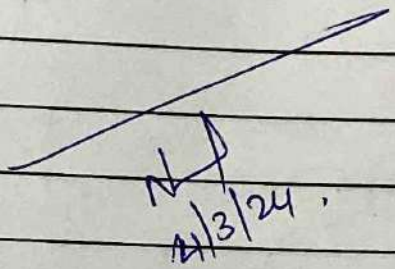
Index

Lab No	Date	Program name	Submission
1	07/12/2023	Practice Programs	07/12/2023
2	21/12/2023	swapping two variables Dynamic memory Allocation stack Implementation	21/12/2023
3	28/12/2023	Infix to postfix conversion Postfix Evaluation Queue implementation circular queue implementation	18/01/2024
4	11/01/2024	Implementation of linked list Leetcode - min stack	11/01/2024
5	18/01/2024	Delection in linked list	18/01/2024
6	25/01/2024	Linked list operations stack & Queue in Linked list	1/2/2024
7	01/02/2024	Doubly Linked List	1/02/2024
8	15/02/2024	Binary search tree Linked list - Rotate LSE	15/02/2024

store™
67

9	22/02/2024	BFS	22/02/2024
		DFS	
		Hacker Rank	
10	29/02/2024	Hashing - Linear Probing	29/02/2024



4/3/24.

store™
67

21 12 2023

PROGRAM 1 [Swapping of two numbers using pointers]

```
#include <stdio.h>
void swap (int *x, int *y)
{
  int t;
  t = *x;
  *x = *y;
  *y = t;
}
int main ()
{
  int num1, num2;
  printf ("Roshni P IBM22CS223\n");
  printf ("Enter value of num1: ");
  scanf ("%d", &num1);
  printf ("Enter value of num2: ");
  scanf ("%d", &num2);
  printf ("Before swapping : num1 is : %d, num2 is : %d
          \n", num1, num2);
  swap (&num1, &num2);
  printf ("After swapping : num1 is : %d, num2 is : %d\n"
          , num1, num2);
  return 0;
}
```

store™ 67

Output :-

Enter value of num1 : 3

Enter value of num2 : 2

Before swapping : num1 is : 3, num2 is : 2

After swapping : num1 is : 2, num2 is : 3

PROGRAM 2 [Dynamic Memory allocation (malloc, free : calloc, realloc)]

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int *ptr;
  int n, i;
  n = 5;
  printf("Enter number of elements : %d \n", n);
  ptr = (int*) malloc (n* sizeof(int));
  if (ptr == NULL) {
  printf("Memory not allocated. \n");
  exit(0);
  }
  else {
  printf("Memory successfully allocated using malloc. \n");
  for (i=0; i<n; ++i) {
   ptr[i] = i+1;
  }
```

store™
67

```
printf("Memory The elements of the array are :");
for (i=0; i<n; ++i) {
printf("Mer %d, ", ptr[i]);
}
ptr = (int*) calloc (n, sizeof(int));
if (ptr == NULL) {
printf("Memory not allocated.\n");
exit(0);
}
else {
printf("Memory successfully allocated using
        calloc.\n");
for (i=0; i<n; ++i) {
ptr[i] = i+1;
}
printf ("The elements of the array are :");
for (i=0; i<n; ++i) {
printf ("%d:", ptr[i]);
}
n = 10;
printf ("\n\n Enter the new size of the array : %d\n
        , n);
ptr=(int*) realloc (ptr, n*sizeof(int));
printf("Memory successfully re-allocated
        using realloc.\n");
```

store™
67

```
for (i=5; i<n; ++i) {
  ptr[i] = i+1;
}
printf("%d, ", ptr[i]);
}
free(ptr);
}
return 0;
}
```

Output :-

Enter number of elements : 5
Memory successfully allocated using malloc.
The elements of the array are : 1, 2, 3, 4, 5,
Memory successfully allocated using calloc.
The elements of the array are : 1, 2, 3, 4, 5,

Enter the new size of the array : 10
Memory successfully re-allocated using realloc.
The elements of the array are : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

store™ 67

```
#include <stdio.h>
#include <conio.h>
#define N 10
void push();
void push();
void peek();
void display();
int stack[N];
int top = -1;
void main()
{
  int ch;
  do {
      printf("enter a choice 1: push\t 2: pop\t
              3: display\n");
      printf("enter your choice.\n");
       scanf("%d", &ch);
    switch(ch){
     case 1: push();
      break;
     case 2: pop();
      break;
     case 3: display();
      break;
      default: printf("not a valid number\n");
    }
```

store™
67

```
}
while(ch!=0);
getch();
}
void push()
{
  int x;
  printf("enter a value \n");
  scanf("%d",&x);
  if(top==N)
{
  printf("overflow");
}
else{
      top++;
   stack[top]=x;
}
}
void pop()
{
  int y;
  if(top==-1){
   printf("%d.",y);
   }
}
```

store™
67

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

Output :

```
enter a choice 1: push  2: pop  3: display
enter your choice
1
enter a c a value
4
enter a choice 1: push 2: pop 3: display
enter your choice
2
4
enter a choic 1: push  2: pop  3: display
enter your choice
3
enter a choice 1: push 2: pop 3: display
enter your choice
```

store™
67

LAB-3

28/12/23

1]
```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define size 20
char stack [size];
int top = -1;
void push (char a) {
stack [++top] = a;
}
char pop () {
return stack [top--];
}
int precedence (char a) {
if (a == '^')
return 3;
else if (a == '*' || a == '/')
return 2;
else if (a == '+' || a == '-')
return 1;
else
return 0;
}
```

NP 18/1/24

store™ 67

