Task # 1

Simple Array Searching

using System;

namespace abc

{

class Program

{

int[] array = new int[5];

int y, i;

public void search()

{

Console.WriteLine("Enter array 5 members: ");

for (i = 0; i < 5; i++)

{

array[i] = Convert.ToInt32(Console.ReadLine());

}

Console.WriteLine("which number do u want to search: ");

y = Convert.ToInt32(Console.ReadLine());

for (i = 0; i < 5; i++)

{

if (y == array[i])

{

Console.WriteLine("index is {0} ", i);

}

}

}

static void Main(string[] args)

{

Program p = new Program();

p.search();

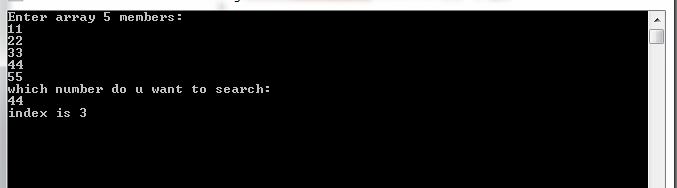
Console.ReadLine();

}

}

}

Console



Task # 2

Reverse Array

using System;

namespace ds\_Labfile\_t2

{

class RevTraverse

{

public void tra()

{

int[] array1 = new int[50] ;

int i,UB;

Console.Write("\n\n\t\t\tEnter The Array Size : " );UB = int.Parse(Console.ReadLine());

Console.Write("\t\tEnter The Elements In Array : \n");

for(i=0;i<UB;i++){ array1[i]= int.Parse(Console.ReadLine());}

Console.WriteLine("\t\tReverse of Array is : \n");

for (i = UB; i >= 0; i--) { Console.Write("\n\t" + array1[i]); }

}

}

class Program

{

static void Main(string[] args)

{

RevTraverse rt = new RevTraverse();

rt.tra();

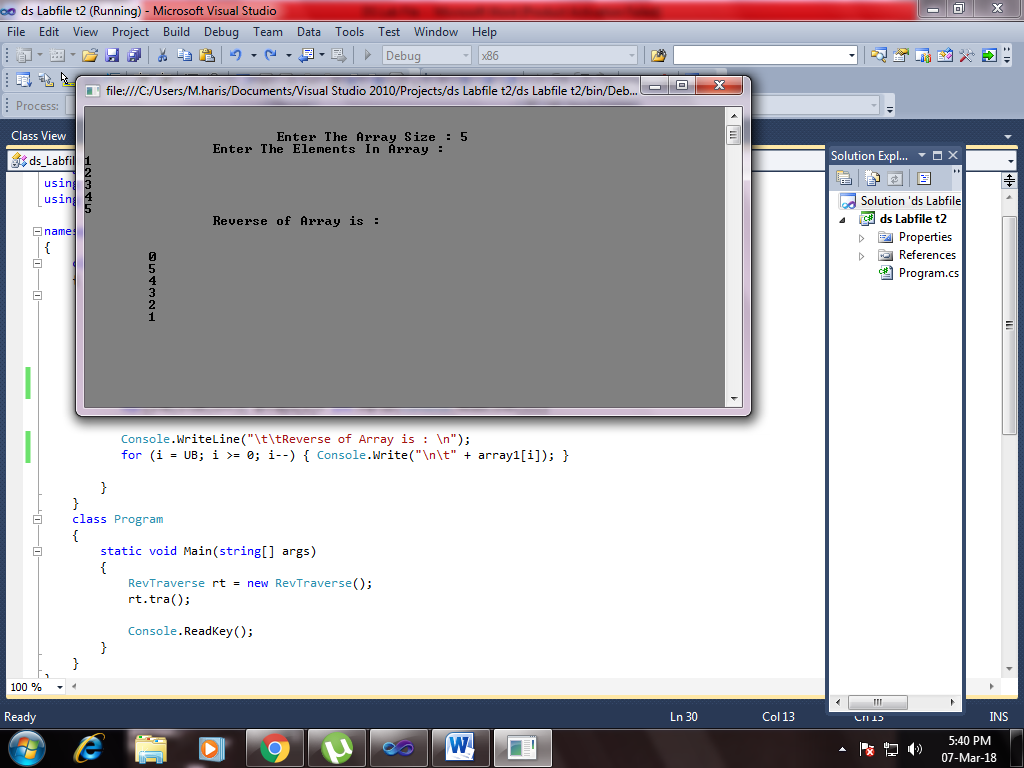
Console.ReadKey();

