USN										

Internal Assessment Test 4– March 2022

Sub:	Data Structures and Applications					Sub Code:	18CS32	Branch	ISE
Date:	17/03/2022	Duration:	3 Hrs	Max Marks:	100	Sem/Sec:	III / A, B and C		OBE

Answer any FIVE FULL Questions

			MARKS	CO	RBT
		Module 1			
1	a	Define Data Structures. Give its classification. List and explain data structure operations.	10	CO1	L2
	b	Design, Develop and Implement a menu driven Program in C for the following array operations: a. Creating an array of N Integer Elements. b. Display of array Elements with Suitable Headings. c. Inserting an Element (ELEM) at a given valid Position (POS). d. Deleting an Element at a given valid Position (POS) e. Exit.	10	CO4	L3
		OR			
2	a	What is dynamic memory allocation? Explain different functions associated with dynamic memory allocation and deallocation with syntax and example. Code a C program to illustrate the same for allocating memory to store n integers and find the sum using dynamic memory allocation.	10	CO3	L2
	b	Design, Develop and Implement a Program in C for the following operations on Strings. a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP). b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR.	10	CO4	L3
		Module 2			
3	a	i. Define a Stack. List the operations on a Stack. ii. What is a Queue/Linear Queue? List different types of Queue.	10	CO3	L2
	b	Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX): a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit	10	CO4	L3
		OR			
4	a	What is recursion? Write C function: a. Tower of Hanoi. b. GCD of two numbers. c. Ackermann's Function. d. Fibonacii number	10	CO3	L2
	b	Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.	10	CO4	L3
		Module 1, 2,3			
5	a	What is a pointer? How do you declare and initialize a pointer? How do you access the value pointed to by a pointer. A(x) = $4x^{15} + 3x^4 + 5$ and B(x) = $x^4 + 10x^2 + 1$ with a diagram show how these polynomials are	10	CO3	L2

		stored in 1D array.			
	b	Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Programme, Sem, PhNo:* a. Create a SLL of N Students Data by using *front insertion*. b. Display the status of SLL and count the number of nodes in it. c. Perform Insertion / Deletion at End of SLL. d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack). e. Exit	10	CO4	L3
		OR			
6	a	Convert the infix expression ((6+(3-2)*4)^5+7) to postfix expression and evaluate that postfix expression for given data. (using stack representation).	10	CO3	L2
	b	Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes: a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$ b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z).	10	CO4	L3
		Module 2			
7	a	Convert the infix expression ((a/(b-c+d))*(e-a)*c) to postfix expression and evaluate that postfix expression for given data a=6, b=3, c=1, d=2, e=4 (using stack representation).	10	CO3	L2
	b	Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX) a. Insert an Element on to Circular QUEUE. b. Delete an Element from Circular QUEUE. c. Demonstrate Overflow and Underflow situations on Circular QUEUE. d. Display the status of Circular QUEUE. e. Exit.	10	CO4	L3
		OR			
8	a	Write a note on applications of stacks, queues, recursion and linked list.	10	CO3	L2
	b	Design, Develop and Implement a Program in C for the following Stack Applications a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^. b. Solving Tower of Hanoi problem with n disks.	10	CO4	L3
		Module 3			
9	a	What is Linked List? Explain the different types of linked lists with neat diagram.	10	CO3	L2
	b	Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: *SSN, Name, Dept, Designation, Sal, PhNo:* a. Create a DLL of N Employees Data by using *end insertion*. b. Display the status of DLL and count the number of nodes in it. c. Perform Insertion and Deletion at End of DLL. d. Perform Insertion and Deletion at Front of DLL. e. Demonstrate how this DLL can be used as Double Ended Queue. f. Exit	10	CO4	L3
		OR			
10	a	Describe the doubly linked lists with advantages and disadvantages. For the given sparse matrix,	10	CO3	L2

		give the diagrammatic linked representation. 2 0 0 0 4 0 0 3 0 0 0 0 8 0 0 1 0 0 6 0			
	b	Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Programme, Sem, PhNo* a. Create a SLL of N Students Data by using *front insertion*. b. Display the status of SLL and count the number of nodes in it. c. Perform Insertion/Deletion at End of SLL. d. Perform Insertion/Deletion at Front of SLL(Demonstration of stack). e. Exit.	10	CO4	L3

1(a). Define Data Structures. Explain the different types of data structures with examples.

Data may be organized in many different ways. The logical or mathematical model of a particular organization of data is called a data structure.

Data structures are generally classified into Primitive data Structures & Non-primitive data Structures.

1. Primitive data Structures: Primitive data structures are the fundamental data types which are supported by a programming language. Basic data types such as integer, real, character and Boolean are known as Primitive data Structures. These data types consists of characters that cannot be divided and hence they also called simple data types.

2. Non- Primitive data Structures: Non-primitive data structures are those data structures which are created using primitive data structures. Examples of non-primitive data structures is the processing of complex numbers, linked lists, stacks, trees, and graphs.

Based on the structure and arrangement of data, non-primitive data structures is further classified into Linear Data Structure & Non-linear Data Structure.

1.Linear Data Structure: A data structure is said to be linear if its elements form a sequence or a linear list. There are basically two ways of representing such linear structure in memory-One way is to have the linear relationships between the elements represented by means of

sequential memory location. These linear structures are called arrays. The other way is to have the linear relationship between the elements represented by means of pointers or links. These linear structures are called linked lists.

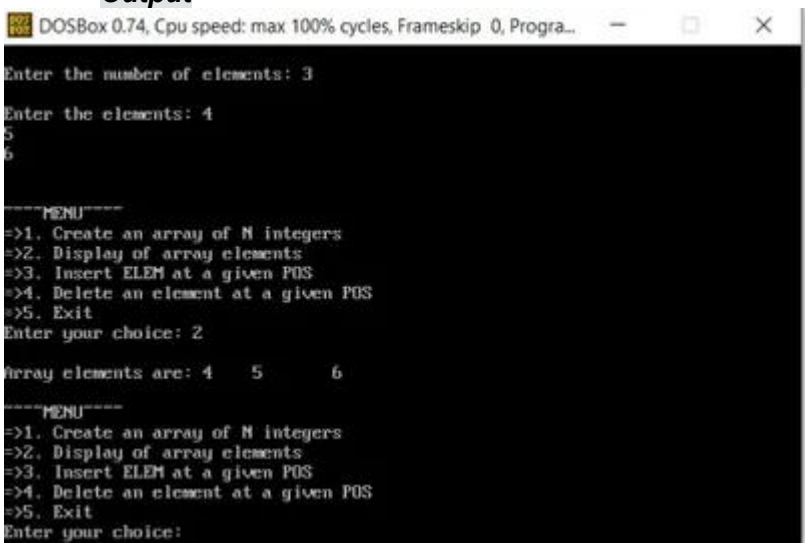
The common examples of linear data structure are Arrays, Queues, Stacks, Linked lists

2.Non-linear Data Structure: A data structure is said to be non-linear if the data are not arranged in sequence or a linear. The insertion and deletion of data is not possible in linear fashion. This structure is mainly used to represent data containing a hierarchical relationship between elements. Trees and graphs are the examples of non-linear data structure.