```
void main() {
char infix[size] ; postfix[size];
int j=0;
for (int i=0; i< strlen(infix); i++) {
    if (infix[i] == '(')
    push ('(');

else {
    if (precedence[stack[top]) < precedence
        (infix[i])
    push (infix[i]);
    else {
        while (stack[top] != '(') {
            postfix[j++] = pop();
        }
        push (infix[i]);
    }
}
}
}
    while (top != -1) {
        postfix[j++] = pop();
    }
    postfix[j] = '\0';
    printf ("Postfix expression is %s",
                postfix);
```

store™
67

Output

enter Expression : A*B+C*D-E

Postfix Expression is AB* CD*-E-

2] #include <stdio.h>
#include <string.h>
#include <ctype.h>
#define max 20

int stack [20];
int top = -1;
void push (int a) {
stack [++top] = a;
}
int pop () {
return stack [top--];
}

void main () {

char postfix [max] = "12*34*+5-";
int result = 0, a, b;
for (int i=0; i < strlen (postfix); i++) {
if (isalnum (postfix [i]))
push (postfix [i] - '0');

store™
67

```
else
b=pop();
a=pop();
switch(postfix[i])
{
case '+';
    push(a+b);
    break;
case '-':
    push(a-b);
    break;
case '*':
    push(a*b);
    break;
case '/':
    push(a/b);
    break;
case '^':
    push(a^b);
    break;
}
}
result=pop();
printf("%s=%d", postfix, result);
}
```

store™
67

Output:

Enter postfix expression : 12*34*+5-

12*34*+5- : 9

3] #include <stdio.h>
#define size 30
int queue [size];
int front = -1, rear = -1;

void insert (int a) {
if (rear = size-1) {
printf (" Queue overflow \n ");
return;
}
else {
if (front = = -1)
front = 0;
queue [++rear] = a
}

void delete () {
if (front = = -1 || front > rear) {
printf (" Queue Empty \n");
else

store™
67

```
    front++;
}
void display(){
    if(front==-1)
        printf("Queue empty\n");
        return;
    printf("Queue");
for(int i=post; i<=rear; i++){
    printf("%d", queue[i]);
}
    printf("\n....\n");
}
void main(){
    int choice, a;
while(1){
    printf("Queue Operations\n Insert\n
            Delete\n Display");
scanf("%d", &choice);

switch(choice)
{
    case 1;
        scanf("%d", &a);
        insert(a);
```

store™ 67

```
        display ();
        break;
    case 2:
        delete ();
        display ();
        break;
    case 3:
        display ();
        break;
}
}
```

18/1/24

Output :

Queue Operations

1 Insert
2 Delete
3 Display

choice : 1
Enter Element : 3
Queue : 3

store™
67

choice : 1

Enter element : 4

Queue : 3 4

choice : 2

Queue : 4

choice : 3

Queue : 4

store™
67

11 01 2024

LAB4

1] circular. queue

```
#include <stdio.h>
#include <stdlib.h>
#define size 5

int q[size], f=0, r=-1;
int count = 0;

void enqueue (int item) {
  if (count == size) {
  printf ("\n Queue full!");
  return;
}
  q[(++r) % size] = item;
  count++;
}
void dequeue () {
  if (count == 0) {
  printf ("\n Queue empty!");
  return;
}
  f = (f+1) % size;
  count--;
}
```

store™
67

```
void display (){

if (count==0){
    printf ("\n Queue empty !");
    return;
}
  int front = f;
  for (int i=0; i<count; i++){
    printf ("%d ", q[front]);
    front = (front+1) % size;
  }
}
 int main (){
  int ch, item;
  while (1){
   printf ("\n select choice \n 1. Enqueue \n
            2. Dequeue \n 3 Display \n Choice: ");
   scanf ("%d", &ch);
  switch (ch){
   case 1:
     printf ("\n Enter value to insert: ");
     scanf ("%d", &item);
     enqueue (item);
     break;
```

store™ 67

```
case 2:
    dequeue();
    printf("\n Item popped");
    break;
case 3:
    display()
    break;
    default;
    exit(0);
    {
  }
}
```

Output :

```
select choice:
1] Enqueue
2] Dequeue
3] Display

choice : 1
enter value: 3
Queue = 3
```

choice: 1

enter value = 4

Queue = 34

choice: 2

Queue = 4

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

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

void insertAtHead (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;
  }
```

store™ 67

```
void insertAtMiddle (struct node *prev_node, int new_data)
{
  if (prev_node == NULL) {
    printf("The given previous node cannot be NULL");
    return;
  }
  struct node *new_node = (struct node*) malloc
                          (sizeof (struct node));
  new_node -> data = new_data;
  new_node -> next = prev_node -> next;
  prev_node -> next = new_node;
}
void insertAtEnd (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);
  if (*head_ref == NULL) { new_node;
    return;
  }
  while (last -> next != NULL) {
    last = last -> next;
```