}

}

}

Console



Task # 3

Single linked list

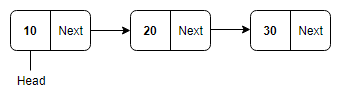
Introduction

Linked List is a linear data structure which consists of a group of nodes in a sequence. Each node contains two parts.

Data− Each node of a linked list can store a data.

Address − Each node of a linked list contains an address to the next node, called "Next".

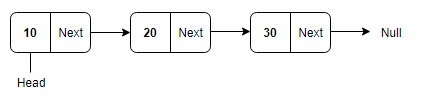
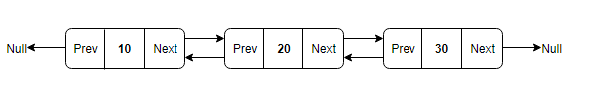
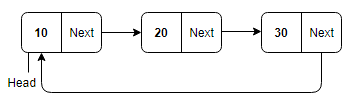
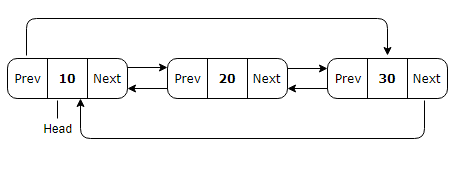
The first node of a Linked List is referenced by a pointer called Head



Advantages of Linked List

* They are dynamic in nature and allocate memory as and when required.
* Insertion and deletion is easy to implement.
* Other data structures such as Stack and Queue can also be implemented easily using Linked List.
* It has faster access time and can be expanded in constant time without memory overhead.
* Since there is no need to define an initial size for a linked list, hence memory utilization is effective.
* Backtracking is possible in doubly linked lists.

Types of Linked List

* Singly Linked List: Singly linked lists contain nodes which have a data part and an address part, i.e., Next, which points to the next node in the sequence of nodes. The next pointer of the last node will point to null.  
    
  
* Doubly Linked List: In a doubly linked list, each node contains two links - the first link points to the previous node and the next link points to the next node in the sequence.The prev pointer of the first node and next pointer of the last node will point to null.  
  
* Circular Linked List: In the circular linked list, the next of the last node will point to the first node, thus forming a circular chain.  
    
  
* Doubly Circular Linked List: In this type of linked list, the next of the last node will point to the first node and the previous pointer of the first node will point to the last node.  
  

using System;

namespace singlelinklistp

{

class demo

{

static void Main(string[] args)

{

int choice, data, k, x;

singlelinklist list = new singlelinklist();

while (true)

{

Console.WriteLine("\n\n\t\t\t\a\a Single Linked List ");

Console.WriteLine("\n\t\t13: Create list ");

Console.WriteLine("\t\t1: display list ");

Console.WriteLine("\t\t2: count nodes ");

Console.WriteLine("\t\t3: search in list ");

Console.WriteLine("\t\t4: insert in empty list/at beginning ");

Console.WriteLine("\t\t5: insert a node at the end of list ");

Console.WriteLine("\t\t6: inserta node after a specifed ndoe ");

Console.WriteLine("\t\t7: inserta node before a specifed posotion ");

Console.WriteLine("\t\t8: inserta node at given position ");

Console.WriteLine("\t\t9: delet first node ");

Console.WriteLine("\t\t10: delete last node ");

