**Program: - 3**

|  |
| --- |
| **Create a structure with following Data members:**  **1. Integer Array**  **2. Size of the Array**  **Perform the following operations on stack using user-defined functions:**  **1. Push**  **2. Pop**  **3. Isempty**  **4. Isfull**  **5. Peep**  **Create a file which stores all values of Array through Stack. Has it reversed the order of the elements of the Array? Why?** |

* **Code: -**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<malloc.h>

struct ArrayStack

{

int top;

int capacity;

int \*array;

};

struct ArrayStack\* createstack(int cap)

{

struct ArrayStack \*stack;

stack=malloc(sizeof(struct ArrayStack));

stack->capacity=cap;

stack->top=-1;

stack->array=malloc(sizeof(int)\*stack->capacity);

return(stack);

}

int isFull(struct ArrayStack \*stack)

{

if(stack->top==stack->capacity)

{

return(1);

}

else

{

return(0);

}

}

int isEmpty(struct ArrayStack \*stack)

{

if(stack->top==-1)

{

return(1);

}

else

{

return(0);

}

}

void push(struct ArrayStack \*stack,int item)

{

if(!isFull(stack))

{

stack->top++;

stack->array[stack->top]=item;

}

}

int peep(struct ArrayStack \*stack)

{

int item;

if(!isFull(stack))

{

item=stack->array[stack->top];

return(item);

}

else

{

return(-1);

}

}

int pop(struct ArrayStack \*stack)

{

int item;

if(!isEmpty(stack))

{

item=stack->array[stack->top];

stack->top--;

return(item);

}

return(-1);

}

main()

{

struct ArrayStack \*stack;

int choice,item,cap;

int peepitem;

/\* printf("Enter the Capicity of the Stack");

scanf("%d",&cap);\*/

stack=createstack(4);

do

{

clrscr();

printf("\n1. Push");

printf("\n2. Pop");

printf("\n3. Peep");

printf("\n4. Exit");

printf("\nEnter Your Choice");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("\nEnter n number");

scanf("%d",&item);

push(stack,item);

break;

case 2:

item=pop(stack);

if(item==-1)

{

printf("Stack is Empty");

}

else

{

printf("\n Poped value is %d",item);

}

break;

case 3:

peepitem=peep(stack);

if(peepitem==-1)

{

printf("Stack is Empty");

}

else

{

printf("\nYour Peep Element is : %d",peepitem);

}

break;

case 4:

exit(0);

break;

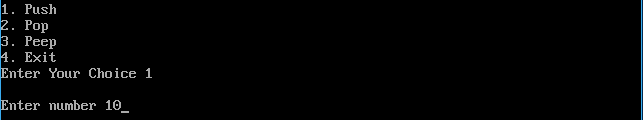
}

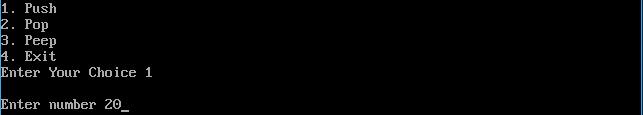
getch();

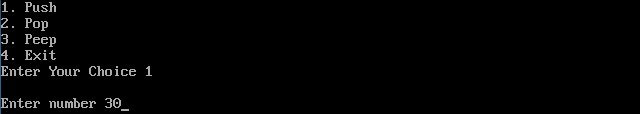
} while(choice!=4);

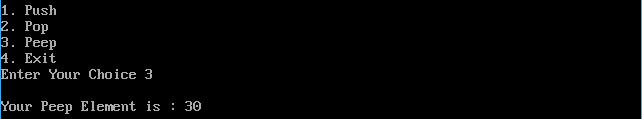
}

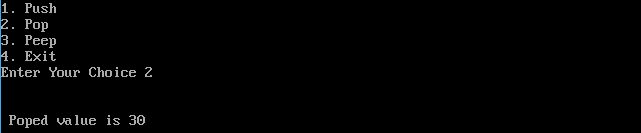
* **Output: -**











**Program: - 4**

|  |
| --- |
| **Create a user-defined structure with the following data members:**  **1. A Data**  **2. A link to the next node**  **Perform the following operations on stack using user-defined functions:**  **1. Push**  **2. Pop**  **3. Isempty**  **4. Isfull**  **5. Peep**  **Create a file which stores all values of list.** |

* **Code: -**

#include<stdio.h>

#include<conio.h>

struct Node

{

int data;

struct Node \*next;

}\*top = NULL;

void push(int);

void pop();

void display();

void isFull();

void isEmpty();

void main()

{

int choice,value;

clrscr();

printf("\n:: Stack using Linked List ::\n");

while(1){

printf("\n\*\*\*\*\*\* MENU \*\*\*\*\*\*\n");

printf("1. Push\n2. Pop\n3. Display\n4. isFull\n5. isEmpty\n6. Exit\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the value to be insert: ");

scanf("%d", &value);

push(value);

break;

case 2: pop(); break;

case 3: display(); break;

case 4: isFull(); break;

case 5: isEmpty(); break;

case 6: exit(0);

default: printf("\nWrong selection!!! Please try again!!!\n");

}

}

}

void push(int value)

{

struct Node \*newNode;

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

newNode->data = value;

if(top == NULL)

newNode->next = NULL;

else

newNode->next = top;

top = newNode;

printf("\nInsertion is Success!!!\n");

}

void pop()

{

if(top == NULL)

printf("\nStack is Empty!!!\n");

else{

struct Node \*temp = top;

printf("\nDeleted element: %d", temp->data);

top = temp->next;

free(temp);

}

}

void display()

{

if(top == NULL)

printf("\nStack is Empty!!!\n");

else{

struct Node \*temp = top;

while(temp->next != NULL){

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

temp = temp -> next;

}

printf("%d--->NULL",temp->data);

}

}

void isFull()

{

if(top<=5)

{

printf("\nStack is Full\n");

}

else

{

printf("\nStack is Not Full\n");

}

}

void isEmpty()

{

if(top==NULL)

{

printf("\nStack is Empty\n");

}

else

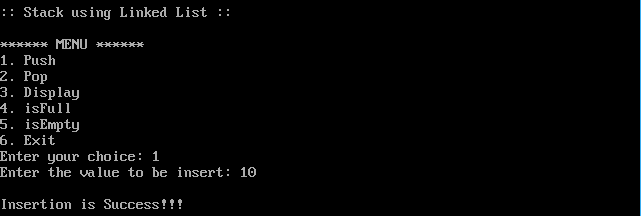
{

printf("\nStack is Not Empty\n");

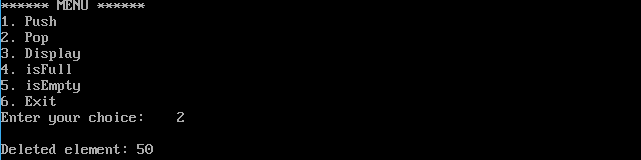
}

}

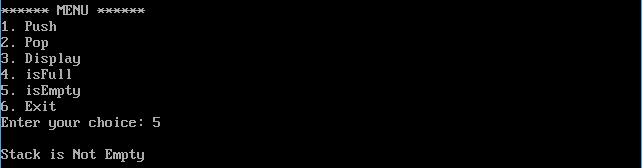
* **Output: -**







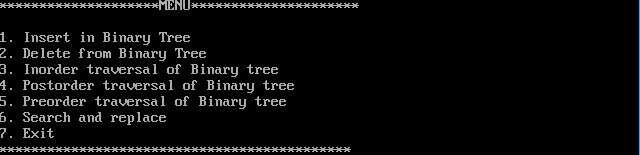


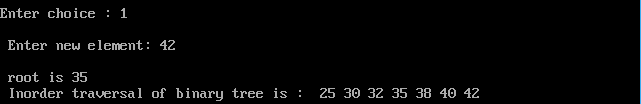
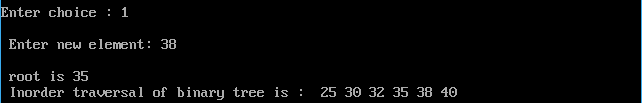
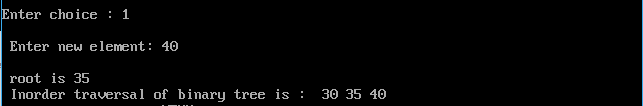
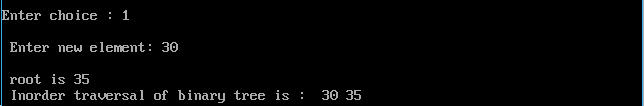


