Pragmatic Programming
(for the non-competitive programmer)
Session 1 - Introduction

Max Nilsson

19th September

How This Course Will Work

- This course will be the solution to the optimization problem

$$\underset{\substack{\text{hp, fun}\\ \text{overhead}}}{\text{maximize}}\ \text{hp} + \text{fun} - \text{overhead}$$

- The Aim: improve your problem solving skills, while learning fundamental concepts, algorithms and coding C/C++ - which can help you in your future research!
- The details:
  - 7.5 hp
  - Weekly 2 hour sessions
  - Examiner: Pontus Giselsson
- Each week we will
  - Discuss last week's topics
  - Introduce a new topic with accompanied light reading and **Kattis** problems
  - Code review each others solutions
  - Expand the **corpus**

For the Non-Competitive Programmer

- This is not a course in competitive programming.
- Instead, we will try to *extract* the best things from competitive programming, in a study circle setting, such as
  - Quickly identify what class a problem lies in (Graph/Dynamic Programming (DP)/Searching)
  - Knowledge of a large variety of algorithms
  - Experience of solving problems
  - Being able to debug your code without a debugger
  - Being able to write semi-efficient code
  - Being comfortable in `C`/`C++`

Why Follow This Study Circle?

- *To successfully solve various data-related problems, both solid theoretical knowledge and the ability to apply this knowledge in practice are required. This course focuses on the ability to apply theoretical knowledge of algorithms, data structures, and complexity to given problems. Being able to quickly analyze a problem, assess the complexity of proposed algorithms, and implement a solution rapidly and accurately are valuable skills in the workplace.* - from the KTH course "Problem Solving and Programming Under Pressure"
- Hopefully it will be fun! and you will find your fun niche!
  - Solving easy/hard problems
  - Solving problems efficiently
  - Solving problems fast
  - Implementing data structures/algorithms efficiently

Resources

- We will have a look at the following books
  - Principles of Algorithmic Problem Solving, *by Johan Sannemo* (once used in the Competetive Programming course at KTH)[1]
  - Guide to Competitive Programming - Learning and Improving Algorithms Through Contests, *by Antti Laaksonen*[2]
  - *(Supplementary Material) Competitive Programming,* by Steven and Felix Halim[3]
- We will solve problems from the **Kattis** website

  `https://open.kattis.com/`
- The gitlab page
- Join the "Pragmatic Programming" WhatsApp group

---

[1] Johan Sannemo. "Principles of Algorithmic Problem Solving". In: *Draft version* (2018).

[2] Antti Laaksonen. *Guide to competitive programming*. Springer, 2020.

[3] Steven Halim, Felix Halim and Suhendry Effendy. *Competitive programming 4: The new lower bound of programming contests in the 2020s*. Lulu. com, 2018.

Examination

Each week, a set of problems will be provided for you to solve, and if you successfully tackle at least half of these problems every week, you will pass the course.

You can find the problems here: `https://lth.kattis.com/courses`

This Week

- Setting up your workspace
  - Install a `C++` compiler (I use g++ installed with

    `sudo apt-get install g++`)

  - Choose an editor (I use `Sublime`)
  - Make sure you can compile a `Hello World` program (I compile with

    ```
    g++ -std=c++14 -o solve solve.cpp ; ./solve
    ```

    on macOS)
  - If you run into any problems - write in the Whatsapp group and I am sure *someone* can help you (this *someone*'s name starts with *Max N* but does **not** end in *ilsson*)
  - Solve your first **Kattis** problem (I recommend Hello World)
- Skim-read Chapters 1 and 2 in *Sannemo* (the team leader for the Swedish team in the Olympics of competitive programming, also at rank 13 on the Kattis global leaderboard) and *Laaksonen* (Department of Computer Science, University of Helsinki).
- Solve some of the first week's problems (see upcoming slide)

Subjective Tips

- Don't spend to much time optimizing your workflow - spend the time solving problems.
- Use `C++`. Otherwise you will learn the hard way why you should use `C++`. The main reasons:
  - `C` has no built-in "dynamic memory types" such as

    `string, vector, map, set, pair<T, S>`

  - `C` has no built-in algorithms such as

    `sort, next_permutation, lower_bound`

  - I/O in `C` is fast but not very easy (for instance, dynamically `realloc()` for `char*` ...)
  - `C++` has all the great things `C` has such as pointers and structs.
- Use `include <bits/stdc++.h>` for all relevant standard libraries. On macOS there is some problems with this, but on Linux it should work fine.

Subjective Tips

- Learn coding C++ by doing easy problems - I claim that there is no faster and more fun way of learning the basic syntax/structures of a language.
- Use already implemented algorithms when you need them. A good source is

  `https://github.com/kth-competitive-programming/kactl`

  and soon our **corpus** of algorithms.
- Use `using namespace std` and well-chosen `#define`, `typedef` and some debug flag.

