Worksheet - M0 Fundamentals
CS 315: Data Structures and Algorithms
Jeremiah Webb ID: 2545328

1 Introduction

1.1 What is a data structure?

A **data structure** is *an organization of units of data within a computing system*. They enable access and modification of the stored data.

They are **composed of**:

- **Values** to be stored;
- **Relationship** between values; and
- **Algorithms** that enable:
    - Insertion,
    - Deletion,
    - Retrieval, and
    - Modification of the data.

1.2 What is an algorithm?

An **algorithm** is a *well-defined sequence of operations that when executed can solve a problem.*

For example, in mathematics, you have learned algorithms to determine the mean value out of a sequence of numbers.

Q: What is that algorithm?

In computing, *our algorithms represent a sequence of software commands that when executed likewise solve a problem such as data storage/retrieval, processing, organization, etc.*

Our programs include algorithms. For instance, data structures require a variety of algorithms, which we mentioned earlier.

Another example of an algorithm is a **search algorithm**. Given a problem declaration, an agent's state, goal criteria, and a set of actions that the agent can make, an algorithm can find an optimal or near optimal solution path.

A variety of algorithms may be applicable to a problem.
We will see that while an algorithm might be able to solve a problem it **might not be the best solution**. We will start learning about **algorithm analysis** techniques in this module.

Many algorithms are **language independent**. They can be **tailored** to the particulars of the implementing language. For example, binary search on a sorted array is an algorithm follows the same steps to achieve the goal if implemented in Java or C++.

In addition to data structures and algorithms, **this class is about abstraction**. The data structures and algorithms may seem generic but can be applied to a variety of problems.

1.3 Why study data structures and algorithms?

Data structures and algorithms are **essential tools in the toolbox of computer scientists**, software engineers, computer engineers, and any other profession in which software must be written (many).

We will be studying both **basic and advanced** data structures within the course.

This class is **taught in Java**, but you can apply these algorithms to other languages and domains.

Our **textbook** demonstrates these tools by **first showing Java's built-in functions and then unpacking the details.**

We will use the same approach. First, we will learn what the data structure and/or algorithm can achieve, and then unwrap how the data is organized and how the algorithms utilize that organization.

1.4 About this Worksheet

Each module will give you an opportunity to practice the concepts on paper. This includes writing pseucode, interpreting code snippets, drawing pictures depicting the state of data structure, etc.

For this first module, the worksheet problems will primarily cover review topics. There are fewer problems than typical for a worksheet to permit everyone to orient themselves with the course its online modality.

You may type your answers or you can digitize it in some way to an image file or PDF via scanner or mobile phone camera.

Note: Microsoft Lens works quite well for generating PDFs from a number of images scanned by your cell phone and includes some common filters to sharpen and flatten your image. Just a hint.

1. **Given the following statements, indicate the data type and value returned for each expression (see Section 1.1 of textbook's discussion of type conversion).**

   a. (1 + 3.25)/2

      Double ; 2.125

   b. 3 % 2

      Integer ; 1

   c. 4.1 > 4

      Boolean ; True

   d. 1 + 2 + 3.0 + 4

   Double ; 10.0

2. **Implement in pseudocode a Java method that returns true if an input integer, val, is even; otherwise, false.**

   If the modolous of val and 2 equals 0, return True, else False.

3. **Trace the execution of the non-recursive BinarySearch algorithm implemented in the textbook for the following array of data.**

```
int position = BinarySearch.indexOf({0, 1, 2, 4, 6, 8, 9}, 2)
BS indexOf(a,2)
    lo = 0
    mid = 3
    2<a[3] (4)
        hi = 2
        lo is less than hi
        mid = 1
        2>a[1] (1)
        lo = 1
        lo still less than hi (1<2
        mid = 2
        key == a[2]
        return mid (a[2])
        key & mid value are same, so return mid which is 2.
```

