Gokaraju Rangaraju Institute of Engineering and Technology

2021-22

DATA STRUCTURE IN C

LAB

KANDUKURI JASWANTH

Bachupally, Survey No. 288, Nizampet Rd, Kukatpally, Hyderabad, Telangana 500090

Table of Contents

**TASK 1 ……………………………………………………………………………….3-14**

a. Implement Bubble sort using a C program. …………………………………………. 3

b. Implement Selection sort using a C program. ………………………………………. 7

c. Implement Insertion Sort using a C program. ………………………………………. 11

**TASK 2 ………………………………………………………………………….... 15-24**

a. Implement Quick sort using a C program. …………………………………………. 15

b. Implement Merge sort using a C program. …………………………………………19

**TASK 3 ……………………………………………………………………………..25-38**

a. Implementation of Stack operations using arrays …………………………………..25

b. Implementation of Queue operations suing arrays. ……………………………….33

**TASK 4………………………………………………………………………………39-48**

a. Write a c program to convert Infix to postfix expression. ……………………….39

b. Write a c program toevaluate a Postfix evaluation ………………………………….44

**TASK 5** Implementation of Circular Queue operations …………………………….49

**TASK6**  Implementation of Single Linked List operations. …………………………57

**TASK 7** Implementation of Circular Linked List operations …………………………68

**TASK 8** Implementation of Double Linked List operations …………………………84

**TASK 9** Implementation of the following operations on BST operations. ……………101

i. Create

ii. insert

iii. search

**TASK 10** Implementation of PreOrder, InOrder and PostOrder in

a BST using recursion. ………………………………………………………107

**TASK 11** Implementation of DFS operations ……………………………………….111

**TASK 12**  Implementation of BFS operations ……………………………………...115

TASK 1:

a. Implement Bubble sort using a C program.

Aim: To implement Bubble sort using a C program.

Description:

This is the simplest sorting technique when compared with all the other sorting techniques. It is also called an exchange sort. In this sorting technique, the adjacent elements are compared and interchanged if necessary.

Process: Compare the first and second elements. If the first element is greater than the second element, then interchange these two elements.

1. Compare the second and third elements. If the second element is greater than the third element then make an interchange.
2. The process is repeated till the last element is reached.
3. When the last element is reached, it is said to be one pass. At the end of the first pass, the largest element is bubbled out. That is it occupies the last position.
4. Steps 1 to 4 are repeated for the elements between 1 to n-1 because the nth element is already sorted.
5. Repeat the above steps for n-1 passes. At the end of the last pass, the entire list is sorted.

Algorithm:

Bubblesort(A,N)

repeat step 2 for i=0 to n-1

repeat for j=0 to n-i-1

If A[j]>A[j+1]

Swap A[j] and A[j+1]

[End of inner loop]

[End of outer loop]

Exit

Example for Bubble sort: 5,1 ,4 ,2 ,8

Pass-1:

Bubble sort starts with the very first two elements, comparing them to check which one is greater.

( 5 1 4 2 8 ) –> ( 1 5 4 2 8 ), Here, algorithm compares the first two elements, and swaps since 5 > 1.

( 1 5 4 2 8 ) –>  ( 1 4 5 2 8 ), Swap since 5 > 4

( 1 4 5 2 8 ) –>  ( 1 4 2 5 8 ), Swap since 5 > 2

( 1 4 2 5 8 ) –> ( 1 4 2 5 8 ), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Pass-2:

Now, during second iteration it should look like this:

( 1 4 2 5 8 ) –> ( 1 4 2 5 8 )

( 1 4 2 5 8 ) –> ( 1 2 4 5 8 ), Swap since 4 > 2

( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )

( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )

Pass-3:

Now, the array is already sorted, but our algorithm does not know if it is completed.

The algorithm needs one whole pass without any swap to know it is sorted.

( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )

( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )

( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )

( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )

Sorted list (after n-1 passes) : 1 2 4 5 8

**Note:** The bubble sort technique has n-1 passes where n is the number of elements

**Time Complexity of Bubble Sort :**

The complexity of the sorting algorithm depends upon the number of comparisons that are

made. Total comparisons in Bubble sort is: n ( n – 1) / 2 ≈ n 2 – n

Best case :O (n2)

Average case :O (n2)

Worst case:O (n2)

Program:

#include<stdio.h>

void selectionsort(int[],int);

int main()

{

int a[10],i,n;

printf("enter the size of array:\n");

scanf("%d",&n);

printf("enter the elements of array:\n");

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

scanf("%d",&a[i]);

printf("\nThe unsorted elements are:\n");

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

printf("%3d",a[i]);

selectionsort(a,n);

printf("\nThe sorted elements are:\n");

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

printf("%3d",a[i]);

}

void selectionsort(int a[],int n)

{

int min,temp,i,j;

for(i=0;i<n-1;i++)

{

min=i;

for(j=i+1;j<n;j++)

{

if(a[min]>a[j])

min=j;

}

if(min!=i)

{

temp=a[min];

a[min]=a[i];

a[i]=temp;

}

}

}

Output:

enter the size of array:

5

enter the elements of array:

5 1 4 2 8

The unsorted elements are:

5 1 4 2 8

The sorted elements are:

1. 2 4 5 8

Result: Thus , in the above program successfully executed without errors using Bubble sort

b. Implement Selection sort using a C program.

Aim: To implement Selection sort using a C program.

Description:

Selection sort is another algorithm that is used for sorting. This sorting algorithm, iterates through the array and finds the smallest number in the array, and swaps it with the first element if it is smaller than the first element. Next, it goes on to the second element and so on until all elements are sorted.

Process:

From the list select the smallest element and interchange it with the first location(0th location)

1. element. Now the first element is sorted.
2. From the elements in positions 2 to n, select the next smallest element and interchange it with the second location (1st location) element. Now the second element is sorted.

Repeat the above steps n-1 times. The entire list gets sorted within pass

Algorithm:

selectionsort(A,N)

repeat step 2 to 5 for i=0 to N-1

set min=i

set j=i

repeat for j=i+1 to N

if A[min] > A[j]

set min=j

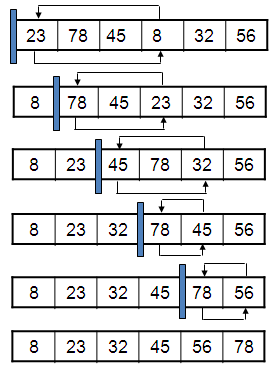
[End inner loop]

swap A[i] and A[min]

[End for loop]

Exit

Ex:- A list of unsorted elements are: 23 78 45 8 32 56



A list of sorted elements now : 8 23 32 45 56 78

Time Complexity of selection sort:

Best case :O(n2) Average case:O(n2) Worst case :O(n2)

Program:

#include <stdio.h>

void selectionsort(int[],int);

int main()

{

int a[10],i,n;

printf("enter the size of array:\n");

scanf("%d",&n);

printf("enter the elements of array:\n");

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

scanf("%d",&a[i]);

printf("\nThe unsorted elements are:\n");

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

printf("%3d",a[i]);

selectionsort(a,n);

printf("\nThe sorted elements are:\n");

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

printf("%3d",a[i]);

}

void selectionsort(int a[],int n)

{

int min,temp,i,j;

for(i=0;i<n-1;i++)

{

min=i;

for(j=i+1;j<n;j++)

{

if(a[min]>a[j])

min=j;

}

if(min!=i)

{

temp=a[min];

a[min]=a[i];

a[i]=temp;

}

}

}

Output:

enter the size of array:

6

enter the elements of array:

23 78 45 8 32 56

The unsorted elements are:

23 78 45 8 32 56

The sorted elements are:

8 23 32 45 56 78

Result: Thus , in the above program successfully executed without errors using Selection

Sort

c. Implement Insertion Sort using a C program.

Aim: To implement Insertion Sort using a C program.

Description:

**Insertion sort** is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Process:

1. Select the second element in the list and compare it with the first element. If the first element is greater than the second element then the second element is inserted at the first location by shifting the first element to the second position. Otherwise, proceed with the next step.
2. Select the third element in the list and compare it with the sorted two elements and insert it at the appropriate position.
3. Select the fourth element and compare it with the previous three sorted elements and insert it in the proper position among the elements which are compared.
4. Repeat the above steps n-1 times. The entire list gets sorted within

pass.