Console.WriteLine("\t\t11: delet any node ");

Console.WriteLine("\t\t12: Quit ");

Console.Write("Select : "); choice = Convert.ToInt32(Console.ReadLine());

if (choice == 12)

break;

switch(choice)

{

case 13:

list.CreateList();

break;

case 1:

list.DisplayList();

break;

case 2:

list.CountNodes();

break;

case 3:

Console.Write("Enter the element to be searched : ");

data = Convert.ToInt32(Console.ReadLine());

list.Search(data);

break;

case 4:

Console.Write("Enter the element to be inserted : ");

data=Convert.ToInt32(Console.ReadLine());

list.InsertInBeginning(data);

break;

case 5:

Console.Write("Enter the element to be inserted : ");

data = Convert.ToInt32(Console.ReadLine());

list.InsertAtEnd(data);

break;

case 6:

Console.Write("Enter the element to be inserted : ");data = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter the element after which to insert : ");x = Convert.ToInt32(Console.ReadLine());

list.InsertAfter(data,x);

break;

case 7:

Console.Write("Enter the element to be inserted : ");data = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter the element after which to insert : ");x = Convert.ToInt32(Console.ReadLine());

list.InsertBefore(data,x);

break;

case 8:

Console.Write("Enter the element to be inserted : "); data = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter the position at which to insert : "); k = Convert.ToInt32(Console.ReadLine());

list.InsertAtPosition(data, k);

break;

case 9:

list.DeleteFirstNode();

break;

case 10:

list.DeleteLastNode();

break;

case 11:

Console.Write("Enter the element to be deleted : "); data = Convert.ToInt32(Console.ReadLine());

list.DeleteNode(data);

break;

default:

Console.WriteLine("Wrong Choice");

break;

}

Console.WriteLine();

}

Console.WriteLine("Exiting");

}

}

}

using System;

namespace singlelinklistp

{

class node

{

public int info;

public node link;

public node(int i)

{

info = i;

link = null;

}

}

}

using System;

namespace singlelinklistp

{

class singlelinklist

{

private node start;

public singlelinklist()

{

start = null;

}

public void DisplayList()

{

node p;

if (start == null)

{

Console.Write("List is empty");

return;

}

Console.Write("List is : ");

p = start;

while (p != null)

{

Console.Write(p.info + " ");

p=p.link;

}

Console.WriteLine();

}

public void CountNodes()

{

int n=0;

node p=start;

while(p!=null)

{

n++;

p=p.link;

}

Console.WriteLine("Number of Nodes in List : " + n);

}

public bool Search(int x)

{

int position=1;

node p= start;

while(p!=null)

{

if(p.info==x)

break;

position++;

p=p.link;

}

if(p==null)

{

Console.WriteLine( x + " not found in list");

return false;

}

else

{

Console.WriteLine( x + " is at postion " + position);

return true;

}

}

public void InsertInBeginning(int data)

{

node temp = new node(data);

temp.link=start;

start = temp;

}

public void InsertAtEnd(int data)

{

node p;

node temp = new node(data);

if(start ==null)

{ start = temp; return; }

p=start;

while(p.link!= null)

p=p.link;

p.link=temp;

}

public void CreateList()

{

int i, n, data;

Console.Write("Enter the number of nodes : "); n = Convert.ToInt32(Console.ReadLine());

if (n == 0)

return;

for (i = 1; i <= n; i++)

{

Console.WriteLine("Enter the element to be insertd : ");data = Convert.ToInt32(Console.ReadLine());

InsertAtEnd(data);

}

}

public void InsertAfter(int data, int x)

{

node p = start;

while (p != null)

{

if (p.info == x)

break;

p = p.link;

}

if (p == null)

Console.WriteLine(x + " not present in the list");

else

{

node temp = new node(data);

temp.link = p.link;

p.link = temp;

}

}

public void InsertBefore(int data, int x)

