|  |  |  |  |
| --- | --- | --- | --- |
| Q No | Set by | Question | marks |
| 1 | HSV | Given the following declarations:  int num[10] = {23, 3, 5, 7, 4, -1, 6, 12, 10, -23};  int i = 2;  int j = 4;  Write the values of the following expressions.  a) \*(num + 2)  b) \*(num + i + j)  c) \*(num + \*(num + 1))  d) \*(num + j)  e) \*(num + i) + \*(num + j)  f) \*(num + i) \* \*(num + j)  ----------------  Scheme:  5  6  7  4  9  20 6\*0.5M=3M | 3 |
| 2 | HSV | Write a recursive function with the function declaration int rcntVowels(char \*s);to find and return number of vowels (only lower case) in a string passed as parameter using static variable. Write main() to read a string from the keyboard, call the function, and display the count. To access element of the array use dereference operator(\*) only. -------------------  Scheme:  #include<stdio.h>  int rcntVowels(char \*s) {  static int cnt = 0;  if(\*s == '\0')  return cnt; 1M  if(\*s == 'a' || \*s == 'e' ||  \*s == 'i' || \*s== 'o' || \*s == 'u')  cnt++;  rcntVowels(s+1);  } 1M    int main() {  char s[20];  int count;  printf("Enter a string: ");  gets(s);  count = rcntVowels(s);  printf("%d", count);  } 1M | 3 |
| 3 | GB | Consider a **Circular Queue** implemented using a dynamic array of size 3 with array doubling. Show the status of the queue using the table below for each of the following operations. Show appropriate messages whenever required. [Table shows the first   1. Begin 2. Insert 5 3. Insert 10 4. Delete 5. Insert 20 6. Delete 7. Insert 30 8. Delete 9. Delete 10. Delete 11. Insert 5 12. Insert 10 13. Insert 15 14. Insert 20  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **S.No** | **Operation** | **Element Inserted/Deleted/ Message** | **Front** | **Rear** | **Array** | |  |  |  |  |  | 0 1 2   |  |  |  | | --- | --- | --- | |  |  |  | |   Scheme:  1m – Operation i-v 1m – Operation vi -x 1m – Operation x-iv 1m – Error Messages  ***Deductions:***  ***If array memory is not freed, when queue is empty – 1m deducted***  ***If Front and Rear not reset when the queue is empty – 1m deducted***   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | S.No | Operation | Element Inserted/Deleted/ Message | Front | Rear | Array | | 1 | Begin | Create Array | 0 | 0 | 0 1 2     |  |  |  | | --- | --- | --- | |  |  |  | | | 2 | Insert | 5 | 0 | 1 | 0 1 2     |  |  |  | | --- | --- | --- | |  | 5 |  | | | 3 | Insert | 10 | 0 | 2 | 0 1 2     |  |  |  | | --- | --- | --- | |  | 5 | 10 | | | 4 | Delete | 5 | 1 | 2 | 0 1 2     |  |  |  | | --- | --- | --- | |  |  | 10 | | | 5 | Insert | 20 | 1 | 0 | 0 1 2     |  |  |  | | --- | --- | --- | | 20 |  | 10 | | | 6 | Delete | 10 | 2 | 0 | 0 1 2     |  |  |  | | --- | --- | --- | | 20 |  |  | | | 7 | Insert | 30 | 2 | 1 | 0 1 2     |  |  |  | | --- | --- | --- | | 20 | 30 |  | | | 8 | `Delete | 20 | 0 | 1 | 0 1 2     |  |  |  | | --- | --- | --- | |  | 30 |  | | | 9 | `Delete | 30  Free memory | 0 | 0 |  | | 10 | Delete | Queue Empty | 0 | 0 |  | | 11 | Insert | 5 | 0 | 1 | 0 1 2     |  |  |  | | --- | --- | --- | |  | 5 |  | | | 12 | Insert | 10 | 0 | 2 | 0 1 2     |  |  |  | | --- | --- | --- | |  | 5 | 10 | | | 13 | Insert | 15  Queue Full  Array Doubling | 0 | 3 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 5 | 10 | 15 |  |  | | | 14 | Insert | 20 | 0 | 0 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 5 | 10 | 15 | 20 |  | | | 4 |
| 4 | GNS, GSP | Write a function ***struct node*** \* ***insert\_order (struct node \*first, int reg\_no )*** which inserts a new node into a singly linked list without header node(list may be initially empty) in the ascending order of the registration number and returns the new list. The node structure of the linked list is as given below:    struct node {  int registration;  struct node\* next;  };    Scheme:  struct node \* insert\_order(struct node \*first,int reg\_no)  {  struct node \*n=(struct node \*)malloc(sizeof(struct node));  n->reg\_no=reg\_no;  n->next=NULL;  if(first==NULL)return n;  if(reg\_no<=first->reg\_no)  {n->next=first;  return n;}  struct node \*prev=NULL, \*cur=first;  while(cur!=NULL && reg\_no> cur->reg\_no)  {  prev=cur;  cur=cur->next;  }  prev->next=n;  n->next=cur;  return first;  }    (  Allocating memory à 0.5 M  Initialization and checking for empty condition à0.5 M  Checking for key value less than first node key value à 0.5 M  Traversing and connecting the node à1.5 M) | 3 |
| 5 | GNS, GSP | Given a singly linked list without header node consisting of nodes in the ascending order of registration number, write a function ***void Remove\_Duplicates( struct node \*first )*** which deletes the nodes with duplicate registration numbers, retaining the first occurrence in the list.  The node structure of the linked list is as given below:    struct node {  int registration;  struct node\* next;  };      Scheme:  void Remove\_Duplicates(struct node \*first)  {  /\* Pointer to traverse the linked list \*/  struct node \* current = first,\*next\_next;    if (current == NULL)  return;    /\* Traverse the list till last node \*/  while (current->next != NULL)  {  /\* Compare current node with next node \*/  if (current->reg\_no == current->next->reg\_no)  {  /\* The sequence of steps is important\*/  next\_next = current->next->next;  free(current->next);  current->next = next\_next;  }  else /\* This is tricky: only advance if no deletion \*/    current = current->next;    }    ( Initialization and return if empty à 0.5M  Checking for duplicate nodeà 0.5M  Updating the links if duplicates found à0.5 M  Remove the node with duplicate keyà 0.5M  Update the pointer to go the next nodeà 0.5 M  Update the pointe if no duplicate(s) foundà 0.5M) | 3 |
| 6 | GSP | Given two **Circular Doubly Linked Lists** representing long binary numbers, write a function **Add (),** to add the two binary numbers and return a **Circular Doubly Linked List** representing the sum. The prototype of the Add function is as follows:  **Nodeptr Add(Nodeptr A, Nodeptr B);**  Scheme:  Initialization: 0.5M  Adding corresponding bits with carry: 1.5M  Adding carry to remaining bits of bigger number: 1M  Insert the last carry, if present: 0.5 M  Return: 0.5M  Decimal Number Addition – out of 2  Implemented other than specified data structures– out of 2 | 4 |