CPSC 490 – Problem Solving in Computer Science

Lecture 1: Introduction, IO, Floating Point Arithmetic

Jason Chiu and Raunak Kumar

Based on slides by Paul Liu (2014), Kuba Karpierz and Bruno Vacherot (2015)

2017/01/04

University of British Columbia

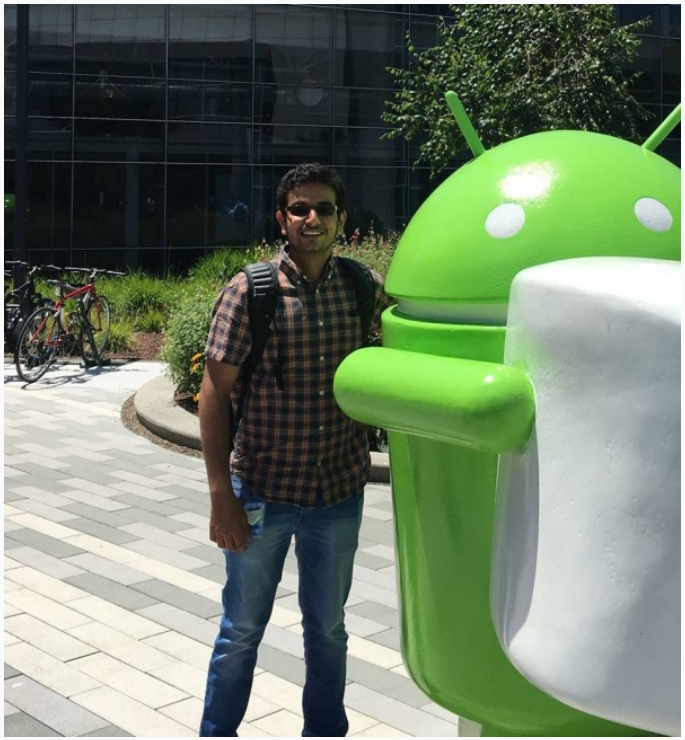
Coordinators

Jason Chiu

- Section 201
- MWF 11am-12pm, ORCH 3062

Raunak Kumar

- Section 202
- MWF 3-4pm, ORCH 3068



Why CPSC 490?

Why CPSC 490?

- **Learn** useful algorithms not taught in any other course

Why CPSC 490?

- **Learn** useful algorithms not taught in any other course
- **Solve** problems with algorithms from CPSC 221/320/420 and new algorithms from this course!

Why CPSC 490?

- **Learn** useful algorithms not taught in any other course
- **Solve** problems with algorithms from CPSC 221/320/420 and new algorithms from this course!
- **Practice** coding, problem solving, debugging

Why CPSC 490?

- **Learn** useful algorithms not taught in any other course
- **Solve** problems with algorithms from CPSC 221/320/420 and new algorithms from this course!
- **Practice** coding, problem solving, debugging
- **Rock** at programming contests and interviews

Why CPSC 490?

- **Learn** useful algorithms not taught in any other course
- **Solve** problems with algorithms from CPSC 221/320/420 and new algorithms from this course!
- **Practice** coding, problem solving, debugging
- **Rock** at programming contests and interviews

Make UBC Great Again

Course Information

Course website: `http://www.ugrad.cs.ubc.ca/~cs490/2016W2/`

- Proposal and syllabus
- Lecture slides
- Assignments
- Presentation topics

Piazza: `https://piazza.com/ubc.ca/winterterm22016/cpsc490`

How to get help: after class or Piazza

Grading Scheme

No exams!

- 80% homework (6)
- 15% presentations (2)
- 5% participation

Assignments

There will be 6 assignments, approximately one every two weeks.
The first assignment will be released at 8pm today!

Programming

- Write code to solve problem
- Submit to online judge at course website
- 5-8 problems per assignment – need 4-6 for 100%

Written

- Optional extra marks instead of programming
- Carefully justify your solution

Marks over 100 in any assignment “overflow” to other assignments!

Presentations

Share your knowledge of esoteric algorithms or your ingenious solutions to difficult problems!

Two presentations per person: before reading week (02/08 - 02/17), before end of term (03/29 - 04/06).

Academic Policy

Using other people's code

- Allowed: lecture notes/slides, UBC/Stanford Code Archive
- Not allowed: any other source (internet, other students, etc.)

