Lab 2: C++ for Competitions

Algorithmic Programming Contests 2023-2024

Wietze Koops

September 30, 2024

This lab focuses on programming in C++. In addition, it provides further training in mathematical and implementation questions. There are also a number of problems that can be solved using data structures from the C++ Standard Library.

1 C++ input and output

C++ I/O functions are available after using `#include <iostream>`. Here is an overview.

- A single word or number (e.g. an `int n;`) can be read in using `cin >> n;`
- Occasionally `getchar();` or `getline();` can be useful to read in by character or line.
- Output can be written using `cout << n;`
- To write exactly two decimal places after the point, use an I/O manipulator available after using `#include <iomanip>` as follows: `cout << fixed << setprecision(2);`.
- In interactive problems you need to flush the stream after writing output. This can be done by inserting `flush` into the output stream, i.e. `cout << flush;`. You can also write a newline and flush the stream using `cout << endl;`
- All of this is from the standard library, so you need to write `using namespace std;` at the start of your code, or write `std::cin`, etc. (In competitions, using `using namespace std;` should be easier; in other code there may be valid reasons to avoid it.)

2 C++ string library

C++ strings are available after using `#include <string>`, and are much more convenient to use than C strings (i.e. character arrays), e.g. because they automatically resize.
Some useful functions for strings can be found in Chapter 5 of the C++ Annotations.

3 Data Structures and the C++ Standard Library

- Pairs and tuples: Support structure which can combine two variables of (potentially) different data types, e.g. a `pair<int, string> p`. For a pair `p`, its first and second element are accessed using `p.first` and `p.second`.

  A pair can be constructed using `make_pair`. This allows one to directly add a new pair to a vector of pairs, for example: `vec.push_back(make_pair(42, "answer"))`.

  Tuples are similar, but for more than two elements, e.g. `tuple<int, float, char> t`. The jth element of the tuple can can be accessed using `get<j>(t)`.

- Vector: essentially a variable length array, similar to Python. Access is $O(1)$ and very fast in practice (same syntax as array, i.e. `vec[j]`), appending at the tail is $O(1)$. Inserting in the middle is $O(n)$.

  Memory can be preallocated as follows: `vector<int> vec(n)` makes a vector of length n. In this case, one can just assign new elements using e.g. `vec[j] = 42`. If no memory was preallocated, use `vec.push_back(42)`. Do not use this if you already preallocated the memory, because then it will add it after the preallocated memory!

- Lists (implemented as doubly linked list): Inserting at any position is $O(1)$ provided that you have an iterator to the position at which you insert.

- Sets: There are two types of sets, `set` and `unordered_set`. The main strength of a set is that membership can be checked efficiently: $O(\log n)$ for set, $O(1)$ for the unordered set. Use set if you need to iterate over the elements of the set, and unsorted set otherwise (e.g. if you are only using it to check membership). Useful functions: `s.insert(val)`, `s.erase(val)`, `s.count(val)` (1 if `val` is in the set, 0 if not).

- Maps (dictionaries): C++ does not offer dictionaries, but maps act very similarly, mapping a key to a value. Inserting and accessing can be done using square brackets.

  Like with sets, there are types: `map` and `unordered_map`. The latter is faster, but you cannot iterate over all keys/values stored in the map.

- Stack: Last In First Out (LIFO) data structure.

  Functions: `s.pop()` (removes first element), `s.push(value)` (adds element at the front), `s.top()` (get first element), `s.size()` all in $O(1)$.

- Queue: First In First Out (FIFO) data structure.

  Functions: `q.pop()` (removes first element), `q.push_back(value)` (adds element at the back), `q.front()` (get first element), `q.back()` (get last element), `q.size()` all in $O(1)$.

- Priority queue: Keeps the elements in a certain order, by default decreasing. For increasing order, use `priority_queue<int, vector<int>, greater<int>>` (the second argument is the underlying container, and can be taken to be a vector of the first argument). In this way it is also possible to define custom comparators.

  Functions: `q.pop()` (removes first element) and `q.top()` (get first element) in $O(1)$, `q.push()` (add element in sorted place) in $O(\log n)$.

4 Ordered list

An ordered list is a list where we can insert in $O(\log n)$ and do binary search on (i.e. find the smallest element larger than a given value, or the largest element smaller than a given value) in $O(\log n)$. Note that in a normal array/vector we can do the latter but not insertion in $O(\log n)$ time. In C++ it can be done using a multiset, which is ordered. The useful C++ functions for th lookup are `lower_bound` and `upper_bound`.

5 Reading from the book

Reading from the Competitive Programmer's Handbook by Antti Laaksonen:

- Ch.1 (Introduction): Section 1.1 (Programming languages) and 1.2 (Input and output). From Section 1.4 (Shortening code), the `typedef` command is the most useful part.
- Ch.3 (Sorting): Section 3.2 (Sorting in C++).
- Ch.4 (Data structures): Entire chapter.

6 Documentation

The C++ documentation, e.g. on these data structures (called *containers*), is available from cppreference.com. This documentation is also available during external contests (such as BAPC/NWERC) which do not allow access to the internet. Hence, it makes sense to practice finding what you need in this documentation. If there is anything that you could not find in the documentation, make a note to include it in your team reference document.

7 Data structures in Python

Since we focus in this lab on C++, we do not provide an elaborate overview of the built-in data structures in Python. However, we do want to warn you for a common mistake/inefficiency when using data structures in Python: The data structures from the `Queue` module are relatively inefficient (i.e. have a large constant factor in their running time), because they are compatible with multi-threading (i.e. the class makes sure that different threads do not overwrite each other's actions). This is not necessary for single-threaded programs (multi-threading is not allowed in Competitive Programming), but causes a lot of overhead.

For a priority queue, use the `heapq` module. For a stack or queue, use the `deque` module, which provided a doubly-ended queue which can be used both as stack and as queue.

8 Assignment

8.1 Assignment instruction

The lab assignment is due two weeks after the lab session. If you cannot meet this deadline due to exceptional circumstances, send an email before the deadline to `apc@rug.nl`.

Deliverable, to be sent to `apc@rug.nl` before the deadline:

- For each problem that you solved, your solution and a one or two sentence summary of your solution. In addition, you can send in any failed attempts if you are unsure why they failed to receive feedback on them.
- For each problem that you did not solve, your failed attempts. If you already know/suspect the reason why it doesn't work, also mention this.

  If you didn't write any code (because you had no idea where to start), write a few paragraphs on what you tried; in particular, did you figure out why the answer to the Sample Input is what it is? Did you think about the required complexity? What techniques/data structures/algorithms did you consider?

Please include the word 'assignment' in the subject and body of your email.

You may discuss solution strategies with other students. However, the solution that you submit via email (Deliverable) must be written completely by yourself.

8.2 Assignment problems

The assignment problems corresponding to this lab are:

- (CSES problem set) Ferris Wheel
- (NCPC 2015) Disastrous Downtime
- (CSES problem set) Concert Tickets