Name: - V D Panduranga Sai Guptha

Section: - B

Roll No: -AP21110010091

DATA STRUCTURES LAB RECORD

INDEX

Lab Experiment number: - 01

Experiment Topic: - **Conversion of infix expression to postfix expression (stacks)**

AIM: - Implementation of C language code to execute the program to convert infix expression to postfix expression by using arrays concept and with the help of stacks (Data Structures).

OBJECTIVES: - By doing this experiment students finally understand,

1. how to write efficient algorithm to a specific problem.
2. How to use arrays in data structures.
3. Concept of stacks and evaluation of expression problems.
4. How to frame prefix expressions and postfix expressions manually.
5. Concept of prefix and infix and its mechanics and properties.

CONDITIONS: -

Conditions to follow while writing code for experiment 01: -

1. Arrays should be named different.
2. Only one element can be popped once.
3. Array should take input alphanumeric.

ALGORITHM: -

```
Step 1: -start
Step 2: - input Arrays, top=-1
Step 3: - check stack is empty or not
Step 4: - push operator to stack
Step 5: - check for operands
Step 6: - pop operators according to priority
Step 7: - If the precedence of the scanned operator is greater than the precedence of the operator in the stack (or the stack is empty or the stack contains a '('), push it
Step 7: - copy popped operators and operands to another stack
Step 8: - print the stack which contains popped elements or expression elements
Step 9: -stop
```

PROGRAM: -

File name: - inpostfix.c

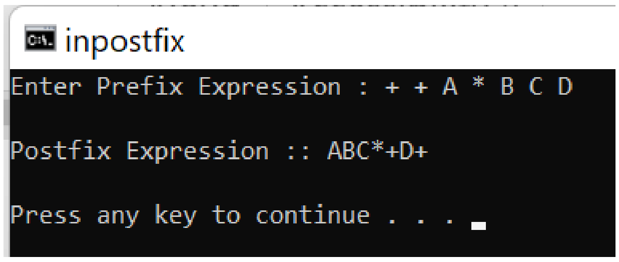
Google Drive link: -https://drive.google.com/file/d/1kFV3WNH1Qn3ovC20GkVqyqJpF3_DNF9v/view?usp=sharing

Implementation: -

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#define BLANK ' '  //blank space is initialized to BLANK


#define TAB '\t’
#define MAX 50
char prefix[MAX];
char stack1[MAX][MAX];
int top;

char *pop()   //function to pop the element from stack
{
        if(top == -1 )
        {
                printf("\nStack underflow \n");
                exit(2);
        }
        else
                return (stack1[top--]);
}
void push(char *str)  // function to insert element into the stack
{
        if(top > MAX)
        {
                printf("\nStack overflow\n");
                exit(1);
        }
        else
        {
                top=top+1; //incrementing top before progress
                strcpy( stack1[top], str);//copying string
        }
}
int isempty()  //function to check whether the stack is empty or not
{
        if(top==-1)
                return 1;
        else
                return 0;
}
int white_space(char symbol) //function check whether the element tab or black space etc
{
        if(symbol==BLANK || symbol==TAB || symbol=='\0')
                return 1;
        else
                return 0;
}
void prefix_to_postfix() //function to convert scanned expression to postfix
{
        int i;
        char operand1[MAX], operand2[MAX];
        char symbol;
        char temp[2];
        char strin[MAX];
        for(i=strlen(prefix)-1;i>=0;i--)
        {
                symbol=prefix[i];
                temp[0]=symbol;
                temp[1]='\0';
                if(!white_space(symbol))
                {
                        switch(symbol)
                        {
                        case '+':
                        case '-':
                        case '*':
                        case '/':
                        case '%':
                        case '^':
                                strcpy(operand1,pop());
                                strcpy(operand2,pop());
                                strcpy(strin,operand1);
                                strcat(strin,operand2);
                                strcat(strin,temp);
                                push(strin);
                                break;
                        default:
                             push(temp);
                        }
                }
        }
        printf("\nPostfix Expression :: ");
        puts(stack1[0]);
}

int main()
{
        top = -1;
        printf("Enter Prefix Expression : ");
        gets(prefix);   //reading the input
        prefix_to_postfix();

}
```

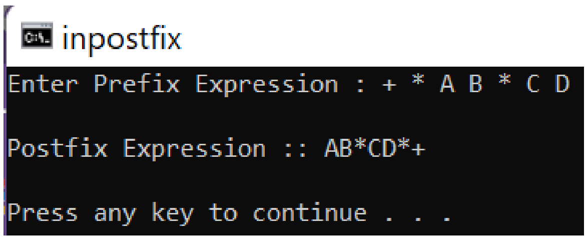
<u>OUTPUT</u>: -

Test case :1



```
inpostfix
Enter Prefix Expression : + + A * B C D

Postfix Expression :: ABC*+D+

Press any key to continue . . .
```

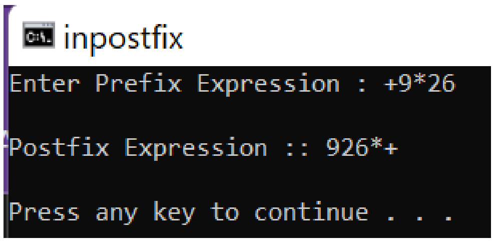
Test case :2



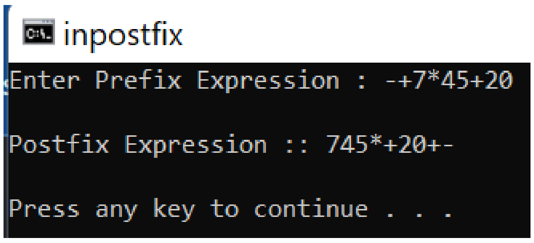
```
inpostfix
Enter Prefix Expression : + * A B * C D

Postfix Expression :: AB*CD*+

Press any key to continue . . .
```

Test case :3

Test case :4



```
Enter Prefix Expression : +9*26

Postfix Expression :: 926*+

Press any key to continue . . .
```



```
Enter Prefix Expression : -+7*45+20

Postfix Expression :: 745*+20+-

Press any key to continue . . .
```

PROBLEM FACED: -

1. While coding the problem confused in taking number of arrays.
2. Even deciding the priority return values got confused.
3. In runtime it showing popup message like some error but can be executable.
4. Issues in compile time due to syntax error more than 3 times in testing level.

CONCLUSION: -

1. This experiment allowed me to think in efficient manner.
2. I was improved my programming skills.
3. Learned how to use stacks with arrays.
4. Completely understood the concept of stacks.
5. Learned the mechanics of infix and postfix expression.
6. Learned how to convert infix to postfix manually.

Lab Experiment number: - 02

Experiment Topic: - Evaluation of expressions (stacks)

AIM: - Implementation of C language code to execute the program to evaluation of expressions by using arrays concept and with the help of stacks.

OBJECTIVES: - By the end of this assignment, we will be able to

1. write efficient algorithm to a specific problem.
2. Use arrays in data structures.
3. Concept of stacks and evaluation of expression problems and results after solving expressions.
4. Framing prefix expressions and postfix expressions and evaluation of expression manually.

CONDITIONS: -

Conditions to follow while writing code for experiment 02: -

1. Arrays should be named different.
2. Code should take input postfix expression.
3. Array should take input alphanumeric (digits and characters).
4. Maintain less runtime.
5. Operands Priority management

ALGORITHM: -

```
Step 1: -start
Step 2: -input expression
Step 3: -initialize top= -1
Step 3: -functions to operate stacks
Step 4: -check the scanned   array not to be empty
Step 5: - to check if the passed character is a digit or not
Step 5: -conditions to evaluation
Step 6: -add all the pop() elements
Step 7: -print sum of popped elements
Step 8: -end
```

PROGRAM: -

File name: -preevalue.c

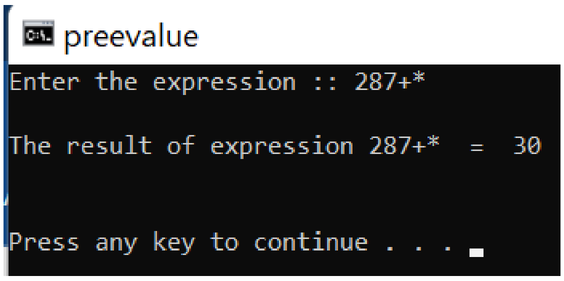
Google drive link: - https://drive.google.com/file/d/1iikfJfbvVD-vwhQVByuQi24fjYXn90bx/view?usp=sharing

Implementation: -

