# Data Structures with C :

1) Lab 1:
(week 1) -> i) Swap 2 numbers using pointers — 21/12/23
ii) dynamic memory allocation
iii) stack implementation

2) week 2 i) infix to postfix — 28/12/23
ii) postfix evaluation

3) week 3 i) circular queue — 11/01/24 ~~11/01/23~~
4) week 4 ii) Single linked list [insertion part]

5) week 5) i) Single linked list [deletion part] — 18/1/24

6) week 6) i) sort & Reverse linked list, Concatenation of two linked list — 25/1/24
ii) Stack & Queue Operations using Single linked list

7) week 7) doubly linked list — 1/2/24

8) Bweek8) Binary Search Tree — 15/2/24

9) week 9) i) program to traverse using BFS method — 22/2/24
ii) check whether given graph is connected or not using DFS method

10) week 10) i) Hashing [linear probing — 29/2/24

21/12/23

1) write a program with a function to swap two numbers using pointers.

->

```
#include<stdio.h>
int main(){
int a, b, temp;
int *ptr1, *ptr2;
printf("enter the value of a and b:");
scanf("%d %d", &a, &b);
printf("\n before swapping the numbers a=%d and
                                b=%d", a, b);

ptr1=&a;
ptr2=&b;
temp=*ptr1;
*ptr1=*ptr2;
*ptr2=temp;
printf("\n after swapping a=%d and b=%d", a, b);
return 0;
}
```

output:

enter the value of a and b: 2 3
before swapping the numbers a=2 and b=3
After swapping a=3 and b=2

2) Write a program to implement dynamic memory allocation functions like malloc, calloc, free, realloc

→

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int *ptr, *ptr1;
int n, i;
n=5;
printf("Enter number of elements : %d \n", n);
ptr = (int *) malloc (n* sizeof(int));
ptr1 = (int *) calloc (n, sizeof(int));
if (ptr == NULL || ptr1 == NULL) {
        printf("Memory not allocated. \n");
        exit(0);
  }
  else {
     printf ("Memory successfully allocated using
      malloc. \n");
    free (ptr);
    printf("Malloc Memory successfully freed. \n");
    print ("\n Memory successfully allocated using
         calloc. \n");
    free (ptr1);
    print f ("Calloc Memory successfully freed. \n");
  }
return 0;
}
```

realloc :-

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

int main() {
  int *pts, i, n1, n2;
  printf("Enter size:");
  scanf("%d", &n1);
  pts = (int *) malloc (n1 * sizeof(int));
  printf("Addresses of previously allocated memory:\n");
  for (i = 0; i<n1; ++i)
    printf("%pc\n", pts + i);

  printf("\nEnter the new size:");
  scanf("%d", &n2);

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

  printf("Addresses of newly allocated memory:\n");
  for (i = 0; i<n2; ++i)
    printf("%pc\n", pts + i);

  free(pts);
  return 0;
}
```

output :-

```
Enter size: 5
Addresses of previously allocated Memory:
000000000736F50C
000000000736F54C
000000000736F58C
000000000736F5CC
000000000736F60C
Enter the new size: 3
Addres of newly allocated memory:
000000000736F50C
000000000736F54C
000000000736F58C
```

3) stack implementation:

→

```
#include <stdio.h>
# define SIZE 5
int stack[SIZE];
int top = -1;
void push (int elements);
void pop();
void display();
int main() {
    int choice, element;
    do {
      printf("\n stack operations.: \n");
      printf("1. push \n");
      printf("2. pop\n");
      printf("3. Display \n");
      printf("4. Exit \n");
      printf("Enter your choice:");
      scanf("%d", &choice);
    switch (choice){
      case 1:
        printf("Enter the element to push: ");
        scan("%d" & element);
        push (element);
        break;
      case 2:
          pop();
          break;
      Case 3:
          display();
          break;
```

```
        case 4:
            printf("Exiting the program.\n");
            break;
        default:
            printf("Invalid choice. please enter a valid
                    option.\n");
    }
} while (choice != 4);
return 0;
}
void push(int element){
    if (top == SIZE - 1) {
        printf("Stack overflow! Cannot push element.\n");
    } else {
        top++;
        stack[top] = element;
        printf("%d pushed onto the stack.\n",
                element);
    }
}
void pop() {
    if (top == -1) {
        printf("Stack underflow! Cannot pop
                element.\n");
    } else {
        printf("%d popped from the stack.\n",
                stack[top]);
        top--;
    }
}
```

```
void display () {
    if (top == -1) {
        printf (" Stack is empty. Nothing to
            display. \n");
    } else {
        printf (" Elements in the Stack: \n");
        for (int i = 0; i <= top; i++) {
            printf ("%d ", stack[i]);
        }
        printf ("\n");
    }
}
```

output :

Stack operations :

1. push
2. pop
3. display
4. Exit

Enter your choice: 1
Enter the element to push : 34
34 pushed onto the stack.

Stack operations:
1. push
2. pop
3. display
4. Exit
Enter your choice = 1
Enter the element to push : 9
9 pushed onto the stack

Stack operations:

1. push
2. pop
3. display
4. Exit

Enter your choice: 3

Elements in the stack:

34

Stack operations:

1. push
2. pop
3. display
4. Exit

Enter your choice = 2

9 popped from the stack.

Stack operations:

1. push
2. pop
3. Display
4. Exit

Enter your choice : 4

Exiting the program.