store™
67

```
    }
    last->next = new_node;
    return;
}
void display(struct node *node){
printf("Linked list:");
while(node != NULL){
printf("%d", node->data);
 node = node->next;
}
 printf("\n");
}

int main(){
   struct node *head = NULL;
    int choice = 0, a;
    while(choice != 3){
    printf("Select an option\n 1. Insert\n
                2. Display\n 3. Exit\n choice:");
    scanf("%d", &choice);

    switch(choice)
{
    case 1:
       printf("Enter the value to inserted:");
```

store™ 67

```
scanf ("%d", &a);
int choice2;
printf (" ------ \n 1. Insert at Head \n
        2. Insert at end\n 3. Insert in
        the middle \n choice :" );
scanf ("%d", &choice2);
switch (choice2)
{
 case 1:
   insertAtHead (&head, a);
   display (head);
   break;
   case 2:
   insertAtMiddle (&head, a);
   display(head);
    break;
   case 3:
    insertAtEnd(&head, a);
    display (head);
    break;
break;
case 2:
    display (head);
    break;
default:
  break;
}
}
return 0;
}
```

store™
67

Output:

Select an option
1 Insert
2 Display:
3 Exit

choice: 1
1 At start
2 At Middle
3 At end

choice : 1
Enter value : 5
Linked List : 5

choice : 1
1. At start
2 At Middle
3 At end

11/1/24

choice : 3
enter value : 8
Linked list : 5 8

store™
67

choice : 2

Linked List 5 8

store™
67

18 01 2024

LAB-5

1]

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

void pop();
void end_delete();
void delete_at_pos();
void display();
void append();
struct node
{
  int data;
  struct node *next;
};
struct node *head = NULL;
void main()
{
 printf("Insert the element in list \n");
 append();
 printf("1. Delete from beginning\n 2. Delete at end\n
        3. Delete at end \n 3. Delete at particular
           position \n 4. Display \n 5. Exit \n");
 int ch;
 while (ch != 5)
 {
```

store™ 67

```
{
    printf ("Enter choice : ");
    scanf ("%d", &ch);
switch (ch)
{
  case 1:
        pop ();
        break;
  case 2:
        end_delete ();
        break;
  case 3:
        delete_at_pos ();
        break;
  case 4:
        display ();
        break;
  default: printf ("Invalid choice");
        break;
  }
 }
}
void append ()
{
  int data, n;
```

```
printf ("Enter no. of nodes : ");
scanf ("%d", &n);
for (int i= 0 ; i<n; i++)
  {
   struct node * last = head;
   struct node * new_node;
   new_node = (struct node *) malloc (sizeof (struct node));
  printf ("Enter the data :");
   scanf ("%d", &data);
   new_node -> data = data;
   new_node -> next = NULL;
   if (head == NULL)
        head = new_node;
   else
   {
    while ( last -> next != NULL)
     {
      last = last -> next;
    }
    last -> next = new_node;
    }
  }
}
void Pop ()
{
```

store™
67

```
struct node * ptr;
if (head == NULL)
    printf (" List is empty \n");
else
{
    ptr = head;
    head = ptr -> next;
    free (ptr);
    printf (" Node deleted from beginning \n");
}
}
Void end_delete ()
{
  struct node *ptr;
  struct node * ptr1;
  if (head == NULL)
      printf ("List is empty \n");
  else if (head -> next == NULL)
  {
    free (head);
    head = Null;
  }
  else
  {
    ptr = head;
    ptr 1 = head;
```

store™
67

```
while ( ptr -> next != NULL )
{
    ptr1 = ptr;
    ptr = ptr -> next;
}
ptr1 -> next = NULL;
free (ptr);
printf ("Node deleted from end\n");
}
}
void delete_at_pos()
{
    struct node *ptr;
    struct node* ptr1;
    int pos;
    printf ("Enter the position of deletion:\n");
    scanf ("%d", &pos);
    ptr = head;
    for (int i= 0; i< pos -1; i++)
    {
        ptr1 = ptr;
        ptr = ptr -> next;
        if (ptr == NULL
        {
            printf ("There are less elements in the
                list\n");
```

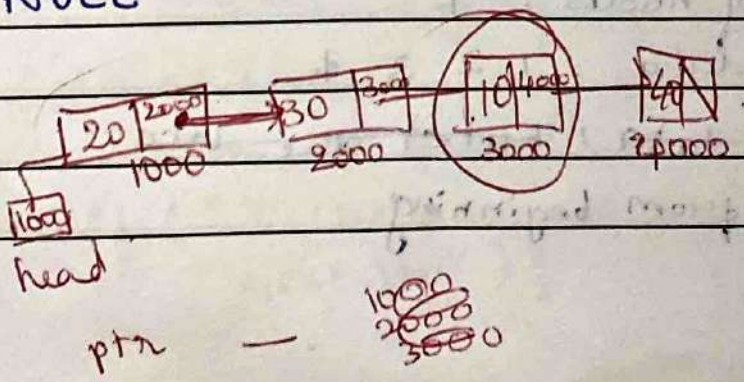
store™

