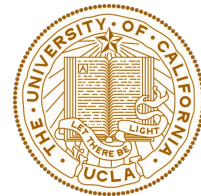




Samueli
Computer Science



CS32: Introduction to Computer Science II

Discussion Week 2

Yichao (Joey)

Jan. 17, 2019

- Course website: <http://web.cs.ucla.edu/classes/winter20/cs32/>
- CS32 Discussion 2E
 - Time and place: 2:00-3:50pm, Friday @Boelter Hall 5422
 - Discussion sessions on week 2-10
- TA: Yichao(Joey) Zhou
 - Email: yz@cs.ucla.edu
 - Office Hours: Wednesdays 9:30am-10:30am & 4:30pm-6:30pm @BH3256S
- LA: Matthew Ko
 - Email: mk9917@ucla.edu
 - Office Hours: Mondays 2-3 PM @BH3256S
- Discussion Website: <https://yz-joeey.github.io/teaching/>
(Slides will be posted here!)

- I am a third-year PhD candidate at ScAi Data Mining Lab, Department of Computer Science, working with [Prof. Wei Wang](#) and [Prof. Kai-Wei Chang](#).
- Other courses I taught: CS145 (Intro to data mining)
CS146 (Intro to machine learning), CS30
- Research Interests:
 - Natural Language Processing
 - Language Understanding
 - Data Mining
 - Information Extraction, Text Mining
 - Healthcare + AI

More about my research

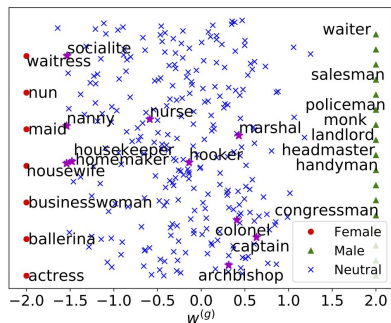
→ Natural Language Understanding

Pun Recognition and Generation

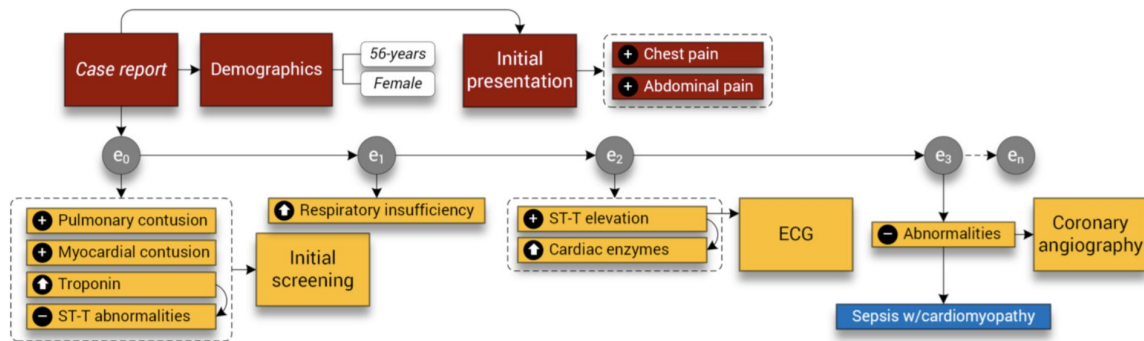
Homographic Puns
1. Did you hear about the guy whose whole left side was cut off? He's all right now.
2. I'd tell you a chemistry joke but I know I wouldn't get a reaction .
Heterographic Puns
1. The boating store had its best sail (sale) ever.
2. I lift weights only on Saturday and Sunday because Monday to Friday are weak (week) days.

Table 1: Examples of homographic and heterographic puns.

Gender-neutral Embedding



→ Health Care + AI



- Start your research experiences in AI/Data mining/NLP areas ***EARLY!***
- Ask yourself:
 - Your background on programming (python preferred), math & statistics (linear algebra, probability, calculus) and machine learning/data mining knowledge
 - Your past project/research experiences
 - How many hours per week (on average) can you contribute to research? Balance your coursework and other activities.

