**Data Structures**

**Semester-3**

**Unit-I**

**Overview of data structure, basics of algorithm analysis including running time calculations, abstract data type, arrays, arrays and pointers, multi-dimensional arrays, string processing, general lists and list ADT, list manipulations, single, double and circular lists. Stacks and stack ADT, stack manipulation, prefix, infix and postfix expressions, recursion. Queues and queue ADT, queue manipulation.**

**Unit-II**

**Sparse matrix representation (array and link list representation) and arithmetic (addition, subtraction and multiplication), polynomials and polynomial arithmetic.**

**Trees, properties of trees, binary trees, binary tree traversal, tree manipulation algorithms, expression trees and their usage, binary search trees, AVL trees,**

**heaps and their implementation, priority queues, b-trees, b\* Tree, b+ tree**

**Unit-III**

**Sorting concept, order, stability, selection sorts (straight, heap), insertion sort (straight**

**insertion, shell sort), exchange sort (bubble, quicksort), merge sort (external sorting) (natural merge, balanced merge and polyphase merge). Searching — list search, sequential search, binary search, hashing methods, collision resolution in hashing.**

**Unit-IV**

**Disjoint sets representation, union find algorithm, graphs, graph representation, graph traversals and their implementations (BFS and DFS). Minimum spanning tree algorithms, shortest path algorithms.**

**Guru Tegh Bahadur Institute Of Technology**

**Data structure’s list of practicals**

**Semester - 3**

|  |  |
| --- | --- |
| **Topic** | **Arrays** |
| **1.** | **Insert an element at user defined position in an array of type float (unsorted).**  **Description of program:**   1. **Input an array of float.** 2. **Ask position from the user where the new element has to be inserted.** 3. **Insert the element into the array.** 4. **Print the upgraded array.** |
| **2.** | **Insert an element at user defined position in an array of type float (sorted).**  **Description of program:**   1. **Input an array of float.** 2. **Search for the position where the new element has to be inserted.** 3. **Insert the element into the array.** 4. **Print the upgraded array.** |
| **3.** | **Delete an element from user defined position in an array of type float**  **Description of program:**   1. **Input an array.** 2. **Ask element has to be deleted.** 3. **Search the position of the element.** 4. **Delete the element.** 5. **Print the upgraded array.** |
| **4.** | **Perform linear search on an array.**  **Description of program:**   1. **Read an array of int.** 2. **Input element from user for searching.** 3. **Search the element by passing the array to a function and then returning the position of the element from the function else return -1 if the element is not found.** 4. **Display the position where the element has been found.** |
| **5.** | **Using iteration method, find the element in an array using binary search method.**  **Description of program:**   1. **Read an array** 2. **Input element from user for searching.** 3. **Display the position where the element has been found.** |
| **6.** | **Using recursion method, find the element in an array using binary search method.**  **Description of program:**   1. **Read an array** 2. **Input element from user for searching.** 3. **Display the position where the element has been found, the recursive function should receive the array, position of top and bottom.** |
| **7.** | **Add two polynomials using array.**  **Description of program:**   1. **Input two polynomials from screen use array of structures.** 2. **Add these polynomials and display the output stored in the array of structure form.** |
| **8.** | **Implement sparse matrix using array.**  **Description of program:**   1. **Read a 2d array from the user.** 2. **Store it in the sparse matrix form, use array of structures.** 3. **Print the final array.** |
| **Topic** | **Linked list** |
| **9.** | **Create a linked list with nodes having information about a student and perform following operations on it by creating separate functions with appropriate arguments:**  **Description of program:**   1. **Insertion a new node at specified position.** 2. **Deletion of a node with the roll number of student specified.** 3. **Reverse the linked list.** |
| **10.** | **Create doubly linked list with nodes having information about an employee and perform following operations on it by creating separate functions with appropriate arguments:**  **Description of program:**   1. **Insertion at front.** 2. **Deletion at end.** |
| **11.** | **Create circular linked list having information about an college and perform following operations on it:**  **Description of program:**   1. **Insertion at front.** 2. **Deletion at end.** |
| **Topic** | **Stack** |
| **12.** | **Implement push and pop operations in a stack using an array. The array should be storing the roll numbers of the students in the integer form. Separate functions for display, push and pop should be designed with appropriate arguments. The pop function should return the element which is poped out.** |
| **13.** | **Implement push and pop operations in a stack using linked list. The linked list should contain the information about the student in structure form. Separate functions for display, push and pop should be designed with appropriate arguments. The pop function should return the structure which is poped out.** |
| **Topic** | **Queue** |
| **14.** | **Implement insert and delete operations in a queue using an array. The array should be storing the employee numbers of the employees in the integer form. Separate functions for display, insert and delete should be designed with appropriate arguments.** |
| **15.** | **Implement insert and delete operations in a queue using an array. The array should be storing the employee numbers of the employees in the integer form. Separate functions for display, insert and delete should be designed with appropriate arguments.** |
| **16.** | **Implement insertion and deletion operations on a circular queue using linked list and each node of the linked list should store information about the lab with name of the lab and number of computers in that lab. Separate functions should be designed to insert and display information in the queue.** |
| **Topic** | **Trees** |
| **17.** | **Implement insertion, deletion and display (inorder, preorder and postorder) on binary search tree with the information in the tree about the details of a automobile (type, company, year of make).** |
| **Topic** | **Sorting** |
| **18.** | **Sort the list using following sorting algorithms:**  **56, 77, 23, 99, 68, 11, 9, 29, 33, 45, 10, 87**   1. **Selection sort** 2. **Bubble sort** 3. **Insertion sort** 4. **Quick sort** 5. **Merge sort** 6. **Radix sort** |
| **Topic** | **Graphs** |
| **19.** | **Implement the insertion in a graph and then traversal in graph using bfs and dfs.** |
| **20.** | **Implement single source shortest path algorithm.** |
| **21.** | **Implement all pair shortest path algorithm.** |

**Data Structures Concepts**

**Concept Of Structure Of A Node Of Singly Linked List.**

**The basic structure of a node contains two parts: information part (NAME, ROLL of student) and pointer part (N). The node in the algorithm is represented using a structure student. In a program, the following statements represent a node:**

|  |  |
| --- | --- |
| **Information part of a singly linked list node** |  |

**STRUCTURE OF A NODE**

**struct STUDENT**

**{**

**char NAME[20];**

**int ROLL;**

**struct STUDENT \*N;**

**};**

**Start**

**NULL**

|  |  |
| --- | --- |
| **Gaurav**  **11** |  |

|  |  |
| --- | --- |
| **Amit**  **22** |  |

|  |  |
| --- | --- |
| **Vishal**  **33** |  |

|  |  |
| --- | --- |
| **Karan**  **44** |  |

**Algorithm To Insert A Node In The End Of A Singly Linked List.**

**INSERT\_IN\_L\_LIST()**

**This algorithm inserts a node at the end of a linked list. ‘FRESH’ is a new node to be inserted into the end of the linked list. ‘START’ pointer points towards first node of the linked list (if present).**

1. **Allocate memory to a node FRESH.**
2. **Check if FRESH == NULL, then:**

**Print "NO MEMORY RECEIVED" and return.**

1. **Set NAME[FRESH] := DATA\_NAME.**

**Set ROLL[FRESH] := DATA\_ROLL.**

**Set N[FRESH] := NULL.**

1. **Check if START = NULL, then: [If linked list is absent]**

**Set START := FRESH.**

1. **Else [If linked list is present]**
2. **Set LOC := START.**
3. **Repeat while N[LOC] != NULL.**

**Set LOC := N[LOC].**

**[End of while loop of step (a)]**

1. **Set N[LOC] := FRESH.**

**[End of step 4 if structure]**

1. **Exit.**

**Algorithm To Traverse A Singly Linked List.**

**DISPLAY\_L\_LIST()**

**This algorithm traverses and displays nodes of a linked list. ‘START’ pointer points towards first node of the linked list (if present).**

