Date / /
Page

Swapping of Two number using pointers.
Dynamic memory allocation
Stack Implementation

Stack Implementation

```
# include <stdio.h>
# include <stdlib.h>
# define SIZE 4

int top = -1; inputs[SIZE];
void push();
void pop();
void ~~show~~; display();

int main():
{
  int D;
  while (1)
  {
    printf("Performance operation on
          the stack");
    printf("\n1. Push element \n2. Pop
          element \n3. Display element");
    printf("\n\n Enter choice 1\2\3\4");
    scanf("%d", &D);
    switch(D)
    {
     case 1:
        push();
        break;
     case 2:
        pop(); break;
```

```
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\n Invalid Choice");
    }
  }
}
void push()
{
    int x;
    if(top == SIZE-1)
    {
        printf("\n Overflow!!");
    }
    else
    {
        printf("\n Enter element to be
                added onto the stack:");
        scanf("%d", &x);
        top = top+1;
        inpu[top] = x;
    }
}

void pop()
{
    if(top == -1)
    {
        printf("\n Underflow!!");
    }
```

```
    else
    {
        printf("\n Poped element : %d
            ", inpu[top]);
        top = top - 1;
    }
}

void display()
{
    if (top == -1)
    {
        printf("\n Underflow!!");
    }
    else
    {
        printf("\n Element present in
            the stack:");
        for(int i = top; i >= 0; i--)
        {
            printf("%d\n", inpu[i]);
        }
    }
}
```

Date __/__/__
Page

Output:

Operation on Stack
N1. Push element
N2. Pop element
N3. Display element

Enter the choice: 1

Enter element to be added: 4

Operation on Stack.

N1. Push element
N2. Pop element
N3. Display element

Enter the choice: 3

Element present in stack = 4

21/12

Date __/__/__
Page _____

Week - 1

1) Enter
1 to make a new account
2 to withdraw
3 to deposit
4 to check balance
5 to exit

1

Enter account name
Pranav
Enter age 18
4
Balance is 5500
3
Enter the amount to deposit.
600
2
Enter the amount to withdraw
1000
5

3) Enter the element to search
9

$$\begin{bmatrix} 4 & 8 & 3 \\ 7 & 5 & 9 \\ 7 & 2 & 6 \end{bmatrix}$$

9 is present.

Date __/__/__
Page

4) Enter the string

Try name is Pavanav
Enter substring to be searched
is
Substring is present.

5) Enter the element

11 22 33 44 55 66 77 88 99

Enter element to search
77

Last occurance of 77 is at 7.

6) Enter elements

11 22 33 44 55 66 77 88 99
Enter element to be searched
22

Element 22 is at 2.

8) Enter elements

11 22 33 44 55

Biggest is 55
Smallest is 11

Date / /
Page

Swapping of Two numbers

```
#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 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.

20
30
Before Swapping a = 20 & b = 30
After Swapping a = 30 & b = 20

Infix to Postfix

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

# define MAX_SIZE 100

int isoperator(char ch)
{
  return (ch == '+' || ch == '-' || ch == '*' ||
   ch == '/' || ch == '%');
}
int precedence(char operator)
{
   if (operator == '+' || operator == '-')
     return 1;
   if (operator == '*' || operator == '/' ||
     operator == '%')
   return 2;
   return 0;
}
void topostfix(char infix[], char postfix[])
{
  char stack[MAX_SIZE];
  int top = -1;
  int i, j;
  for (i=0, j=0; infix[i] != '\0'; i++)
  {
     if (infix[i] >= '0' && infix[i] <=
       '9')
     {
        postfix[j++] = infix[i];
```

```
else if (isOperator(infix[i]))
{
    while (top >= 0 && precedence(stack[top]) >= precedence(infix[i]))
    {
        postfix[j++] = stack[top--];
    }
    stack[++top] = infix[i];
}
else if (infix[i] == '(')
{
    stack[++top] = infix[i];
}
else if (infix[i] == ')')
{
    while (top >= 0 && stack[top] != '(')
    {
        postfix[j++] = stack[top--];
    }
    if (top >= 0 && stack[top] == '(')
    {
        top--;
    }
}
while (top >= 0)
{
    postfix[j++] = stack[top--];
}
postfix[j] = '\0';
```

```
int main()
{
    char infix[MAX_SIZE],
    postfix[MAX_SIZE];

    printf("Enter infix expression:");
    scanf("%s", infix);

    topostfix(infix, postfix);

    printf("Postfix expression: %s\n",
    postfix);

    return 0;

}
```

Output: Enter infix expression

A * B + C

Postfix expression: AB* +

Date __/__/____
Page _____

Evaluation of Postfix

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

#define MAX 100.

int stack [MAX];
int top = -1;

void push (int item)
{
  if top >= MAX = -1)
  {
    printf ("Stack Overflow \n");
    return;
  }
    stack [++top] = item;
}
int pop ()
{
  if (top < 0)
  {
    printf ("Stack Underflow \n");
    return -1;
  }
  return stack [top--];
}
int eval_postfix (char postfix[])
{
  int i, operand1, operand2, result;
```