What you will learn in this course...

More about C++ and data structures. Well, that's a lot. And very important.

- Review of C++
- **Object-oriented programming:** Data abstraction, Inheritance and Polymorphism, Recursion, ...
- **Data structures:** Arrays, Lists, Trees, Graphs, Hash tables, ...
- **Algorithms** (eg. sorting) **and complexity analysis**

CS32 is the most beneficial course for your future job interview if you apply for Software Development Engineers (SDE).

- Part 1: Review lectures
 - Review some topics, questions and examples based on last week's lectures
 - Some extended topics from lectures
 - Some typical questions & examples
 - Homework & projects
- Part 2: Exercise
 - Hands-on exercise questions (from worksheets) in groups

Tell me what you'd like to do in the discussion!

- If you have conceptual questions or problems about lectures, ...
- If you find some problems (not limited in worksheets) are particularly difficult for you, ...
- Then, write emails about it to me with **keyword "CS32-DisQ"** !

- Check course website: <http://web.cs.ucla.edu/classes/winter20/cs32/>
- Homework 1 is due on **Tuesday, January 21**.
- Project 2 is due on Tuesday, January 28.
- Midterm 1 is scheduled on Thursday, January 30.

Important: Special notes about homework and projects!

1. Check projects & homework requirements first! You may get 0 if you do not follow these guidelines and restrictions!
2. When debugging, check your error message! Search Google or Stackoverflow for similar error and find possible reasons.
3. Do **NOT** let you TAs or LAs debug through emails (almost impossible actually)!
4. Others...



- Dynamic memory allocation in C++
- Constructor, destructor, copy constructors, and assignment operators
- Homework 1
- Data structure: Arrays & Linked Lists (probably just start)

- Let's first compare with *Static Memory Allocation*!
- If we want to type in a paragraph and save it into a C-string,

```
#define MAXLENGTH 10000  
char s[MAXLENGTH+1];  
cin.getline(s);
```

- What if the paragraph is extremely long? → Out-of-bound
- What if the paragraph has only five words? → Overallocated Memory (Waste)

Dynamic memory allocation

new in C++

What if we want to fit the paragraph into a C-string with right the sufficient size?

```
<type> *<name> = new <type>[<#elements>];  
char *article = new char[length];
```

Unsigned int variable

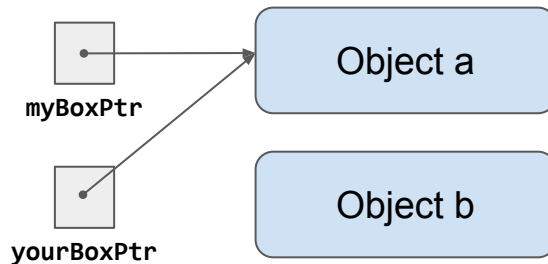
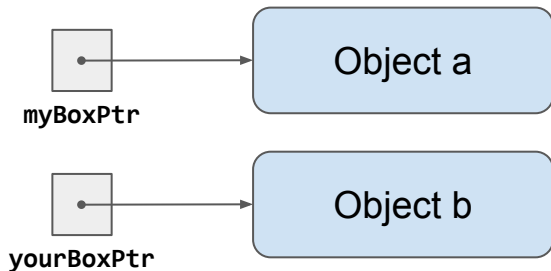
new will dynamically allocate the sequential memory space for the requested data type and size and return the starting address of the allocated memory space.

