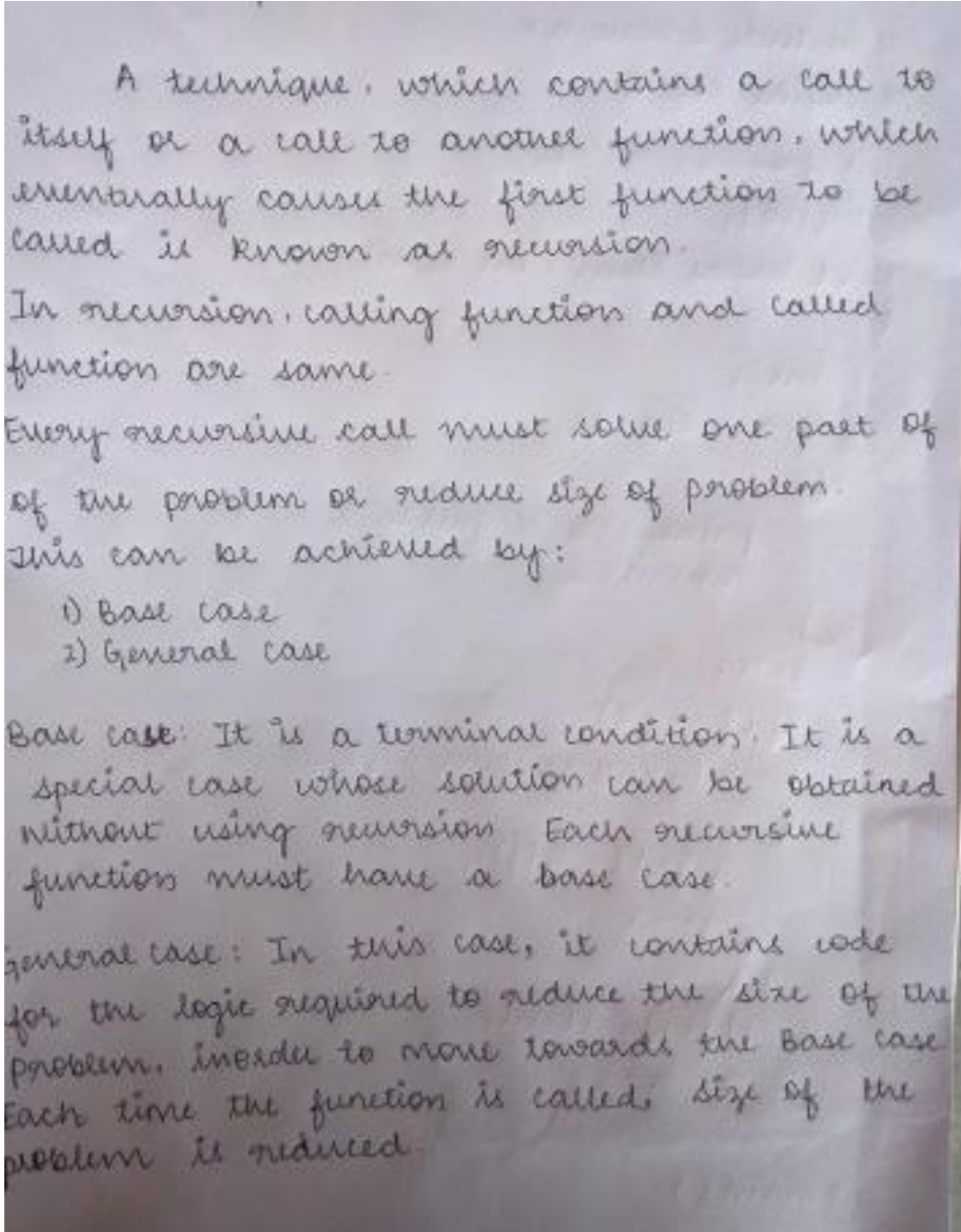


Internal Assessment Test 2 – October 2019 Scheme and Solution									
Sub:	Data Structures and Applications				Sub Code:	18CS32	Branch:	ISE	
Date:	15/10/2019	Duration:	90 mins	Max Marks:	50	Sem/Sec:	3 rd /A,B,C		OBE
<u>Answer any FIVE FULL Questions</u>							MARK S	CO	RB T
<p>1. a) Define recursion. What are the properties of recursive procedure?</p> 							[2+2]	CO3	L1

1. b) Write a C program to perform operations on a priority Queue.