```
#include<stdio.h>
int stack1[20];
int top = -1;
void push(int x) // function to insert element into the stack
{
    stack1[++top] = x;
}
int pop()////function to pop the element from stack
{
    return stack1[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)) //checks entered elements are numbers or digits or characters
        {
            num = *e - 48;  // ascii code transformation
            push(num);
        }
        else
        {
            n1 = pop();
            n2 = pop();
            switch(*e) // operands declaration
            {
            case '+':
            {
                n3 = n1 + n2;
                break;
            }
            case '-':
            {
                n3 = n2 - n1;
                break;
            }
            case '*':
            {
                n3 = n1 * n2;
                break;
            }
            case '/':
            {
                n3 = n2 / n1;
                break;
            }
            }
            push(n3);
        }
        e++;
    }
    printf("\nThe result of expression %s  =  %d\n\n",exp,pop()); //prints output sum
    return 0;
}
```

<u>OUTPUT</u>: -

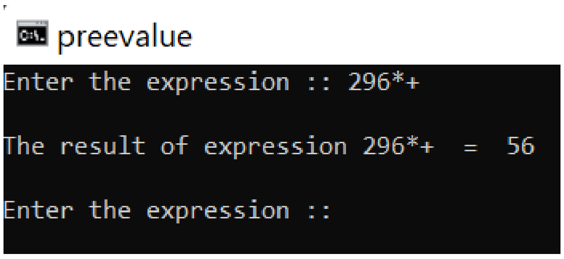
Test case 1: -



preevalue

```
Enter the expression :: 287+*

The result of expression 287+*  =  30


Press any key to continue . . .
```

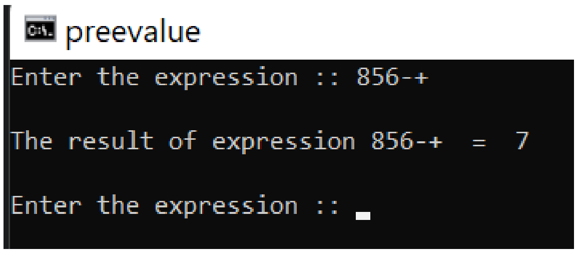
Test case 2: -



preevalue

```
Enter the expression :: 296*+

The result of expression 296*+  =  56

Enter the expression ::
```

Test case 3: -



preevalue

```
Enter the expression :: 856-+

The result of expression 856-+  =  7

Enter the expression ::
```

<u>PROBLEM FACED</u>: -

1. While coding confused in taking number of arrays.
2. Even deciding the priority return values I were got confused.
3. In runtime it showing popup message error but can be executable.
4. Issues in compile time due to syntax error more than 8 times in testing level.

<u>CONCLUSION</u>: -

1. This experiment allowed me to think in efficient manner.
2. I improved my programming skills.
3. Learned using stacks with arrays.
4. Completely understood the concept of stacks in evaluation of expression values.
5. Learned the mechanics of infix and postfix expression and evaluation of expression.
6. Learned how to evaluate the expressions manually.

Lab Experiment number: - 03A

Experiment Topic: - Implementation the following operations i.e., enqueue, dequeue and finding an element in the following cases:

a. Linear Queue using arrays or

b. Circular queue arrays

Experiment 03A on linear queue using arrays

AIM: - Implementation of c code to perform operations on queue data structure like inserting elements to queue and deleting the elements from queue and checking the first element in queue.

OBJECTIVES: - By the end of this, we will be able to know how

5. write efficient algorithm to linear queue operations.
6. Use arrays in queue data structures.
7. Concept of queue and its operations performs in a code.
8. Improves coding experience.

CONDITIONS: -

Conditions to follow while writing code for experiment 03A: -

6. Arrays should be named different.
7. Two pointers must be used.
8. Basic mathematical tricks should be used.
9. Memory management should be done.
10. All conditions of queue should be satisfied by this experiment.
11. Initialization of function should be done in header file. (choice)

ALGORITHM: -

Algorithm for main function :

Step 1: start

Step 2: initialize Maximum elements

Step 3: initialization of queue array and front=0 and rear =-1

Step 4: include header file “queue.h” (user defined)

Step 5: input elements to insert in queue

Step 6: delete elements in queue

Step 7: prints elements which are deleted

Step 8: stop

HEADER FILE(queue.h) FUNCTIONS ALGORITHM

Algorithm for insert (int data) function: -

```
Step 1: start
Step 2: check if rear pointer is equal to one less than MAXIMUM elements if yes
Step 3: prints queue is full
Step 4: else increment rear pointer by one and assignment data variable
Step 5: stop
```

Algorithm for delete() function:

```
Step 1: start
Step 2: check if rear is equal to -1 print queue is empty
Step 3: else print the front element of the queue array
Step 4: initialize front element of queue array to data element
Step 5: increment front pointer by one
Step 6: check if front pointer is greater than rear pointer if yes then reinitialize both pointers to -1
Step 6: return data element
Step 7: stop
```

Algorithm for first( ) function:(checks the first element of queue and prints it in output)

```
Step 1: start
Step 2: initialize data variable to front pointed element in queue array
Step 3: prints data variable
Step 4: stop
```

Algorithm for prints(int arr[]) function :

```
Step 1: check if rear pointer is -1 if yes print stack is empty
Step 2: else print array elements by increment loop with condition i=front and i<rear
Step 3: stop
```

PROGRAM: -

Folder name: queueoperations

C File name: -queueoperations.c

Header file name: queue.h

Google drive link: https://drive.google.com/drive/folders/1RGpikQkaPbyxsKOiaHoFsJC4fxouCXda?usp=sharing

Implementation: -

Queueoperations.c

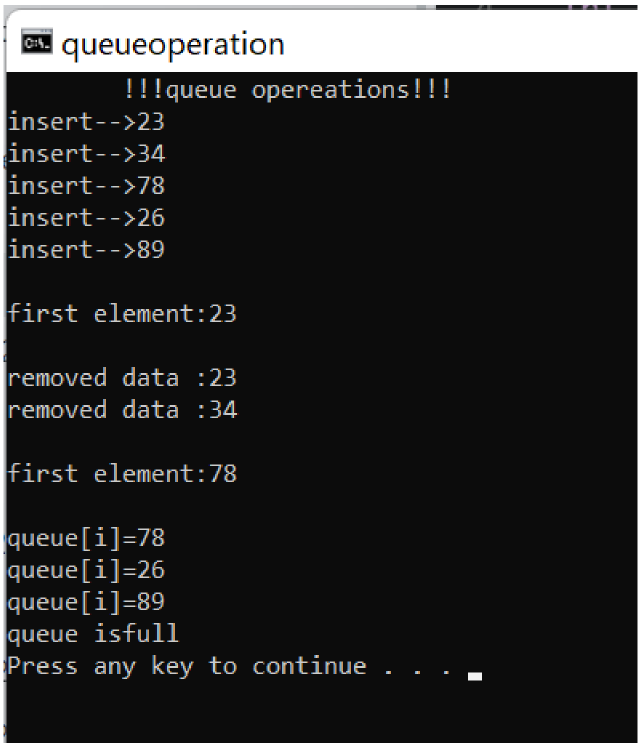
```
#include<stdio.h>
#define MAX 5
int queue[MAX];
int front=0;
int rear=-1;
#include"queue.h"
void main()
{
  printf("\t!!!queue opereations!!!");
  insert(23);
  insert(34);
  insert(78);
  insert(26);
  insert(89);
  printf("\n");
  first();
  printf("\n");
  delete();
  delete();
  printf("\n");
  first();
  printf("\n");
  prints();
  insert(89);
}
```

Header file:

Queue.h:

```
void insert(int data)
{
  if(rear==MAX-1)  //condition for checing queue is full or not
    printf("\nqueue isfull ");
  else
  {
    queue[++rear]=data; //post increment rear ptr and assigning to data
   printf("\ninsert-->%d",data);
  }
}
void delete()
{
  if(rear==-1) //checking queue is empty or !
  {
      printf("\nqueue is empty");
    }
      else
      {
        int data;
        printf("\nremoved data :%d",queue[front]);
          queue[front]=data; //first element in the array is assigned to data
          front++;
        if(front>rear)
          front=rear=-1;//reinitializing  2 ptr to -1
  }
}
void first()
{
  int data=queue[front];//printing first element in tne aray
  printf("\nfirst element:%d",data);
}
void prints()
{
  if(rear==-1)
  printf("\nstack is empty");
  else
  {
    int  i;
    for(i=front;i<=rear;i++){
  printf("\nqueue[i]=%d",queue[i]);
    }
  }
}
```

OUTPUT:
test case 1:



