|  |  |
| --- | --- |
| 1 | Recursion is a method in which the solution of a problem depends on \_\_\_\_\_\_\_\_\_\_\_\_   1. Larger instances of different problems b) Larger instances of the same problem c**) Smaller instances of the same problem** d) Smaller instances of different problems |
| 2 | The declaration int(\*p)[5] means   1. P is an array of pointers to integers 2. **P is a pointer to an array of 5 elements** 3. Same as int \*p[5] 4. None of these |
| 3 | **Which of the following operations is performed more efficiently by doubly linked list than by singly linked list?**  a) **Deleting a node whose location in given** b) Searching of an unsorted list for a given item c) Inverting a node after the node with given location d) Traversing a list to process each node |
| 4 | What does the following function do for a given Linked List with first node as head?  void fun1(struct node\* head)  {  if(head == NULL)  return;  fun1(head->next);  printf("%d ", head->data);  }  a)print all the nodes of a linked list  b)print all nodes of linked list in the reverse order  c)print alternate nodes of the list  d) print alternate nodes of the list in reverse order |
| 5 | The following C function takes a simply-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank. Choose the correct alternative to replace the blank line  Node \*move\_to\_front(Node \*head)  {    Node \*p, \*q;    if((head == NULL: || (head->next == NULL))      returnhead;    q = NULL; p = head;    while(p-> next !=NULL)    {      q = p;      p = p->next;    }    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_    returnhead;  }   1. q = NULL; p->next = head; head = p; **(B)** q->next = NULL; head = p; p->next = head; **(C)** head = p; p->next = q; q->next = NULL; **(D)q->next = NULL; p->next = head; head = p;** |
| 6 | Consider the following function to traverse a linked list.  voidtraverse(structNode \*head)  {     while(head->next != NULL)     {         printf("%d  ", head->data);         head = head->next;     }  }  Which of the following is **FALSE** about above function?   1. The function may crash if the head is NULL 2. The function doesn’t print the last node when the linked list is not empty 3. **The function is implemented incorrectly because it changes head** 4. None of these |
| 7 | What is the sequence of code to delete a node in a doubly linked list pointed by q   1. **q->prev->next=q->next;q->next->prev=q->prev** 2. q->prev->prev=q->next; q->next->next=q->prev 3. a->next->prev=q->next;q->prev->next=q->prev 4. None of these |
| 8 | What the does the following code do in a doubly linked list pointed by first  Foo(struct node \*first)//first points to the first node  {  While(first->next!=NULL)  first=first->next;  first->prev->next=NULL;  free(first);  }   1. **Deletes the last node** 2. Deletes the first node 3. Inserts at the end 4. None of these |
| 9 | Implement a function to insert a node in front of a doubly linked list  {  NODE \*temp;  temp=(NODE\*)malloc(sizeof(NODE));  temp->data=x;  temp->prev=temp->next=NULL;  if (\*p==NULL)//if list empty  \*p=temp;  else  {  temp->next=\*p;  (\*p)->prev=temp;  \*p=temp;  }  } |
| 10 | Implement a function to delete a node given its value in a singly linked list  void delete(NODE \*\*p, int x)  {  NODE\*q,\*prev;  q=\*p;  prev=NULL;  while((q!=NULL)&&(q->data!=x))  {  prev=q;  q=q->next;  }  if(q==NULL)  printf("Node node found\n");  else  {  if(prev==NULL)//first node  \*p=q->next;  else  prev->next=q->next;//middle or last node  }  free(q);  } |
| 11 | Pushing an element into stack already having five elements and stack size of 5 , then stack becomes  a) **Overflow** b) Crash c) Underflow d) User flow |
| 12 | Which of the following applications may use a stack?  a) A parentheses balancing program b) implementation of recursion c) conversion of infix to postfix d)**All of these** |
| 13 | A linear list of elements in which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as a ?  a)**Queue** b) Stack c) Tree d) Linked list |
| 14 | **A circular queue is implemented using an array of size 10. The array index starts with 0, front is 6, and rear is 9. The insertion of next element takes place at the array index.**  a**) 0** b) 7 c) 9 d) 10 |
| 15 | **If the MAX\_SIZE is the size of the array used in the implementation of circular queue. How is rear manipulated while inserting an element in the queue?**  a) rear=(rear%1)+MAX\_SIZE b) rear=rear%(MAX\_SIZE+1) c) **rear=(rear+1)%MAX\_SIZE** d) rear=rear+(1%MAX\_SIZE) |
| 16 | **If the MAX\_SIZE is the size of the array used in the implementation of circular queue, array index start with 0, front point to the first element in the queue, and rear point to the last element in the queue. Which of the following condition specify that circular queue is FULL?**  a) Front=rear= -1 b) **Front=(rear+1)%MAX\_SIZE** c) Rear=front+1 d) Rear=(front+1)%MAX\_SIZE |
| 17 | **The postfix form of the expression (A+ B)\*(C\*D- E)\*F / G is?**  **a) AB+ CD\*E - FG /\*\*** b) AB + CD\* E - F \*\*G / c) AB + CD\* E - \*F \*G / d) AB + CDE \* - \* F \*G / |
| 18 | **Consider the usual implementation of parentheses balancing program using stack. What is the maximum number of parentheses that will appear on stack at any instance of time during the analysis of ( ( ) ( ( ) ) ( ( ) ) )?**  a) 1 b) 2 c) 3 d) 4 |
| 19 | **Implement push and pop operation of stack using a structure**  int push(int x, STACK \*p)  {  if(p->top==p->size-1)  {  printf("Stack overflow\n");  return -1;  }  ++p->top; //or ++(\*p).top or (\*p).top++  p->s[p->top]=x;  return 1;  }  int pop(STACK \*p)  {  int x;  if(p->top==-1)  {  printf("stack underflow..\n");  return -1;  }  x=p->s[p->top];  --p->top;  return x;  } |
| 20 | **Implement a insert and delete operation of queue implemented by a linked list having front and rear pointers**  void qinsert(int x, struct node \*\*f, struct node \*\*r)  {  struct node \*temp;  temp=(struct node\*)malloc(sizeof(struct node));  temp->data=x;  temp->next=NULL;  //if this is the first node  if(\*f==NULL)  \*f=\*r=temp;  else //insert at the end  {  (\*r)->next=temp;  \*r=temp;  }  }  intqdelete(struct node \*\*f, struct node \*\*r)  {  struct node \*q;  int x;  q=\*f;  if(q==NULL)  {  printf("Empty queue\n");  return -1;  }  else  {  x=q->data;  if(\*f==\*r) //only one node  \*f=\*r=NULL;  else  {  \*f=q->next;  free(q);  return x;  }  }  } |