{

node temp;

if (start == null)

{

Console.WriteLine("list is empty");

return;

}

if (x == start.info)

{

temp = new node(data);

temp.link = start;

start = temp;

return;

}

node p = start;

while (p.link != null)

{

if (p.link.info == x)

break;

p=p.link;

}

if (p.link == null)

Console.WriteLine(x + " not present in the list");

else

{

temp = new node(data);

temp.link = p.link;

p.link = temp;

}

}

public void InsertAtPosition(int data, int k)

{

node temp;

if (k == 1)

{

temp = new node(data);

temp.link = start;

start = temp;

return;

}

node p = start;

for (int i = 1; i <= k - 1 && p != null; i++)

{

p = p.link;

if (p == null)

Console.WriteLine("You can insert only upto " + i + "the position");

else

{

temp = new node(data);

temp.link = p.link;

p.link = temp;

}

}

}

public void DeleteFirstNode()

{

if (start == null)

return;

start = start.link;

}

public void DeleteLastNode()

{

if (start == null)

return;

if (start.link == null)

{

start = null;

return;

}

node p = start;

while (p.link.link != null)

p = p.link;

p.link = null;

}

public void DeleteNode(int x)

{

if (start == null)

{

Console.WriteLine("List is empty");

return;

}

if (start.info == x)

{

start = start.link;

return;

}

node p = start;

while (p.link != null)

{

if (p.link.info == x)

break;

p = p.link;

}

if (p.link == null)

Console.WriteLine("Elment " + x + " not in list");

else

p.link = p.link.link;

}

}

}

Task # 4

STACK

Introduction

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the push-down stacks only two operations are allowed:

* push the item into the stack
* pop the item out of the stack.

A stack is a limited access data structure - elements can be added and removed from the stack only at the top.

push adds an item to the top of the stack, pop removes the item from the top.

A stack is a recursive data structure. Here is a structural definition of a Stack:

a stack is either empty or it consists of a top and the rest which is a stack

Application

* To reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.
* An "undo" mechanism in text editors; this operation is accomplished by keeping all text changes in a stack.
* Undo/Redo stacks in Excel or Word.
* Language processing :
  + space for parameters and local variables is created internally using a stack.
  + compiler's syntax check for matching braces is implemented by using stack.
* A stack of plates/books in a cupboard.
* A garage that is only one car wide. To remove the first car in we have to take out all the other cars in after it.
* Wearing/Removing Bangles.
* Back/Forward stacks on browsers.
* Support for recursion
* Activation records of method calls.

Stack via array

using System;

namespace stk

{

public class MyStack

{

public int[] st;

public int top = -1;

public int size;

public void stack(int size)

{

this.size = size;

st = new int[this.size];

}

public void push()

{

Console.WriteLine("Enter the element to insert in stack at index {0} : ",top+1); int e = Int32.Parse(Console.ReadLine());

if (top == size-1)

{

throw new Exception("bs krde ");

}

top++;

st[top] = e;

Console.WriteLine("Element inserted\n");

}

public void pop()

{

if (top == - 1)

{

throw new Exception("Stack underflowing");

}

Console.WriteLine("Popped item is : {0} from Index {1}",st[top],top);

top--;

}

public void print()

{

for (int a = 0; a <= top; a++)

{

Console.WriteLine("\t {0}", st[a]);

}

}

}

public class main

{

public static void Main(string[] Args)

{

int ch=0;

MyStack ms = new MyStack();

Console.WriteLine("\n\tStack via Array by initializing position 0 as bottom of the stack \n");

Console.WriteLine("\t\t\tStack(Pushdown store) & Array(static) \n");

Console.Write("Define the Length of Stack :\a\a\a\a ");int size=Int32.Parse(Console.ReadLine());

ms.stack(size);

while (ch != -1)

{

Console.WriteLine("Select :\n1: Push at top\n2: Pop from top\n3: print stack elements\n4: To quit ");

ch = Int32.Parse(Console.ReadLine());

switch (ch)

