CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test II – November 2016

Sub:	DATA STRUCTURES AND APPLICATIONS			Code:	15CS33
Date:	03/11/2016	Duration: 90 mins Max Marks: 50	Sem: III -D	Branch:	CSE

Note: Answer FIVE full Questions **Total marks: 50**

PART-A

SNo	Questions	Marks	OBE CO	OBE RBT
1	Define Stack. Mention the operations of stack and Implement the operations in C using functions.	10	CO2	L3
2	Write an Algorithm to convert infix to postfix expression. Trace the same on following infix expression. ((a+(b-c)*d)^e+f)	10	CO4	L3
3	a)Write a C program to implement Ackerman's function using recursion. b) Write a C program to implement Tower of Hanoi using recursion	5 5	CO2	L3
4	Define Circular Queue. Write a C program to implement Circular queue operations? a)insert b)delete c)display	10	CO2	L3
5	a)Define Linked List. Explain how it is different from an Array. b)Explain Multiple Stack concept along with Push( ) and Pop( ) function	5 5	CO2	L3 L2
6	Write a C program to Implement Singly Linked List(SLL) operations. a)insert front b)delete front c)display	10	CO2	L3
7	Write a C program to Implement Singly Circular Linked List(SCLL) operations. a)insert at given position b)delete front c)display	10	CO2	L3

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Explain data structures – Array, Stack, Queue, List, Tree, Graphs, Hash tables.	2	1	2	-	1	-	-	-	-	-	1	-
CO2:	Develop modules for Arrays, Stack, Queue, List, Tree, Graphs, Hash tables.	2	1	2	-	1	-	-	-	-	-	2	1
CO3:	Implement sorting and search Algorithm – bubble sort, selection sort, linear search, binary search	2	1	2	-	1	-	-	-	-	-	-	-
CO4:	Apply data structures and algorithm to typical problems – Infix to postfix conversion, postfix evaluation.	2	1	2	-	1	-	-	-	-	-	-	-
CO5:	Implement Traversal methods- In-order, Pre-order, post-order of tree	2	1	2	-	1	-	-	-	-	-	-	-
CO6:	Apply appropriate data structure to problem using hash tables and files.	2	1	2	-	1	-	-	-	-	-	2	1

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

1.Define Stack. Mention the operations of stack and Implement the operations in C using functions

⇒ STACK:

* A Stack is special type of data structure (an Ordered collection of items) where elements are inserted ~~from one end~~ & elements are deleted from the same end.
* This position from where elements are inserted and from where elements are deleted is called top of the stack.
* The last element inserted is the first element to be deleted out, hence stack is called Last in first out (LIFO) data Structure.

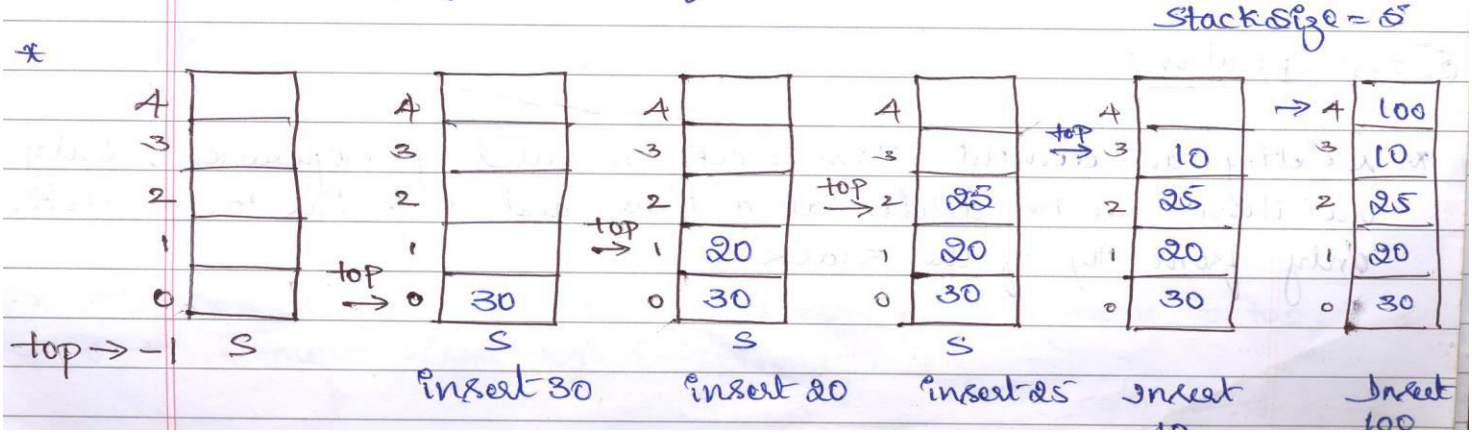
⇒ Stack operation:

* The various operations that can be performed on stacks are
  1) Push (Inserting an element on the top of the stack)
  2) Pop (Deleting an element from the top of stack)
  3) Display (Display contents of stack)

① PUSH operation

* Push operation used to insert an element into the stack. Only one item is inserted at a time and item has to be inserted only from top of the Stack.

*



* when elements are being inserted, there is a possibility of sta being full.
* Once stack is full, it is not possible to insert any element. If we try to insert an element, even when the stack is full results in overflow of stack.

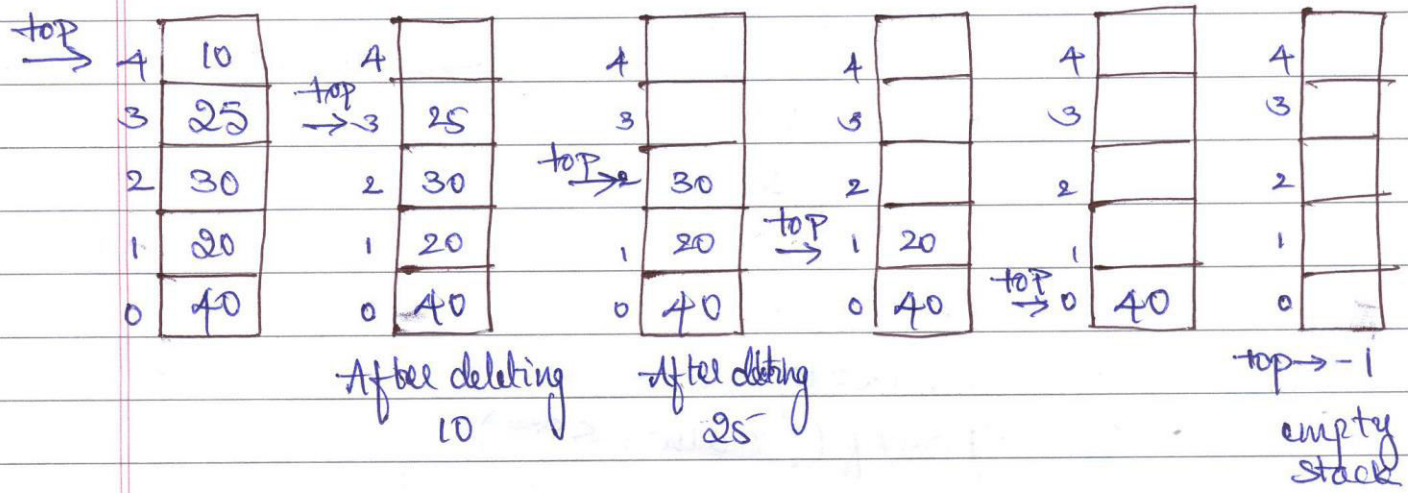
* In the stack figure with stack size 5, we can insert at the most 5 elements. After inserting 30, 20, 25 10 & 100 there is no space to insert any item. Then we can say stack is full.