```
return ;
}
}
{
  ptr1 -> next = ptr -> next;
  free (ptr);
  printf ("Node deleted from position %d\n", pos);
}
void display ()
{
  struct node *p = head;
  printf ("List : n");
  while (p != NULL)
  {
    printf ("%d ->", p -> data);
    p = p -> next;
  }
printf ("NULL\n");
```

Output :-

Insert the elements in list
Enter no. of nodes : 4
Enter the data : 1 2 3 4
~~Enter the data : Enter the data~~
1 Delete from beginning

2 Delete at end
3 Delete at particular position
4 Display
5 Exit
Enter choice 4
List
1 → 2 → 3 → 4 → NULL
Enter choice : 1
Node deleted from beginning
Enter choice : 4
List
2 → 3 → 4 → NULL
Enter choice : 2
Node deleted from end
Enter choice : 4
List :
2 → 3 → NULL
Enter the position of deletion : 2
Node deleted from position 2
Enter choice 4
List
2 → NULL



2] Minstack

Leetcode Program

```
#include <stdlib.h>
typedef struct {
    int *stack;
    int *minStack;
    int top;
} MinStack;

MinStack* minStackCreate() {
MinStack* stack = (MinStack*) Malloc(sizeof(MinStack));
stack -> stack = (int*) malloc(sizeof(int) * 50);
stack -> minStack = (int*) malloc(sizeof(int) * 50);
stack -> top = -1;
return stack;
}
void minStackPush(MinStack* obj, int val) {
obj -> top++;
obj -> stack[obj -> top] = val;
if (obj -> top == 0 || val <= obj -> minStack[obj -> top - 1]) {
obj -> minStack[obj -> top] = val;
} else {
obj -> minStack[obj -> top] = obj -> minStack[obj -> top - 1];
}
}
```

store™

```
void minStackPop (MinStack* obj) {
obj->top--;
}
int minStackTop (MinStack * obj) {
return obj->stack [obj->top];
}
int minStackGetMin (MinStack * obj) {
return obj->minStack [obj->top];
}
void minStackFree (MinStack*obj) {
free(obj->stack);
free(obj->minStack);
free(obj);
}
```

Output

Accepted.

store™

25 01 2024

LAB 6

Queue

```
#include<stdio.h>
#include<stdlib.h>
struct node {
int data;
struct node* next;
};

void append(struct node** head, int new_data){
    struct node* new_node = (struct node*) malloc
                                (sizeof(struct node));
new_node -> data = new_data;
new_node -> next = NULL;
struct node* last = *head;
if (*head == NULL)
*head = new_node;
else {
while (last -> next != NULL)
    last = last ->next;
  last -> next = new_node;
}
}
void display (struct node* head){
 if (head == NULL) {
 printf ("Linked list empty.\n");
  return;
}
```

1/2/24

```
printf ("Queue:");
while (head != NULL) {
printf ("%d", head -> data);
head = head -> next;
}
void del-head (struct node ** head) {
if (*head == NULL) {
printf ("List is empty \n");
return;
}
struct node * temp = (*head) -> next;
free (*head);
*head = temp;
}
int main () {
struct node * head = NULL;
int choice, a;
while (choice < 4) {
printf ("1. Push \n 2. pop \n 3. Display \n choice:");
scanf ("%d", &choice);
switch (choice)
{
case 1:
printf ("Enter value :");
scanf ("%d", &a);
```

store™
67

```
append (&head, a);
display (head);
break;
case 2:
    del-head (&head);
    display (head);
    break;
    case 3:
        display (head);
    default:
    break;
}
}
return 0;
}
```

Output:

1 Push
2 Pop
3 Display
choice : 1
Enter value : 1
Queue : 1

store™

1 Push
2 Pop
3 Display
choice : 3
Queue : '
1 Push
2 Pop
3 Display
Queue : Linked List empty.

Stack Program 2

Single Linked List : sort, Reverse, concatenation
Program 3

Stack Program 2

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node * next
};
void append ( struct node ** head, int new_data){
```

store™
67

```
struct node * new_node = (struct node *) malloc
(sizeof (struct node));

new_node -> data = new_data;
new_node -> next = NULL;
struct node * last = *head;
if (*head == NULL)
    *head = new_node;
else {
    while (last -> next != NULL)
    last = last -> next;
    last -> next = new_node;
}
}
void display (struct node * head) {
if (head == NULL) {
printf ("Linked list empty.\n");
return;
} printf ("stack:");
    while (head != NULL) {
printf ("%d", head -> data);
head = head -> next;
}
printf ("\n");
}
```

11/2/24

store™
67

```
void del_end(struct node *head){
if (head = NULL){
printf("List Empty\n");
return;
}
  struct node * last = head;
  struct node * prev;
  while (last->next != NULL){
   prev = last;
   last = last -> next;
  }
  free(last);
  prev->next = NULL;
}

int main(){
  struct node * head = NULL;
  int choice, a;
  while (choice<4){
  printf("1. Push\n2. Pop\n3. Display\n choice:");
  scanf("%d", &choice);
   switch (choice)
  {
   case 1:
    printf("Enter value :");
    Scanf("%d", &a)
```

store™ 67

```
append (&head, a);
display (head);
break;
case 2:
del_end (head);
display (head);
break;
case 3:
display (head);
default:
break;
}
}
return 0;
}
```

output:

