|  |  |  |  |
| --- | --- | --- | --- |
| Q No | Set by | Question | marks |
| 1 | HSV | Given the following declarations:  int num[10] = {20, 5, 7, 4, -1, 6, 12, 10, 3, -23};  int m = 3;  int h = 4;  Write the values of the following expressions.  a) \*(num + 2)  b) \*(num + m + h)  c) \*(num + 1) + \*(num + h)  d) \*(num + h)  e) \*num + \*(num + h)  f) \*(num + m) \* \*(num + h)  ------------------------  Scheme:  7  10  4  -1  19  -4 6\*0.5M=3M | 3 |
| 2 | HSV | Create a structure PLAYER which includes member variables id, name and address, where address is a pointer variable of structure ADDRESS created outside the PLAYER with fields pinCode and city. Write a program to read the information of one such player using the pointer variable of the structure PLAYER.  -----------------------------  Scheme:  #include <stdio.h>  #include <stdlib.h>    struct ADDRESS {  int pinCode;  char city[10];  };    struct PLAYER {  int id;  char name[10];  struct ADDRESS \*address;  }; 1M    int main() {  struct PLAYER \*person1;  person1 = (struct PLAYER \*) malloc(sizeof(struct PLAYER));  person1->address = (struct ADDRESS\*) malloc(sizeof(struct ADDRESS));  printf("Enter the name of the player\n"); 1M  scanf(" %s", person1->name);  printf("Enter the id of the player\n");  scanf("%d", &person1->id);  printf("Enter the pin code of the city\n");  scanf("%d", &person1->address->pinCode);  printf("Enter the name of the city \n");  scanf(" %s", person1->address->city); 1M  } | 3 |
| 3 | GB | Consider a **MultiStack** consisting of 3 stacks (Stack0, Stack1, and Stack2) implemented using a fixed array of size 6. Show the status of the queue using the table below for each of the following operations. Show appropriate messages whenever required.   1. Begin 2. Push0 5 3. Push1 10 4. Push1 11 5. Push1 12 6. Push2 100 7. Push2 101 8. Push0 9 9. Push0 9 10. Push0 9 11. Pop1 12. Pop1 13. Pop1 14. Pop1 15. Pop0 16. Pop2 17. Pop2  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **S.No** | **Operation** | **Element Inserted/Deleted/ Message** | **Top[0]** | **Top[1]** | **Top[2]** | **Array** | |  |  |  |  |  |  | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | |   Scheme:  1m – Operation i-vi 1m – Operation vii -xi 1m – Operation xi-xvii 1m – Error Messages  ***Deductions:***  ***If 101 is not inserted – 1m deducted***  ***If 9 is inserted –1m deducted***   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **S.No** | **Operation** | **Message** | **Top[0]** | **Top[1]** | **Top[2]** | **Array** | | 1 | Begin | Stack Empty | -1 | 1 | 3 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | | | 2 | Push0 | 5 | 0 | 1 | 3 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 |  |  |  |  |  | | | 3 | Push1 | 10 | 0 | 2 | 3 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 |  | 10 |  |  |  | | | 4 | Push1 | 11 | 0 | 3 | 3 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 |  | 10 | 11 |  |  | | | 5 | Push1 | 12 | 0 | 4 | 4 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 |  | 10 | 11 | 12 |  | | | 6 | Push2 | 100 | 0 | 4 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 |  | 10 | 11 | 12 | 100 | | | 7 | Push2 | 101 | 0 | 3 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 | 10 | 11 | 12 | 100 | 101 | | | 8 | Push0 | 9  Array Full | 0 | 3 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 | 10 | 11 | 12 | 100 | 101 | | | 8 | Push1 | 99  Array Full | 0 | 3 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 | 10 | 11 | 12 | 100 | 101 | | | 9 | Push2 | 999  Array Full | 0 | 3 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 | 10 | 11 | 12 | 100 | 101 | | | 10 | Pop1 | 12 | 0 | 2 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 | 10 | 11 |  | 100 | 101 | | | 11 | Pop1 | 11 | 0 | 1 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 | 10 |  |  | 100 | 101 | | | 12 | Pop1 | 10 | 0 | 0 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 |  |  |  | 100 | 101 | | | 13 | Pop1 | Stack 1 Empty  No pop | 0 | 0 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 5 |  |  |  | 100 | 101 | | | 14 | Pop0 | 5 | -1 | 0 | 5 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  | 100 | 101 | | | 15 | Pop2 | 101 | -1 | 0 | 4 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  | 100 |  | | | 16 | Pop2 | 100 | -1 | 0 | 3 | 0 1 2 3 4 5     |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | | | 4 |
| 4 | GNS, GSP | Write a function ***void insert\_order (struct node \*first, int reg\_no )*** which inserts a new node into a circular singly linked list with header node(list may be initially empty) in the ascending order of the registration number. The node structure of the linked list is as given below:  struct node {  int registration;  struct node\* next;  };      Scheme:    void insert\_order(struct node \*head, int reg\_no)  {  struct node \*n=(struct node \*)malloc(sizeof(struct node));  n->reg\_no=reg\_no;  n->next=NULL;  if(head->next==head)  {head->next=n;n->next=head;head->reg\_no++;return;}  if(reg\_no<=head->next->reg\_no)  { n->next=head->next;  head->next=n;head->reg\_no++;return;}  struct node \*prev=NULL, \*cur=head->next;  while(cur->next!=head && reg\_no>cur->reg\_no)  {  prev=cur;  cur=cur->next;  }  if(reg\_no>cur->reg\_no)  { cur->next=n;  n->next=head;  head->reg\_no++;    }  else {prev->next=n;  n->next=cur;head->reg\_no++;}  return ;  }  (  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 with 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 \*head)  {  if (head->next == head)  return;  struct node \*current = head->next,\*next\_next;    /\* Traverse the list till last node \*/  while (current->next !=head)  {  /\* 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 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 **Circular** **Doubly Linked Lists** **with header node** representing long binary numbers, write a function **Add (),** to add the two binary numbers and return a **Circular 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 |