1(b). Design, Develop and Implement a menu driven Program in C for the following array operations: a. Creating an array of N Integer Elements. b. Display of array Elements with Suitable Headings. c. Inserting an Element (ELEM) at a given valid Position (POS). d. Deleting an Element at a given valid Position (POS) e. Exit.

```
#include<stdio.h>
#include<stdlib.h>
int a[10], pos, elem;
int n = 0;
void create();
void display();
void insert();
void del();
void main()
{
          int choice;
          while(1)
          {
           printf("\n\n~~~~MENU~~~~");
          printf("\n=>1. Create an array of N integers");
          printf("\n=>2. Display of array elements");
          printf("\n=>3. Insert ELEM at a given POS");
         printf("\n=>4. Delete an element at a given POS");
         printf("\n=>5. Exit");
         printf("\nEnter your choice: ");
         scanf("%d", &choice);
         switch(choice)
         {
           case 1: create();
                break;
          case 2: display();
                break;
         case 3: insert();
                break;
         case 4:del();
                break;
         case 5:exit(1);
                break;
       default:printf("\nPlease enter a valid choice:");
                   }
          }
}
void create()
{
```

Output



2(a). What is dynamic memory allocation? Explain different functions associated with dynamic memory allocation and deallocation with syntax and example. Code a C program to illustrate the same for allocating memory to store n integers and find the sum using dynamic memory allocation.

C Dynamic Memory Allocation

In this tutorial, you'll learn to dynamically allocate memory in your C program using standard library functions: malloc(), calloc(), free() and realloc().

As you know, an array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.

Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.

To allocate memory dynamically, library functions are `malloc()`, `calloc()`, `realloc()` and `free()` are used. These functions are defined in the `<stdlib.h>` header file.

C malloc()

The name "malloc" stands for memory allocation.

The `malloc()` function reserves a block of memory of the specified number of bytes. And, it returns a pointer of `void` which can be casted into pointers of any form.

Syntax of malloc()

```
ptr = (castType*) malloc(size);
```

Example

```
ptr = (float*) malloc(100 * sizeof(float));
```

The above statement allocates 400 bytes of memory. It's because the size of `float` is 4 bytes. And, the pointer `ptr` holds the address of the first byte in the allocated memory.

The expression results in a `NULL` pointer if the memory cannot be allocated.

C calloc()

The name "calloc" stands for contiguous allocation.

The `malloc()` function allocates memory and leaves the memory uninitialized, whereas the `calloc()` function allocates memory and initializes all bits to zero.

Syntax of calloc()

```
ptr = (castType*)calloc(n, size);
```

Example:

```
ptr = (float*) calloc(25, sizeof(float));
```

The above statement allocates contiguous space in memory for 25 elements of type `float`.

C free()

Dynamically allocated memory created with either `calloc()` or `malloc()` doesn't get freed on their own. You must explicitly use `free()` to release the space.

Syntax of free()

```
free(ptr);
```

This statement frees the space allocated in the memory pointed by `ptr`.

Example 1: malloc() and free()

```
// Program to calculate the sum of n numbers entered by the user

#include <stdio.h>#include <stdlib.h>

int main() {

  int n, i, *ptr, sum = 0;



  printf("Enter number of elements: ");

  scanf("%d", &n);



  ptr = (int*) malloc(n * sizeof(int));



  // if memory cannot be allocated

  if(ptr == NULL) {

    printf("Error! memory not allocated.");

    exit(0);
```

```
  }



  printf("Enter elements: ");

  for(i = 0; i < n; ++i) {

    scanf("%d", ptr + i);

    sum += *(ptr + i);

  }



  printf("Sum = %d", sum);



  // deallocating the memory

  free(ptr);



  return 0;

}
```

Run Code

Output

```
Enter number of elements: 3

Enter elements: 1002036

Sum = 156
```

Here, we have dynamically allocated the memory for `n` number of `int`.

Example 2: calloc() and free()

```c
// Program to calculate the sum of n numbers entered by the user

#include <stdio.h>#include <stdlib.h>

int main() {

  int n, i, *ptr, sum = 0;

  printf("Enter number of elements: ");

  scanf("%d", &n);



  ptr = (int*) calloc(n, sizeof(int));

  if(ptr == NULL) {

    printf("Error! memory not allocated.");

    exit(0);

  }



  printf("Enter elements: ");

  for(i = 0; i < n; ++i) {

    scanf("%d", ptr + i);

    sum += *(ptr + i);

  }



  printf("Sum = %d", sum);

  free(ptr);

  return 0;
```

```
}
```

Run Code

Output

```
Enter number of elements: 3

Enter elements: 1002036

Sum = 156
```

C realloc()

If the dynamically allocated memory is insufficient or more than required, you can change the size of previously allocated memory using the `realloc()` function.

Syntax of realloc()

```
ptr = realloc(ptr, x);
```

Here, `ptr` is reallocated with a new size `x`.

Example 3: realloc()

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

int main() {

  int *ptr, i , n1, n2;

  printf("Enter size: ");

  scanf("%d", &n1);
```

```
  ptr = (int*) malloc(n1 * sizeof(int));



  printf("Addresses of previously allocated memory:\n");

  for(i = 0; i < n1; ++i)

    printf("%pc\n",ptr + i);



  printf("\nEnter the new size: ");

  scanf("%d", &n2);



  // rellocating the memory

  ptr = realloc(ptr, n2 * sizeof(int));



  printf("Addresses of newly allocated memory:\n");

  for(i = 0; i < n2; ++i)

    printf("%pc\n", ptr + i);



  free(ptr);



  return 0;

}
```

Run Code

Output

```
Enter size: 2
```

```
Addresses of previously allocated memory:

26855472

26855476



Enter the new size: 4

Addresses of newly allocated memory:

26855472

26855476

26855480

26855484
```

2b. Design, Develop and Implement a Program in C for the following operations on Strings.
a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP).
b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR.