**Program: - 18**

|  |
| --- |
| **Create a user-defined structure with the following data members:**  **1. A Data**  **2. A link to the Left child**  **3. A link to the Right child**  **Perform the following operations on Binary Search Tree using recursion:**  **1. Create**  **2. Traverse (Inorder, Preorder, Postorder)**  **3. Insert**  **4. Delete**  **5. Search**  **Create a file which stores all values of traversal.** |
| * **Code: -**   #include<stdio.h>  #include<alloc.h>  #include<conio.h>  #include<stdio.h>  struct tree {  int info;  struct tree \*left;  struct tree \*right;  };  struct tree \*insert(struct tree \*,int);  void inorder(struct tree \*);  void postorder(struct tree \*);  void preorder(struct tree \*);  struct tree \*delet(struct tree \*,int);  struct tree \*search(struct tree \*);  int main(void) {  struct tree \*root;  int choice, item,item\_no;  root = NULL;  clrscr();  /\* rear = NULL;\*/  do {  do {  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\n1. Insert in Binary Tree ");  printf("\n2. Delete from Binary Tree ");  printf("\n3. Inorder traversal of Binary tree");  printf("\n4. Postorder traversal of Binary tree");  printf("\n5. Preorder traversal of Binary tree");  printf("\n6. Search and replace ");  printf("\n7. Exit ");  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  printf("\nEnter choice : ");  scanf(" %d",&choice);  if(choice<1 || choice>7)  printf("\n Invalid choice - try again");  }  while (choice<1 || choice>7);  switch(choice) {  case 1:  printf("\n Enter new element: ");  scanf("%d", &item);  root= insert(root,item);  printf("\n root is %d",root->info);  printf("\n Inorder traversal of binary tree is : ");  inorder(root);  break;  case 2:  printf("\n Enter the element to be deleted : ");  scanf(" %d",&item\_no);  root=delet(root,item\_no);  inorder(root);  break;  case 3:  printf("\n Inorder traversal of binary tree is : ");  inorder(root);  break;  case 4:  printf("\n Postorder traversal of binary tree is : ");  postorder(root);  break;  case 5:  printf("\n Preorder traversal of binary tree is : ");  preorder(root);  break;  case 6:  printf("\n Search and replace operation in binary tree ");  root=search(root);  break;  default:  printf("\n End of program ");  }  /\* end of switch \*/  }  while(choice !=7);  return(0);  }  struct tree \*insert(struct tree \*root, int x) {  if(!root) {  root=(struct tree\*)malloc(sizeof(struct tree));  root->info = x;  root->left = NULL;  root->right = NULL;  return(root);  }  if(root->info > x)  root->left = insert(root->left,x); else {  if(root->info < x)  root->right = insert(root->right,x);  }  return(root);  }  void inorder(struct tree \*root) {  if(root != NULL) {  inorder(root->left);  printf(" %d",root->info);  inorder(root->right);  }  return;  }  void postorder(struct tree \*root) {  if(root != NULL) {  postorder(root->left);  postorder(root->right);  printf(" %d",root->info);  }  return;  }  void preorder(struct tree \*root) {  if(root != NULL) {  printf(" %d",root->info);  preorder(root->left);  preorder(root->right);  }  return;  }  /\* FUNCTION TO DELETE A NODE FROM A BINARY TREE \*/  struct tree \*delet(struct tree \*ptr,int x) {  struct tree \*p1,\*p2;  if(!ptr) {  printf("\n Node not found ");  return(ptr);  } else {  if(ptr->info < x) {  ptr->right = delet(ptr->right,x);  /\*return(ptr);\*/  } else if (ptr->info >x) {  ptr->left=delet(ptr->left,x);  return ptr;  } else  /\* no. 2 else \*/ {  if(ptr->info == x)  /\* no. 2 if \*/ {  if(ptr->left == ptr->right)  /\*i.e., a leaf node\*/ {  free(ptr);  return(NULL);  } else if(ptr->left==NULL)  /\* a right subtree \*/ {  p1=ptr->right;  free(ptr);  return p1;  } else if(ptr->right==NULL)  /\* a left subtree \*/ {  p1=ptr->left;  free(ptr);  return p1;  } else {  p1=ptr->right;  p2=ptr->right;  while(p1->left != NULL)  p1=p1->left;  p1->left=ptr->left;  free(ptr);  return p2;  }  }  /\*end of no. 2 if \*/  }  /\* end of no. 2 else \*/  /\* check which path to search for a given no. \*/  }  return(ptr);  }  /\* function to search and replace an element in the binary tree \*/  struct tree \*search(struct tree \*root) {  int no,i,ino;  struct tree \*ptr;  ptr=root;  printf("\n Enter the element to be searched :");  scanf(" %d",&no);  fflush(stdin);  while(ptr) {  if(no>ptr->info)  ptr=ptr->right; else if(no<ptr->info)  ptr=ptr->left; else  break;  }  if(ptr) {  printf("\n Element %d which was searched is found and is = %d",no,ptr->info);  printf("\n Do you want replace it, press 1 for yes : ");  scanf(" %d",&i);  if(i==1) {  printf("\n Enter new element :");  scanf(" %d",&ino);  ptr->info=ino;  } else  printf("\n\t It's okay");  } else  printf("\n Element %d does not exist in the binary tree",no);  return(root);  } |

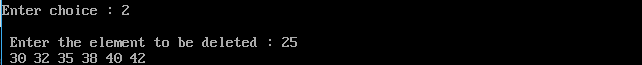
* **Output: -**

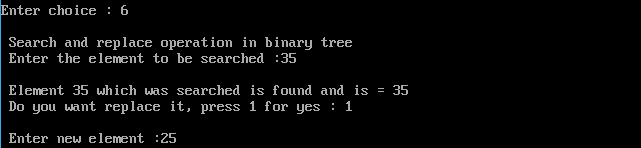












**Practical : 1**

**Create a Structure with following Data Members: 1. Integer Array 2. Size of the Array Sort the Array using various Sorting algorithms such as (i) Selection Sort (ii) Bubble Sort (iii) Twoway Merge Sort (iv) Insertion Sort (v) Quick Sort (vi) Radix Sort (vii) Heap Sort and (viii) Shell Sort. And store the sorted Array in a text file.**

* **Code :-**

#include<stdio.h>

int arr[100],n,i;

void display(){

printf("\nSorted Array...\n");

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

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

}

}

void set\_val(){

printf("Enter array limit: ");

scanf("%d",&n);

printf("Enter elements: \n");

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

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

}

}

void bubble\_sort(){

int temp,j;

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

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

if(arr[i]>arr[j]){

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

display();

}

void insertion\_sort(){

int j,temp;

int i = 0;

printf("\nAt start %d: ",i);

for(j=0;j<n;j++){

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

}

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

for(j=i;j>=0;j--){

if(arr[j]<arr[j-1]){

temp = arr[j-1];

arr[j-1]=arr[j];

arr[j]=temp;

}

}

printf("\nAfter pass %d: ",i);

for(j=0;j<n;j++){

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

}

}

printf("\nFinal Answer");

display();

}

void selection\_sort(){

int j,arr\_2[n],min,temp;

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

min=i;

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

if(arr[j]<arr[min]){

min = j;

}

}

temp = arr[i];

arr[i] = arr[min];

arr[min] = temp;

printf("\nAfter pass %d: ",i+1);

for(j=0;j<n;j++){

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

}

}

printf("\nFinal Answer");

display();

}

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there

are any \*/

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void merge\_sort(int arr[], int l, int r)

{

if (l < r)

{

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l+(r-l)/2;

//printf("\n--------\nl is %d\nr is %d\nm is %d\n---------\n",l,r,m);

// Sort first and second halves

merge\_sort(arr, l, m);

merge\_sort(arr, m+1, r);

merge(arr, l, m, r);

printf("\n--------\narray is >>> ");

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

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

}

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

}

}

/\* This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot \*/

int partition (int arr[], int low, int high)

{

int pivot = arr[high],temp; // pivot

int i = (low - 1); // Index of smaller element

for (int j = low; j <= high- 1; j++)

{

// If current element is smaller than or

// equal to pivot

if (arr[j] <= pivot)

{

i++; // increment index of smaller element

temp = arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

temp = arr[i + 1];

arr[i + 1]=arr[high];

arr[high]=temp;

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index \*/

void quick\_sort(int arr[], int low, int high)

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

int pi = partition(arr, low, high);

// Separately sort elements before

// partition and after partition

quick\_sort(arr, low, pi - 1);

quick\_sort(arr, pi + 1, high);

}

}

// A utility function to get maximum value in arr[]

int getMax(int arr[], int n)

{

int mx = arr[0];

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

if (arr[i] > mx)

mx = arr[i];

return mx;

}

// A function to do counting sort of arr[] according to

// the digit represented by exp.

void countSort(int arr[], int n, int exp)

{

int output[n]; // output array

int i, count[10] = {0};

// Store count of occurrences in count[]

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

count[ (arr[i]/exp)%10 ]++;

// Change count[i] so that count[i] now contains actual

// position of this digit in output[]

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

// Build the output array

for (i = n - 1; i >= 0; i--)

{

output[count[ (arr[i]/exp)%10 ] - 1] = arr[i];

count[ (arr[i]/exp)%10 ]--;

}

// Copy the output array to arr[], so that arr[] now

// contains sorted numbers according to current digit

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

arr[i] = output[i];

}

// The main function to that sorts arr[] of size n using

// Radix Sort

void radix\_sort()

{

// Find the maximum number to know number of digits

int m = getMax(arr, n);

// Do counting sort for every digit. Note that instead

// of passing digit number, exp is passed. exp is 10^i

// where i is current digit number

for (int exp = 1; m/exp > 0; exp \*= 10)

countSort(arr, n, exp);

}

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2\*i + 1; // left = 2\*i + 1

int r = 2\*i + 2; // right = 2\*i + 2

int temp;

// If left child is larger than root

if (l < n && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest so far

if (r < n && arr[r] > arr[largest])

largest = r;

// If largest is not root

if (largest != i)

{

temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

// main function to do heap sort

void heap\_sort()

{

// Build heap (rearrange array)

int temp;

for (i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// One by one extract an element from heap

for (i=n-1; i>=0; i--)

{

// Move current root to end

temp = arr[0];

arr[0] = arr[i];

arr[i] =temp;

// call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

/\* function to sort arr using shellSort \*/

void shell\_sort()

{

int gap,temp,j;

// Start with a big gap, then reduce the gap

for (gap = n/2; gap > 0; gap /= 2)

{

// Do a gapped insertion sort for this gap size.

