|  |  |
| --- | --- |
|  | **School of Engineering & Technology**  **Global Campus**  **Jakkasandra Post, Kanakapura Taluk, Ramanagara District**  **Pin Code: 562 112** |
| **Department of Computer Science and Engineering** | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **USN** |  |  |  |  |  |  |  |  |  |  |

**III SEMESTER TEST –II**

**Solutions & Scheme**

|  |  |
| --- | --- |
| **Subject: Data Structures Using ‘C’** | **Session: June – Dec 2019** |
| **Subject Code:18CSI301** | **Duration: 90 Minutes** |
| **Date of Examination:04-11-2019** | **Max Marks: 03 X 15 = 45** |

**Note:**

* Answer **3 full questions** and each full question carries **15 Marks.**
* **Provide neat diagrams wherever applicable.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Q.No** | **Question** | | **Marks** | **CO’s** | **Bloom’s**  **Level** |
| **1a** | Illustrate with a suitable pictorial representation to perform rear insertion and front of a node in doubly linked list.  **Insertion at the end of the list:**  C:\Users\Lenovo\Desktop\www.jpg  Suppose **p** is a pointer pointing to the node **P** which is the last node of the list.  Node **T** becomes the last node so its next should be **NULL**  **tmp🡪 next=null;**  next part of node **P** should point to node **T**  **p🡪next=tmp;**  prev part of node T should point to node P  tmp🡪prev=p;  struct node \*addatend(struct node \*start, int data)  struct node \*tmp,\*p;  tmp=(struct node \*)malloc(sizeof(struct node));  tmp🡪info=data;  P=start;  while(p🡪next!=NULL)  p=P🡪next;  p🡪next=tmp;  tmp🡪prev=p;  return start;  }/\*end of addatend()\*/  **Insertion at the beginning of the list:**  C:\Users\Lenovo\Desktop\wwww.jpg  Node T has become the first node so its **prev**  should be NULL  **tmp🡪prev=null**  the next part of the node T should point to node P, and address of the node P is in START so we should writ  tmp🡪next=start;  node T is inserted before node P sp prev part of the node P should now point to node T.  **start🡪prev=tmp;**  **new node T**  has become the first node so START should point to it  start=tmp;  C:\Users\Lenovo\Desktop\eee.jpg | | **10**  **5**  **5** | **CO-3** | **L3** |
| **1b** | *Provide a C function to search a node in a singly linked list.*  For searching an element, we traverse the linked list and while traversing we compare the info part of each element with the given element to be searched.  In the function given below , **item** is the element which we want to search  C:\Users\Lenovo\Desktop\ioio.jpg | | **05** | **CO-3** | **L3** |
| **OR** | | | | | |
|  |  | |  |  |  |
| **2a** | Describe the various operations performed in a double ended queue. Write C functions to perform front insertion and rear deletion using arrays.  Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (front and rear). That means, we can insert at both front and rear positions and can delete from both front and rear positions.  **Front insertion**  Step-1 : [Check for the front position]  if(front<=1)  Print("Cannot add item at the front”);  return;  Step-2 : [Insert at front]  else  front=front-1;  q[front]=no;  Step-3 : Return    **Rear deletion**  Step-1 : [Check for the rear pointer]  if rear=0  print(“Cannot delete value at rear end”);  return;  Step-2: [ perform deletion]  else  no=q[rear];  [Check for the front and rear pointer]  if front= rear  front=0;  rear=0;  else  rear=rear-1;  print(“Deleted element is”,no);  Step-3 : Return | | **10**  **5**  **5** | **CO-3** | **L3** |
| **2b** | What is priority queue? Explain the different types of priority queues?  A heap is a tree-based data structure in which all the nodes of the tree are in a specific order. For example, if is the parent node of , then the value of follows a specific order with respect to the value of and the same order will be followed across the tree.  Based on heap structure, priority queue also has two types max- priority queue and min - priority queue. Max Priority Queue is based on the structure of max heap and can perform following operations: maximum(Arr) : It returns maximum element from the Arr | | **05** | **CO-3** | **L3** |
| **OR** | | | | | |
| **3a** | Explain the following tree terminologies: Internal node, leaf node, degree of a node, level of a node, depth of a node, siblings and find the same for the given tree.  internal nodes:  leaf node:  **Degree of a node:**    level of a node:  Siblings of a tree: | C:\Users\SWATHI\Desktop\r1.jpg  An internal node (also known as an inner node, inode for short, or branch node) is any node of a tree that has child nodes. Similarly, an external node (also known as an outer node, leaf node, or terminal node) is any node that does not have child nodes  C,K are internal nodes  F,L,Y,R,W,Z,J,D  The degree of a node is the number of children of that node  The depth of a node is the number of edges from the node to the tree's root node. Level – The level of a node is defined by 1 + the number of connections between the node and the root. It starts from 1 and the level of the root is 1  level of a node:4  Nodes with the same parent are called siblings  F,L,Y,R,W,Z,J,D | **06** | **CO-3** | **L3** |
| **3b** | Write the recursive steps to perform the Pre-order, In-order and Post-order traversal. Apply the traversal technique for the given binary tree  **Preorder**:  **Inorder**:  **Post Order:** | C:\Users\SWATHI\Desktop\rr2.jpg  C:\Users\Lenovo\Desktop\123.jpg | **09**  **03**  **03**  **03** | **CO-3** | **L4** |
|  |  | |  |  |  |
| **4** | Draw the binary search tree whose elements are inserted in the following order: 50, 30,60,38,35,55,22,59,94,13,60,100,20,17,10   1. What is the height of the tree? 2. **Write the nodes on level2.: 30,60** 3. **List the leaf nodes.:13,35,59,98** 4. Find the minimum and maximum values in Tree. 5. Find the pre-order and post-order traversal for the Tree. 6. Show how the tree would look after the insertion of nodes containing 66, 70, 78, 48, 8 and 5 (in that order).   C:\Users\Lenovo\Desktop\rttt.jpg | | **15**  **2**  **2**  **2**  **3**  **3**  **3** | **CO-3** | **L4** |
| **OR** | | | | | |
| **5a** | Write the advantages and disadvantages of linked list over arrays.  **Advantages:**  Size of the list doesn't need to be mentioned at the beginning of the program, certainly dynamic memory allocation and deallocation.  As the linked list doesn't have a size limit, we can go on adding new nodes (elements) and increasing the size of the list to any extent.  **Disadvantages:**  The main disadvantage of linked list over array is access time to individual elements. Array is random-access, which means it takes o(1) to access any element in the array. Linked list takes O(n) for access to an element in the list in the worst case. | | **06**  **3**  **3** | **CO-3** | **L3** |
| **5b** | Implement stacks using linked list.  In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its previous node in the list. The next field of the first element must be always NULL.  push(value) - Inserting an element into the Stack  **We can use the following steps to insert a new node into the stack...**  Step 1 - Create a newNode with given value.  Step 2 - Check whether stack is Empty (top == NULL)  Step 3 - If it is Empty, then set newNode → next = NULL.  Step 4 - If it is Not Empty, then set newNode → next = top.  Step 5 - Finally, set top = newNode.  pop() - Deleting an Element from a Stack  **We can use the following steps to delete a node from the stack...**  Step 1 - Check whether stack is Empty (top == NULL).  Step 2 - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function  Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.  Step 4 - Then set 'top = top → next'.  Step 5 - Finally, delete 'temp'. (free(temp)).  display() - Displaying stack of elements  **We can use the following steps to display the elements (nodes) of a stack...**  Step 1 - Check whether stack is Empty (top == NULL).  Step 2 - If it is Empty, then display 'Stack is Empty!!!' and terminate the function.  Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with top.  Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack. (temp → next != NULL).  Step 5 - Finally! Display 'temp → data ---> NULL'. | | **09**  **3**  **3**  **3** | **CO-3** | **L3** |