Collaboration

- Allowed: discuss solutions with other people
- Not allowed: share code

You must cite

- All sources other than: lecture, UBC/Stanford Code Archive
- Everyone you collaborated with

Bonus Points

For each programming problem, mark = base + bonus.

Base mark is determined by coordinators, based on difficulty.

Bonus depends on how many people solve the problem.

Number of solves	Bonus
0	+1000000 marks
1-2	+10 marks
3-4	+8 marks
5-8	+5 marks
9-∞	no bonus

In addition there will be one or two "special" class-wide problems, most likely in assignment 3 and/or 6, where if anyone solves it, the entire class gets +1% on the final grade.

Prerequisites

- CPSC 320 recommended
- Familiar with one of: C, C++, Java, Python (slow!)
  (Extra languages may be provided on request, but judge time limit is set based on C++ / Java)
- Willing to ~~sell your soul~~ solve LOTS of problems!

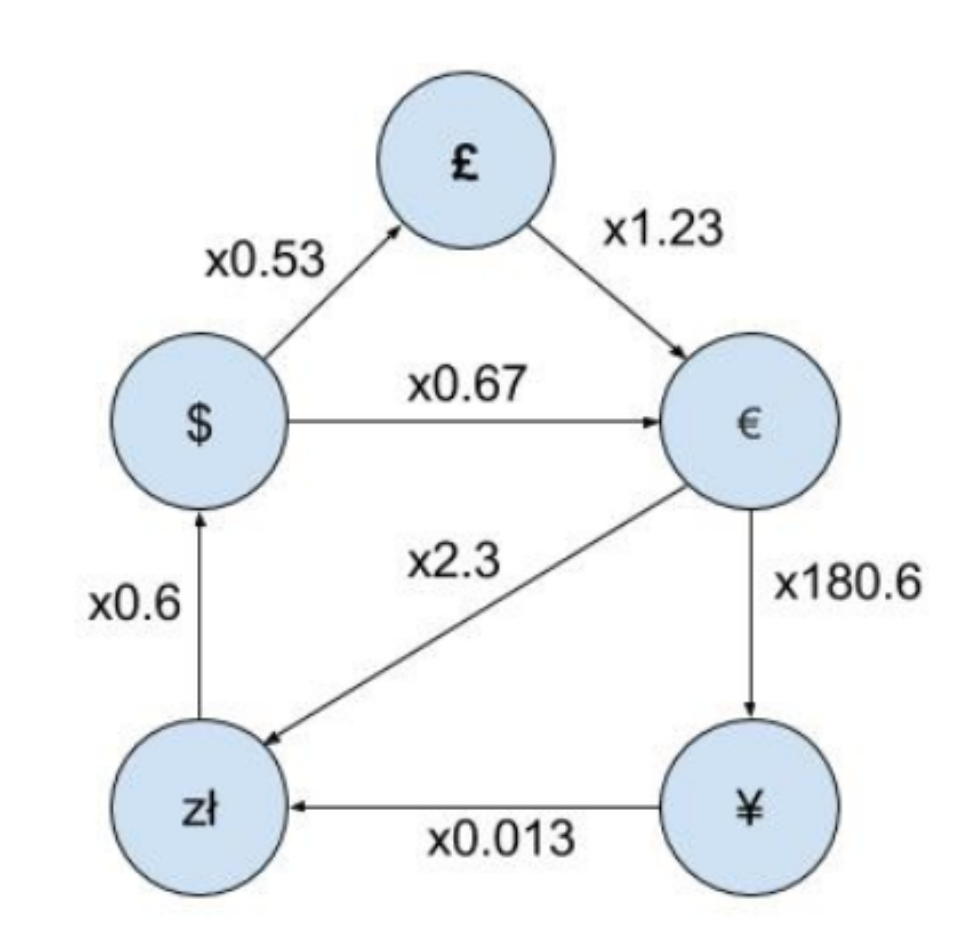
Topics

Non-exhaustive list of topics (suggestions are welcome!):

Topic	Examples
Introduction	IO, floating point, brute force
Graph Algorithms	BFS, DFS, MST, shortest path
Dynamic Programming	LCS, LIS, edit distance, bitmask, matrix exponentiation, graph/tree DP
Strings	trie, KMP, Aho Corasick, suffix array
Math**	number theory, combinatorics, linear algebra, numerical optimization
Data Structure	segment tree, bit-indexed tree
Computational Geometry	convex hull, rotating calipers, sweep
Network flow	maximum matching, maximum flow