// The first gap elements a[0..gap-1] are already in gapped order

// keep adding one more element until the entire array is

// gap sorted

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

{

// add a[i] to the elements that have been gap sorted

// save a[i] in temp and make a hole at position i

temp = arr[i];

// shift earlier gap-sorted elements up until the correct

// location for a[i] is found

for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)

arr[j] = arr[j - gap];

// put temp (the original a[i]) in its correct location

arr[j] = temp;

}

}

}

void save()

{

FILE \*fptr;

fptr = fopen("sort.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

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

fprintf(fptr,"\n%d",arr[i]);

}

printf("\n sort.txt is saved \n");

fclose(fptr);

}

void main(){

int choice,first\_run=1;

char again;

do

{

if(first\_run==1){

set\_val();

first\_run++;

}

else{

printf("Do you want to continue? Y to continue, Any Other to exit!! :- ");

scanf(" %c",&again);

if(again=='y' || again=='Y'){

set\_val();

}

else{

exit(0);

}

}

printf("\n SORT OPERATIONS USING ARRAY");

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

printf("\n 1.Bubble Sort\n 2.Insertion Sort \n 3.Selection Sort \n 4.Merge Sort \n 5. Quick Sort \n 6. Radix Sort \n 7. Heap Sort \n 8. Shell Sort \n 9. Exit \n");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

bubble\_sort();

save();

break;

}

case 2:

{

insertion\_sort();

save();

break;

}

case 3:

{

selection\_sort();

save();

break;

}

case 4:

{

merge\_sort(arr,0,n-1);

display();

save();

break;

}

case 5:

{

quick\_sort(arr,0,n-1);

display();

save();

break;

}

case 6:

{

radix\_sort();

display();

save();

break;

}

case 7:

{

heap\_sort();

display();

save();

break;

}

case 8:

{

shell\_sort();

display();

save();

break;

}

case 9:

{

printf("\nHave A Good Day!!!\n");

}

default:

{

printf ("\n Please Enter a Valid Choice(1/2/3/4/5/6/7/8/9)");

}

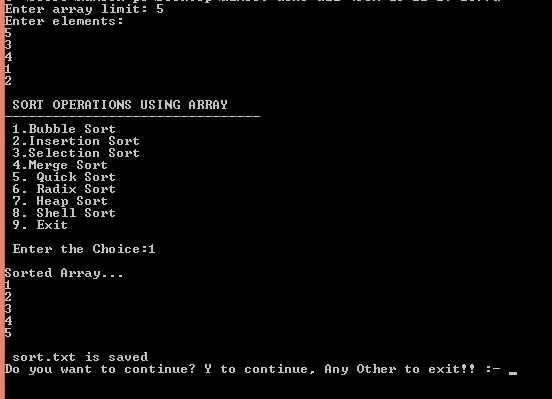
}

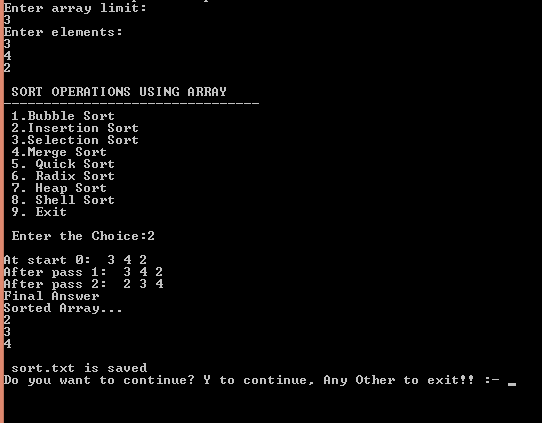
}

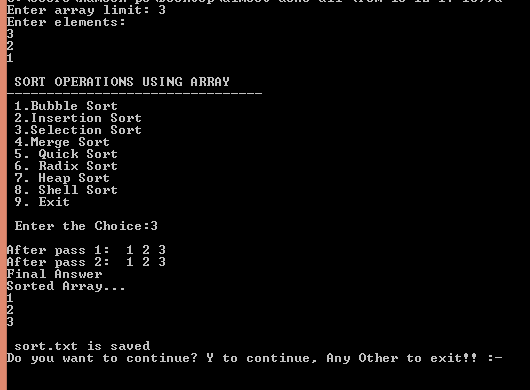
while(choice!=9);

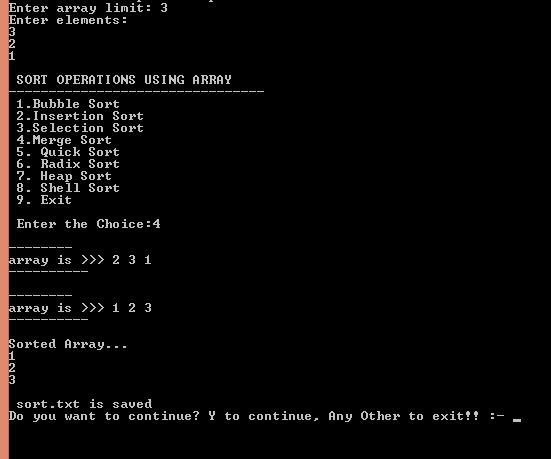
}

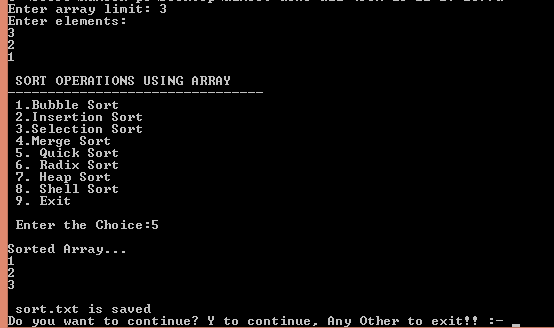
**OUTPUT :**

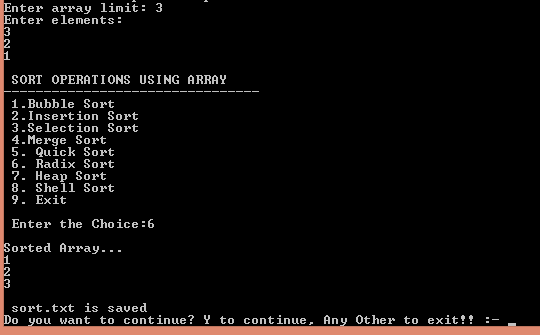


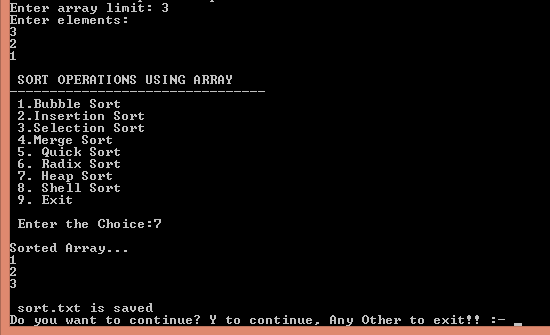


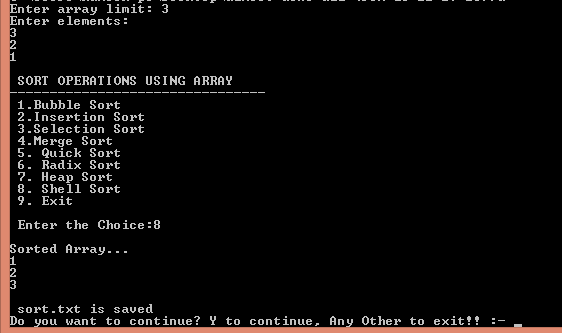


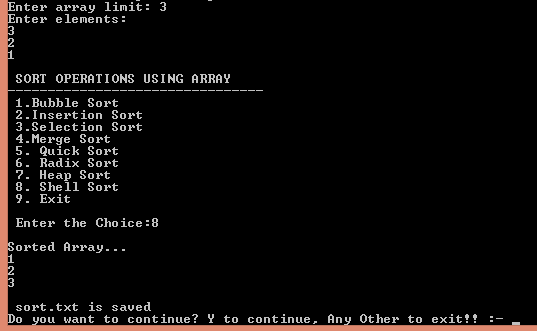












**Practical : 2**

**Create a Structure with following Data Members: 1. Integer Array 2. Size of the Array Search an element in Array using Linear (Sequential) Search and Binary Search. And Display result in file. For Sequential Search, assume (a) Unordered Array, and (b) Ordered Array and develop programs accordingly.**

**CODE :**

#include<stdio.h>

struct array{

int arr[100],n;

}array1;

int choice,num,i;

void linear();

void binary();

void sort(){

int temp,j;

for(i=0;i<array1.n;i++){

for(j=i+1;j<array1.n;j++){

if(array1.arr[i]>array1.arr[j]){

temp=array1.arr[i];

array1.arr[i]=array1.arr[j];

array1.arr[j]=temp;

}

}

}

}

int main()

{

//clrscr();

printf("\n Enter the size of Array[MAX=100]:");

scanf("%d",&array1.n);

printf("\n Enter the Elements:\n");

for(i=0;i<array1.n;i++){

scanf("%d",&array1.arr[i]);

}

printf("\n\t ARRAY SEARCHING");

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

printf("\n Enter the Number to be searched:");

scanf("%d",&num);

printf("\n\t 1.Sequential Search \n\t 2.Binary Search");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

linear();

break;

}

case 2:

{

sort();

printf("After Sorting Array... \n");

for(i=0;i<array1.n;i++){

printf("%d \n",array1.arr[i]);

}

binary();

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice... program is closing...");

}

}

return 0;

}

void binary(){

int f,l,m;

FILE \*fptr;

fptr = fopen("array\_search.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

f = 0;

l = array1.n - 1;

m = (f+l)/2;

while (f <= l) {

if (array1.arr[m] < num){

f = m + 1;

}

else if (array1.arr[m] == num) {

printf("\n Element found at position: %d", m+1);

fprintf(fptr,"\n Element found at position: %d",m+1);

break;

}

else{

l = m - 1;

}

m = (f+l)/2;

}

if (f > l){

printf("\n Element is not found");

fprintf(fptr,"\n Element is not found");

}

fclose(fptr);

}

void linear(){

int flag=0;

FILE \*fptr;

fptr = fopen("array\_search.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

for(i=0;i<array1.n;i++){

if(num==array1.arr[i]){

printf("\n Element found at position: %d",i+1);

fprintf(fptr,"\n Element found at position: %d",i+1);

flag=1;

}

}

if(flag==0){

printf("\n Element is not found");

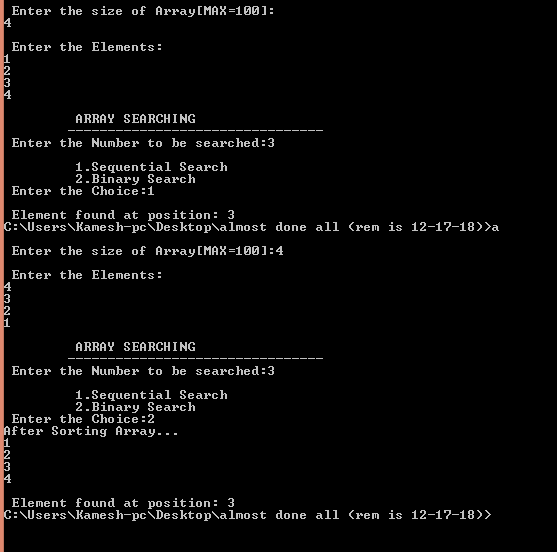
fprintf(fptr,"\n Element is not found");

}

fclose(fptr);