1. **Check if START = NULL, then: [If linked list is absent]**

**Print "THE LIST IS EMPTY.” and return.**

1. **Else [If linked list is present]**
2. **Set LOC := START.**
3. **Repeat while LOC != NULL.**

**Print NAME[LOC] and ROLL[LOC].**

**Set LOC := N[LOC].**

**[End of while loop of step (a)]**

**[End of step 1 if structure]**

1. **Exit.**

**Algorithm To Search a ITEM In A Singly Linked List.**

**SEARCH\_L\_LIST()**

**This algorithm traverses and displays nodes of a linked list. ‘START’ pointer points towards first node of the linked list (if present).**

1. **Check if START = NULL, then: [If linked list is absent]**

**Print "THE LIST IS EMPTY.” and return.**

1. **Else [If linked list is present]**
2. **Set LOC := START.**
3. **Repeat while LOC != NULL.**

**(c) If ROLL[LOC] = ITEM, then:**

**Set PTR := LOC and return**

**[End of step (c) if structure]**

**Set LOC := N[LOC].**

**[End of while loop of step (a)]**

**[End of step 1 if structure]**

1. **Exit.**

**Algorithm To Insert New Node After A Given Node In Linked List.**

**INSERT\_AFTER()**

**‘AFTER\_ROLL’ is the value after which new node ‘A\_FRESH’ will be inserted. The node containing value of ‘AFTER\_ROLL’ will be pointed by pointer ‘ALOC’.**

1. **Allocate memory to a node A\_FRESH.**
2. **Check if A\_FRESH == NULL, then:**

**Print "NO MEMORY RECEIVED" and return.**

1. **Set NAME[A\_FRESH] := DATA\_NAME.**

**Set ROLL[A\_FRESH] := DATA\_ROLL.**

**Set N[A\_FRESH] := NULL.**

1. **Set ALOC := START.**
2. **Repeat while N[ALOC] != NULL.**
3. **If ROLL[ALOC] = AFTER\_ROLL, then:**

**Goto step 6.**

**[End of step (a) if structure]**

1. **Set ALOC := N[ALOC].**

**[End of while loop of step 5]**

1. **Set N[A\_FRESH] := N[ALOC].**
2. **Set N[ALOC] := A\_FRESH.**
3. **Exit.**

**Algorithm To Insert New Node Before A Given Node In Linked List.**

**INSERT\_BEFORE()**

**‘BEFORE\_ROLL’ is the value before which new node ‘B\_FRESH’ will be inserted. The node containing value of ‘BEFORE\_ROLL’ will be pointed by pointer ‘BLOC’.**

1. **Allocate memory to a node B\_FRESH.**
2. **Check if B\_FRESH == NULL, then:**

**Print "NO MEMORY RECEIVED" and return.**

1. **Set NAME[B\_FRESH] := DATA\_NAME.**

**Set ROLL[B\_FRESH] := DATA\_ROLL.**

**Set N[B\_FRESH] := NULL.**

1. **Set PBLOC := BLOC.**
2. **Set BLOC := START.**
3. **Repeat while N[BLOC] != NULL.**
4. **If ROLL[BLOC] = BEFORE\_ROLL, then:**

**Goto step 7.**

**[End of step (a) if structure]**

1. **Set PBLOC := BLOC and BLOC := N[BLOC].**

**[End of while loop of step 6]**

1. **Set N[B\_FRESH] := BLOC.**
2. **Check if N[PBLOC] = NULL, then:**

**Set START := B\_FRESH.**

**Else**

**Set N[PBLOC] := B\_FRESH.**

**[End of step 8 if structure]**

1. **Exit.**

**Algorithm On Deletion Of First Node From Linked List.**

**DELETE\_BEGIN()**

1. **Check if START = NULL, then: [If linked list is absent]**

**Print "THE LIST IS EMPTY.” and return.**

1. **Else [If linked list is present]**
2. **Set LOC := START.**
3. **Set START := N[LOC].**
4. **Free the memory occupied by LOC.**

**[End of step 1 if structure]**

1. **Exit**

**Algorithm On Deletion Of Last Node From Linked List.**

**DELETE\_LAST()**

1. **Check if START = NULL, then: [If linked list is absent]**

**Print "THE LIST IS EMPTY.” and return.**

1. **Else [If linked list is present]**
2. **Set LOC := START.**
3. **Repeat while LOC != NULL.**

**Set PLOC := LOC and LOC := N[LOC].**

**[End of while loop of step (a)]**

1. **Set N[PLOC] := NULL.**
2. **Free memory occupied by LOC.**

**[End of step 1 if structure]**

1. **Exit.**

**Algorithm On Concept Of Delete After A Given Node In Linked List.**

**DELETE\_AFTER()**

**‘AFTER\_ROLL’ is the value after which the node will be deleted. The node containing value of ‘AFTER\_ROLL’ will be pointed by pointer ‘LOC’.**

1. **Check if START = NULL, then: [If linked list is absent]**

**Print "THE LIST IS EMPTY.” and return.**

1. **Else [If linked list is present]**
2. **Set POS := START.**
3. **Repeat while POS != NULL.**

**If ROLL[POS] = AFTER\_ROLL, then:**

**Goto step 3.**

**[End of step (a) if structure]**

1. **Set POS := N[POS].**

**[End of while loop of step (a)]**

1. **Set LOC := N[POS].**
2. **N[POS] := N[LOC].**
3. **Free memory occupied by LOC.**

**[End of step 1 if structure]**

1. **Exit.**

**Algorithm On Concept Of Delete Before A Given Node In A Linked List.**

**DELETE\_BEFORE()**

**‘BEFORE\_ROLL’ is the value before which node will be deleted. The node containing value of ‘BEFORE\_ROLL’ will be pointed by pointer ‘POS’.**

1. **Check if START = NULL, then: [If linked list is absent]**

**Print "THE LIST IS EMPTY.” and return.**

1. **Else [If linked list is present]**
2. **Set PLOC := START, LOC := START and POS := START.**
3. **Repeat while POS != NULL.**

**If ROLL[POS] = BEFORE\_ROLL, then:**

**Goto step 3.**

**[End of step (a) if structure]**

1. **Set PLOC := LOC.**
2. **Set LOC := POS.**
3. **Set POS := N[POS].**

**[End of while loop of step (a)]**

1. **Set N[PLOC] := POS.**
2. **Free memory occupied by LOC.**

**[End of step 1 if structure]**

1. **Exit.**

**Algorithm On Concept Of Reversal Of A Linked List.**

1. **Set PREV := NULL.**
2. **Repeat steps (a) to (d) while(LOC)**
3. **Set** **TEMP := N[LOC].**
4. **Set N[LOC] := PREV.**
5. **Set PREV := LOC.**
6. **Set** **LOC := TEMP**

**[End of while loop of step 2]**

1. **Set LOC := PREV.**
2. **Exit.**

**Given Only A Pointer To The Node To Be Deleted In A Singly Linked, Discuss Function On How To Remove That Node.**

**DeleteNode(NODE LOC)**

1. **Check if(LOC), then:**
2. **Set N[LOC] := N[N[LOC]].**
3. **Free memory pointed by LOC.**
4. **Exit.**

**Discuss Logic To Detect A Loop In A Singly Linked List.**

**STEPS:**

1. **Start two pointers at the head of the list.**
2. **Loop infinitely.**
3. **If the fast pointer reaches a NULL pointer return that the list is NULL terminated.**
4. **If the fast pointer moves onto or over the slow pointer return that there is a cycle.**
5. **Advances the slow pointer one node.**
6. **Advances the fast pointer two node.**

**Algorithm:**

**Two pointers are used in this algorithm “fast” and “slow”. “fast” pointer will move twice faster than “slow” pointer.**

**int DetermineTermination(NODE \*head)**

1. **Set fast := slow := START;**
2. **Repeat steps (a) to (c) while(1). [Infinite Loop]**
3. **Check** **if(!fast or !N[fast]), then: return 0.**
4. **Else Check if (fast==slow or N[fast]==slow), then:**