```
char ch;

for (i=0; postfix[i] != '\0'; i++)
{
    ch = postfix[i];
    if (isdigit(ch))
    {
      push (ch - '0');
    }
    else
    {
      operand2 = pop();
      operand1 = pop();
    switch (ch)
    {
    case '+':
      result = operand1 + operand2;
      break;
case '-':
      result = operand1 - operand2;
      break;
case '*':
      result = operand1 * operand2;
      break;
case '/':
      result = operand1 / operand2;
    break;
default:
      printf("Invalid Expression\n");
      exit(1);
    }
  push(result);
}
```

```
}
    return pop();
}
int main()
{
    char postfix[MAX];
    printf("Enter postfix expression: ");
    gets(postfix);
    int result = evaluate_postfix(postfix);
    printf("Result : %d \n", result);
    return 0;
}
```

Output:

Enter postfix Expression: 34+

Result = 7

Enter postfix Expression: 12*34*+

Result = 14

Sk

28/12

Date __/__/__
Page

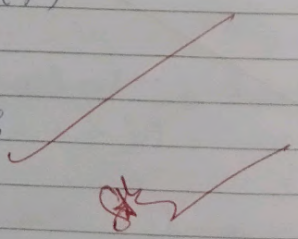
Circular Queue

```
#include <stdio.h>
#define max 10.
int queue [max];
int front = -1;
int rear = -1;
void enqueue (int element)
{
  if (front == -1 && rear == -1)
  {
    front = 0;
    rear = 0;
    queue [rear] = element;
  }
  else if ((rear + 1) % max == front)
  {
    printf ("Queue is overflow!!");
  }
  else
  {
    rear = (rear+1) % max;
    queue[rear] = element;
  }
}
int dequeue()
{
  if ((front == -1) && (rear == -1))
  {
    printf ("\n Queue is underflow!!");
  }
else if (front == rear)
```

```
{
  printf("\n The dequed element is %d",
   queue[front]);
  front = -1;
  rear = -1;
}
else
{
  printf("\n The dequeued element is %d",
  queue[front]);
    front = (front + 1) % max;
}
}
void display ()
{
  int i = front;
  if (front == -1 || rear == -1)
  {
    printf("\n Queue is empty!! ");
  }
  else
  {
    printf("\n Elements in Queue are: ");
    while (i <= rear)
    {
      printf("%d,", queue[i]);
      i = (i+1) % max;
    }
  }
}
```

```
int main()
{
 int choice = 1, x;
 while (choice < 4 && choice != 0)
 {
  printf("\n Press 1: Insert an element");
  printf("\n Press 2: Delete an element");
  printf("\n Press 3: Display the element");
  printf("\n Enter your choice");
  scanf("%d", &choice);

  switch (choice)
  {
   case 1:
   printf("Enter the element which is to
   be inserted");
   scanf("%d", &x);
   enqueue(x); break;
   case 2:
    dequeue(); break;
    case 3:
    display();
  }
 }
 return 0;
}
```



Date __/__/__
Page _____

Output:

Press 1 : Insert an element
Press 2 : Delete an element
Press 3 : Display the element
Enter your choice 1

Enter the element which is to be inserted
1 2

Press 1 Insert an element
Press 2 Delete an element
Press 3 Display the elements
Enter your choice 1

Enter the element which is to be inserted
1 3

Press 1 Insert an element
Press 2 Delete an element
Press 3 Display the elements
Enter your choice.

Enter the element which is to be deleted
1 3

Press 1 Insert an element
Press 2 Delete an element
Press 3 Display the element.
Enter your choice. 3

Element in Queue are : 12

Date __/__/__
Page _____

Linear Queue

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

int front = -1, rear = -1;
int Q[MAX];
void insert(int item)
{
  if (rear == MAX-1)
  {
    printf("overflow\n");
    exit(EXIT_FAIL);
  }
  else
  {
    if (rear == -1 && front == -1)
    {
      front = rear = 0;
      Q[rear] = item;
    }
    else
    {
      rear = rear + 1;
      Q[rear] = item;
    }
  }
}
int delete()
{
  if (front == -1 || front > rear)
  {
```

```
    return -1;
    }
    else
    {
    return a[front++];
    }
}
int main()
{
    int n, ele, d;
    do
    {
        printf("1. Insert element \n
                2. Delete element \n
                3. Exit \n");
        scanf("%d", &n);

        switch (n)
        {
        case 1:
            printf("Enter the element:");
            scanf("%d", &ele);
            insert(ele);
            break;

        case 2:
            d = delete();
            if (d <= -1)
            {
                printf("Underflow \n");
                exit(EXIT_FAIL);
            }
```

```
        printf("The element deleted is:
        %d\n", d);
        break;
        case 3:
            printf("Exiting the program\n");
            break;
        default:
            printf("Please enter the right
            choice\n");
        }
    }
    while (n != 3);
    return 0;
}
```

Output:

```
1. Insert element
2. Delete element
3. Exit.
1
Enter the element: 12

1. Insert element
2. Delete element
3. Exit.
1
Enter the element: 13
1. Insert element
2. Delete element
3. Exit
2
The element deleted is: 12.
```

Date / /
Page

1. Insert element
2. Delete element
3. Exit.

3

Exiting Program.