```
queueoperation
        !!!queue opereations!!!
insert-->23
insert-->34
insert-->78
insert-->26
insert-->89

first element:23

removed data :23
removed data :34

first element:78

queue[i]=78
queue[i]=26
queue[i]=89
queue isfull
Press any key to continue . . .
```

Input inserting elements into the program and deleting entire queue and showing empty queue program.

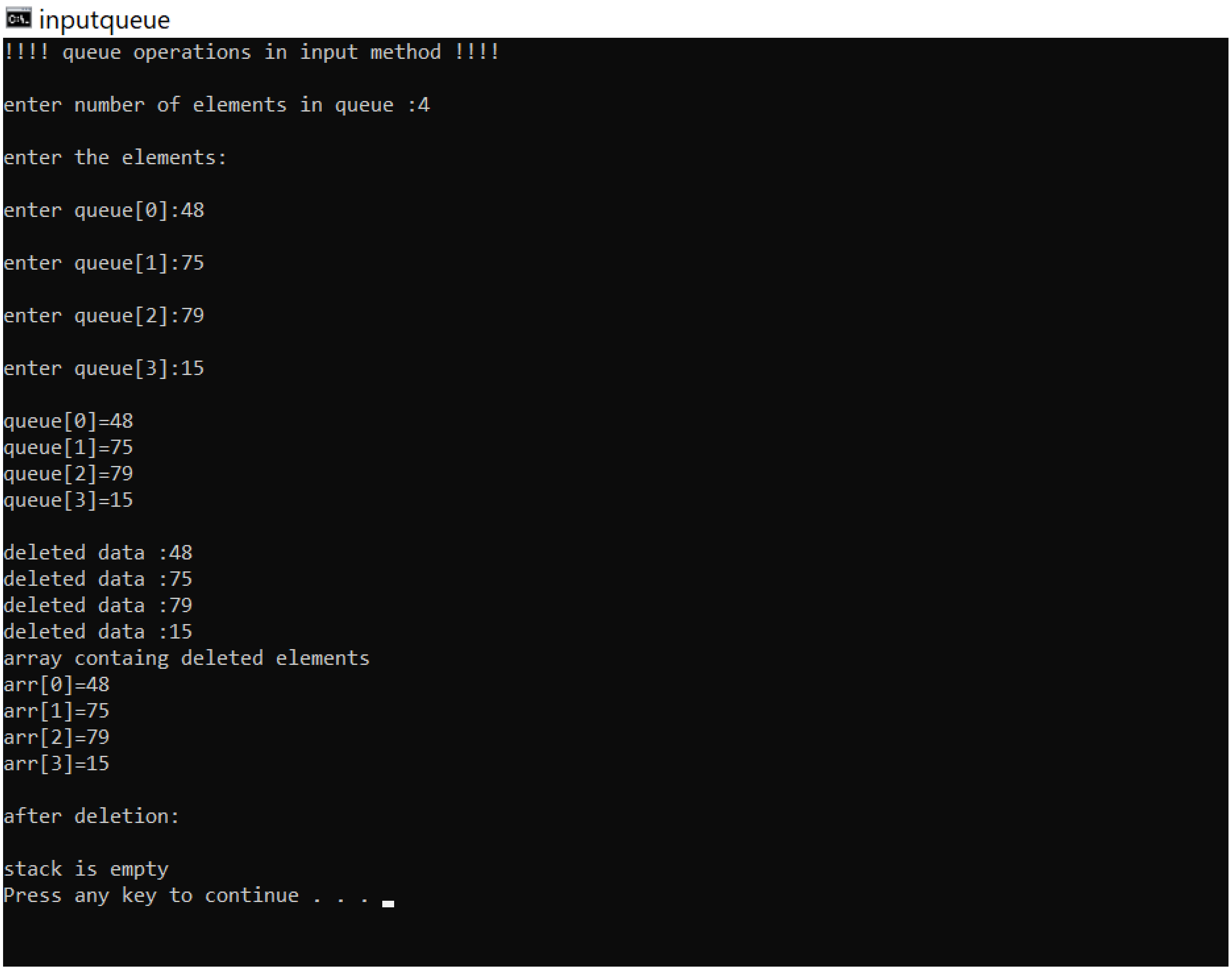
Drive link: - https://drive.google.com/file/d/1XDx9OgvozwN-1jsL8iFQ7exzdE_uKGJb/view?usp=sharing

Inputqueue.c

```
#include<stdio.h>
int queue[12];
int front=0;
int rear=-1;
int MAX;
int arr[12];
#include"queue.h"
int main()
{
  printf("!!!! queue operations in input method !!!!\n\n");
  printf("enter number of elements in queue :");
  scanf("%d",&MAX);
  printf("\nenter the elements:\n");
 for(int i=0;i<MAX;i++)
 {
   int data;
   printf("\nenter queue[%d]:",i);
   scanf("%d",&data);
   insert(data);
 }
 prints(queue);
printf("\n");
 for(int i=0;i<MAX;i++)
 {
  arr[i]=queue[front];
  delete();
 }
 printf("\narray containg deleted elements");
 for(int i=0;i<MAX;i++)
 {
   printf("\narr[%d]=%d",i,arr[i]);}
 printf("\n\nafter deletion:\n");
 prints(queue);
}
```

OUTPUT:

Test case 01:



```
inputqueue
!!!! queue operations in input method !!!!

enter number of elements in queue :4

enter the elements:

enter queue[0]:48

enter queue[1]:75

enter queue[2]:79

enter queue[3]:15

queue[0]=48
queue[1]=75
queue[2]=79
queue[3]=15

deleted data :48
deleted data :75
deleted data :79
deleted data :15
array containg deleted elements
arr[0]=48
arr[1]=75
arr[2]=79
arr[3]=15

after deletion:

stack is empty
Press any key to continue . . .
```

PROBLEM FACED: -

1. While writing algorithm I faced some problem.
2. While compile time few issues with function word terminology.
3. While runtime I faced issue in minimizing the run time.
4. Failed in dynamic memory allocation.
5. Failed in collecting deleting elements to another array.

CONCLUSION: -

1. This experiment helped me to know where my skills are.
2. Learned queue concept and using of it.
3. Learned writing code with good indentation.
4. Learned implementation of header files and learned how to store queues operations in header file.
5. This experiment helped me to improve my minimum coding speed.

Lab Experiment number: - 03 B

Experiment Topic: - Implementation the following operations i.e., enqueue, dequeue and finding an element in the following cases:

a. Linear Queue using arrays or

b. Circular queue arrays

Experiment 03B on circular queue using arrays

<u>AIM</u>: - Implementation of c code to perform operations on circular queue data structure like inserting elements to queue and deleting the elements from queue and inserting elements to empty memory locations in queue checking the first element in queue.

<u>OBJECTIVES</u>: - By the end of this, we will be able to know how

9. write efficient algorithm to circular queue operations.
10. Use arrays in circular queue data structures.
11. Concept of queue and its operations performs in a code.
12. Improves coding experience.
13. Memory management in queue data structure.

<u>CONDITIONS</u>: -

Conditions to follow while writing code for experiment 03A: -

12. Arrays should be named different.
13. Two pointers must be used.
14. Basic mathematical tricks should be used.
15. Memory management should be done.
16. Reusability of empty memory locations present in queue.
17. All conditions of circular queue should be satisfied by this experiment.
18. Initialization of function should be done in header file. (choice)

<u>ALGORITHM</u>: -

Algorithm for insert function (used to insert element in circular queue): -