**return 1. [Loop Detected]**

1. **Else set slow := N[slow] and fast := N[N[fast]].**

**[End of if structure of step (a)]**

**[End of while loop of step 2]**

1. **Exit.**

**Concept Of Structure Of A Node Of Circular Linked List.**

**The basic structure of a node contains two parts: information part (NAME, ROLL of student) and pointer part (N). The node in the algorithm is represented using a structure student. In a program, the following statements represent a node:**

|  |  |
| --- | --- |
| **Information part of a circular linked list node** |  |

**STRUCTURE OF A NODE**

**struct STUDENT**

**{**

**char NAME[20];**

**int ROLL;**

**struct STUDENT \*N;**

**};**

**Start**

|  |  |
| --- | --- |
| **Gaurav**  **11** |  |

|  |  |
| --- | --- |
| **Amit**  **22** |  |

|  |  |
| --- | --- |
| **Vishal**  **33** |  |

|  |  |
| --- | --- |
| **Karan**  **44** |  |

**Algorithm To Insert A Node In The End Of A Circular Linked List.**

**INSERT\_IN\_L\_LIST()**

**This algorithm inserts a node at the end of a linked list. ‘FRESH’ is a new node to be inserted into the end of the linked list. ‘START’ pointer points towards first node of the linked list (if present).**

1. **Allocate memory to a node FRESH.**
2. **Check if FRESH == NULL, then:**

**Print "NO MEMORY RECEIVED" and return.**

1. **Set NAME[FRESH] := DATA\_NAME.**

**Set ROLL[FRESH] := DATA\_ROLL.**

**Set N[FRESH] := NULL.**

1. **Check if START = NULL, then: [If linked list is absent]**

**Set N[FRESH] : FRESH and START := FRESH.**

1. **Else [If linked list is present]**
2. **Set LOC := START.**
3. **Repeat while N[LOC] != START.**

**Set LOC := N[LOC].**

**[End of while loop of step (a)]**

1. **Set N[FRESH] := START and N[LOC] := FRESH.**

**[End of step 4 if structure]**

1. **Exit.**

**Algorithm To Traverse A Circular Linked List.**

**DISPLAY\_L\_LIST()**

**This algorithm traverses and displays nodes of a linked list. ‘START’ pointer points towards first node of the linked list (if present).**

1. **Check if START = NULL, then: [If linked list is absent]**

**Print "THE LIST IS EMPTY.” and return.**

1. **Else [If linked list is present]**
2. **Set LOC := START.**
3. **Do**
4. **Print NAME[LOC] and ROLL[LOC].**

**Set LOC := N[LOC].**

1. **Repeat while LOC != START.**

**[End of while loop of step (a)]**

**[End of step 1 if structure]**

1. **Exit.**

**Concept Of Structure Of A Node Of Doubly Linked List.**

**The basic structure of a node contains three parts: information part (NAME, ROLL of student), next pointer part (NXT) and previous pointer part (PRV). The node in the algorithm is represented using a structure student. In a program, the following statements represent a node:**

**struct STUDENT**

|  |  |  |
| --- | --- | --- |
|  | **Information part of a doubly linked list node** |  |

**STRUCTURE OF A NODE**

**{ int ROLL;**

**char NAME[100];**

**struct STUDENT \*NXT;**

**struct STUDENT \*PRV;**

**};**

**Head**

**NULL**

|  |  |  |
| --- | --- | --- |
|  | **Karan**  **44** |  |

|  |  |  |
| --- | --- | --- |
|  | **Dev**  **33** |  |

|  |  |  |
| --- | --- | --- |
|  | **Amit**  **22** |  |

|  |  |  |
| --- | --- | --- |
|  | **Kapil**  **11** |  |

**Tail**

**Algorithm To Insert A Node In The End Of A Doubly Linked List.**

**INSERT\_IN\_DLL()**

**This algorithm inserts a node at the end of a doubly linked list. ‘FRESH’ is a new node to be inserted into the end of the linked list. ‘HEAD’ pointer points towards first node of the linked list (if present) and ‘TAIL’ pointer points towards last node of the linked list (if present).**

1. **Allocate memory to a node FRESH.**
2. **Check if FRESH == NULL, then:**

**Print "NO MEMORY RECEIVED" and return.**

1. **Set NAME[FRESH] := DATA\_NAME.**

**Set ROLL[FRESH] := DATA\_ROLL.**

**Set NXT[FRESH] := NULL.**

**Set PRV[FRESH] := NULL.**

1. **Check if HEAD = NULL and TAIL = NULL , then:**

**Set HEAD := FRESH and TAIL := FRESH.**

1. **Else [If linked list is present]**
2. **Set PRV[FRESH] := TAIL.**
3. **Set NXT[TAIL] := FRESH.**
4. **Set TAIL := FRESH.**

**[End of step 4 if structure]**

1. **Exit.**

**Algorithm To Display Doubly Linked List (Forward)**

1. **Repeat steps 2 and 3 while HEAD != NULL.**
2. **Apply process to ROLL[HEAD],NAME[HEAD].**
3. **Set HEAD := NXT[HEAD].**

**[End of step 1 while loop]**

1. **Exit.**

**Algorithm To Display Doubly Linked List (Reverse)**

1. **Repeat steps 2 and 3 while TAIL != NULL.**
2. **Apply process to ROLL[TAIL],NAME[TAIL].**
3. **Set TAIL := PRV[TAIL].**

**[End of step 1 while loop]**

1. **Exit.**

**Algorithm To Delete A Node After A Given Node In A Doubly Linked List**

**DELETE\_AFTER\_DLL()**

**‘LOC’ points to the node after which deletion is to be performed. ‘AFTER\_ROLL’ is the value to be searched for.**

1. **Set LOC := HEAD.**
2. **Repeat steps 3 to 5 while NXT[LOC] != NULL.**
3. **Check if ROLL[LOC] = AFTER\_ROLL, then:**
4. **Goto step 6**

**[End of if structure of step 3]**

1. **Set LOC := NXT[LOC]**

**[End of while loop of step 2]**

1. **Check if NXT[NXT[LOC]] = NULL, then:**
2. **Set PTR := NXT[LOC].**
3. **Set NXT[LOC] := NULL.**
4. **Set TAIL := LOC.**
5. **Free the memory allocated by PTR.**
6. **Else**
7. **PTR := NXT[LOC].**
8. **NXT[LOC] := NXT[PTR].**
9. **PRV[NXT[PTR]] := LOC.**
10. **Free the memory allocated by PTR.**

**[End of if structure of step 6]**

1. **Exit.**

**Algorithm To Insert A Node After A Given Node In A Doubly Linked List**

**INSERT\_AFTER\_DLL()**

**‘LOC’ points to the node after which insertion is to be performed. ‘AFTER\_ROLL’ is the value to be searched for.**

1. **Allocate memory to AFRESH.**
2. **Set ROLL[AFRESH] := data\_roll and NAME[AFRESH] := data\_name.**
3. **Set NXT[AFRESH] := NULL and PRV[AFRESH] := NULL.**
4. **Set LOC := HEAD.**
5. **Repeat steps 6 and 7 while NXT[LOC] != NULL.**
6. **Check if ROLL[LOC] = AFTER\_ROLL, then:**

**Goto step 6.**

**[End of if structure of step 6]**

1. **Set NXT[LOC] := LOC.**

**[End of while loop of step 5]**

1. **Check if NXT[LOC] = NULL, then:**
2. **Set NXT[LOC] := AFRESH.**
3. **Set PRV[AFRESH] := TAIL.**
4. **TAIL := AFRESH.**
5. **Else**
6. **Set PRV[AFRESH] := LOC.**
7. **Set NXT[AFRESH] := NXT[LOC].**
8. **PRV[NXT[LOC]] := AFRESH.**
9. **NXT[LOC] := AFRESH.**

**[End of if structure of step 8]**

1. **Exit.**

**Algorithm on Realization Of Stack Concept Using Array.**

**The algoritm implement the concept of stack (LIFO). Here, ‘MAX’ is set to 10 which are total number of elements of the stack. The value of ‘full’ is set to MAX-1, initially ‘top’ is set to -1 (Note: In this algo. the first element of the stack will be stored at position 0.)**