{

case 1:

ms.push();

break;

case 2:

ms.pop();

break;

case 3:

ms.print();

break;

case 4:

Console.Clear();

break;

}

}

Console.ReadKey();

}

}

}

Stack via link list

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace stack\_via\_linklist

{

//internal is the default access specifiers/modifier for a class in C# programming.

internal class Node

{

internal int data;

internal Node next;

// Constructor to create a new node.Next is by default initialized as null

public Node(int d)

{

data = d;

next = null;

}

}

internal class LinkListStack

{

//head

Node top;

public LinkListStack()

{

this.top = null;

}

public void Push(int value)

{

//newNode = address

Node newNode = new Node(value);

if (top == null)

{

newNode.next = null;

top = newNode;

}

else

{

newNode.next = top;

top = newNode;

}

Console.WriteLine("{0} is pushed to stack", value);

}

internal void Pop()

{

if (top == null)

{

Console.WriteLine("Stack Underflow. Deletion not possible");

return;

}

Console.WriteLine("Item popped is {0}", top.data);

top = top.next;

}

internal void Peek()

{

if (top == null)

{

Console.WriteLine("Stack Underflow.");

return;

}

Console.WriteLine("{0} is on the top of Stack", this.top.data);

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Stack via Linked List");

LinkListStack ll = new LinkListStack();

while (true)

{

Console.Write("\n\t\t\t\t 1: Push \n\t\t\t\t 2: Pop \n\t\t\t\t 3: Peek \n\t\t\t\t 4: Quit \n\n\t\t\tSelect : "); int c = Int32.Parse(Console.ReadLine());

if (c == 1)

{

Console.Write("Enter element to Push : ");

int d = Int32.Parse(Console.ReadLine());

ll.Push(d);

}

else if (c == 2) { ll.Pop(); }

else if (c == 3) { ll.Peek(); }

else if (c == 4) { System.Environment.Exit(1); ; }

else { Console.WriteLine("Invalid selection"); }

}

Console.ReadKey();

}

}

}

Task # 5

QUEUE

Introduction

Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.

The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Operations on Queue:

Mainly the following four basic operations are performed on queue:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

Front: Get the front item from queue.

Rear: Get the last item from queue.

Applications of Queue

Queue is used when things don’t have to be processed immediately, but have to be processed in First In First Out order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

* When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
* When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

Array implementation Of Queue

using System;

namespace que

{

public class MyQueue

{

public int[] q;

public int rear = -1,size,front=0;

public void queue(int size)

{

this.size = size;

q = new int[this.size];

}

public void queue()

{

Console.WriteLine("Enter the element to insert in queue at index {0} : ", rear + 1); int e = Int32.Parse(Console.ReadLine());

if (rear == this.size - 1)

{

throw new Exception("Queue Overflowing ");

}

rear++;

q[rear] = e;

Console.WriteLine("Element inserted\n");

}

public void dequeue()

{

if (front == 0 && rear == -1)

{

throw new Exception("Queue underflowing");

}

else

{

Console.WriteLine("Dequeued item is : {0} from Index {1}\n", q[front], front);

int p = q[front];

front++;

}

}

public void print()

{

int a;

for (a = front; a <= rear; a++)

{

Console.WriteLine("\t {0}", q[a]);

}

}

}

public class main

{

public static void Main(string[] Args)

{

int ch = 0;

MyQueue ms = new MyQueue();

Console.WriteLine("\n\tQueue via Array \n");

Console.WriteLine("\t\t Array(static) \n");

Console.Write("Define the Length of Queue :\a\a\a\a "); int size = Int32.Parse(Console.ReadLine());

ms.queue(size);

while (ch != -1)

{

Console.WriteLine("Select :\n1: Queue at Front\n2: Dequeue from Rear\n3: print queue elements\n4: To quit ");

ch = Int32.Parse(Console.ReadLine());

switch (ch)