Malloc Function

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
  int *alloc;
  int n, i;
  printf("Enter the number of elements");
  scanf("%d", &n);
  printf("Entered number of elements", n);
  alloc = (int*) malloc(15*sizeof(int));
  if(alloc == NULL)
  {
    printf("Memory not Allocated");
  }
  else
  {
    printf("Memory Allocation
    successfull");
  for(i=0; i<n; i++)
  {
    printf("%d", alloc[i]);
  }
  } return 0; }
```

Date __/__/__
Page _____

output:

Enter the number of element 4

Entered number of elements

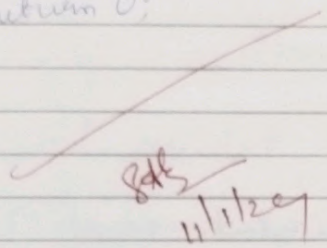
Memory Allocation successful 1, 2, 3, 4

Calloc Function

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
  int *alloc;
  int n, i;
  printf("Enter the number of elements");
  scanf("%d", &n);
  printf("Entered number of elements\n");
  alloc = (int*) calloc(15, sizeof(int));
  if (alloc == NULL)
  {
  printf("Memory not Allocated");
  }
  else
  {
  printf("Memory Allocation successfull");
  for (i=0; i<n; i++)
  {
  alloc[i] = i+1;
  }
```

Date / /
Page

```
printf("The elements of array are");
for(i=0; i<n; i++)
{
    printf("%d", arr[i]);
}
return 0;
}
```



11/1/24

Date / /
Page

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)* 50);

  stack -> minstack = (int*) malloc(
  sizeof(int * 50);
   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;
}
```

Date __/__/__
Page _____

```
else
 {
  obj -> minstack [obj -> top] = obj -> minsta
 - ck [ obj -> top - 1];
 }
}
void minstack POP (minstack *obj {
  obj -> top --;
 }
 int minstack top (min stack* obj)
  {
    return obj -> stack [obj -> top];
  }
  int minstack get min (minstack *
  obj)
  {
    return obj -> minstack [obj -> top];
  }
  void minstack free (minstack *obj)
  {
   free (obj -> stack)
   free (obj -> minstack)
   free (obj);
  }
```

output

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

Date __/__/__
Page _____

Leet Code - 2

```
Struct ListNode * Reverse Between
    (struct listnode * start, int a, int b)

{   a -= 1;
    b -= 1;
    struct listnode* node1 = NULL;
                   * node2 = NULL,*
                   *node b = NULL,
                   * nod 3 = NULL, * ptr = start

int  c = 0;
while (ptr != NULL)
{
   if (c == a-1)
       node b = ptr* ;
   else if (c = a)
         node 1 = ptr;
   else if (c == b)
         node 2 = ptr;
   else if (c == b+1)
         node a = ptr;
       break;
    }
   c += 1;
   ptr++ = ptr -> next;
}
Struct listnode * ptr = node, *temp;
ptr = struct;
   c = 0;
```

Date __/__/__
Page

Date __/__/__
Page

```
etween
t, int a, int b)

