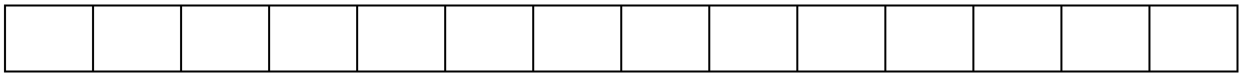
April 6th Workshop Summary

Data Structures

Introduction

- Data Structures are essentially objects that organizes your data
- They are like containers for a lot of variables, usually of the same type

1D Arrays



You can imagine an array to look like the above ^
Arrays are a sequence of data that is:

1. Of the same type (you cannot store a boolean along with an integer in the same array)
2. Of fixed length (once you defined the length of an array, you cannot modify it)
3. With indexed elements (every element has a position in the array)

Create an array:

```
<type> <name-of-array>[] = new <type>[<length>];
```

E.g.

```
int myIntegerArray[] = new int[12];  // this will create an integer array of length 12
String myStringArray[] = new String[22]; // creates a String array of length 22
```

Access the 'ith' element in the array:

```
<name-of-array>[i];
```

E.g.

```
int arrayElement = myIntegerArray[0];
// accesses the first element in the array and assigns it to a variable called arrayElement
```

```
myStringArray[4] = "Annie";
// accesses the 5th element and assigns this element the String "Annie" (visualization below)
```

					"Annie"			
0	1	2	3	4	5	6	7	8…

Note: Arrays start with index 0

Getting the length of the array:

`<name-of-array>.length;`

E.g.

```
int lengthOfMyArray = myIntegerArray.length; //gets the length of myIntegerArray
```

Iterating an array:

By iterating, we mean trying to access every single element in the array.
We know, from above, that in order to access every single element, we can do:

`myIntegerArray[0]` - to access the first element in the array
`myIntegerArray[1]` - to access the second element in the array
…

However what if we have an array of length 100? We cannot just type that 100 times.

Since we want to repeatedly go to every single element, we can use a for-loop.
Below is a quick review.example of for loop:

```
for (int i = 0; i < 5; i++){
      System.out.print(i);
}
```

The code above will print out "01234".
Since "i" changes from 0 to 1 to 2 to 3 … we can also use "i" to access every single element of the array, just like doing myIntegerArray[0], myIntegerArray[1]... etc.

Here's the code to access every single element:
e.g.

```
for (int i = 0; i < myIntegerArray.length; i++){
      myIntegerArray[i] = 3;
}
```

We have assigned every single element of the array to equal 3.
Note: if you try to access something outside of the bounds of the array (for example myIntegerArray has length of 12, but we want to access `myIntegerArray[12]`, which is the 13th element of the array), we will get an exception!!!

HashSets

HashSet is a data structure that has the following properties:

1. Unlike an array, you can have a HashSet of any size; which means you can remove or add as many items as you like.
2. You **cannot** have duplicate data in a HashSet.
3. HashSets are not indexed or ordered, so you **cannot** access an element directly

Create a HashSet:

```
HashSet<TYPE> <name-of-hashset> = new HashSet<>();
```

E.g.

```
HashSet<Integer> myFirstIntegerSet = new HashSet<>();
```
//creates a set of type ‘int’

note: we need to put the primitive types into a ‘wrapper class’ to create a HashSet of that type. The wrapper classes are:

```
char -> Character, String -> String, double -> Double,
boolean -> Boolean
```

Add an element in HashSet:

```
<name-of-hashset>.add(<element>);
```

E.g.

`myFirstIntegerSet.add(1);` //adds the number ‘1’ to the set

Remove an element in HashSet:

```
<name-of-hashset>.remove(<element>);
```

E.g.

`myFirstIntegerSet.remove(1);` // removes the number ‘1’ from the set

See if element exists in HashSet:

```
<name-of-hashset>.contains(<element>);
```

E.g.

`myFirstIntegerSet.contains(1);` // returns false - we removed 1 from our set
`myFirstIntegerSet.add(1);`
`myFirstIntegerSet.contains(1);` // returns true - we just added 1 to our set

Find the size of the HashSet:

```
<name-of-hashset>.size();
```

E.g.

```
myFirstIntegerSet.size();  //returns 1, since we have a ‘1’ in our set
myFirstIntegerSet.add(1);
//because a set CANNOT have duplicates, we can’t actually add ‘1’
myFirstIntegerSet.size();  //still returns 1, since the previous ‘1’ wasn’t added
myFirstIntegerSet.add(123);
myFirstIntegerSet.size();  //returns 2, since now we have 1 and 123
myFirstIntegerSet.remove(1);
myFirstIntegerSet.size(); //returns 1, since now we only have 123, as we removed 1.
```

Iterating through a HashSet (optional):

This part is optional, but here’s an example of how you would iterate through a set:

```
for(int myInteger : myFirstIntegerSet){
      <statements to execute>
}
```

You can read the above code as: “for every integer, name it ‘myInteger’, in myFirstIntegerSet, we want to execute the “statements to execute”.

Feedback

We would really appreciate it if you fill out a feedback survey on this workshop! That way we’ll know how to improve our future workshops :)
https://pearl64.typeform.com/to/kdB1L1

Additional Resources

Github link: https://github.com/CSFoundations
We’ll be posting workshop summaries, solutions to our workshop problems under this link.
Slack link:
https://join.slack.com/t/csfoundations/shared_invite/enQtMzM1Nzg1NjE5MjgzLTM0NTMwOGE3YmUwOWZjMDFiOGIzYTY5NzViNTgxNjQ5OTg2MmU5Njk4OTkxOGQ1NTcxMGQyNWQzYzBjMTg4MWE
If you have any questions you can slack any one of us and we’ll answer ASAP :)