[2+2+2]

CO3

L3

Answer:

```
b) Write a C program to perform operations on a priority queue.

#include <stdio.h>
#define size 5
int f = -1, r = -1;
int q[20];
void insert item (int item)
{
    int j;
    if (r == size - 1)
    {
        printf("Q is full\n");
        return;
    }
    j = r;
    if (f == -1)
    while (j >= 0 && item < q[j])
    {
        q[j+1] = q[j];
        j--;
    }
    q[j+1] = item;
    r = r + 1;
}

int main ()
{
    int choice, item;
}
```

```

for (i; i)
{
    printf ("1: insert, 2: delete, 3: display\n");
    scanf ("%d", & choice);
    switch (choice)
    {
        case 1: printf ("Enter the items to be inserted\n");
                scanf ("%d", & item);
                insert_item (item);
                break;
        case 2: delete_front ();
                break;
        case 3: display ();
                break;
        default: printf ("Invalid choice");
                return 0;
    }
}
}

```

What is a Queue/Linear Queue? List different types of Queue. Write C implementation for insertQ() and deleteQ() operation.

[2+2+3+3]

CO1

L3

A Queue is a special data structure, where elements are inserted from one end and elements are deleted from the other end.

The end from the elements are inserted is called rear end and where the elements are deleted is called front end. Queue is also called as first in first out data structure.

Types of Queue:

1. Linear Queue.
2. Circular Queue.
3. Double ended Queue
4. Priority Queue

program for insert Q()

```
void insert (int item)
{
    if (r == size - 1)
    {
        printf("Queue overflow");
        return;
    }
}
```

```
if (f == -1)
    f = 0;
r = r + 1;
q[r] = item;
```

Program for delete Q():

```
void delete ()
{
    if (f == -1 && r == -1)
    {
        printf("Queue underflow\n");
        return;
    }
    printf("The element deleted is: %d", q[f]);
    f = f + 1;
    if (f > r)
    {
        f = -1;
        r = -1;
    }
}
```

3a. Explain tower of Hanoi for n = 3 disks with example

[2+3]

CO4

L3

Answer:

There are three pegs say A, B and C. There are n discs and all placed one above the other, such that always a smaller disc is placed above larger disc.

Two pegs B and C are empty. All the discs from A to be transferred to needle C using needle B as temporary storage.

Rules to be followed:

- 1) Only one disc is moved at a time from one needle to another needle.
- 2) Smaller disc is on top of the larger disc at any time.
- 3) Only one needle can be used for storing intermediate disks.

- A → C, 1
- A → B, 2
- C → B, 1
- A → C, 3
- B → A, 1
- B → C, 2
- A → C, 1

3b. Write a note on Ackermann's Function. Calculate for $A(1, 3)$.

[2+3]

CO3

L3

Ackermann function with two arguments each of which can be assigned by any non-negative integer $0, 1, 2, \dots$

It can be defined as

a) if $m=0$, then $A(m, n) = n + 1$

b) if $m \neq 0$ but $n = 0$ then $A(m, n) = A(m-1, 1)$.

c) if $m \neq 0$ and $n \neq 0$, then $A(m, n) = A(m-1, A(m, n-1))$

Ackermann function is a recursive function not primitive and it grows very quickly and its size increases.

$$A(1, 3) \Rightarrow m=1, n=3.$$

$$A(1, 3) = A(0, A(1, 2))$$

$$A(1, 2) = A(0, A(1, 1))$$

$$A(1, 1) = A(0, A(1, 0))$$

$$A(1, 0) = A(0, 1)$$

$$A(0, 1) = 1 + 1 = 2$$

$$A(1, 0) = A(0, 1) = 2$$

$$A(1, 1) = A(0, A(1, 0)) = A(0, 2) = 3$$

$$A(1, 2) = A(0, A(1, 1)) = A(0, 3) = 4$$

$$A(1, 3) = A(0, A(1, 2)) = A(0, 4) = 5$$

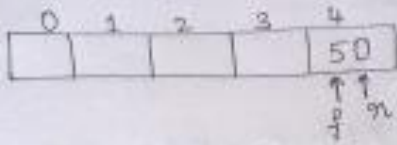
4. What is the disadvantage of a queue? How can this be overcome? Explain with an example.

3+1+3+3

CO3

L2

Consider the following queue as shown below:-



This situation arises when 5 elements say 10, 20, 30, 40, 50 are inserted and then deleting first four elements.

- If we try to insert an element say 60, since rear (r) has the value MAX-1 we get an overflow condition and it is not possible to insert any elements.
- Even though queue is not full, in this case it is not possible to insert any items into the queue.

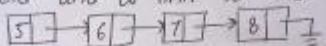
This disadvantage can be overcome if we use the circular queue.

5. What is a list? Explain different types of lists. Demonstrate with a C program how singly linked list can be used to implement a Queue. 2+2+2+2+2 CO4 L3

Implement a Queue.

