# Course Intro / Logistics

# layers of abstraction: programs (1)

| | |
|---|---|
| `map.insert(key, value)` | Objects, etc. |
| `x += y` | High-level lang.: C++ |
| `add rax, rbx` | Assembly: X86-64/IBCM |
| `1110 1111` | Machine code: IBCM |
| | Hardware: (not this course) |

# layers of abstraction: data (2)

| | |
|---|---|
| `string, map<int, int>` | Data Structures |
| `char data[10]` | Arrays |
| `char data` | Primitive Types |
| `@ 0x9cdf4123: 0x12` | Addresses/Memory |
| `01101011` | Bits |

# lectures

I (Charles Reiss) will audio+screenrecord my lectures

- intend to find a way to post them later in the same day
- suggest VLC for viewing (supports changing speed!)
- how posted (where on webiste, etc.) to-be-determined

lecture attendence is strongly recommended, but ...

I won't check

# different lecturers?

Mark Floryan also teaches this class

we are giving seperate lectures

different slidedecks

- but similar
- I made my slides by looking at Floryan's...
- (but have some different preferences/style than him...)

# homeworks AKA labs

weekly assignments with three parts:

*pre-lab* due Tuesday morning

*in-lab* done physically in the lab section you are registered for

*post-lab* due Friday morning

# course staff

lecturers: Mark Floryan and I (Charles Reiss)

more than 20 TAs

some graduate student graders

# announcements

course twitter feed — `@UVaCS2150`

  shown on Collab

emails to class — very sparingly

# prerequisites

C- or better in CS2110 or CS 2220

- references, classes, objects, generics (or templates)
- control structures, procedures, recursion
- writing programs longer than a screenful

C- or better in CS2102

- logarithms, sets, graphs
- proof techniques, including by contradiciton

# CS2102 as co-requisite

you may take CS2102 as a co-requisite instead

but at your own risk

we may ask exam questions that require CS2102 material

# lab swapping

no, we cannot

- change lab you are enrolled in ourselves
- increase lab capacities beyond 45 (fire marshall limits)

to switch to an *open* lab, you can use “Edit Class” in SIS

- do not drop the course and readd (you may end up on the waitlist)

if you and another student want to swap labs,
Engineering main office in Thornton A122 may be able to do this

- you can try to find students to do this with on Piazza

# honor-related policies

do **not** share your code

do **not** look at another student's code

do **not** try to hack the submission system

do **not** share midterm details with students who haven't taken it yet

do **not** release your source code online

when we ask for assembly files, do **not** submit compiler-generated files unless otherwise allowed

# honor-related policies

do **not** share your code

do **not** look at another student's code

do **not** try to hack the submission system

do **not** share midterm details with students who haven't taken it yet

do **not** release your source code online

when we ask for assembly files, do **not** submit compiler-generated files unless otherwise allowed

you **must not** do your work in a public github repo

# honor-related policies

do **not** share your code

do **not** look at another student's code

do **not** try to hack the submission system

do **not** sh en't taken it yet

do **not** re

probably the lab is about writing assembly, **not** using compilers...

when we ask for assembly files, do **not** submit compiler-generated files unless otherwise allowed

# honor-related policies

do **not** share your code

do **not** look at another student's code

past student, present student, …

do **not** try to hack the submission system

do **not** share midterm details with students who haven't taken it yet

do **not** release your source code online

when we ask for assembly files, do **not** submit compiler-generated files unless otherwise allowed

# academic honesty

we will refer to honor violations/cheating to the honor commitee

we will also give you an F in the course for them

# grading

45% labs

30% midterms — in lab!

25% final exam

# midterms

20 February

3 April

# late policy

see discussion linked from first lab

- summary (1): -25% for first 24 hours
- summary (2): you can request an extension for any in-lab

lab due times are strictly enforced

# compilation

does not compile = no credit

- copy and paste error? we are not going to fix it

the lab submission system tells you if it compiles

# final exam

7 May at 7PM

tell us if you have a conflict *this month*

- via support request link in git repo (later)

conflict = cannot attend the exam

- (e.g. another exam at same time)
- exams at other times on 7 May is not a conflict

# accounts

Unix lab acccounts (Olsson 001, Rice 340)

    you should get an email

Collab account

Piazza account (created when you log in first)

# git

revision control system

repositories (“repos”) of stuff

tracks changes

commonly used for group work

# course git repo

online at

- view of files: `https://github.com/markfloryan/pdr/`
- website view: `https://markfloryan.github.io/pdr/`

you can get a local copy (which is part of the first lab)

# outside of the git repo

course tools (linked from git)

- support requests
- lab submission and regrades
- office hour queue

Collab: mailing list, anonymous feedback, grading guidelines

# getting a copy of the repo

(already done on the supplied VM image)

need to have git installed

`git` command to get a copy of the repo (run once):

```
git clone https://github.com/markfloryan/pdr.git
```

creates `pdr` directory containing:
slides, labs, tutorials, etc.

