Data Structures and Algorithms (in JAVA)

Course objectives

- Be familiar with different data structures available to represents data
- Be able to trace algorithms and verify correctness.
- Be able to develop and implement algorithms using different data structures
- Be able to select appropriate data structures and algorithms for given problems
- Be able to use JAVA language to implement different algorithms pseudo codes.

Course Outline

- Fundamentals of data structures and algorithms
- Static and dynamic data structures
- Basic searching and sorting algorithms
- Recursion
- Abstract data types
- Stacks and queues
- Trees

Readings/references

- Text Book:
  - **Data Structures & Algorithms in JAVA** (5th Edition), by M. Goodrich & R. Tamassia, John Wiley & Sons, inc., 2010.
- Additional Readings:
  - **Data Structures and Problem Solving with JAVA** (3rd Edition), by Mark Allen Weiss, Addison Wesley, 2006.
  - **Lecture slides and handouts**

What is data?

- **Data**
  - A collection of facts from which conclusion may be drawn
  - e.g. **Data**: Temperature 35°C; **Conclusion**: It is hot.
- Types of data
  - Textual: For example, your name (Muhammad)
  - Numeric: For example, your ID (090254)
  - Audio: For example, your voice
  - Video: For example, your voice and picture
  - (...)

What is data structure?

- A particular way of **storing and organizing data in a computer** so that it can be used efficiently and effectively.
- Data Structures are **the programmatic way of storing data so that data can be used efficiently**.
- Data structure is the logical or mathematical model of a particular organization of data.
- A group of data elements grouped together under one name.
  - For example, an array of integers

- **Data Structure**

is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way.

- Data Structures

is about rendering data elements in terms of some relationship, for better organization and storage.

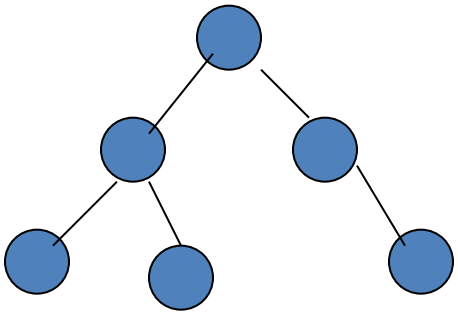
For example, we have some data which has, player's **name** "Virat" and **age** 26. Here "Virat" is of **String** data type and 26 is of **integer** data type.

- We can organize this data as a record like **Player** record, which will have both player's name and age in it.
- Now we can collect and store player's records in a file or database as a data structure. **For example**: "Dhoni" 30, "Gambhir" 31, "Sehwag" 33

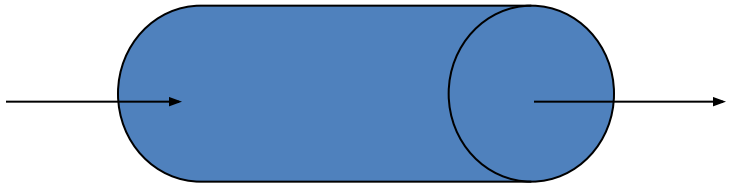
Types of data structures

1-D Array

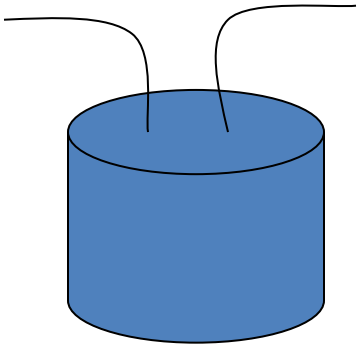
Linked List



Tree



Queue



Stack

There are many, but we named a few. We'll learn these data structures in great detail!

The Need for Data Structures

- Goal: to **organize data**

- Criteria: to facilitate **efficient**
  - **storage** of data
  - **retrieval** of data
  - **manipulation** of data

- Design Issue:
  - **select and design** appropriate data types
    (This is the main motivation to learn and understand data structures)

Data Structure Operations

(Demonstrate using class room example!)