= NULL;
2 = NULL,
= NULL,
LL, *ptr = start
```

```
while (ptr != NULL)
{
    if (c >= a && c < b)
    {
        temp = ptr -> next;
        ptr -> next = ptr;
        ptr -> ptr;
        ptr = temp;
    }
    else if (c == b)
    {
        ptr -> next = ptr;
        if (a == 0)
            start = ptr;
        else
            node b -> next = ptr;
            break;
    }
    else
        ptr = ptr -> next;
        c + 1;
    }
    return start;
}
```

Output

Date __/__/__
Page _____

Linked lists

Insertion & display

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

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

Node * head = NULL;
void push();
void append();
void insert();
void display();

int main()
{
  int choice;
  while (1)
{
  printf("1. Insert at beginning \n");
  printf("2. Insert at end \n");
  printf("3. Insert at position \n");
  printf("4. Display \n");
  printf("5. Exit \n");
  printf("Enter choice: ");
  scanf("%d", &choice);
  switch (choice)
  {
    case 1:
      push();
```

```
        break;
        case 2:
          appendend();
          break;
        case 3:
          Insert ();
          break;
        case 4:
          display ();
          break;
        default:
          printf("Exiting the program");
          return 0;
        }
    }
}

void push()
{
  Node* temp = (Node*)malloc(sizeof(Node));
  int new_data;
  printf("Enter data in the new node: ");
  scanf("%d", &new_data);
  temp->data = new_data;
  temp->next = head;
  head = temp;
}
void append()
{
  Node* temp = (Node*)malloc(sizeof(Node));
  int new_data;
  printf("Enter data in the new node:");
  scanf("%d", &new_data);
  temp->data = new_data;
```

```
temp -> next = NULL;
if (head == NULL)
{
  head = temp;
  return;
}
Node * temp1 = head;
while (temp1 -> next != NULL)
{
  temp1 = temp1 -> next;
}
temp1 -> next = temp;
}

void insert ()
{
  Node * temp = (Node *) malloc (sizeof (Node));
  int new_data, pos;
  printf ("Enter data in new node: ");
  scanf ("%d", &new_data);
  printf ("Enter the position of new node: ");
  scanf ("%d", &pos);
  temp -> data = new_data;
  temp -> next = NULL;
  if (pos == 0)
{
  temp -> next = head;
    head = temp;
    return;
}
Node * temp1 = head;
  while (pos --)
```

```
}
    temp1 = temp1 -> next;
}
Node *temp2 = temp1 -> next;
 temp -> next = temp2;
  temp1 -> next = temp;
}
void display ()
{
Node * temp1 = head;
 while (temp1 != NULL)
{
printf (" %d -> ", temp1 -> data);
 temp1 = temp1 -> next;
}
printf ("NULL\n");
}
```

Date __/__/__
Page _____

Linked List

Deletion and Display

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node* next;
};
void insend (struct Node** head_ref, int
newdata)
{
  struct Node* newnode = (struct Node*) malloc
(sizeof (struct Node));
  struct Node* last = *head_ref;
  newnode -> data = newdata;
  newnode -> next = NULL;
  if (*head_ref == NULL)
  {
    *head_ref = newnode;
    return;
  }
  while (last -> next != NULL)
  last = last -> next;
  last -> next = newnode;
  return;
  }
  void delete (struct Node** head_ref,
int key)
```

```
{
struct Node *temp = *head_ref, *pre;
if (temp != NULL && temp -> data == key)
{
*head_ref = temp -> next;
free (temp);
return;
}
while (temp != NULL && temp -> data != key)
{
pre = temp;
temp = temp -> next;
}
if (temp == NULL)
    return;
pre -> next = temp -> next;
free (temp);
}
void display (struct Node *node)
{
while (node != NULL)
 {
  printf ("%d", node -> data);
   node = node -> next;
 }
}
int main()
{
struct Node *head = NULL;
insert (&head, 1);
insert (&head, 4);
insert (&head, 5);
insert (&head, 2);
```

Date __/__/__
Page _____

```
printf("Linked list: ");
display(head);

printf("\n After deleting an element: ");
delete(&head, 5);
display(head);
}
```

Sort

#include
#include
Struct nod
{

Date __/__/____
Page _____

Single linked list

Sort, Reverse, Concatenation

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int data;
struct node * next;
};
struct node * s1 = NULL;
struct node * s2 = NULL;
struct node* start = NULL;
struct node* create (struct node*);
void sort ();
struct node* concatenate (struct node*,
 struct node*);
void reverse ();
  void display (struct node*);

int main ()
{
 int option;
  struct node* a = NULL;
 do {
    printf (" \n Options \n
          1. Create a linked list \n
          2. Create two linked list
          for concatenation \n
          3. Sort \n
          4. Concatenate \n
          5. Reverse \n
```

```
6. Display linked list \n
7. Display concatenated linked list\n
8. Exit \n");

printf("\n Enter an option to perform the following operations:");
scanf("%d", &option);
switch(option)
{
    case 1: start = create(start);
            printf("\n Linked list generated successfully \n");
            break;
    case 2: printf("\n Linked list 1:\n");
            s1 = create(s1);
            printf("\n Linked list 2:\n");
            s2 = create(s2);
            printf("Linked lists created successfully \n");
            break;
    case 3: sort();
            printf("\n Linked list sorted\n");
            break;
    case 4: a = concatenate(s1, s2);
            printf("\n Linked lists concatenated \n");
            break;
    case 5: reverse();
            printf("\n Linked list reversed\n");
            break;
```

```
case 6: printf("\n Elements in Linked
        list \n");
        display(start);
        break
case 7: Printf("\n Elements in Linked
        list after concatenation :\n");
        display(a);
        break;
    }
}
while (option != 8);
    return (0);
}
struct node* create( struct node *start)
{
    Struct node *ptr, *new_node;
    int num;
    printf("Enter -1 to exit \n");
    printf("\n Enter the data : ");
    scanf("%d", &num);

    while (num != -1)
    {
    new_node = (struct node*)malloc(sizeof
                (struct node));

    new_node -> data = num;
    if(start == NULL)
    {
        start = new_node;
        newnode -> new = NULL
    }
```

```
else
{
    ptr = start;
    while (ptr -> next != NULL)
      ptr = ptr -> next;
      ptr -> new = new_node;
      new_node -> next = NULL;
    }
    printf ("Enter the data:");
    scanf ("%d", &num);
  }
  return start;
}
void sort ()
{
    struct node *i, *j;
    int temp;
      for (i = start; i -> next != NULL;
           i = i -> next)
      {
        for (j = i -> next; j != NULL;
        j = j -> new)
        {
          if (i -> data > j -> data)
        {
          temp = i -> data;
          i -> data = j -> data;
          j -> data = temp;
        }
      }
    }
}
```

```
struct node * concatenate (struct node *t1, struct node *t2)
{
    struct node *ptr;
    ptr = t1;
    while (ptr->next != NULL)
    {
        ptr = ptr->next;
    }
}
void reverse ()
{
    struct node *prev = NULL;
    struct node *next = NULL;
    struct node *cur = start;
    while (cur != NULL)
    {
        next = cur->next;
        cur->next = prev;
        prev = cur;
        cur = next;
    }
    start = prev;
}
void display (struct node *p)
{
    struct node *ptr;
    ptr = p;
    while (ptr != NULL)
    {
        printf ("%d", ptr->data);
        ptr = ptr->next;
    }
    printf ("\n");
}
```