21/12/2023

28/12/23

Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands & the binary operators + (plus), - (minus), * (multiply) and ^ (power).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
char stack[MAX];
char infix[MAX];
char postfix[MAX];
int top = -1;
void push(char);
char pop();
int isEmpty();
void inToPost();
void print();
int precedence(char);
int main()
{
 printf("enter infix expression:");
 gets(infix);
 inToPost();
 print();
 return 0;
}
void inToPost()
{
   int i,j=0;
   char symbol,next;
   for(i=0; i<strlen(infix); i++)
   {
      symbol = infix[i];
      switch(symbol)
      {
       case '(':
           push(symbol);
           break;
       case ')':
           while((next=pop())!='(')
```

```
        postfix[j++] = next;
        break;
    case '+':
    case '-':
    case '*':
    case '/':
    case '^':
        while (!isEmpty() && precedence(stack[top]) >=
                                precedence(symbol))
            postfix[j++] = pop();
        push(symbol);
        break;
    default:
        postfix[j++] = symbol;
    }
}
while (!isEmpty())
    postfix[j++] = pop();
postfix[j] = '\0';
}
int precedence(char symbol) ---> (2)
{
switch(symbol)
{
case '^':
    return 3;
case '/':
case '*':
    return 2;
case '+':
case '-':
    return 1;
default:
    return 0;
}
}
void print() ---> (3)
{
int i=0;
```

```
printf("The equivalent postfix expression is : ");
while (postfix[i])
  {
    printf("%c", postfix[i++]);
  }
  printf("\n");
}
void push (char c)
{
  if(top == Max-1)
  {
    printf("stack overflow");
    return;
  }
  top++;
  stack[top] = c;
}
char pop()
{
  char c;
  if(top == -1)
  {
    printf("stack underflow");
    exit(1);
  }
  c = stack[top];
  top = top-1;
  return c;
}
int isEmpty()
{
  if(top == -1)
    return 1;
  else return 0;
```

output:

enter infix expression: a*b+c*d-e

The equivalent postfix expression is: ab*cd*+e-

2) postfix evaluation:

```
#include<stdio.h>
int stack[20];
int top = -1;
void push(int x)
{
 stack[++top] = x;
}
int pop()
{
 return stack[top--];
}
int main()
{
 char exp[20];
 char *e;
 int n1,n2,n3,num;
 printf("Enter the expression: ");
 scanf("%s", exp);
 e = exp;
 while(*e != '\0')
 {
   if(isdigit(*e))
   {
    num = *e - 48;
    push(num);
   }
   else
   {
    n1 = pop();
    n2 = pop();
    switch(*e)
    {
     case '+':
     {
      n3 = n1+n2;
      break;
     }
     case '-':
     {
      n3 = n2-n1;
      break;
     }
     case '*':
     {
      n3 = n1*n2;
      break;
```

NP
18/1/24

```
    case '/':
    {
      n3 = n2/n1;
      break;
    }
    }
    push(n3);
  }
  e++;
}
printf("\n The result of expression %s = %d\n\n", exp, pop());
return 0;
}
```

output: Enter the expression: 23*5+

The result of expression 23*5+ = 11

3) Linear Queue :-

```
#include<stdio.h>
#define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = -1;
int front = -1;
main()
{
  int choice;
  while (1)
  {
    printf("1. Insert element to queue\n");
    printf("2. delete element from queue\n");
    printf("3. display all elements of queue\n");
    printf("4. Quit \n");
    printf("Enter your choice: ");
    scanf("%d", &choice: ");
    switch (choice)
    {
      case 1:
      insert();
      break;
      case 2:
      delete();
      break;
      case 3:
      display();
      break
      case 4:
      exit(1);
      default
```

```
        print("wrong choice\n");
    }
  }
}
void insert()
{
 int add_item;
 if(rear == MAX-1)
 printf("Queue overflow\n");
 else
 {
   if(front == -1)
   front = 0;
   printf("Insert the element in queue:");
   scanf("%d", &add_item);
   rear = rear + 1;
   queue_array[rear] = add_item;
 }
}
void delete()
{
  if(front == -1 || front > rear)
  {
   printf("Queue underflow\n");
   return;
  }
  else
  {
   printf("Element deleted from queue is: %d\n", queue_array[front]);
   front = front + 1;
  }
}
void display()
{
  int i;
  if(front == -1)
    printf("Queue is empty\n");
  else
  {
    printf("Queue is :\n");
    for(i = front; i <= rear; i++)
        printf("%d", queue_array[i]);
    printf("\n");
  }
}
```

output :

1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit

> Enter your choice : 1
Insert the element in queue : 67
> Enter your choice : 3
~~3. Display all~~
Queue is :
67
> Enter your choice : 2
Element deleted from queue is : 67
> Enter your choice : 4

NP
18/11/24

3b) Circular Queue

```
#include <stdio.h>
#include<stdlib.h>
#define Size 50
int Q[Size];
int rear = -1;
int front = -1;
int IsFull()
{
 if(front == (rear+1) % Size)
 {
  return 0;
 }
 else
  {
   return -1;
  }
}
int IsEmpty()
 {
  if(front == -1 && rear == -1)
   {
    return 0;
   }
   else
    {
     return -1;
    }
  }
 void Enqueue(int x)
 {
    int item;
    if(IsFull() == 0)
    {
      printf("Queue overflow \n");
      return;
    }
   else
   {
    if(IsEmpty() == 0)
     {
      front = 0;
      rear = 0;
     }
    else
    {
     rear = (rear+1) % Size;
    }
    Q[rear] = x;
}
```