}

**OUTPUT :**



**Practical : 5**

**Create a Structure with following Data Members: 1. Integer Array 2. Size of the Array Sort the Array using various Sorting algorithms such as (i) Selection Sort (ii) Bubble Sort (iii) Twoway Merge Sort (iv) Insertion Sort (v) Quick Sort (vi) Radix Sort (vii) Heap Sort and (viii) Shell Sort. And store the sorted Array in a text file.**

**CODE:**

#include<stdio.h>

char stack[20];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

}

void main()

{

char exp[20];

char \*e, x;

printf("Enter the expression :: ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c",pop());

push(\*e);

}

e++;

}

while(top != -1)

{

printf("%c",pop());

}

}

OUTPUT :





**Practical : 6**

**Write a program to evaluate a postfix expression.**

**CODE:**

**#include<stdio.h>**

**int stack[20];**

**int top = -1;**

**void push(int x)**

**{**

**stack[++top] = x;**

**}**

**int pop()**

**{**

**return stack[top--];**

**}**

**int main()**

**{**

**char exp[20];**

**char \*e;**

**int n1,n2,n3,num;**

**printf("Enter the expression :: ");**

**scanf("%s",exp);**

**e = exp;**

**while(\*e != '\0')**

**{**

**if(isdigit(\*e))**

**{**

**num = \*e - 48;**

**push(num);**

**}**

**else**

**{**

**n1 = pop();**

**n2 = pop();**

**switch(\*e)**

**{**

**case '+':**

**{**

**n3 = n1 + n2;**

**break;**

**}**

**case '-':**

**{**

**n3 = n2 - n1;**

**break;**

**}**

**case '\*':**

**{**

**n3 = n1 \* n2;**

**break;**

**}**

**case '/':**

**{**

**n3 = n2 / n1;**

**break;**

**}**

**}**

**push(n3);**

**}**

**e++;**

**}**

**printf("\nThe result of expression %s = %d\n\n",exp,pop());**

**return 0;**

**}**

**OUTPUT :**



**Practical : 7**

**Create a structure with following Data members:**

**1. Integer Array**

**2. Size of the Array Search an element in a given list using Binary search by recursion. And Display result in file.**

**CODE:**

#include<stdio.h>

struct array{

int arr[100],n;

}array1;

int binary(int,int);

void save(int);

int i,num;

void sort(){

int temp,j;

for(i=0;i<array1.n;i++){

for(j=i+1;j<array1.n;j++){

if(array1.arr[i]>array1.arr[j]){

temp=array1.arr[i];

array1.arr[i]=array1.arr[j];

array1.arr[j]=temp;

}

}

}

}

void main()

{

int loc;

//clrscr();

printf("\n Enter the size of Array[MAX=100]:");

scanf("%d",&array1.n);

printf("\n Enter the Elements:\n");

for(i=0;i<array1.n;i++){

scanf("%d",&array1.arr[i]);

}

printf("\n\t ARRAY SEARCHING");

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

printf("\n Enter the Number to be searched:");

scanf("%d",&num);

sort();

printf("After Sorting Array... \n");

for(i=0;i<array1.n;i++){

printf("%d \n",array1.arr[i]);

}

loc=binary(0,array1.n);

if(loc!=0){

printf("\nElement Found At :- %d\n",loc);

save(loc);

}

else{

printf("\nElement not found...\n");

save(0);

}

}

int binary(f,l){

int m;

m = (f+l)/2;

if (l >= f)

{

if (array1.arr[m] == num) {

return m+1;

}

else if (array1.arr[m] < num){

f = m + 1;

binary(f,l);

}

else if (array1.arr[m] > num){

l = m - 1;

binary(f,l);

}

}

else{

return 0;

}

}

void save(num){

FILE \*fptr;

fptr = fopen("binary\_search\_rec.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

if(num==0){

fprintf(fptr,"\n Element found not Found");

}

else{

fprintf(fptr,"\n Element found at position: %d",num);

}

fclose(fptr);

}

**OUTPUT :**

**Practical : 8**

**Create a structure with following Data members:**

**1. Integer Array**

**2. Size of the Array Perform the following operations on Simple queue using user-defined functions: 1. Insert an element**

**2. Remove an element**

**3. Display**

**4. Isfull**

**5. Isempty Create a file which stores all values of Array.**

**CODE:**

#include<stdio.h>

struct queue{

int q[100],n;

}struct1;

int choice,start=-1,rear=-1,x,i;

void insert();

void rem();

void display();

void save();

int main()

{

//clrscr();

start=-1;

printf("\n Enter the size of Queue[MAX=100]:");

scanf("%d",&struct1.n);

do

{

printf("\n\t Queue OPERATIONS USING ARRAY");

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

printf("\n\t 1.Insert\n\t 2.Remove\n\t 3.Display\n\t 4.Save in file\n\t 5.EXIT");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

insert();

break;

}

case 2:

{

rem();

break;

}

case 3:

{

display();

break;

}

case 4:

{

save();

break;

}

case 5:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4/5)");

}

}

}

while(choice!=5);

return 0;

}

void insert()

{

if(rear==struct1.n-1)

{

printf("\n\tQueue is full");

}

else

{

printf(" Enter a value to be inserted:");

scanf("%d",&x);

rear++;

struct1.q[rear]=x;

if(start==-1){

start=0;

}

}

}

void rem()

{

if(start==-1)

{

printf("\n\t Queue is empty");

}

else

{

printf("\n\t The remd element is %d",struct1.q[start]);

if(start==rear){

start=-1;

rear=-1;

}

else{

start++;

}

}

}

void display()

{

if(start>=0)

{

printf("\n The elements in queue \n");

for(i=start;i<=rear;i++){

printf("\n%d",struct1.q[i]);

}

printf("\n Press Next Choice");

}

else

{

printf("\n The queue is empty");

}

}

void save()

{

if(start>=0)

{

FILE \*fptr;

fptr = fopen("queue.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

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

{

fprintf(fptr,"\n%d",struct1.q[i]);

}

printf("\n File is saved");

fclose(fptr);

}

else

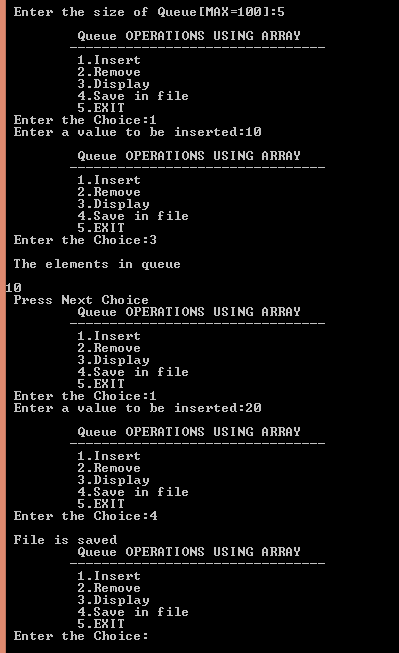
{

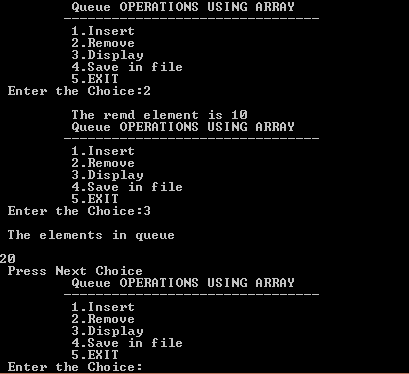
printf("\n The queue is empty");

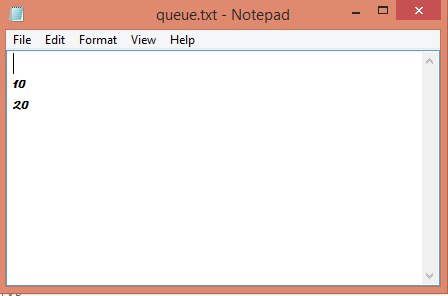
}

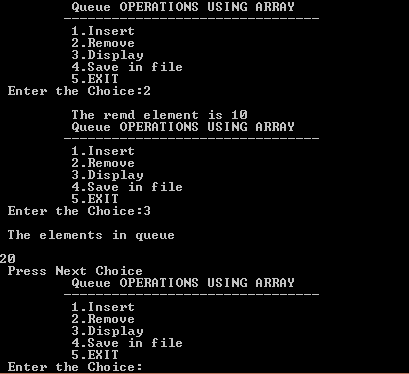
}

**OUTPUT :**









**Practical : 9**

**Create a user-defined structure with the following data members:**

**1) A Data**

**2) A link to the next node**

**Perform the following operations on Simple queue using user-defined functions:**

**1. Insert an element**

**2. Remove an element**

**3. Display**

**4. Isfull**

**5. Isempty**

**Create a file which stores all values of list.**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct queue{

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void insert();

void rem();

void display();

void save();

int main()

{

do

{

printf("\n\t Queue OPERATIONS USING LINKED LIST");

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

printf("\n\t 1.Insert\n\t 2.Remove\n\t 3.Display\n\t 4.Save in file\n\t 5.EXIT");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

insert();

break;

}

case 2:

{

rem();

break;

}

case 3:

{

display();

break;

}

case 4:

{

save();

break;

}

case 5:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4/5)");

}

}

}

while(choice!=5);

return 0;

}

void insert()

{

node \*temp;

temp = (node\*)malloc(sizeof(node));

if(temp==NULL){

printf("\n\tQueue is Full\n");

}

else{

printf("Enter a value to be inserted:");

scanf("%d",&i);

temp->data = i;

temp->next=NULL;

if(start==NULL){

start = temp;

rear = temp;

}

else{

rear->next = temp;

rear = temp;

}

}

}

void rem()

{

node \*temp;

if(start==NULL)

{

printf("\n\t Queue is Empty \n");

}

else

{

temp = start;

start = start->next;

printf("\n\t The deleted element is %d",temp->data);

free(temp);

}

}

void display()

{

node \*temp;

if(start!=NULL)

{

temp = start;

while(temp->next!=NULL){

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

temp = temp->next;

}

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

}

else

{

printf("\n The Queue is empty");

}

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("queue\_linked.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next!=NULL){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

