KONGU ENGINEERING COLLEGE, PERUNDURAI, ERODE – 638 052

EVEN SEMESTER 2016 – 2017

MODULE TEST – I

Roll No.....

Programme : B.E Branch : CSE Semester : III	Date : 9.8.2016 Time : 2.45pm to 4.15pm
Course Code :14CST31 Course Name : Data Structures	Duration : 1 ½ Hours Max. Marks : 50

PART - A (10 X 2 = 20 Marks)

ANSWER ALL THE QUESTIONS

1. Define Data structure.
2. Differentiate between linear and non-linear data structures.
3. Specify the condition under which the stack will be empty.
4. Convert the following infix expression to postfix (a+b*c)/(d+e*f)*g+h
5. Write the procedure to perform enqueue operation in queue.
6. List any four real-time applications of queue.
7. Evaluate the postfix expression 5 7 7 / * 2 3 * + using stack.
8. Write a function to test whether the linked list is empty or not.
9. Mention the methods to allocate and deallocate memory dynamically.
10. Write the syntax for creating structure of a node.

PART - B (3 X 10 = 30 Marks)

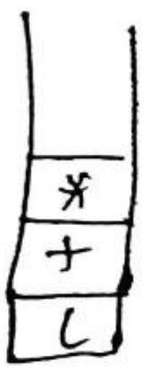
ANSWER ANY THREE QUESTIONS

11. Write a C program to implement the concept of Last In First Out (LIFO). (10)
12. List the operations of circular queue and explain each of them with its routine. (10)
13. a) Check whether the parenthesis is balanced or not (4)
    i) (((A*B)+(C/D))
    ii) (2+(((3+4)*(5*6)))
    b) Recursive function follows the concept of stack. Justify your answer. (6)
14. Write the procedure to insert an element at first and delete an element in the middle using singly linked list. (10)

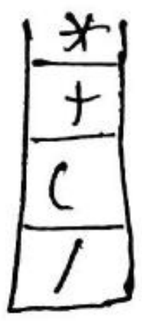
14CST31 - Data Structures

Part-A

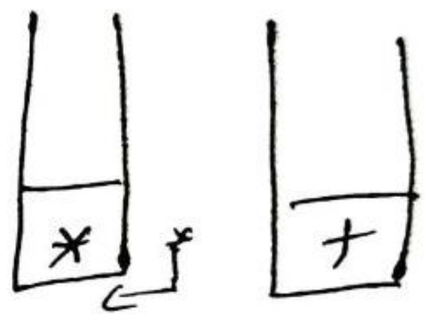
1. Way of organising the data. Eg. Stack ADT, Queue ADT
2. Linear - The data are arranged in sequential / ordered manner
   Eg - Linked list
   Non-Linear - The data are not present in sequential manner
   Eg - Trees
3. No elements in stack or stack top = -1.
4. (a+b*c)/(d+e*f)*g+h



abc*+



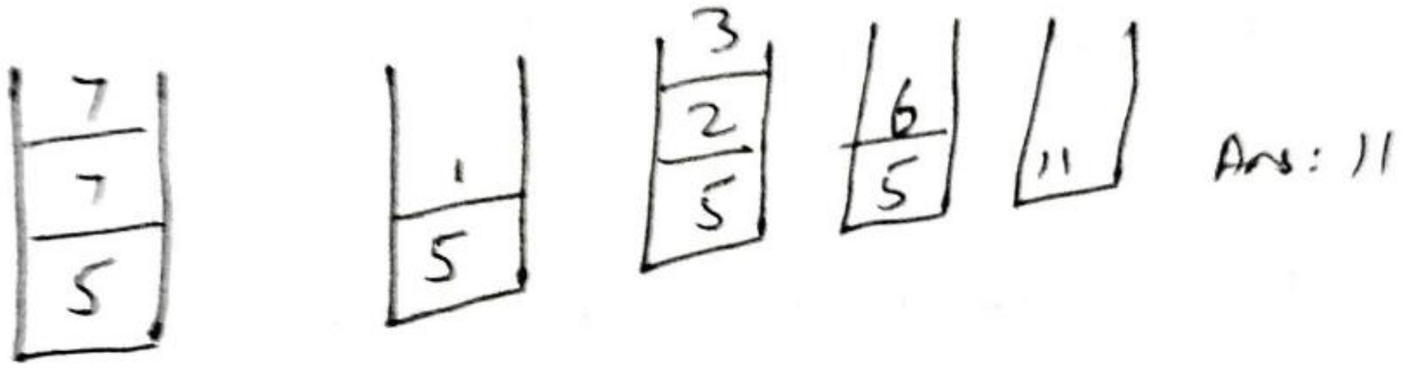
abc*+def*+/



abc*+def*+/g*h+

5.
```
Enqueue()
{ if (rear > max)
        printf ("Queue Overflow");
  else if (is empty())
  {  front = 0;
     rear = 0;
     a[rear] = x;
  }
  else
  {
     rear++;
     a[rear] = x;
  }
}
```

