DATA STRUCTURE & ALGORITHMS

CSCR2104

INTRODUCTIONS

Instructor:

Marwa Elfatih Ahmed.

Email

Marwalfatih.uofb@gmail.com

Lecture Contents:

- Part one:
  - Course Outlines.
- Part two:
  - Introduction to Data Structure & Algorithms.

Description

- This course covering
  1. some of the more advanced fundamentals of programming including basic data structures (such as lists, stacks and queues, binary trees, and hash tables),
  2. recursion, common algorithms (such as searching and sorting).
  3. It is also focusing on algorithm design and analysis and the relationships between data representation, algorithm design, and program efficiency.
  4. Topics include key algorithm design techniques, analysis of the time and space requirements of algorithms, and characterizing the difficulty of solving a problem

Goals of the course

- Provide background in **data structures** (arrangements of data in memory) and **algorithms** (methods for manipulating this data)
- Why is this important?
  - The answer is there is far more to programming than finding a solution that will work:
    - Execution time.
    - Memory consumption.
- So by the end of the course, you should be more equipped to not just develop an accurate solution to a problem, but the *best* solution possible.

Learning Outcomes

- **On the completion of this course student would be able to:**
  - **Write programs that use each of the following data structures: arrays, records, strings, linked lists, stacks, queues, and hash tables.**
  - **Describe the concept of recursion.**
  - **Determine the time and space complexity of simple algorithms.**
  - **Describe the several kinds of algorithm (brute force, greedy, divide-and-conquer, backtracking, branch-and-bound, and heuristic).**
  - **Design and implement the most common sorting algorithms.**
  - **Illustrate by example the basic terminology of graph theory and trees**

Course Outline

- Fundamental data structures:
  - Arrays; Stacks; queues; linked lists; hash tables; trees; graphs
- Fundamental computing algorithms:
  - simple searching and sorting algorithms (linear and binary search, selection and insertion sort); hash tables, including collision-avoidance strategies; binary search trees; representations of graphs; depth- and breadth-first traversals
- Recursion:
  - The concept of recursion; recursive mathematical functions; simple recursive procedures; divide-and-conquer strategies; recursive backtracking; implementation of recursion

Course Outline (Cont’d)

- **Basic algorithmic analysis:**
  - symptotic analysis of upper and average complexity bounds; identifying differences among best, average, and worst case behaviors; big “O,” little “o,” omega, and theta notation; standard complexity classes; empirical measurements of performance; time and space tradeoffs in algorithms; using recurrence relations to analyze recursive algorithms
- **Algorithmic strategies:**
  - Brute-force algorithms; greedy algorithms; divide-and conquer; backtracking; branch-and-bound; heuristics; pattern matching and string/text algorithms; numerical approximation algorithms

Exams

- Final semester examinations 60%.
  - 50% written examinations.
  - 10% final lab exam or project.
- continuous assessment 40%
  - 20% theoretical.
  - 20% practical (Lab).
- continuous assessment 20% theoretical
  - mid-term exam.
  - quizzes,
  - assignments,
  - scientific reports,
  - seminars.

Part two

- **Data Structure.**
- **Algorithms.**

Data Structures

- Elementary data types
  - Are characterized by the fact that the values they admit are atomic that is, they admit no further decomposition.
    - Integer, real and character.
- Structures
  - If the values of a data type do admit a decomposition into several components .
    - Arrays, records, struct.
- Data Structures
  - The Structures and the organization of each component and the relationships between component.

Data Structures (Cont'd)

- **Data structure is a special way of organizing and storing data in a computer so that it can be used efficiently**
- **Data structure is the study of the organization and manipulation of data, in terms of**
  - **how they are arranged logically in relation to each other,**
  - **how they are stored physically in memory / storage**
  - **how they are accessed and processed**

Data Structures and Algorithms

- **A data structure is an arrangement of data in a computer’s memory/disk.**
- **Algorithms manipulate the data in these structures in various ways, such as searching for a particular data item and sorting the data**
- **sorts of problems can be solved:**
  - **Real-World Data Storage.**
  - **Programmer’s Tools.**
  - **Real-World Modeling.**