Algorithm:

insertionsort(A,N)

repeat step 2 to 5 for i=1 to N

set index=A[i]

set j=i

repeat while j>0 and A[j-1]>index

Set A[j]=A[j-1]

Set j=j-1

[End while loop]

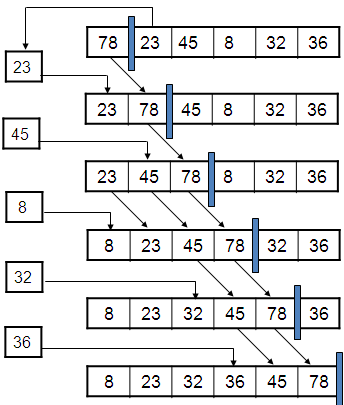
Set a[j]=index

[End of loop]

Exit

Ex:- A list of unsorted elements are: 78 23 45 8 32 36 .

The results of insertion sort for each pass is as follows:-



A list of sorted elements now : 8 23 32 36 45 78

Program:

#include <stdio.h>

void insertionsort(int[],int);

int main()

{

int a[10],i,n;

printf("enter the size of array:\n");

scanf("%d",&n);

printf("enter the elements of array:\n");

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

scanf("%d",&a[i]);

printf("\nThe unsorted elements are:\n");

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

printf("%3d",a[i]);

insertionsort(a,n);

printf("\nThe sorted elements are:\n");

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

printf("%3d",a[i]);

}

void insertionsort(int a[],int n)

{

int i,j,index;

for(i=1;i<n;i++)

{

index=a[i];

j=i;

while((j>0)&&(a[j-1]>index))

{

a[j]=a[j-1];

j--;

}

a[j]=index;

}

}

Output:

enter the size of array:

6

enter the elements of array:

78 23 45 8 32 36

The unsorted elements are:

78 23 45 8 32 36

The sorted elements are:

8 23 32 36 45 78

Result: Thus , in the above program successfully executed without errors using Insertion Sort.

TASK 2

a. Implement Quick sort using a C program.

Aim: To implement Quick sort using a C program.

Description:

Sorting is a way of arranging items systematically. Quick sort is the widely used sorting algorithm that makes **n log n** comparisons in an average case for sorting an array of n elements. It is a faster and highly efficient sorting algorithm. This algorithm follows the divide and conquers approach. Divide and conquer is a technique of breaking down the algorithms into sub-problems, then solving the sub-problems, and combining the results back together to solve the original problem.

Here mainly concentrate on three steps:

1. Divide
2. Conque
3. Combine

i)Divide: Re-arrange the elements and slip the arrays into two sub-arrays each element in the left sub-array is less than or equal to the middle element and each element in the right sub-array is greater than the mid element.

ii) **Conquer:** Recursively, sort two sub-arrays with quick sort.

1. **Combine:** Combine the already sorted array.

**Algorithm:**

**Quicksort(a,low,high)**

**{**

**pivot=a[low];**

**lb=low;**

**ub=high;**

**while(lb do**

**{**

**while(lbpivot)**

**lb=lb+1;**

**while(a[ub]≥pivot)**

**ub:=ub-1;**

**if(lb<ub) then**

**swap(a,low,high)**

**}**

**a[low]=a[ub];**

**a[ub]:=pivot;**

**return ub;**

**}**

Time complexity:

1. Worst case

aT n=1

T(n)=

T(n-1)+n n>1

Now T(n)=T(n-1)+n-------1

Put N=n-1 in equation 1 we get

T(n-1)=T(n-1-1)+n-1

T(n-1)=(n-2)+n-1

Sub T(n-1) in equation1

T(n)=T(n-2)+(n-1)+n

: :

: :

T(n)=T(n-3)+T(n-2)+(n-1)+n

:

:

:

T(n)=

=

T(n)=

O(

Best case and average case:

Now the recurrence relation is

T(n)=O(

a n=0

T(n)

n>1

By using master method

Compare with general term:

T(n)=aT

a=2

b=2

d=power of n=1

a=

2= (condition is True)

Θ(log n)

Θ(log n)

Θ(nlog n)

Program:

#include<stdio.h>

void quicksort(int number[25],int first,int last)

{

int i, j, pivot, temp;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j)

{

while(number[i]<=number[pivot]&&i<last)

i++;

while(number[j]>number[pivot])

j--;

if(i<j)

{

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main()

{

int i, count, number[25];

printf("How many elements are u going to enter?: ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

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

scanf("%d",&number[i]);

quicksort(number,0,count-1);

printf("Order of Sorted elements: ");

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

printf(" %d",number[i]);

return 0;

}

Output:

How many elements are u going to enter?: 5

Enter 5 elements: 20 4 52 63 88

Order of Sorted elements: 4 20 52 63 88

Result: Thus , in the above program successfully executed without errors using quick sort.

b. Implement Merge sort using a C program

Aim: To implement Merge sort using a C program

Description:

Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithms. It divides the given list into two halves, calls itself for the two halves, and then merges the two sorted halves. We have to define the **merge()** function to perform the merging.

The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs are merged into the four-element lists, and so on until we get the sorted list.

Here mainly concentrate on three steps:

1. Divide
2. Conque
3. Combine

i)*Divide*:

Divide the array into sub-arrays that is s1 and s2 with

ii)*Conque*:

Recurrisally sort s1 and s2 sub-arrays

iii*)Combine*:

Combine is sub-arrays s1 and s2

*Algorithm:*

mergesort(low, high)

{

if(n=1)then

return;

else

{

if(low<high) then

;

mergesort(low, high);

mergesort(mid+1, high);

combine(low, mid, high);

}

}

Time complexity of merge sort:

The time following the merge operation is proportional to n. Then the computing time for merge sort is described in recurrence relation.

a n=1

T(n) =

2T(n/2)+cn n>1

Here a is constant

Now

Let T(n)=2T(n/2)+cn-1

Put n= in equation 1 we get  
T =2T+c

Sub T in equation 2 we get

T(n)= 2[2T+ c]+cn

=T+2c]+cn

T(n)=T+2cn

: :

: :

T(n)= T+3cn

: :

: :

T(n)=T+kcn

Now put n= i.e k=

T(n)=nT+ (cn)

=n+T(1)+

= n+cn

T(n)=n[1+]

O(n[1+c])

O(n log n)

Time complexity of merge sort in =O(n log n)

Program:

#include <stdio.h>

#define max 10

int a[11] = { 10, 14, 19, 26, 27, 31};

int b[10];

void merging(int low, int mid, int high)

{

int l1, l2, i;

for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++)

{

if(a[l1] <= a[l2])

b[i] = a[l1++];

else

b[i] = a[l2++];

}

while(l1 <= mid)

b[i++] = a[l1++];

while(l2 <= high)

b[i++] = a[l2++];

for(i = low; i <= high; i++)

a[i] = b[i];

}

void sort(int low, int high)

{

int mid;

if(low < high)

{

mid = (low + high) / 2;

sort(low, mid);

sort(mid+1, high);

merging(low, mid, high);

}

else

{

return;

}

}

int main()

{

int i;

printf("List before sorting\n");

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

printf("%d ", a[i]);

sort(0, max);

printf("\nList after sorting\n");

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

printf("%d ", a[i]);

}

Output:

List before sorting

10 14 19 26 27 31 85 2 3 25 41

List after sorting

1. 3 10 14 19 25 26 27 31 41 85

Result: Thus , in the above program successfully executed without errors using merge sort.

TASK 3

a. Implementation of Stack operations using arrays

Aim: To implementation of Stack operations using arrays

Description:

**Linear Data Structures:** Linear data structures are those data structures in which data elements are accessed (read and written) in a sequential fashion ( one by one)

Eg: Stacks, Queues, Linked list

Stack:

A stack is **a conceptual structure consisting of a set of homogeneous elements and is based on the principle of last in first out (LIFO)**. It is a commonly used abstract data type with two major operations, namely, push and pop.

**Push(**a**):** It adds element an on top of the stack. It takes O(1)O(1) time as each element is inserted starting from the table of the array; there is no need to shift existing elements to make room for the new element.

