Do Now Exercise

To prepare you for the lecture today, please do the following exercise.

List name of data structures that you have heard of.

COMP15: Data Structures

Week 11, Summer 2019

Admin

P5: **Word Frequency Database**

Project Due by 6pm on Sunday, August 4

About P5 grading rubrics from Matt

Questions about P5?

A10: **Showcasing portfolios**

Due by 6pm on Friday, August 9

Final Exam

on Wednesday, August 7

Final Exam

- in class; 90 mins
- closed books, closed notes, no electronic devices
- You can bring a sheet of paper (US letter size) with your handwritten notes.
  We will collect your paper at the end of the exam, so please put your name on it.
- Topics include everything from the lectures, in-class activities, labs, programming projects and Teach Yourself reports.
  (P5 and L10 are included.)

Question about the Final Exam?

Graph

Pop Quiz (Fill in the blanks.)

	Adjacency Matrix	**Adjacency List**
add an edge		
remove an edge		
has edge from a to b		
space complexity		

- Is there a cycle in the graph?
- Is there a path from Vertex A to Vertex B in the graph?
- Is the graph connected?
- etc.

Discussion:

The running time complexity of BFS and DFS?

(Assumption: Graph is implemented with Adjacency List.)

(Note on Dijkstra's algorithm)

No edges with negative weights.

The results of Hashing competition (L8)

1st Place:

2nd Place:

3rd Place:

4th Place:

5th Place:

1st Place:

2nd Place:

3rd Place:

4th Place: 90

5th Place: 92

1st Place:

2nd Place:

3rd Place: 63

4th Place: 90

5th Place: 92

1st Place:

2nd Place: 58

3rd Place: 63

4th Place: 90

5th Place: 92

1st Place: 53

2nd Place: 58

3rd Place: 63

4th Place: 90

5th Place: 92

Operator Overloading

(Notes from the live demo or live coding. Please do NOT assume the code is complete.)

```
1//Array.hpp
2#ifndef ARRAY_HPP
3#define ARRAY_HPP
4
5#include <ostream>
6
7class Array{
8public:
9  Array();
10  Array(int capacity);
11  Array(const Array& other);
12  Array& operator=(const Array& other);
13  ~Array();
14
15  void add(int number);
16  int at(int index) const;
17  int& operator[](int index); //(1)
18
19  int getCapacity() const;
20  int getSize() const;
21
22private:
23  int* numbers;
24  int size;
25  int capacity;
26};
27
28bool operator==(const Array& left, const Array& right); //(2)
29bool operator!=(const Array& left, const Array& right); //(2)
30
31std::ostream& operator<<(std::ostream& out, const Array& array); //(3)
32#endif
```

Note: (1), (2) and (3) were added to the code that we have written in Week 1.

(Notes from the live demo or live coding. Please do NOT assume the code is complete.)

```
61//In Array.cpp
62
63int& Array::operator[](int index){
64  //ToDo: check if the index is within the boundary
65  return this->numbers[index];
66}
67
68bool operator==(const Array& left, const Array& right){
69  if(left.getSize() != right.getSize()){
70    return false;
71  }
72  if(left.getCapacity() != right.getCapacity()){
73    return false;
74  }
75  for(int i = 0; i < left.getSize(); i++){
76    if(left.at(i) != right.at(i)){
77        return false;
78    }
79  }
80  return true;
81}
82
83bool operator!=(const Array& left, const Array& right){
84  return !(left == right);
85}
86
87std::ostream& operator<<(std::ostream& out, const Array& array){
88  for(int i = 0; i < array.getSize(); i++){
89    out << std::to_string(array.at(i)) << std::endl;
90  }
91  return out;
92}
```

(Notes from the live demo or live coding. Please do NOT assume the code is complete.)

```cpp
//test.cpp
#include <cassert>
#include <iostream>
#include <string>

#include "Array.hpp"

int main(){
  Array a;
  a.add(0);
  a.add(1);
  a.add(2);

  assert(a.at(0) == 0);
  assert(a.at(1) == 1);
  assert(a.at(2) == 2);

  assert(a[0] == 0);
  assert(a[1] == 1);
  assert(a[2] == 2);

  a[0] = 10;
  assert(a.at(0) != 0);
  assert(a.at(0) == 10);
  assert(a[0] != 0);
  assert(a[0] == 10);

  a.operator[](0) = 20;
  assert(a.at(0) != 10);
  assert(a.at(0) == 20);
  assert(a[0] != 10);
  assert(a[0] == 20);

  Array a2;
  a2.add(0);
  a2.add(1);
  a2.add(2);

  Array a3;
  a3.add(0);
  a3.add(1);
  a3.add(2);

  assert(a2 == a3);
  assert(a3 == a2);

  assert(a != a2);
  assert(a != a3);

  for(int i = 0; i < a3.getSize(); i++){
    std::cout << std::to_string(a3[i]) << std::endl;
  }

  std::cout << a3 << std::endl;

  return 0;
}
```

Special Topic:

Object-Oriented Design (OOD)

(Note: The OOD slides will not be posted on the course page.)

Do Now Exercise

To prepare you for the lecture today, please do the following exercise.

List name of data structures that you have heard of.

(Slide from Week 1)

Why do we concern ourselves with **Data Structures**?

(Slide from Week 1)

What are **performances** of program?

(Slide from Week 1)

Time and **Space** (and more...)