```
#include<stdio.h>
#include<conio.h>
char str[50], pat[20], rep[20], ans[50];
int     c=0, m=0, i=0, j=0, k, flag=0;
void stringmatch()
{
               while(str[c] !='\0')
              {
                             if(str[m] == pat[i])
                             {
                                           i++;
                                           m++;
                                           if(pat[i] == '\0')
                                           {
                                                         flag = 1;
                                                         for(k=0; rep[k]!='\0'; k++, j++)
                                                         {
                                                                       ans[j] = rep[k];
                                                         }
                                                         i = 0;
                                                         c = m;
                                           }
                             }
                             else
                             {
                                           ans[j]= str[c];
                                           j++;
                                           c++;
                                           m=c;
                                           i=0;
                             }
              }
              ans[j]='\0';
}
void main()
{
              printf("\nEnter the main string:");
              gets(str);
              printf("\nEnter the pat string:");
```

Output

```
Enter the main string:hey atul ...atul is a boy

Enter the pat string:atul

Enter the replace string:girl

Resultant string is hey girl ...girl is a boy
Enter the main string:one two three four

Enter the pat string:one

Enter the replace string:1

Resultant string is 1 two three four
Enter the main string:my name is mikki and i am chikki ka dost

Enter the pat string:ikki

Enter the replace string:meetha

Resultant string is my name is mmeetha and i am chmeetha ka dost
```

3. I) Define a Stack. List the operations on a Stack.
II) What is a Queue/Linear Queue? List different types of Queue.

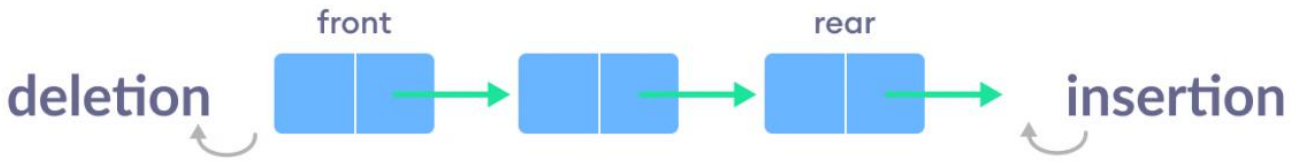
A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

There are four different types of queues:

- Simple Queue
- Circular Queue
- Priority Queue
- Double Ended Queue

Simple Queue

In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.

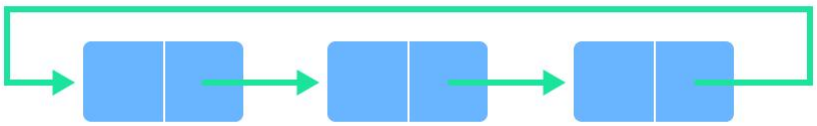


Simple Queue Representation

To learn more, visit Queue Data Structure.

Circular Queue

In a circular queue, the last element points to the first element making a circular link.



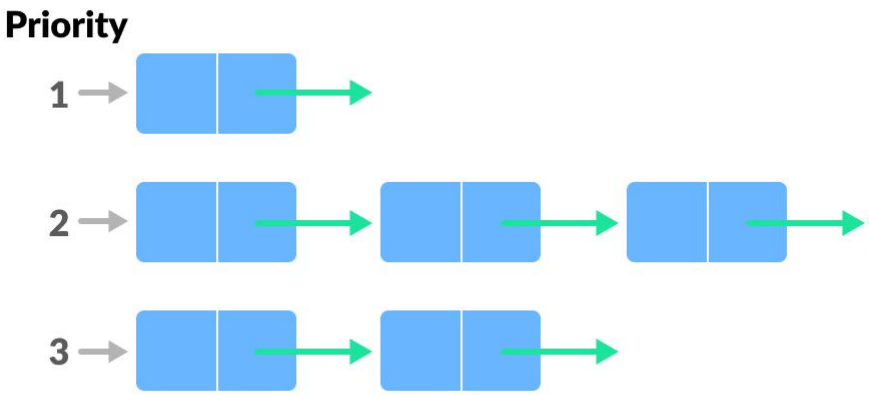
Circular Queue Representation

The main advantage of a circular queue over a simple queue is better memory utilization. If the last position is full and the first position is empty, we can insert an element in the first position. This action is not possible in a simple queue.

To learn more, visit Circular Queue Data Structure.

Priority Queue

A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in the queue.



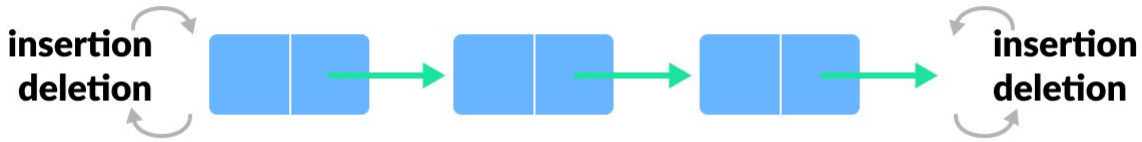
Priority Queue Representation

Insertion occurs based on the arrival of the values and removal occurs based on priority.

To learn more, visit Priority Queue Data Structure.

Deque (Double Ended Queue)

In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow the FIFO (First In First Out) rule.

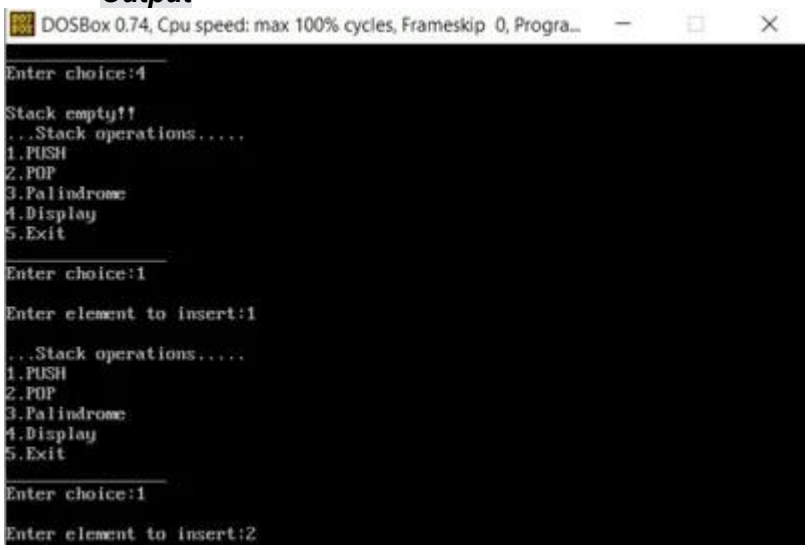


Deque Representation

3b. Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX):
a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit.