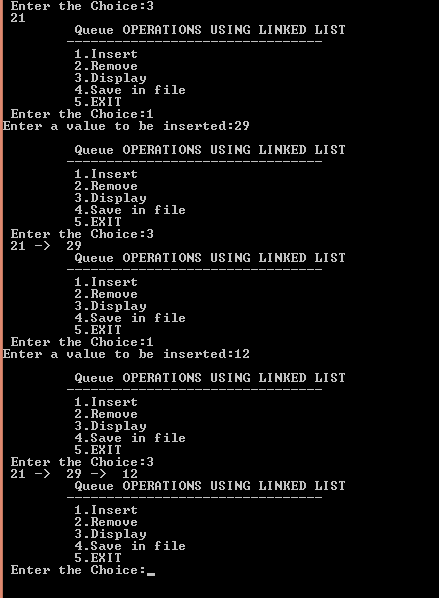
{

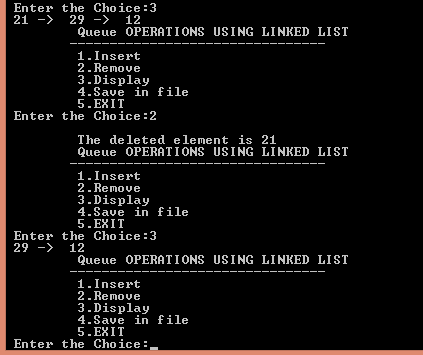
printf("\n The Queue is empty");

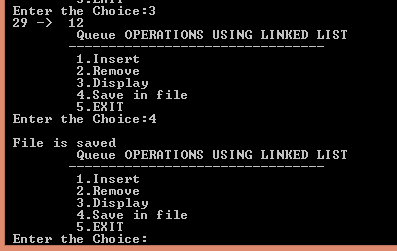
}

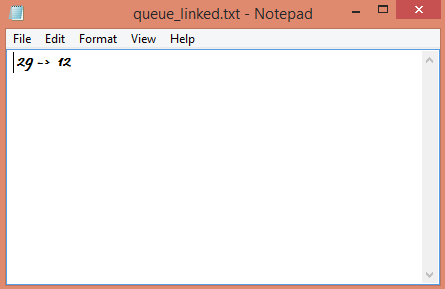
}

**OUTPUT :**









**Practical : 10**

**Create a structure with following Data members:**

1. **Integer Array**
2. **. Size of the Array Perform the following operations on Circular queue using user-defined functions:**

**1. Insert an element**

**2. Remove an element**

**3. Display**

**4. Isfull**

**5. Isempty**

**Create a file which stores all values of Array.**

**CODE:**

#include<stdio.h>

struct queue{

int q[100],n;

}struct1;

int choice,start=-1,rear=-1,x,i;

void insert();

void rem();

void display();

void save();

int main()

{

//clrscr();

start=-1;

printf("\n Enter the size of Queue[MAX=100]:");

scanf("%d",&struct1.n);

do

{

printf("\n\t Circular Queue OPERATIONS USING ARRAY");

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

printf("\n\t 1.Insert\n\t 2.Remove\n\t 3.Display\n\t 4.Save in file\n\t 5.EXIT");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

insert();

break;

}

case 2:

{

rem();

break;

}

case 3:

{

display();

break;

}

case 4:

{

save();

break;

}

case 5:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4/5)");

}

}

}

while(choice!=5);

return 0;

}

void insert()

{

if(rear==(start-1)||(start==0&&rear==(struct1.n-1)))

{

printf("\n\tQueue is full");

}

else

{

printf(" Enter a value to be inserted:");

scanf("%d",&x);

rear++;

if(start==-1){

start=0;

struct1.q[start]=x;

}

else if(rear==struct1.n){

rear=0;

struct1.q[rear]=x;

}

else{

struct1.q[rear]=x;

}

}

}

void rem()

{

if(start==-1)

{

printf("\n\t Queue is empty");

}

else

{

printf("\n\t The remd element is %d",struct1.q[start]);

if(start==rear){

start=-1;

rear=-1;

}

else{

start++;

if(start==struct1.n){

start=0;

}

}

}

}

void display()

{

if(start>=0)

{

printf("\n The elements in queue \n");

if(rear<start){

for(i=start;i<struct1.n;i++){

printf("\n%d",struct1.q[i]);

}

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

printf("\n%d",struct1.q[i]);

}

}

else{

for(i=start;i<=rear;i++){

printf("\n%d",struct1.q[i]);

}

}

printf("\n Press Next Choice");

}

else

{

printf("\n The queue is empty");

}

}

void save()

{

if(start>=0)

{

FILE \*fptr;

fptr = fopen("circular\_queue\_array.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

if(rear<start){

for(i=start;i<struct1.n;i++){

fprintf(fptr,"\n%d",struct1.q[i]);

}

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

fprintf(fptr,"\n%d",struct1.q[i]);

}

}

else{

for(i=start;i<=rear;i++){

fprintf(fptr,"\n%d",struct1.q[i]);

}

}

printf("\n File is saved");

fclose(fptr);

}

else

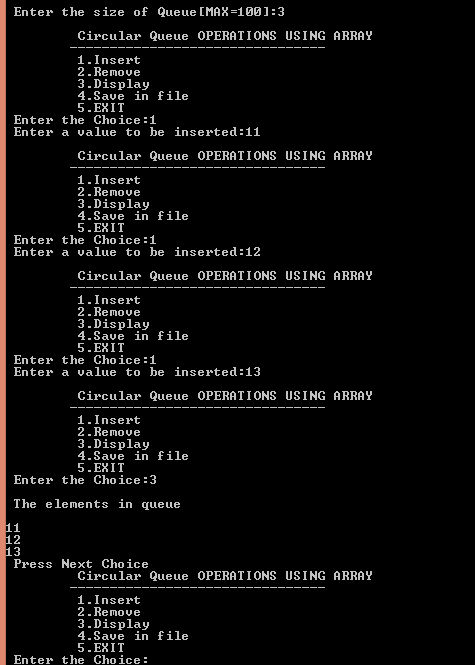
{

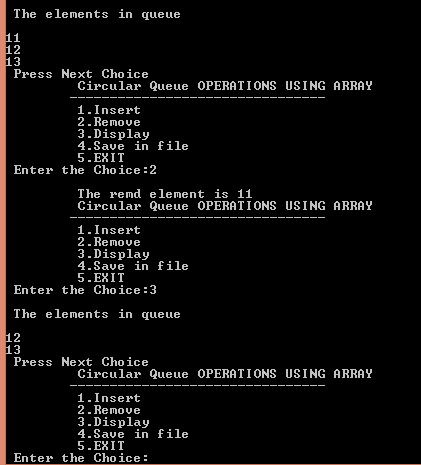
printf("\n The queue is empty");

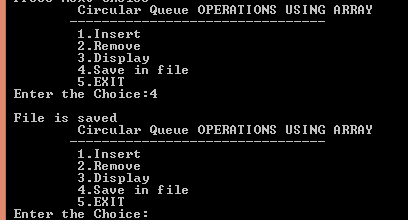
}

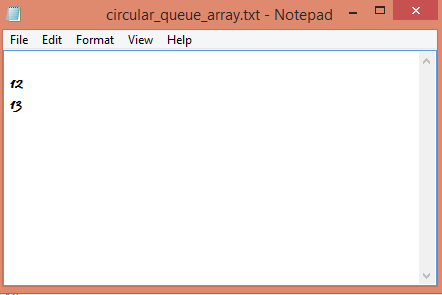
}

**OUTPUT :**









**Practical : 11**

**Create a user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node Perform the following operations on Circular queue using user-defined functions:**

**1. Insert an element**

**2. Remove an element**

**3. Display**

**4. Isfull**

**5. Isempty**

**Create a file which stores all values of list.**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct queue{

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void insert();

void rem();

void display();

void save();

int main()

{

do

{

printf("\n\t Circular Queue OPERATIONS USING Linked List");

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

printf("\n\t 1.Insert\n\t 2.Remove\n\t 3.Display\n\t 4.Save in file\n\t 5.EXIT");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

insert();

break;

}

case 2:

{

rem();

break;

}

case 3:

{

display();

break;

}

case 4:

{

save();

break;

}

case 5:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4/5)");

}

}

}

while(choice!=5);

return 0;

}

void insert()

{

node \*temp;

temp = (node\*)malloc(sizeof(node));

if(temp==NULL){

printf("\n\tQueue is Full\n");

}

else{

printf("Enter a value to be inserted:");

scanf("%d",&i);

temp->data = i;

temp->next=start;

if(start==NULL){

start = temp;

rear = temp;

}

else{

rear->next = temp;

rear = temp;

}

}

}

void rem()

{

node \*temp;

if(start==NULL)

{

printf("\n\t Queue is Empty \n");

}

else

{

temp = start;

start = start->next;

rear->next=start;

printf("\n\t The deleted element is %d",temp->data);

free(temp);

}

}

void display()

{

node \*temp;

if(start!=NULL)

{

temp = start;

while(temp->next->data!=start->data){

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

temp = temp->next;

}

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

}

else

{

printf("\n The Queue is empty");

}

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("circular\_queue\_linked.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next->data!=start->data){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

{

printf("\n The Queue is empty");

}

}

**OUTPUT :**

**Practical : 12**

**Create a user-defined structure with the following data members:**

**1. A Co-efficient**

**2. A Exponent**

**3. A link to the next node Perform the following operations on Singly list using user-defined functions:**

**1. Create**

**2. Display**

**3. Addition**

**4. Multiplication**

**Create a file which stores all values of list.**

**CODE:**

/\*Polynomial- Representation, Addition, Multiplication\*/

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

struct poly

{

int coeff;

int exp;

struct poly \*next;

}\*head1=NULL,\*head2=NULL,\*head3=NULL,\*head4=NULL,\*temp,\*ptr;

void create();

void makenode(int,int);

struct poly \*insertend(struct poly \*);

void display(struct poly \*);

struct poly \*addtwopoly(struct poly \*,struct poly \*,struct poly \*);

struct poly \*subtwopoly(struct poly \*,struct poly \*,struct poly \*);

struct poly \*multwopoly(struct poly \*,struct poly \*,struct poly \*);

struct poly \*dispose(struct poly \*);

int search(struct poly \*,int);

void main()

{

int ch,coefficient,exponent;

int listno;

while(1)