Variables allocated with **new** will remain in the memory unless we manually delete it which means dynamic allocation has to be deleted once we no longer use it.

```
delete [] article
```

Dynamic memory allocation

Memory Leak



```
// Create first object
MagficBox* myBoxPtr = new MagficBox

// Create second object
MagficBox* yourBoxPtr = new MagficBox

// Assignment causes an inaccessible object (b)
yourBoxPtr = myBoxPtr
```

Dynamic memory allocation

A Problematic Code (1)

```
#include <iostream>
#include <string>
using namespace std;

class Node{
public:
    Node(int id){
        cout << "Constructor:" << id << endl;
        this->id = id;
    }
    ~Node(){
        cout << "Destructor:" << this->id << endl;
    }
private:
    int id;
};

int main(){
    Node node1(31);
    Node* node2 = new Node(32);
    // do something
    return 0;
}
```

Q: What is the output of this simple “node” program?

Constructor:31
Constructor:32
Destructor:31

Node(32) does not deconstruct!

Dynamic memory allocation

A Problematic Code (2)

Let's check it by [Valgrind](#)!

```
#include <iostream>
#include <string>
using namespace std;

class Node{
public:
    Node(int id){
        cout << "Constructor:" << id << endl;
        this->id = id;
    }
    ~Node(){
        cout << "Destructor:" << this->id << endl;
    }
private:
    int id;
};

int main(){
    Node node1(31);
    Node* node2 = new Node(32);
    // do something
    return 0;
}
```

```
jeffhao@jeffhao:~/Desktop/CS32_example/memleak$ valgrind --leak-check=yes ./node_prog
==10005== Memcheck, a memory error detector
==10005== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==10005== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==10005== Command: ./node_prog
==10005==
Constructor:31
Constructor:32
Destructor:31
==10005==
==10005== HEAP SUMMARY:
==10005==    in use at exit: 72,708 bytes in 2 blocks
==10005==    total heap usage: 3 allocs, 1 frees, 73,732 bytes allocated
==10005==
==10005== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==10005==    at 0x4C2E0EF: operator new(unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64
-linux.so)
==10005==    by 0x400ACA: main (in /home/jeffhao/Desktop/CS32_example/memleak/node_prog)
==10005==
==10005== LEAK SUMMARY:
==10005==    definitely lost: 4 bytes in 1 blocks
==10005==    indirectly lost: 0 bytes in 0 blocks
==10005==    possibly lost: 0 bytes in 0 blocks
==10005==    still reachable: 72,704 bytes in 1 blocks
==10005==    suppressed: 0 bytes in 0 blocks
==10005==
==10005== Reachable blocks (those to which a pointer was found) are not shown.
==10005== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==10005==
==10005== For counts of detected and suppressed errors, rerun with: -v
==10005== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Dynamic memory allocation

Good memory example from textbook

```
/** @file GoodMemory.cpp */
# include "GoodMemory.h"

GoodMemory::GoodMemory() : someBoxPtr(nullptr)
{
} // end default constructor

GoodMemory::~~GoodMemory()
{
    delete someBoxPtr;
} // end constructor

void GoodMemory::unleakyMethod(const double& someItem)
{
    someBoxPtr = new ToyBox<double>();
    someBoxPtr->setItem(someItem)
} // end unleakyMethod
```


Construction & Destruction

Class Composition

Class composition is when a class contains one or more member variables that are objects.

Order of construction:

- Member variables are constructed in order.
- Then the current class constructor is executed.

Order of destruction:

- The current class destructor is executed first.
- The member variables are destructed in the reverse order.

Summary:

- Constructor: Inside -> Outside (In-order)
- Destructor: Outside -> Inside (Reverse Order)

Question: For classes containing members of a class type, what is the order of construction and destruction?

What is the output?

Answer:

A(444)

A(888)

B()

~B()

~A(888)

~A(444)

```
#include <iostream>
#include <string>
using namespace std;

class A
{
public:
    A(){cout << "A()" << endl;}
    A(int x){cout << "A(" << x << ")" << endl; this->id = x;}
    ~A(){cout << "~A(" << this->id << ")" << endl;}
private:
    int id;
};

class B
{
public:
    B():a1(888),a2(444){cout << "B()" << endl;}
    ~B(){cout << "~B()" << endl;}
private:
    A a2;
    A a1;
};

int main()
{
    B b;
    return 0;
}
```

Construction & Destruction

Note: There is a difference between class composition and class inheritance. → Will be explained in later lectures.