```
#include <stdio.h>
#include <stdlib.h>
#include<conio.h>
int s[5],top=-1;
void push()
{
   if(top==4)
      printf("\nStack overflow!!!!");
   else
   {
      printf("\nEnter element to insert:");
      scanf("%d",&s[++top]);
   }
}
void pop()
{
   if(top==-1)
      printf("\nStack underflow!!!");
   else
      printf("\nElement popped is: %d",s[top--]);
}
void disp()
{
   int t=top;
   if(t==-1)
      printf("\nStack empty!!");
   else
      printf("\nStack elements are:\n");
   while(t>=0)
      printf("%d ",s[t--]);
}
void pali()
{
   int num[5],rev[5],i,t;
   for(i=0,t=top;t>=0;i++,t--)
      num[i]=rev[t]=s[t];
   for(i=0;i<=top;i++)
      if(num[i]!=rev[i])
      break;
```

Output



4. Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

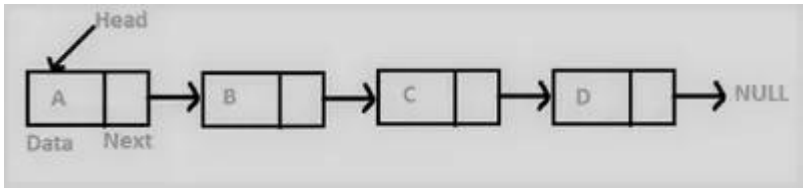
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <string.h>
char infix_string[20], postfix_string[20];
int top; int stack[20]; int pop();
int precedence(char symbol);
int isEmpty();
void infix_to_postfix();
int check_space(char symbol);
void push(long int symbol);
int main()
{
   int count, length;
   char temp;
   top = -1;
   printf("\nINPUT THE INFIX EXPRESSION : ");
   scanf("%s", infix_string);
   infix_to_postfix();
   printf("\nEQUIVALENT POSTFIX EXPRESSION : %s\n", postfix_string);
   return 0;
}
void infix_to_postfix()
{
   unsigned int count, temp = 0;
   char next;
   char symbol;
   for(count = 0; count < strlen(infix_string); count++)
   {
      symbol = infix_string[count];   // Scanning the input expression
      if(!check_space(symbol))
      {
         switch(symbol)
         {
            case '(': push(symbol);
               break;
            case ')':
               while((next = pop()) != '(')     // pop until '(' is encountered
               {
```

Output

```
INPUT THE INFIX EXPRESSION : (3^2*5)/(3*2-3)+5

EQUIVALENT POSTFIX EXPRESSION : 32^5*32*3-/5+
```

5. Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Programme, Sem, PhNo:*
a. Create a SLL of N Students Data by using *front insertion*. b. Display the status of SLL and count the number of nodes in it. c. Perform Insertion / Deletion at End of SLL. d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack). e. Exit.

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.



- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.

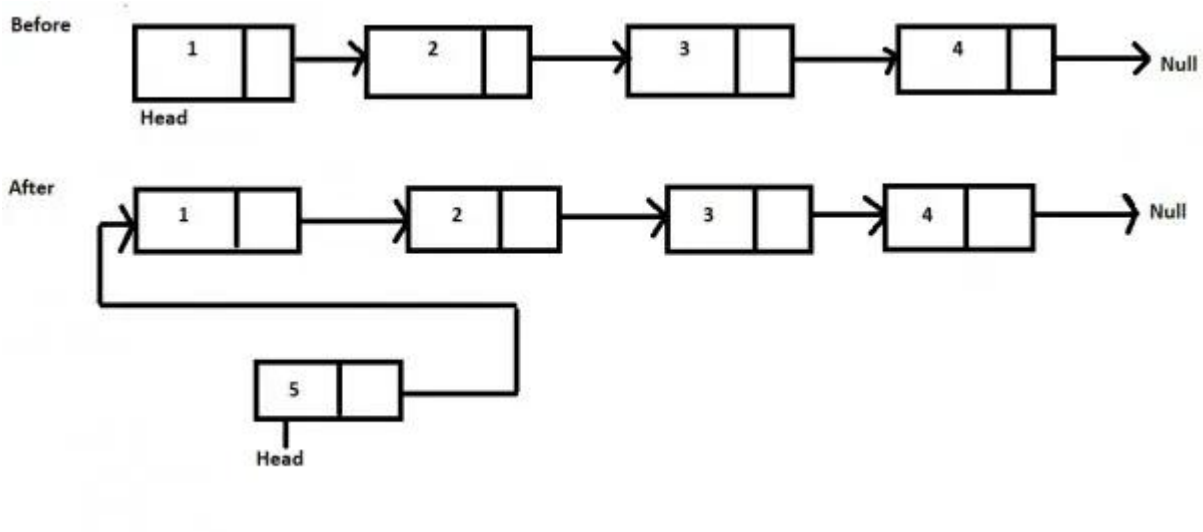
The simplest kind of linked list is a singly liked list (SLL) which has one link per node. It has two parts, one part contains data and other contains address of next node. *In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.*The structure of a node in a SLL is given as in C:

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

Insertion of node at front

1. allocate node
2. put in the data
3. Make next of new node as head
4. move the head to point to the new node