6. * Jobs to printer
   * queuing theory
   * calls to large companies
   * In large universities when resources are limited, students must sign a waiting list if all terminals are occupied

7. 

Ans: 11

8. is empty()
```
{
    if (head == NULL)
        printf("list empty");
}
```

9. Malloc() to allocate
   free() to deallocate

10. struct node
```
{
    int data;
    struct node *next;
};
```

Part-B

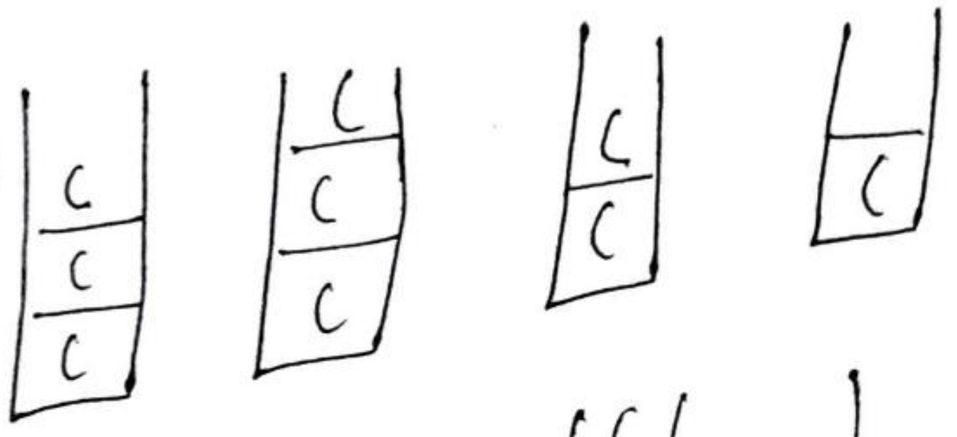
1. void push (int ele)
{ if (top<=max)
{ top++;
a[top]=ele;
}
else
printf("stack overflow");
}
void pop()
{ if (top<=0)
printf("stack empty");
else
top--;
}
Void top()
{
printf("top element = %d", a[top]);
}

12. Void enqueue (Int ele)
{ rear = (rear+1)%n;
if (rear == front)
printf("queue is full");
else
a[rear] = ele;
}

Void dequeue()
{ if (front == rear)
printf("Queue empty");

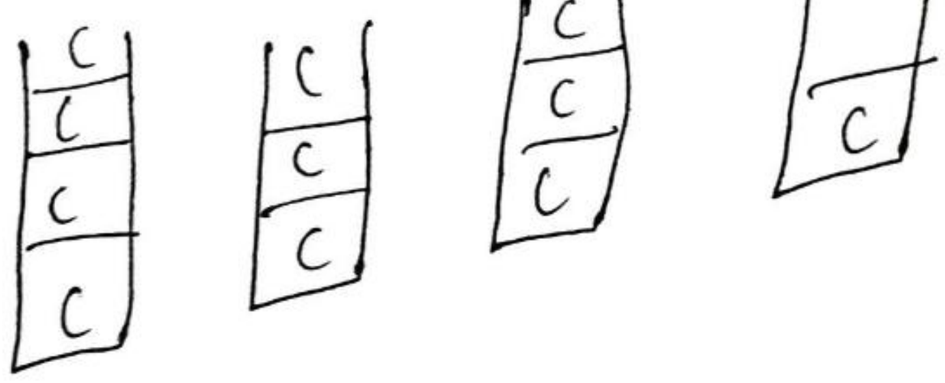
```
else
    { front = (front+1)%n;
      x = a[front];
      printf(" Dequeued element is %d", a[front]);
    }
}
```

13. (i)



-> not balanced

(ii)



-> not balanced

b. Recursive function follows the concept of stack.

* during function call the important information that needs to be saved as register values and then return the address
* The control is transfered to the new function which is free to replace the registers with its values.
* if it calls the function again it follows the same steps
* so the values are stored in the order of stack.