**Pop():** It removes the element on top of the stack. It also takes O(1)O(1) time as the top contains the index of the most recently added element.

**Top():** It returns the element on top of the stack. It takes O(1)O(1) time as finding the value stored at a particular index in an array is a constant time operation.

peek() − get the top data element of the stack, without removing it.

isFull() − check if stack is full.

isEmpty() − check if stack is empty

Algorithm:

Push:

begin procedure push: stack, data

if stack is full

return null

endif

top ← top + 1

stack[top] ← data

end procedure

pop:

begin procedure pop: stack

if stack is empty

return null

endif

data ← stack[top]

top ← top - 1

return data

end procedure

peek:

begin procedure peek

return stack[top]

end procedure

full:

begin procedure isfull

if top equals to MAXSIZE

return true

else

return false

endif

end procedure

Empty:

begin procedure isempty

if top less than 1

return true

else

return false

endif

end procedure

Program:

#include <stdio.h>

#include <stdlib.h>

#define STACK\_MAX\_SIZE 10

int arr[STACK\_MAX\_SIZE];

int top = -1;

int main()

{

int op, x;

printf("\n STACK USING ARRAYS");

while(1)

{

printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");

printf("Enter your option : ");

scanf("%d",&op);

switch(op)

{

case 1:

printf("Enter element : ");

scanf("%d", &x);

push(x);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

isEmpty();

break;

case 5:

peek();

break;

case 6:

exit(0);

}

}

}

void push(int element)

{

if(top == STACK\_MAX\_SIZE - 1)

{

printf("Stack is overflow.\n");

}

else

{

top = top + 1;

arr[top] = element;

printf("Successfully pushed.\n");

}

}

void pop()

{

if(top ==-1)

{

printf("Stack is underflow.\n");

}

else

{

int element=arr[top];

top = top - 1;

printf("Popped value = %d\n",element);

}

}

void peek()

{

if(top < 0)

printf("Stack is underflow.\n");

else

printf("Peek value = %d\n",arr[top]);

}

void isEmpty()

{

if(top < 0)

printf("Stack is empty.\n");

else

printf("Stack is not empty.\n");

}

void display()

{

if (top < 0)

{

printf("Stack is empty.\n");

}

else

{

printf("Elements of the stack are : " );

for(int i = top; i >= 0; i--) {

printf("%d ",arr[i]);

}

printf("\n");

}

}

Output:

/tmp/F2acUICNBH.o

STACK USING ARRAYS1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option : 1

Enter element : 25

Successfully pushed.

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option : 1

Enter element : 25

Successfully pushed.

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option : 1

Enter element : 66

Successfully pushed.

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option : 2

Popped value = 66

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option : 3

Elements of the stack are : 25 25

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option : 4

Stack is not empty.

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option : 5

Peek value = 25

1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit

Enter your option : 6

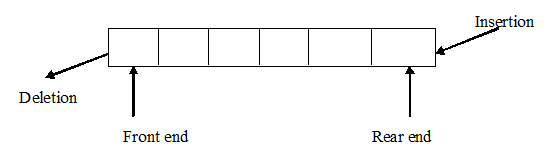
Result: Thus , in the above program successfully executed without errors using stack operation.

b. Implementation of Queue operations using arrays

Aim: To implementation of Queue operations using arrays

Description:

Queue is a linear data structure which follows First in First out (FIFO) mechanism. It means: the first element inserted is the first one to be removed. Queue uses two variables rear and front. Rear is incremented while inserting an element into the queue and front is incremented while deleting element from the queue



Valid Operations on Queue:

* Inserting an element into the queue
* Deleting an element into the queue
* Displaying the elements in the queue

Enqueue: Addition of an element to the queue. Adding an element will be performed after checking whether the queue is full or not. If *rear < n* which indicates that the array is not full then store the element at *arr[rear]* and increment *rear* by *1* but if *rear == n* then it is said to be an Overflow condition as the array is full.

Algorithm:

Step1- Check whether queue is FULL. (rear == SIZE-1)

Step2 - If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.

Step3 - If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value

Dequeue: Removal of an element from the queue. An element can only be deleted when there is at least an element to delete i.e. *rear > 0*. Now, an element at *arr[front]* can be deleted but all the remaining elements have to shift to the left by one position in order for the de-queue operation to delete the second element from the left on another de-queue operation.

Algorithm:

Step 1 - Check whether queue is EMPTY. (front == rear)

Step 2 - If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.

Step 3 - If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).

Front: Get the front element from the queue i.e. *arr[front]* if a queue is not empty.

Display: Print all elements of the queue. If the queue is non-empty, traverse and print all the elements from index *front* to *rear*.

Step1- Check whether queue is EMPTY. (front == rear)

Step2- If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.

Step3- If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'.

Step4- Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value reaches to rear (i <= rear)

Algorithm: queue

**Step1-**Include all the **header files** which are used in the program and define a constant **'SIZE'** with a specific value.

**Step 2 -**Declare all the **user-defined functions** which are used in queue implementation.

**Step 3 -**Create a one-dimensional array with above-defined SIZE (**int queue[SIZE]**)

**Step 4 -**Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**)

**Step 5 -**Then implement the main method by displaying a menu of operations list and make suitable function calls to perform operations selected by the user on queue.

Program:  
#include<stdio.h>

#include<stdlib.h>

#define MAX 5

int queue[MAX];

int front=-1,rear=-1;

void enqueue(int);

void dequeue();

void display();

void main()

{

int ch,ele;

while(1)

{

printf("\n QUEUE USING ARRAYS\n");

printf("\n 1.enqueue 2.dequeue 3.display 4.exit\n");

printf("Enter your choice :\n");

scanf("%d",&ch);

switch(ch)

{

case 1:printf("\n enter the element to insert into Q:");

scanf("%d",&ele);

enqueue(ele);

break;

case 2:dequeue();

break;

case 3:display();

break;

case 4:exit(1);

default:printf("invalid choice\n");

}

}

}

void enqueue(int ele)

{

if(rear==MAX-1)

{

printf("\n Queue overflow--cannot Enqueue\n");

return;

}

if(rear==-1)

front=rear=0;

else

rear=rear+1;

queue[rear]=ele;

}

void dequeue()

{

int e;

if(front==-1)

printf("\nQueue underflow--cannot dequeue\n");

else

{

e=queue[front];

printf("\n Deleted element is:%d",e);

if(front==rear)

front=rear=-1;

else

front=front+1;

}

}

void display()

{

int i;

if(front==-1)

printf("\nQueue is empty---cannot display\n");

else

{

for(i=front;i<=rear;i++)

printf("%5d",queue[i]);

}

}

Output:

QUEUE USING ARRAYS

1.enqueue 2.dequeue 3.display 4.exit

Enter your choice :

1

enter the element to insert into Q:40

1.enqueue 2.dequeue 3.display 4.exit

Enter your choice :

2

Deleted element is:10

1.enqueue 2.dequeue 3.display 4.exit

Enter your choice :

2

Deleted element is:20

1.enqueue 2.dequeue 3.display 4.exit

Enter your choice :

3

40

1.enqueue 2.dequeue 3.display 4.exit

Enter your choice :

2

Deleted element is:40

1.enqueue 2.dequeue 3.display 4.exit

Enter your choice :

2

Queue underflow--cannot dequeue

1.enqueue 2.dequeue 3.display 4.exit

Enter your choice :

4

Result: Thus , in the above program successfully executed without errors using queue operations.

TASK 4

a. Write a c program to convert Infix to postfix expression

Aim: To write a c program to convert Infix to postfix expression

Description:

Infix expressions are readable and solvable by humans. We can easily distinguish the order of operators, and also can use the parenthesis to solve that part first during solving mathematical expressions. The computer cannot differentiate the operators and parenthesis easily, that’s why postfix conversion is needed.

To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining their precedence of them.

Algorithm:

1. Scan the infix notation from left to right and repeat step2 to step 6 for each character in infix notation until the stack is empty.
2. If left parenthesis ‘(‘ is encountered then push it onto the stack
3. If the operand is encountered then add to the postfix expression
4. If an operator is encountered then

(a) Repeatedly pop from the stack and add to postfix expression which has the same precedence as the scanned operator

(b) Push scanned operator onto the stack.