```
Step 1: -start
Step 2: -check queue is full or not (front==0 and rear==MAX-1)
Step 3: -if queue is empty then check (front==-1)
Step 4: -if true then reinitialize front==rear==←0
Step 5: -else if check condition (rear==MAX-1)
Step 6: -if yes assign rear=0 or no then assign rear←rear+1
Step 7: -assign array front element to temp variable (data)(user input)
Step 8: -stop
```

Algorithm for delete function (delete function helps to delete element from the queue): -

Step 1: -start

Step 2: -check if front==-1 then queue is empty

Step 3: -check condition front==rear if yes then initialize front==rear🡐-1

Step 4: -check if front🡐MAX-1

Step 5: -front=0 (reinitializing)

Step 6: -else front🡐front+1

Step 7: -stop

Algorithm for print function (print() function helps to find the elements in the queue): -

Step 1: -start loop with i=0 and I <MAX and increment loop

Step 2: -print the queue elements

Step 3: -stop

<u>PROGRAM</u>: -

Folder name: circular queue

C File name: -circularqueue.c

Header file name: none

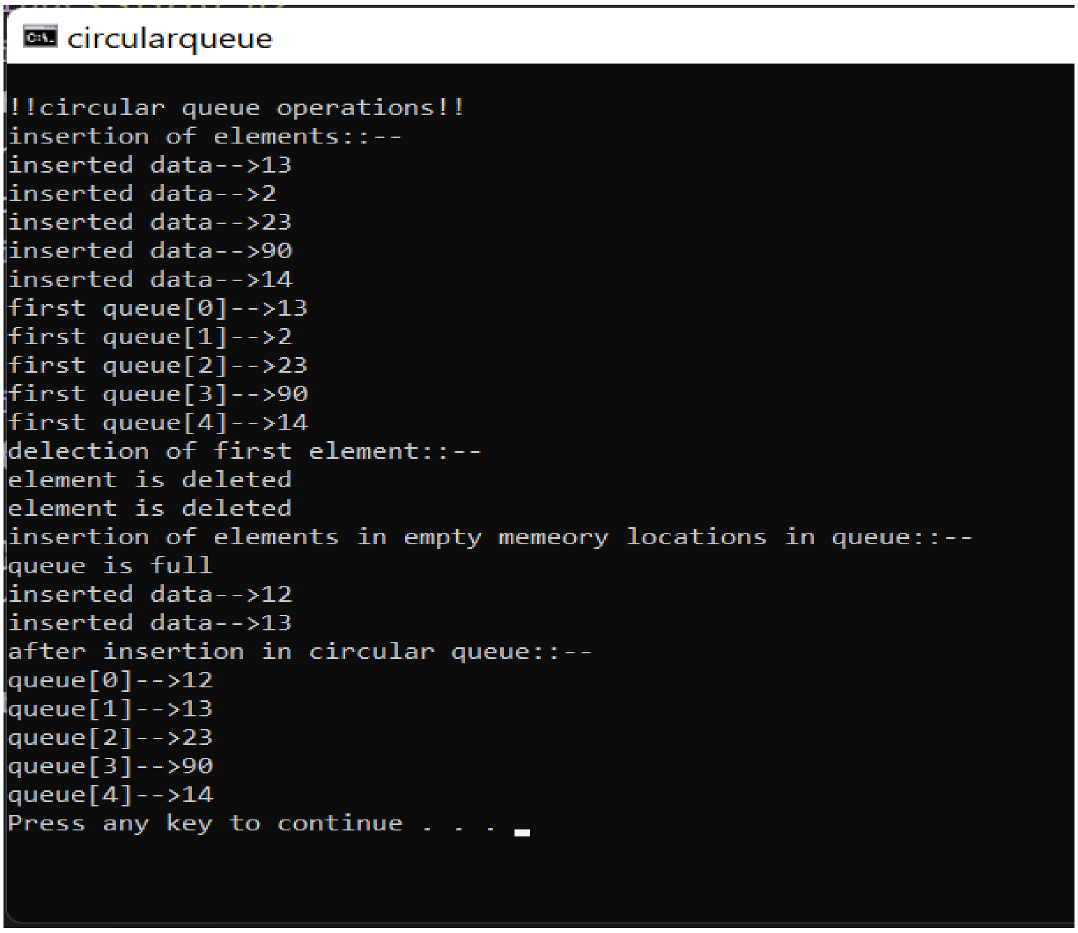
Google drive link: https://drive.google.com/file/d/19fXnrvcy2CADWH02PzE0cGgaOrIG-mHD/view?usp=sharing

Implementation: -

```
#include<stdio.h>
#define MAX 5
int front=-1;
int rear=-1;
int queue[MAX];
int insert(int data)
{

    if(front==0 && rear==MAX-1) //checking queue is full or not
    {
        printf("\nqueue is full");
    }
if(front == -1) //checking front ptr pointed to -1  is yes then intialising to 0
{
front = 0;
rear = 0;
}
else
{
if(rear == MAX-1)//checking is queue rear pointer is equal to maximum size of array
rear = 0;
else
rear = rear+1;   //rear++ increment
}
queue[rear] =data;  //assign data variable to queue front element array
printf("\ninserted data-->%d",data);
}
int delete()
{
     printf("\nelement is deleted");
    if(front==-1)  //checking queue is empty
    {
        printf("\nqueue is empty");
    }
    if(front==rear)  //checking queue front and rear pointer pointing to same memory
location
    {
        front==rear==-1;//reintizilsing ptr to -1
    }
    if(front=MAX-1)  // checking front ptr is pointing to MAX-1 location
    {
        front=0;
    }
    else
    {
        front=front+1;  //increment front ptr by 1

    }
}
void print()
{
        for(int i=0;i<MAX;i++) //loop conditions
    {
       printf("\nqueue[%d]-->%d",i,queue[i]); //printing elements in queue
    }
}
int main()
{
    printf("\n!!circular queue operations!!");
    printf("\ninsertion of elements::--");
    insert(13);
    insert(2); //insering
    insert(23);
    insert(90);
    insert(14);
    for(int i=0;i<MAX;i++)
    {
       printf("\nfirst queue[%d]-->%d",i,queue[i]);
    }
    printf("\ndelection of first element::--");
    delete();
    delete();  //deleting
    printf("\ninsertion of elements in empty memeory locations in queue::--");
    insert(12);
    insert(13);
    printf("\nafter insertion in circular queue::--");
    print();//printing
    return 0;
}
```

OUTPUT:

Test case 01:


circularqueue

```
!!circular queue operations!!
insertion of elements::--
inserted data-->13
inserted data-->2
inserted data-->23
inserted data-->90
inserted data-->14
first queue[0]-->13
first queue[1]-->2
first queue[2]-->23
first queue[3]-->90
first queue[4]-->14
delection of first element::--
element is deleted
element is deleted
insertion of elements in empty memeory locations in queue::--
queue is full
inserted data-->12
inserted data-->13
after insertion in circular queue::--
queue[0]-->12
queue[1]-->13
queue[2]-->23
queue[3]-->90
queue[4]-->14
Press any key to continue . . .
```

<u>PROBLEM FACED</u>: -

1. While writing algorithm I faced some problem.
2. While compile time few issues with function word terminology.
3. While runtime I faced issue in minimizing the run time.
4. Failed in collecting deleting elements to another array.
5. Failed in displaying deleted element.
6. Problems with indentation.

<u>CONCLUSION</u>: -

1. This experiment helped me to know where my skills are.
2. Learned circular queue concept and using of it.
3. Learned writing code with good indentation.
4. Learned implementation of conditions for if cases properly.
5. This experiment helped me to improve my minimum coding speed.
6. Learned managing memory reusability and management of memory.

Lab Experiment number: - 04

Experiment Topic: **- Create a singly linked list and perform the following operations:**

a. Add an element at the end of the list

b. Delete an element from the beginning of the list

c. Find the middle element of the list

<u>AIM</u>: - Implementation of c code to perform operations on singly linked list data structure like Adding an element at the end of the list, deleting an element from the beginning of the list Finding the middle element of the list Search the given key form the list

<u>OBJECTIVES</u>: - By the end of this, we will be able to know how

1. write efficient algorithm to linked list operations.
2. Use structures in singly linked list.
3. Concept of linked list and its operations performs in a code.
4. Improves coding experience.
5. Memory management in linked list data structure.

<u>CONDITIONS</u>: -

Conditions to follow while writing code for experiment 04: -

19. Pointers should be named different.
20. Two pointers must be used.
21. Basic mathematical tricks should be used.
22. Memory management should be done.
23. Reusability of empty memory locations present in linked list.
24. All conditions of linked list should be satisfied by this experiment.
25. Initialization of function should be done in header file. (choice)

<u>ALGORITHM</u>: -

Algorithm for insertAtbeginning(struct node head_reg, int new_data)**

```
Step 1: - start
Step 2: -allocate memory to a structure variable
Step 3: -new_node→data ← new_data
Step 4: -new_node→next← head_ref
Step 5: -head_ref←new_node
Step 6: -stop
```

Algorithm for deleteNode(struct node head_ref, int key): -**

```
Step 1: -start
Step 2: - allocate memory to structure variable
Step 3: -if temp! =NULL and temp→data==key
Step 4: - prev =temp
Step 5: -temp→next
Step 6: -free space
Step 7: -stop
```

Algorithm for searchNode(struct node head_ref, int key): -**

```
Step 1: -start
Step 2: -check if current is not equal to NULL
Step 3: -if yes then check current→data==key
Step 4: -return 1
Step 5: -current→next
Step 6: - stop
```

Algorithm for LinkedList(struct node head_ref): -**

```
Step 1: -start
Step 2: -allocate memory for struct variable
Step 3: -head_ref==NULL
Step 4: -index =current→next
Step 5: -index!=NULL
Step 6: -temp →current->data
Step 7: -index =index→next
Step 8: - stop
```

<u>PROGRAM</u>: -

Folder name: data structures lab

C File name: -linkedlistexperiment4.c

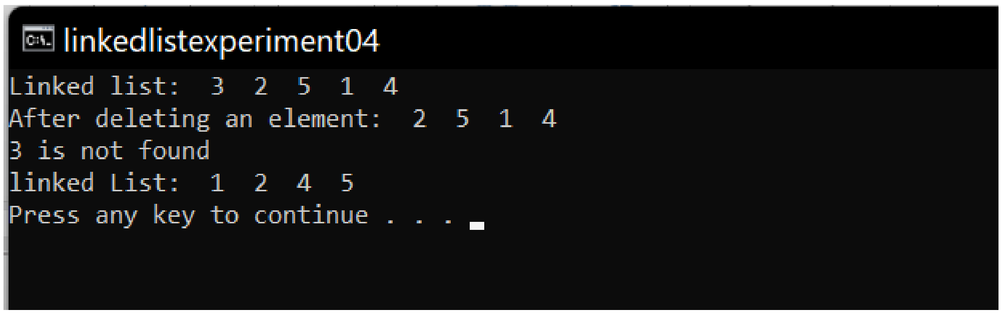
Header file name: none

Google drive link: https://drive.google.com/file/d/1iRaBvBC58B3x5rwcqNUyNX778IwM8gjv/view?usp=sharing

Implementation: -

