**Lesson Proper for Week 1**

Completion requirements

**Data Structure**

Specialized format to store and organize data in a computer’s memory or disk collection of variables, possibly of several different data types connected in various ways

Types of Data Structures

Array

Linked list

Stacks/Queues

Trees

Hash tables

**Data Types**

Data that a variable can hold in a programming language all programming language has a set of built-in data types

Examples:

**char** - data type representing character values

**int** - data type representing integer values

**Abstract Data Types**

Specification of a set of data and set of operations performed in a data storage for data defined in terms of set of operations to be perform on the data

**Algorithms**

Finite set of instructions that specify a sequence of operations to be carry out recipe for solving a problem

Example of simple algorithm

ü  Apply to wet hair

ü  Massage gently

ü  Leave on for a few moments

ü  Rinse Off

**Note:**

All the tasks that can be carry out by a computer can be state as algorithms

**Criteria for Algorithm**

Every algorithm must satisfy the following criteria:

**1. Input -**zero or more quantities are externally supply

**2. Output -**at least one quantity is produce

**3. Definiteness -**each instruction must be clear and unambiguous

**4. Finitness -**all instructions of an algorithm will terminate after a finite number of steps

**5. Effectiveness -**each operation must be definite, but must also be feasible

**Notes for Criteria for Algorithm:**

**Inputs -**are the data items presented to the algorithm. An algorithm has either no input or a predetermined number of them.

**Output -**are the data items presented to the outside world as the result of the execution of a program based on the algorithm. An algorithm ought to produce at least one output.

**Procedure**

Essential tool in programming that generalizes the notion of an operator, Procedure used to encapsulate parts of an algorithm by localizing in one section of a program all the statements relevant to a certain aspect of a program.

Raw data is an input to a computer and an algorithm used to transform this into a refined data.

Computer Science also refers to the study of data. It include:

Machines that holds data

Languages for describing data manipulation

Foundations, which describe what kinds of refined

Data can be produce from raw data

Structures for representing data

**PSEUDOCODE**

Textual presentation of a flowchart close to a natural language the control structures impose the logic may become part of the program documentation could be translated into a program

**STEPWISE REFINEMENT**

The process by which a programmer refines an initial idea to a problem's solution into more specific terms. The last phase of refinement results in a program ready to be coded for execution.

**Analysis of Algorithms**

Determining the amount of resources necessary to execute it such as time and storage,

Usually in terms of CPU time and memory requirements

Analysis of Algorithms

Best-case analysis

Worst-case analysis

Average case analysis

**Lesson Proper for Week 2**

Completion requirements

**Introduction to Array**

**LIST -**ordered set of a variable number of elements to which additions and deletions may be made one of the simplest and most commonly found type of data

**LINEAR LIST -**list which displays the relationship of physical adjacency

finite sequence of simple data, items, or records

Example

Days of the Week     (Monday, Tuesday, Wednesday,

                                     Thursday, Friday, Saturday,

                                     Sunday)

Values in a Deck of  (2, 3, 4, 5, 6, 7, 8, 9, 10, Jack,

Cards                                     Queen, King, Ace)

Floors of a Building Basement, Lobby, Mezzanine,

                                     First, Second, Third

**Arrays -**ordered collection of data items of the same type referred to collectively by a single name

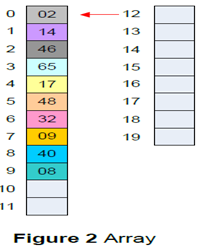
**Index**– each variable or cell in an array

**Elements** – individual data / items in an array indicated by the array name followed by its dimensions appears in a square brackets

**Dimensions** – an integer from 1 –n called dimensioned variables

      arrayName [0][0]

**Operations in Arrays**



**Insert**

Ø  Adds item to the indicated place/cell The process is very fast because it only includes one step

Ø  Every data inserted is placed in the first vacant cell

**Search**

Ø  Another process that the algorithm carries out he red arrow from the figure states where the search starts

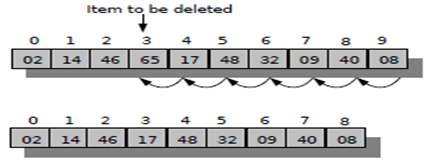
Ø  Moves methodically downwards to search for the item

**Delete**

Ø  The algorithm must first locate the item to delete it

Ø  The deleted item causes hole in the array

**Hole** one or more cells that have filled cells above them



**Basics of Arrays in C++**

Ø  Arrays are treated as primitive type, they are treated as objects

Ø  use the new operator to declare an array

      int[] intArray;                    // declares a reference to an array

      intArray = new int[100];             // initialize the array, and sets intArray to