(this command is in the first lab)

you do *not* need a github account

# updating your copy of the repo

change into the `pdr` directory:

`git`command:

```
git pull
```

(this command is in the first lab)

error messages? you do not have the latest version

# future assignments

preliminary future assignments in repo

may be changed up until they are released

- start early? you must figure out what these changes are

official release: announcement on twitter feed

- Wednesday/Thursday before due week

# Unix environment

you will use a Unix environment in this course

required before the first in-lab

options for your personal machine:

- a virtual machine (recommended for Windows)
- OS X: natively by installing developer tools
- install Linux, etc. on your machine

options otherwise:

- use the lab machines physically
  - but we share them with other courses

# other pre-lab tasks

complete a Unix tutorial

edit and compile some C++ code

# our VM setup

tutorial in repo

download virtualbox

download our VM image (2.5GB — suggest using University network)

login `student` (“L33T Haxor” in interface); password `password`

# demo

# questions, etc.?

Piazza

support request tool

- linked off website (later)
- preferred way for individual concerns

office hours (faculty and TA)

- Google calendar linked off website

my (or Floryan's) office if door is open

anonymous feedback on Collab

- visible to both instructors

# office hours

will start next week

announced on calendar (linked from git)

mine in Rice 205

- if my door is open, I might talk otherwise

Floryan in Rice 203

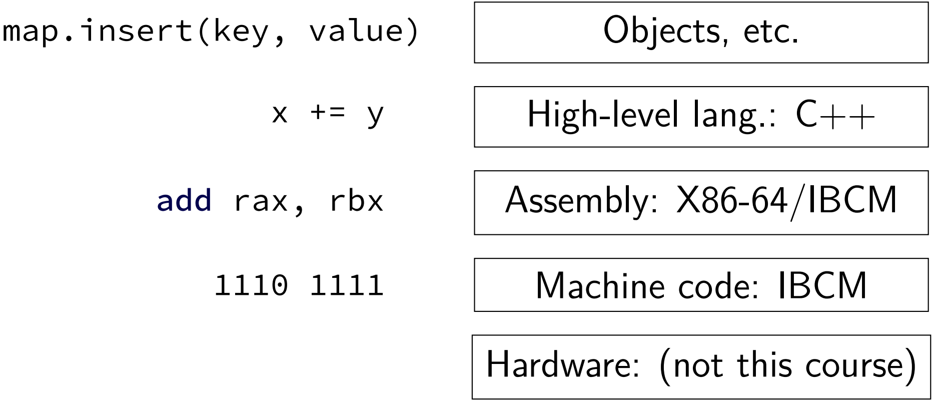
TAs in Stacks (Thornton A120)

# office hours and privacy

I generally will not close my door in my office hours

arrange a separate time if you have sensitive matters to discuss

# layers of abstraction: programs (1)



`map.insert(key, value)` — Objects, etc.

`x += y` — High-level lang.: C++

`add rax, rbx` — Assembly: X86-64/IBCM

`1110 1111` — Machine code: IBCM

Hardware: (not this course)

# layers of abstraction: data (2)

| | |
|---|---|
| `string, map<int, int>` | Data Structures |
| `char data[10]` | Arrays |
| `char data` | Primitive Types |
| `@ 0x9cdf4123: 0x12` | Addresses/Memory |
| `01101011` | Bits |

# data structures

linked lists

stacks

queues

hash tables

heaps

trees

etc.