/* C function to insert element into stack */

```
void push(int item)
{
    /* check for overflow of stack */
    if (top == stacksize - 1)
    {
        printf("Stack overflow\n");
        return;
    }
    top = top + 1;        /* Increment top by 1 */
    S[top] = item;        /* Insert into stack */
}
```

Pop operation:

* Deleting an element from stack is called popoperation. Only one item can be deleted at a time and item has to be deleted only from top of the stack.



* When elements are being deleted, there is a possibility of Stack being empty.
* When Stack is empty, it is not possible to delete any item. If we are trying to delete an element from an empty Stack result in Stack underflow.

```
/* C Function to delete an element from stack */

int pop()
{
    int item_deleted;
    if (top == -1)      /* stact is empty */
        return 0;
    else
        itemp_deleted = S[top];     /* Access the item to
        top--;                          delete */
        return itempdeleted;
}
```

③ Display Operation

* The contents of Stack are displayed from bottom to top. i.e 0 to element where top is pointing.

```
void display()
{
    int i;
/* stack is empty */
if(top==-1)
{
    printf("Stack is empty");
    return;
}
/* Display contents of stack */
printf("The contents of stack are\n");
for(i=0; i<=top; i++)
    printf("%d\n", s[i]);

}
```

Write an Algorithm to convert infix to postfix expression. Trace the same on following infix expression. ((a+(b-c)*d)^e+f)

* Scan the infix expression from left to right

* If the scanned character is Right parenthesis (')'), push into stack.

* If the scanned character is an Operand or Alphabet, directly place it in postfix expression.

* If the scanned character is left parenthesis ('('), then pop all the symbol from the stack until we get matching Right parenthesis

* If the scanned character is an operator, then compare the precedence of symbol and precedence of symbol which is on top of the stack. i.e if the precedence of stack symbol is greater than precedence of scanned symbol, pop the symbol from stack and place it in postfix expression.

i.e
```
while(precedence(s[top]) >= prodence(symbol))
{
    Postfix[j] = s[top--];
    j++;
}
```

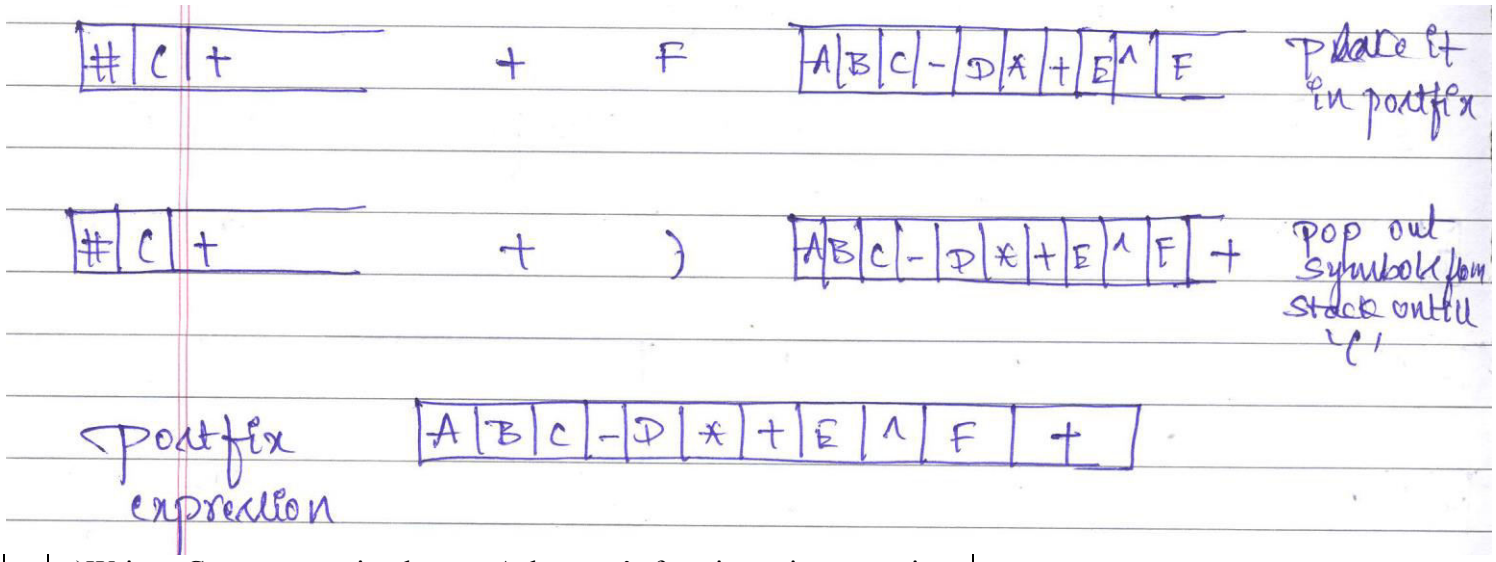
if not push the scanned symbol into stack

```
if (precedence(s[top]) != precedence(symbol))
    s[++top] = symbol;
else
    top--;
```