Organization of the Data structure

- A data structure may be **organized** to **facilitate** specific types of **usage:**
  - Stored in **first-come**, **first-served** manner (a waiting line or **queue**)
  - Stored in **last-in**, **first out** manner (stack)
  - Stored in order of priority (**priority-queue**)
  - Stored in **sorted** order
  - Organized for **efficient storage** and **retrieval** or for efficient **access** to **particular items**
  - Organized for **minimal memory** usage
  - Organized for **fastest access**

Manipulation of Data Structures

- Each type of data structure has its own unique **characteristics**
- There are **trade-offs** in deciding which **data structure** to use
  - **Memory** requirements.
  - **Efficiency** (**speed**) of storing, finding, retrieving, or processing the data.
  - **Ease** of
    - Understanding
    - Programming
    - Testing/debugging/maintenance.
  - A **fast algorithm** on a given data structure may require **more memory** and be **more difficult** to **understand**, but a **slower algorithm** may take **less memory** and be **easier** to **maintain**

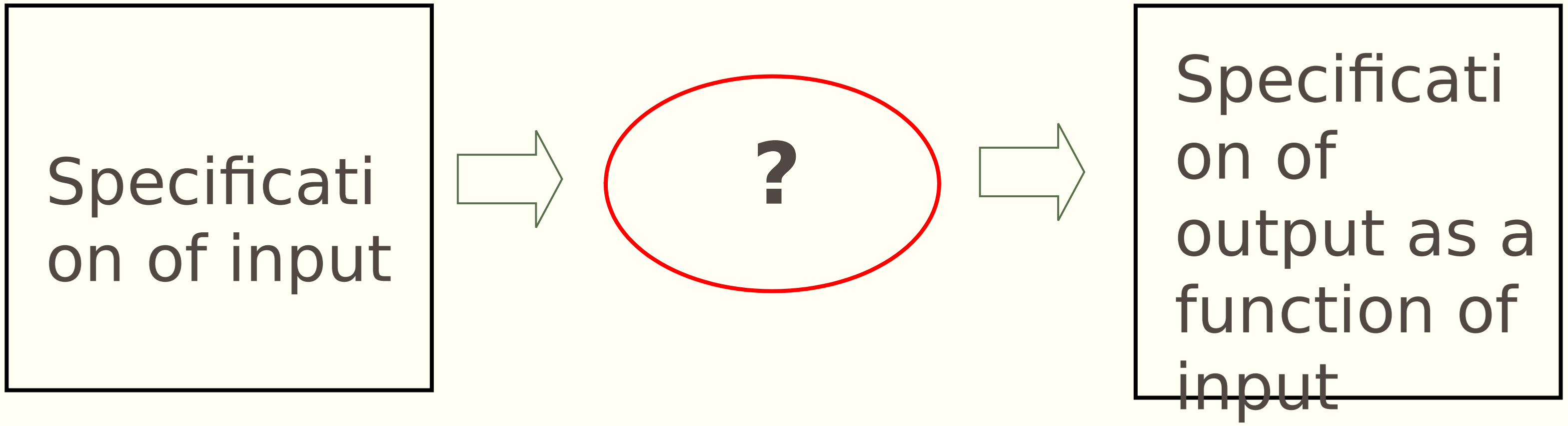
Data Structures (Cont’d)

- Take linked lists as examples of applications of pointers.
- In general, a data structure can be characterized by
  - the type of the items contained in the list,
  - the internal list structure: arrays or linked list of items,
  - the operations that can be performed on the list.
- From user's point of view, the following operations are essential:
  - CreateList
  - SetPositions (SearchList)
  - InsertElement
  - DeleteElement