Insert at the first

```
void insbeg(int value)
{
    vae = (struct node *) malloc (size of (struct node));
    vae -> data = value;
    if (head == NULL)
    {
        head = vae;
        head->next = NULL;
    }
    else
    {
        vae -> next = head;
        head = vae;
    }
}
```

Delete at middle

```
Void deletemid ( )
{
    if (head == NULL)
        printf ("list is empty");
    else
    {
        temp = head;
        while (temp -> data != val)
        {
            val = temp;
            temp = temp -> next;
        }
        vae -> next = temp -> next;
    }
}
```

K.Venu	
Name and Signature of Hall Supdt. with Date	

KONGU ENGINEERING COLLEGE

Estd . 1984

PERUNDURAI, ERODE - 638 052.

(Autonomous)

Name of the Student	S. RADHIKA DEVI	Register No.	1	5	C	S	R	1	5	9
Programme	B.E	Branch & Semester	COMPUTER SCIENCE & ENGINEERING & SEMESTER III							
Course Code and Name	14CST31 & DATA STRUCTURES.	Date	09.08.2016			No. of Pages Used		13.		

MARKS TO BE FILLED IN BY THE EXAMINER

PART - A		PART - B			Grand Total Max. Marks : 50
Question No.	Max Marks : 2	Question No.		Max Marks : 10	
1	2	11	i)	9	45
2	2		ii)		
3	2	12	i)	9	
4	2		ii)		
5	2	13	i)	—	90%
6	2		ii)		
7	2	14	i)	9	V.Good
8	1		ii)		S. Radhika Devi
9	1				
10	2				
TOTAL	18	TOTAL		27	

Total Marks in Words : NINE ZERO

INSTRUCTION TO THE CANDIDATE

1. Check the Question Paper, Programme, Course Code, Branch Name etc., before answering the questions.
2. Use both sides of the paper for answering questions.
3. POSSESSION OF ANY INCRIMINATING MATERIAL AND MALPRACTICE OF ANY NATURE IS PUNISHABLE AS PER RULES.

S.V.KOGILAVANI
Name of the Examiner

S Kogilavani 10/8/16
Signature of the Examiner with Date

PART - A.

1. DATA STRUCTURE:

Data structure is defined as way of organizing the data.

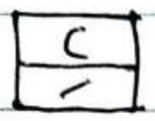
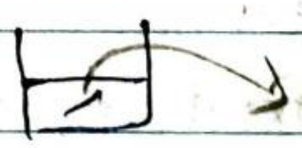
2. Linear	Non-Linear
eg) list, queue, set, stack	eg). tree, graph, polynomial, matrices
These are arranged in linearly	These are arranged non-linearly but joined with other
like, first node connected second node. in list.	Likewise, stem is connected to all branches in tree.

3. condition under which the stack will be empty:

If (top == -1), the stack is empty. or stack is not yet initialized.

4. (a+b*c)/(d+e*f)*g+h.

current token	stack	postfix
(	(c)	
a	(c)	a
+	(+ c)	a
b	(+ c)	ab

*		ab
c		abc
)		abc*+
/		abc*+
(		abc*+
d		abc*+d
+		abc*+d
e		abc*+de
*		abc*+de
f		abc*+def
)		abc*+def*+
*		abc*+def*+/
g		abc*+def*+/g
+		abc*+def*+/g*
h		abc*+def*+/g*h
EOF		abc*+def*+/g*h+

5. procedure to perform enqueue operation in queue:

Program:

```
void enqueue()
{
int ele;
if (rear>(MAX-1))
  Printf("Queue is overflow\n");
else
  if (front==-1)
    front=0;
  printf("enter the element");
  scanf("%d", &ele);
  rear++;
  queue[rear]= ele;
}
```

procedure:

step 1: start.

step 2: check the condition if (rear>(MAX-1)), then print Queue is overflow.

step 3: Otherwise, check front equal to -1, then assign front equal to 0. get the element, increament the rear by 1 and assign the element in queue of rear.

step 4: stop.

6. four real time applications of queue:

In ration shop, in theatres, near water tanker lorry, near free food service, in temples during crowd etc., are real time applications of queue.

7. Evaluation: 5 7 7 / * 2 3 * +

5	5	-
7	7 5	-
7	7 7 5	-
/	5	7 / 7 = 1
*	1 5	5 * 1 = 5
2	2 5	-
3	3 2 5	-
*	5	2 * 3 = 6
+	6 5	5 + 6 = 11
EOF	11	popped to display as a result 11.