                                                                        refer to it

**Note:**

1.    intArray name is a reference to an array and not the array itself

2.    The length field of an array can be used to find the size in bytes:

      int arrayLength = intArray.length;                   //find array length

      The size of an array, when created, can no longer be modified

3.    Array elements can be accessed through the use of square brackets

      temp=intArray[6];                        // get contents of seventh element of

                                                                        array

      intArray[7]=48;                // insert 48 into the eight cell

4.    the first element is numbered 0 so that the indices in an array of 10 elements run from 0 to 9

5.    initialize first an array before attempting to access it to avoid syntax error

6.    can initiate an array of primitive type to something besides 0 using:

      int[] intArray = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};

      Syntax below gives you a runtime error

      autoData[] carArray = new autoData[4000];

**Dimensionality of an Array**

§  Onedimensional Array

      Array1[j]

§  Twodimensional Array

      Array1[i ][j ]

§  Multidimensional Array

      Array[i][j][k]…

**One Dimensional Array**

ü  It is the basic array

ü  A vertical table with number of columns and only one row

      type[] variableName;

      An array of ints

                                     int[] intArray;

      An array of Random objects

                                     Random[] randomArray;

Arrays are fixed length structure

What is the difference of the two-array syntax below?

      int[] intArray = new intArray[5];

                                     and

      int size = 5;

      Random[] randomArray = new Random[size];

**Index and Length**

**Indices**–used to access each location of arrays from 0 ranging up to the instantiated

Array

An array of length 10, instantiated as

      int[] intArray = new intArray[10]

Attempting to access an index which is out of range, gives you

      ArrayIndexOutOfBoundsException

Example

int size = 5;

int[] intArray = new int[size];

for(int index = 0; index < size;

index++)

{ intArray[index] = index; }

**Changing the Size of an Array**

         Analyze the syntax below and give its output:

int[] intArray = new int[5];

for (int index = 0; index < 3;

index ++)

{intArray[index] = index + 1;}

How to make your array “grow” and “shrink”?

      create a temporary array of the desired size

      using for loop, access each item from the

      original array and save it to the

      corresponding index of the temporary array

      save the temporary array into the variable

      of your original array

Example (continued..)

int[] intArray = new int[5];

for (int index = 0; index < 3;

index ++)

{ intArray[index] = index + 1; }

int[] tempArray = new int[3];

for (int index = 0; index <

tempArray.length; index++)

{ tempArray[index] =

intArray[index]; }

intArray = tempArray;

**Two Dimensional Array**

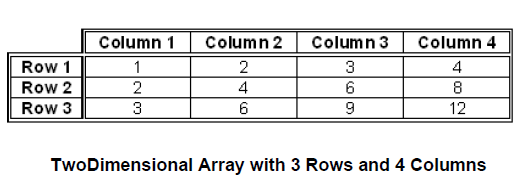
ü  It is an array of an array

ü  Also called as tablesor matrices

**Row**–first dimension

**Column**–second dimension

      Arr1[2][3]



 Declaration and instantiation of 2D array is similar to that of 1D array

int rows = 3, cols = 4;

int[][] tableArray = new int[rows][cols];

for (int rowIndex = 0; rowIndex < rows;

rowIndex ++)

{

for (int colIndex = 0; colIndex <

cols; colIndex ++)

{tableArray[rowIndex][colIndex] =

rowIndex \* colIndex + 1; }

}}

Curly braces is the alternative constructor for 2D

arrays

int[][] tableArray = { {1,2,3,4,5},

{6,7,8,9,0} };

For a more look of like a table

int[][] tableArray = { {1,2,3,4,5} }

{ {6,7,8,9,0} };

You can initialize the row dimension without the

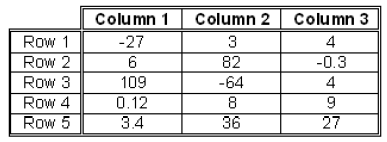
columns but not vice versa

int[][] tableArray = new int[5][]; //

illegalint[][]

tableArray = new int[][5]; // accepted

**Matrix-**is a mathematical object which arises in many physical problems and consists of mrows and n columns

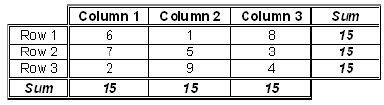


mx n (read as m by n) is written to

designate a matrix with mrows and n

columns

**Magic Square**–is an n x n matrix of the integers 1 to n2



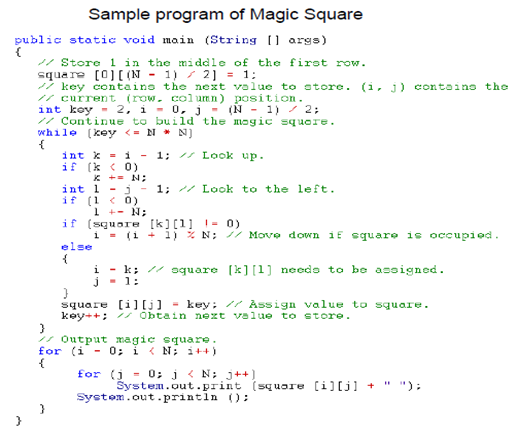
**Algorithm to create a magic square by H. Coxeter**