**Algorithm On Realization Of Stack Concept Using Array To Insert An Element In The Stack (PUSH).**

**PUSH(int ele)**

**The new data to be added in the stack is represented by ‘ele’.**

1. **Check if top = full, then:**

**Print “OVERFLOW ! NO DATA CAN BE INSERTED" and return.**

1. **Else set top := top + 1 and set top[stack] := ele.**

**[End of if structure in step 1]**

1. **Exit.**

##### Push Operation

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** |  |  | **9** |  |  | **9** |  |  | **9** |  |  | **9** |  |
| **8** |  |  | **8** |  |  | **8** |  |  | **8** |  |  | **8** |  |
| **7** |  |  | **7** |  |  | **7** |  |  | **7** |  |  | **7** |  |
| **6** |  |  | **6** |  |  | **6** |  |  | **6** |  |  | **6** |  |
| **5** |  |  | **5** |  |  | **5** |  |  | **5** |  |  | **5** |  |
| **4** |  |  | **4** |  |  | **4** |  |  | **4** |  |  | **4** |  |
| **3** |  |  | **3** |  |  | **3** |  |  | **3** |  |  | **3** | **44** |
| **2** |  |  | **2** |  |  | **2** |  |  | **2** | **33** |  | **2** | **33** |
| **1** |  |  | **1** |  |  | **1** | **22** |  | **1** | **22** |  | **1** | **22** |
| **0** |  |  | **0** | **11** |  | **0** | **11** |  | **0** | **11** |  | **0** | **11** |

#### TOP = -1 TOP = 0 TOP = 1 TOP = 2 TOP = 3

**TOP = TOP + 1**

**STACK[TOP] = ELE**

**Algorithm On Realization Of Stack Concept Using Array To Remove An Element From Stack (POP).**

**POP()**

1. **Check if top = -1, then:**

**Print “UNDERFLOW ! NO DATA TO DELETE" and return.**

1. **Else set top := top - 1**

**[End of if structure in step 1]**

1. **Exit.**

##### Pop Operation

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** |  |  | **9** |  |  | **9** |  |  | **9** |  |  | **9** |  |
| **8** |  |  | **8** |  |  | **8** |  |  | **8** |  |  | **8** |  |
| **7** |  |  | **7** |  |  | **7** |  |  | **7** |  |  | **7** |  |
| **6** |  |  | **6** |  |  | **6** |  |  | **6** |  |  | **6** |  |
| **5** |  |  | **5** |  |  | **5** |  |  | **5** |  |  | **5** |  |
| **4** |  |  | **4** |  |  | **4** |  |  | **4** |  |  | **4** |  |
| **3** | **44** |  | **3** | **44** |  | **3** | **44** |  | **3** | **44** |  | **3** |  |
| **2** | **33** |  | **2** | **33** |  | **2** | **33** |  | **2** | **33** |  | **2** |  |
| **1** | **22** |  | **1** | **22** |  | **1** | **22** |  | **1** | **22** |  | **1** |  |
| **0** | **11** |  | **0** | **11** |  | **0** | **11** |  | **0** | **11** |  | **0** |  |

**TOP=3 TOP=2 TOP=1 TOP=0 TOP=-1**

**TOP = TOP – 1**

**Algorithm On Realization Of Push into Stack Using Linked List.**

**PUSH(int ele)**

**Here pointer ‘TOP’ points to the node at top position, i.e. where the insertion is to take place.**

1. **Allocate memory to a node FRESH.**
2. **Check if FRESH = NULL, then:**

**Print "NO MEMORY RECEIVED" and return.**

1. **Set INFO[FRESH] := ITEM.**

**Set N[FRESH] := NULL.**

1. **Check if TOP = NULL, then: [If stack linked list is absent]**

**Set TOP := FRESH.**

1. **Else [If stack linked list is present]**
2. **Set LOC := TOP**
3. **Set TOP := FRESH.**
4. **Set N[FRESH] := LOC.**

**[End of step 4 if structure]**

1. **Exit.**

**Push Operation**

**NULL**

**TOP**

**11**

TOP

NULL

**NULL**

**TOP**

**22**

**33**

TOP

NULL

**11**

**TOP**

**22**

**11**

NULL

TOP

**NULL**

**Push Operation In Detail**

**Case 1:**

**FRESH**

**99**

NULL

\*N

FRESH

**NULL**

**TOP = NULL**

**Case 2:**

**FRESH**

**99**

NULL

\*N

FRESH

**NULL**

**TOP**

**NULL**

**90**

NULL

TOP

**11**

**22**

\*N

**Algorithm On Realization Of Pop into Stack Using Linked List.**

**POP()**

1. **Check if TOP = NULL, then: [If stack linked list is absent]**

**Print "UNDERFLOW, THE STACK IS EMPTY.” and return.**

1. **Else [If stack linked list is present]**
2. **Set PTR := TOP.**
3. **Set TOP := N[TOP].**
4. **Free the memory pointed by PTR.**

**[End of step 1 if structure]**

1. **Exit.**

**Pop Operation**

**NULL**

**TOP**

**22**

**33**

TOP

NULL

**11**

**22**

**11**

NULL

TOP

**NULL**

**TOP**

**NULL**

**TOP**

**11**

TOP

NULL

**Pop Operation In Detail**

**TOP**

**NULL**

**90**

NULL

TOP

**11**

**22**

NULL

TOP

##### Insert Operation in Array Queue

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** |  |  | **9** |  |  | **9** |  |  | **9** |  |  | **9** |  |
| **8** |  |  | **8** |  |  | **8** |  |  | **8** |  |  | **8** |  |
| **7** |  |  | **7** |  |  | **7** |  |  | **7** |  |  | **7** |  |
| **6** |  |  | **6** |  |  | **6** |  |  | **6** |  |  | **6** |  |
| **5** |  |  | **5** |  |  | **5** |  |  | **5** |  |  | **5** |  |
| **4** |  |  | **4** |  |  | **4** |  |  | **4** |  |  | **4** |  |
| **3** |  |  | **3** |  |  | **3** |  |  | **3** |  |  | **3** | **44** |
| **2** |  |  | **2** |  |  | **2** |  |  | **2** | **33** |  | **2** | **33** |
| **1** |  |  | **1** |  |  | **1** | **22** |  | **1** | **22** |  | **1** | **22** |
| **0** |  |  | **0** | **11** |  | **0** | **11** |  | **0** | **11** |  | **0** | **11** |

FRONT = -1 FRONT = 0 FRONT = 0 FRONT = 0 FRONT = 0

**REAR = -1 REAR = 0 REAR = 1 REAR = 2 REAR = 3**

**REAR++**

#### QUEUE[REAR]=ELE

##### Delete Operation from Array Queue

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **9** |  |  | **9** |  |  | **9** |  |  | **9** |  |  | **9** |  |
| **8** |  |  | **8** |  |  | **8** |  |  | **8** |  |  | **8** |  |
| **7** |  |  | **7** |  |  | **7** |  |  | **7** |  |  | **7** |  |
| **6** |  |  | **6** |  |  | **6** |  |  | **6** |  |  | **6** |  |
| **5** |  |  | **5** |  |  | **5** |  |  | **5** |  |  | **5** |  |
| **4** |  |  | **4** |  |  | **4** |  |  | **4** |  |  | **4** |  |
| **3** | **44** |  | **3** | **44** |  | **3** | **44** |  | **3** | **44** |  | **3** |  |
| **2** | **33** |  | **2** | **33** |  | **2** | **33** |  | **2** | **33** |  | **2** |  |
| **1** | **22** |  | **1** | **22** |  | **1** | **22** |  | **1** | **22** |  | **1** |  |
| **0** | **11** |  | **0** | **11** |  | **0** | **11** |  | **0** | **11** |  | **0** |  |