{

case 1:

ms.queue();

break;

case 2:

ms.dequeue();

break;

case 3:

ms.print();

break;

case 4:

System.Environment.Exit(1);

break;

}

}

Console.ReadKey();

}

}

}

Link list implementation Of Queue

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace queue\_via\_linklist

{

//internal is the default access specifiers/modifier for a class in C# programming.

internal class Node

{

internal int data;

internal Node next;

// Constructor to create a new node.Next is by default initialized as null

public Node(int d)

{

data = d;

next = null;

}

}

internal class LinkListStack

{

//head

//Node top;

Node rear;

Node front;

//Node temp;

Node del;

public LinkListStack()

{

//rear will work as top

//this.top = null;

this.rear = null;

this.front = null;

//newNode have temp functionality

//this.temp = null;

}

public void EnQueue(int value)

{

//newNode = address

Node newNode = new Node(value);

if (this.rear == null && this.front==null)

{

newNode.next = null;

this.front = this.rear = newNode;

}

else

{

rear.next = newNode;

this.rear = newNode;

}

Console.WriteLine("{0} is Enqueued in Queue", value);

}

internal void DeQueue()

{

if (front == null && rear==null)

{

Console.WriteLine("Queue Underflow. Dequeue not possible");

}

else if (this.front == this.rear)

{

this.del = this.front;

Console.WriteLine("Item deqeued is {0}", front.data);

//front = front.next;

front = rear = null;

}

else

{

//if more than 1 node

this.del = this.front;//to hold address of front

Console.WriteLine("Item deqeued is {0}", front.data);

front = front.next;

int x = del.data;

}

}

internal void Print()

{

this.del = this.front;

while (this.del != null)

{

int n = 1;

Console.WriteLine("{0} : {1} ", n, del.data);

del = del.next;

n++;

Console.WriteLine("{0} : {1} ", n, del.data);

del = del.next;

n++;

Console.WriteLine("{0} : {1} ", n, del.data);

del = del.next;

n++;

}

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Stack via Linked List");

LinkListStack ll = new LinkListStack();

while (true)

{

Console.Write("\n\t\t\t\t 1: Enqueue \n\t\t\t\t 2: Dequeue \n\t\t\t\t 3: Print \n\t\t\t\t 4: Quit \n\n\t\t\tSelect : "); int c = Int32.Parse(Console.ReadLine());

if (c == 1)

{

Console.Write("Enter element to EnQueue : ");

int d = Int32.Parse(Console.ReadLine());

ll.EnQueue(d);

}

else if (c == 2) { ll.DeQueue(); }

else if (c == 3) { ll.Print(); }

else if (c == 4) { System.Environment.Exit(1); ; }

else { Console.WriteLine("Invalid selection"); }

}

Console.ReadKey();

}

}

}

Task # 6

SEARCHING

Introduction To Linear Search

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

using System;

namespace ds\_Labfile\_t6

{

class linear

{

int i, s;

int[] array1 = new int[] { 55,66,77,88,99 };

public void Lin()

{

Console.WriteLine("\n\n\t\t\tLinear Search"); Console.WriteLine("\t\tDefualt Values in Array are \n");

foreach (int value in array1)

{

Console.WriteLine(value);

}

Console.WriteLine("\t\tEnter element to search : "); s = int.Parse(Console.ReadLine());

for (i = 0; i < array1.Length; i++)

{

if (s == array1[i])

{

Console.WriteLine("{0} found at index {1}", s, i);

break;

}

else { Console.WriteLine("{0} not found ", s); }

}

}

}

class Program

{

static void Main(string[] args)

{

linear l = new linear();

l.Lin();