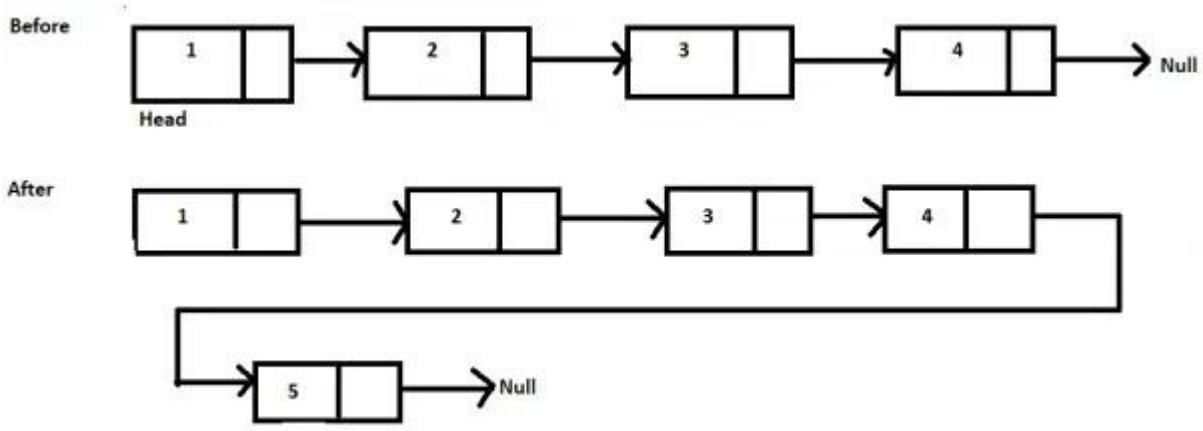
```
void insert(struct Node** head_ref, int new_data)
{
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    new_node->data  = new_data;
    new_node->next = (*head_ref);
    (*head_ref)    = new_node;
}
```



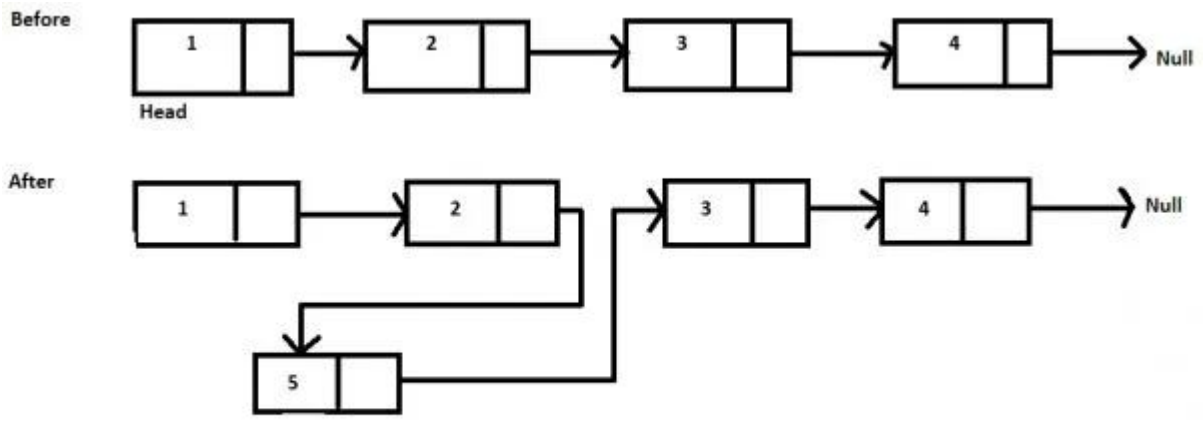
Insertion at end

1. allocate node
2. put in the data
3. This new node is going to be the last node, so make next of it as NULL
4. If the Linked List is empty, then make the new node as head
5. Else traverse till the last node
6. Change the next of last node

```
void end(struct Node** head_ref, int new_data)
{
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    struct Node *last = *head_ref;
    new_node->data  = new_data;
    new_node->next = NULL;
    if (*head_ref == NULL)
    {
      *head_ref = new_node;
      return;
    }
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
    return;
}
```



Insertion after a node



6.Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes:
a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$
b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z).

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define COMPARE(x, y)       ( (x == y) ? 0 : (x > y) ? 1 : -1)

struct node
{
        int coef;
        int xexp, yexp, zexp;
        struct node *link;
};
typedef struct node *NODE;

NODE getnode()
{
        NODE x;
        x = (NODE) malloc(sizeof(struct node));
        if(x == NULL)
        {
                printf("Running out of memory \n");
                return NULL;
        }
        return x;
}

NODE attach(int coef, int xexp, int yexp, int zexp, NODE head)
{
        NODE temp, cur;
        temp = getnode();
```

```
        temp->coef = coef;
        temp->xexp = xexp;
        temp->yexp = yexp;
        temp->zexp = zexp;
        cur = head->link;
        while(cur->link != head)
        {
                cur = cur->link;
        }
        cur->link = temp;
        temp->link = head;
        return head;
}

NODE read_poly(NODE head)
{
        int i, j, coef, xexp, yexp, zexp, n;
        printf("\nEnter the no of terms in the polynomial: ");
        scanf("%d", &n);
        for(i=1; i<=n; i++)
        {
                printf("\n\tEnter the %d term: ",i);
                printf("\n\t\tCoef =  ");
                scanf("%d", &coef);
                printf("\n\t\tEnter Pow(x) Pow(y) and Pow(z): ");
                scanf("%d", &xexp);
                scanf("%d", &yexp);
                scanf("%d", &zexp);
                head = attach(coef, xexp, yexp, zexp, head);
        }
         return head;
}

void display(NODE head)
{
        NODE temp;
        if(head->link == head)
        {
                printf("\nPolynomial does not exist.");
                return;
        }
        temp = head->link;

        while(temp != head)
        {
                printf("%dx^%dy^%dz^%d", temp->coef, temp->xexp, temp->yexp, temp-
>zexp);
                temp = temp->link;
                if(temp != head)
                        printf(" + ");
        }
```

```
}
int poly_evaluate(NODE head)
{
        int x, y, z, sum = 0;
        NODE poly;

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

        poly = head->link;
        while(poly != head)
        {
                sum += poly->coef * pow(x,poly->xexp)* pow(y,poly->yexp) * pow(z,poly-
>zexp);
                poly = poly->link;
        }
        return sum;
}
NODE poly_sum(NODE head1, NODE head2, NODE head3)
{
   NODE a, b;
   int coef;
   a = head1->link;
   b = head2->link;

   while(a!=head1 && b!=head2)
   {
        while(1)
        {
                if(a->xexp == b->xexp && a->yexp == b->yexp && a->zexp == b->zexp)
                {
                        coef = a->coef + b->coef;
                        head3 = attach(coef, a->xexp, a->yexp, a->zexp, head3);
                        a = a->link;
                        b = b->link;
                        break;
                } //if ends here
                if(a->xexp!=0 || b->xexp!=0)
                {
                        switch(COMPARE(a->xexp, b->xexp))
                        {
                         case -1 :  head3 = attach(b->coef, b->xexp, b->yexp, b->zexp, head3);
                                    b = b->link;
                                    break;

