|  |  |  |  |
| --- | --- | --- | --- |
| Q No | Set by | Question | marks |
| 1 | HSV | Given the following declarations:  int num[10] = {23, 5, 7, 4, -1, 6, 12, 10, 3, -23};  int k = 2;  int h = 4;  Write the values of the following expressions.  a) \*(num + 2)  b) \*(num + k + h)  c) \*(num + 1) + \*(num + h)  d) \*(num + h)  e) \*num + \*(num + h)  f) \*(num + k) \* \*(num + h)  --------------------  Scheme:  7  12  4  -1  22  -7 6\*0.5M=3M | 3 |
| 2 | HSV | Create a type STUDENT which is used to represent a student structure with reg\_no, name and cgpa. Write a complete program to dynamically allocate memory for N such students, where N is read from keyboard and to read and display information for each student.  -------------------  Scheme:  #include<stdio.h>  #include<stdlib.h>    typedef struct {  int reg\_no;  char name[20];  float cgpa;  } STUDENT; 1M    int main() {  int i, n;  printf("Enter the number of students\n");  scanf("%d", &n);  STUDENT \*s = (STUDENT \*) calloc (n, sizeof(STUDENT)); 1M  printf("enter the students information\n");  for(i = 0; i < n; i++)  scanf("%d%s%f", &(s+i)->reg\_no, (s+i)->name, &(s+i)->cgpa);  for(i = 0; i < n; i++)  printf("%d\t%s\t%f\t\n", (s+i)->reg\_no, (s+i)->name, (s+i)->cgpa);  } 1M | 3 |
| 3 | GB | Consider a **Circular Deque** implemented using a fixed array of size 5. Show the status of the queue using the table below for each of the following operations. Show appropriate messages whenever required.    2. Begin 3. InsertF 5 4. InsertR 10 5. InsertF 3 6. InsertF 2 7. DeleteR 8. DeleteR 9. DeleteR 10. InsertF 5 11. InsertR 1 12. InsertR 0 13. DeleteF 14. DeleteF 15. DeleteF 16. DeleteF 17. DeleteF  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **S.No** | **Operation** | **Element Inserted/Deleted/ Message** | **Front** | **Rear** | **Array** | |  |  |  |  |  | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |   Scheme:  1m – Operation i-vi 1m – Operation vii -xi 1m – Operation xi-xvii 1m – Error Messages  ***Deduction:***  ***If InsertR is performed at the end of the array at index 4, 2mark deducted***   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | S.No | Operation | Element Inserted/Deleted/ Message | Front | Rear | Array | | 1 | Begin |  | -1 | -1 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | | | 2 | InsertF | 5 | 0 | 0 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 |  |  |  |  | | | 3 | InsertR | 10 | 0 | 1 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 | 10 |  |  |  | | | 4 | InsertF | 3 | 4 | `1 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 | 10 |  |  | 3 | | | 5 | InsertF | 2 | 3 | 1 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 | 10 |  | 2 | 3 | | | 6 | DeleteR | 10 | 3 | 0 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 |  |  | 2 | 3 | | | 7 | DeleteR | 5 | 3 | 4 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  | 2 | 3 | | | 8 | DeleteR | 3 | 3 | 3 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  | 2 |  | | | 9 | InsertF | 5 | 2 | 3 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  | 5 | 2 |  | | | 10 | InsertR | 1 | 2 | 4 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  | 5 | 2 | 1 | | | 11 | InsertR | 0 | 2 | 0 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 0 |  | 5 | 2 | 1 | | | 12 | DeleteF | 5 | 3 | 0 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 0 |  |  | 2 | 1 | | | 13 | DeleteF | 2 | 4 | 0 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 0 |  |  |  | 1 | | | 14 | DeleteF | 1 | 0 | 0 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | | 0 |  |  |  |  | | | 15 | DeleteF | 0 | -1 | -1 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | | | 16 | DeleteF | Queue Empty | -1 | -1 | 0 1 2 3 4     |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | | | 4 |
| 4 | GNS, GSP | Write a function ***struct node*** \* ***insert\_order (struct node \*first, int reg\_no )*** which inserts a new node into a circular 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){n->next=n;return n;}  if(reg\_no<=first->reg\_no)  { struct node \*l=first;  while(l->next!=first)  l=l->next;  n->next=first;  l->next=n;  return n;}  struct node \*prev=NULL, \*cur=first;  while(cur->next!=first && reg\_no>cur->reg\_no)  {  prev=cur;  cur=cur->next;  }  if(reg\_no>cur->reg\_no)  { cur->next=n;  n->next=first;  }  else {prev->next=n;  n->next=cur;}  return first;  }    (  Allocating memory à 0.5 M  Initialization ,checking for empty condition and make the node circular à0.5 M  Checking for key value less than first node key value and connecting it à 1 M  Traversing and connecting the node à1 M) | 3 |
| 5 | GNS, GSP | Given a circular 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)  {  if (first == NULL)  return;  if(first->next==first){ free(first); return NULL;}  struct node \*current = first,\*next\_next;    /\* Traverse the list till last struct node \*/  while (current->next !=first)  {  /\* Compare current struct node with next struct 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 dreg\_notion \*/    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 **Doubly Linked Lists** **with header node** representing long binary numbers, write a function **Add (),** to add the two binary numbers and return a **Doubly Linked List with header node** 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 |