{

clrscr();

printf(“ntMenu”);

printf(“nt1. Create First Polynomial.”);

printf(“nt2. Display First Polynomial.”);

printf(“nt3. Create Second Polynomial.”);

printf(“nt4. Display Second Polynomial.”);

printf(“nt5. Add Two Polynomials.”);

printf(“nt6. Display Result of Addition.”);

printf(“nt7. Subtract Two Polynomials.”);

printf(“nt8. Display Result of Subtraction.”);

printf(“nt9. Multiply Two Polynomials.”);

printf(“nt10. Display Result of Product.”);

printf(“nt11. Dispose List.”);

printf(“nt12. Exit”);

printf(“nntEnter your choice?”);

scanf(“%d”,&ch);

switch(ch)

{

case 1:

printf(“nGenerating first polynomial:”);

printf(“nEnter coefficient?”);

scanf(“%d”,&coefficient);

printf(“nEnter exponent?”);

scanf(“%d”,&exponent);

makenode(coefficient,exponent);

head1 = insertend(head1);

break;

case 2:

display(head1);

break;

case 3:

printf(“nGenerating second polynomial:”);

printf(“nEnter coefficient?”);

scanf(“%d”,&coefficient);

printf(“nEnter exponent?”);

scanf(“%d”,&exponent);

makenode(coefficient,exponent);

head2 = insertend(head2);

break;

case 4:

display(head2);

break;

case 5:

printf(“nDisposing result list.”);

head3=dispose(head3);

head3=addtwopoly(head1,head2,head3);

printf(“Addition successfully done!”);

break;

case 6:

display(head3);

break;

case 7:

head3=dispose(head3);

head3=subtwopoly(head1,head2,head3);

printf(“Subtraction successfully done!”);

getch();

break;

case 8:

display(head3);

break;

case 9:

head3=dispose(head3);

head4=dispose(head4);

head4=multwopoly(head1,head2,head3);

break;

case 10:

display(head4);

break;

case 11:

printf(“Enter list number to dispose(1 to 4)?”);

scanf(“%d”,&listno);

if(listno==1)

head1=dispose(head1);

else if(listno==2)

head2=dispose(head2);

else if(listno==3)

head3=dispose(head3);

else if(listno==4)

head4=dispose(head4);

else

printf(“Invalid number specified.”);

break;

case 12:

exit(0);

default:

printf(“Invalid Choice!”);

break;

}

}

}

void create()

{

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

if(ptr==NULL)

{

printf(“Memory Allocation Error!”);

exit(1);

}

}

void makenode(int c,int e)

{

create();

ptr->coeff = c;

ptr->exp = e;

ptr->next = NULL;

}

struct poly \*insertend(struct poly \*head)

{

if(head==NULL)

head = ptr;

else

{

temp=head;

while(temp->next != NULL)

temp = temp->next;

temp->next = ptr;

}

return head;

}

void display(struct poly \*head)

{

if(head==NULL)

printf(“List is empty!”);

else

{

temp=head;

while(temp!=NULL)

{

printf(“(%d,%d)->”,temp->coeff,temp->exp);

temp=temp->next;

}

printf(“bb “);

}

getch();

}

struct poly \*addtwopoly(struct poly \*h1,struct poly \*h2,struct poly \*h3)

{

/\*

(5,3)->(6,1) + (7,3)->(9,2) = (12,3)->(6,1)->(9,2)

\*/

struct poly \*temp1,\*temp2,\*temp3;

temp1=h1;

temp2=h2;

while(temp1!=NULL || temp2!=NULL)

{

if(temp1->exp==temp2->exp)

{

makenode(temp1->coeff+temp2->coeff,temp1->exp);

h3=insertend(h3);

}

else

{

makenode(temp1->coeff,temp1->exp);

h3=insertend(h3);

makenode(temp2->coeff,temp2->exp);

h3=insertend(h3);

}

temp1=temp1->next;

temp2=temp2->next;

}

if(temp1==NULL && temp2!=NULL)

{

while(temp2!=NULL)

{

makenode(temp2->coeff,temp2->exp);

h3=insertend(h3);

temp2=temp2->next;

}

}

if(temp2==NULL && temp1!=NULL)

{

while(temp1!=NULL)

{

makenode(temp2->coeff,temp2->exp);

h3=insertend(h3);

temp1=temp1->next;

}

}

return h3;

}

struct poly \*subtwopoly(struct poly \*h1,struct poly \*h2,struct poly \*h3)

{

/\*

(5,3)->(6,1) – (7,3)->(9,2) = (-2,3)+(6,1)-(9,2)

\*/

struct poly \*temp1,\*temp2,\*temp3;

temp1=h1;

temp2=h2;

while(temp1!=NULL || temp2!=NULL)

{

if(temp1->exp==temp2->exp)

{

makenode(temp1->coeff-temp2->coeff,temp1->exp);

h3=insertend(h3);

}

else

{

makenode(temp1->coeff,temp1->exp);

h3=insertend(h3);

makenode(-temp2->coeff,temp2->exp);

h3=insertend(h3);

}

temp1=temp1->next;

temp2=temp2->next;

}

if(temp1==NULL && temp2!=NULL)

{

while(temp2!=NULL)

{

makenode(temp2->coeff,temp2->exp);

h3=insertend(h3);

temp2=temp2->next;

}

}

if(temp2==NULL && temp1!=NULL)

{

while(temp1!=NULL)

{

makenode(-temp2->coeff,temp2->exp);

h3=insertend(h3);

temp1=temp1->next;

}

}

return h3;

}

struct poly \*multwopoly(struct poly \*h1,struct poly \*h2,struct poly \*h3)

{

/\*

h1=(5,3)->(6,1) \* h2=(7,3)->(9,2)

(5,3)->(7,3),(9,2) = (35,6),(45,5)

(6,1)->(7,3),(9,2) = (42,4),(54,3)

h3->(35,6)->(45,5)->(42,4)->(54,3)

(35,6)+(45,5)+(42,4)+(54,3)=Result

\*/

int res=0;

struct poly \*temp1,\*temp2,\*temp3;

printf(“nDisplaying First Polynomial:ntt”);

display(h1);

printf(“nDisplaying Second Polynomial:ntt”);

display(h2);

temp1=h1;

while(temp1!=NULL)

{

temp2=h2;

while(temp2!=NULL)

{

makenode(temp1->coeff\*temp2->coeff,temp1->exp+temp2->exp);

h3=insertend(h3);

temp2=temp2->next;

}

temp1=temp1->next;

}

printf(“nDisplaying Initial Result of Product:ntt”);

display(h3);

getch();

temp1=h3;

while(temp1!=NULL)

{temp2=temp1->next;

res=0;

while(temp2!=NULL)

{

if(temp1->exp==temp2->exp)

res += temp2->coeff;

temp2=temp2->next;

}

if(search(head4,temp1->exp)==1)

{

makenode(res+temp1->coeff,temp1->exp);

head4=insertend(head4);

}

temp1=temp1->next;

}

return head4;

}

int search(struct poly \*h,int val)

{

struct poly \*tmp;

tmp=h;

while(tmp!=NULL)

{if(tmp->exp==val)

return 0;

tmp=tmp->next;

}

return 1;

}

struct poly \*dispose(struct poly \*list)

{

if(list==NULL)

{

printf(“List is already empty.”);

return list;

}

else

{

temp=list;

while(list!=NULL)

{

free(temp);

list=list->next;

temp=list;

}

return list;

}

}

**OUTPUT :**

**Practical : 13**

**Create a user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node**

**Perform the following operations on list using user-defined functions:**

**1. Create a list**

**2. Traverse the whole list\**

**3. Delete first node**

**4. Delete last node**

**5. Delete a node before specified data**

**6. Insert at first position**

**7. Insert at last position**

**8. Insert a node before specified data**

**9. Insert a node at specified position**

**10. Count**

**11. Copy**

**12. Merge two list**

**13. Reverse**

**14. Search**

**15. Sort**

**Create a file which stores all values of list.**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct queue{

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void create(){

node \*temp;

printf("\nCreating List\nEnter Data (Enter -1 to stop)...\n");

scanf("%d",&i);

while(i!=-1){

temp = (node\*)malloc(sizeof(node));

temp->next=NULL;

temp->data=i;

if(start==NULL){

start=temp;

rear=temp;

}

else{

rear->next=temp;

rear=temp;

}

scanf("%d",&i);

}

printf("\n-1 encountered\n");

}

void traverse(){

node \*temp;

if(start==NULL){

printf("\nCreate A LIST first...");

create();

}

else{

temp=start;

printf("\nThe list as below\n");

while(temp->next!=NULL){

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

temp=temp->next;

}

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

}

}

void delete\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

start=start->next;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

while(temp->next!=NULL){

rear=temp;

temp=temp->next;

}

rear->next=NULL;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_before(){

node \*temp,\*prev;

int flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("\nEnter value before to be deleted\n");

scanf("%d",&i);

temp=start;

while (temp!=NULL){

count++;

if(temp->data==i){

flag=1;

break;

}

temp=temp->next;

}

if(flag==1){

if(count==1){

printf("Cannot delete before element of %d\n",i);

}

else if(count==2){

temp=start;

start=start->next;

free(temp);

}

else{

temp=start;

while(temp!=NULL){

prev=temp;

temp=temp->next;

if(temp->next->data==i){

prev->next=temp->next;

free(temp);

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=start;

start=temp;

}

void insert\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

rear->next=temp;

rear=temp;

}

void insert\_before(){

node \*temp,\*trav,\*prev;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter before value to insert: ");

scanf("%d",&val);

temp=start;

while (temp!=NULL){

count++;

if(temp->data==val){

flag=1;

break;

}

temp=temp->next;

}

if(flag==1){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

if(count==1){

temp->next=start;

start=temp;

}

else{

trav=start;

while(trav!=NULL){

prev=trav;

trav=trav->next;

if(trav->data==val){

prev->next=temp;

temp->next=trav;

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_specified(){

node \*temp,\*trav,\*prev;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter position to insert: ");

scanf("%d",&val);

val--;

if(val<1){

insert\_first();

}

else{

trav=start;

while(count!=val && trav->next!=NULL){

count++;

prev=trav;

trav=trav->next;

}

if(count==val){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=trav;

prev->next=temp;

}

else{

insert\_last();

}

}

}

void count(){

node \*temp;

int count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

count++;

temp=temp->next;

}

printf("\nTotal Nodes: %d\n",count);