```
/*
Name: -V D Pandurangasai Guptha
Sec:B
Experiment no: -04<A<B<C<D
topic:Linked list
*/

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

struct node// Create a node
 {
  int data;
  struct node* next;
};


void insertAtBeginning(struct node** head_ref, int new_data) // Insert at the
beginning
{

  struct node* new_node = (struct node*)malloc(sizeof(struct node));// Allocate memory
to a node


  new_node->data = new_data;// insert the data

  new_node->next = (*head_ref);


  (*head_ref) = new_node;// index points to the node next to current
}

// Insert a node after a node
void insertAfter(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));//memory
allocation to pointer
  new_node->data = new_data; //rearangement
  new_node->next = prev_node->next;
  prev_node->next = new_node;
}

// Insert the the end
void insertAtEnd(struct node** head_ref, int new_data) {
  struct node* new_node = (struct node*)malloc(sizeof(struct node));
  struct node* last = *head_ref; /* used in step 5*/

  new_node->data = new_data;
  new_node->next = NULL;

  if (*head_ref == NULL)
  {
  *head_ref = new_node;
  return;
  }

  while (last->next != NULL) last = last->next;

  last->next = new_node;
  return;
}

// Delete a node
void deleteNode(struct node** head_ref, int key)
{
  struct node *temp = *head_ref, *prev;

  if (temp != NULL && temp->data == key)
  {
  *head_ref = temp->next;
  free(temp);
  return;
  }
  // Find the key to be deleted
  while (temp != NULL && temp->data != key)
  {
  prev = temp;
  temp = temp->next;
  }

  // If the key is not present
  if (temp == NULL) return;

  // Remove the node
  prev->next = temp->next;

  free(temp);
}

// Search a node
int searchNode(struct node** head_ref, int key)
 {
  struct node* current = *head_ref;

  while (current != NULL)
   {
  if (current->data == key) return 1;
  current = current->next;
  }
  return 0;
}

// Sort the linked list
void LinkedList(struct node** head_ref)
 {
  struct node *current = *head_ref, *index = NULL;
  int temp;

  if (head_ref == NULL)
  {
  return;
  } else {
  while (current != NULL)// index points to the node next to current
   {
    index = current->next;

    while (index != NULL)
    {
    if (current->data > index->data)
    {
      temp = current->data;
      current->data = index->data;
      index->data = temp;
    }
    index = index->next;
    }
    current = current->next;
  }
  }
}

// Print the linked list
void printList(struct node* node)
{
  while (node != NULL)
  {
  printf(" %d ", node->data);
  node = node->next;
  }
}

// Driver program
int main()
{
  struct node* head = NULL;

  insertAtEnd(&head, 1);
  insertAtBeginning(&head, 2);
  insertAtBeginning(&head, 3);
  insertAtEnd(&head, 4);
  insertAfter(head->next, 5);

  printf("Linked list: ");
  printList(head);
  printf("\nAfter deleting an element: ");
  deleteNode(&head, 3);

  printList(head);

  int item_to_find = 3;
  if (searchNode(&head, item_to_find))
  {
  printf("\n%d is found", item_to_find);
  } else
  {
  printf("\n%d is not found", item_to_find);
  }

  LinkedList(&head);
  printf("\nlinked List: ");
  printList(head);
}
```

Output: -



linkedlistexperiment04

```
Linked list:  3  2  5  1  4
After deleting an element:  2  5  1  4
3 is not found
linked List:  1  2  4  5
Press any key to continue . . .
```

<u>PROBLEM FACED</u>: -

1. While writing algorithm I faced some problem.
2. While compile time few issues with function word terminology.
3. While runtime I faced issue in minimizing the run time.
4. Confused while creating pointers.
5. Problems with indentation.

<u>CONCLUSION</u>: -

1. This experiment helped me to know where my skills are.
2. Learned linked list concept and using of it.
3. Learned writing code with good indentation.
4. Learned implementation of conditions for a while and for loops properly.
5. This experiment helped me to improve my minimum coding speed.
6. Learned managing memory reusability and management of memory.
7. Learned how to use pointers efficiently.

Lab Experiment number: - 04B

Experiment Topic: -

e. Priority queue singly linked list.

f. Polynomial addition using linked list

g. Sparse matrix operations using linked list

AIM: - Implementation of c code to perform operations on priority queue singly linked list and data structure like Adding an element at the end of the list, deleting an element from the beginning of the list Finding the middle element of the list Search the given key form the list and experiment on polynomial addition used linked list and operations on sparse matrix operations using linked list.

OBJECTIVES: - By the end of this, we will be able to know how

1. write efficient algorithm to priority and polynomial and sparse matric linked list operations.
2. Use structures in priority and polynomial and sparse matrix singly linked list.
3. Concept of linked list and its operations performs in a code.
4. Improves coding experience.
5. Memory management in linked list data structure.

CONDITIONS: -

Conditions to follow while writing code for experiment 04**B**: -

26. Pointers should be named different.
27. Two pointers must be used.
28. Basic mathematical tricks should be used.
29. Management on [polynomial and sparse matrix concepts.
30. Memory management should be done.
31. Reusability of empty memory locations present in linked list.
32. All conditions of linked list should be satisfied by this experiment.
33. Initialization of function should be done in header file. (choice)

Priority queue singly linked list program: -

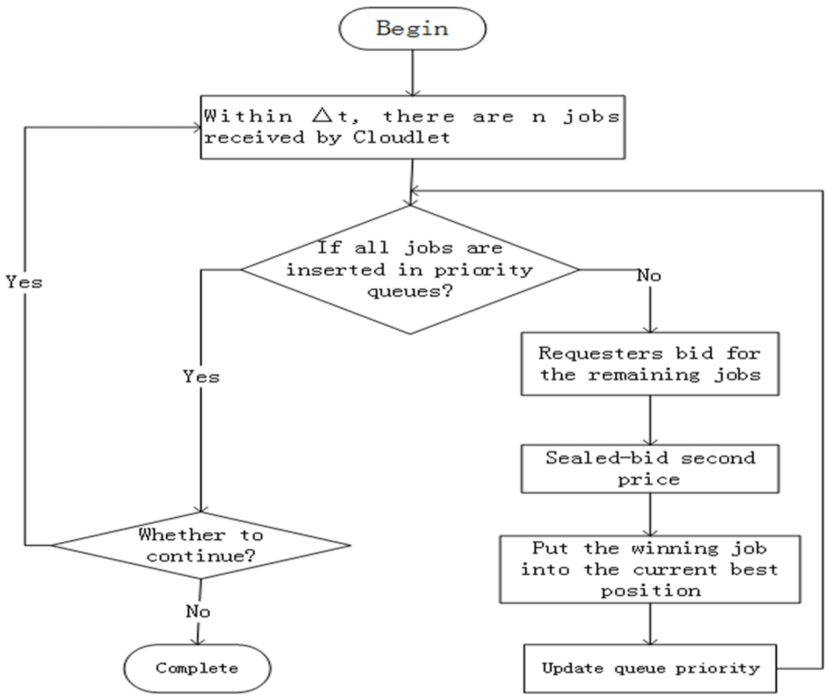
Folder name: data structures lab

C File name: -experiment.E.c

Header file name:

Google drive link: -https://drive.google.com/file/d/1X7umA6BLymS0pCvVyNAdYih6rLwS0p_C/view?usp=sharing

Flow chart: -

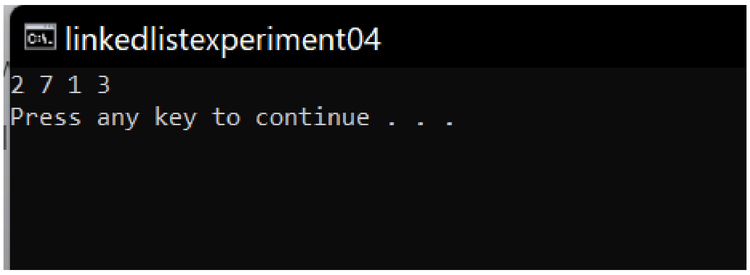

Begin
Within △t, there are n jobs received by Cloudlet
If all jobs are inserted in priority queues?
No
Yes
Requesters bid for the remaining jobs
Sealed-bid second price
Put the winning job into the current best position
Update queue priority
Yes
Whether to continue?
No
Complete

Implementation: -