                        case 0 :    if(a->yexp > b->yexp)
                                    {
                                          head3 = attach(a->coef, a->xexp, a->yexp, a->zexp,
head3);
                                          a = a->link;
```

```
                                        break;
                                    }
                                    else if(a->yexp < b->yexp)
                                    {
                                        head3 = attach(b->coef, b->xexp, b->yexp, b->zexp,
head3);
                                        b = b->link;
                                        break;
                                    }
                                    else if(a->zexp > b->zexp)
                                    {
                                        head3 = attach(a->coef, a->xexp, a->yexp, a->zexp,
head3);
                                        a = a->link;
                                        break;
                                    }
                                    else if(a->zexp < b->zexp)
                                    {
                                        head3 = attach(b->coef, b->xexp, b->yexp, b->zexp,
head3);
                                        b = b->link;
                                        break;
                                    }
                            case 1 :    head3 = attach(a->coef,a->xexp,a->yexp,a->zexp,head3);
                                        a = a->link;
                                        break;
                            } //switch ends here
                            break;
                    } //if ends here
                    if(a->yexp!=0 || b->yexp!=0)
                    {
                            switch(COMPARE(a->yexp, b->yexp))
                            {
                                case -1 :   head3 = attach(b->coef, b->xexp, b->yexp, b->zexp,
head3);
                                            b = b->link;
                                            break;
                                 case 0 :   if(a->zexp > b->zexp)
                                            {
                                                head3 = attach(a->coef, a->xexp, a->yexp, a->zexp,
head3);
                                                a = a->link;
                                                break;
                                            }
                                            else if(a->zexp < b->zexp)
                                            {
                                                head3 = attach(b->coef, b->xexp, b->yexp, b->zexp,
head3);
                                                b = b->link;
                                                break;
```

```
                                        }
                                case 1 :    head3 = attach(a->coef, a->xexp, a->yexp, a->zexp,
head3);
                                        a = a->link;
                                        break;
                        }
                        break;
                }
                if(a->zexp!=0 || b->zexp!=0)
                {
                        switch(COMPARE(a->zexp,b->zexp))
                        {
                                case -1 :   head3 = attach(b->coef,b->xexp,b->yexp,b-
>zexp,head3);
                                            b = b->link;
                                            break;
                                case 1 :    head3 = attach(a->coef, a->xexp, a->yexp, a->zexp,
head3);
                                            a = a->link;
                                            break;
                        }
                        break;
                }
        }
   }
   while(a!= head1)
   {
         head3 = attach(a->coef,a->xexp,a->yexp,a->zexp,head3);
        a = a->link;
   }
   while(b!= head2)
   {
        head3 = attach(b->coef,b->xexp,b->yexp,b->zexp,head3);
        b = b->link;
   }
   return head3;
}




void main()
{
 NODE head, head1, head2, head3;
 int res, ch;
 head =  getnode();    /* For polynomial evalaution */
 head1 = getnode();    /* To hold POLY1 */
 head2 = getnode();    /* To hold POLY2 */
 head3 = getnode();    /* To hold POLYSUM */

 head->link=head;
```

```
head1->link=head1;
head2->link=head2;
head3->link= head3;

while(1)
{
    printf("\n~~~Menu~~~");
    printf("\n1.Represent and Evaluate a Polynomial P(x,y,z)");
    printf("\n2.Find the sum of two polynomials POLY1(x,y,z)");
    printf("\nEnter your choice:");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1:       printf("\n~~~~Polynomial evaluation P(x,y,z)~~~\n");
                    head = read_poly(head);
                    printf("\nRepresentation of Polynomial for evaluation: \n");
                    display(head);
                    res = poly_evaluate(head);
                    printf("\nResult of polynomial evaluation is : %d \n", res);
                    break;

      case 2:       printf("\nEnter the POLY1(x,y,z):  \n");
                    head1 = read_poly(head1);
                    printf("\nPolynomial 1 is:  \n");
                    display(head1);

                    printf("\nEnter the POLY2(x,y,z):  \n");
                    head2 = read_poly(head2);
                    printf("\nPolynomial 2 is: \n");
                    display(head2);