My Setup

```
#include <bits/stdc++.h>
using namespace std;
#define Mod(x,y) (((x)%(y)+(y))%(y))
#define rep(i, a, b) for(int (i) = (a); (i) < (b); ++(i))
#define all(x) begin(x), end(x)
#define pb push_back
#define gcd __gcd
#define sz(x) (int)(x.size())
typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> pii;
typedef vector<int> vi;
typedef vector<pii> vii;
```

My Setup

```
const bool debug = false;
void solve() {
    [...  code here ...]
}
int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    cout << setprecision(10) << fixed;
    int test = 1;
    //cin >> test;
    while(test--) solve();
    return 0;
}
```

To Get You Started - I/O

- I/O is really simple in C++. Almost all you need to know is

  `cin >>, cout <<, cin.peek(), cin.ignore(), getline(cin, s).`

  - `cin >> x;` reads into `x` given the type of `x` (`int, ll, double, string`)
  - `cout << x << '\n';` outputs `x` to the standard output, with a new-line.
  - `cin.peek()` gives the next character in the standard input, **without** consuming it.
  - `cin.ignore()` consumes the next character in the standard input.
  - `getline(cin, s)` reads the entire line into `string s;`
- Example: How to read a vector of n integers

```
const int mxn = 1e5;
vi v(mxn);
int n;
cin >> n;
rep(i, 0, n) cin >> v[i];
```

To Get You Started - I/O

- Example: How to read an unknown amount of strings and display the first letter of each string

```
string s;
while(cin >> s) cout << s[0] << '\n';
```

- Example: How to read an $n \times m$ matrix of characters

```
const int mxn = 100;
char mat[mxn][mxn];
int n, m;
cin >> n >> m;
rep(i, 0, n) rep(j, 0, m) cin >> mat[i];
```

To Get You Started - Basic Data Structures

- The usual suspects: `char`, `string`, `int`, `ll`, `double`. Largest values of `int`, `ll` are

$$\texttt{INT\_MAX} = 2^{31} - 1 \approx \texttt{2e9}, \texttt{LLONG\_MAX} = 2^{63} - 1 \approx \texttt{9e18}.$$

  Sometimes in Kattis you need larger values, then you write in Python/Java. No such problems will be chosen in this course.
- Arrays of static size: `int a[mxn]`
- Vectors of dynamic size: `vector<double> v` with methods `.pb(x)`, `.size()`
- Types of queues: `stack<int>`, `queue<int>`, `priority_queue<int>`
- Maps: `set<int>`, `map<double, int>` and their `unordered_` variants, with look-up time of $\mathcal{O}(\log n), \mathcal{O}(1)$ respectively.
- Pairs of types: `pair<int, double>` with some different initializations:

```
pair<int, double> p(0, 1.);
pair<int, pii> q = make_pair(0, pii(1, 2));
```

To Get You Started - Kattis Error Messages

- There are several types of things that can go wrong when submitting to Kattis
  - **Compile Error** - Your program failed to compile. This should never happen if you can compile your program locally.
  - **Run Time Error** - Error during the run time of your program. Probably either a segmentation fault or a signed integer overflow.
  - **Output Limit Exceeded** - This has never happened to me.
  - **Memory Limit Exceeded** - This has happened to me once or twice. Try using more efficient data structures or use `int` instead of `ll` (usually there is a more elegant solution which circumvents these issues).
  - **Time Limit Exceeded** - Your program is too inefficient (in `C++` this is usually equivalent with wrong time complexity) or is stuck in some infinite loop/recursion.
  - **Wrong Answer** - Your program is wrong. In very very rare cases there is an error in the Kattis test cases (this has happened to me twice).
  - **Judge Error** - This has never happened to me (and you are very lucky if it happens to you).

Input Size vs. Time Complexity

If you already know about time complexity, the following table is good to know about when dealing with time limit exceeded:

Input Size	Time Complexity	Examples
$n \leq 10$	$\mathcal{O}(n!)$	`next_permutation`
$n \leq 20$	$\mathcal{O}(2^n)$	Generate all subsets
$n \leq 500$	$\mathcal{O}(n^3)$	Floyd-Warshall, Bipartite Matching
$n \leq 5000$	$\mathcal{O}(n^2)$	2-D Dynamic Programming
$n \leq 10^6$	$\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$	Segment Trees, Convex Hull
n is large	$\mathcal{O}(1)$	Bitwise operations, Math

Table: Table 3.1 from Laaksonen[4]

For an example of "n is large", see Keep Calm And Carry Off or nnnnn where $n \leq 10^{10^6}$.

[4] Antti Laaksonen. *Guide to competitive programming*. Springer, 2020.

Preliminary Schedule

Week 1	First Steps: Setup C++
Week 2	Data Structures and Complexity
Topic 1	Graphs
Topic 2	Dynamic Programming
Topic 3	Square Root Techniques
Topic 4	Flows, Matchings, and Cuts
Topic 5	Fenwick and Segment Trees
Topic 6	Treaps and Topological Sorting
Topic 7	Tries, Suffix Arrays, and Strings
Topic 8	Geometry, Number Theory, and Probability
Topic 9	Computational Game Theory (Sprague–Grundy)

Problems - Week 1

- Hello World!
- R2
- Faktor
- Herman
- Pizza Crust
- Aaah!
- Quadrant Selection
- Spavanac
- Stuck In A Time Loop
- Rot
- Fizz Buzz (Easy)
- Fizz Buzz 2 (Medium)
- Fizz Buzz 3 (Hard)
- Cold-puter Science
- Tarifa
- Trik
- Arithmetic Functions
- Logic Functions
- Booking a Room
- Everywhere

Live Coding!

Left Beehind