```
#include <iostream>
#include <string>
using namespace std;

class A
{
public:
    A(){cout << "A()" << endl;}
    A(int x){cout << "A(" << x << ")" << endl;}
    ~A(){cout << "~A()" << endl;}
};

class B
{
public:
    B():a1(1),a2(2){cout << "B()" << endl;}
    ~B(){cout << "~B()" << endl;}
private:
    A a2;
    A a1;
};

int main()
{
    B b;
    return 0;
}
```

```
#include <iostream>
#include <string>
using namespace std;

class A
{
public:
    A(){cout << "A()" << endl;}
    A(int x){cout << "A(" << x << ")" << endl;}
    ~A(){cout << "~A()" << endl;}
};

class B : public A
{
public:
    B():a1(1),a2(2){cout << "B()" << endl;}
    ~B(){cout << "~B()" << endl;}
private:
    A a2;
    A a1;
};

int main()
{
    B b;
    return 0;
}
```

Copy Constructors

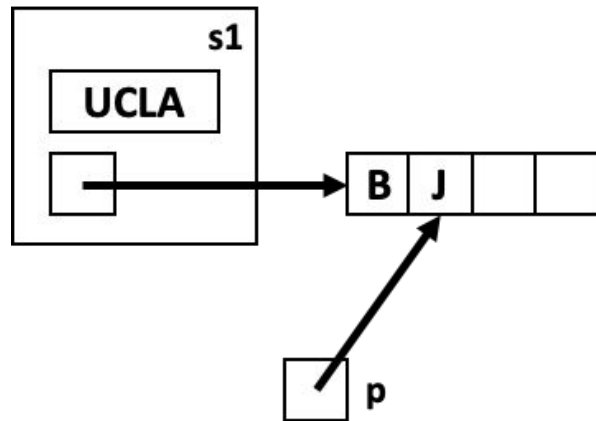
Motivation

Consider an example of UCLA and its students:

```
Student st1("Brian");  
Student st2("John");  
School s1("UCLA");  
s1.addStudent(st1);  
s1.addStudent(st2);  
Student *p = s1.getStudent("John");
```

We want to create a new School called s2, with exactly the same content as s1. In other words, we want to clone s1.

```
School s2(""); s2=s1; // Is this correct?
```



Copy Constructors

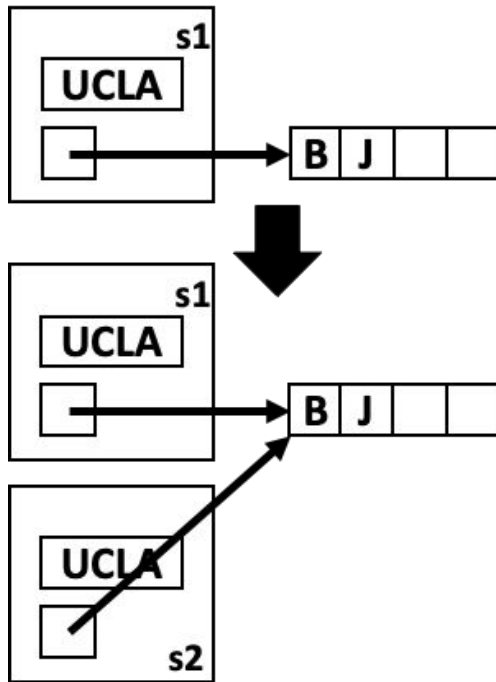
Shallow copy problem

We want to create a new School called s2, with exactly the same content as s1. In other words, we want to clone s1.

```
School s2(""); s2=s1; // Definitely not!
```

What if grab values out of s1 and manually copy them into s2?