5. If the right parenthesis ‘)’ is encountered then

(a) Repeatedly pop from the stack and add it to the postfix expression until ‘(‘ is encountered

(b) Remove ‘(‘ from the stack and don’t add it to the postfix expression

6. Print postfix expression

Ex:((A\*B)-(C\*D))

|  |  |  |
| --- | --- | --- |
| Infix character scanned | Stack | Postfix expression |
| ( | ( |  |
| ( | (,( |  |
| A | (,( | A |
| \* | (,(,\* | A |
| B | (,(,\* | AB |
| ) | ( | AB\* |
| - | (,- | AB\* |
| ( | (,-,( | AB\* |
| C | (,-,( | AB\*C |
| \* | (,-,(,\* | AB\*C |
| D | (,-,(,\* | AB\*CD |
| ) | (,- | AB\*CD\* |
| ) |  | AB\*CD\*- |

Program:

#include<stdio.h>

#define STACK\_SIZE 100

char stack[STACK\_SIZE];

int top=-1;

void push(char[],char);

char pop(char[]);

int getpriority(char);

void main()

{

char infix[100],postfix[100],temp;

int i,j=0;

printf("Enter any infix exp:");

scanf("%s",infix);

for(i=0;infix[i]!='\0';i++) //a+b-c\*d //ab+cd\*-

{

if(infix[i]=='(')

push(stack,infix[i]); //-\*

else if(isalpha(infix[i])||isdigit(infix[i]))

{

postfix[j]=infix[i]; //ab+cd\*-

j++;

}

else if(infix[i]=='+'||infix[i]=='-'||infix[i]=='\*'||infix[i]=='/'||infix[i]=='%'||infix[i]=='^')

{

while((getpriority(stack[top])>=getpriority(infix[i]))) //0>=1

{

postfix[j]=pop(stack);

j++;

}

push(stack,infix[i]); //+

}

else if(infix[i]==')') //

{

while((stack[top]!='('))

{

postfix[j]=pop(stack);

j++;

}

temp=pop(stack);

}

}//for

while((top!=-1)) //1!=-1 0!=-1 -1!-1 -\* top=1

{

postfix[j]=pop(stack);

j++;

}

postfix[j]='\0';

printf("postfix exp is:%s",postfix); // ab+cd\*-

}

void push(char stack[],char element)

{

if(top==STACK\_SIZE-1)

printf("stackoverflow:");

else

{

top++;

stack[top]=element;

}

}

char pop(char stack[])

{

char val;

if(top==-1)

printf("under flow:");

else

{

val =stack[top];

top--;

}

return val;

}

int getpriority(char op)

{

if(op=='/'||op=='\*'||op=='%')

return 1;

else if(op=='+'||op=='-')

return 0;

else

return -1;

}

Output:

Enter any infix exp:

((A\*B)-(C\*D))postfix exp is:AB\*CD\*-

Result: Thus , in the above program successfully executed without errors using convert Infix to postfix expression.

b) Write a c program to evaluate a Postfix evaluation

Aim: To write a c program to evaluate a Postfix evaluation

Description:

The computer usually evaluates an arithmetic expression written in infix notation in two steps. First, it converts the expression to postfix notation, and then it evaluates the postfix expression. In each step stack is the main tool that is used to accomplish the given task.

Algorithm for evaluation of postfix expression:

Step-1:Scan every character of the postfix expression from left to right and repeat step 3 to step 4until the end of the postfix expression

Step-2:If an operand is encountered, then push it onto the stack.

Step-3:If an operator is encountered, then

Step-4:Pop the top two operands from the stack, where A is the top operand and B is the next-to-top operand.

Step-5:Evaluate B operator A

Step-6:Push the result of the evaluation onto the stack

Step-7:Print “result of expression” at the top of the stack.

Evaluate the following expression by using postfix evaluation algorithm

934\*8+4/-

Sol: 934\*8+4/-

|  |  |
| --- | --- |
| Character scanned | Stack |
| 9  3  4  \*  8  +  4  /  - | 9  9,3  9,3,4  9,12  9,12,8  9,20  9,20,4  9,5  4 |

Program:

#include<stdio.h>

#include<ctype.h>

#define MAX 10

char post[MAX],ch;

int stack[MAX],op1,op2,temp,val,res,n;

int top=-1,i=0,j=0;

void push(int val);

int operate(int op1,int op2,char ch);

int pop();

int main()

{

printf ("enter the postfix exp\n");

scanf("%s",post);

while(post[i]!='\0')

{

ch=post[i];

if(isalpha(ch))

{

printf("enter valuue for %c=",ch);

scanf("%d",&val);

push(val);

}

else

{

if(ch=='\*'||ch=='/'||ch=='+'||ch=='-'||ch=='^')

{

op2=pop();

op1=pop();

res=operate(op1,op2,ch);

push(res);

}

}

i=i+1;

}

res=pop();

printf("the simplified answer for %s =",post);

printf("%d",res);

}

int pop()

{

n=stack[top];

top=top-1;

return n;

}

int operate(int op1,int op2,char ch) //abc \*

{

switch(ch)

{

case '+':temp=op1+op2;

break;

case '-':temp=op1-op2;

break;

case '\*':temp=op1\*op2;break;

case '/':temp=op1/op2;break;

//case '^':temp=pow(op1,(int)op2);

}

return(temp);

}

void push(int val)

{

top++;

stack[top]=val;

}

Output:

enter the postfix exp 962\*4+5

the simplified answer for 962\*4+5 =0

Result: Thus , in the above program successfully executed without errors using postfix expression

TASK 5

Implementation of Circular Queue operations

Aim: To implementation of Circular Queue operations

Description:

Circular Queue:

A circular queue solved the limitations of the normal queue. Thus making it a better pick than the normal queue. It also follows the first come first serve algorithm. A circular Queue is also called a ring Buffer.

Operations On A Circular Queue

1. En-queue- adding an element in the queue if there is space in the queue.
2. Dequeue- Removing elements from a queue if there are any elements in the queue
3. Front- get the first item from the queue.
4. Rear- get the last item from the queue.
5. isEmpty/isFull- checks if the queue is empty or full.

Applications Of A Circular Queue

* Memory management: circular queue is used in memory management.
* Process Scheduling: A CPU uses a queue to schedule processes.
* Traffic Systems: Queues are also used in traffic systems.

**Algorithm to insert an element in a circular queue**

**Step 1:** IF (REAR+1)%MAX = FRONT Write " OVERFLOW "

Goto step 4

[End OF IF]

**Step 2:** IF FRONT = -1 and REAR = -1 SET FRONT = REAR = 0

ELSE IF REAR = MAX - 1 and FRONT ! = 0

SET REAR = 0

ELSE

SET REAR = (REAR + 1) % MAX

[END OF IF]

**Step 3:** SET QUEUE[REAR] = VAL

## **Step 4:** EXIT

**Algorithm to delete an element from the circular queue**

**Step 1:** IF FRONT = -1 Write " UNDERFLOW " Goto Step 4

[END of IF]

**Step 2:** SET VAL = QUEUE[FRONT]

**Step 3:** IF FRONT = REAR

SET FRONT = REAR = -1

ELSE

IF FRONT = MAX -1

SET FRONT = 0

ELSE

SET FRONT = FRONT + 1

[END of IF]

[END OF IF]

**Step 4:** EXIT

Program:

#include<stdio.h>

#include<stdlib.h>

#define QUEUE\_SIZE 4

int queue[QUEUE\_SIZE];

int front= -1,rear= -1;

void enqueue(int);

void dequeue();

void display();

void main()

{

int ch,ele;

while(1)

{

printf("\n\n-----options----\n\n");

printf("1.enqueue\n2.dequeue\n3.display\n4.exit\n");

printf("enter your choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("enter the element to inserted:");

scanf("%d",&ele);

enqueue(ele);

break;

case 2: dequeue();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("invalid choice\n");

}

}

}

void enqueue(int ele)

{

if(((rear==QUEUE\_SIZE-1) && (front== -1)) || ((front == rear) && (front != -1)))

printf("\nqueue is overflow");

else if((rear == QUEUE\_SIZE-1) && (front != -1)) //Inserting elemets in 2nd round

{

rear=0;

queue[rear] = ele;