8. Function to test linked list is empty or not:

```
void isempty()
{
    if (head->data == '\0')
        printf("The linked list is empty or not yet initialized");
    else
```

printf ("The linked list is not empty");

3.

9. Method to allocate memory dynamically:

Using malloc, memory is allocated dynamically wherever we want.

Method to deallocate memory dynamically:

Using dealloc, memory is deallocated dynamically wherever we want.

10. Syntax for creating structure of a node:

```
struct node
{
  int num;
  struct node *next;
}head *;
```

PART - B.

11. C-program to implement the concept of Last In First Out (LIFO).

Implementation of stack and its operations.

```
#include <stdio.h>
# include < stdlib.h>
# define MAX 10
int stack[MAX], top=-1;
void push();
void pop();
void topl();
void fsfull();
void isempty();
void display();
void main()
{
int ch;
printf(" 1.push \n 2.pop \n 3.topl \n 4.full \n 5.empty \n 6.display \n
                                                    7.exit \n);
do
{
print f ["enter the choice");
scanf("%d", &ch);
switch (ch)
{
 case 1: push();
         break;
```

```
        case 2: pop();
                break;
        case 3: top1();
                break;
        case 4: isfull();
                break;
        case 5: isempty();
                break;
        case 6: display();
                break;
        case 7: exit(0);
        }
    }while(1);
}
void push()
{
    int ele;
    if (top >= MAX-1)
        printf("the stack is full\n");
    else
        printf("enter the element\n");
        scanf("%d", &ele);
        top++;
        stack[top]= ele;
}
void pop()
{
    int ele;
```

```
if (top == -1)
    printf("the stack is empty");
else
    ele = stack[top];
    top--;
}
void top1()
{
  if (top == -1)
    printf("The stack is empty");
  else
    printf("The top value is %d", stack[top]);
}
void isfull()
{
  if (top >= MAX-1)
    printf("The stack is full \n");
  else
    printf("The stack is not full");
}
void isempty()
{
  if (top == -1)
    printf("The stack is empty\n");
  else
    printf("The stack is not empty\n");
}
```

```
void display()
{
  int i;
  for(i=top; i>=0; i--)
     printf("The stack element %d", stack[i]);
}
```

12. Operations of circular queue:

* Enqueue
* Dequeue
* Display are the operations of circular queue.

Routine of enqueue function in circular queue:

```
void enqueue()
{
  int ele;
  if(rear==(MAX-1)) && (front==0 || front==rear+1))
     printf("Circular queue is overflow");
  else
     if(front==-1)
        front=0;
  printf("enter the element");
  scanf("%d", &ele);
  rear=(rear+1)%MAX;
  queue[rear]=ele;
}
```

Routine of dequeue function in circular queue:

```
Void dequeue()
{
   Int ele;
   If (front == -1)
      printf("Circular queue is underflow");
   else
      ele = queue[front];
      If (front == rear)
         front = rear = -1;
      else
         front = (front+1)%MAX;
      }
}
```