**FRONT = 0 FRONT = 1 FRONT = 2 FRONT = 3 FRONT = -1**

**REAR = 3 REAR = 3 REAR = 3 REAR = 3 REAR = -1**

**FRONT++**

**Algorithm On Insertion Into A Queue Using Concept Of Array.**

**INSERT\_IN\_QUEUE(int ele)**

**Here the name of the array is “queue”, with 10 elements. Value of full is set to 10 and the starting position of the “queue” is 0. Initially “front” and “rear” is set to -1.**

1. **Check if rear = full, then:**

**Print "OVERFLOW ! NO DATA CAN BE INSERTED " and return.**

1. **Else Check if rear = -1, then:**

**Set front := 0 and rear := 0 and goto step 4.**

1. **Else**

**Set rear := rear + 1 goto step 4.**

**[End of if structure of step 1]**

1. **Set rear[queue] := ele.**
2. **Exit.**

**Algorithm On Deletion From A Queue Using Concept Of Array.**

**DELETE\_FROM\_QUEUE()**

1. **Check if front = -1,then:**

**Printf "UNDERFLOW ! EMPTY QUEUE" and return.**

1. **Else**
2. **Check if front = rear,then:**

**Set front = -1 and rear = -1.**

1. **Else**

**Set front++.**

**[End of if structure of step a]**

**[End of if structure of step 1]**

1. **Exit.**

##### Insert Operation into Linked Queue

**REAR**

**FRONT**

**NULL;**

**11**

Tf

NULL

**REAR**

**FRONT**

**NULL;**

**11**

**22**

NULL

FRONT

**REAR**

**FRONT**

**22**

**11**

FRONT

NULL

**90**

**NULL;**

##### Delete Operation from Linked Queue

**22**

**11**

FRONT

**REAR**

NULL

**90**

**NULL;**

**FRONT**

**22**

**90**

**REAR**

FRONT

**FRONT**

**NULL;**

NULL

**REAR**

**FRONT**

**NULL;**

**90**

FRONT

NULL

**Algorithm On Realization Of Insertion into Queue Using Linked List.**

**INSERTION(int ele)**

**Here pointer ‘FRONT’ points to the node at first position and “REAR” points to last node of the queue.**

1. **Allocate memory to a node FRESH.**
2. **Check if FRESH = NULL, then:**

**Print "NO MEMORY RECEIVED" and return.**

1. **Set INFO[FRESH] := ITEM.**

**Set N[FRESH] := NULL.**

1. **Check if FRONT = NULL, then: [If queue linked list is absent]**

**Set FRONT := FRESH and REAR := FRESH.**

1. **Else [If queue linked list is present]**

**Set N[REAR] := FRESH and REAR := FRESH.**

1. **Exit.**

**Insert Operation In Detail**

**FRONT = NULL**

**REAR = NULL**

**CASE 1:**

**FRESH**

**11**

NULL

\*N

FRESH

**NULL;**

FRONT = NULL

**CASE 2:**

**FRESH**

**11**

NULL

\*N

FRESH

**NULL;**

**22**

**11**

FRONT

**REAR**

NULL

**90**

**NULL;**

**FRONT**

\*N

**Algorithm On Realization Of Deletion From Queue Using Linked List.**

**DELETE\_FROM\_QUEUE()**

1. **Check if FRONT = NULL, then:**

**Print "UNDERFLOW ! NO DATA PRESENT IN QUEUE" and return.**

1. **Else**
2. **Set PTR := FRONT.**
3. **Set FRONT := N[FRONT];**
4. **free the memory pointed by PTR.**
5. **Exit.**

**Delete Operation In Detail**

**22**

**11**

FRONT

**REAR**

NULL

**90**

**NULL;**

**FRONT**

**PTR**

**22**

**11**

FRONT

**REAR**

NULL

**90**

PTR

**NULL;**

**FRONT**

PTR

**PTR**

**22**

**11**

FRONT

**REAR**

NULL

**90**

X

**NULL;**

**FRONT**

**Drawback Of Array Queue**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **Front** |  |  | **Rear** |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|  |  |  |  | **22** | **11** | **33** | **55** |

##### Insert Operation in Circular Queue

**11**

**20**

**41**

**77**

**15**

**56**

**98**

**61**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**Front**

**Rear**

**Case 1 : Front=0 and Rear=Max-1**

**Front**

**--**

**--**

**41**

**77**

**15**

**56**

**98**

**61**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**Rear**

**Case 3 : Rear=Max-1**

**Front**

**--**

**--**

**41**

**77**

**15**

**56**

**--**

**--**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**Rear**

**Case 4**

**--**

**--**

**--**

**--**

**--**

**--**

**--**

**--**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**Case 5 : Front=Rear=-1**

**Case 2 : Front=Rear+1**

**11**

**20**

**41**

**77**

**15**

**56**

**98**

**61**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**Rear**

**Front**

##### Delete Operation From Circular Queue

**Rear**

**Front**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**11**

**20**

**41**

**--**

**--**

**--**

**--**

**61**

**Case 1 : Front=Max-1**

**--**

**--**

**--**

**--**

**56**

**--**

**--**

**--**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**Rear**

**Front**

**Case 2 : Front=Rear**

**--**

**--**

**--**

**--**

**--**

**--**

**--**

**--**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**Case 3 : Front=Rear=-1**

**Front**

**--**

**--**

**41**

**77**

**15**

**56**

**--**

**--**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**Rear**

**Case 4**

**Algorithm On Insertion Into A Circular Queue Using Concept Of Array.**

**Value of MAX is set to 8, “arr” is the arry used to implement circular queue.**

**INITIALIZE\_QUEUE()**

1. **Repeat for i=0 to MAX:**

**Set arr[i] := 0.**

**[End of for loop of step 1]**

1. **Exit.**

**ADD\_ELEMENT(front, rear, ele)**

1. **Check if((rear=MAX-1 and front=0) or (rear+1=front)), then:**

**Print "OVERFLOW ! QUEUE IS FULL" and return.**

**[End of if structure of step 1]**

1. **Check if rear=MAX-1, then:**
2. **Set rear := 0.**
3. **Else if rear != MAX-1, then:**
4. **Check if rear=-1, then: set rear := 0 and front := 0.**
5. **Else set rear := rear + 1.**

**[End of if structure of step b]**

**[End of if structure of step 2]**

1. **Set rear[arr] := ele.**
2. **Exit.**

**Algorithm On Deletion From A Circular Queue Using Concept Of Array.**

**DELETE\_ELEMENT(front, rear)**

1. **Check if front = -1, then:**

**Print "UNDERFLOW ! QUEUE IS EMPTY" and return.**

**[End of if structure of step 1]**

1. **Check if front = rear, then: set front := -1 and rear := -1.**
2. **Else if front != rear, then:**
3. **Check if front = MAX-1, then: set front := 0.**
4. **Else set front = front + 1.**

**[End of if structure of step a]**

**[End of if structure of step 2]**

**Sorting: Means Arranging Of Data in Ascending / Descending Order.**

**Algorithm to Sort an Array Using Bubble Sort.**

**BUBBLE\_SORT(int a[], int len)**

**Here array ‘a’ is an array of ‘len-1’ elements.**

1. **Repeat step 2 for i := 0 to len-1**

**2. Repeat steps (a) and (b) for j := 0 to len-i-1**

**(a) if a[j] > a[j+1] then**

**(b) swap a[j] and a[j+1]**

**[End of step 2 loop]**

**[End of step 1 loop]**

1. **Exit**

**Processing Of Bubble Sort:**

**Total Elements = 6**

**Input Array**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **33** | **22** | **55** | **44** | **66** | **11** |

**Value of i=0 Value of i=0**

**Value of j=0 Value of j=1**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **55** | **44** | **66** | **11** | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **55** | **44** | **66** | **11** | |

**Value of i=0 Value of i=0**

**Value of j=2 Value of j=3**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **44** | **55** | **66** | **11** | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **44** | **55** | **66** | **11** | |

**Value of i=0 Value of i=1**

**Value of j=4 Value of j=0**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **44** | **55** | **11** | **66** | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **44** | **55** | **11** | **66** | |

**Value of i=1 Value of i=1**

**Value of j=1 Value of j=2**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **44** | **55** | **11** | **66** | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **44** | **55** | **11** | **66** | |