ü  Begin with 1 inthe middle of the top row

ü  Move up and left assigning numbers in increasing order to empty squares

ü  If you fall off the square imagine the same square as tilting the plane and continue

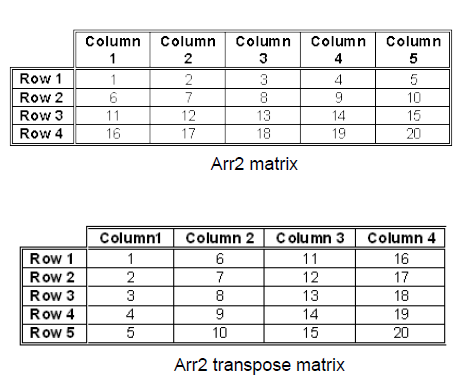
ü  If a square is occupied, move down instead and continue



**Transpose**–another operation that can be performed on matrices that computes for the transpose of matrix

ü  the elements in the [i,j] position are transferred to the [j,i] position

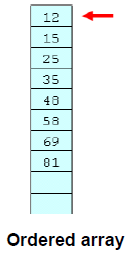
ü  elements in the diagonal will always remain in the same position since i= j



**Linear Search**

ü  checks every element of a list one at a time in sequence

ü  also called as sequential search



**Algorithm for linear search**

1.    Create an ordered array

2.    Initialize a target value or the key

3.    If index == key, return true

4.    If index < key, return false

5.

**Binary Search**

ü  Usually coded as

      mid = (high + low) /2;

ü  Algorithm for binary search

1.    Array elements should be sorted

2.    Find the middle

3.    Compare the middle item to the target value or the key

4.    If not found read, search parameters, half the size and start over

5.    If found, return

**Lesson Proper for Week 3**

Completion requirements

**Searching**

v  great algorithms have been devised for searching because it is so important

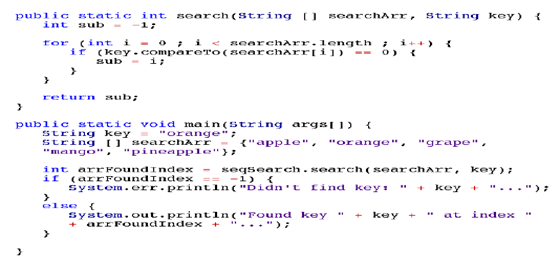
v  information retrieval is one of the most common operations performed by computer applications

v  search key uniquely identifies the data being requested by the user

**Key field** contains the key of the record

**Information field** contains the information associated with the key in the key field

What does the program do?



**Sorting**

How to sort?

v  as humans, we can do this by lining up the items

v  computers not like humans can visually compare things two items at once It uses two steps to execute over and over until data are sorted

§  compare the two items

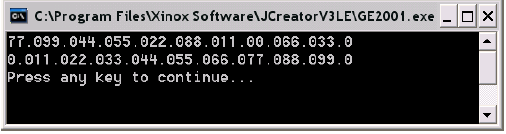
§  swap the two items or copy one

§  item but still; the details are

§  handled in different ways

**Bubble sort** simplest in sorting process Compare two elements in the array If the element on the left is larger than the element in the right, swap them If vice versa, no move will be done Compare now the element that was swapped to the element on its right position

Below is the output of bubbleSort.cpp



**invariants** a condition that is always true at a certain point in a program

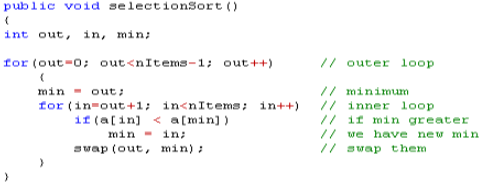
**Selection** **sort** is done by selecting an element in the list and moving it to the

proper position

1.    find the minimum value in the list

2.    swap it with the value in the first position

3.    repeat the steps for the remainder of the list starting the next position after the swapping to the first position