1 Push
2 Pop
3 Display
choice 1
Enter value : 1
stack : 1
1 Push
2 Pop
3 Display.

store™

choice : 1

Enter value : 2

Stack : 1 2

1 Push

2 Pop

3 Display

choice : 2

Stack : 1

1 Push

2 Pop

3 Display

choice : 3

Stack : 1

store™
67

Program 3

Linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
  int data;
  struct node *next;
};
void append(struct node **head, int new_data)
{
  struct node* new_node = (struct node *) malloc
                          (sizeof(struct node));
  new_node->data = new_data;
  new_node->next = NULL;
  struct node *last = *head;
  if (*head == NULL)
    *head = new_node;
  else
  {
    while (last->next != NULL
      last = last->next;
    last->next = new_node;
  }
}
```

11/2/24

store™
67

```
void display (struct node *head)
{
  if (head == NULL)
{
  printf ("Linked list empty.\n");
  return;
}
printf ("Linked list:");
while (head != NULL)
{
  printf ("%d", head->data);
  head = head->next;
}
  printf ("\n");
}
void bubble_sort (struct node *head)
{
   struct node *prev;
   struct node *cur;
   int nex;
   int flag = 1;
   int flag = 1;

   while (flag)
   {
```

store™
67

```
prev = head;
while (prev != NULL && prev -> next != NULL)
{
  cur = prev -> next ;
  if (cur -> data < prev -> data)
  {
    nex = cur -> data ;
    cur -> data = prev -> data ;
    prev -> data = nex ;
  }
  prev = prev -> next ;
}
  int max = 0 ;
  prev = head ;
while (prev != NULL)
{
  if (max > prev -> data)
  {
    flag2 = 0;
    break;
  }
    max = prev -> data ;
    prev = prev -> next ;
  }
  if (flag2)
    flag = 0 ;
```

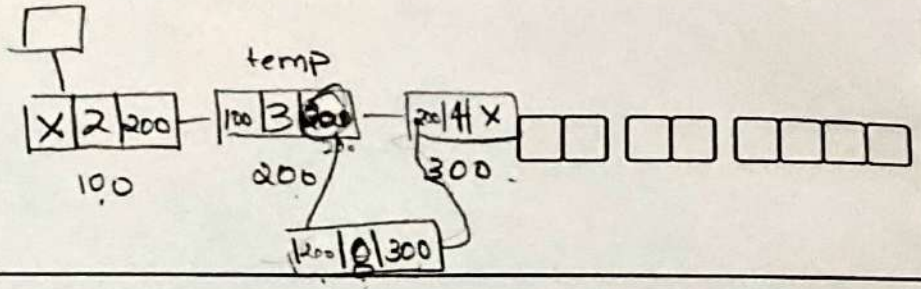
store™

```
else
    flag2 = 1;
}
}
void reverse(struct node **head)
{
  struct node *prev = NULL;
  Struct node* current = *head;
  Struct node* next = NULL;
  while (current != NULL)
  {
  next = current -> next;
  current -> next = prev;
  prev = current;
  current = next;
  }
 *head = prev;
}
void concat(struct node * head1, struct node *head2){
 struct node * prev = head2;
while (prev1 != NULL) {
    append(&head1, prev -> data1);
    prev = prev -> next;
}
}
```

store™
67

```
int main()
{
 struct node *head = NULL;
append (&head, 5);
append (&head, 2);
append (&head, 3);
append (&head, 4);
append (&head, 1);
append (&head, 6);
display (head);
bubble_sort (head);
display (head);
 reverse (&head);
 display (head);
struct node * head2 = NULL;
append (&head2, 76);
append (&head2, 43);
 append (&head2, 34);
concat (head, head2);
display (head);
 return 0;
 }
```

store™



Output :-

Linked list : 5 2 3 4 1 6

Linked list : 1 2 3 4 5 6

Linked list : 6 5 4 3 2 1

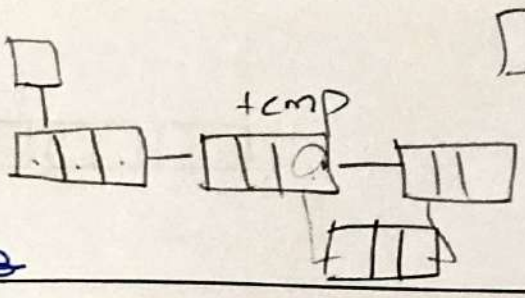
Linked list : 6 4 3 2 1 76 ~~7~~3 34

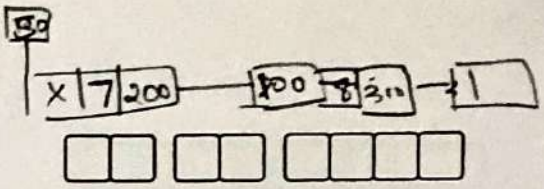
1/7/2024

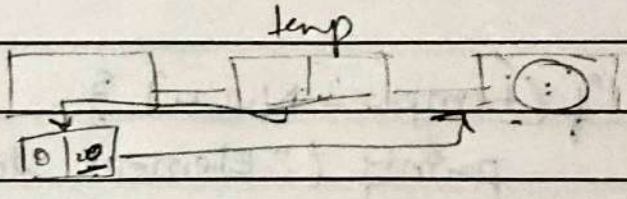
LAB - 7

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct Node {
 int data;
 struct Node *next;
 struct Node* prev;
} Node;
 Node * head = NULL;
int count = 0;
 void insert (int data, int position) {
   if (position == 0) {
 Node *newNode = malloc (sizeof (Node));
 newNode -> data = data;
 newNode -> next = head;
 newNode -> prev = NULL;
if (head != NULL) head -> prev = newNode;
```

store™







~~AB~~

```
head = newNode;
count++;
return;
} else {
Node * temp = head;
for (int i = 0; i < position - 1; i++)
  temp = temp -> next;
Node * newNode = malloc (sizeof (Node));
  newNode -> data = data;
  newNode -> next = temp -> next;
  newNode -> prev = newNode;
  temp -> next = newNode;
  count++;
  return;
}
}
void delet (int element) {
  int position = 0;
  Node * temp = head;
  if (head == NULL) {
    printf ("List is empty, cannot delete \n");
    return;
  }
  for (; position < count; temp = temp -> next,
       position++)
```