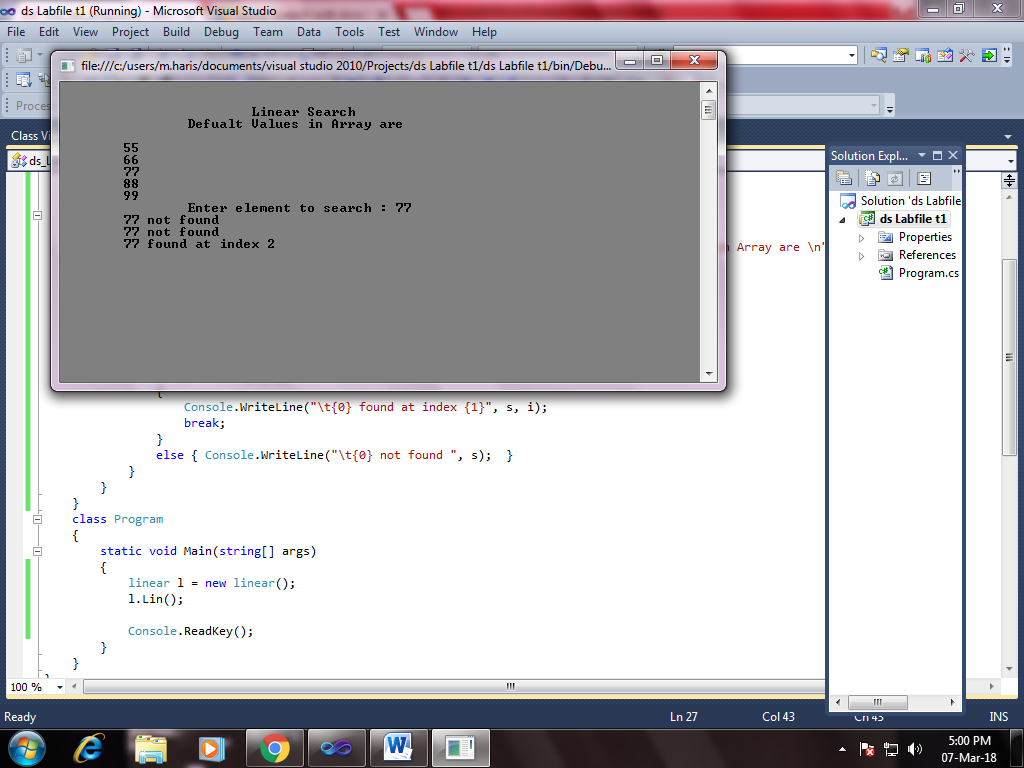
Console.ReadKey();

}

}

}

Console



Introduction To Binary Search

Binary search is a fast search algorithm with run-time complexity of

Ο(log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item.

This process continues on the sub-array as well until the size of the subarray reduces to zero.

using System;

namespace forgetCode

{

class Program

{

public static void Main()

{

int[] a = new int[100];

Console.Write("Number of elements in the array : ");

string s = Console.ReadLine();

int x = Int32.Parse(s);

Console.WriteLine("-----------------------");

Console.WriteLine(" Enter array elements ");

Console.WriteLine("-----------------------");

for (int i = 0; i < x; i++)

{

string s1 = Console.ReadLine();

a[i] = Int32.Parse(s1);

}

Console.WriteLine("--------------------");

Console.WriteLine("Enter Search element");

Console.WriteLine("--------------------");

string s3 = Console.ReadLine();

int x2 = Int32.Parse(s3);

int low = 0;

int high = x - 1;

while (low <= high)

{

int mid = (low + high) / 2;

if (x2 < a[mid])

high = mid - 1;

else if (x2 > a[mid])

low = mid + 1;

else if (x2 == a[mid])

{

Console.WriteLine("-----------------");

Console.WriteLine("Search successful");

Console.WriteLine("-----------------");

Console.WriteLine("Element {0} found at location {1}\n", x2, mid + 1);

return;

}

}

Console.WriteLine("Search unsuccessful");

Console.ReadKey();

}

}

}

Console