- Traversing
  - Accessing each data element exactly once so that certain items in the data may be processed
- Searching
  - Finding the location of the data element (key) in the structure
- Insertion
  - Adding a new data element to the structure

Data Structure Operations (cont.)

- Deletion
  - Removing a data element from the structure
- Sorting
  - Arrange the data elements in a logical order (ascending/descending)
- Merging
  - Combining data elements from two or more data structures into one

What is algorithm?

- A finite set of instructions which accomplish a particular task
- A method or process to solve a problem
- Transforms input of a problem to output

Algorithm = Input + Process + Output

Algorithm development is an art – it needs practice, practice and only practice!

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.
- Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.
- From the data structure point of view, following are some important categories of algorithms −
- **Search** − Algorithm to search an item in a data structure.
- **Sort** − Algorithm to sort items in a certain order.
- **Insert** − Algorithm to insert item in a data structure.
- **Update** − Algorithm to update an existing item in a data structure.
- **Delete** − Algorithm to delete an existing item from a data structure.

What is a good algorithm?

- It must be correct
- It must be finite (in terms of time and size)
- It must terminate
- It must be unambiguous
  - Which step is next?
- It must be space and time efficient

A program is an instance of an algorithm, written in some specific programming language

A simple algorithm

- Problem: Find maximum of `a, b, c`
- **Algorithm**
  - Input = `a, b, c`
  - Output = `max`
  - Process
    - `Let max = a`
    - `If b > max then`
      `max = b`
    - `If c > max then`
      `max = c`
    - `Display max`

Order is very important!!!

Algorithm development: Basics

- Clearly identify:
  - what output is required?
  - what is the input?
  - What steps are required to transform input into output
    - The most crucial bit
    - Needs problem solving skills
    - A problem can be solved in many different ways
    - Which solution, amongst the different possible solutions is optimal?

How to express an algorithm?

- A sequence of steps to solve a problem
- We need a way to express this sequence of steps
  - Natural language (NL) is an obvious choice, but not a good choice. Why?
    - NLs are notoriously ambiguous (unclear)
  - Programming language (PL) is another choice, but again not a good choice. Why?
    - Algorithm should be PL independent
  - We need some balance
    - We need PL independence
    - We need clarity
    - Pseudo-code provides the right balance

What is pseudo-code?

- **Pseudo-code** is a short hand way of describing a computer program
- Rather than using the specific syntax of a computer language, more general wording is used
- It is a mixture of NL and PL expressions, in a systematic way
- Using pseudo-code, it is easier for a non-programmer to understand the general workings of the program

Pseudo-code: general guidelines

- Use PLs construct that are consistent with modern high level languages, e.g. C++, Java, ...
- Use appropriate comments for clarity
- Be simple and precise

Components of Pseudo-code

❑ Expressions

- Standard mathematical symbols are used
  - Left arrow sign (←) as the assignment operator in assignment statements (equivalent to the = operator in Java)
  - Equal sign **(=)** as the equality relation in Boolean expressions (equivalent to the "= =" relation in Java)
  - For example

    Sum ← 0

    Sum ← Sum + 5

  What is the final value of sum?

Components of Pseudo-code (cont.)

- **Decision structures** (if-then-else logic)
  - **`if`** condition **`then`** true-actions [**`else`** false-actions]
  - We use indentation to indicate what actions should be included in the true-actions and false-actions
  - For example

```
if marks > 50 then
print "Congratulation, you are passed!"
else
print "Sorry, you are failed!"
end if
```

What will be the output if marks are equal to 75?

Components of Pseudo-code (cont.)

- Loops (Repetition)
  - **Pre-condition loops**
    - While loops
      - **while** condition **do** actions
      - We use indentation to indicate what actions should be included in the loop actions
      - For example

```
while counter < 5 do
    print "Welcome to CS204!"
    counter ← counter + 1
end while
```

What will be the output if counter is initialised to 0, 7?

Components of Pseudo-code (cont.)