```
int Dequeue()
{
  int x;
  if(IsEmpty()==0)
  {
    printf("Queue underflow \n");
  }
  else
  {
    if(front==rear)
    {
      x=Q[front];
      front=-1;
      rear=-1;
    }
    else
    {
      x=Q[front];
      front=(front+1)%Size;
    }
    return x;
  }
}
void Display()
{
  int i;
  if(IsEmpty()==0)
  {
    printf("Queue is empty\n");
  }
  else
  {
    printf("Queue elements:\n");
    for(i=front; i!=rear; i=(i+1)%size)
    {
      printf("%d\n", Q[i]);
    }
    printf("%d\n", Q[i]);
  }
}
void main()
{
  int choice, x, b;
  while(1)
  {
```

```
printf("1. Enqueue, 2. Dequeue, 3. Display. 4. Exit \n");
printf("Enter your choice: \n");
scanf("%d", &choice);
switch (choice)
{
  case 1:
    printf("Enter the number to be inserted into the queue\n");
    scanf("%d", &x);
    Enqueue(x);
    break;
  case 2:
    b = Dequeue();
    printf("%d was removed from the queue \n", b);
    break;
  case 3:
    Display();
    break;
  case 4:
    exit(1);
  default:
    printf("Invalid input \n");
}
}
```

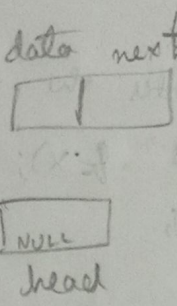
output:

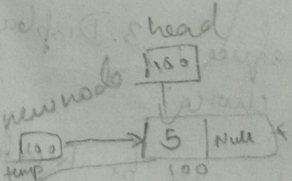
1. Enqueue, 2. Dequeue, 3. Display, 4. Exit
Enter your choice: 1
Enter the number to be inserted into the queue: 23

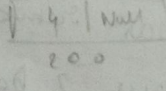
~~1. Exit~~
1. Enqueue, 2. Dequeue, 3. Display, 4. Exit
enter your choice:
1
enter the number to be inserted into the queue
13
1. Enqueue, 2. Dequeue, 3. Display, 4. Exit
enter your choice:
2
23 was removed from the queue
1. Enqueue, 2. Dequeue, 3. Display, 4. Exit
enter your choice:
3
queue elements:
13

# Single linked list

```
#include<stdio.h>
#include<stdlib.h>
  struct node
  {
    int data;
    struct node * next;
  };
  void display();
  void insert_begin();
  void insert_end();
  void insert_pos();
  struct node *head=NULL;
  void display()
  {
    printf("Elements are : \n");
    struct node * ptr;
    if(head==NULL)
    {
      printf("List is empty \n");
      return;
    }
    else
    {
      ptr=head;
      while(ptr!=NULL)
      {
        printf("%d \n", ptr->data);
        ptr=ptr->next;
      }
    }
  }
void insert_begin()
{
  struct node * temp;
  temp=(struct node*)malloc(sizeof(struct node));
  printf("Enter the value to be inserted \n");
  scanf("%d", &temp->data);
  temp->next=NULL;
  if(head==NULL)
    head=temp;
  else
  {
    temp->next=head;
  }
  head=temp;
```



```
void insert_end()
{
 struct node *temp, *ptr;
 temp=(struct node *) malloc (sizeof(struct node));
 printf("Enter the value to be inserted \n");
 scanf("%d", &temp->data);
 temp->next = NULL;
 if(head ==NULL)
 {
  head = temp;
 }
 else
 {
  ptr = head; // -> Ptr should travel till the last node
  while (ptr->next != NULL)
  {
   ptr = ptr->next;
  }
  ptr->next = temp; // -> once reached the last node
 }
}
void insert_pos()
{
 int pos, i;
 struct node * temp, * ptr;
 printf("Enter the position");
 scanf("%d", &pos);
 temp=(struct node *) malloc (sizeof(struct node));
 printf("enter the value to be inserted \n");
 scanf("%d", &temp->data);
 temp->next = NULL;
 if(pos ==0)
 {
  temp->next = head;
  head = temp;
 }
 else
 {
  for(i=0, ptr=head; i<pos-1; i++)
  {
   ptr = ptr->next;
  }
  temp->next = ptr->next;
  ptr->next = temp;
 }
```

```
void main()
{
 int choice;
 while(1)
 {
 printf("\n1. to insert at the beginning\n
          2. to insert at the end \n
          3. to insert at the position \n
          4. to display \n 5. exit \n");
    printf("Enty your choice: \n");
    scanf("%d", &choice);
    switch(choice)
    {
     case 1:
         insertbegin();
         break
     case 2:
         insert_end();
         break;
     case3:
         insert_pos();
         break;
     case4:
         display();
         break;
     case5:
         exit(0);
         break;
     default:
         printf("Invalid choice \n");
         break;
        }
    }
```

NLP
11/1/24

o/p
1. to insert at the beginning
2. to insert at the end
3. to insert at the position
4. to display
5. exit

enter your choice:
1
enter the value to be inserted
12
1. to insert at the beginning
2. to insert at the end
3. to insert at the position
4. to display
5. exit
enter your choice
1
enter the value to be inserted
13
1. to insert at the beginning
2. to insert at the end
3. to insert at the position
4. to ~~insert at~~ display
5. exit
enter your choice
4
elements are:
12
13
1. to insert at the beginning
2. to insert at the end
3. to insert at the position
4. to display
5. exit
enter your choice:
2
enter the value to be inserted
80
1. to insert at the beginning
2. to insert at the end
3. to insert at the position
4. to display
5. exit
enter your choice:
3
enter the position 1
enter the value to be inserted
999
elements are:
12
13
80
12
999
13
80

18/1/24

Single linked list

[To delete at beginning, end and at specific position]