15/2/24

```
if (temp == NULL) {
    printf ("Element does not exist in the list \n");
    return;
}
    if (position == 0) {
    Node * temp = head;
    temp -> prev = NULL;
    free (head);
    head = temp;
    count --;
    return;
    }
    else if (position > count || position < 0) {
    printf ("Unable to delete at the given position \n");
    return;
} else {
    Node * temp = head;
    for (int i = 0; i < position; i++)
     temp = temp -> next;
    temp -> prev -> next = temp -> next;
    free (temp);
    count --;
    return;
    }
}
```

store™ 67

```
void display () {
Node * temp = head;
printf ("Linked List : ");
while (temp -> next != NULL) {
printf ("%d", temp -> data);
temp = temp -> next;
}
printf ("%d", temp -> data);
printf ("\n");
}
int main () {
int data, choice, pos;
printf (" 1. Insert \n 2. Delete \n 3. Exit : \n choice:");
scanf ("%d", &choice);
while (choice != 3) {
if (choice == 1) {
printf (" Enter data and position : ");
scanf ("%d %d", &data, &pos);
insert (data, pos);
printf ("count: %d \n", count);
}
else if (choice == 2) {
printf ("Enter element :");
scanf ("%d", &pos);
delete (pos);
```

store™

```
printf("count : %d \n", count);
}
display ();
printf("Enter choice : ");
scanf("%d", &choice);
}
return 0;
}
```

Output :

1] Insert
2 Delete
3 Exit
choice : 1
Enter data and position : 4 0
count : 1
Linked List : 4
Enter choice : 1
Enter data and position : 7 1
Count : 2
Linked list : 4 7
Enter choice : 2
Enter element : 4
count

store™
67

Leet code

```
void append (struct ListNode **head, int val) {
    struct ListNode *new = (struct ListNode*)
                           malloc (sizeof(struct ListNode));
new -> val = val;
new -> next = NULL;
if (*head == NULL) {
    *head = new;
else {
    while (prev -> next != NULL)
        prev = Prev -> next
    prev -> next = new
}
}
int length (struct ListNode* head) {
    int len = 0;
    while (Prev != NULL) {
    len++;
    prev = prev -> next;
}
```

store™

15 02 2024

LAB - 8

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

typedef struct node {
    int data;
    struct Node * left;
    struct Node * right;
} node;

node * root = NULL

void insert (node ** root, int data);
void preorder (node ** root);
void postorder (node ** root);
void inorder (node **root);

int main ()
    int choice, data;
    insert (& root, 8);
    insert (& root, 3);
    insert (& root, 1);
    insert (& root, 6);
    inser (& root, 4);
```

store™
67

```
insert (&root, 7);
insert (&root, 10);
insert (&root, 14);
insert (&root, 13);

printf ("1. Preorder \n 2. Inorder \n 3. Postorder
        4. Exit \n choice: ");
      scanf ("%d", &choice);
      while (choice != 4) {
        if (choice == 1) {
        preorder (&root);
        printf ("\n");
      } else if (choice == 2) {
        inorder (&root);
        printf ("\n");
      } else if (choice == 3) {
        postorder (&root);
        printf ("\n");
      }
      printf ("Enter choice: ");
      scanf ("%d", &choice);
    }
}
```

store™
67

```
void insert (node ** root, int data) {
    if (*root == NULL) {
    node *new_node = malloc (sizeof (node));
    new_node -> data = data;
    new_node -> right = NULL;
    new_node -> left = NULL;
     *root = new node;
     return;                    root, left
    }
if (data < (*root) -> data) {
      insert (&((*root) -> left), data);
} else if (data > (*root) -> data) {
      insert (&((*root) -> right), data);
    }
  return;
}                               root, left right
void preorder (node **root) {
  if (*root != NULL) {
      print ("%d", (*root) -> data);
      preorder (&((*root) -> left));
      preorder (&((*root) -> right));
    }
}
```

store™
67

left right root

```
void postorder (node **root) {
    if (*root != NULL) {
    postorder (&((*root) -> left));
    postorder (&((*root) -> right));
    printf ("%d", (*root) -> data);
    }
}
```

left 1st → right