**Value of i=1 Value of i=2**

**Value of j=3 Value of j=0**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **44** | **11** | **55** | **66** | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **44** | **11** | **55** | **66** | |

**Value of i=2**

**Value of i=2**

**Value of j=1 Value of j=2**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **44** | **11** | **55** | **66** | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **11** | **44** | **55** | **66** | |

**Value of i=3 Value of i=3**

**Value of j=0 Value of j=1**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **33** | **11** | **44** | **55** | **66** | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **22** | **11** | **33** | **44** | **55** | **66** | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Value of i=4**  **Value of j=0** | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** | | **11** | **22** | **33** | **44** | **55** | **66** | |

**Algorithm To Find Smallest Element Of An Array.**

**FIND\_SMALLEST(int a[], int start, int len)**

**Here array ‘a’ is an array of ‘len-1’ elements and ‘start’ is set to 0 which is the index of first element of the array.**

1. **Set start := 0**
2. **Set smallest := a[start]**
3. **Set pos := start**
4. **Repeat steps (a), (b) and (c) for j := start to len-1**
5. **If smallest > a[j] then**
6. **Set smallest := a[j]**
7. **Set pos := j**

**[End of step 4 loop]**

1. **Exit**

**Processing Of Finding the Smallest Element:**

**Total Elements = 6**

**Input Array**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **55** | **22** | **33** | **11** | **44** | **66** |

**Value Of j = 0**

**Before Comparison: a[0] = 55 , Smallest = 55 , Position = 0**

**After Comparison: a[0] = 55 , Smallest = 55 , Position = 0**

**j = 1**

**Before Comparison: a[1] = 22 , Smallest = 55 , Position = 0**

**After Comparison: a[1] = 22 , Smallest = 22 , Position = 1**

**j = 2**

**Before Comparison: a[2] = 33 , Smallest = 22 , Position = 1**

**After Comparison: a[2] = 33 , Smallest = 22 , Position = 1**

**j = 3**

**Before Comparison: a[3] = 11 , Smallest = 22 , Position = 1**

**After Comparison: a[3] = 11 , Smallest = 11 , Position = 3**

**j = 4**

**Before Comparison: a[4] = 44 , Smallest = 11 , Position = 3**

**After Comparison: a[4] = 44 , Smallest = 11 , Position = 3**

**j = 5**

**Before Comparison: a[5] = 66 , Smallest = 11 , Position = 3**

**After Comparison: a[5] = 66 , Smallest = 11 , Position = 3**

**Smallest Element = 11**

**Smallest Element Position = 3**

**Algorithm to Sort an Array Using Selection Sort.**

**SELECTION\_SORT(int a[], int len)**

**Here array ‘a’ is an array of ‘len-1’ elements.**

1. **Set pos := 0**
2. **Repeat step 3 to 7 for i :=0 to len-1**
3. **Set start := i**
4. **Set smallest := a[start]**
5. **Set pos := start**
6. **Repeat steps (a), (b) and (c) for j := start to len-1**
7. **If smallest > a[j] then**
8. **Set smallest := a[j]**
9. **Set pos := j**

**[End of step 6 loop]**

1. **Interchange element a[i] with a[pos]**

**[End of step 2 loop]**

1. **Exit**

**Processing Of Selection Sort:**

**Total Elements = 6**

**Input Array**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **44** | **22** | **66** | **55** | **11** | **33** |

**i = 0**

**a[0] = 44**

**Smallest = 11**

**Position = 4**

**Array (Before Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **44** | **22** | **66** | **55** | **11** | **33** |

**Array (After Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **66** | **55** | **44** | **33** |

**i = 1**

**a[1] = 22**

**Smallest = 22**

**Position = 1**

**Array (Before Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **66** | **55** | **44** | **33** |

**Array (After Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **66** | **55** | **44** | **33** |

**i = 2**

**a[2] = 66**

**Smallest = 33**

**Position = 5**

**Array (Before Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **66** | **55** | **44** | **33** |

**Array (After Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **33** | **55** | **44** | **66** |

**i = 3**

**a[3] = 55**

**Smallest = 44**

**Position = 4**

**Array (Before Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **33** | **55** | **44** | **66** |

**Array (After Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **33** | **44** | **55** | **66** |

**i = 4**

**a[4] = 55**

**Smallest = 55**

**Position = 4**

**Array (Before Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **33** | **44** | **55** | **66** |

**Array (After Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **33** | **44** | **55** | **66** |

**i = 5**

**a[5] = 66**

**Smallest = 66**

**Position = 5**

**Array (Before Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **33** | **44** | **55** | **66** |

**Array (After Swapping)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **33** | **44** | **55** | **66** |

**Displaying Sorted Array: 11 22 33 44 55 66**

**Algorithm to Sort an Array Using Insertion Sort.**

**INSERTION\_SORT(int a[],int len)**

1. **Repeat steps for i := 1 to len-1**
2. **Set temp := a[i]**
3. **Set j := i-1**
4. **Repeat steps (a) and (b) while temp < a[j] and j >= 0**
5. **a[j+1]=a[j]**
6. **j=j-1**

**[End of loop in step 4]**

1. **set a[j+1] := temp**

**[End of loop in step 1]**

1. **Exit**

**Processing of Insertion Sort:**

**Total Elements = 6**

**Input Array**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **44** | **22** | **55** | **33** | **66** | **11** |

**Value of i = 1**

**Value of Temp = 22**

**Value of j = 0**

**Code Processing When j = 0**

**while (22 < a[0] && 0>=0) Both Conditions are True, Hence a[0+1]=a[0].**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
|  | **44** | **55** | **33** | **66** | **11** |

**Value of j = -1**

**Code Processing When j = -1**

**while (22 < a[-1] && -1>=0) Both Conditions are False, Hence The While Loop Terminates.**

**Status of Array After i = 1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **22** | **44** | **55** | **33** | **66** | **11** |

**Value of i = 2**

**Value of TEMP = 55**

**Value of j = 1**

**Code Processing When j = 1**

**while (55 < a[1] && 1>=0) One Condition is False, Hence The While Loop Terminates.**

**Status of Array After i = 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **22** | **44** | **55** | **33** | **66** | **11** |

**Value of i = 3**

**Value of TEMP = 33**

**Value of j = 2**

**Code Processing When j = 2**

**while (33 < a[2] && 2>=0) Both Conditions are True, Hence a[2+1]=a[2].**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
|  |  |  | **55** | **66** | **11** |

**Value of j = 1**

**Code Processing When j = 1**

**while (33 < a[1] && 1>=0) Both Conditions are True, Hence a[1+1]=a[1].**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
|  |  | **44** | **55** | **66** | **11** |

**Value of j = 0**

**Code Processing When j = 0**

**while (33 < a[0] && 0>=0) One Condition is False, Hence The While Loop Terminates.**

**Status of Array After i = 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **22** | **33** | **44** | **55** | **66** | **11** |

**Value of i = 4**

**Value of TEMP = 66**

**Value of j = 3**

**Code Processing When j = 3**

**while (66 < a[3] && 3>=0) One Condition is False, Hence The While Loop Terminates.**

**Status of Array After i = 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **22** | **33** | **44** | **55** | **66** | **11** |

**Value of i = 5**

**Value of TEMP = 11**

**Value of j = 4**

**Code Processing When j = 4**

**while (11 < a[4] && 4>=0) Both Conditions are True, Hence a[4+1]=a[4].**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
|  |  |  |  |  | **66** |

**Value of j = 3**

**Code Processing When j = 3**

**while (11 < a[3] && 3>=0) Both Conditions are True, Hence a[3+1]=a[3].**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
|  |  |  |  | **55** | **66** |

**Value of j = 2**

**Code Processing When j = 2**

**while (11 < a[2] && 2>=0) Both Conditions are True, Hence a[2+1]=a[2].**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
|  |  |  | **44** | **55** | **66** |

**Value of j = 1**

**Code Processing When j = 1**

**while (11 < a[1] && 1>=0) Both Conditions are True, Hence a[1+1]=a[1].**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
|  |  | **33** | **44** | **55** | **66** |