* Pop remaining symbols and place them in postfix expression.

infix expression: ((A+(B-C)*D)^E+F)

Stack	top of the stack	Symbol	Postfix	Operation
#	#	(		Push into stack
# (	(	(		Push into stack
# ((	(	A	A	Place it in postfix
# ((	(	+	A	+ has higher precedence. Push into stack
# ((+	+	(	A	Push into stack
# ((+ (	(	B	A B	Place it in postfix
# ((+ (	(	-	A B	- has higher precedence push into stack
# ((+ (-	-	C	A B C	Push into stack
# ((+ (-)	-	)	A B C -	Pop out until we get matching '('
# ((+	+	*	A B C -	* has higher precedence than + push
# ((+ *	*	D	A B C - D	Place it in postfix
# ((+ *)	*	)	A B C - D * +	Pop until we get matting '('
# (	(	^	A B C - D * +	Push into stack
# (^	^	E	A B C - D * + E	Place it in postfix
# (^)	^	+	A B C - D * + E ^	+ has less precedence than ^, pop out ^ from stack Place it in Postfix



3	a)Write a C program to implement Ackerman's function using recursion. b) Write a C program to implement Tower of Hanoi using recursion

```
#include<stdio.h>

void tower (int n, char src, char temp, char dest)
{
    if (n==1)
    {
     printf("Move disk %d from %c to %c", n, src, dest);
     return;
    }

  // Move n-1 disk from source to temp
    tower (n-1, src, dest, temp);

  // Move nth disk from source to destination
  printf("Move disk %d from %c to %c", n, src, dest);

  // Move n-1 disk from temp to destination
    tower (n-1, temp, src, dest);
}

int main()
{
    int n;
  printf("Enter the number of disks\n");
  scanf("%d", &n);
      tower (n, 'A', 'B', 'C');
  return 0;
}
```

```
int Ack(int m, int n)
{
    if (m==0)
        return (n+1);
    else if ((m>0) && (n==0))
        return (Ack(m-1, 1));
    else
        return Ack(m-1, ack(m, n-1));
}

main()
{
    int m, n, val;
    printf("enter the value of m, n");
    scanf("%d %d", &m, &n);
    val = Ack(m, n);
    printf("Ack(%d, %d) = %d", m, n, val);
}
```

Define Circular Queue. Write a C program to implement Circular queue operations? a)insert b)delete c)display

* In Circular Queue, the elements of a given queue can be stored efficiently in array, so that end of the queue is followed by the front of the queue.