```
void begin_delete()
{
 struct node *ptr;
 if(head == NULL)
   {
     printf("\nList is empty\n");
   }
 else
   {
     ptr = head;
     head = ptr -> next;
     free(ptr);
     printf("\n Node deleted from the begining\n");
   }
}
Void last_delete()
{
 struct node *ptr, *ptr1;
 if(head == NULL)
   {
     printf("\n List is empty");
   }
 else if(head -> next == NULL)
 {
   head = NULL;
   free(head);
   printf("\n Only node of the list deleted\n");
 }
else
   {
     ptr = head;
     while(ptr -> next != NULL)
     {
       ptr1 = ptr;
       ptr = ptr -> next;
     }
     ptr1 -> next = NULL;
     free(ptr);
     printf("\n Deleted node from the last\n");
   }
}
```

ALP 18/1/24

```
void random_delete ()
{
    struct node *ptr, *ptr1;
    int loc, i;
    printf(" \n Enter the location to perform deletion \n");
    scanf("%d", &loc);
    ptr = head;
    for(i=0; i<loc; i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;
        if(ptr == NULL)
        {
            printf("\n can't delete");
            return;
        }
    }
    ptr1->next = ptr->next;
    free(ptr);
    printf("\n Deleted node %d", loc+1);
}
```

O/P:

elements are:

5
4
3
2
1
11
12
13
14

1. to insert at the beginning
2. to insert at the end
3. to insert at the position
4. to display
5. delete from beginning
6. delete from end
7. random delete
8. exit

enter your choice

5

4
3
2
1
11
12
13
14

enter your choice

6

4
3
2
1
12
13

enter your choice

7

Enter the location of the node after which you want to perform deletion

1

4
2
1
11
12
13

18/1/24

leet code

1)

```
#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) * 10000);
   stack->minStack = (int*) malloc (sizeof(int) * 10000);
   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];
     }
 }
void minStactPop (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);
}
```

o/p:

Input:

["MinStack", "push", "push", "push", "getMin", "pop",
"top", "getMin"]

[[], [-2], [0], [-3], [], [], [], []]

output:

[null, null, null, null, -3, null, 0, -2]

Expected:

[null, null, null, null, -3, null, 0, -2]

Concatenate, sort, reverse :- 25/1/24

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

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

void append(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;
}

void printList (struct node *node)
{
    while (node != NULL)
    {
        printf ("%d -> ", node->data);
        node = node->next;
    }
    printf("Null\n");
}
```

NOTE:
Double pointer used to store the address of another pointer.

```
void sorthist (struct Node **head_ref)
{
if(*head_ref == NULL)
    {
     return;
    }
int swapped, temp;
struct Node *ptr1;
struct Node *lptr = NULL;

do
  {
    swapped = 0;
    ptr1 = *head_ref;

    while (ptr1->next != lptr)
        {
        if(ptr1->data > ptr1->next->data)
          {
            temp = ptr1->data;
            ptr1->data = ptr1->next->data;
            ptr1->next->data = temp;
            swapped = 1;
          }
          ptr1 = ptr1->next;
        }
        lptr = ptr1;

    }
    while(swapped);
}

void
```

```
void reverselist(struct Node * *head_ref)
  {
    struct Node * prev = NULL;
    struct Node * current = *head_ref;
    struct Node * next = NULL;

    while (current != NULL)
        {
         next = current->next;
         current->next = prev;
         prev = current;
         current = next;
        }
       *head_ref = prev;
  }

void concatenatelists(struct Node* *head1, struct Node* head2)
    {
     if(*head1 == NULL)
         {
          *head1 = head2;
          return;
         }
     struct Node* temp = *head1;
     while (temp->next != NULL)
         {
           temp = temp->next;
         }
     temp->next = head2;
    }
int main()
    {
     struct Node *list1 = NULL;
     struct Node *list2 = NULL;
     int n, data;
```

```
printf("Enter the number of elements for list 1:");
scanf("%d", &n);
printf("Enter the elements for list 1:\n");
for(int i=0; i<n; i++)
    {
    scanf("%d", &data);
    append(&list1, data);
    }
printf("Enter the number of elements for list 2:");
scanf("%d", &n);
printf("Enta the elements for List 2:\n");
for(int i=0; i<n; i++)
    {
    scanf("%d", &data);
    append(&list2, data);
    }
printf("\n Original list1:");
printList(list1);
printf("Original List 2:");
printList(list2);

sortList(&list1);
sortList(&list2);
printf("\nSorted list 1:");
printlist(list1);
printf("Sorted list 2:");
printList(list2);

concatenateLists(&list1, list2);
printf("\n concatenated list:");
printList(list1);
reverseList(&list1);
printf("\n Reversed List:");
printlist(list1);
return 0;
}
```