}

else

{

rear = rear+1;

queue[rear] = ele;

}

}

void dequeue()

{

if((front == rear) && (front == -1))

printf("queue is underflow\n");

else

{

if(front == QUEUE\_SIZE-1)

front = -1;

front = front+1;

printf("deleted element is:%d",queue[front]);

if(front == rear)

front = rear = -1;

}

}

void display()

{

int i;

if((front == rear) && (front == -1))

printf("\n queue is empty");

else

{

if(front<rear)

{

for(i=front+1; i<=rear; i++)

printf("%5d", queue[i]);

}

else

{

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

printf("%5d",queue[i]);

for(i=front+1; i<=QUEUE\_SIZE-1; i++)

printf("%5d",queue[i]);

}

}

}

Output:

/tmp/A7mCtbddRY.o

-----options----

1.enqueue

2.dequeue

3.display

4.exit

enter your choice

1

enter the element to inserted:25 35 25 2

-----options----

1.enqueue

2.dequeue

3.display

4.exit

enter your choice

invalid choice

-----options----

1.enqueue

2.dequeue

3.display

4.exit

enter your choice

invalid choice

-----options----

1.enqueue

2.dequeue

3.display

4.exit

enter your choice

deleted element is:25

-----options----

1.enqueue

2.dequeue

3.display

4.exit

enter your choice

2

queue is underflow

-----options----

1.enqueue

2.dequeue

3.display

4.exit

enter your choice

3

queue is empty

-----options----

1.enqueue

2.dequeue

3.display

4.exit

enter your choice

4

Result: Thus , in the above program successfully executed without errors using Circular Queue operations.

TASK6

Implementation of Single Linked List operations.

Aim: To implementation of Single Linked List operations.

Description:

A singly linked list, or simply a linked list, is a linear collection of data items. The linear order is given utilizing POINTERS. These types of lists are often referred to as a linear linked lists.

\* Each item in the list is called a node.

\* Each node of the list has two fields:

1. Data - contains the element being stored in the list.
2. Next address- contains the address of the next node in the list.

\* The last node in the list contains a NULL pointer to indicate that it is the end of the list

Following are the basic operations supported by a list.

1. Creation: Linked list allows us to insert or delete nodes whenever necessary. First we have to create the node.
2. Insertion − Adds an element at the beginning of the list.
3. Deletion − Deletes an element at the beginning of the list.
4. Display − Displays the complete list.
5. Search − Searches an element using the given key.
6. Delete − Deletes an element using the given key

Algorithms:

1. Creation:
2. Allocate memory for the new node
3. Assign data part of the new node to element and next field to NULL
4. Assign START pointer to a new node
5. Take pointer variable ptr which points to START
6. Move ptr to which points to the next field of ptr =NULL
7. Set next field of ptr points to new nodes
8. Repeat the above steps for the required number of nodes in the list
9. Insertion:

1) A new node can be inserted into the linked list which requires resetting of pointers.

2)Inserting a new node into the list has 4 situations

3). Inserting a node at the beginning

4). Inserting a node at the end

5). Inserting a node after a given node

## 6). Inserting a node before a given node

1. Deletion:

**Step-1:** IF HEAD = NULL

Write UNDERFLOW

   Go to Step 8

  [END OF IF]

**Step-2:** SET PTR = HEAD

**Step-3:** Repeat Steps 4 and 5 while PTR -> NEXT!= NULL

**Step-4:** SET PREPTR = PTR

**Step-5:** SET PTR = PTR -> NEXT

[END OF LOOP]

**Step-6:** SET PREPTR -> NEXT = NULL

**Step-7:** FREE PTR

**Step-8:** EXIT

1. Display:

**Step-1:**Take pointer ptr which points to START

Step-2:Move ptr which points to NULL

Step-3: The print data part of ptr

1. Search :

**Step-1:**Take pointer variable ptr which points to START

Step-2:Move ptr which points to NULL

Step-3:If data part of ptr =key then return ptr otherwise return NULL

1. Delete:

Step-1: Find the previous node of the node to be deleted.

Step-2:Change the next of the previous node.

Step-3:Free memory for the node to be deleted

Program:

#include<stdio.h>

struct node

{

int data;

struct node \*next;

}

\*start=NULL;

void create();

void insert\_after();

void delete\_after();

void display();

struct node \*search(int);

void main()

{

struct node \*ptr;

int ch,key;

while(1)

{

printf("\nMAIN MENU\n");

printf("1.create\n 2.Insert After\n3.Delete After\n4.print \n5.search\n 6.exit");

printf("Enter Choice");

scanf("%d",&ch);

switch(ch)

{

case 1: create();

break;

case 2 :insert\_after();

break;

case 3: delete\_after(); break;

case 4: display();break;

case 5: printf("Enter Key element");

scanf("%d",&key);

ptr=search(key);

if(ptr==NULL)

printf("key is not found");

else

printf("key is found");

case 6: exit(0);

default:printf("invalid option");

}

}

}

void create()

{

struct node \*ptr,\*new\_node;

int ele;

printf("Enter -1 to end the list \nEnter element");

scanf("%d",&ele);

while(ele!=-1)

{

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node->data=ele;

new\_node->next=NULL;

if(start==NULL)

start=new\_node;

else

{

ptr=start;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=new\_node;

}

printf("Enter -1 to end the list \nEnter element");

scanf("%d",&ele);

}

}

void insert\_after()

{

struct node \*keyptr,\*newnode;

int ele,key;

printf("\nEnter nEnter element");

scanf("%d",&ele);

printf("\nEnter the key element");

scanf("%d",&key);

newnode=(struct node \*)malloc(sizeof(struct node));

newnode->data=ele;

keyptr= search(key);

if(keyptr==NULL)

printf("\nKey is not found");

else

{

newnode->next=keyptr->next;

keyptr->next=newnode;

}

}

struct node\* search(int key)

{

struct node \*ptr;

ptr=start;

while(ptr!=NULL)

{

if(ptr->data==key)

return ptr;

ptr=ptr->next;

}

return NULL;

}

void display()

{

struct node \*ptr;

ptr=start;

if(ptr==NULL)

printf("\n list is empty");

else

{

while(ptr!=NULL)

{

printf("%5d->",ptr->data);

ptr=ptr->next;

}

}

}

void delete\_after()

{

struct node \*keyptr,\*Aptr;

int key;

printf("\nEnter the key element");

scanf("%d",&key);

keyptr= search(key);

if(keyptr==NULL)

printf("\nKey is not found");

else

{

Aptr=keyptr->next;

keyptr->next=Aptr->next;

free(Aptr);

}

}

Output:

MAIN MENU

1.create

2.Insert After

3.Delete After

4.print

5.search

6.exitEnter Choice1

Enter -1 to end the list

Enter element-1

MAIN MENU

1.create

2.Insert After

3.Delete After

4.print

5.search

6.exitEnter Choice2

Enter nEnter element123

Enter the key element3

Key is not found

MAIN MENU

1.create

2.Insert After

3.Delete After

4.print

5.search

6.exitEnter Choice3

Enter the key element2

Key is not found

MAIN MENU

1.create

2.Insert After

3.Delete After

4.print

5.search

6.exitEnter Choice4

list is empty

MAIN MENU

1.create

2.Insert After

3.Delete After

4.print

5.search

6.exitEnter Choice2

Enter nEnter element123456

Enter the key element3

Key is not found

MAIN MENU

1.create

2.Insert After

3.Delete After

4.print

5.search

6.exitEnter Choice3

Enter the key element3

Key is not found

MAIN MENU

1.create

2.Insert After

3.Delete After

4.print

5.search

6.exitEnter Choice2

Enter nEnter element7

Enter the key element3

Key is not found

MAIN MENU

1.create

2.Insert After

3.Delete After

4.print

5.search

6.exitEnter Choice5

Enter Key element3

key is not found

Result: Thus , in the above program successfully executed without errors using Single Linked List operations.

TASK 7

Implementation of Circular Linked List operations

Aim: To implementation of Circular Linked List operations

Description:

In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We can have a circular singly linked list as well as a circular doubly linked list.

We traverse a circular singly linked list until we reach the same node where we started. The circular singly liked list has no beginning and no end. There is no null value present in the next part of any of the nodes.