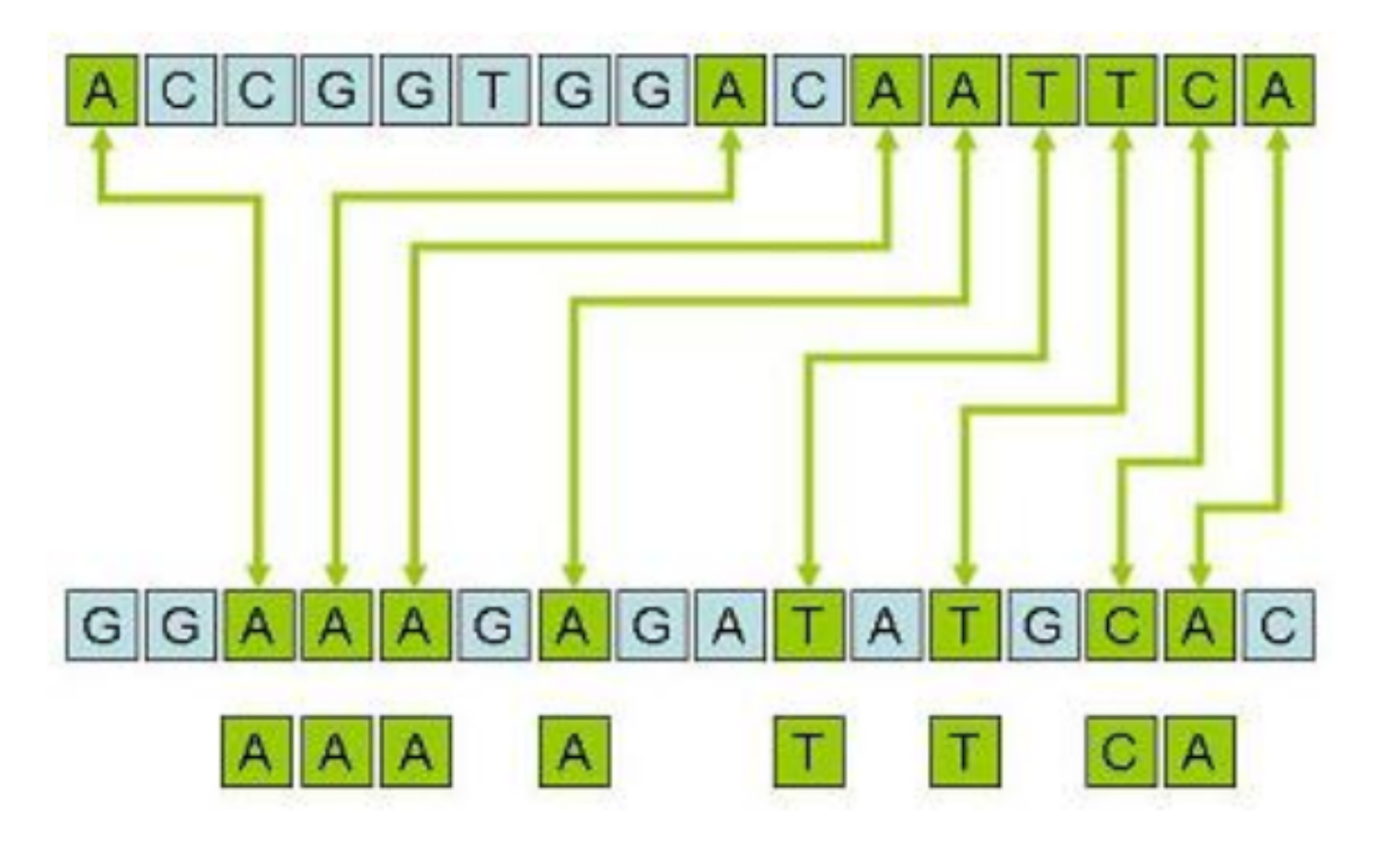
** Will be covered by student presentation, or lecture if we have time.