}

void reverse\_list(){

node \*temp;

int count=0,\*rev\_array;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

count++;

temp=temp->next;

}

rev\_array=(int \*)malloc(count\*sizeof(int));

temp=start;

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

\*(rev\_array+i)=temp->data;

temp=temp->next;

}

//printf("%d\n%d\n",sizeof(rev\_array),count);

temp=start;

for(i=count-1;i>=0;i--){

temp->data=\*(rev\_array+i);

rear=temp;

temp=temp->next;

}

}

void search(){

node \*temp;

int count=0,flag=0,val;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter value to search: ");

scanf("%d",&val);

temp=start;

while(temp!=NULL){

count++;

if(val==temp->data){

flag=1;

break;

}

temp=temp->next;

}

if(flag==0){

printf("Entered value is not found...\n");

}

else{

printf("%d is found at position: %d\n",val,count);

}

}

void sort\_list(){

node \*temp,\*temp2;

int temp3;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

temp2=temp->next;

while(temp2!=NULL){

//printf("%d\t%d\n",temp->data,temp2->data);

if(temp2->data<temp->data){

temp3=temp2->data;

temp2->data=temp->data;

temp->data=temp3;

}

temp2=temp2->next;

}

temp=temp->next;

}

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("custom\_list.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next!=NULL){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

{

printf("\n The Queue is empty");

}

}

int main()

{

do

{

printf("\n\t Queue OPERATIONS USING LINKED LIST");

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

printf("\n\t 1. Create a list\n\t 2. Traverse the whole list\n\t 3. Delete first node\n\t 4. Delete last node\n\t 5. Delete a node before specified data\n\t 6. Insert at first position\n\t 7. Insert at last position\n\t 8. Insert a node before specified data\n\t 9. Insert a node at specified position\n\t 10. Count\n\t 11. Copy (NOT DONE YET)\n\t 12. Merge two list (NOT DONE YET)\n\t 13. Reverse\n\t 14. Search\n\t 15. Sort\n\t 16. Save\n\t 17. Exit");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

create();

break;

}

case 2:

{

traverse();

break;

}

case 3:

{

delete\_first();

break;

}

case 4:

{

delete\_last();

break;

}

case 5:

{

delete\_before();

break;

}

case 6:

{

insert\_first();

break;

}

case 7:

{

insert\_last();

break;

}

case 8:

{

insert\_before();

break;

}

case 9:

{

insert\_specified();

break;

}

case 10:

{

count();

break;

}

case 11:

{

break;

}

case 12:

{

break;

}

case 13:

{

reverse\_list();

break;

}

case 14:

{

search();

break;

}

case 15:

{

sort\_list();

break;

}

case 16:

{

save();

break;

}

case 17:

{

printf("\nExit point...\n");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1 to 17)");

}

}

}

while(choice!=17);

return 0;

}

**OUTPUT :**

**Practical : 14**

**Create a user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node**

**Perform the following operations on Circular list using user-defined functions:**

**1. Create a list**

**2. Traverse the whole list\**

**3. Delete first node**

**4. Delete last node**

**5. Delete a node before specified data**

**6. Insert at first position**

**7. Insert at last position**

**8. Insert a node before specified data**

**9. Insert a node at specified position**

**10. Count**

**11. Copy**

**12. Merge two list**

**13. Reverse**

**14. Search**

**15. Sort Create a file which stores all values of list.**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct queue{

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void create(){

node \*temp;

printf("\nCreating List\nEnter Data (Enter -1 to stop)...\n");

scanf("%d",&i);

while(i!=-1){

temp = (node\*)malloc(sizeof(node));

temp->next=NULL;

temp->data=i;

if(start==NULL){

start=temp;

rear=temp;

}

else{

rear->next=temp;

rear=temp;

rear->next=start;

}

scanf("%d",&i);

}

printf("\n-1 encountered\n");

}

void traverse(){

node \*temp;

if(start==NULL){

printf("\nCreate A LIST first...");

create();

}

else{

temp = start;

while(temp->next->data!=start->data){

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

temp = temp->next;

}

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

}

}

void delete\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

start=start->next;

rear->next=start;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_last(){

node \*temp,\*prev;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

while(temp->next->data!=start->data){

rear=temp;

temp=temp->next;

}

rear->next=start;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_before(){

node \*temp,\*prev;

int flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("\nEnter value before to be deleted\n");

scanf("%d",&i);

temp=start;

do{

count++;

if(temp->data==i){

flag=1;

break;

}

temp=temp->next;

}while (temp->data!=start->data);

if(flag==1){

if(count==1){

printf("Cannot delete before element of %d\n",i);

}

else if(count==2){

temp=start;

start=start->next;

rear->next=start;

free(temp);

}

else{

temp=start;

do{

prev=temp;

temp=temp->next;

if(temp->next->data==i){

prev->next=temp->next;

free(temp);

break;

}

}while(temp->data!=start->data);

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=start;

start=temp;

rear->next=start;

}

void insert\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=start;

rear->next=temp;

rear=temp;

}

void insert\_before(){

node \*temp,\*trav,\*prev;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter before value to insert: ");

scanf("%d",&val);

temp=start;

do{

count++;

if(temp->data==val){

flag=1;

break;

}

temp=temp->next;

}while (temp->data!=start->data);

if(flag==1){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

if(count==1){

temp->next=start;

start=temp;

rear->next=start;

}

else{

trav=start;

do{

prev=trav;

trav=trav->next;

if(trav->data==val){

prev->next=temp;

temp->next=trav;

break;

}

}while(trav->data!=start->data);

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_specified(){

node \*temp,\*trav,\*prev;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter position to insert: ");

scanf("%d",&val);

val--;

if(val<1){

insert\_first();

}

else{

trav=start;

while(count!=val && trav->next->data!=start->data){

count++;

prev=trav;

trav=trav->next;

}

if(count==val){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=trav;

prev->next=temp;

}

else{

insert\_last();

}

}

}

void count(){

node \*temp;

int count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

count++;

temp=temp->next;

}while(temp->data!=start->data);

printf("\nTotal Nodes: %d\n",count);

}

void reverse\_list(){

node \*temp;

int count=0,\*rev\_array;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

count++;

temp=temp->next;

}while(temp->data!=start->data);

rev\_array=(int \*)malloc(count\*sizeof(int));

temp=start;

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

\*(rev\_array+i)=temp->data;

temp=temp->next;

}

//printf("%d\n%d\n",sizeof(rev\_array),count);

temp=start;

for(i=count-1;i>=0;i--){

temp->data=\*(rev\_array+i);

rear=temp;

temp=temp->next;

}

}

void search(){

node \*temp;

int count=0,flag=0,val;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter value to search: ");

scanf("%d",&val);

temp=start;

do{

count++;

if(val==temp->data){

flag=1;

break;

}

temp=temp->next;

}while(temp->data!=start->data);

if(flag==0){

printf("Entered value is not found...\n");

}

else{

printf("%d is found at position: %d\n",val,count);

}

}

void sort\_list(){

node \*temp,\*temp2;

int temp3;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

temp2=temp->next;

while(temp2->data!=start->data){

//printf("%d\t%d\n",temp->data,temp2->data);

if(temp2->data<temp->data){

temp3=temp2->data;

temp2->data=temp->data;

temp->data=temp3;

}

temp2=temp2->next;

}

temp=temp->next;

}while(temp->data!=start->data);

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("custom\_circular\_list.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next->data!=start->data){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

{

printf("\n The Queue is empty");

}

}

int main()

{

do

{

printf("\n\t Queue OPERATIONS USING LINKED LIST");

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

printf("\n\t 1. Create a list\n\t 2. Traverse the whole list\n\t 3. Delete first node\n\t 4. Delete last node\n\t 5. Delete a node before specified data\n\t 6. Insert at first position\n\t 7. Insert at last position\n\t 8. Insert a node before specified data\n\t 9. Insert a node at specified position\n\t 10. Count\n\t 11. Copy (NOT DONE YET)\n\t 12. Merge two list (NOT DONE YET)\n\t 13. Reverse\n\t 14. Search\n\t 15. Sort\n\t 16. Save\n\t 17. Exit");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

create();

break;

}

case 2:

{

traverse();

break;

}

case 3:

{

delete\_first();

break;

}

case 4:

{

delete\_last();

break;

}

case 5:

{

delete\_before();

break;

}

case 6:

{

insert\_first();

break;

}

case 7:

{

insert\_last();

break;

}

case 8:

{

insert\_before();

break;

}

case 9:

{

insert\_specified();

break;

}

case 10:

{

count();

break;

}

case 11:

{

break;

}

case 12:

{

break;

}

case 13:

{

reverse\_list();

break;

}

case 14:

{

search();

break;

}

case 15:

{

sort\_list();

break;

}

case 16:

{

save();

break;

}

case 17:

{

printf("\nExit point...\n");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1 to 17)");

}

}

}

while(choice!=17);

return 0;

}

**OUTPUT :**

**Practical : 15**

**Create a user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node**

**3. A link to the previous node**

**Perform the following operations on the doubly-linked list using user-defined functions:**

**1. Create a list**

**2. Traverse the whole list\**

**3. Delete first node**

**4. Delete last node**

**5. Delete a node before specified data**

**6. Insert at first position**

**7. Insert at last position**

**8. Insert a node before specified data**

**9. Insert a node at specified position**

**10. Count**

**11. Copy**

**12. Merge two list**

**13. Reverse**

**14. Search**

**15. Sort Create a file which stores all values of list.**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct queue{

struct queue \*prev;

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void create(){

node \*temp;

printf("\nCreating List\nEnter Data (Enter -1 to stop)...\n");

scanf("%d",&i);

while(i!=-1){

temp = (node\*)malloc(sizeof(node));

temp->next=NULL;

temp->prev=NULL;

temp->data=i;

if(start==NULL){

start=temp;

rear=temp;

}

else{

temp->prev=rear;

rear->next=temp;

rear=temp;

}

scanf("%d",&i);

}

printf("\n-1 encountered\n");

}

void traverse(){

node \*temp;

if(start==NULL){

printf("\nCreate A LIST first...");

create();

}

else{

temp=start;

printf("\nThe list as below\n");

while(temp->next!=NULL){

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

temp=temp->next;