Date __/__/__
Page

Output

Options
1. Create a linked list
2. Create two linked for concatenation
3. Sort
4. Concatenate
5. Reverse
6. Display linked list
7. Display concatenated linked list
8. Exit.

Enter an option to perform the following operation: 1
Enter -1 to exit
Enter the data : 1
Enter the data : 23
Enter the data : 17
Enter the data : -1

Linked list created Successfully.

Options

1
2
3
4
5
6
7
8

Date: __/__/__
Page: _____

Enter an option to perform the following operation: 3

Linked list Sorted

Options
1
2
3
4
5
6
7
8

Enter an option to perform the operation: 6

Elements in linked list
1 1 1 2 3

8k2
1/2/24

Date / /
Page

Single Linked List to Simulate Stack & Queue

Stack:

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

struct node
{
  int data;
  struct node *next;
};
struct node* start= NULL
 void push ();
 void pop ();
 void display ();
 int main()
 {
  int val, option;
  do
  {
   printf ("\n Enter the number of
   operation below \n");

   1. Push \n
   2. Pop \n
   3. Display \n
   4. Exit \n");
scanf ("%d", &option);

switch (option)
{
  case 1: push ();
      break;
```

```
case 2: pop();
    break;
case 3: display();
    break;
  }
}
while (option != 4);
  return (0);
}
void push()
{
  struct node* new_node;
    int num;
    printf("Enter the data\n");
    scanf("%d", &num);
    new_node = (struct node*) malloc
        (sizeof(struct node));
    new_node->data = num;
    new_node->next = start;
    start = new_node;
}
void pop()
{
  struct node* ptr;
  ptr = start;
  if (start = NULL)
    {
      printf("Stack is empty\n");
      exit(0);
    }
  else
    {
      ptr = start;
```

```
    start = ptr -> next;
    printf("\n Element popped from
    the stack is : %d \n", ptr -> data);
    free (ptr);
  }
}
void display ()
{
  struct node* ptr;
  ptr = start;
  while (ptr != NULL)
  {
    printf("%d", ptr -> data);
    ptr = ptr -> next;
  }
  printf("\n");
}
```

Output:

Enter the "number" of operation below

1 PUSH
2 POP
3 DISPLAY
4 Exit.

1

Enter the DATA :7

1 PUSH
2 POP
3 DISPLAY
4 Exit

1

Date / /
Page

Enter the data 11

1. PUSH
2 POP
3 DISPLAY
4 Exit

1

Enter the data 16

1. PUSH
2. POP
3. DISPLAY
4. Exit

2.

Element Poped = 16

1. PUSH
2. POP
3 DISPLAY
4 EXIT

3

17 11 ~~16~~ 12/24

Date / /
Page

Queue:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node * next;
};
struct node * start = NULL;
void enqueue();
void dequeue();
void display();
int main()
{
    int val, option;
    do
    {
        printf("\n Enter the no of
perform operation \n
        1 Enqueue \n
        2 Dequeue \n
        3 Display \n
        4 Exit \n");
    scanf("%d", &option);
    switch (option)
    {
        case 1: enqueue();
        break;
        case 2: dequeue();
        break;
        case 3: display();
```

```
        break;
    }
}
while (option != 4);
    return (0);
}
void enqueue()
{
    struct node * new_node;
    int main num;
    printf ("Enter the data \n");
    scanf ("%d", &num);
    new_node = (struct node*) malloc
        (sizeof(struct node));
    new_node -> data = num;
    new_node -> next = start;
    start = new_node;
}
void dequeue()
{
    struct node * ptr, * prevptr;
    ptr = start;
    if (start == NULL)
    {
printf ("Stack is empty \n");
    exit(0);
    }
else if (start -> next == NULL)
{
    start = start -> next;
printf ("\n Element Popped: ",
    ptr -> data);
free (ptr);
}
```

```
else
{
while (ptr->next != NULL)
{
preptr = ptr;
ptr = ptr->next;
}
preptr->next = NULL;
printf("\n Element poped: "
       ptr->data);
free(ptr);
}
}
void display()
{
    struct node *ptr;
    ptr = start;
    while (ptr != NULL)
    {
    printf("%d", ptr->data);
    ptr = ptr->next;
    }
    printf("\n");
}
```

Output:

Enter the number to perform operation.