* Initially when Stack is empty f=-1, r=-1. To insert an item, the rear index can be incremented first.

r = (r+1) % Queue_size

```
#include<stdio.h>
#define size 5
int f,r,q[20];
void insertq(int item,int *count)
{
      if(*count==size)
            printf("overflow in queue\n");
      else
            r=(r+1)%size;
            q[r]=item;
            *count+=1;
}
void deleteq(int *count)
{
      if(*count==0)
            printf("underflow\n");
      else
            printf("the deleted element is %d\n",q[f]);
            f=(f+1)%size;
            *count-=1;
}
void display(int *count)
{      int i;
      if(*count==0)
         printf("underflow\n");
       else
            printf("the contents of queue\n");
            for(i=1;i<=*count;i++)
            {
                  printf("%d\n",q[f]);
                  f=(f+1)%size;
             }
}



int main()
{
   int ch,item,count=0;
   f=0;r=-1;
   for(;;)
   {
      printf("1:insert \n 2:delete\n3:display\n4:exit\n");
      printf("enter your choice\n");
      scanf("%d",&ch);
      switch(ch)
      {
        case 1:
              printf("enter the item to be inserted\n");
              scanf("%d",&item);
              insertq(item,&count);
              break;
        case 2: deleteq(&count);
              break;
        case 3:display(&count);
              break;
        case 4: return 0;
```

```
        case 5:printf("invalid choice\n");
       }
    }
 }
```

6.SLL

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
void create();
void insert_atfirst();
void insert_atlast();
void insert();
void delete_bypos();
void delete_byval();
void display();
int menu();
struct node
{
    int data;
    struct node *next;
}*first=NULL,*last=NULL,*temp=NULL;
void main()
{
    int ch;
    clrscr();
    do{
        ch=menu();

        switch(ch)
        {
            case 1:create();
                break;
            case 2:insert_atfirst();
                break;
            case 3:insert_atlast();
                break;
            case 4:insert();
                break;
            case 5:delete_bypos();
                break;
            case 6:delete_byval();
                break;
            case 7:display();
                break;
            case 8:exit(0);
            case 9:clrscr();
            default:printf("\nError-->Enter a valid choice!!");
                exit(0);
        }
    }while(1);
}

void create()
{
    temp=(struct node*)malloc(sizeof(struct node));
    printf("\nEnter the Data in the Node:");
    scanf("%d",&temp->data);
    temp->next=NULL;
```

```
    if(first==NULL)
    {
        first=temp;
        last=temp;
    }
    else
    {
        last->next=temp;
        last=temp;
    }
}

void insert_atfirst()
{
    temp=(struct node*)malloc(sizeof(struct node));
    printf("\nEnter Element:");
    scanf("%d",&temp->data);
    temp->next=first;
    first=temp;
}
void insert()
{
    struct node *t, *t1;
    int pos, count=1,AB;
    printf("\nEnter Position to be inserted:");
    scanf("%d",&pos);
    printf("\nAfter[1] or Before[2]:");
    scanf("%d",&AB);
    temp=(struct node*)malloc(sizeof(struct node));
    printf("\nEnter data:");
    scanf("%d",&temp->data);
    temp->next=NULL;
    t=t1=first;
    while(t!=NULL)
    {
        if(pos==count)
            break;
        else
        {
            t=t1;
            t=t->next;
            count++;
        }
    }
    if(AB==1)
    {
        temp->next=t->next;
        t->next=temp;
    }
    else if(AB==2)
    {
        temp->next=t;
        t1->next=temp;
    }
    else
        printf("\nInvalid Input");
}

void insert_atlast()
{
    temp=(struct node*)malloc(sizeof(struct node));
```

```
    printf("\nEnter value to be inserted at last:");
    scanf("%d",&temp->data);
    last->next=temp;
    last=temp;
}

void delete_bypos()
{
    struct node *t,*t1;
    int pos, count=1;
    printf("\nEnter the Position of the element you would like to delete:");
    scanf("%d",&pos);
    t=t1=first;
    while(t!=NULL)
    {
        if(pos==count)
        t1=t;
        t=t->next;
        count++;
    }
    if(pos==1)
    {
        first=t->next;
        printf("\nExceuted-->First Node is deleted!!");
    }
    else if(t==last)
    {
        t1->next=NULL;
        free(t);
        printf("\nExecuted-->Last Node is deleted!!");
    }
    else
    {
        t1->next=t->next;
        free(t);
        printf("\nExecuted-->Node has been deleted!!");
    }
}