In Your Pocket

In Your Pocket

arrays

man, ssh, exit

(Slide from Week 2)

In Your Pocket

arrays
linked lists

man ssh exit pwd
cd ls valgrind

(Slide from Week 3)

In Your Pocket

arrays
linked lists
stacks
queues

man ssh exit pwd
cd ls valgrind
touch mkdir cp

(Slide from Week 4)

In Your Pocket

arrays
linked lists
stacks
queues

man ssh exit pwd cd ls
valgrind touch mkdir cp
rm rmdir mv

Sorting Algorithms
- Selection sort

(Slide from Week 5)

In Your Pocket

arrays
linked lists
stacks
queues
(trees)

man ssh exit pwd cd ls
valgrind touch mkdir cp
rm rmdir mv cat head
tail less

Sorting Algorithms
- Selection sort
- Insertion sort
- Merge sort
- Quicksort
- Counting sort

(Slide from Week 6 and 7)

In Your Pocket

arrays
linked lists
stacks
queues
trees
heaps

man ssh exit pwd cd ls
valgrind touch mkdir cp rm
rmdir mv cat head tail less
redirect (>, >>, <) pipe (|)
(echo, sort, uniq, wc)

Sorting Algorithms
- Selection sort
- Insertion sort
- Merge sort
- Quicksort
- Counting sort

(Slide from Week 8)

In Your Pocket

arrays
linked lists
stacks
queues
trees
heaps
hash tables

man ssh exit pwd cd ls valgrind touch mkdir cp rm rmdir mv cat head tail less redirect (>, >>, <) pipe (|) (echo, sort, uniq, wc) diff grep clear

Sorting Algorithms
- Selection sort
- Insertion sort
- Merge sort
- Quicksort
- Heapsort
- Counting sort

(Slide from Week 9)

In Your Pocket

arrays
linked lists
stacks
queues
trees
heaps
hash tables
graphs

man ssh exit pwd cd ls
valgrind touch mkdir cp
rm rmdir mv cat head tail
less redirect (>, >>, <)
pipe (|) (echo, sort, uniq,
wc) diff grep clear
clang++ valgrind make

Sorting Algorithms
- Selection sort
- Insertion sort
- Merge sort
- Quicksort
- Heapsort
- Counting sort

(Slide from Week 10)

We have actually learned more.

In Your Pocket

arrays
linked lists
stacks
queues
trees
heaps
hash tables
graphs

man ssh exit pwd cd ls
valgrind touch mkdir cp
rm rmdir mv cat head tail
less redirect (>, >>, <)
pipe (|) (echo, sort, uniq,
wc) diff grep clear
clang++ valgrind make

Sorting Algorithms
- Selection sort
- Insertion sort
- Merge sort
- Quicksort
- Heapsort
- Counting sort

(Slide from Week 10)

In Your Pocket

arrays
linked lists
stacks
queues
trees
heaps
hash tables
graphs

man ssh exit pwd cd ls
valgrind touch mkdir cp
rm rmdir mv cat head tail
less redirect (>, >>, <)
pipe (|) (echo, sort, uniq,
wc) diff grep clear
clang++ valgrind make

Sorting Algorithms
- Selection sort
- Insertion sort
- Merge sort
- Quicksort
- Heapsort
- Counting sort

ring buffer
deque
binary search
splay tree
priority queue
BFS, DFS
Dijkstra's algorithm
exception
project planning
unit testing
memory management
time and space complexity analysis
recursion
(C++) templates
(C++) const correctness
(C++) STL
(C++) operator overloading

(Slide from Week 1)

An engineer is to solve a problem by selecting the best solution from a set of feasible solutions.

(Slide from Week 1)

To become a better engineer,

- Acquire knowledges and skills to be able to select a solution.
- Increase the size of the potential solution set.

=> We want to have a lots of experiences!

In Your Pocket

arrays
linked lists
stacks
queues
trees
heaps
hash tables
graphs

man ssh exit pwd cd ls
valgrind touch mkdir cp
rm rmdir mv cat head tail
less redirect (>, >>, <)
pipe (|) (echo, sort, uniq,
wc) diff grep clear
clang++ valgrind make

Sorting Algorithms
- Selection sort
- Insertion sort
- Merge sort
- Quicksort
- Heapsort
- Counting sort

ring buffer
deque
binary search
splay tree
priority queue
BFS, DFS
Dijkstra's algorithm
exception
project planning
unit testing
memory management
time and space complexity analysis
recursion
(C++) templates
(C++) const correctness
(C++) STL
(C++) operator overloading

Your potential solution set

arrays
linked lists
stacks
queues
trees
heaps
hash tables
graphs

man ssh exit pwd cd ls valgrind touch mkdir cp rm rmdir mv cat head tail less redirect (>, >>, <) pipe (|) (echo, sort, uniq, wc) diff grep clear clang++ valgrind make

Sorting Algorithms
- Selection sort
- Insertion sort
- Merge sort
- Quicksort
- Heapsort
- Counting sort

ring buffer
deque
binary search
splay tree
priority queue
BFS, DFS
Dijkstra's algorithm
exception
project planning
unit testing
memory management
time and space complexity analysis
recursion
(C++) templates
(C++) const correctness
(C++) STL
(C++) operator overloading

(Slide from Week 1)

By the end of this semester, try finding an answer to the following question.

What do **Data Structures** mean to you?

Were you able to find your own answer to the question?

"The first story is about connecting the dots.

...

Of course it was impossible to connect the dots looking forward when I was in college. But it was very, very clear looking backward 10 years later.

Again, you can’t connect the dots looking forward; you can only connect them looking backward."

by Steve Jobs
(2005 Stanford Commencement Address)
https://news.stanford.edu/2005/06/14/jobs-061505/

This course was designed, so you can have many experiences (dots) in class, instead of memorizing C++ code. The next step is your turn to connect the dots.

Please try skimming (reading) the relevant chapters of any textbooks, so that you can connect the experiences that you had in this course using concrete definitions.

Let's thank **Matt**, our grad TA,
for his hard work through the semester!

Let me thank **you all** for your participation!

Teach Yourself and Your Classmates Presentations

To the (last) lab!