1 Enqueue
2 Dequeue
3 Display
4 Exit

1

Enter the data 11

1 Enqueue
2 Dequeue
3 Display
4 Exit.

1

Enter the data 23

1 Enqueue
2 Dequeue
3 Display
4 Exit.

1

Enter the data 42.

1 Enqueue
2 Dequeue
3 Display
4 Exit.

2

~~Enter the~~ Element poped : 11

1 Enqueue
2 Dequeue
3 Display
4 Exit.

3

23 42.

11/2/24

Implement 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 = (StructNode*)
  malloc(sizeof( Struct Node));
  newNode -> data = data;
  newNode -> prev = NULL;
  newNode -> next = NULL;
  return newNode;
}

void insertleft (struct Node** head,
int value, int target)
{
  Struct Node* newNode = CreateNode(value)
  Struct Node* current = *head;

  while (current != NULL && current->
  data != target)
  {
    current = current -> next;
  }
```

```
if (current != NULL)
{
    newNode -> prev = current -> prev;
    newNode -> next = current;
    if (current -> prev != NULL)
    {
        current -> prev -> next = newNode;
    }
    else
    {
        *head = newNode;
    }
    current -> prev = newNode;
}
else
{
    printf ("Node with value %d not
    found \n", target);
}
}
void deleteNode (Struct Node** head, int value)
{
    Struct Node* current = *head;
    while (current != NULL && current -> data
    != value)
    {
        current = current -> next;
    }
    if (current != NULL)
    {
        if (current -> prev != NULL)
        {
            current -> prev -> next = current -> next;
```

```
}
else
{
*head = current -> next;
}
if (current -> next != NULL)
{
current -> next -> prev = current -> prev;
}
free (current);
}
else
{
printf ("Node with value %d not found \n", value);
}
}
void displayList(Struct Node* head)
{
Struct Node* current = head;
while (current != NULL)
{
printf ("%d -> ", current -> data);
current = current -> next;
}
printf ("NULL\n");
}

int main ()
{
Struct Node* head = NULL;
int choice, value, target;
```

```
do {
    printf("\n Main Menu\n");
    printf(" 1. Create a doubly linked
    list \n");
    printf(" 2. Insert a new node to
    the left of the node \n");
    printf(" 3. Delete the node based
    on specific value \n");
    printf(" 4. Exit \n");
    printf(" Enter your choice: ");
    scanf("%d", &choice);

switch (choice)
{
    case 1:
    if (head != NULL)
    {
    printf(" Doubly linked list already
    exists \n");
    }
    else
    {
    head = CreateNode(1);
    head -> next = CreateNode(2);
    head -> next -> prev = head;
    head -> next -> next = CreateNod(3);
head -> next -> next -> prev = head -> next
printf(" Doubly linked list created \n");
}
break;
```

```
case 2:
printf("Enter the value to be inserted:");
scanf("%d", &value);
printf("Enter the target value:");
scanf("%d", &target);
insertLeft(&head, value, target);
printf("After insertion:");
displayList(head);
break;

case 3:

printf("Enter the value to be deleted:");
scanf("%d", &value);
deleteNode(&head, value);
printf("After deletion:");
displayList(head);
break;

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

default:
printf("Invalid choice. Please enter
a valid option\n");
}
}
while(choice != '4');
return(0);
}
```

Date __/__/__
Page _____

Output

Menu:
1. Create a Doubly Linked List
2. Insert a new node to the left of the node.
3. Delete the node based on specific value
4. Exit.

Date / /
Page

Leet Code - 03

Split Linked List

```
Struct ListNode ** splitListToParts
(Struct ListNode* head, int k, int *
returnSize)
{
  Struct listnode* ptr = head;
  * returnSize = k;
    int count = 0;
  while (ptr != NULL)
  {
    count ++;
    ptr = ptr → next;
  }
  int nums = count/k; a = count%k;

  Struct ListNode ** L = (struct ListNode**)
  calloc (k, sizeof (Struct ListNode*));

  ptr = head;

for (int i = 0; i < k; i++)
  {
    L[i] = ptr
```

```
int segmentsize = nums + (a -> 0 ! L=0];
for (int j = 1; j < segment size; j++)
{
  ptr = ptr → next;
}
```

```
if(ptr != NULL)
{
  struct ListNode* next = ptr->next;
  ptr->next = NULL;
    ptr = next;
  }
}
return L;
}
```

Output

Input ⇒ Load: [1,2,3], k=5

[1] [2] [3] [4] [5]

Date / /
Page

Trees Program