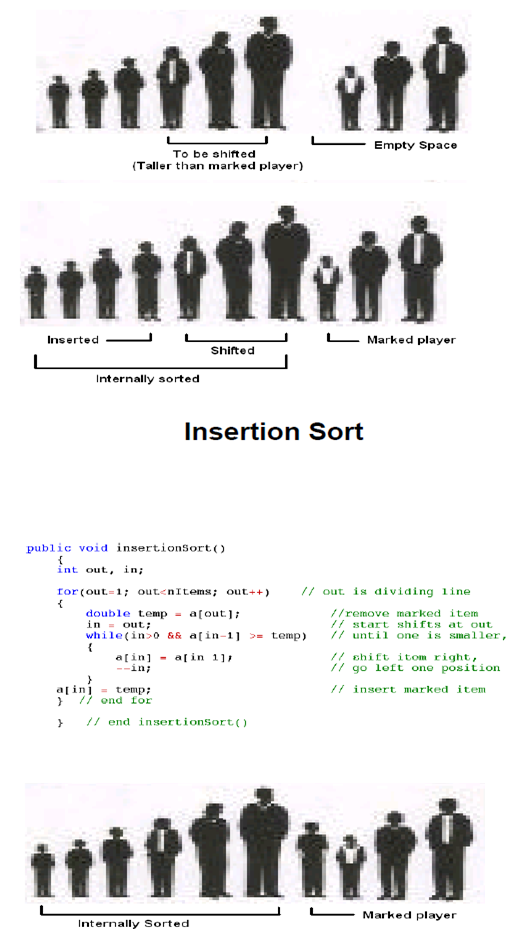


**Insertion Sort**

To perform insertion sort:

1.    Divide the list into two: sorted part and the unsorted part

2.    Unsorted part: Transfer one by one to their correct location in the  sorted area



**Lesson Proper for Week 4**

Completion requirements

**Recursion**technique in programming that calls itselfhas the capability to save the condition it was inor the particular process it was serving whencalling itself

The design of a recursive method consists of the following elements:

Ø    One or more stopping condition that can be directly evaluated for certain arguments.

Ø    One or more recursive step, sin which a current value of the method can be computed by repeated calling of the method with arguments that will eventually arrive at a stopping condition

Two classes of recursion

Ø    Direct Recursion

Ø    Indirect Recursion

**Recursion** is viewed as a somewhat mystical technique which is only useful for some very special class of problems

ü    Can be written using:

Assignment statement

Ifelse statement

While statement

ü    Can also be written using:

Assignment statement

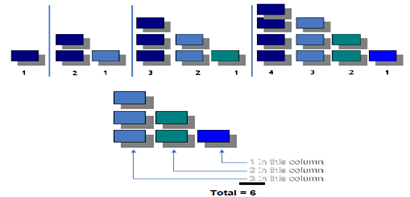
iIfelse statement

recursion

**Triangular Numbers**the sum of 1 to n numbers arranged inequilateral triangle

seen by row where each row is longer than the

previous row the sum of consecutive integers



**Triangular Numbers**

int triangle(int n)

{

int total = 0;

while(n > 0) // until n

is 1

{

total = total + n; // add n

(column height) to total

n;

// decrement column height

}

return total;

}

**Finding the Nth term using Recursion**

Ø    The first (tallest) column, which has the value of n.

Ø    The sum of all the remaining columns.

To find the remaining column:

int triangle(int n)

{

return (n +

sumRemainingColumns);

}

//find the sum of all the remaining columns for

term n:

int triangle(int n)

{

return(n + sumAllColumns(n1));

}

//Instead

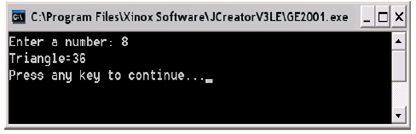
int triangle

{

return(n + triangle(n1));

}

**Sample Output**



 Recursive methods carries several features

Ø    Calls itself

Ø    Calls itself to solve a smaller problem

Ø    Some version of the problem that is simple where the routine can solve it without calling          itself

**Recursive algorithm**

**1.**There must be at least one case (the basecase), for a small value of n that can be solved directly

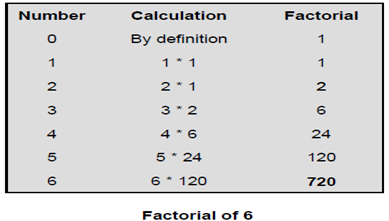
2.            A problem of a given size n can be split into one or more smaller versions of the same problem

3.            Identify the base case and provide a solution to it

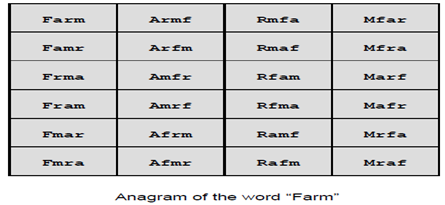
4.            Create a strategy to split the problem into smaller versions of itself while making progress toward the base case

5.            Merge the solution that will solve the larger problem

**Factorial**It is the product of a series of consecutivepositive integers from 1 to a given number, nExpressed by the symbol !