Graph Algorithms

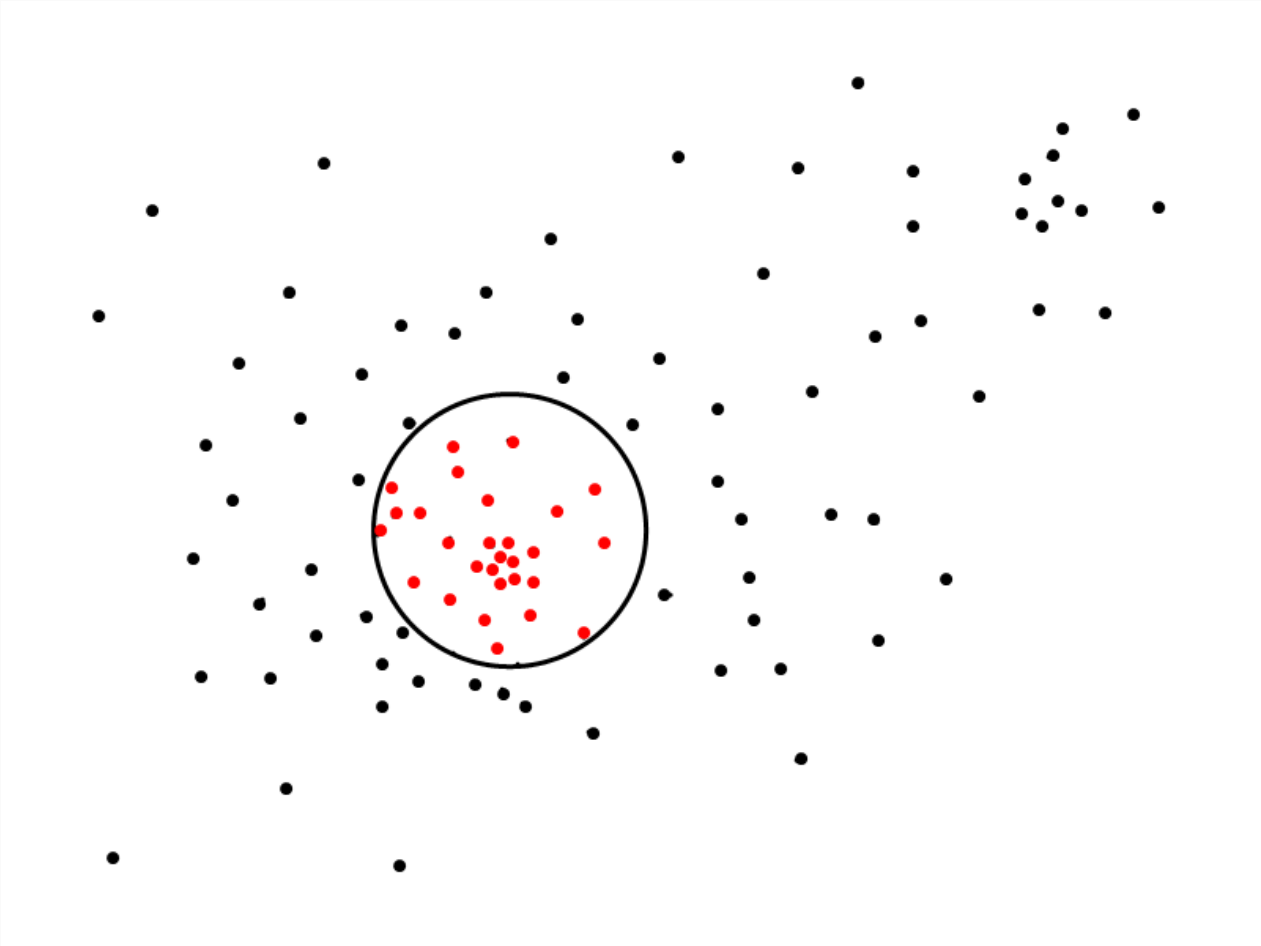


Can we make infinite money from these exchange rates?