Algorithms

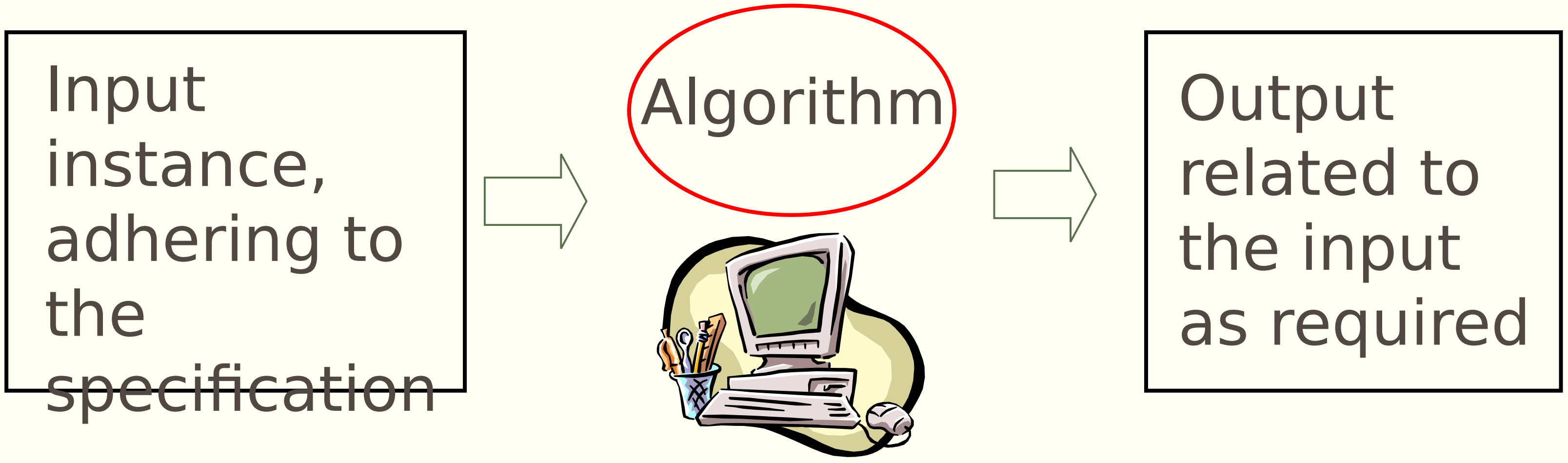
- The must common premises of algorithm is:
  1. An algorithm is a sequence of instruction for solving a problem.
  2. The problem to be solved has specifiable goal conditions.
  3. The intent of each instruction in the algorithm must be apparent to and within the capabilities of the agent that will be carrying out the instruction.
  4. As soon as one instruction in the algorithm has been carried out, the next instruction to be performed is uniquely determined by the instruction just completed and the current status of the solution.

Algorithmic problem



- Infinite number of input *instances* satisfying the specification. For example:
  - A sorted, non-decreasing sequence of natural numbers. The sequence is of non-zero, finite length:
    - 1, 20, 908, 909, 100000, 1000000000.

Algorithmic Solution


Input instance, adhering to the specification
Algorithm
Output related to the input as required

- Algorithm describes actions on the input instance
- Infinitely many correct algorithms for the same algorithmic problem

Algorithm

An algorithm is an <u>unambiguous</u> <u>sequence</u> of <u>clear instruction</u> that will solve a given problem in a <u>finite amount of time</u> and <u>then test</u>.

Programs

- *Programs = ? + ?*
- *Programs = Data Structures + Algorithms*
- While programs contain other components such as user interfaces, the heart of any program is the combination of its data and what the program does with the data
- To design a good program, one must select appropriate data structures and algorithms
- The purpose of this course is to enable one to make intelligent decisions about these

DATA STRUCTURE & ALGORITHMS

Array

Objectives

You should be able to describe:

- One-Dimensional Arrays
- Array Initialization
- Arrays as Arguments
- Two-Dimensional Arrays
- Common Programming Errors

Arrays :
One-Dimension Arrays

- One-Dimension Array(Single-Dimension Array or Vector): a list of related values
  - All items in list have same data type
  - All list members stored using single group name
- Example: a list of grades

  98, 87, 92, 79, 85

One-Dimension Arrays (continued)

- Array declaration statement provides:
  - The array(list) name
  - The data type of array items
  - The number of items in array
- Syntax

  dataType arrayName[numberOfItems]

  - Common programming practice requires defining number of array items as a constant before declaring the array

Arrays as Arguments

- Array elements are passed to a called function in same manner as individual scalar variables
    - Example:

      ```
      findMax(grades[2], grades[6]);
      ```

- Passing a complete array to a function provides access to the actual array, not a copy
    - Making copies of large arrays is wasteful of storage

Arrays as Arguments (continued)

- Examples of function calls that pass arrays

```
int nums[5];    // an array of five integers

char keys[256];   // an array of 256 characters

double units[500], grades[500];// two arrays of 500
    //doubles
```

- The following function calls can then be made:

```
        findMax(nums);
        findCharacter(keys);
        calcTotal(nums, units, grades);
```

Two-Dimensional Arrays

- Two-dimensional array (table): consists of both rows and columns of elements
- **Example**: two-dimensional array of integers