2) Stack using linked list

```
#include <stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();
struct node
   {
     int val;
     struct node *next;
   };
struct node *head;
void main()
 {
     int choice=0;
     printf("\n stack operation using linked list\n");
     while(choice!=4)
        {
        printf("\n choose one from the below
                                options..\n");
        printf("\n1.Push\n2.pop\n3.show\n4.exit");
         printf("\n Enter your choice\n");
         scanf("%d",&choice);
         switch(choice)
         {
           case 1:
             {
              push();
              break;
             }
           case 2:
             {
              pop();
              break;
             }
```

```
case 3 :
{
  display( );
  break;
}
case 4 :
{
  printf("Exiting..");
  break;
}
default :
{
  printf(" Please enter valid choice ");
}
};
}
```

Exit
25/1/24

```
void push ()
{
  int val;
  struct node *ptr = (struct node *) malloc(sizeof(struct node));
  if(ptr == NULL) ——> (If malloc fails to allocate memory (returns NULL), an error message is printed)
  {
    printf("not able to push the element ");
  }
  else
  {
    printf("Enter the value");
    scanf("%d", &val);
    if(head == Null)
    {
      ptr->val = val;
      ptr->next = NULL;
      head = ptr; //set the head of the stack to point to the new node
    }
    else
    {
      ptr->val = val;
      ptr->next = head;
      head = ptr; //update the head of the stack to point to new node
    }
    printf("Item pushed");
}
```

```
void pop()
{
    int item;
    struct node *ptr;
    if (head == NULL)
      {
        printf("underflow");
      }
    else
      {
        item = head->val;
        ptr = head;
        head = head->next;
        free(ptr);
        printf("Item popped");
      }
}
void display()
{
   int i;
   struct node *ptr;
   ptr = head;
   if(ptr == NULL)
     {
       printf("stack is empty\n");
     }
   else
     {
       printf("Printing stack elements \n");
       while(ptr != NULL)
         {
           printf("%d\n", ptr->val);
           ptr = ptr->next;
         }
     }
}
```

## 3) Queue using Single linked list :

```
#include <stdio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *next;
};
struct node *front;
struct node *rear;
void insert();
void delete();
void display();
void main()
{
   int choice;
   while(choice !=4)
    {
      printf("\n Queue operation using linked list");
      printf("\n 1. insert an element \n 2. Delete an element
            \n 3. Display the queue \n 4. exit \n");
      printf("\n Enter your choice ");
      scanf("%d", &choice);
      switch(choice)
       {
         case 1:
         insert();
         break;
         case 2:
            delete();
            break;
         case 3:
            display();
            break;
         case 4:
            exit(0);
            break;
         default:
            printf("\nEnter valid choice\n");
       }
}
```

25/1/24

```
void insert()
 {
   struct node *ptr;
   int item;
   ptr = (struct node *) malloc (sizeof(struct node));
   if(ptr == NULL)
     {
       printf("\n Overflow\n");
       return;
     }
   else
     {
       printf("\n Enter value \n");
       scanf("%d", &item);
       ptr->data = item;
       if(front == NULL)
         {
           front = ptr;
           rear = ptr;
           front->next = NULL;
           rear->next = NULL;
         }
       else
         {
           rear->next = ptr;
           rear = ptr;
           rear->next = NULL;
         }
     }
 }
void delete()
 {
   struct node *ptr;
   if(front == NULL)
     {
       printf("\n underflow \n");
       return;
     }
   else
     {
       ptr = front;
       front = front->next;
       free(ptr);
     }
}
```

```
void display()
{
struct node *ptr;
ptr = front;
if(front == NULL)
 {
  printf("\n Empty queue \n");
 }
else
 {
  printf("\n printing values .. \n");
  while(ptr != NULL)
   {
    ptr printf("\n %d \n", ptr->data);
      ptr = ptr->next;
   }
 }
}
```

o/p:

~~Enter the choice~~
Queue operation using linked list
1. insert an element
2. delete an element
3. display the queue
4. exit
enter the choice 1
enter the value
2
enter the choice 1
enter the value
66
enter the choice 3
printing values
2
66
enter choice 2
enter choice 3
66

8/2/24

WAP to implement doubly link list with primitive operations

- Create a doubly linked list.
- Insert a new node to the left of the node
- Delete the node based on a specific value
- Display the contents of the list.

—

leetcode - 3 :-

```
struct ListNode** splitListToParts(struct ListNode* head, int k,
                                   int* returnSize)
{
    int length = 0;
    struct ListNode* current = head;
    while (current != NULL)
    {
        length++;
        current = current->next;
    }
    int partSize = length/k;
    int extraNodes = length % k;
struct ListNode** result = (struct ListNode**)malloc(k * sizeof(struct
                                                    ListNode*));

current = head;
for (int i=0; i<k; i++)
{
    int currentPartSize = partSize + (i < extraNodes ? 1 : 0);
    result[i] = current;
for (int j=0; j<currentPartSize - 1 && current != NULL; j++)
    {
        current = current->next;
    }
if (current != NULL)
    {
        struct ListNode* nextNode = current->next;
        current->next = NULL;
        current = nextNode;
}   }
```

```
*returnSize = k;
return result;

}
```