```
#include <stdio.h>
#include <stdlib.h>
struct node
{ int data;
    struct node *left;
    struct node *right;
};
struct node* newNode (int data)
{ struct node *node = (struct node*)malloc
        (sizeof (struct node));
  node -> data = data;
  node -> left = NULL;
  node -> right = NULL;
  return node;
}
struct node * insert (struct node *root, int data
{ if (root == NULL)
{ return newNode (data);
}
  else {
    if (data < root -> data)
    { root -> right = insert (root -> right, data
    }
} return root;
}

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

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

void postOrder (struct Node * root){
  if root != NULL)
{ postOrder (root -> left);
  postOrder (root -> right);
  printf ("%d", root -> data);
}
}
void display (struct node * root)
{ printf ("In-order traversal:");
  inOrder (root);
   printf ("\n");
  printf (" preorder traversal:");
  preorder (root);
   printf ("\n");
   printf ("\n");
  printf (" Postorder traversal:");
   postOrder (root);
  printf ("\n");
}
int main ()
struct node * root = NULL;
int data;
printf ("Enter the elements of the tree
  (-1 to stop ) :");
while (scanf ("%d", &data) &&
   data != -1)
```

```
{ root = insert(root, data); }
display (root);
free (root);
return 0;
```

Output

```
Enter the elements of the tree
(-1 to Stop): 9
5
7
2
4
0
-1
Inorder : 0 2 4 5 7 9
Pre order: 9 5 2 0 4 7
Post order: 0 4 2 7 5 9.
```

Date / /
Page

Leet Code - 15/02

```
class Solution {
public:
    ListNode* reverse(ListNode* head)
    { ListNode* curr = head, *prev = NULL,
    *forward = NULL;
    while(curr != NULL)
    { forward = curr->next;
      curr->next = prev;
      prev = curr;
      curr = forward;
    } return prev;
}

int getLength (ListNode* head)
{ int len = 0;
  while (head != NULL)
  { head = head->next;
    len++;
  }
  return len;
}

ListNode* rotateRight (ListNode* head,
int k) { if (head == NULL || head->next
== NULL) return head;

int len = getLength (head);
  k = k % len;
  if (k == 0)
      return head;
  ListNode* newHead = reverse(head);
  ListNode* p1 = newHead; *p2 = newHead,
  *prev1 = newHead;
```

```
while (k--)
{ prev1 = p1;
  p1 = p1 -> next;
}
prev1 -> next = NULL;
ListNode *finalHead = reverse(p2);
ListNode *joinHead = reverse(p1);

p1 = finalHead;
while (p1 -> next != NULL)
      p1 = p1 -> next;
   p1 -> next = joinHead;
    return finalHead
}
};
```

Output

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

papergrid

Date: / /

BFS

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
#define MAX_NODES 100
#define MAX_EDGES 100

int graph[MAX_NODES][MAX_NODES];
int visited[MAX_NODES];
int queue[MAX_NODES];
int front = -1, rear = -1;

void BFS(int start, int n)
{
  visited[start] = 1;
  queue[++rear] = start;

  while(front != rear)
  { int current = queue[++front];
    printf("%d", current);

    for(int i = 0; i<n; i++)
    {
      if (graph[current][i] == 1 && !visited[i])
      {
        visited[i] = 1;
        queue[++rear] = i;
      }
    }
  }
}

int main()
```

```
{
  int n, m;
  printf("Enter the number of nodes and edges");
  scanf("%d %d", &n, &m);

  printf("Enter the edges: \n");
   for(int i = 0; i < m; i++)
  {
   int a, b;
   scanf("%d %d", &a, &b);
   graph[a][b] = 1;
   graph[b][a] = 1;
  }
  int start;
  printf("Enter the starting node:");
   scanf("%d", &start);
   printf("Enter BFS traversal:");
    BFS(start, n);

    return 0;
}
```

Output

```
Enter the number of nodes and edges:
6 6

Enter the edges
1 2
1 3
2 4
2 5
3 6
5 6
Enter starting node: 1
BFS traversal: 1 2 3 4 5 6.
```

DSF

```
#include <stdio.h>
#include <stdlib.h>
int arr[20][20];
int visited[20];

void dfs(int start, int n)
{
    visited[start] = 1;
    for(int i = 0; i < n; i++)
    {
        if (arr[start][i] == 1 && !visited[i])
        {
            dfs(i, n);
        }
    }
}
int isConnected(int n)
{
    dfs(0, n);
    for(int i=0; i<n; i++)
    {
        if(!visited[i])
        {
            return 0;
        }
    }
    return 1;
}
int main()
{ int n, m;
printf("Enter the number of nodes and
        edges");
```

```
scanf ("%d %d", &n, &m);

printf ("Enter the edges: \n");
for (int i = 0; i < m; i++)
{
  int a, b;
  scanf ("%d %d", &a, &b);
  arr[a][b] = 1;
  arr[b][a] = 1;
}
if (isConnected(n))
{
  printf ("The graph is connected. \n");
}
else
{
  printf ("The graph is not connected. \n");
}
return 0;
}
```

output

```
Enter the number of nodes and edges: 4 3
Enter the edges:
0 1
1 2
2 3
```