```
School s2("");  
s2.setName(s1.getName( ));  
... // copy all members and properties
```



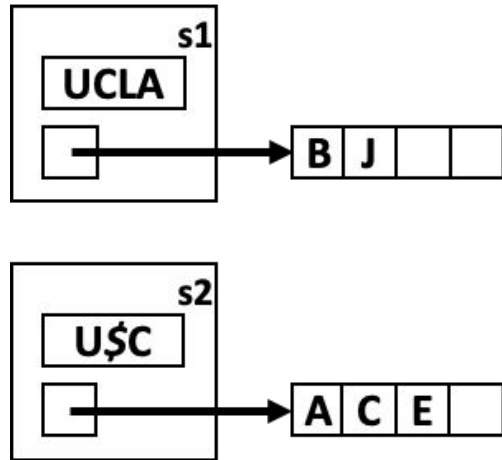
Copy Constructors

Example

Immutable. We don't change the object we're copying from.

```
School::School(const School &aSchool){  
    m_name = aSchool.m_name;  
    m_numStudents = aSchool.m_numStudents;  
    m_students = new Students[m_numStudents];  
    for (int i = 0; i < m_numStudents; ++i)  
        m_students[i] = aSchool.m_students[i];  
}
```

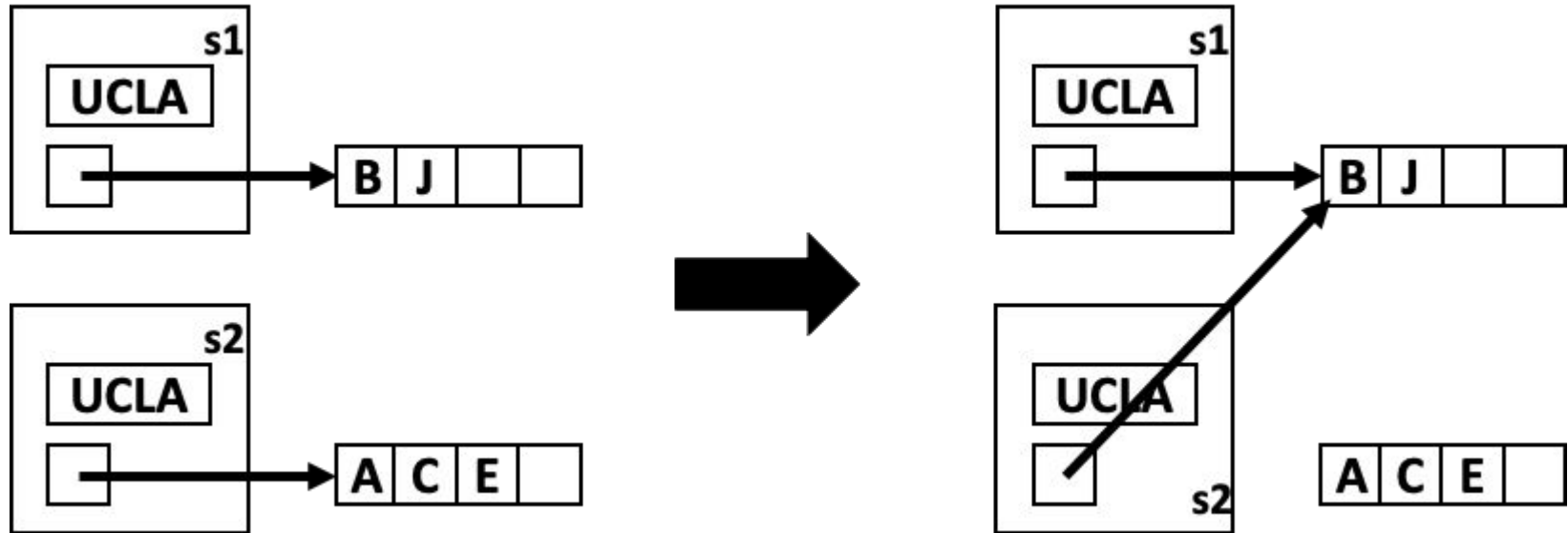
Now you can do: `School s2(s1);`



Assignment Operator

Motivation

What if $s2 = s1$?



Assignment Operator

Example

Assignment operator

- Overload the operator (in this case, we overload the assignment operator).