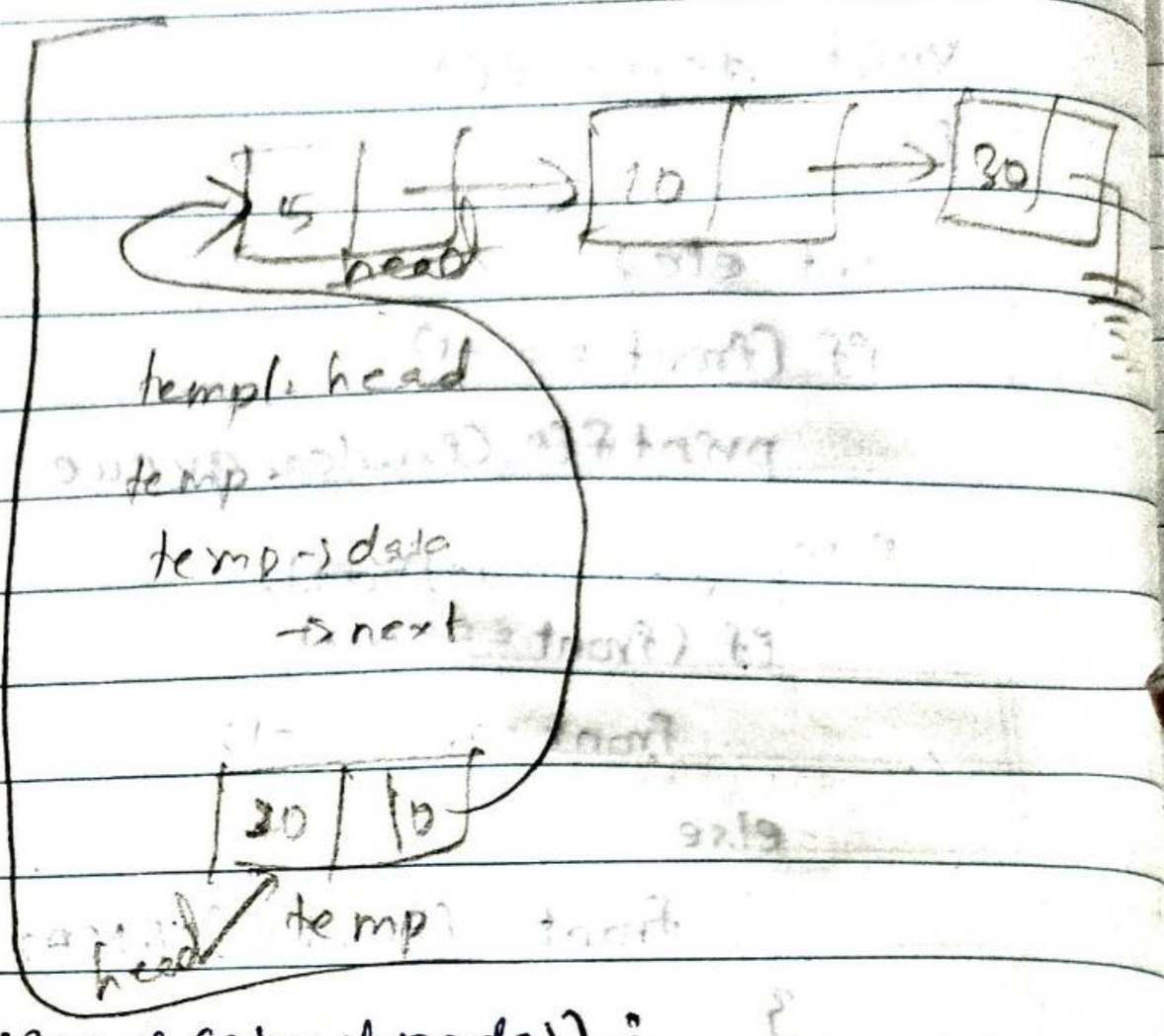
Routine of display function in circular queue:

```
void display()
{
 Int i
  if (front == -1 && rear == -1)
      printf("Circular queue is underflow");
  else
      For (i = front; i != rear; i = (i+1)%MAX)
         printf("The queue element %d", queue[i]);
  printf("%d", queue[rear]);
}
```

14. Procedure to insert an element at first:

Program:

```
Pnt
void Pnsert (Pnt num, Pnt pos)
{
  struct node *temp;
  Pnt num;
  while (1)
  {
    if (num != -1)
    {
      temp=
  temp=head;
  temp = (struct node *) malloc (size of (struct node));
  temp → data = num;
  temp → next = null;
  if (pos == 1)
    {
    temp → next = head;
     head = temp;
    }
  else
     break;
}.
```



Procedure:

Step 1: Start.

Step 2: declare the pointers temp, and allocate the memory for the struct node pointer using malloc.

step 3: get the num as argument is assigned to temp data and assign null character to temp next.

Step4: Check if the position to be inserted is first then assign head to temp next and temp to head.

step5: Otherwise, get out from the program.

step6: Stop.

procedure to delete an element in the middle:

program:

```
Int delete (Int num)
{
 Struct node *prev, *cur;
 cur = head;
while (curb = '\0')
 {
  If (cur → data = num)
    {
   if (cur = head)
     break;
   else
     prev → next = cur → next;
     free (cur);
     return 0;
  }
else
```

```
        {
            prev = cur;
            cur = cur->next;
        }
    }
    printf("%d is not found", num);
}
```

procedure:

Step 1: Start.

Step 2: Declare the pointers previous, current. Assign head to current.

Step 3: Introduce while loop, it gets executed until the current → next gets '\0' character.

Step 4: Check the current data is number, that current data will be head means break from the program.

Step 5: Otherwise, assign current next to previous next, free of current and return the value 0.

Step 6: If the current data is not a given number, then assign current to previous, current next to current.

Step 7: If the number is not found in the list, then print the given number is not found.

Step 8: Stop