Ans: Linked list:-

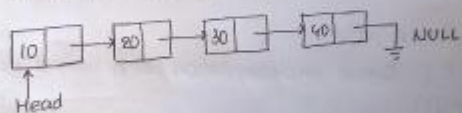
A linked list is a linear data structure in which the elements are not stored at contiguous memory locations. A linked list consists of nodes where each node contains a data field and a link to the next node in the list.



Types of linked list:-

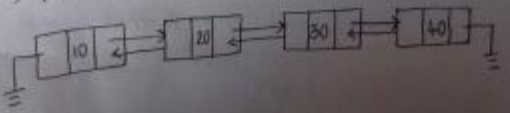
① Singly linked list:-

It is a collection of nodes linked together in a sequential way where each node of singly linked list contains a data field and an address field which contains reference of next node. In singly linked list address field of last node must contain a NULL character.



② Doubly linked list:-

It is a link data structure that consists of a set of sequentially linked nodes. Each node contains 3 fields called links that are references to the previous and to the next node in sequence of nodes. The beginning and ending nodes, previous and next links respectively point to NULL.



③ Circular linked list:-

It is a linked list in which the last node points to head or front node making the data structure to look like a circle. A circular linked list node can be implemented using singly or doubly linked list.

C program to implement Queue using singly linked list:-

```
#include <stdio.h>
struct Node
{
    int data;
    struct Node *next;
    *front = NULL, *rear = NULL;
}
void insert (int);
void delete ();
void main()
{
    int choice, value;
    printf ("Queue implementation using linked list:\n");
    while (1)
    {
        printf ("1: insert \n 2: delete \n 3: exit \n");
        printf ("Enter your choice:");
        scanf ("%d", &choice);
        switch (choice)
        {
            case 1: printf ("Enter the value to be inserted:");
                    scanf ("%d", &value);
                    insert (value);
                    break;
            case 2: delete ();
                    break;
            case 3: exit (0);
            default: printf ("Invalid choice \n");
        }
    }
}
```

```

}
}
}
void insert (int value)
{
    struct Node *newNode;
    newNode = (struct Node *) malloc (sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (front == NULL)
    {
        front = rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
}
void delete()
{
    if (front == NULL)
    {
        printf ("Queue is empty\n");
    }
    else
    {
        struct Node *temp = front;
        front = front->next;
        printf ("deleted element : %d\n", temp->data);
        free (temp);
    }
}
}

```

6a. Implement the concept of multiple stack using arrays in C. Explain with suitable example pictorially.

[4+2+2+2]

CO4

L3

Ans- Multiple stack :-

When a stack is created using single array, we can not be able to store large amount of data, thus this problem is rectified using more than one stack in same array of sufficient array. This technique is called multiple stack.

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	d	c	b	a

C - Program :-

```
#include <stdio.h>
#include <malloc.h>
#define MAX 10
int stack[MAX], topA = -1, topB = MAX;
void pushA(int val)
{
    if (topA == topB - 1)
        printf("\n overflow");
    else
        topA += 1;
        stack[topA] = val;
}
int popA()
{
    int val;
    if (topA == -1)
        return -1;
}
```

```

printf("\n underflow");
val = -999;
}
else
    val = stack[topA];
    topA--;
}
return val;
}

void pushB(int val)
{
    if(topB - 1 == topA)
    {
        printf("\n overflow");
    }
    else
    {
        topB--;
        stack[topB] = val;
    }
}

int popB()
{
    int val;
    if(topB == MAX)
    {
        printf("\n underflow");
        val = -999;
    }
    else
    {
        val = stack[topB];
        topB++;
    }
}
}

```

```

int main()
{
    int option, val;
    while(1)
    {
        printf("\n 1. Push an element into stack A");
        printf("\n 2. Push an element into stack B");
        printf("\n 3. Pop an element from stack A");
        printf("\n 4. Pop an element from stack B");
        printf("\n 5. Exit");
        printf("\n Enter your choice");
        scanf("%d", &option);
        switch(option)
        {
            case 1: printf("\n Enter the value to push on stack A:");
                    scanf("%d", &val);
                    pushA(val);
                    break;
            case 2: printf("\n Enter the value to push on stack B");
                    scanf("%d", &val);
                    pushB(val);
                    break;
            case 3: if(val != -999)
                    printf("\n The value popped from stack A
                    = %d", val);
                    break;
            case 4: if(val != -999)
                    printf("\n The value popped from stack B
                    = %d", val);
                    break;
        }
    }
    return 0;
}

```

7. a Write C functions to perform the following operation on a doubly linked list:

3+3

CO4

L3

7. a Concatenation of 2 lists in doubly linked list:-

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *prev;
    struct node *next;
};
struct node *list = NULL;
struct node *list_last = NULL;
struct node *even = NULL;
struct node *even_last = NULL;
struct node *odd = NULL;
struct node *odd_last = NULL;
struct node *current = NULL;
void insert (int data)
{
    struct node *link = (struct node *) malloc (sizeof (struct node));
    link->data = data;
    link->prev = NULL;
    link->next = NULL;
    if (data % 2 == 0)
    {
        if (even == NULL)
        {
            even = link;
            return;
        }
        else
        {
            current = even;
            while (current->next != NULL)
            {
                current = current->next;
            }
            current->next = link;
            link->prev = current;
        }
    }
    else
    {
        if (odd == NULL)
        {
            odd = link;
            return;
        }
        else
        {
            current = odd;
            while (current->next != NULL)
            {
                current = current->next;
            }
            current->next = link;
            link->prev = current;
        }
    }
}
```

```
link->prev = current;
}
else
{
if (odd == NULL)
{
odd = link;
return;
}
else
{
current = odd;
while (current->next != NULL)
{
current = current->next;
}
current->next = link;
odd->last = link;
link->prev = current;
}
}
}

void print-backward (struct node *head)
{
struct node *ptr = head;
printf("\n [last] =>");
while (ptr != NULL)
{
printf ("%d => ", ptr->data);
ptr = ptr->prev;
}
printf (" [head] \n");
}

void printlist (struct node *head)
{
struct node *ptr = head;
printf ("\n [head] =>");
while (ptr != NULL)
```

```

printf ("%d (=>)", ptr->data);
ptr = ptr->next;
}
printf ("\n");
}
void combine ()
{
struct node *link;
list = even;
link = list;
while (link->next != NULL)
{
link = link->next;
}
link->next = odd;
odd->prev = link;
link->last = link;
}
int main ()
{
int i;
for (i = 1; i <= 10; i++)
{
insert(i);
printf ("Even : ");
printlist (even);
printf ("Even (R) ");
print_backward (even->last);
printf ("Odd : ");
printlist (odd);
printf ("Odd (R) ");
print_backward (odd->last);
combine ();
printf ("Combined list : \n");
printlist (list);
printf ("Combined list (R) : \n");
print_backward (list->last);
}
return 0;
}

```

Reverse a doubly linked list:-

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node * next;
    struct Node * prev;
};
void reverse (struct Node ** head_ref)
{
    struct Node * temp = NULL;
    struct Node * current = *head_ref;
    while (current != NULL)
    {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }
    if (temp != NULL)
        *head_ref = temp->prev;
}
void push (struct Node ** head_ref, int new_data)
{
    struct Node * new_node = (struct Node *) malloc (sizeof
        (struct Node));
    new_node->data = new_data;
    new_node->prev = NULL;
    new_node->next = (*head_ref);
    if ((*head_ref) != NULL)
    {
        ((*head_ref)->prev) = new_node;
        (*head_ref) = new_node;
    }
}
```

7. b) Write a C function to evaluate a given polynomial (For eg: $5x^3 + 2x^2 + 9$) using single linked list for a given value of X

[2+2]

CO3

L3

```
void evaluate_poly()
```

```
{
    int x, sum = 0;

    printf("\nEnter the value of x: ");
    scanf("%d", &x);

    POLY temp;

    temp = first;

    while(temp != NULL)
    {
        sum = sum + (temp->coeff*pow(x, temp->px));
        temp = temp->link;
    }

    printf("\nThe sum is: %d", sum);
}
```


8.a What is the disadvantage of singly linked list? How is it overcome? Explain front

2.5+1.5+3+
3

CO4

L3

insertion and end deletion on a doubly linked list.

Ans - Disadvantages of singly linked list:-

⇒ More memory is required to store elements in linked list.
⇒ If we have to go to a particular element then we have to go through all those elements that come before that element.

⇒ We can't traverse it from last but only from beginning.

It can be overcome by using doubly linked list. As we have 2 links #right and left, we can go through a particular element without passing through all elements.

Code for front insertion:-

```
struct Node
{
    int data;
    struct Node * next;
    struct Node * prev;
};

void push(struct Node ** head_ref)
{
    struct Node * new_node = (struct Node *) malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    new_node->prev = NULL;
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;
    (*head_ref) = new_node;
}
```

Code for end deletion:-

```
struct Node
{
    int data;
    struct Node * next;
    struct Node * prev;
};

void pop()
{
    struct Node * ptr;
    if (head == NULL)
    {
        printf("\n underflow\n");
    }
    else if (head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\n Node deleted\n");
    }
    else
    {
        ptr = head;
        if (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        ptr->prev->next = NULL;
        free(ptr);
        printf("\n Node deleted\n");
    }
}
```