Dynamic Programming

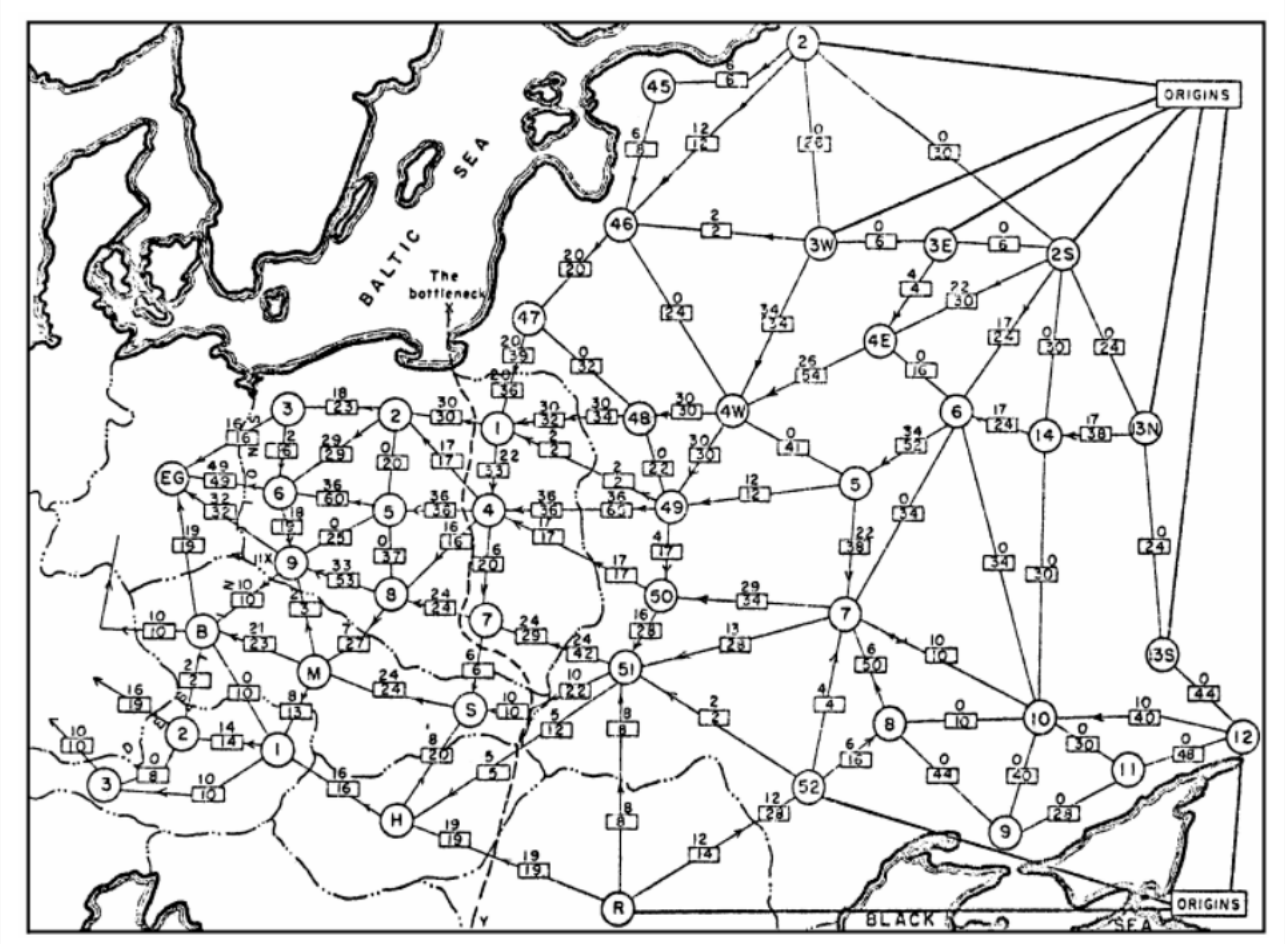


How well do these genes match up?



How many aliens can we kill with a single bomb?

Network Flow



How many rail links do we have to destroy to cut off supply?

Let’s start coding!

Problem 1

Input: A single integer, $1 \leq n \leq 10$, followed by n lines each with two integers, $0 \leq a_i, b_i \leq 10^6$, separated by a space.

Output: On the i-th line, output $a_i \times b_i$

Try at: `http://www.ugrad.cs.ubc.ca/~cs490/2016W2/judge/io/`

Easy, right??

Problem 2

Input: A single integer, $1 \leq n \leq 2,000,000$, followed by n lines each with two space-separated signed 32-bit ints, a_i and b_i ($1 \leq i \leq n$)

(In other words, $-2^{32} \leq a_i, b_i \leq 2^{32} - 1$ for all $1 \leq i \leq n$)

Output: On the i-th line, output $a_i \times b_i$

Try at: `http://www.ugrad.cs.ubc.ca/~cs490/2016W2/judge/io/`

What can possibly go wrong???