                    printf("\nPolynomial addition result: \n");
                    head3 = poly_sum(head1,head2,head3);
                    display(head3);
                    break;
        case 3:     exit(0);
    }
  }
}
```

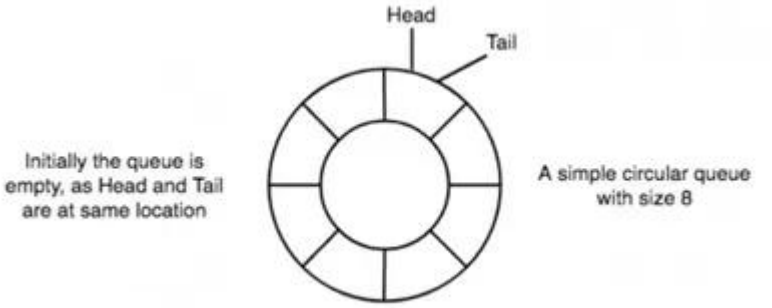
7. Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)
a. Insert an Element on to Circular QUEUE. b. Delete an Element from Circular QUEUE. c. Demonstrate Overflow and Underflow situations on Circular QUEUE. d. Display the status of Circular QUEUE. e. Exit.

Circular Queue is also a linear data structure, which follows the principle of **FIFO(First In First Out)** and the last position is connected back to the first position to make a circle. It is also called 'Ring Buffer'., but instead of ending the queue at the last position, it again starts from the first position after the last, hence making the queue behave like a circular data structure.

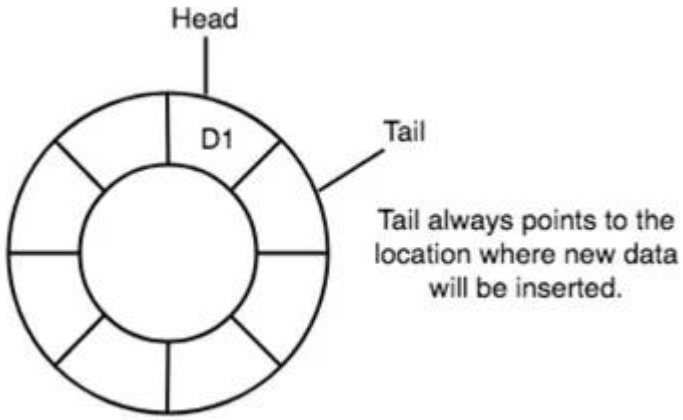
Deletions and insertions can only be performed at front and rear end respectively, as far as linear queue is concerned.

Some Important points to Remember :

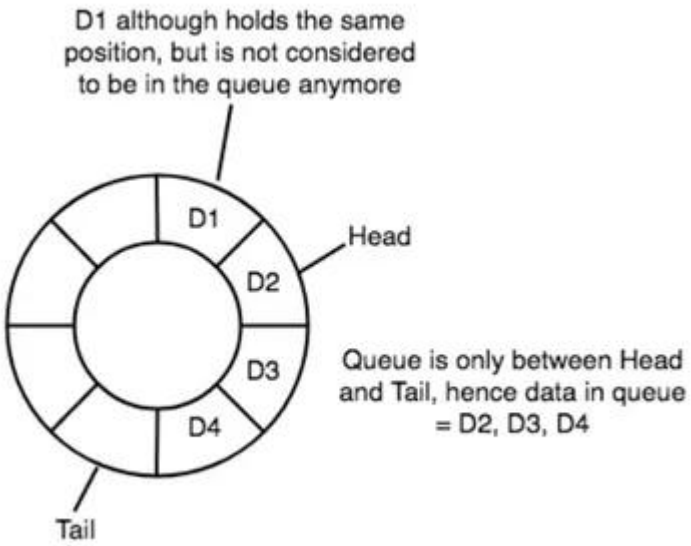
- In case of a circular queue, head pointer will always point to the front of the queue, and tail pointer will always point to the end of the queue.
- Initially, the head and the tail pointers will be pointing to the same location, this would mean that the queue is empty.



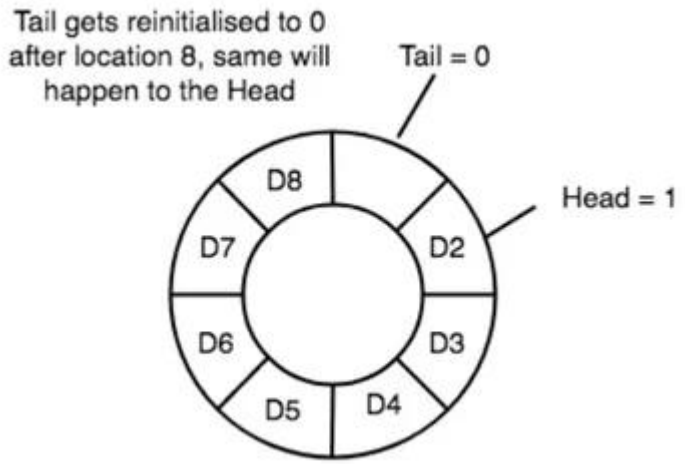
- New data is always added to the location pointed by the tail pointer, and once the data is added, tail pointer is incremented to point to the next available location.



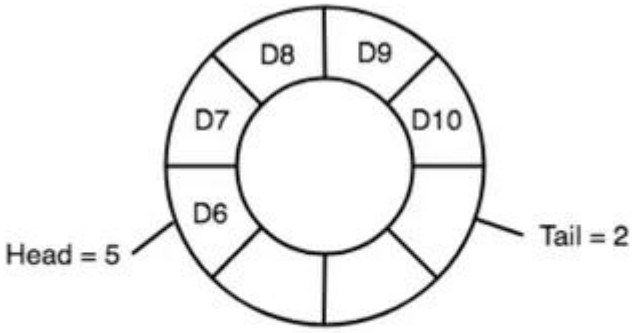
- In a circular queue, data is not actually removed from the queue. Only the head pointer is incremented by one position when dequeue is executed. As the queue data is only the data between head and tail, hence the data left outside is not a part of the queue anymore, hence removed.



- The head and the tail pointer will get re-initialised to 0 every time they reach the end of the queue.



- Also, the head and the tail pointers can cross each other. In other words, head pointer can be greater than the tail. Sounds odd? This will happen when we dequeue the queue a couple of times and the tail pointer gets re-initialised upon reaching the end of the queue.



In such a situation the value of the Head pointer will be greater than the Tail pointer

8. Design, Develop and Implement a Program in C for the following Stack Applications
a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^. b. Solving Tower of Hanoi problem with n disks.

```
#include<stdio.h>#include"stdafx.h"#include<conio.h>#include<math.h>#include<string.h>#define MAX 50int stack[MAX];char post[MAX];int top = -1;void pushstack(int tmp);void calculator(char c);void main(){int i;printf("Insert a postfix notation :: ");//"23+"gets(post);//post=5051+//scanf_s("%s", post);for (i = 0; i<strlen(post); i++){if (post[i] >= '0' && post[i] <= '9'){pushstack(i);//pushstack(0)}
```

9. Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: *SSN, Name, Dept, Designation, Sal, PhNo:*
a. Create a DLL of N Employees Data by using *end insertion*. b. Display the status of DLL and count the number of nodes in it. c. Perform Insertion and Deletion at End of DLL. d. Perform Insertion and Deletion at Front of DLL. e. Demonstrate how this DLL can be used as Double Ended Queue. f. Exit.

```
#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

struct node

{

        char ssn[25],name[25],dept[10],designation[25];

        int sal;

        long int phone;

        struct node *llink;

        struct node *rlink;

};

typedef struct node* NODE;

NODE first = NULL;

int count=0;

NODE create()

{

         NODE enode;

         enode = (NODE)malloc(sizeof(struct node));

         if( enode== NULL)

         {

                 printf("\nRunning out of memory");

                 exit(0);

          }

          printf("\nEnter the ssn,Name,Department,Designation,Salary,PhoneNo of the
employee: \n");

        scanf("%s %s %s %s %d %ld", enode->ssn, enode->name, enode->dept, enode-
>designation, &enode->sal, &enode->phone);

          enode->llink=NULL;

          enode->rlink=NULL;

          count++;

          return enode;

}

NODE insertfront()

{

        NODE temp;

        temp = create();

         if(first == NULL)

        {

              return temp;