```
void inorder (node **root) {
    if (*root != NULL) {
    inorder (&(*root) -> left);
    printf ("%d", (*root) -> data);
    inorder (&(*root -> right));
```

Output :

1. Preorder
2. Inorder
3. Postorder
4. Exit

1st 2nd 24.

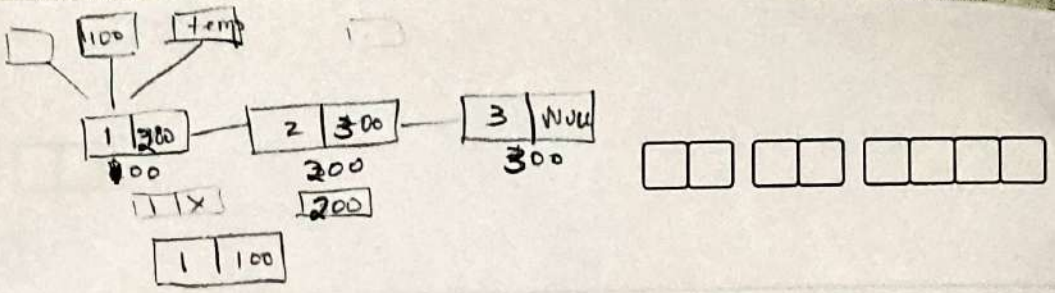
choice : 1

8, 3, 1 6 4 7 10 14 13

Enter choice 2: 1 3 4 6 7 8 10 13 14

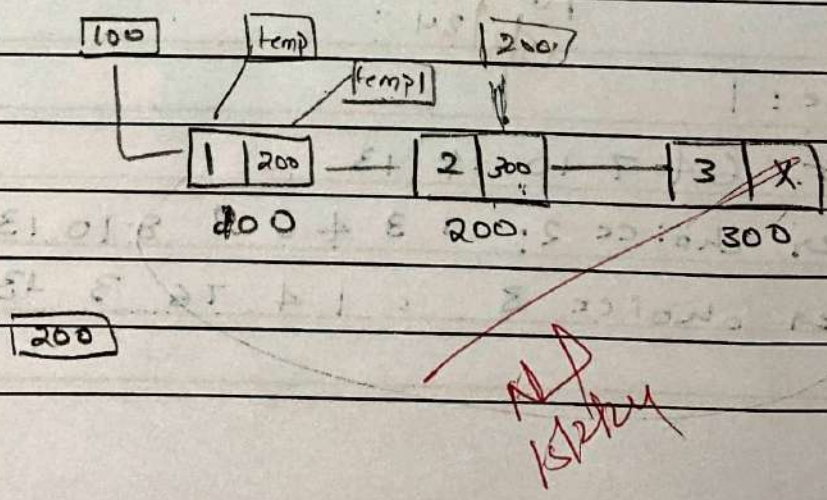
Enter choice 3 : 1 4 7 6 3 13 14 10 8

store™ 67



Leet code:

```
struct ListNode* rotateRight(struct ListNode* head,
                             int k) {
struct ListNode *temp = head;
if (head == NULL) return NULL;
if (head -> next == NULL) return head;
if (k == 0) return head;
int size = 1;
for (temp -> next != NULL; temp = temp -> next,
        size++);
k %= size;
if (k == 0) return head;
temp -> next = head;
struct ListNode *temp1 = head;
for (int i = 0; i < (size - k - 1); i++)
        temp1 = temp1 -> next;
head = temp1 -> next;
temp1 -> next = NULL;
return head;
}
```



store™
67

LAB 9

22 02 2023

bfs

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

#define size 7

void push (int a);
int pop();
void display();
void bfs (int graph [7] [7]);

int fpos = -1, rpos = -1;
int queue [size];

int main() {
int adj_matrix [7] [7] = {
        {0, 1, 0, 1, 0, 0, 0},
        {1, 0, 1, 1, 0, 1, 1},
        {0, 1, 0, 1, 1, 1, 0},
        {1, 1, 1, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 1},
        {0, 1, 1, 0, 0, 0, 0},
        {0, 1, 0, 0, 1, 0, 0},
};
```

store™
67

```
for (int i = 0; i < 7; i++) queue[i] = NULL;

bfs (adj_matrix);
return 0;
}

void bfs (int graph[][7]) {

int visited[7];
for (int i = 0; i < 7; i++) visited[i] = 0;
push (0); visited[0] = 1;
while (fpos != size) {
    for (int i = 0; i < 7; i++) {
  if (graph[queue[fpos]][i] == 1
      && visited[i] != 1) {
    push (i);
     visited[i] = 1;
  }
}
  printf ("%d", pop());
}
}
void push (int a) {
    if (fpos == -1 && rpos == -1) {
```

store™
67

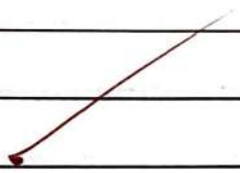
```
queue [++rpos] = a;
fpos++;
return;
}
else if (rpos == size-1) {
printf ("Queue overflow condition \n");
return;
}
else {
    queue [++rpos] = a;
    return;
  }
}

int pop() {
if (fpos == -1) {
  printf ("Queue underflow condition \n");
}
  int n = queue [fpos];
  queue [fpos] = (int) NULL;
  fpos++;
  return n;
}
```

store™

```
Void display (){
    printf ("Queue : ");
for (int i = 0; i < size ; i++)
    printf ("%d", queue [i]);
    printf ("\n");
}
```

Output :.