Hacker Rank - 1

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

typedef struct Node
{
int data;
struct Node* left;
struct Node* Right;
} Node;

Node* createNode (int data)
{ Node* newNode = (Node*)malloc(sizeof
(Node));
new node -> data = data;
new node -> left = NULL;
new node -> right = NULL;
return newNode;
}
void inOrderTraversal (Node * root, int * result,
int* index)
{ if ( root == NULL ) return;
inOrderTraversal ( root -> left, result,
index );
result [ (index)++ ] = root -> data,
inOrderTraversal ( root -> right, result,
index ); }
void swap ( Node* root, int k, int level)
{
```

```
if(root == NULL) return;
if(level % k == 0)
{ Node* temp = root -> left;
root -> left = root -> right;
root -> right = temp;
}

int** swapNodes(int indexRows,
int indexColumns, int** indexes, int
queriesCount, int* queries, int* resultRows,
int* resultColumns){
Node** nodes = (Node**)malloc((indexRows
Rows+1) * sizeof(Node*));
for(int i=1; i<=indexRows-Rows;i++)
nodes[i]=createNode(i);}

for(int i=0; i<indexRows; i++)
{ int leftIndex = indexes[i][0];
int rightIndex=indexes[i][1];
if(leftIndex != -1) nodes[i+1] -> left =
nodes[leftIndex];
if(rightIndex != -1) nodes[i+1] ->
right = nodes[rightIndex];
}

int** result = (int**)malloc(queries-
Count* sizeof(int*));
*resultRows = queriesCount;
*resultColumns = indexesRows;
for(int i=0; i<queriesCount;i++)
{ swap At level(nodes[1], queries[i]);
int* traversalResult = (int*)malloc
(indexesRows* sizeof(int));
int index=0;
inorderTraversal(nodes[1], traverse
Result,
```

```
    (*indexes);
*result (*) = traversal Result;
}
free (nodes);
    return result;
}
int main ()
{
    free (result);
    for (int i = 0; i < n ; i++)
{
    free .(indexes[i]);
}
    free (indexes);
    free (queries);
    return 0;
```

output.

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

Expected

3, 2
2, 3.

papergrid

Date: / /

Hacker Rank - 2

```
#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_node, 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", &t, &r);
   if (l == -1)
     tree[i] . left = NULL;
   else
```

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

Output

Enter Key: 5
Enter Data: 50

Enter Key: 15
Enter Data: 150

Enter Key: 25
Enter Data: 250
Enter Key: 35
Enter Data: 350

Hash Table:
Index 0: Empty
Index 1: Empty
Index 2: Empty
Index 3: Empty
Index 4: Empty
Index 5: Empty
Index 6: Empty
Index 7: Key 15, Data 150
Index 8: Key 25, Data 250
Index 9: Key 35, Data 350
Index 10: Empty.

Hashing

```
#include <stdio.h>
#define TABLE_SIZE 10

int hash_table[TABLE_SIZE] = {0};
void insert(int key)
{ int i = 0;
  int hkey = key % TABLE_SIZE;
  int index = hkey;

  while (hash_table[index] != 0)
  { ++i;
    index = (hkey + i) % TABLE_SIZE;
    if (i == TABLE_SIZE)
    { printf("Hash table is full!\n");
      return;
    }
  }
  hash_table[index] = key;
}
void search(int key)
{ int i = 0;
  int hkey = key % TABLE_SIZE;
  int index = hkey;
  while (hash_table[index] != key)
  { ++i;
    index = (hkey + i) % TABLE_SIZE;
    if (i == TABLE_SIZE)
    { printf("Element not found in hash table!\n");
      return;
    }
  }
```

```
    printf("Element found at index %d
    \n", index);
}
int main()
{
    int choice;
    int key;
    while(1){
    printf("1. Insert\n");
    printf("2. Search\n");
    printf("3. Exit\n");
    printf("Enter choice:");
    scanf("%d", &choice);
    switch(choice)
{   case 1: printf("Enter key:");
            scanf("%d", &key);
            insert(key);
            break;
    case 2: printf("Enter key:");
            scanf("%d", &key);
            search(key);
            break;
        default: return 0;
    }
}
return 0;
}
```

Output

1. Insert
2. Search
3. Exit
Enter choice: 1
Enter key: 5

1. Insert
2. Search
3. Exit
Enter choice: 1
Enter key: 15.

1. Insert
2. Search
3. Exit
Enter choice: 2
Enter key: 5
Element found at index 5.

1. Insert
2. Search
3. Exit.
Enter choice: 2.
Enter key: 0
Element not found in hash table!