```
/*
     Name: - V D Panduranga Sai Guptha
     sec: -  B
     assignment: -04B
     Experiment topic: - Priority queue singly linked list.
*/
#include <stdio.h>
#include <stdlib.h>// header file
// priority Node
typedef struct node {
   int data;
   int priority;  //priority file
   struct node* next;
} Node;
Node* newNode(int d, int p) {  //function to link new noder o the priority linked list
   Node* temp = (Node*)malloc(sizeof(Node));// Temp node
   temp->data = d;
   temp->priority = p; //temp node family members
   temp->next = NULL;
   return temp;  //returningthe temp node tho the function
}
int peek(Node** head) { //function to check the first element in the priority linked list
   return (*head)->data;
}
void pop(Node** head) {
   Node* temp = *head;
   (*head) = (*head)->next;  //condition for making the priority linked list
   free(temp);  //free the the memory in the heap location
}
void push(Node** head, int d, int p) { //function to push the element into the linke list
   Node* start = (*head);
   Node* temp = newNode(d, p);
   if ((*head)->priority > p) {// condtion to add rhe element to the singly priority
linked list
```

```
      temp->next = *head;
      (*head) = temp;
   } else {
      while (start->next != NULL &&
      start->next->priority < p) {
         start = start->next;
      }
      // Either at the ends of the list
      // or at required position
      temp->next = start->next;
      start->next = temp;
   }
}
// Function to check the queue is empty
int isEmpty(Node** head) {
   return (*head) == NULL;
}
// main function
int main() {
   Node* pq = newNode(7, 1);
   push(&pq, 1, 2);
   push(&pq, 3, 3);  //calling thre function
   push(&pq, 2, 0);   //callintg the function
   while (!isEmpty(&pq)) {
      printf("%d ", peek(&pq));
      pop(&pq);
   }
   return 0; //returning program to 0
}
```

Output: -



```
2 7 1 3
Press any key to continue . . .
```

<u>PROBLEM FACED</u>: -

1. While writing algorithm I faced some problem.
2. While compile time few issues with function word terminology.
3. While runtime I faced issue in minimizing the run time.
4. Confused while creating pointers.
5. Fee the allocated memory perfectly.
6. Problems with indentation.

<u>CONCLUSION</u>: -

1. This experiment helped me to know where my skills are.
2. Learned priority linked list concept and using of it.
3. Learned writing code with good indentation.
4. Learned implementation of conditions for a while and for loops properly.
5. This experiment helped me to improve my minimum coding speed.
6. Learned managing memory reusability and management of memory.
7. Learned how to use pointers efficiently.

Polynomial addition using linked list experiment: -

Step 1: loop around all values of linked list and follow step 2& 3.

Step 2: if the value of a node's exponent. is greater copy this node to result node and head towards the next node.

Step 3: if the values of both node's exponent is same adding the coefficients and then copy the added value with node to the result.

Step 4: Print the resultant node.

Implementation: -

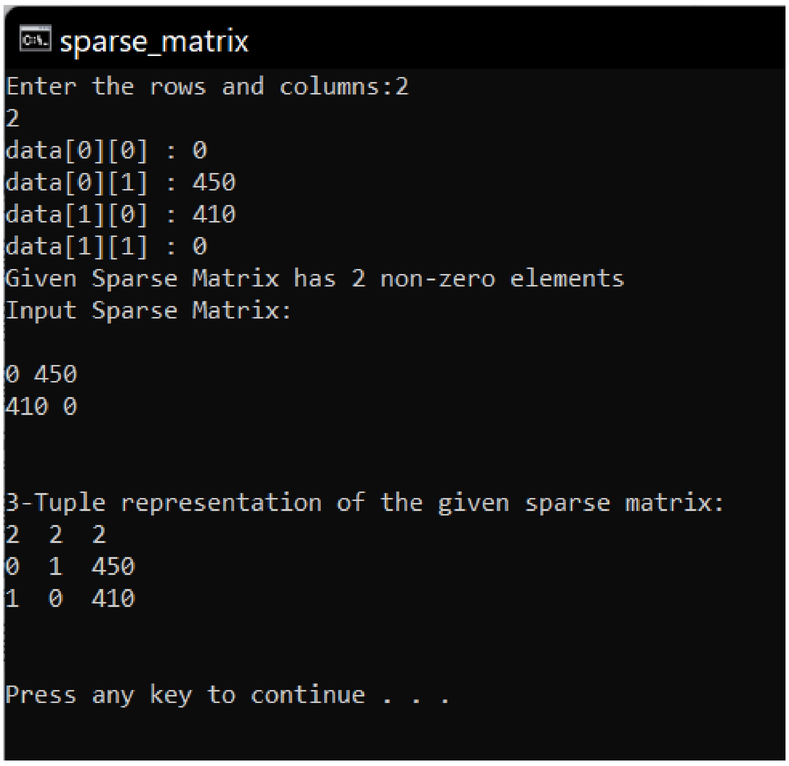
```
#include <stdio.h>
  #include <stdlib.h>

  /* structure to store data */
  struct node {
        int row, col, val;
        struct node *right, *down;
  };

  /* structure of column head */
  struct chead {
        int col;
        struct chead *next;
        struct node *down;
  };

  /* structure of row head */
  struct rhead {
        int row;
        struct rhead *next;
        struct node *right;
  };

  /* structure of additional head */
  struct sparsehead {
        int rowCount, colCount;
```

```
      struct rhead *frow;
      struct chead *fcol;
};

/* main node */
struct sparse {
      int row, *data;
      struct node *nodePtr;
      struct sparsehead *smatrix;
      struct rhead **rowPtr;
      struct chead **colPtr;
};

int count = 0;

/* Establish row and column links */
void initialize(struct sparse *sPtr, int row, int col) {
      int i;
      sPtr->rowPtr = (struct rhead **) calloc(1, (sizeof (struct rhead) * row));
      sPtr->colPtr = (struct chead **) calloc(1, (sizeof (struct chead) * col));
      for (i = 0; i < row; i++)
              sPtr->rowPtr[i] = (struct rhead *) calloc(1, sizeof (struct rhead));

      for (i = 0; i < row - 1; i++) {
              sPtr->rowPtr[i]->row = i;
              sPtr->rowPtr[i]->next = sPtr->rowPtr[i + 1];
      }

      for (i = 0; i < col; i++)
              sPtr->colPtr[i] = (struct chead *) calloc(1, sizeof (struct chead));

      for (i = 0; i < col - 1; i++) {
              sPtr->colPtr[i]->col = i;
              sPtr->colPtr[i]->next = sPtr->colPtr[i + 1];
      }
```

```
        /* update additional head information  */
        sPtr->smatrix = (struct sparsehead *) calloc(1, sizeof (struct sparsehead));
        sPtr->smatrix->rowCount = row;
        sPtr->smatrix->colCount = col;
        sPtr->smatrix->frow = sPtr->rowPtr[0];
        sPtr->smatrix->fcol = sPtr->colPtr[0];
        return;
}

 /* input sparse matrix */
void inputMatrix(struct sparse *sPtr, int row, int col) {
        int i, n, x = 0, y = 0;
        n = row * col;
        sPtr->data = (int *) malloc(sizeof (int) * n);
        for (i = 0; i < n; i++) {
                if (y != 0 && y % col == 0) {
                        x++;
                        y = 0;
                }
                printf("data[%d][%d] : ", x, y);
                scanf("%d", &(sPtr->data[i]));
                if (sPtr->data[i])
                        count++;
                y++;
        }
        return;
}

/* display sparse matrix */
void displayInputMatrix(struct sparse s, int row, int col) {
        int i;
        for (i = 0; i < row * col; i++) {
                if (i % col == 0)
                        printf("\n");
                printf("%d ", s.data[i]);
        }
        printf("\n");
        return;
}

/* create 3-tuple array from input sparse matrix */
void createThreeTuple(struct sparse *sPtr, struct sparse s, int row, int col) {
        int i, j = 0, x = 0, y = 0, l = 0;
        sPtr->row = count;
        sPtr->data = (int *) malloc(sizeof (int) * (sPtr->row * 3));

        for (i = 0; i < row * col; i++) {
                if (y % col == 0 && y != 0) {
                        x++;
                        y = 0;
                }
                if (s.data[i] != 0) {
                        sPtr->data[l++] = x;
                        sPtr->data[l++] = y;
                        sPtr->data[l++] = s.data[i];
                }
                y++;
        }
        return;
}

/* insert element to the list */
void insert(struct sparse *sPtr, int row, int col, int val) {
        struct rhead *rPtr;
        struct chead *cPtr;
        struct node *n1, *n2;
        struct sparsehead *smat = sPtr->smatrix;
        int i, j;
        /* update node values */
        sPtr->nodePtr = (struct node *) malloc(sizeof (struct node));
        sPtr->nodePtr->row = row;
        sPtr->nodePtr->col = col;
        sPtr->nodePtr->val = val;