**OR**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **6a** | Define heap tree. Construct a max-heap tree for the following numbers and show the individual step.  90, 70, 105, 120, 170, 45, 175, 350, 30, 90, 130,115  A heap is a specialized tree-based data structure that satisfied the heap property:  if B is a child node of A, then key(A) ≥ key(B).  This implies that an element with the greatest key  is always in the root node, and so such a heap is  sometimes called a max-heap. Of course, there's also a min-heap. | | **07** | **CO-3** | **L4** |
| **6b** | Explain the memory representation of binary tree. Also represent the given tree in memory (array based and linked based).  **Array Representation**  **Linked Representation** | **C:\Users\Lenovo\Desktop\ssss.jpg** | **08**  **04**  **04** | **CO-3** | **L4** |

**Course Outcomes:**

|  |  |
| --- | --- |
|  | Understand the various types of data structures, operations and algorithms |
|  | Analyze the various algorithms used in linear and non-linear data structures. |
|  | Design the algorithm for stack, queues, list, trees and graphs. |
|  | Apply appropriate data structures for solving computing problems. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| L1 | L2 | L3 | L4 | L5 | L6 |
| Remembering | Understanding | Applying | Analyzing | Evaluating | Creating |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CO/PO: Mapping** | | | | | | | | | | | | |
| (H/M/L indicates strength of correlation) H-High, M-Medium, L-Low | | | | | | | | | | | | |
| **Course Outcome**  **(COs)** | **Program Outcome (POs)** | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | H | H | H | H | M | L | M | M | M | L | H | H |
|  | H | H | L | H | M | L | M | M | M | L | H | H |
|  | H | H | M | H | H | L | M | M | M | L | H | H |
|  | M | M | H | M | M | L | M | M | M | L | M | H |