8	16	9	52
3	15	27	6
14	25	2	10

- **Array declaration**: names the array val and reserves storage for it

```
int val[3][4];
```

Two-Dimensional Arrays (continued)

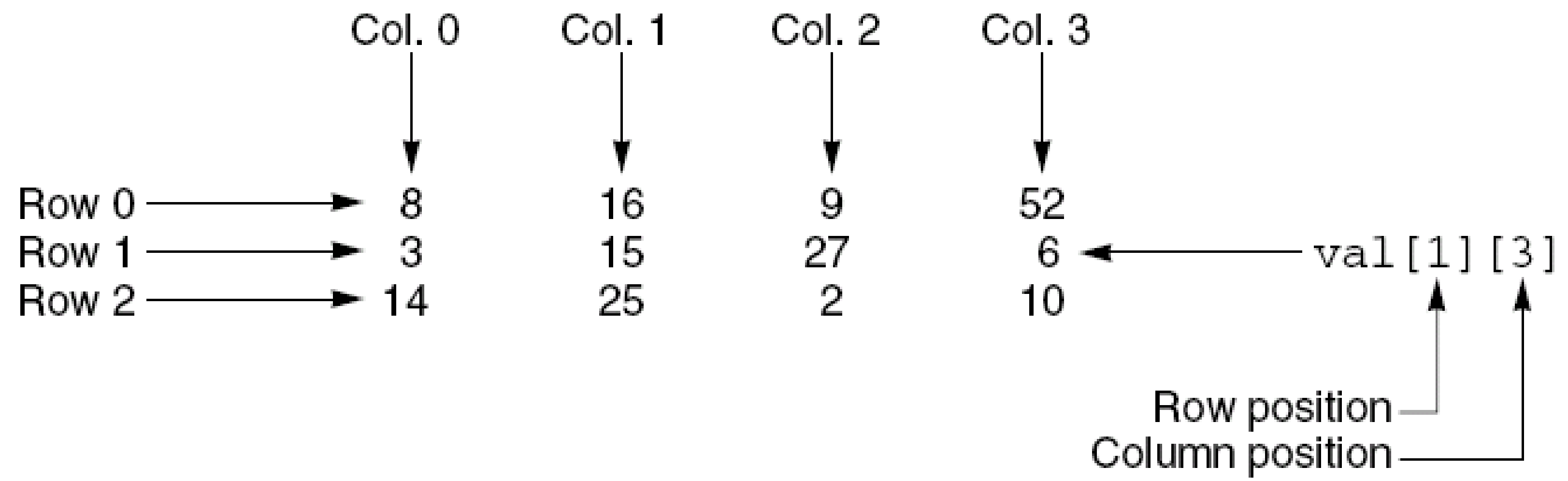
- Locating array elements
  - val[1][3] uniquely identifies element in row 1, column 3
- Examples using elements of val array:

```
price = val[2][3];
val[0][0] = 62;
newnum = 4 * (val[1][0] - 5);
sumRow = val[0][0] + val[0][1] + val[0][2] + val[0][3];
```

  - The last statement adds the elements in row 0 and sum is stored in sumRow

Two-Dimensional Arrays (continued)

FIGURE 8.9 *Each Array Element Is Identified by Its Row and Column Position*



2-Dimensional Arrays Initialization

- can be done within declaration statements (as with single-dimension arrays)
- Example:

```
int val[3][4] = { {8,16,9,52},
                  {3,15,27,6},
                  {14,25,2,10} };
```

  - First set of internal braces contains values for row 0, second set for row 1, and third set for row 2
  - Commas in initialization braces are required; inner braces can be omitted

2-Dimensional Arrays Processing

- Processing two-dimensional arrays: nested for loops typically used
    - Easy to cycle through each array element
        - A pass through outer loop corresponds to a row
        - A pass through inner loop corresponds to a column
    - Used Nested for loop to multiply each val element by 10 and display results

2-Dimensional Arrays as Arguments

- Prototypes for functions that pass two-dimensional arrays can omit the row size of the array

  - Example ():

    Display (int nums[ ][4]);