Algorithm:

Step-1:Adding a node into the circular singly linked list at the beginning

Step-2:Adding a node into a circular singly linked list at the end.

Step-3:Removing the node from the circular singly linked list at the beginning.

Step-4:Removing the node from the circular singly linked list at the end.

Step-5:Compare each element of the node with the given item and return the location at which the item is present in the list otherwise return null.

Step-6:Visiting each element of the list at least once to perform some specific operation.

Program:

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

void beginsert ();

void lastinsert ();

void randominsert();

void begin\_delete();

void last\_delete();

void random\_delete();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 7)

{

printf("\n\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\n");

printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from last\n5.Search for an element\n6.Show\n7.Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

beginsert();

break;

case 2:

lastinsert();

break;

case 3:

begin\_delete();

break;

case 4:

last\_delete();

break;

case 5:

search();

break;

case 6:

display();

break;

case 7:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

}

void beginsert()

{

struct node \*ptr,\*temp;

int item;

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter the node data?");

scanf("%d",&item);

ptr -> data = item;

if(head == NULL)

{

head = ptr;

ptr -> next = head;

}

else

{

temp = head;

while(temp->next != head)

temp = temp->next;

ptr->next = head;

temp -> next = ptr;

head = ptr;

}

printf("\nnode inserted\n");

}

}

void lastinsert()

{

struct node \*ptr,\*temp;

int item;

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\nOVERFLOW\n");

}

else

{

printf("\nEnter Data?");

scanf("%d",&item);

ptr->data = item;

if(head == NULL)

{

head = ptr;

ptr -> next = head;

}

else

{

temp = head;

while(temp -> next != head)

{

temp = temp -> next;

}

temp -> next = ptr;

ptr -> next = head;

}

printf("\nnode inserted\n");

}

}

void begin\_delete()

{

struct node \*ptr;

if(head == NULL)

{

printf("\nUNDERFLOW");

}

else if(head->next == head)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

while(ptr -> next != head)

ptr = ptr -> next;

ptr->next = head->next;

free(head);

head = ptr->next;

printf("\nnode deleted\n");

}

}

void last\_delete()

{

struct node \*ptr, \*preptr;

if(head==NULL)

{

printf("\nUNDERFLOW");

}

else if (head ->next == head)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

while(ptr ->next != head)

{

preptr=ptr;

ptr = ptr->next;

}

preptr->next = ptr -> next;

free(ptr);

printf("\nnode deleted\n");

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag=1;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

if(head ->data == item)

{

printf("item found at location %d",i+1);

flag=0;

}

else

{

while (ptr->next != head)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

}

if(flag != 0)

{

printf("Item not found\n");

}

}

}

void display()

{

struct node \*ptr;

ptr=head;

if(head == NULL)

{

printf("\nnothing to print");

}

else

{

printf("\n printing values ... \n");

while(ptr -> next != head)

{

printf("%d\n", ptr -> data);

ptr = ptr -> next;

}

printf("%d\n", ptr -> data);

}

}

Output:

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

1

Enter the node data?25

node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

2

Enter Data?20

node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

2

Enter Data?30

node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

3

node deleted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

4

node deleted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

1

Enter the node data?35

node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

2

Enter Data?26

node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

5

Enter item which you want to search?

26

Item not found

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

6

printing values ...

35

20

26

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search for an element

6.Show

7.Exit

Enter your choice?

7 \*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Result: Thus , in the above program successfully executed without errors using circular linked list operations.

TASK 8

Implementation of Double Linked List operations

Aim: To implementation of Double Linked List operations

Description:

In a singly linked list, each element contains a pointer to the next element. We have seen this before. In a single linked list, traversing is possible only in one direction. Sometimes, we have to traverse the list in both directions to improve the performance of algorithms. To enable this, we require links in both the directions, that is, the element should have pointers to the right element as well as to its left element. This type of list is called a doubly linked list.

Basic Operations

Following are the basic operations supported by a list.

* 1. Insertion − Adds an element at the beginning of the list.
  2. Deletion − Deletes an element at the beginning of the list.
  3. Insert Last − Adds an element at the end of the list.
  4. Delete Last − Deletes an element from the end of the list.
  5. Insert After − Adds an element after an item on the list.
  6. Delete − Deletes an element from the list using the key.
  7. Display forward − Displays the complete list in a forward manner.
  8. Display backward − Displays the complete list in a backward manner.

Algorithm:

Step-1:Start

## Step-2: Adding the node into the linked list at the beginning.

Step-3: Adding the node into the linked list to the end.

Step-4: Adding the node into the linked list after the specified node.

Step-5 :Removing the node from the beginning of the list.

Step-6: Removing the node from end of the list.

Step-7: Removing the node which is present just after the node containing the given data.

Step-8: Comparing each node data with the item to be searched and return the location of the item in the list if the item is found else return null.

Step-9: Visiting each node of the list at least once in order to perform some specific operation like searching, sorting, displaying, etc.

Step-10: Stop.

Program:

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

struct node \*next;

int data;

};

struct node \*head;

void insertion\_beginning();

void insertion\_last();

void insertion\_specified();

void deletion\_beginning();

void deletion\_last();

void deletion\_specified();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 9)

{

printf("\n\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\n");

printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n

5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

insertion\_beginning();

break;

case 2:

insertion\_last();

break;

case 3:

insertion\_specified();

break;

case 4:

deletion\_beginning();

break;

case 5:

deletion\_last();

break;

case 6:

deletion\_specified();

break;

case 7:

search();

break;

case 8:

display();

break;

case 9:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

}

void insertion\_beginning()

{

struct node \*ptr;

int item;

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter Item value");

scanf("%d",&item);

if(head==NULL)

{

ptr->next = NULL;

ptr->prev=NULL;

ptr->data=item;

head=ptr;

}

else

{

ptr->data=item;

ptr->prev=NULL;

ptr->next = head;

head->prev=ptr;

head=ptr;

}

printf("\nNode inserted\n");

}

}

void insertion\_last()

{

struct node \*ptr,\*temp;

int item;

ptr = (struct node \*) malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value");

scanf("%d",&item);

ptr->data=item;

if(head == NULL)

{

ptr->next = NULL;

ptr->prev = NULL;

head = ptr;

}

else

{

temp = head;

while(temp->next!=NULL)

{

temp = temp->next;

}

temp->next = ptr;

ptr ->prev=temp;

ptr->next = NULL;

}

}

printf("\nnode inserted\n");

}

void insertion\_specified()

{

struct node \*ptr,\*temp;

int item,loc,i;

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\n OVERFLOW");

}

else

{

temp=head;

printf("Enter the location");

scanf("%d",&loc);

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

{

temp = temp->next;

if(temp == NULL)

{

printf("\n There are less than %d elements", loc);

return;

}

}

printf("Enter value");

scanf("%d",&item);

ptr->data = item;

ptr->next = temp->next;

ptr -> prev = temp;

temp->next = ptr;

temp->next->prev=ptr;

printf("\nnode inserted\n");

}

}

void deletion\_beginning()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

head = head -> next;

head -> prev = NULL;

free(ptr);

printf("\nnode deleted\n");

}

}

void deletion\_last()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

if(ptr->next != NULL)

{

ptr = ptr -> next;

}

ptr -> prev -> next = NULL;

free(ptr);

printf("\nnode deleted\n");

}

}

void deletion\_specified()

{

struct node \*ptr, \*temp;

int val;

printf("\n Enter the data after which the node is to be deleted : ");

scanf("%d", &val);

ptr = head;

while(ptr -> data != val)

ptr = ptr -> next;

if(ptr -> next == NULL)

{

printf("\nCan't delete\n");

}

else if(ptr -> next -> next == NULL)

{

ptr ->next = NULL;

}

else

{

temp = ptr -> next;

ptr -> next = temp -> next;

temp -> next -> prev = ptr;

free(temp);

printf("\nnode deleted\n");

}

}

void display()

{

struct node \*ptr;

printf("\n printing values...\n");

ptr = head;

while(ptr != NULL)

{

printf("%d\n",ptr->data);

ptr=ptr->next;

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data == item)

{

printf("\nitem found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("\nItem not found\n");

}

}

}

Output:

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

8

printing values...