        /* get the row headnode */
        rPtr = smat->frow;

        /* move to corresponding row */
        for (i = 0; i < row; i++)
                rPtr = rPtr->next;

        /* traverse the nodes in current and locate new node */
        n1 = rPtr->right;
        if (!n1) {
                rPtr->right = sPtr->nodePtr;
                sPtr->nodePtr->right = NULL;
        } else {
                while (n1 && n1->col < col) {
                        n2 = n1;
                        n1 = n1->right;
                }
                n2->right = sPtr->nodePtr;
                sPtr->nodePtr->right = NULL;
        }

        /* get the column head node */
        cPtr = sPtr->smatrix->fcol;

        /* move to corresponding column (1/2/3..) */
        for (i = 0; i < col; i++)
                cPtr = cPtr->next;

        /*
```

```
         * traverse the node in current column and locate
         * new node in appropriate position
         */
        n1 = cPtr->down;

        if (!n1) {
                cPtr->down = sPtr->nodePtr;
                sPtr->nodePtr->down = NULL;
        } else {
                while (n1 && n1->row < row) {
                        n2 = n1;
                        n1 = n1->down;
                }
                n2->down = sPtr->nodePtr;
                sPtr->nodePtr->down = NULL;
        }
        return;
  }

  /* create list for 3-Tuple representation */
  void createList(struct sparse *sPtr) {
        int i, j = 0;
        for (i = 0; i < sPtr->row; i++) {
                insert(sPtr, sPtr->data[j], sPtr->data[j + 1], sPtr->data[j + 2]);
                j = j + 3;
        }
        return;
  }

  /* Display data from linked list of 3-Tuple*/
  void displayList(struct sparse s) {
        struct node *n;
        int row = s.smatrix->rowCount, i;
        for (i = 0; i < row; i++) {
                n = s.rowPtr[i]->right;
                if (n) {
                        while (n->right) {
                                printf("%d  %d  %d\n", n->row, n->col, n->val);
                                n =  n->right;
                        }
                        if (n->row == i) {
                                printf("%d  %d  %d\n", n->row, n->col, n->val);
                        }
                }
        }
        printf("\n");
}

int main() {
        struct sparse input, output;
        int row, col;
        printf("Enter the rows and columns:");
        scanf("%d%d", &row, &col);
        initialize(&input, row, col);
        initialize(&output, row, col);
        inputMatrix(&input, row, col);
        printf("Given Sparse Matrix has %d non-zero elements\n", count);
        printf("Input Sparse Matrix:\n");
        displayInputMatrix(input, row, col);
        printf("\n\n");
        createThreeTuple(&output, input, row, col);
        createList(&output);
        printf("3-Tuple representation of the given sparse matrix:\n");
        printf("%d  %d  %d\n", output.smatrix[0].rowCount,
                output.smatrix[0].colCount, count);
        displayList(output);
        return 0;
}
```

Output: -



```
Enter the rows and columns:2
2
data[0][0] : 0
data[0][1] : 450
data[1][0] : 410
data[1][1] : 0
Given Sparse Matrix has 2 non-zero elements
Input Sparse Matrix:

0 450
410 0


3-Tuple representation of the given sparse matrix:
2  2  2
0  1  450
1  0  410


Press any key to continue . . .
```

PROBLEMS FACED

visualizing and doing this problem is hard but doing many times it was easy .

CONCLUSION

By doing this assignment I got to learn about 1.)Sparse matrix operations using linked list

Lab Experiment number: - 05

Experiment Topic: **- operations on trees**

AIM: - Implementation of C language code to execute the program to run operations on trees data structures

OBJECTIVES: - By the end of this assignment, we will be able to

1. write efficient algorithm to a specific problem.
2. Use dynamic memory allocation.
3. Concept of trees and operations.

CONDITIONS: -

Conditions to follow while writing code for experiment 05: -

1. Pointers should be named different.
2. Code should have compile time inputs.
3. Memory allocation should be done carefully (structure data type).
4. Maintain less runtime.
5. Operations with perfect function name.

ALGORITHM: -

Inorder function algorithm: -

Step1: -Traverse the left subtree call Inorder

Step2: - Visit the root.

Step3: -Traverse the right subtree call Inorder

Preorder function algorithm: -

Step1: - Visit the root.

Step2: - Traverse the left subtree call Preorder

Step3: - Traverse the right subtree call Preorder

Postorder function algorithm: -

Step1: -Traverse the left subtree call Postorder

Step2: -Traverse the right subtree call Postorder

PROGRAM: -

File name: -trees

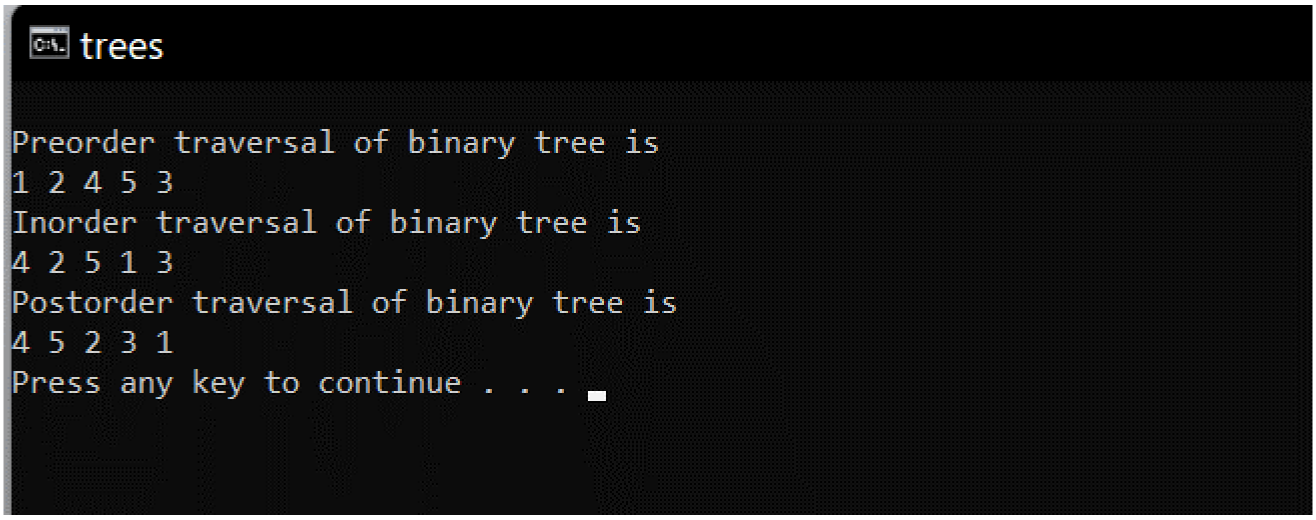
Google drive link: https://drive.google.com/file/d/1mxK92JNQrdUzp8HH-lcXO-sWAaFaosz0/view?usp=sharing

Implementation: -

```
/*
Name: -V D Panduranga Sai Guptha
Sect: -B
Topic: -Implementation of Binary tree traversals techniques:
a. pre-order
b. in-order, and
c. post-order.
*/
#include <stdio.h>
#include <stdlib.h> //standard library
struct node {  //structure to acess the node
    int data;
    struct node* left;
    struct node* right;
};
struct node* newNode(int data)//function to create new node
{
    struct node* node
        = (struct node*)malloc(sizeof(struct node));//memory allocation
    node->data = data; //condition
    node->left = NULL;//condition
    node->right = NULL;//condition

    return (node); //returning node
}
void printPostorder(struct node* node)//function to print the post order of trees
{
    if (node == NULL)
        return;
    printPostorder(node->left);//recurive tool
    printPostorder(node->right);
    printf("%d ", node->data);
}
void printInorder(struct node* node)//function to print inorder of the tree
datastructure
{
    if (node == NULL)
        return;
    printInorder(node->left);
    printf("%d ", node->data);
    printInorder(node->right);//calling the printinorder function
}
void printPreorder(struct node* node)//function to print the preorder of the trees
{
    if (node == NULL)
        return;
    printf("%d ", node->data);
    printPreorder(node->left);
    printPreorder(node->right);//calling the function into the function
}
int main()
```

```
{
    struct node* root = newNode(1);//declaration of ther variable of the type oof the
node
    root->left = newNode(2);//condition
    root->right = newNode(3); //condition
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    printf("\nPreorder traversal of binary tree is \n");
    printPreorder(root);//parameter tof the function
    printf("\nInorder traversal of binary tree is \n");
    printInorder(root);
    printf("\nPostorder traversal of binary tree is \n");
    printPostorder(root);
    return 0;
}
```

<u>Output:</u> -