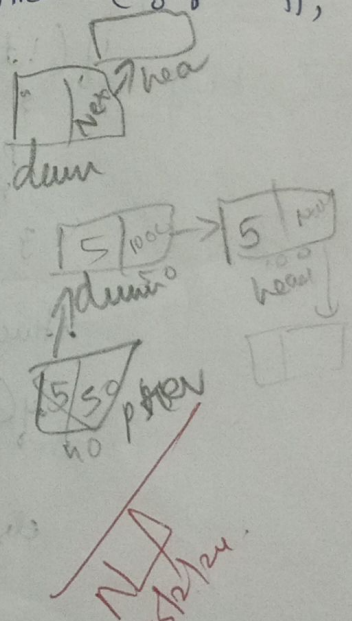
doubly linked list

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

leetcode 2:-

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
{
  if(head == NULL || left == right)
  {
    return head;
  }
  struct ListNode *dummy = (struct ListNode*)malloc(sizeof(struct ListNode));
  dummy->next = head;
  struct ListNode* prev = dummy;

  for(int i=1; i<left; ++i)
  {
    prev = prev->next;
  }
  struct ListNode* current = prev->next;
  struct ListNode* next = NULL;
  struct ListNode* tail = current;

  for(int i=left; i<=right; ++i)
  {
    struct ListNode* temp = current->next;
    current->next = next;
    next = current;
    current = temp;
  }

  prev->next = next;
  tail->next = current;

  struct ListNode* result = dummy->next;
  free(dummy);
  return result;
}
```

indicate position of element to reverse



# doubly linked list

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

struct Node
    {
    int data;
    struct Node * prev;
    struct Node *next;
    };
struct Node** *createNode(int data)
     {
      struct Node *newNode = (struct Node*) malloc (sizeof(struct Node));
      newNode->data = data;
      newNode->prev = NULL;
      newNode->next = NULL;
      return newNode;
     }
void insertLeft(struct Node** head, struct Node* targetNode,
                                    int data)

     {
       if(!targetNode)
         {
          printf("Error: Target node is Null\n");
          return;
         }
       struct Node *newNode = createNode(data);
        if(targetNode->prev != NULL)
           targetNode->prev->next = newNode;
        else
           *head = newNode;
        newNode->prev = targetNode->prev;
        newNode->next = targetNode;
        targetNode->prev = newNode;
     }
```

```
void deleteNode(struct Node ** head, int value)
{
  struct Node * current = * head;
while (current != NULL)
  {
    if(current->data == value)
      {
        if (current->prev != NULL)
          current->prev->next = current->next;
        else
          *head = current->next;
        if (current->next != NULL)
          current->next->prev = current->prev;

        free (current);
        printf ("Node with value %d deleted
                successfully \n", value);

        return;
      }
      current = current->next;

    }
    printf("Node with value %d not found \n", value);
  }

void displayList(struct Node *head)
  {
  printf ("Doubly linked list : ");
  while (head != NULL)
    {
      printf ("%d <-> ", head->data);
      head = head->next;
    }
    printf ("NULL \n");
  }
```

```
int main()
{
    struct Node* head = NULL;
    int choice, data, insertValue, deleteValue;
    do {
        printf("\n Menu : \n");
        printf(" 1. Insert a node \n");
        printf(" 2. Delete a node \n");
        printf(" 3. Display the list \n");
        printf(" 4. Exit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:
                printf("Enter data for the new node : ");
                scanf("%d", &data);

                if(head == NULL)
                    head = createNode(data);
                else
                {
                    struct Node* current = head;
                    while(current->next != NULL)
                        current = current->next;

                    struct Node* newNode = createNode(data);
                    current->next = newNode;
                    newNode->prev = current;
                }
                break;

            Case 2:
                printf("Enter the value of the node to delete : ");
                scanf("%d", &deleteValue);
                deleteNode(&head, deleteValue);
                break;

            case 3:
                displayList(head);
                break;

            case 4:
                printf("Exiting the program \n");
                break;

            default:
                print("Invalid choice");
```

```
}
Julia (choice != 4);
return 0;

}
```

week 8 15/2/24

Binary Search Tree

```
#include<stdio.h>
#include<stdlib.h>
struct node
 {
  int value;
  struct node *left;
  struct node *right;
 } *root = NULL, *temp = NULL, *t2, *t1, *t0;
   root pointed to binary search tree        temporary pointers
int key;
int i;
int n; x
int count;
void search(struct node *t);

void insert()
{
 int data;
 printf("Enter data to be inserted-");
 scanf("%d", &data);
     temp = (struct node *) malloc (sizeof(struct node));
     temp->value=data;
     temp->left = temp->right = NULL;
 if(root == NULL)
     root = temp;
 else
     search(root);
}
```

```
void search(struct node *t)
{
 if((temp->value > t->value) && (t->right != NULL))
    search(t->right);
 else if((temp->value > t->value) && (t->right == NULL))
    t->right = temp;
 else if((temp->value < t->value) && (t->left != NULL))
    search(t->left);
 else if((temp->value < t->value) && (t->left == NULL))
    t->left = temp;
}
void inorder(struct node *t)
{
 if(root == NULL)
   {
    printf("No elements in the tree\n");
    return;
   }
 if(t->left != NULL)
 inorder(t->left);
 printf("%d->", t->value);
 if(t->right != NULL)
 inorder(t->right);
}
void preorder(struct node *t)
{
 if(root == NULL)
   {
    printf("No elements in the tree\n");
    return;
   }
 printf("%d->", t->value);
 if(t->left != NULL)
 preorder(t->left);
 if(t->right != NULL)
 preorder(t->right);
}
```