4. **Trace the execution of the RecursiveBinarySearch algorithm implemented by Dr. Stansbury for the following array of data.**

```
int position = RecursiveBinarySearch.indexOf({0, 1, 2, 4, 6, 8, 9}, 2)

BS indexOf(a,2,0,6)
    mid = 3
    2 < a[3]
BS indexOf(a,2,0,2)
    mid = 1
    2 < a[1]
BS indexOf(a,2,2,2)
    mid = 2
    2 = a[2]
```

5. **Given the following API for an abstract data type, Point**

Constructor:

```
Point(double x, double y) //Creates a point at location x,y
```

Methods:

```
String toString();              //Returns string representation of point
Double distanceTo(Point p);  //Returns Euclidean Distance
```

Usage:

```
Point p1 = new Point(0,0);
Point p2 = new Point(1,1);
System.out.println(p1);
System.out.println(p1.distanceTo(p2));
```

Output:

```
Point(0,0)
1.41421356
```

Write out in pseudocode a Java interface to define the API above. Do not implement the class. ONLY define the interface.

The interface IPoint will have two public interface methods, one String method “toString” and a Double method “distanceTo”.

6. **Given the interface *Shape2D* and a sample implementation of the interface, *Rectangle*, as a reference, write out the pseudocode to implement the *Shape2D* interface to define a *Circle* object.**

```
public interface Shape2D {
        public double getPerimeter();
        public double getArea();
}


public class Rectangle implements Shape2D {
        private double length;
        private double width;

        public Rectangle(double length, double width) {
                this.length = length;
                this.width = width;
        }
        public double getPerimeter() {
                return length*2 + width*2;
        }
        public double getArea() {
                return length * width;
        }
}
```

A public class named Circle implements the interface Shape2D.

Initialize one private double variable, radius.

A new public constructor, Circle, will take in a double “radius” variable and set it to the private variable, radius.

A public double method named getPerimeter will be made, it will return 2 times the number pi times the private radius variable.

A public double method named getArea will be made, will return the number pi times the radius squared.

7. **Write a test client to evaluate your implementation of Circle above. A test client should be a Java method that creates an instance(s) of the rectangle object and test its correct function.**
    a. Define a test to evaluate the correctness of getPerimeter()
    b. Define a test to evaluate the correctness of getArea()

```
        public static void circleTestClient(){
                Circle c= new Circle(4.0);
                System.out.println("Circle 1 Perimeter: " + c.getPerimeter()); // Expected:
25.13274
                System.out.println("Circle 1 Area: " + c.getArea()); // Expected: 50.26548

                Circle cc = new Circle(8.0);
                System.out.println("Circle 2 Perimeter: " + cc.getPerimeter()); // Expected:
50.26548
                System.out.println("Circle 2 Area: " + cc.getArea()); //Expected: 201.06193

}
```

For example, I could write a test client for the rectangle class using several known input/output pairs. Once I run my code, I can determine if the results given were as expected.

```
public static void rectangleTestClient() {
        Rectangle r = new Rectangle(2.0, 2.0);
        System.out.println("R1 Perimeter: " + r.getPerimeter()); //Expected 8.0
        System.out.println("R1 Area: " + r.getArea()); //Expected 4.0

        Rectangle r = new Rectangle(2.0, 4.0);
        System.out.println("R2 Perimeter: " + r.getPerimeter()); //Expected 12.0
        System.out.println("R2 Area: " + r.getArea()); //Expected 8.0
}
```

8. Consider the following recursive method

```
public static int mystery(int a, int b) {
            if (b==0) return 0;
            if (b%2 == 0) return mystery(a+a, b/2);
            return mystery(a+a, b/2) + a;
}
```

a. Label (mark code above) the code above identifying the base case(s) and progress case(s).
b == 0 is the base case,
b%2 == 0 is the progress case.

b. Trace the execution with the following inputs:

Base case, return 0 if b is equal to 0.
Progress case, b%2 when equal 0.
Mystery(2, 25);
Mystery(4,12) + 2;
Mystery(8,6)
Mystery(16,3)
Mystery(32, 1) + 16
Mystery(64, 0) + 32
final result = 50

Mystery(3, 11);
Mystery(6,5)+3
Mystery(12,2)+6
Mystery(24,1)
Mystery(48,0) + 24

final result=33