- Loops (Repetition)
  - **Pre-condition loops**
    - For loops
      - **for** variable-increment-definition **do** actions
      - For example

```
for counter ← 0; counter < 5; counter ← counter + 2 do
    print "Welcome to CS204!"
end for
```

What will be the output?

Components of Pseudo-code (cont.)

- Loops (Repetition)
  - **Post-condition loops**
    - Do loops
      - **`do`** actions **`while`** condition
      - For example

```
do
    print "Welcome to CS204!"
    counter ← counter + 1
while counter < 5
```

What will be the output, if counter was initialised to 10?

The body of a post-condition loop must execute at least once

Components of Pseudo-code (cont.)

- **Method declarations**
  - Return_type method_name (parameter_list) method_body
  - For example

```
integer sum ( integer num1, integer num2)
start
        result ← num1 + num2
end
```

- **Method calls**
  - object.method (args)
  - For example

```
mycalculator.sum(num1, num2)
```

Components of Pseudo-code (cont.)

- **Method returns**
  - **`return`** value
  - For example

```
integer sum ( integer num1, integer num2)
start
        result ← num1 + num2
        return result
end
```

Components of Pseudo-code (cont.)

- Comments
  - /* Multiple line comments go here. */
  - // Single line comments go here
  - Some people prefer braces {}, for comments
- Arrays
  - *A[i]* represents the *i*th cell in the array *A*.
  - The cells of an *n*-celled array *A* are indexed from $A[0]$ to $A[n-1]$ (consistent with Java).

Algorithm Design: Practice

- Example : Determining even/odd number
  - A number divisible by 2 is considered an even number, while a number which is not divisible by 2 is considered an odd number. Write pseudo-code to display first N odd/even numbers.

Even/ Odd Numbers

```
Input range
for num←0; num<=range; num←num+1 do
  if num % 2 = 0 then
    print num is even
  else
    print num is odd
  endif
endfor
```

Homework

1. **Write an algorithm to find the largest of a set of 10 numbers.**
2. **Write an algorithm in pseudocode that finds the average of (10) numbers.**

1. **Write an algorithm to find the largest of a set of 10 numbers.**

```
Input: 10 positive integers
Output: Max integer
Process:
Range=10;
Max 0;
Counter 1;
for counter←0;counter<=range; counter←counter+1 do
   if  integer>=  max then
      max=integer;
      endif
Endfor
Return max;
```

```
FindLargest
Input: 1000 positive integers
1.  Set Largest to 0
2.  Set Counter to 0
3.  while (Counter less than 1000)
     3.1  if (the integer is greater than Largest)
                then
                  3.1.1  Set Largest to the value of the integer
               End if
      3.2  Increment Counter
    End while
4.  Return Largest
    End
```

1. **Write an algorithm in pseudocode that finds the average of (10) numbers.**

```
Input: 10 positive integers
Output: average of 10 integers
Process:
sum 0;
for i←0; i<=10; i←i+1 do
      input x;
       sum=sum+x;
Endfor
Avg=sum/10;
Return Avg;
```

Write an algorithm which requires a number between 10 and 20, until the response is appropriate. If the number is more than 20, it will display a message: “Bigger!" If the number is less than 10, it will display “smaller!"

```
Begin
Input: num
Output: numbers between 10 and 20
Process:
Start
      if (num<10) Then
        print “Smaller !”
    elseif (num >20)
       print “Bigger !”
   End if
End
```

What are the values of the variables A, B and C after execution of the following instructions?

Begin
A ← *3*
B ← *10*
C ← *A* + *B*
B ← *A* + *B*
A ← *C*
End

Write an algorithm to swap the value the 2 variables A and B.

Input: A and B and C
Output: Swapping
Process:

```
Start
    C A;
    A B;
    B C;
    Return A and B;
End
```

Write pseudocode that will take a number as input and tells whether a number is positive, negative or zero.

Solution:

```
Begin
WRITE “Enter a number”
READ num
IF num> 0  THEN
     WRITE “The number is positive”
ELSE IF num = 0 THEN
   WRITE “The number is zero”
ELSE
    WRITE “The number is negative”
    ENDIF
ENDIF
End
```