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

1

Enter Item value12

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

1

Enter Item value123

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

1

Enter Item value1234

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

8

printing values...

1234

123

12

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

2

Enter value89

node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

3

Enter the location1

Enter value12345

node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

8

printing values...

1234

123

12345

12

89

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

4

node deleted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

5

node deleted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

8

printing values...

123

12345

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

6

Enter the data after which the node is to be deleted : 123

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

8

printing values...

123

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

7

Enter item which you want to search?

123

item found at location 1

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

6

Enter the data after which the node is to be deleted : 123

Can't delete

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete the node after the given data

7.Search

8.Show

9.Exit

Enter your choice?

9

Exited.

Result: Thus , in the above program successfully executed without errors using Double Linked List operations

TASK 9

a. Implementation of the following operations on BST operations.

i. Create

ii. insert

iii Search

Aim: To implement of the following operations on BST operations are create, insert and search.

Description:

**A binary Search Tree (or BST)** is a special kind of binary tree in which the values of all the nodes of the left subtree of any node of the tree are smaller than the value of the node. Also, the values of all the nodes of the right subtree of any node are greater than the value of the node.

Insertion:

Insert function is used to add a new element in a binary search tree at appropriate location. Insert function is to be designed in such a way that, it must node violate the property of binary search tree at each value.

Algorithm:

Step-1:Allocate the memory for the tree.

Step-2:Set the data part to the value and set the left and right pointer of the tree, point to NULL.

Step-3:If the item to be inserted will be the first element of the tree, then the left and right of this node will point to NULL.

Step-4:Else, check if the item is less than the root element of the tree, if this is true, then recursively perform this operation with the left of the root.

Step-5:If this is false, then perform this operation recursively with the right sub-tree of the root.

Delete:

The delete function is used to delete the specified node from a binary search tree. However, we must delete a node from a binary search tree in such a way, that the property of a binary search tree doesn't violate. There are three situations of deleting a node from a binary search tree.

Algorithm:

**Delete (TREE, ITEM)**

**Step 1:**IF TREE = NULL

  Write "item not found in the tree" ELSE IF ITEM < TREE -> DATA

 Delete(TREE->LEFT, ITEM)

 ELSE IF ITEM > TREE -> DATA

  Delete(TREE -> RIGHT, ITEM)

ELSE IF TREE -> LEFT AND TREE -> RIGHT

 SET TEMP = findLargestNode(TREE -> LEFT)

 SET TREE -> DATA = TEMP -> DATA

  Delete(TREE -> LEFT, TEMP -> DATA)

 ELSE

  SET TEMP = TREE

  IF TREE -> LEFT = NULL AND TREE -> RIGHT = NULL

  SET TREE = NULL

 ELSE IF TREE -> LEFT != NULL

 SET TREE = TREE -> LEFT

 ELSE

   SET TREE = TREE -> RIGHT

 [END OF IF]

 FREE TEMP

[END OF IF]

**Step 2:** END

Search:

Searching means finding or locating some specific element or node within a data structure. However, searching for some specific node in a binary search tree is pretty easy because the element in BST are stored in a particular order.

Algorithm:

Step-1:Compare the element with the root of the tree.

Step-2:If the item is matched then returns the location of the node.

Step-3:Otherwise check if an item is less than the element present on the root, if so then move to the left sub-tree.

Step-4:If not, then move to the right sub-tree.

Step-5:Repeat this procedure recursively until a match is found.

Step-6:If an element is not found then return NULL.

Program:

//BINARY SEARCH TREE OPERATIONS

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*lchild;

int info;

struct node \*rchild;

};

struct node \*search(struct node \*ptr, int skey);

struct node \*insert(struct node \*ptr, int ikey);

struct node \*del(struct node \*ptr, int dkey);

void display(struct node \*ptr);

int main( )

{

struct node \*root=NULL,\*ptr;

int choice,k;

while(1)

{

printf("\n");

printf("\n 1.Insert 2.Delete 3. Search 4.Display 5.Quit \nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\nEnter the key to be inserted : ");

scanf("%d",&k);

root = insert(root, k);

break;

case 2:

printf("\nEnter the key to be deleted : ");

scanf("%d",&k);

root = del(root,k);

break;

case 3:

printf("\nEnter the key to be searched : ");

scanf("%d",&k);

ptr = search(root, k);

if(ptr!=NULL)

printf("\nKey is found\n");

break;

case 4:

printf("\n Elements of BST are :\n");

display(root);

break;

case 5: exit(1);

default: printf("\n Invalid Choice\n");

}/\*End of switch \*/

}/\*End of while \*/

return 0;

}

struct node \*search(struct node \*ptr, int skey)

{

if(ptr==NULL)

{

printf("key not found\n");

return NULL;

}

else if(skey < ptr->info)/\*search in left subtree\*/

return search(ptr->lchild, skey);

else if(skey > ptr->info)/\*search in right subtree\*/

return search(ptr->rchild, skey);

else /\*skey found\*/

return ptr;

}

struct node \*insert(struct node \*ptr, int ikey )

{

if(ptr==NULL)

{

ptr = (struct node \*) malloc(sizeof(struct node));

ptr->info = ikey;

ptr->lchild = NULL;

ptr->rchild = NULL;

}

else if(ikey < ptr->info) /\*Insertion in left subtree\*/

ptr->lchild = insert(ptr->lchild, ikey);

else if(ikey > ptr->info) /\*Insertion in right subtree \*/

ptr->rchild = insert(ptr->rchild, ikey);

else

printf("\nDuplicate key\n");

return ptr;

}

struct node \*del(struct node \*ptr, int dkey)

{

struct node \*tmp, \*succ;

if( ptr == NULL)

{

printf("\nDkey not found\n");

return(ptr);

}

if( dkey < ptr->info )/\*delete from left subtree\*/

ptr->lchild = del(ptr->lchild, dkey);

else if( dkey > ptr->info )/\*delete from right subtree\*/

ptr->rchild = del(ptr->rchild, dkey);

else

{

/\*key to be deleted is found\*/

if( ptr->lchild!=NULL && ptr->rchild!=NULL ) /\*2 children\*/

{

succ=ptr->rchild;

while(succ->lchild)

succ=succ->lchild;

ptr->info=succ->info;

ptr->rchild = del(ptr->rchild, succ->info);

}

else

{

tmp = ptr;

if( ptr->lchild != NULL ) /\*only left child\*/

ptr = ptr->lchild;

else if( ptr->rchild != NULL) /\*only right child\*/

ptr = ptr->rchild;

else /\* no child \*/

ptr = NULL;

free(tmp);

}

}

return ptr;

}

void display(struct node \*ptr)

{

if(ptr == NULL )/\*Base Case\*/

return;

display(ptr->lchild);

printf("%d ",ptr->info);

display(ptr->rchild);

}

Output:

/tmp/puoV2bf4Bp.o

1.Insert 2.Delete 3. Search 4.Display 5.Quit

Enter your choice : 1

Enter the key to be inserted : 25

1.Insert 2.Delete 3. Search 4.Display 5.Quit

Enter your choice : 1

Enter the key to be inserted : 26

1.Insert 2.Delete 3. Search 4.Display 5.Quit

Enter your choice : 1

Enter the key to be inserted : 36

1.Insert 2.Delete 3. Search 4.Display 5.Quit

Enter your choice : 1

Enter the key to be inserted : 45

1.Insert 2.Delete 3. Search 4.Display 5.Quit

Enter your choice : 2

Enter the key to be deleted : 45

1.Insert 2.Delete 3. Search 4.Display 5.Quit

Enter your choice : 1

Enter the key to be inserted : 35

1.Insert 2.Delete 3. Search 4.Display 5.Quit

Enter your choice : 4

Elements of BST are :

25 26 35 36

1.Insert 2.Delete 3. Search 4.Display 5.Quit

Enter your choice : 3

Enter the key to be searched : 35

Key is found

1.Insert 2.Delete 3. Search 4.Display 5.Quit

Enter your choice : 5

Result: Thus , in the above program successfully executed without errors using BST operations are create, insert and search.