```
trees

Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
Press any key to continue . . .
```

<u>PROBLEM FACED</u>: -

7. While writing algorithm I faced some problem.
8. While compile time few issues with function word terminology.
9. While runtime I faced issue in minimizing the run time.
10. Failed in displaying deleted element.
11. Problems with indentation.

<u>CONCLUSION</u>: -

7. This experiment helped me to know where my skills are.
8. Learned trees concept and using of it.
9. Learned writing code with good indentation.
10. Learned implementation of conditions for if cases properly.
11. This experiment helped me to improve my minimum coding speed.
12. Learned managing memory reusability and management of memory.

Lab Experiment number: - 06

Experiment Topic: -implementation of AVL tree and its operations

a. Insertion

b. Deletion

c. Traversing

AIM: - Implementation of c code to perform operations on AVL Trees structure like inserting elements to AVL Trees and deleting the elements from Trees and printing the pre_order and post order.

OBJECTIVES: - By the end of this, we will be able to know how

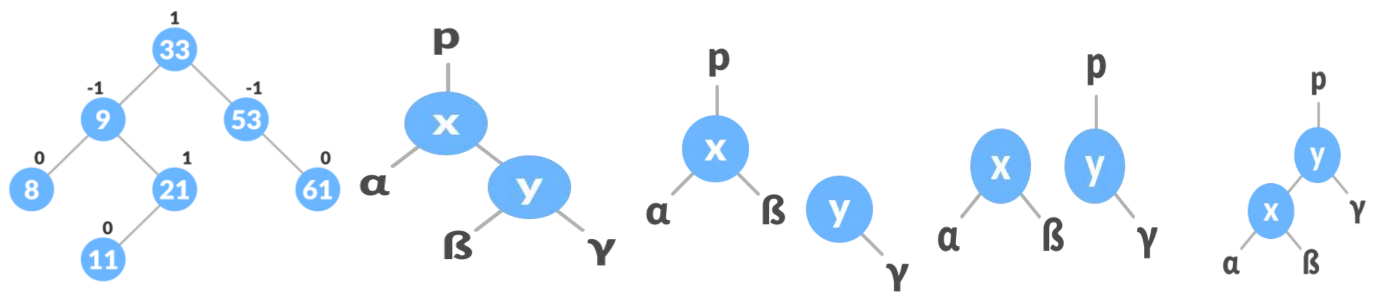
14. write efficient algorithm to linear AVL Trees operations.
15. Use structures and pointers in data structures.
16. Concept of AVL trees and its operations performs in a code.
17. Improves coding experience.

CONDITIONS: -

Conditions to follow while writing code for experiment 03A: -

34. Structures→variables should be named different.
35. Third pointers must be used.
36. Basic mathematical tricks should be used.
37. Memory management should be done.
38. All conditions of AVL trees should be satisfied by this experiment.
39. Initialization of function should be done in header file. (choice)

ALGORITHM: -



PROGRAM: -

C File name: -avl.c

Google drive link: https://drive.google.com/file/d/1uFAU29UBglD6xV__UoOpzcsBT288NEBe/view?usp=sharing

Implementation: -

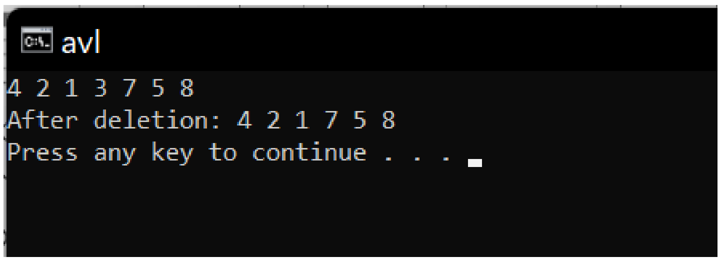
Avl.c

```
/*
   Name:-V D pandurangasai Guptha
   section :- B
   lab_program:-06
*/
#include <stdio.h>
#include <stdlib.h>


struct Node  //  structure node to avl tree
 {
  int key;  //
  struct Node *left;//
  struct Node *right;//  members of the avl tree (parameters)
  int height;//
};

int max(int a, int b);       //unction to find it fmax height of tree
int height(struct Node *N) {
  if (N == NULL)
    return 0;
  return N->height;
}

int max(int a, int b) {
  return (a > b) ? a : b;
}

struct Node *newNode(int key) {  //function to create the new node to the avl tree
  struct Node *node = (struct Node *)
    malloc(sizeof(struct Node));  //dynamic memory allocations
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}

struct Node *rightRotate(struct Node *y) {
  struct Node *x = y->left;
  struct Node *T2 = x->right;

  x->right = y;
  y->left = T2;

  y->height = max(height(y->left), height(y->right)) + 1;
  x->height = max(height(x->left), height(x->right)) + 1;

  return x;
}
```

```
// Left rotate
struct Node *leftRotate(struct Node *x) {
  struct Node *y = x->right;
  struct Node *T2 = y->left;

  y->left = x;
  x->right = T2;

  x->height = max(height(x->left), height(x->right)) + 1;
  y->height = max(height(y->left), height(y->right)) + 1;

  return y;
}

// Get the balance factor
int getBalance(struct Node *N) {
  if (N == NULL)
    return 0;
  return height(N->left) - height(N->right);
}

// Insert node
struct Node *insertNode(struct Node *node, int key) {
  // Find the correct position to insertNode the node and insertNode it
  if (node == NULL)
    return (newNode(key));

  if (key < node->key)
    node->left = insertNode(node->left, key);
  else if (key > node->key)
    node->right = insertNode(node->right, key);
  else
    return node;

  // Update the balance factor of each node and
  // Balance the tree
  node->height = 1 + max(height(node->left),
             height(node->right));

  int balance = getBalance(node);
  if (balance > 1 && key < node->left->key)
    return rightRotate(node);

  if (balance < -1 && key > node->right->key)
    return leftRotate(node);

  if (balance > 1 && key > node->left->key) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
  }

  if (balance < -1 && key < node->right->key) {
    node->right = rightRotate(node->right);
    return leftRotate(node);
  }

  return node;
```

```
}

struct Node *minValueNode(struct Node *node) {
  struct Node *current = node;

  while (current->left != NULL)
    current = current->left;

  return current;
}

// Delete a nodes
struct Node *deleteNode(struct Node *root, int key) {
  // Find the node and delete it
  if (root == NULL)
    return root;

  if (key < root->key)
    root->left = deleteNode(root->left, key);

  else if (key > root->key)
    root->right = deleteNode(root->right, key);

  else {
    if ((root->left == NULL) || (root->right == NULL)) {
      struct Node *temp = root->left ? root->left : root->right;

      if (temp == NULL) {
        temp = root;
        root = NULL;
      } else
        *root = *temp;
      free(temp);
    } else {
      struct Node *temp = minValueNode(root->right);

      root->key = temp->key;

      root->right = deleteNode(root->right, temp->key);
    }
  }

  if (root == NULL)
    return root;

  // Update the balance factor of each node and
  // balance the tree
  root->height = 1 + max(height(root->left),
               height(root->right));

  int balance = getBalance(root);
  if (balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(root);

  if (balance > 1 && getBalance(root->left) < 0) {
    root->left = leftRotate(root->left);
    return rightRotate(root);
  }
  if (balance < -1 && getBalance(root->right) <= 0)
    return leftRotate(root);

  if (balance < -1 && getBalance(root->right) > 0) {
    root->right = rightRotate(root->right);
    return leftRotate(root);
  }

  return root;
}

// Print the tree
void printPreOrder(struct Node *root) {
  if (root != NULL) {
    printf("%d ", root->key);
    printPreOrder(root->left);
    printPreOrder(root->right);
  }
}

int main() {
  struct Node *root = NULL;

  root = insertNode(root, 2);
  root = insertNode(root, 1);
  root = insertNode(root, 7);
  root = insertNode(root, 4);
  root = insertNode(root, 5);
  root = insertNode(root, 3);
  root = insertNode(root, 8);

  printPreOrder(root);

  root = deleteNode(root, 3);

  printf("\nAfter deletion: ");
  printPreOrder(root);

  return 0;
}
```

OUTPUT: -



```
4 2 1 3 7 5 8
After deletion: 4 2 1 7 5 8
Press any key to continue . . .
```

PROBLEM FACED: -

12. While writing algorithm I faced some problem.
13. While compile time few issues with function word terminology.
14. While runtime I faced issue in minimizing the run time.
15. theoretical mistakes.
16. Problems with indentation.

CONCLUSION: -

13. This experiment helped me to know where my skills are.
14. Learned AVL trees concept and using of it.
15. Learned writing code with good indentation.
16. Learned implementation of conditions for if cases properly.
17. This experiment helped me to improve my minimum coding speed.
18. Learned managing memory reusability and management of memory.