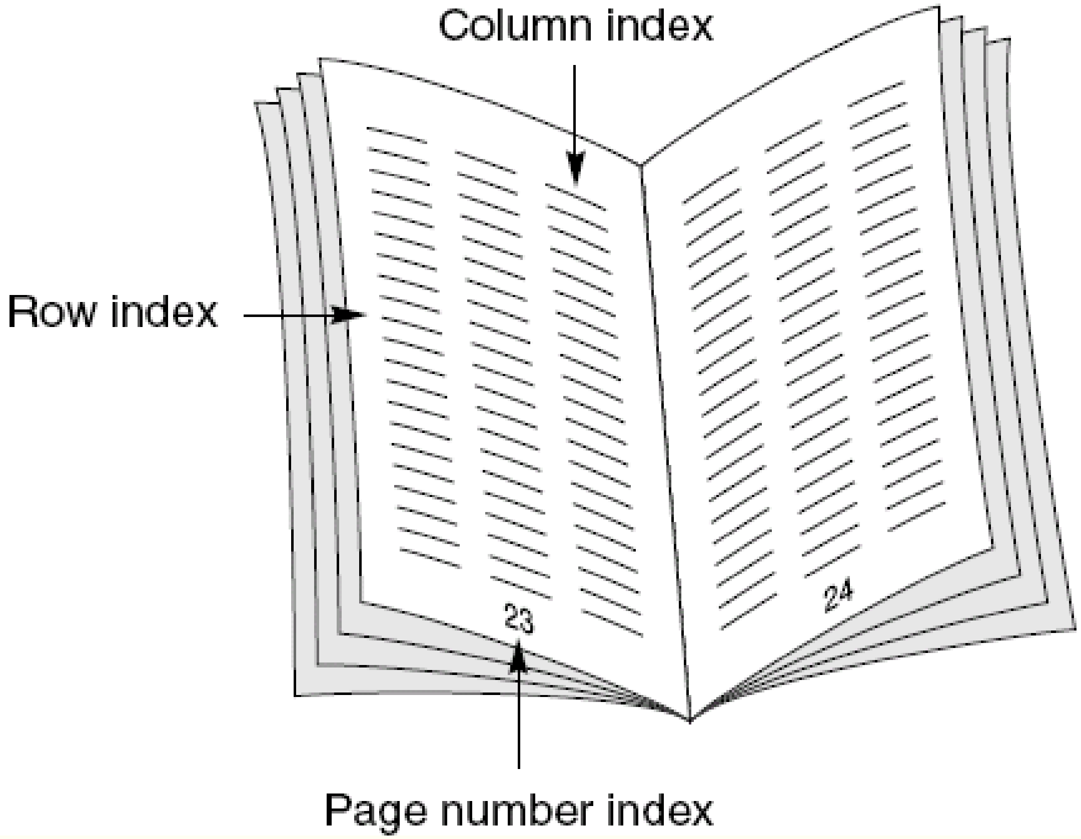
  - Row size is **optional** but column size is **required**

Larger-Dimension Arrays

- Arrays with more than two dimensions allowed but not commonly used
- Example: int response[4][10][6]
  - First element is response[0][0][0]
  - Last element is response[3][9][5]
- A three-dimensional array can be viewed as a book of data tables
  - First subscript (rank) is page number of table
  - Second subscript is row in table
  - Third subscript is desired column

Larger-Dimension Arrays (continued)

FIGURE 8.12 *Representation of a Three-Dimensional Array*



Common Programming Errors

- **Forgetting to declare an array**
  - Results in a compiler error message equivalent to “invalid indirection” each time a subscripted variable is encountered within a program
- **Using a subscript that references a nonexistent array element**
  - For example, declaring array to be of size 20 and using a subscript value of 25
  - Not detected by most C++ compilers and will probably cause a runtime error

Common Programming Errors (continued)

- Not using a large enough counter value in a for loop counter to cycle through all array elements
- Forgetting to initialize array elements
  - Don’t assume compiler does this

Summary

- Single-dimensional array: a data structure that stores a list of values of same data type
  - Must specify data type and array size
  - `int num[100];` creates an array of 100 integers
- Array elements are stored in contiguous locations in memory and referenced using the array name and a subscript
  - For example, `num[22]`

Summary (continued)

- Two-dimensional array is declared by listing both a row and column size with data type and name of array
- Arrays may be initialized when they are declared
  - For two-dimensional arrays you list the initial values, in a row-by-row manner, within braces and separating them with commas
- Arrays are passed to a function by passing name of array as an argument

Questions?