void delete_byval()
{
    int val, count=1;
    struct node *t,*t1;
    printf("\nEnter value:");
    scanf("%d",&val);
    t=t1=first;
    while(t!=NULL)
    {
        if(val==t->data)
        break;
        else
        {
        t=t->next;
        count++;
        }
    }
    if(t==first)
    {
        first=t->next;
        free(t);
    }
```

```
    else if(first==last)
    {
        t1->next=NULL;
        free(t);
        last=t1;
    }
    else
    {
        t1->next=t->next;
        free(t);
    }
}
void display()
{
    temp=first;
    while(temp!=NULL)
    {
    printf("|%d|%d| --> ",temp->data,temp->next);
    temp=temp->next;
    }
}

int menu()
{
    int ch;
    printf("\n------------");
    printf("\nSingle Linked List");
    printf("\n------------");
    printf("\n1.Create\n2.Insert at first\n3.Insert at last\n4.Insert at Middle\n5.Delete by
Position\n6.Delete by Value\n7.Display\n8.Exit");
    printf("\n\n-->Enter Your Choice:");
    scanf("%d",&ch);
    return ch;
}
```

7.SCLL

```
#include<stdio.h>
#include<conio.h>

struct circular
{
    int i;
    struct circular *next;
};


struct circular *temp;
struct circular *head;
struct circular *p;
struct circular *mid;
struct circular *move;

int cnt=0;


void create(void);
void insert(void);
void display(void);
void del(void);

void main()
{
    int ch=0;
    clrscr();
```

```
    while(ch!=5)
    {
        printf("\n1.CREATE");
        printf("\n2.INSERT");
        printf("\n3.DELETE");
        printf("\n4.DISPLAY");
        printf("\n5.EXIT");
        scanf("%d",&ch);


        if(ch==1)
        {
            create();
            cnt++;
            cnt++;
        }

        if(ch==2)
        {
            insert();
            cnt++;
        }
        if(ch==3)
        {
            del();
            cnt--;
        }

        if(ch==4)
        {
            display();
        }

        if(ch==5)
        {
            break;
        }
    }
    getch();
}
void create()
{
    head=(struct circular *)malloc(sizeof(struct circular));
    head->next=head;
    printf("ENETER THE DATA");
    scanf("%d",&head->i);
    temp=head;

    temp->next=(struct circular *)malloc(sizeof(struct circular));
    temp=temp->next;
    temp->next=head;
    printf("ENETER THE DATA");
    scanf("%d",&temp->i);

}
void insert()
{
    int add,t;

    printf("\n\t ENTER ANY NUMBER BETWEEN 1 AND %d",cnt);
    scanf("%d",&add);
    p=head;
    t=1;
    while(t<add)
    {
        p=p->next;
        t++;
    }
    printf("%d",p->i);
```

```
    clrscr();
    mid=(struct circular *)malloc(sizeof(struct circular));
    printf("ENETER THE DATA");
    scanf("%d",&mid->i);
    mid->next=p->next;
    p->next=mid;
}

void display()
{
    p=head;
    printf("%d-->",p->i);
    p=p->next;
    while(p!=head)
    {
        printf("%d-->",p->i);
        p=p->next;
    }
}

void del(void)
{
    int add,t;

    printf("\n\t ENTER ANY NUMBER BETWEEN 1 AND %d",cnt);
    scanf("%d",&add);
    p=head;
    t=1;
    while(t<add-1)
    {
        p=p->next;
        t++;
    }
    printf("%d",p->i);
    clrscr();
    mid=p->next;
    p->next=mid->next;

}
```