15/2/24.

```
void postorder(struct node *t)
{
  if(root == NULL)
  {
    printf("No elements in the tree\n");
    return;
  }
  if(t->left != NULL)
  postorder(t->left);
  if(t->right != NULL)
  postorder(t->right);
  printf("%d->", t->value);
}

struct node * maxvaluenode(struct node *t)
{
  struct node * current = t;
  while(current && current->right != NULL)
      current = current->right;
  return current;
}

int main()
{
  int choice;
  while(1)
  {
    printf("\n menu\n");
    printf("1. insert an element into tree\n");
    printf("2. to get the max value in the tree\n");
    printf("3. to print the tree elements in inorder traversal\n");
    printf("4. to print the tree elements in preorder traversal\n");
    printf("5. to print the tree elements in postorder traversal\n");
    printf("6. to exit\n");
    printf("Enter your choice\n");
    scanf("%d", &choice);
```

```
switch (choice)
 {
case 1:
    insert();
    break;

case 2:
    tp = maxvaluenode(root);
    printf("Max value node");
    printf("Node with Maximum value = %d", tp->value);
     break;

case 3:
    printf("inorder traversal \n");
    inorder(root);
    break;

 case 4:
     printf("preorder traversal \n");
     preorder(root);
     break;

 case 5:
     printf("Postorder traversal \n");
     postorder(root);
     break;

 case 6:
     exit(0);

 default:
     printf("Invalid choice");
     break;
    }
 }
 return 0;
}
```

o/p:-

menu

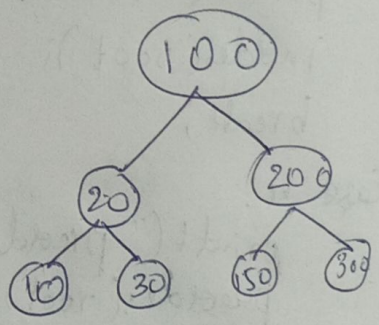
1. insert an element into tree
2. to print the tree element in inorder traversal
3. to print the tree element in preorder traversal
4. to print the tree element in postorder traversal
5. to exit

> Enter your choice

1

Enter data to be inserted 100

20
10
30
200
150
300




> Enter your choice

2

inorder traversal (left, Root, Right)

10 -> 20 -> 30 -> 100 -> 150 -> 200 -> 300 ->

preorder traversal (Root, left, Right)

100 -> 20 -> 10 -> 30 -> 200 -> 150 -> 300 ->

postorder traversal (left, Right, root)

10 -> 30 -> 20 -> 150 -> 300 -> 200 -> 100 ->

ALP
15/2/24

~~Dec 4~~

leetcode 4

## Rotate List

> Given the head of a linked list, rotate the list to the right by K places.
>

```
struct ListNode* rotateRight(struct ListNode *head, int k)
{
    if(head==NULL || head->next==NULL || k==0)
        return head;
    int length=1;
    struct ListNode *tail=head;
    while(tail->next!=NULL)
    {
        length++;
        tail=tail->next;
    }
    k=k%length;
    if(k==0)
        return head;
    struct ListNode *new_tail=head;
    for(int i=0; i<length-k-1; i++)
    {
        new_tail=new_tail->next;
    }
    struct ListNode *new_head=new_tail->next;
    new_tail->next=NULL;
    tail->next=head;
    return new_head;
}
```

o/p :-

Case 1 :

head =
[1, 2, 3, 4, 5]

k =
2

case 2 :

head =
[0, 1, 2]

k =
4

Week 9

22/2/24

# BFS

```
#include<stdio.h>
#define MAX_VERTICES 10
int n, i, j, visited[MAX_VERTICES], queue[MAX_VERTICES],
    front = 0 , rear = 0;
int adj[MAX_VERTICES][MAX_VERTICES];
void bfs(int v)
 {
  visited[v] = 1;
  queue[rear ++]=v;
  while(front < rear)
      {
       int current = queue[front ++];
       printf("%d\t", current);

       for (int i=0; i<n; i++)
        {
          if(adj[current][i] && !visited[i])
             {
               visited[i] = 1;
               queue[rear ++]= i;
             }
        }
      }
  }
int main()
 {
   int v;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   for (i=0; i<n; i++)
      {
        visited[i]=0;
      }
   printf("Enter graph data in matrix form:\n");
```

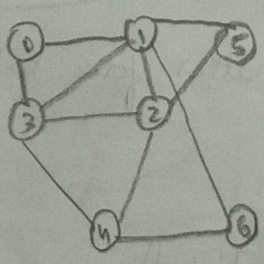
```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        scanf("%d", &adj[i][j]);
printf("Enter the starting vertex: ");
scanf("%d", &v);
bfs(v);

for (i=0; i<n; i++)
  {
    if(!visited[i])
      {
        printf("\n BFS is not possible. Not all
                nodes are reachable.\n");
        return 0;
      }
  }
return 0;
}
```

o/p

```
Enter the number of vertices: 7
Enter graph data in matrix form:
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
Enter the starting vertex: 4
4 2 3 6 1 5 0
```