Analysis of Problem #1 and #2

- Common errors: slow IO, overflow, etc.
- What if the given numbers are 10^{100} instead?
- What if we just want the answer mod c?
- Always think about edge cases, worse case performance, etc.

Input / Output

C++

- `cin`/`cout`: easy to use but slow
  - De-sync with scanf/printf to get faster performance: `ios_base::sync_with_stdio(0); cin.tie(0);`
  - Don't flush buffer: `cout << '\n'` is faster than `cout << endl`
- `scanf`/`printf`: faster than `cin`/`cout`
- Black magic with `getc()`, see e.g. `http://ideone.com/fOK6CW`

Input / Output

C++

- `cin`/`cout`: easy to use but slow
  - De-sync with scanf/printf to get faster performance: `ios_base::sync_with_stdio(0); cin.tie(0);`
  - Don't flush buffer: `cout << '\n'` is faster than `cout << endl`
- `scanf`/`printf`: faster than `cin`/`cout`
- Black magic with `getc()`, see e.g. `http://ideone.com/fOK6CW`

Java

- `Scanner`: versatile but slow
  - has `nextBigInteger()`, `nextLine()`, etc.
- `BufferedReader`/`Writer`: fast, but have to parse manually

Input / Output

C++

- `cin`/`cout`: easy to use but slow
  - De-sync with scanf/printf to get faster performance: `ios_base::sync_with_stdio(0); cin.tie(0);`
  - Don't flush buffer: `cout << '\n'` is faster than `cout << endl`
- `scanf`/`printf`: faster than `cin`/`cout`
- Black magic with `getc()`, see e.g. `http://ideone.com/fOK6CW`

Java

- `Scanner`: versatile but slow
  - has `nextBigInteger()`, `nextLine()`, etc.
- `BufferedReader`/`Writer`: fast, but have to parse manually

Python

- `raw_input`/`print`: easy to use but slow
- `stdin`/`stdout`: fast, slighly annoying to use in comparison

Using cin/cout in C++

```cpp
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
    // un-sync with scanf/printf for fast performance
    ios_base::sync_with_stdio(0); cin.tie(0);

    int a; long long b; double c; string s;
    cin >> a >> b >> c >> s;
    cout << setprecision(16) << fixed
         << a << " " << b << " " << c << " "
         << s << endl;
    return 0;
}
```

Using scanf/printf in C/C++

```c
#include "stdio.h"
int main() {
    int a; long long b; double c; char s[100];
    scanf("%d %lld %lf %s", &a, &b, &c, s);
    printf("%d %lld %.16f %s\n", a, b, c, s);
    return 0;
}
```

Using Scanner in Java

```java
import java.util.*;
public class A {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int a = sc.nextInt();
        long b = sc.nextLong();
        double c = sc.nextDouble();
        String s = sc.next();

        System.out.println(a + " " + b + " " +
                String.format("%.16f", c) + " " + s);
        // OR
        System.out.printf("%d %d %.16f %s\n", a, b, c, d);

        sc.close();
    }
}
```

Using BufferedReader in Java

```java
import java.io.*;
import java.util.*;
public class A {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(
                new InputStreamReader(System.in));
        PrintWriter writer = new PrintWriter(System.out);

        StringTokenizer toks = new StringTokenizer(reader.readLine());
        int a = Integer.parseInt(toks.nextToken());
        long b = Long.parseLong(toks.nextToken());
        double c = Double.parseDouble(toks.nextToken());
        String s = toks.nextToken();

        writer.println(a + " " + b + " " +
                String.format("%.16f", c) + " " + s);
        // OR
        writer.printf("%d %d %.16f %s\n", a, b, c, d);
        reader.close(); writer.close();
    }
}
```

Using raw_input/print in Python

```
1  def main():
2      a, b, c, s = raw_input().split()
3      a = int(a)
4      b = int(b)
5      c = float('{0:.16f}'.format(float(c)))
6      print('{} {} {} {}'.format(a, b, c, s))
7
8  if __name__ == '__main__':
9      main()
```

Using stdin/stdout in Python

```python
from sys import stdin, stdout

def main():
    a, b, c, s = stdin.readline().split()
    a = int(a)
    b = int(b)
    c = float('{0:.16f}'.format(float(c)))
    stdout.write('{} {} {} {}\n'.format(a, b, c, s))

if __name__ == '__main__':
    main()
```

Problem 3

Input: A single line with an integer, $1 \leq n \leq 1000$, followed by a single line with n space-separated decimal numbers $a_1, \ldots, a_n$ where $10^{-9} \leq |a_k| \leq 10^9$ for all $1 \leq k \leq n$.