**Anagrams**forming of real English word using the sameletters, no more, no less, the process is calledanagramming

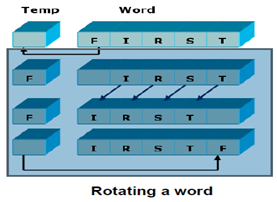


**Algorithm to anagram a word**

1.            Anagram the rightmost n1 letters

2.            Rotate all n letters

3.            Repeat these steps n time



**Anagrams**

public static void rotate(int newSize)

{

int i;

int position = size newSize;

// save first letter

char temp = charArray[position];

//shift others left

for (i = position + 1; i < size; i++)

charArray[i 1] = charArray[i];

//put first on right

charArray[i 1] = temp;

**Recursive Binary Search**

**Algorithm for recursive binary search**

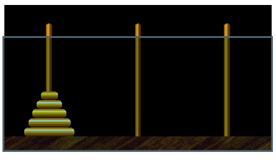
1.            Search in the array A in positions numbered loIndex to hiIndex inclusive, for the specified value

2.            If the value is found, return the index in the array where it occurs.

3.            If the value is not found, return 1

4.            Precondition: The array must be sorted into increasing order

**Towers of Hanoi**invented by Edouard Lucas in 1883its objective is to transfer the entire tower to the last peg, moving only one disk at a time without moving a larger one onto a smaller

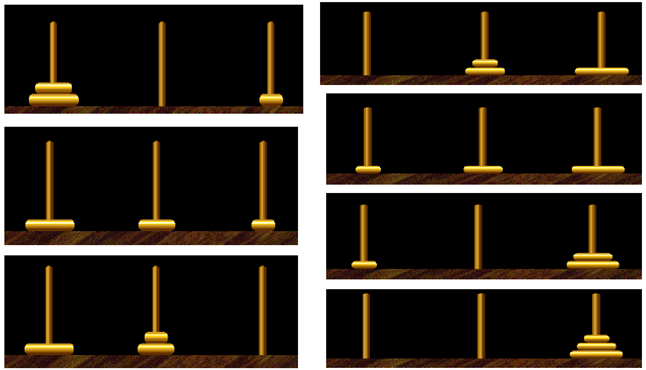


**Algorithm for Towers of Hanoi**

1.            Move the smaller disc from the top n1 from A to B

2.            Move the remaining (larger) disc from A to C

3.            Move the smaller disc from B to C



doTowers(int topN, char from, char inter, char to)

{

if(topN==1)

System.out.println("Disc 1 from " + from + " to " + to);

else

{

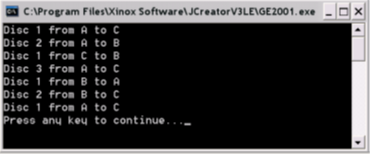
doTowers(topN1,from, to, inter); // from>inter

System.out.println("Disc " + topN + "from " + from + "to " + to);

doTowers(topN1, inter, from, to); // inter>to

}

}



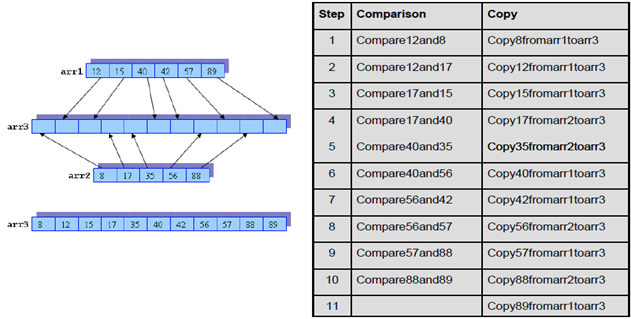
**Merge sort**requires another array in the memory

**Algorithm for merge sort**

1.            Divide the list into two, with your desired size

2.            Sort the first and second list

3.            Megre the two sublists into one sorted list



**Types of Recursion**

                  Linear Recursion

                  Tail Recursion

                  Binary Recursion

                  Exponential Recursion

                  Nested Recursion

                  Mutual Recursion

**Depth of Recursion**number of times the procedure is calledrecursively in the process of evaluating agiven argument