TASK 10

Implementation of Pre-Order, In-Order and Post-Order in a BST using recursion

Aim: To implementation of Pre-Order, In-Order and Post-Order in a BST using recursion

Description:

**A binary Search Tree (or BST)** is a special kind of binary tree in which the values of all the nodes of the left sub-tree of any node of the tree are smaller than the value of the node. Also, the values of all the nodes of the right sub-tree of any node are greater than the value of the node.



Depth First Traversals:   
(a) Inorder (Left, Root, Right) : 4 2 5 1 3   
(b) Preorder (Root, Left, Right) : 1 2 4 5 3   
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

**In-order Traversal:**

Uses of In-order:   
In the case of binary search trees (BST), In-order traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of In-order traversal where In-order traversal s reversed can be used.

Example: In order traversal for the above-given figure is 4 2 5 1 3

Algorithm Inorder(tree)

1. Traverse the left sub-tree, i.e., call In-order(left-sub-tree)

2. Visit the root.

3. Traverse the right sub-tree, i.e., call In-order(right-sub-tree)

**Preorder Traversal:**

Uses of Preorder   
Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on an expression tree

Example: Preorder traversal for the above-given figure is 1 2 4 5 3

Algorithm Preorder(tree)

1. Visit the root.

2. Traverse the left sub-tree, i.e., call Preorder(left-sub-tree)

3. Traverse the right sub-tree, i.e., call Preorder(right-sub-tree)

**Post-order Traversal:**

Uses of Post-order   
Post-order traversal is used to delete the tree.

Post-order traversal is also useful to get the postfix expression of an expression tree.

Algorithm Post-order(tree)

1. Traverse the left sub-tree, i.e., call Post-order(left-sub-tree)

2. Traverse the right sub-tree, i.e., call Post-order(right-sub-tree)

3. Visit the root.

Program:

#include <stdio.h>

#include <stdlib.h>

/\* A binary tree node has data, pointer to left child

and a pointer to right child \*/

struct node

{

int data;

struct node\* left;

struct node\* right;

};

/\* Helper function that allocates a new node with the

given data and NULL left and right pointers. \*/

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);

}

/\* Given a binary tree, print its nodes according to the

"bottom-up" postorder traversal. \*/

void printPostorder(struct node\* node)

{

if (node == NULL)

return;

// first recur on left subtree

printPostorder(node->left);

// then recur on right subtree

printPostorder(node->right);

// now deal with the node

printf("%d ", node->data);

}

/\* Given a binary tree, print its nodes in inorder\*/

void printInorder(struct node\* node)

{

if (node == NULL)

return;

/\* first recur on left child \*/

printInorder(node->left);

/\* then print the data of node \*/

printf("%d ", node->data);

/\* now recur on right child \*/

printInorder(node->right);

}

/\* Given a binary tree, print its nodes in preorder\*/

void printPreorder(struct node\* node)

{

if (node == NULL)

return;

/\* first print data of node \*/

printf("%d ", node->data);

/\* then recur on left subtree \*/

printPreorder(node->left);

/\* now recur on right subtree \*/

printPreorder(node->right);

}

/\* Driver program to test above functions\*/

int main()

{

struct node\* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

printf("\nPreorder traversal of binary tree is \n");

printPreorder(root);

printf("\nInorder traversal of binary tree is \n");

printInorder(root);

printf("\nPostorder traversal of binary tree is \n");

printPostorder(root);

getchar();

return 0;

}

Output:

/tmp/puoV2bf4Bp.o

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

1. 5 2 3 1

Result: Thus , in the above program successfully executed without errors using by Pre-Order, In-Order and Post-Order in a BST using recursion

TASK 11

a. Implementation of DFS Traversals on graphs

Aim: To implementation of DFS Traversals on graphs

Description:

It is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

Because of the recursive nature, a stack data structure can be used to implement the DFS algorithm. The process of implementing the DFS is similar to the BFS algorithm.

The step by step process to implement the DFS traversal is given as follows -

1. First, create a stack with the total number of vertices in the graph.
2. Now, choose any vertex as the starting point of traversal, and push that vertex into the stack.
3. After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4. Now, repeat steps 3 and 4 until no vertices are left to visit from the vertex on the stack's top.
5. If no vertex is left, go back and pop a vertex from the stack.
6. Repeat steps 2, 3, and 4 until the stack is empty.

*Algorithm:*

**Step-1:** SET STATUS = 1 (ready state) for each node in G

**Step-2:** Push the starting node A on the stack and set its STATUS = 2 (waiting for state)

**Step-3:** Repeat Steps 4 and 5 until STACK is empty

**Step-4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)

**Step-5:** Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting for state)

[END OF LOOP]

**Step-6:** EXIT

Program:

#include <stdio.h>

#include <stdlib.h>

struct node

{

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph

{

int numVertices;

int\* visited;

// We need int\*\* to store a two dimensional array.

// Similary, we need struct node\*\* to store an array of Linked lists

struct node\*\* adjLists;

};

// DFS algo

void DFS(struct Graph\* graph, int vertex)

{

struct node\* adjList = graph->adjLists[vertex];

struct node\* temp = adjList;

graph->visited[vertex] = 1;

printf("Visited %d \n", vertex);

while (temp != NULL)

{

int connectedVertex = temp->vertex;

if (graph->visited[connectedVertex] == 0)

{

DFS(graph, connectedVertex);

}

temp = temp->next;

}

}

// Create a node

struct node\* createNode(int v)

{

struct node\* newNode = malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

// Create graph

struct Graph\* createGraph(int vertices)

{

struct Graph\* graph = malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = malloc(vertices \* sizeof(struct node\*));

graph->visited = malloc(vertices \* sizeof(int));

int i;

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

{

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

// Add edge

void addEdge(struct Graph\* graph, int src, int dest)

{

// Add edge from src to dest

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

// Print the graph

void printGraph(struct Graph\* graph)

{

int v;

for (v = 0; v < graph->numVertices; v++)

{

struct node\* temp = graph->adjLists[v];

printf("\n Adjacency list of vertex %d\n ", v);

while (temp)

{

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("\n");

}

}

int main()

{

struct Graph\* graph = createGraph(4);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 2, 3);

printGraph(graph);

DFS(graph, 2);

return 0;

}

Output:

/tmp/JNiyZB0XLB.o

Adjacency list of vertex 0

2 -> 1 ->

Adjacency list of vertex 1

2 -> 0 ->

Adjacency list of vertex 2

3 -> 1 -> 0 ->

Adjacency list of vertex 3

2 ->

Visited 2

Visited 3

Visited 1

Visited 0

Result: Thus , in the above program successfully executed without errors using DFS Traversals on graphs.

**TASK 12**

**Implementation of BFS operations**

Aim: To implementation of BFS operations

Description:

BFS is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. The aim of the BFS algorithm is to traverse the graph as close as possible to the root node. The queue is used in the implementation of the breadth-first search.

Algorithm:

Step-1:Push the root node in the Queue.

Step-2:Loop until the queue is empty.

Step-3:Remove the node from the Queue.

Step-4:If the removed node has unvisited child nodes, mark them as visited and insert the unvisited children in the queue.

Program:

#include<stdio.h>

#include<conio.h>

int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v)

{

for(i = 1; i <= n; i++)

if(a[v][i] && !visited[i])

q[++r] = i;

if(f <= r)

{

visited[q[f]] = 1;

bfs(q[f++]);

}

}

void main()

{

clrscr();

int v;

printf("Enter the number of vertices: ");

scanf("%d",&n);

for(i=1; i <= n; i++)

{

q[i] = 0;

visited[i] = 0;

}

printf("\nEnter graph data in matrix form:\n");

for(i=1; i<=n; i++)

{

for(j=1;j<=n;j++)

{

scanf("%d", &a[i][j]);

}

}

printf("Enter the starting vertex: ");

scanf("%d", &v);

bfs(v);

printf("\nThe node which are reachable are:");

for(i=1; i <= n; i++)

{

if(visited[i])

printf(" %d", i);

else

{

printf("\nBFS is not possible. All nodes are not reachable!");

break;

}

}

getch();

}

Output:

/tmp/JNiyZB0XLB.o

Enter the number of vertices: 3

Enter graph data in matrix form:

2 4 5 2 3 4 1 7 8

Enter the starting vertex: 2

The node which are reachable are: 1 2 3

Result: Thus , in the above program successfully executed without errors using BFS operation.