Output: sum of $a_1, \ldots, a_n$

Common Pitfalls with Floating Point

Common Pitfalls with Floating Point

Numerical Errors

- Cancellation error: 1.00000000001234 - 1.0000000001233

Common Pitfalls with Floating Point

Numerical Errors

- Cancellation error: 1.00000000001234 - 1.00000000001233
- Add/subtract big and small numbers: 100000000 + 0.000000001

Common Pitfalls with Floating Point

Numerical Errors

- Cancellation error: 1.00000000001234 - 1.0000000001233
- Add/subtract big and small numbers: 100000000 + 0.000000001
- Error amplification: $y_n = 1/n - 10y_{n-1}$

Common Pitfalls with Floating Point

Numerical Errors

- Cancellation error: 1.00000000001234 - 1.0000000001233
- Add/subtract big and small numbers: 100000000 + 0.000000001
- Error amplification: $y_n = 1/n - 10y_{n-1}$
- Overflow (NaN!), underflow (denormal numbers = slow!)

Common Pitfalls with Floating Point

Numerical Errors

- Cancellation error: 1.00000000001234 - 1.0000000001233
- Add/subtract big and small numbers: 100000000 + 0.000000001
- Error amplification: $y_n = 1/n - 10y_{n-1}$
- Overflow (NaN!), underflow (denormal numbers = slow!)

Round / Floor / Ceil: not so simple!

- Rounding strategies: up, down, to nearest even

Common Pitfalls with Floating Point

Numerical Errors

- Cancellation error: 1.00000000001234 - 1.0000000001233
- Add/subtract big and small numbers: 100000000 + 0.000000001
- Error amplification: $y_n = 1/n - 10y_{n-1}$
- Overflow (NaN!), underflow (denormal numbers = slow!)

Round / Floor / Ceil: not so simple!

- Rounding strategies: up, down, to nearest even
- 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 0.5 ??

Common Pitfalls with Floating Point

Numerical Errors

- Cancellation error: 1.00000000001234 - 1.0000000001233
- Add/subtract big and small numbers: 100000000 + 0.000000001
- Error amplification: $y_n = 1/n - 10y_{n-1}$
- Overflow (NaN!), underflow (denormal numbers = slow!)

Round / Floor / Ceil: not so simple!

- Rounding strategies: up, down, to nearest even
- 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 0.5 ??
- Before **rounding up**, we should <u>add small epsilon</u>

Common Pitfalls with Floating Point

Numerical Errors

- Cancellation error: 1.00000000001234 - 1.0000000001233
- Add/subtract big and small numbers: 100000000 + 0.000000001
- Error amplification: $y_n = 1/n - 10y_{n-1}$
- Overflow (NaN!), underflow (denormal numbers = slow!)

Round / Floor / Ceil: not so simple!

- Rounding strategies: up, down, to nearest even
- 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 0.5 ??
- Before **rounding up**, we should <u>add small epsilon</u>

Comparing floating point numbers

- Bad: `if (a == b) { ... }`

Common Pitfalls with Floating Point

Numerical Errors

- Cancellation error: 1.00000000001234 - 1.0000000001233
- Add/subtract big and small numbers: 100000000 + 0.000000001
- Error amplification: $y_n = 1/n - 10y_{n-1}$
- Overflow (NaN!), underflow (denormal numbers = slow!)

Round / Floor / Ceil: not so simple!

- Rounding strategies: up, down, to nearest even
- 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 0.5 ??
- Before **rounding up**, we should <u>add small epsilon</u>

Comparing floating point numbers

- Bad: `if (a == b) { ... }`
- Good: `if (abs(a-b) < 1e-8) { ... }`

How to Practice

- UBC ACM Practice
  - Every Wednesday 4-7pm, Saturday 11am-4pm
  - Free food!
- Online judges (CodeForces, TopCoder, SPOJ, Kattis, UVa, etc.)
- Google Code Jam, Facebook Hacker Cup (this month!)

Next Class

- How to implement brute force algorithms efficiently
- How to minimize bugs in implementation
- Graph data structures
- Some O(1) memory problems