<*next)) inc_mono=false;
        if (!(*next<*first)) dec_mono=false;
        ++first;
        ++next;
    }

    return inc_mono || dec_mono;
}
```

5.
```
using namespace std;

typedef map<string, size_t> count_type;
typedef map<string, size_t>::const_iterator count_iter_type;

void count_words(istream& in, count_type& count) {
    string word;
    while (in >> word) {
        ++count[word];
    }
}

bool comp_second(count_iter_type it1, count_iter_type it2)  {
    return it1->second > it2->second;
}

void print_common(const count_type& count, size_t n) {
    vector<count_iter_type> iters;
    for (count_iter_type it = count.begin(); it != count.end(); ++it) {
        iters.push_back(it);
    }
    vector<count_iter_type>::iterator sorted_end = iters.begin() + min(n, iters.size());
    partial_sort(iters.begin(), sorted_end, iters.end(), comp_second);
    for (vector<count_iter_type>::const_iterator vit = iters.begin();
            vit != sorted_end; ++vit) {
        cout << (*vit)->first << "\t" << (*vit)->second << endl;
    }
}

int main(int argc, const char** argv) {
    count_type count;
    for (int i = 1; i < argc; ++i) {
        ifstream in(argv[i]);
        if (!in) {
            cerr << "Cannot open: " << argv[i] << endl;
        } else {
            count_words(in, count);
            in.close();
        }
    }
    print_common(count, 10);
}
```