4, 2, 3, 6, 1, 5, 0

visited: 4, 2, 3, 6




4 -> 2, 3, 6

2 -> 1, 3, 4, 5, 6

1 -> 0, 2, 3, 5, 6

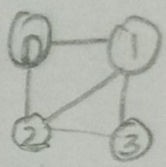
dfs :

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

#define MAX_VERTICES 10

int n, i, j, visited[MAX_VERTICES];
int adj[MAX_VERTICES][MAX_VERTICES];

void dfs(int v)
  {
   visited[v]=1;
   for(int i=0; i<n; i++)
     {
      if(adj[v][i] && !visited[i])
        {
          dfs(i);
        }
     }
  }
int main()
 {
  int v;
  printf("Enter the number of vertices: ");
  scanf("%d", &n);
  for(i=0; i<n; i++).
   {
     visited[i] = 0;
   }
  printf("Enter graph data in matrix form: \n");
  for(i=0; i<n; i++)
    for(j=0; j<n; j++)
       scanf("%d", &adj[i][j]);
```

```
printf("Enter the starting vertex ");
scanf("%d", &v);
dfs(v);

for(i=0; i<n; i++){
    if(!visited[i])
    {
        printf("\n The graph is not connected:\n");
        return 0;
    }
}
printf("\n The graph is connected.\n");
return 0;
}
```

o/p:

Enter the number of vertices: 4
Enter the graph data in matrix form:




```
0 1 1 0
1 0 0 1
1 0 0 0
0 1 0 0
```

Enter the ~~vertex~~ starting vertex: 0

The graph is connected.

Sp.f
12/2/24

29/2/24

Hacker rank :

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

struct node
{
  int id;
  int depth;
  struct node *left, *right;
};

void inorder(struct node * tree)
{
   if(tree = Null)
      return;
   inorder(tree->left);
   printf("%d", tree->id);
   inorder((tree->right));
}

int main(void)
{

   int no_of_nodes, i=0;
   int l,r, max_depth, k;
   struct node * temp = NULL;
   scanf("%d", &no_of_nodes);
   struct node * tree = (struct node *) calloc(no_of_nodes,
                                  sizeof(struct node));

   tree[0].depth = 1;
   while(i < no_of_nodes)
   {
      tree[i].id = i+1;
      scanf("%d %d", &l, &r);
      if(l == -1)
         tree[i].left = NULL;

      else
      { tree[i].left = &tree[l-1];
        tree[i].left->depth = tree[i].depth + 1;
        max_depth = tree[i].left->depth;
      }
```

```
if(n == -1)
    tree[i].right = NULL;
else
    {
      tree[i].right = &tree[n-1];
      tree[i].right->depth = tree[i].depth + 1;
      max_depth = tree[i].right->depth+2;
    }
i++;
}
scanf("%d", &i);
while(i--)
{
  scanf("%d", &l);
  n = l;
  while(l <= max_depth)
  {
    for(k = 0; k < no_of_nodes; ++k)
    {
      if(tree[k].depth == l)
      {
        temp = tree[k].left;
        tree[k].left = tree[k].right;
        tree[k].right = temp;
      }
    }
    d = l + n;
  }
  inorder(tree);
  printf("\n");
}
return 0;
}
```

o/p: input :-

```
3                3 1 2
2 3              2 1 3
-1 -1
-1 -1
2
1
1
```

Week 10

29/2/24

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_EMPLOYEES 100
#define HASH_TABLE_SIZE 7

struct Employee
{
    int key;
};
int hashfunction(int key);
void insertEmployee(struct Employee employees[], int
    hashTable[], struct Employee emp);
void displayHashTable(int hashTable[]);

int main()
{
    struct Employee employees[MAX_EMPLOYEES];
    int hashTable[HASH_TABLE_SIZE] = {0};
    int n, m, i;

    printf("Enter the number of employee: ");
    scanf("%d", &n);

    printf("Enter the number of employees: ");
    scanf("%d", &n);

    printf("Enter employee records:\n");
    for (i=0; i<n; ++i) {
        printf("Employee %d:\n", i);
        printf("Employee Enter 4-digit key");
        scanf("%d", &employees[i].key);
        insertEmployee(employees, hashTable,
                       employees[i]);
    }

    printf("\nHash Table:\n");
    displayHashTable(hashTable);
    return 0;
}
```

```
int hashfunction(int key)
{
    return key % HASH_TABLE_SIZE;
}
void insatEmployee(struct Employee employees[], int
        hashTable[], struct Employee emp)
{
    int index = hashfunction(emp.key);

while (hashTable[index] != 0)
    {
      index = (index + 1) % HASH_TABLE_SIZE;
    }
hashTable[index] = emp.key;
}
void displayHashTable(int hashTable[])
{
  int i;
  for(i=0; i<HASH_TABLE_SIZE; ++i)
    {
      printf("%d ->", i);
      if( hashTable[i] == 0)
        {
          printf("Empty\n");
        } else
          {
            printf("%d \n", hashTable[i]);
          }
    }
}
```

o/p:

Enter the number of employees: 4
Enter employee records:

Employee 1:
Enter digit Key: 700
Employee 2:
Enter digit Key: 85
Employee 3:
Enter digit Key: 101
Employee 4:
Enter digit Key: 73

Hash Table

0 -> 700
1 -> 85
2 -> Empty
3 -> 101
4 -> 73
5 -> Empty
6 -> Empty

29/12/24