# comparing list data structures (1)

benchmark: (linked in git repo (later))

- insert $50\,000$ elements (even integers $0$ to $100\,000$)
- search for $50\,000$ elements ($0$ to $25\,000$)
- delete $10\,000$ elements

on my desktop, Java 8, median of 3 consecutive runs

# comparing list data structures (1)

benchmark: (linked in git repo (later))

- insert $50\,000$ elements (even integers $0$ to $100\,000$)
- search for $50\,000$ elements ($0$ to $25\,000$)
- delete $10\,000$ elements

on my desktop, Java 8, median of 3 consecutive runs

| Data structure | Runtime |
|---|---|
| `Vector` | |
| `ArrayList` | |
| `LinkedList` | |
| `HashSet` | |
| `TreeSet` | |

# comparing list data structures (1)

benchmark: (linked in git repo (later))

- insert $50\,000$ elements (even integers $0$ to $100\,000$)
- search for $50\,000$ elements ($0$ to $25\,000$)
- delete $10\,000$ elements

on my desktop, Java 8, median of 3 consecutive runs

| Data structure | Runtime |
|---|---|
| `Vector` | 0.703 s |
| `ArrayList` | 0.700 s |
| `LinkedList` | 2.037 s |
| `HashSet` | 0.002 s |
| `TreeSet` | 0.010 s |

# comparing list data structures (1)

benchmark: (linked in git repo (later))

- insert 50 000 elements (even integers 0 to 100 000)
- search for 50 000 elements (0 to 25 000)
- delete 10 000 elements

on my desktop, Java 8, median of 3 consecutive runs

| Data structure | Runtime |
|---|---|
| `Vector` | 0.703 s |
| `ArrayList` | 0.700 s |
| `LinkedList` | 2.037 s |
| `HashSet` | 0.002 s |
| `TreeSet` | 0.010 s |

`HashSet`/`TreeSet` more than 350/50x faster than `Vector`/`ArrayList`

# comparing list data structures (1)

benchmark: (linked in git repo (later))

- insert $50\,000$ elements (even integers $0$ to $100\,000$)
- search for $50\,000$ elements ($0$ to $25\,000$)
- delete $10\,000$ elements

on my desktop, Java 8, median of 3 consecutive runs

| Data structure | Runtime |
|---|---|
| `Vector` | 0.703 s |
| `ArrayList` | 0.700 s |
| `LinkedList` | 2.037 s |
| `HashSet` | 0.002 s |
| `TreeSet` | 0.010 s |

`LinkedList`
3x slower than
than `Vector`/`ArrayList`

# comparing list data structures (2)

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| `Vector` | 0.703 | 0.002 | 0.507 | 0.194 |
| `ArrayList` | 0.700 | 0.001 | 0.507 | 0.192 |
| `LinkedList` | 2.037 | 0.002 | 1.521 | 0.514 |
| `HashSet` | 0.002 | 0.002 | 0.000 | 0.000 |
| `TreeSet` | 0.010 | 0.007 | 0.002 | 0.001 |
| `Vector`, sorted | 0.024 | 0.001 | 0.002 | 0.021 |

# comparing list data structures (2)

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| `Vector` | 0.703 | 0.002 | 0.507 | 0.194 |
| `ArrayList` | 0.700 | 0.001 | 0.507 | 0.192 |
| `LinkedList` | 2.037 | 0.002 | 1.521 | 0.514 |
| `HashSet` | 0.002 | 0.002 | 0.000 | 0.000 |
| `TreeSet` | 0.010 | 0.007 | 0.002 | 0.001 |
| `Vector`, sorted | 0.024 | 0.001 | 0.002 | 0.021 |

search is where most the time goes (followed by delete)

# comparing list data structures (2)

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| `Vector` | 0.703 | 0.002 | 0.507 | 0.194 |
| `ArrayList` | 0.700 | 0.001 | 0.507 | 0.192 |
| `LinkedList` | 2.037 | 0.002 | 1.521 | 0.514 |
| `HashSet` | 0.002 | 0.002 | 0.000 | 0.000 |
| `TreeSet` | 0.010 | 0.007 | 0.002 | 0.001 |
| `Vector`, sorted | 0.024 | 0.001 | 0.002 | 0.021 |

vector is slow mostly because searching unsorted list

# comparing list data structures (2)

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| Vector | 0.703 | 0.002 | 0.507 | 0.194 |
| ArrayList | 0.700 | 0.001 | 0.507 | 0.192 |
| LinkedList | 2.037 | 0.002 | 1.521 | 0.514 |
| HashSet | 0.002 | 0.002 | 0.000 | 0.000 |
| TreeSet | 0.010 | 0.007 | 0.002 | 0.001 |
| Vector, sorted | 0.024 | 0.001 | 0.002 | 0.021 |

and then delete time starts mattering

# comparing list data structures (2)

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| `Vector` | 0.703 | 0.002 | 0.507 | 0.194 |
| `ArrayList` | 0.700 | 0.001 | 0.507 | 0.192 |
| `LinkedList` | 2.037 | 0.002 | 1.521 | 0.514 |
| `HashSet` | 0.002 | 0.002 | 0.000 | 0.000 |
| `TreeSet` | 0.010 | 0.007 | 0.002 | 0.001 |
| `Vector`, sorted | 0.024 | 0.001 | 0.002 | 0.021 |

benchmark not precise enough
to measure insert time differences
except for TreeSet