```
String& String::operator=(const String& rhs)
{
    delete [] m_text;
    m_len = rhs.m_len;
    m_text = new char[m_len+1];
    strcpy(m_text, rhs.m_text);
    return *this;
}
```

*Do not forget *this!*

Assignment Operator

Example

Assignment operator

- Overload the operator (in this case, we overload the assignment operator).

```
String& String::operator=(const String& rhs)
{
    delete [] m_text;
    m_len = rhs.m_len;
    m_text = new char[m_len+1];
    strcpy(m_text, rhs.m_text);
    return *this;
}
```

What if it's a self-assignment?

String sa[100];

...
sa[i] = sa[j]; // i == j

Assignment operator

- Overload the operator (in this case, we overloaded the assignment operator).

```
String& String::operator=(const String& rhs)
{
    delete [] m_text;
    m_len = rhs.m_len;
    m_text = new char[m_len+1];
    strcpy(m_text, rhs.m_text);
    return *this;
}
```

* Then `sa[i] = sa[j]`, which the compiler turns into `sa[i].operator=(sa[j])`, calls `String::operator=` with the `_this_` pointer pointing to the same string object that `rhs` is a reference to.

* So the `delete [] text;` gets rid of the dynamic array with the text of the string,

* So `rhs.m_text` is a dangling pointer, yielding undefined behavior for `strcpy`.

Assignment Operator

Example

Make it better:

```
String& String::operator=(const String& rhs)
{
    if (this != &rhs)
    {
        delete [] m_text;
        m_len = rhs.m_len;
        m_text = new char[m_len+1];
        strcpy(m_text, rhs.m_text);
    }
    return *this;
}
```

Since the goal of assignment is to make the left hand side object look like a copy of the right hand side object, if the two objects are the **same object**, the goal is immediately met, so the classic solution is to check for that:

Assignment Operator

Example

The classic way of implementing an assignment operator is problematic, and we don't like the duplication of code between the copy constructor and assignment operator, so try to have the assignment operator call the copy constructor.

One modern approach is the copy-and-swap idiom.

We give our class a `swp` function that swaps the values of two Strings by individually swapping each data member:

```
void String::swp(String& other)
{
    ... // exchange the m_len and other.m_len ints
    ... // exchange the m_text and other.m_text pointers
}
```

Assignment Operator

Example

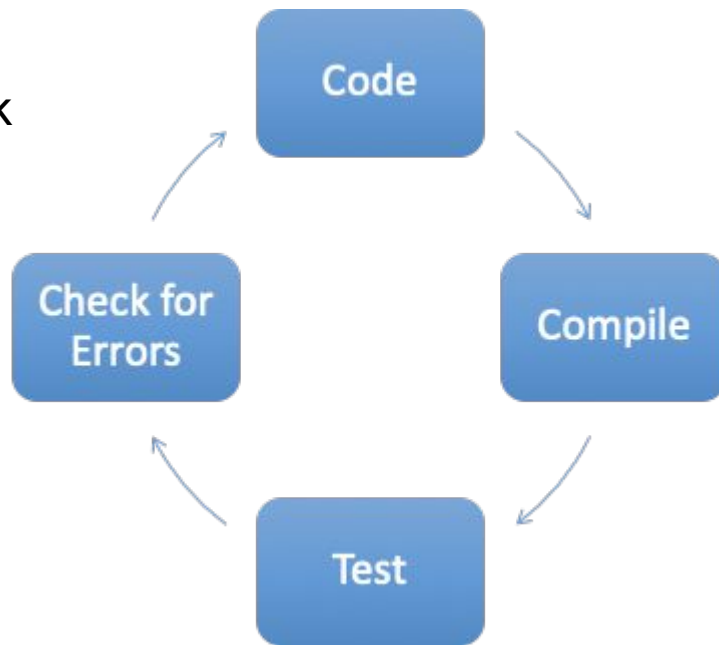
It is essential that this function does **not** instead just swap the two Strings using String's assignment operator; we **don't** do `String temp(other); other = *this; *this = temp.`