**Value of j = 0**

**Code Processing When j = 0**

**while (11 < a[0] && 0>=0) Both Conditions are True, Hence a[0+1]=a[0].**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
|  | **22** | **33** | **44** | **55** | **66** |

**Value of j = -1**

**Code Processing When j = -1**

**while (11 < a[-1] && -1>=0) Both Conditions are False, Hence The While Loop Terminates.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a[0]** | **a[1]** | **a[2]** | **a[3]** | **a[4]** | **a[5]** |
| **11** | **22** | **33** | **44** | **55** | **66** |

**Displaying Sorted Array : 11 22 33 44 55 66**

**Algorithm To Implement Exchange Sort.**

**EXCHANGE\_SORT()**

**In this algorithm an unsorted array named ‘array’ is used. ‘length’ of the array is taken as 10.**

1. **Set i := 0.**
2. **Repeat steps (a),(b) and (c) while i < (length – 1).**
3. **Step j := i + 1.**
4. **Repeat steps (i) and (ii) while j < length.**
5. **Check if array[i] > array[j], then:**
6. **Interchange array[i] and array[j].**

**[End of step (i) if structure]**

1. **Set j := j + 1.**

**[End of step (b) while loop]**

1. **i := i + 1.**

**[End of step (2) while loop]**

1. **Exit.**

**Processing Of Exchange Sort:**

**Size of Array = 7**

**Input Array:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Array[0]** | **Array[1]** | **Array[2]** | **Array[3]** | **Array[4]** | **Array[5]** | **Array[6]** |
| **66** | **55** | **77** | **44** | **22** | **11** | **33** |

**Algorithm To Implement Radix Sort.**

**RADIX(NUM[MAX],SIZE,MAXDIGIT)**

**In this algorithm an unsorted array named ‘NUM’ is used. ‘MAX’ value of the array size is taken as 5. ‘size’ is set to ‘MAX’ and ‘maxdigit’ refers to maximum digits present in an element of array.**

1. **Set div := 0, i := 0 and j := 0.**
2. **Set radix\_i := 0 and radix\_j := 0.**
3. **Set temp := 0 and digit := 1.**
4. **Repeat steps (5) to (7) while digit <= maxdigit.**
5. **Set div := pow10(digit)**
6. **Repeat step (i) for i := 0 to less than size - 1.**
7. **Repeat steps (a) to (d) for j:=i+1 to less than size.**
   1. **Set radix\_i := NUM[i] % div.**
   2. **Set radix\_j := NUM[j] % div.**
   3. **Check if radix\_i > radix\_j, then:**
   4. **Interchange NUM[i] and NUM[j].**

**[End of step (a) if structure]**

**[End of step (i) for loop]**

**[End of step (6) for loop]**

1. **Set digit := digit + 1.**

**[End of step (4) while loop]**

1. **Return.**

**RADIXSORT(NUM[MAX],LIMIT)**

**This algorithm counts the maximum digits present in an element of array and then call the function RADIX().**

1. **Set ndigit := 0, maxdigit :=0, val := 0 and i := 0.**
2. **Repeat step for i := 0 to less than limit.**
3. **Set val := NUM[i].**
4. **Repeat steps (i) and (ii) while val != 0.**
5. **Set val := val / 10.**
6. **Set ndigit := ndigit + 1.**

**[End of step (b) while loop]**

1. **Check if ndigit > maxdigit, then:**

**set maxdigit=ndigit.**

**[End of step (c) if structure]**

1. **Set ndigit := 0.**

**[End of step (2) for loop]**

1. **Call radix(NUM, limit, maxdigit).**
2. **Exit.**

**Processing Of Radix Sort:**

**Size Of Input Array = 5**

**Input Array:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **num[0]** | **num[1]** | **num[2]** | **num[3]** | **num[4]** |
| **446** | **233** | **455** | **587** | **329** |

**Algorithm To Implement Shell Sort.**

**SHELL\_SORT(ARRAY[], SIZE)**

1. **Set gap := size/2.**
2. **Do steps (a) to (c) and (3) while gap != 0.**
3. **Do steps (a) to (c) while exchange\_occurred = 1.**
4. **Set exchange\_occurred := 0.**
5. **Repeat steps () for i = 0 to i<size-gap.**
6. **Check if array[i] > array[i+gap], then:**

**Interchange array[i] with array[i+gap].**

**Set exchange\_occurred := 1.**

**[End of step (c) for loop]**

**[End of step (a) do while loop]**

1. **Set gap := gap / 2.**

**[End of step (2) do while loop]**

**Processing Of Shell Sort:**

**Size of Array = 7**

**Input Array:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Array[0]** | **Array[1]** | **Array[2]** | **Array[3]** | **Array[4]** | **Array[5]** | **Array[6]** |
| **66** | **22** | **77** | **33** | **11** | **44** | **55** |

**Trees**

**Design A Algorithm To:**

1. **Create A Tree**
2. **Traverse It In Inorder**
3. **Traverse It In Preorder**
4. **Traverse It In Postorder**
5. **Delete Leaf Node**
6. **Delete Node With Single Child**
7. **Delete Node With Two Childs**

**Algorithm To Create A BST:**

**The structure of node is represented using BINARY\_S\_TREE it contains three parts ‘left’ pointer, ‘info’ part and ‘right’ pointer part. Another pointer ‘root’ is created which will point to NULL initially and later on it will point to the root node of the tree. When the algorithm starts, ‘loc’ points towards ‘root’.**

**CREATE\_TREE(BINARY\_S\_TREE \*loc, int data)**

1. **Check if data == info[loc], then:**

**Print "DUPLICATE VALUE" and return.**

1. **Else check if data < info[loc], then:**
2. **If left[loc] != NULL, then:**

**Call CREATE\_TREE (left[loc], data).**

1. **Else**

**Print "INSERTING TO LEFT".**

**Allocate memory to a node named FRESH.**

**Set info[FRESH] := data.**

**Set left[FRESH] := NULL.**

**Set right[FRESH] := NULL.**

**set left[loc] := FRESH.**

**3. Else check if data > info[loc], then:**

1. **If right[loc] != NULL, then:**

**Call CREATE\_TREE (right[loc], data).**

1. **Else**

**Print "INSERTING TO RIGHT".**

**Allocate memory to a node named FRESH.**

**Set info[FRESH] := data.**

**Set left[FRESH] := NULL.**

**Set right[FRESH] := NULL.**

**Set right[loc] := FRESH.**

**[End of step 1 if else if structure]**

**4. Exit**

**Algorithm To Inorder Traverse A BST:**

**‘loc’ is a pointer which points to node to be processed. Initially pointer ‘root\_loc’ will point towards ‘root’.**

**INORDER(BINARY\_S\_TREE \* root\_loc)**

1. **Set Loc := root\_loc.**
2. **Check if loc != NULL, then:**
3. **Call INORDER (left[loc]).**
4. **Print info[loc].**
5. **Call INORDER (right[loc]).**

**[End of step 2 if structure]**

1. **Exit**

**Algorithm To Preorder Traverse A BST:**

**‘loc’ is a pointer which points to node to be processed. Initially pointer ‘root\_loc’ will point towards ‘root’.**

**PREORDER(BINARY\_S\_TREE \* root\_loc)**

1. **Set Loc := root\_loc.**
2. **Check if loc != NULL, then:**
3. **Print info[loc].**
4. **Call PREORDER (left[loc]).**
5. **Call PREORDER (right[loc]).**

**[End of step 2 if structure]**

1. **Exit.**

**Algorithm To Postorder Traverse A BST:**

**‘loc’ is a pointer which points to node to be processed. Initially pointer ‘root\_loc’ will point towards ‘root’.**

**POSTORDER(BINARY\_S\_TREE \* root\_loc)**

1. **Set Loc := root\_loc.**
2. **Check if loc != NULL, then:**
3. **Call POSTORDER (left[loc]).**
4. **Call POSTORDER (right[loc]).**
5. **Print info[loc].**