0 1 2 3 4 5 6 7



store™

dfs

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

#define size 7
int pos = -1;
int stack[size];

void push(int a);
int pop();
void display();
void dfs(int graph[][7]);

int main() {
    int adj_matrix[7][7] = {
        {0, 1, 0, 1, 0, 0, 0},
        {1, 0, 1, 1, 0, 1, 1},
        {0, 1, 0, 1, 1, 1, 0},
        {1, 1, 1, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 1},
        {0, 1, 1, 0, 0, 0, 0},
        {0, 1, 0, 0, 1, 0, 0}
    };
```

store™
67

```
for (int i = 0; i < 7 ; i++ ) stack [i] = NULL;
  dfs (adj-matrix);
   return 0;
}

   void dfs (int graph [] [7]) {
     int visited [7];
     for (int i = 0 ; i < 7; i++ ) visited [i] = 0;
     push (0) ; visited [0] = 1; printf ("0");

while (pos != -1) {
     bool new_node = false;
     for (int i = 0; i < 7; i++) {
     if (graph [stack [pos] ] [i] == 1 && visited
                                        [i] != 1)
     {
     new_node = true;
     push (i);
     visited [i] = 1; printf ("%d", i);
     break;
     }
}
     if (! new_node ) pop ();
     }
}
```

store™
67

```
void push (int a) {
  if (pos == size - 1) {
    printf ("stack Overflow condition");
    return ;
  }

  stack [++pos] = a;

int pop () {
  if (pos == -1) {
    printf ("stack underflow condition");
    return (int ) NULL;
  }
  return stack [pos--];
}
void display () {
    for (int i = 0; i < size ; i++) {
      printf ("%d ", stack [i]);
    }
    printf ("\n");
  }
```

Output:-

0 1 2 3 4 6 5

Sp.1
22/2/24

store™
67

hackerrank

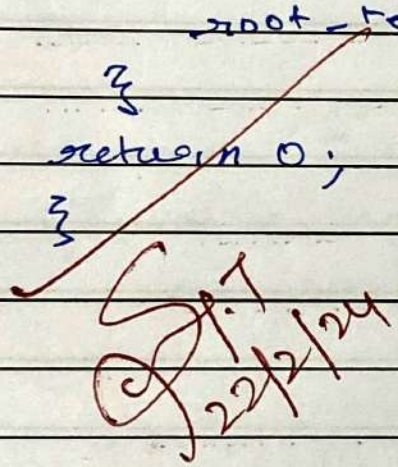
```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

struct node {
    int data;
    struct node *left
    struct node* right;
};

struct node* create_node (int val) {
if (val == -1) {
    return NULL;
}
struct node* temp = (struct node*) malloc
        (sizeof (struct node));
    temp -> data = val
    temp -> left = NULL;
    temp -> right = NULL;
    return temp;
}
```

store™ 67

```
void inorder (struct node * root) {
    if (!root) {
        return;
    }
    inorder (root -> left);
    printf ("%d", root -> data);
    inorder (root -> right);
}
int max (int a, int b) {
        if (a > b) {
            return a;
        } else {
            return b;
        }
    }

int main () {

    scanf ("%d", & nodes_count);
    struct node * root_perm, * root_temp;

    struct node * q [nodes_count];
    for (i=0; i<nodes_count; i++) {
        q [i] = NULL;
    }
```

store™

```
i=0, index=1;
root_temp = root_per = create_node(i);
enqueue(q, root_temp);

while(index <= 2*nodes_count){
    root_temp = dequeue(q);
    scanf("%d", &temp1);

if(temp2==-1){
    }else{
        root_temp -> right = create_node(temp2);
    }
return 0;
}
```



Program 10

```
#include <stdio.h>
#include <stdlib.h>
#define Table_size 10

int h[Table_size] = {NULL};

void insert()
{
  int key, index, i, flag = 0, hkey;
  printf("\n enter a value to insert into hash
          table\n");
  scanf("%d", &key);
  hkey = key % Table_size;
  for (i=0; i < Table_size; i++)
    {
      index = (hkey+i) % Table_size;

      if ( h[index] == NULL )
        {
          h[index] = key;
          break;
        }
    }
}
```

32 % 10
2

53 % 10
3

store™

```
if (i== Table-Size)
printf ("\n element cannot be inserted \n");
}

void search()
{
int key, index, i, flag=0, hkey;
printf ("\n enter search element \n");
scanf ("%d", &key);
hkey = key % Table_size;
for (i=0; i< Table_size; i++)
    {
    index = (hkey+i) % table_size;
    if (h[index] == key)
{
printf (" value is found at index %d",
        index);
break;
}
}
}
if (i== table-size)
printf ("\n value is not found \n");
}
```

store™
67

```
void display ()
{
    int i;
    printf ("\n elements in the hash table are \n");
    for (i=0; i< table_size ; i++)
        printf ("\n at index %d \t value = %d", i, h[i]);
}
main ()
{
    int opt, i;
    while (1)
    {
        printf ("\n Press 1. Insert \t 2. Display \t
                3. Search \t 4. Exit \n");
        scanf ("%d", &opt);
        switch (opt)
        {
            case 1:
                insert ();
                break;
            case 2:
                display ();
                break;
            case 3:
```

store™
67

```
        search ();
        break;
      case 4: exit (0);
    }
  }
}
```

Output

```
Press 1. Insert 2. Display 3. Search 4 Exit
1
enter a value to insert hash table
32
Press 1. Insert 2. Display 3. Search 4 Exit
1
enter a value to insert hash table
53
Press 1. Insert 2. Display 3. Search 4 Exit
3
enter search element
53
value is found at index 3
```

29/2/24

store™
67