# comparing list data structures (2)

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| `Vector` | 0.703 | 0.002 | 0.507 | 0.194 |
| `ArrayList` | 0.700 | 0.001 | 0.507 | 0.192 |
| `LinkedList` | 2.037 | 0.002 | 1.521 | 0.514 |
| `HashSet` | 0.002 | 0.002 | 0.000 | 0.000 |
| `TreeSet` | 0.010 | 0.007 | 0.002 | 0.001 |
| `Vector`, sorted | 0.024 | 0.001 | 0.002 | 0.021 |

TreeSet worse than HashSet?
in this benchmark, yes
but not other benchmarks

# comparing list data structures (2)

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| `Vector` | 0.703 | 0.002 | 0.507 | 0.194 |
| `ArrayList` | 0.700 | 0.001 | 0.507 | 0.192 |
| `LinkedList` | 2.037 | 0.002 | 1.521 | 0.514 |
| `HashSet` | 0.002 | 0.002 | 0.000 | 0.000 |
| `TreeSet` | 0.010 | 0.007 | 0.002 | 0.001 |
| `Vector`, sorted | 0.024 | 0.001 | 0.002 | 0.021 |

LinkedList worse than ArrayList?
in this benchmark, yes
but not other benchmarks

# comparing list data structures (3)

same benchmark, 10x original sizes:

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| `Vector` | 87.818 | 0.004 | 63.202 | 24.612 s |
| `ArrayList` | 87.192 | 0.010 | 62.470 | 24.712 s |
| `LinkedList` | 263.776 | 0.006 | 196.550 | 67.439 s |
| `HashSet` | 0.029 | 0.022 | 0.003 | 0.004 s |
| `TreeSet` | 0.134 | 0.110 | 0.017 | 0.007 s |
| `Vector`, sorted | 2.642 | 0.009 | 0.024 | 2.609 s |

# comparing list data structures (3)

same benchmark, 10x original sizes:

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| Vector | 87.818 | 0.004 | 63.202 | 24.612 s |
| ArrayList | 87.192 | 0.010 | 62.470 | 24.712 s |
| LinkedList | 263.776 | 0.006 | 196.550 | 67.439 s |
| HashSet | 0.029 | 0.022 | 0.003 | 0.004 s |
| TreeSet | 0.134 | 0.110 | 0.017 | 0.007 s |
| Vector, sorted | 2.642 | 0.009 | 0.024 | 2.609 s |

linked lists still 3x slower than vector...

# comparing list data structures (3)

same benchmark, 10x original sizes:

| Data structure | Total | Insert | Search | Delete |
|---|---|---|---|---|
| `Vector` | 87.818 | 0.004 | 63.202 | 24.612 s |
| `ArrayList` | 87.192 | 0.010 | 62.470 | 24.712 s |
| `LinkedList` | 263.776 | 0.006 | 196.550 | 67.439 s |
| `HashSet` | 0.029 | 0.022 | 0.003 | 0.004 s |
| `TreeSet` | 0.134 | 0.110 | 0.017 | 0.007 s |
| `Vector`, sorted | 2.642 | 0.009 | 0.024 | 2.609 s |

…but 350x faster became 3000x faster because of larger size
we will learn asymptotic analysis to predict this

# time/space analysis

*theoretical* analysis of time *or space* usage

theoretical = can do without implementing
…but doesn't capture all the details

general technique — not just data structures

focus: how usage will grow as data gets larger

'big picture' — ignore small factors (e.g. using floats versus doubles)

# layers of abstraction: data (2)

| | |
|---|---|
| `string, map<int, int>` | Data Structures |
| `char data[10]` | Arrays |
| `char data` | Primitive Types |
| `@ 0x9cdf4123: 0x12` | Addresses/Memory |
| `01101011` | Bits |

# the hardware/software interface

how do computers execute programs?

- what the processor wants — assembly/machine language
- how the processor works: the fetch-execute cycle
- what compilers are actually doing

how do computers store value?

- representing all sorts of numbers as bits
- the illusion of fast storage: the memory hierarchy

# course goals (Floryan's list)