```
String& String::operator=(const String& rhs)
{
    if (this != &rhs)
    {
        String temp(rhs);
        swap(temp);
    }
    return *this;
}
```

Elegant!

Creating temp calls the copy constructor, swap installs the copy of rhs in *this and puts the old value of *this into temp, and when temp goes out of scope, the destructor gets rid of the old value.

- Pointers → Memory layout
- Assert
- Debug → Breakpoint, trace, variables, stack
- Pseudocode
- Others...



Before we talk about doing things arrays, let's talk about data structure first!

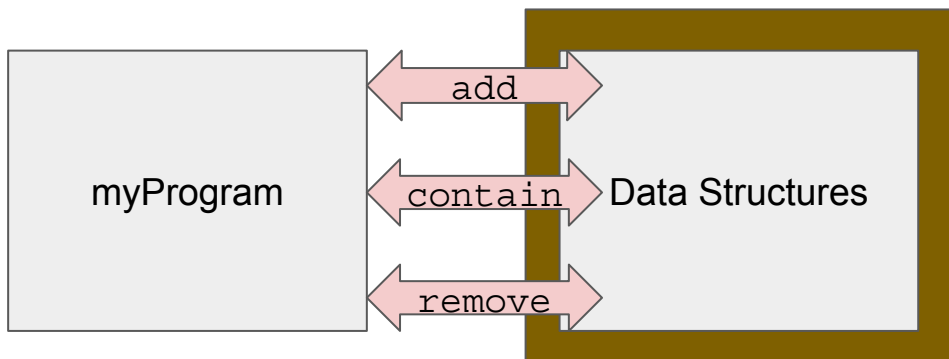
CS32 is not just to write code and run. It is more about organizing data. We call an organization scheme a data structure. For every structure, we define:

- How to store/organize the data items?
- Method to add new data / remove data
- Apply functions on data: most importantly, how to search data?

We also need to know pros and cons of each data structure as well as its efficiency or complexity of algorithms with these data structures.

Arrays (as an example)

- Most basic data structure and most important!



- Add / Contain / Remove → Complexity?
- More implementations: `getIndexOf`, `getFrequencyOf`, ...
- More questions: Using array to implement Linked List? How to sort a array or maintain a sorted array?

- const function: `void/bool/int F() const;`
- `bool get(const std::string& key, double& value);`
- `bool get(int i, std::string& key, double& value);`
- define a **structure type (either struct or class)** containing a member of type KeyType and a member of type ValueType. Have Map's array data member be an array of these structures.
- If you declare another structure to help you implement a Map, put its declaration in Map.h, declare that helper structure in the private section of Map.
- `Map();`

- I am a third year Computer Science student
- Last quarter, I was an LA for CS31
- CS classes I've taken:
 - CS 111, 131, 180, 118
 - CS 31-33
 - CS35L
- Clubs I'm involved in:
 - Daily Bruin
 - DevX
- I have an Instagram page for my fish @boyandhisfish



- Exercise problems from **Worksheet 1** (see “LA worksheet” tab in CS32 website). Answers will be posted after all discussions.
- Our LA will help you go through these problems.
- Worksheet is optional. We encourage you to solve the problem without working on a computer.

<https://tinyurl.com/cs32w20wk2>

Hints for #2

Constructor

- Remember, `b` and `arr` are dynamically allocated!!!

Copy Constructor

- We know that `b` has a default constructor, a copy constructor, and an assignment operator. Which one of these would be helpful?

Assignment Operator

- What happens when we assign an object to itself?
- Did you remember to check for memory leaks?

Hints for #5

- Destruction happens in the reverse order of construction
 - An object's member objects must be constructed first
 - An object's destructor must be called before destructing its objects