**[End of step 2 if structure]**

1. **Exit.**

**Algorithm To Delete A Leaf Node A BST:**

**‘loc’ is a pointer which points to node to be processed. ‘par’ is pointer which points to parent of ‘loc’. ‘item’ contains the information value of the leaf node to be deleted.**

**DELETE\_LEAF()**

1. **Set loc := root**
2. **Repeat steps (a) and (b) while info[loc] != item.**
3. **Check if item < info[loc], then:**

**Set par := loc and loc := left[loc].**

1. **Else check if item > info[loc], then:**

**Set par := loc and loc := right[loc].**

**[End of while loop in step 2]**

1. **Check if info[par] > info[loc], then:**

**Set left[par] := NULL.**

1. **Else Check if info[par] < info[loc], then:**

**Set right[par] := NULL.**

1. **Free the memory occupied by loc.**
2. **Exit.**

**Algorithm To Delete A Node With One Child In A BST:**

**‘loc’ is a pointer which points to node to be processed. ‘par’ is pointer which points to parent of ‘loc’. ‘item’ contains the information value of the leaf node to be deleted. ‘child’ is a pointer which points towards the left or right child of ‘loc’.**

**DELETE\_NODE\_WITH\_SINGLE\_CHILD()**

1. **Set loc := root**
2. **Repeat while info[loc] != item.**
3. **Check if item < info[loc], then:**

**Set par := loc and loc := left[loc].**

1. **Else check if item > info[loc], then:**

**Set par := loc and loc := right[loc].**

**[End of while loop in step 2]**

1. **Check if left[loc] == NULL then:**

**Set child := right[loc].**

1. **Check if right[loc] == NULL then:**

**Set child := left[loc].**

1. **Check if info[par] > info[loc] then:**

**Set left[par] := child.**

1. **Else Check if info[par] < info[loc] then:**

**Set right[par] := child.**

1. **Set right[loc] := NULL and left[loc] := NULL.**
2. **Free the memory occupied by loc.**
3. **Exit.**

**Algorithm To Delete A Node With Two Children In A BST:**

**DELETE\_NODE\_WITH\_TWO\_CHILDS()**

1. **Set loc := root.**
2. **Repeat step (a) while info[loc] != item**
3. **Check if item < info[loc], then**

**Set par := loc and loc := left[loc].**

**Else**

**Set par := loc and loc := right[loc].**

**[End of step (a) if structure]**

**[End of step (2) while loop]**

*Node Position to Be Deleted (25)*

1. **Set ptr=loc;**
2. **Set ptrr := right[loc].**
3. **Repeat step (b) while left[ptrr] != NULL**
4. **Set ptr := ptrr and ptrr := left[ptrr].**

*As The INORDER Successor Of Any Node Is The Left Most Child In The Right Sub Tree Of That Node.*

**[End of step (5) while loop]**

1. **Check if left[ptrr] = NULL and right[ptrr] = NULL, then:**

*Checking For Leaf Node*

1. **Check if info[ptr] > info[ptrr], then:**

**Set left[ptr] := NULL.**

**Else**

**Set right[ptr] := NULL.**

**[End of step (c) if structure]**

1. **Check if left[ptrr] != NULL or right[ptrr] != NULL, then:**

*Checking For Single Child Node*

1. **Check if left[ptrr]= NULL, then:**

**Set child := right[ptrr].**

**Else**

**Set child := left[ptrr].**

**[End of step (d) if structure]**

1. **Check if info[ptr] > info[ptrr], then**

**Set left[ptr] := child.**

**Else**

**Set right[ptr] := child.**

**[End of step (e) if structure]**

**[End of step (6) if structure]**

1. **[Changing position of node to be deleted (25) with node with value 33]**
2. **Check else if loc = left[par], then:**

**Set left[par] := ptrr.**

**Else**

**Set right[par] := ptrr.**

**[End of step (f) if structure]**

1. **Set left[ptrr] := left[loc].**
2. **Set right[ptrr] := right[loc].**
3. **Free the memory occupied by loc.**
4. **Exit.**

**The Input Data Will Be: 40, 60, 50, 33, 55, 11, 45**

**40**

**33**

**11**

**60**

**50**

**45**

**55**

**Tree**

**Deletion Of Leaf Node**

**40**

**33**

**11**

**60**

**50**

**45**

**55**

**When To Delete Leaf Node; Simply Replace The Position Of Node To Be Deleted By The NULL Pointer.**

**40**

**33**

**11**

**60**

**50**

**45**

**Node To Be Deleted Is 55**

**Deletion Of Node With Single Child**

**40**

**33**

**11**

**60**

**50**

**45**

**55**

**When to Delete Node with Single Child; Simply Replace The Position Of That Node With Its Single Child Node**

**40**

**11**

**60**

**50**

**45**

**55**

**Node to Be Deleted Is 33**

**Deletion Of Node With Two Childs**

**60**

**25**

**15**

**75**

**50**

**33**

**44**

**66**

**INORDER Traversal:**

**15, 25, 33, 44, 50, 60, 66, 75**

**Node To Be Deleted Is 25, Since As In The INORDER Traversal The Immediate Node After 25 Is 33 So 33 Will Replace 25.**

**NOTE:**

1. **After Finding The Immediate Node (33)**
2. **Check Whether 33 Is Leaf Node Or Single Child Node**
3. **Adjust The Child Of 33 (If Any)**
4. **Adjust 33 Inplace Of 25 And Add The Childs Of 25 To 33.**

**60**

**33**

**15**

**75**

**50**

**44**

**66**

**When To Delete Node With Two Childs; Simply Replace The Node With Its Immediate Node In The INORDER Traversal.**

**Graph**

**Program To Create A Graph Using Adjacency List.**

**#include<stdio.h>**

**#include<malloc.h>**

**#include<conio.h>**

**struct NODE**

**{**

**char VERTEX;**

**struct NODE \* N;**

**};**

**void READGRAPH(struct NODE \* ADJ[], int NO\_OF\_NODES)**

**{**

**struct NODE \* FRESH, \* LOC;**

**int i,j,NO\_OF\_NEIGH;**

**char VAL;**

**for(i=0;i<NO\_OF\_NODES;i++)**

**{**

**ADJ[i]=NULL;**

**printf("ENTER NUMBER OF NEIGHBOURS OF NODE %d : ",i);**

**scanf("%d",&NO\_OF\_NEIGH);**

**for(j=0;j<NO\_OF\_NEIGH;j++)**

**{**

**printf("ENTER THE %d NEIGHBOUR OF NODE %d : ",i,j);**

**scanf(" %c",&VAL);**

**FRESH=(struct NODE \*) malloc (sizeof(struct NODE));**

**FRESH->VERTEX=VAL;**

**FRESH->N=NULL;**

**if(ADJ[i]==NULL)**

**{**

**ADJ[i]=FRESH;**

**}**

**else**

**{**

**LOC=ADJ[i];**

**while(LOC->N != NULL)**

**{**

**LOC=LOC->N;**

**}**

**LOC->N=FRESH;**

**}**

**}**

**}**

**}**

**void DISPLAYGRAPH(struct NODE \* ADJ[], int NO\_OF\_NODES)**

**{**

**struct NODE \* LOC;**

**int i;**

**printf("\nDISPLAYING ADJACENCY LIST\n");**

**for(i=0;i<NO\_OF\_NODES;i++)**

**{**

**printf("NODE %d : ",i);**

**LOC=ADJ[i];**

**while(LOC != NULL)**

**{**

**printf("%c -> ",LOC->VERTEX);**

**LOC=LOC->N;**

**}**

**printf("\n");**

**}**

**}**

**void main()**

**{**

**struct NODE \* ADJ[100];**

**int NO, i;**

**clrscr();**

**printf("ENTER NUMBER OF NODES OF THE GRAPH ");**

**scanf("%d",&NO);**

**for(i=0;i<NO;i++)**

**{**

**ADJ[i]=NULL;**

**}**

**READGRAPH(ADJ,NO);**

**DISPLAYGRAPH(ADJ,NO);**

**getch();**

**}**