         }
```

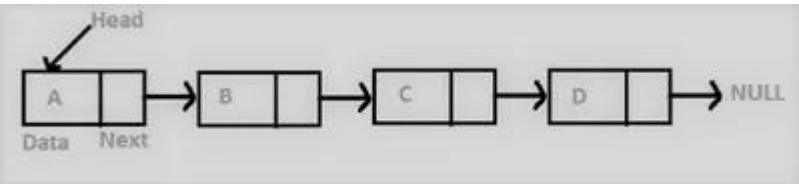
Output

```
~~~Menu~~~
1:Create DLL of Employee Nodes
2:DisplayStatus
3:InsertAtEnd
4:DeleteAtEnd
5:InsertAtFront
6:DeleteAtFront
7:Double Ended Queue Demo using DLL
8:Exit

Please enter your choice: 1

Enter the no of Employees:   3

Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
1
abc
CSE
PROF
1000000
9809876098

Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
2
BCD
PROF
12345
8790987654
9829812918291
```

```
Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
2
22w
34dccad
cvbn
0987509
3456789021


~~~Menu~~~
1:Create DLL of Employee Nodes
2:DisplayStatus
3:InsertAtEnd
4:DeleteAtEnd
5:InsertAtFront
6:DeleteAtFront
7:Double Ended Queue Demo using DLL
8:Exit

Please enter your choice: 2

ENode:1||SSN:1|Name:abc|Department:CSE|Designation:PROF|Salary:1000000|Phone no:9809876098
ENode:2||SSN:2|Name:BCD|Department:PROF|Designation:12345|Salary:201053062|Phone no:9829812918291
ENode:3||SSN:2|Name:22w|Department:34dccad|Designation:cvbn|Salary:987509|Phone no:3456789021
No of employee nodes is 3
```

10. Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Programme, Sem, PhNo*
a. Create a SLL of N Students Data by using *front insertion*. b. Display the status of SLL and count the number of nodes in it. c. Perform Insertion/Deletion at End of SLL. d. Perform Insertion/Deletion at Front of SLL(Demonstration of stack). e. Exit.

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.



- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.

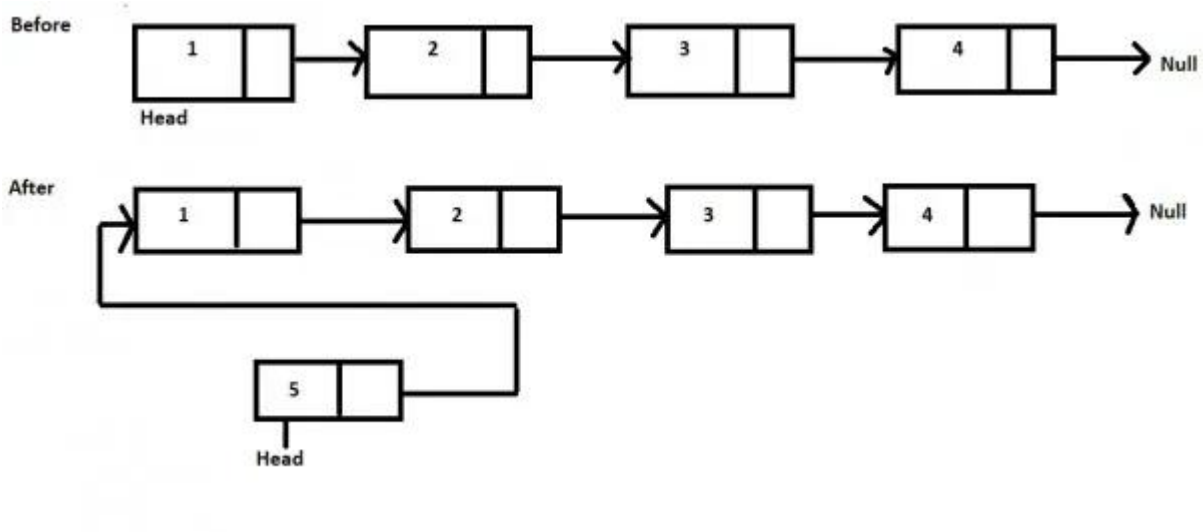
The simplest kind of linked list is a singly liked list (SLL) which has one link per node. It has two parts, one part contains data and other contains address of next node. *In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.*The structure of a node in a SLL is given as in C:

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

Insertion of node at front

1. allocate node
2. put in the data
3. Make next of new node as head
4. move the head to point to the new node

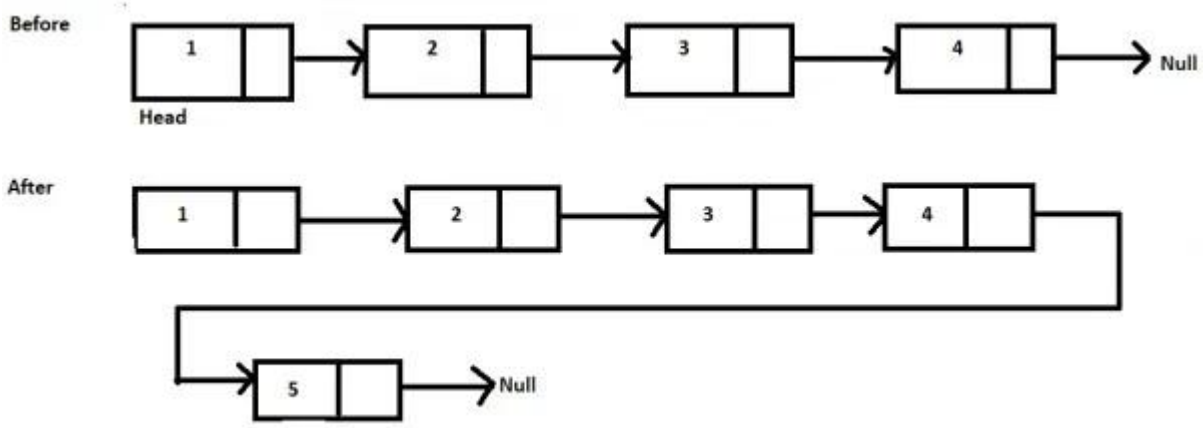
```
void insert(struct Node** head_ref, int new_data)
{
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data  = new_data;
   new_node->next = (*head_ref);
   (*head_ref)    = new_node;
}
```



Insertion at end

1. allocate node
2. put in the data
3. This new node is going to be the last node, so make next of it as NULL
4. If the Linked List is empty, then make the new node as head
5. Else traverse till the last node
6. Change the next of last node

```
void end(struct Node** head_ref, int new_data)
{
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   struct Node *last = *head_ref;
   new_node->data  = new_data;
   new_node->next = NULL;
   if (*head_ref == NULL)
   {
     *head_ref = new_node;
     return;
   }
   while (last->next != NULL)
      last = last->next;
   last->next = new_node;
   return;
}
```



Insertion after a node