}

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

}

}

void delete\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

start=start->next;

start->prev=NULL;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

while(temp->next!=NULL){

rear=temp;

temp=temp->next;

}

rear->next=NULL;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_before(){

node \*temp,\*pre;

int flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("\nEnter value before to be deleted\n");

scanf("%d",&i);

temp=start;

while (temp!=NULL){

count++;

if(temp->data==i){

flag=1;

break;

}

temp=temp->next;

}

if(flag==1){

if(count==1){

printf("Cannot delete before element of %d\n",i);

}

else if(count==2){

temp=start;

start=start->next;

start->prev=NULL;

free(temp);

}

else{

temp=start;

while(temp!=NULL){

pre=temp;

temp=temp->next;

if(temp->next->data==i){

pre->next=temp->next;

temp->next->prev=pre;

free(temp);

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->prev=NULL;

temp->next=start;

start->prev=temp;

start=temp;

}

void insert\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

rear->next=temp;

temp->prev=rear;

rear=temp;

}

void insert\_before(){

node \*temp,\*trav,\*pre;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter before value to insert: ");

scanf("%d",&val);

temp=start;

while (temp!=NULL){

count++;

if(temp->data==val){

flag=1;

break;

}

temp=temp->next;

}

if(flag==1){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

temp->prev=NULL;

if(count==1){

temp->next=start;

start->prev=temp;

start=temp;

}

else{

trav=start;

while(trav!=NULL){

pre=trav;

trav=trav->next;

if(trav->data==val){

pre->next=temp;

temp->prev=pre;

temp->next=trav;

trav->prev=temp;

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_specified(){

node \*temp,\*trav,\*pre;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter position to insert: ");

scanf("%d",&val);

val--;

if(val<1){

insert\_first();

}

else{

trav=start;

while(count!=val && trav->next!=NULL){

count++;

pre=trav;

trav=trav->next;

}

if(count==val){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=trav;

trav->prev=temp;

pre->next=temp;

temp->prev=pre;

}

else{

insert\_last();

}

}

}

void count(){

node \*temp;

int count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

count++;

temp=temp->next;

}

printf("\nTotal Nodes: %d\n",count);

}

void reverse\_list(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=rear;

while(temp->prev!=NULL){

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

temp=temp->prev;

}

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

}

void search(){

node \*temp;

int count=0,flag=0,val;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter value to search: ");

scanf("%d",&val);

temp=start;

while(temp!=NULL){

count++;

if(val==temp->data){

flag=1;

break;

}

temp=temp->next;

}

if(flag==0){

printf("Entered value is not found...\n");

}

else{

printf("%d is found at position: %d\n",val,count);

}

}

void sort\_list(){

node \*temp,\*temp2;

int temp3;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

while(temp!=NULL){

temp2=temp->next;

while(temp2!=NULL){

//printf("%d\t%d\n",temp->data,temp2->data);

if(temp2->data<temp->data){

temp3=temp2->data;

temp2->data=temp->data;

temp->data=temp3;

}

temp2=temp2->next;

}

temp=temp->next;

}

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("custom\_circular\_list.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next!=NULL){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

{

printf("\n The Queue is empty");

}

}

int main()

{

do

{

printf("\n\t Queue OPERATIONS USING LINKED LIST");

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

printf("\n\t 1. Create a list\n\t 2. Traverse the whole list\n\t 3. Delete first node\n\t 4. Delete last node\n\t 5. Delete a node before specified data\n\t 6. Insert at first position\n\t 7. Insert at last position\n\t 8. Insert a node before specified data\n\t 9. Insert a node at specified position\n\t 10. Count\n\t 11. Copy (NOT DONE YET)\n\t 12. Merge two list (NOT DONE YET)\n\t 13. Reverse\n\t 14. Search\n\t 15. Sort\n\t 16. Save\n\t 17. Exit");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

create();

break;

}

case 2:

{

traverse();

break;

}

case 3:

{

delete\_first();

break;

}

case 4:

{

delete\_last();

break;

}

case 5:

{

delete\_before();

break;

}

case 6:

{

insert\_first();

break;

}

case 7:

{

insert\_last();

break;

}

case 8:

{

insert\_before();

break;

}

case 9:

{

insert\_specified();

break;

}

case 10:

{

count();

break;

}

case 11:

{

break;

}

case 12:

{

break;

}

case 13:

{

reverse\_list();

break;

}

case 14:

{

search();

break;

}

case 15:

{

sort\_list();

break;

}

case 16:

{

save();

break;

}

case 17:

{

printf("\nExit point...\n");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1 to 17)");

}

}

}

while(choice!=17);

return 0;

}

**OUTPUT :**

**Practical : 16**

**Create a user-defined structure with the following data members:**

**1. A Data**

**2. A link to the next node 3. A link to the previous node**

**Perform the following operations on doubly-linked Circular list using user defined functions:**

**1. Create a list**

**2. Traverse the whole list\**

**3. Delete first node**

**4. Delete last node**

**5. Delete a node before specified data**

**6. Insert at first position**

**7. Insert at last position**

**8. Insert a node before specified data**

**9. Insert a node at specified position**

**10. Count**

**11. Copy**

**12. Merge two list**

**13. Reverse**

**14. Search**

**15. Sort Create a file which stores all values of list.**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct queue{

struct queue \*prev;

int data;

struct queue \*next;

};

typedef struct queue node;

node \*start=NULL,\*rear=NULL;

int choice,i;

void create(){

node \*temp;

printf("\nCreating List\nEnter Data (Enter -1 to stop)...\n");

scanf("%d",&i);

while(i!=-1){

temp = (node\*)malloc(sizeof(node));

temp->next=NULL;

temp->prev=NULL;

temp->data=i;

if(start==NULL){

start=temp;

rear=temp;

}

else{

temp->prev=rear;

rear->next=temp;

temp->next=start;

start->prev=temp;

rear=temp;

}

scanf("%d",&i);

}

printf("\n-1 encountered\n");

}

void traverse(){

node \*temp;

if(start==NULL){

printf("\nCreate A LIST first...");

create();

}

else{

temp=start;

printf("\nThe list as below\n");

while(temp->next->data!=start->data){

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

temp=temp->next;

}

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

}

}

void delete\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=start;

start=start->next;

start->prev=rear;

rear->next=start;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

}

else{

temp=rear;

rear=rear->prev;

rear->next=start;

start->prev=rear;

printf("\nDeleted element is %d\n",temp->data);

free(temp);

}

}

void delete\_before(){

node \*temp,\*pre;

int flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("\nEnter value before to be deleted\n");

scanf("%d",&i);

temp=start;

do{

count++;

if(temp->data==i){

flag=1;

break;

}

temp=temp->next;

}while (temp->data!=start->data);

if(flag==1){

if(count==1){

printf("Cannot delete before element of %d\n",i);

}

else if(count==2){

temp=start;

start=start->next;

start->prev=rear;

rear->next=start;

free(temp);

}

else{

temp=start;

while(temp!=NULL){

pre=temp;

temp=temp->next;

if(temp->next->data==i){

pre->next=temp->next;

temp->next->prev=pre;

free(temp);

break;

}

}

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_first(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->prev=rear;

temp->next=start;

start->prev=temp;

rear->next=temp;

start=temp;

}

void insert\_last(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=start;

rear->next=temp;

temp->prev=rear;

start->prev=temp;

rear=temp;

}

void insert\_before(){

node \*temp,\*trav,\*pre;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter before value to insert: ");

scanf("%d",&val);

temp=start;

do{

count++;

if(temp->data==val){

flag=1;

break;

}

temp=temp->next;

}while (temp->data!=start->data);

if(flag==1){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=NULL;

temp->prev=NULL;

if(count==1){

temp->next=start;

start->prev=temp;

rear->next=temp;

temp->prev=rear;

start=temp;

}

else{

trav=start;

do{

pre=trav;

trav=trav->next;

if(trav->data==val){

pre->next=temp;

temp->prev=pre;

temp->next=trav;

trav->prev=temp;

break;

}

}while(trav->data!=start->data);

}

}

else{

printf("\nValue is not found\n");

}

}

void insert\_specified(){

node \*temp,\*trav,\*pre;

int val,flag=0,count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter position to insert: ");

scanf("%d",&val);

val--;

if(val<1){

insert\_first();

}

else{

trav=start;

while(count!=val && trav->next->data!=start->data){

count++;

pre=trav;

trav=trav->next;

}

if(count==val){

temp = (node\*)malloc(sizeof(node));

printf("Enter value to insert: ");

scanf("%d",&i);

temp->data=i;

temp->next=trav;

trav->prev=temp;

pre->next=temp;

temp->prev=pre;

}

else{

insert\_last();

}

}

}

void count(){

node \*temp;

int count=0;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

count++;

temp=temp->next;

}while(temp->data!=start->data);

printf("\nTotal Nodes: %d\n",count);

}

void reverse\_list(){

node \*temp;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=rear;

while(temp->prev->data!=rear->data){

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

temp=temp->prev;

}

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

}

void search(){

node \*temp;

int count=0,flag=0,val;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

printf("Enter value to search: ");

scanf("%d",&val);

temp=start;

do{

count++;

if(val==temp->data){

flag=1;

break;

}

temp=temp->next;

}while(temp->data!=start->data);

if(flag==0){

printf("Entered value is not found...\n");

}

else{

printf("%d is found at position: %d\n",val,count);

}

}

void sort\_list(){

node \*temp,\*temp2;

int temp3;

if(start==NULL){

printf("\nThere is no list created\n");

return;

}

temp=start;

do{

temp2=temp->next;

while(temp2!=NULL){

//printf("%d\t%d\n",temp->data,temp2->data);

if(temp2->data<temp->data){

temp3=temp2->data;

temp2->data=temp->data;

temp->data=temp3;

}

temp2=temp2->next;

}

temp=temp->next;

}while(temp->data!=start->data);

}

void save()

{

node \*temp;

if(start!=NULL)

{

FILE \*fptr;

fptr = fopen("custom\_circular\_doublylist.txt","w");

if(fptr == NULL)

{

printf("Error in file!");

exit(0);

}

temp=start;

while(temp->next->data!=start->data){

fprintf(fptr," %d -> ",temp->data);

temp = temp->next;

}

fprintf(fptr," %d",temp->data);

printf("\n File is saved");

fclose(fptr);

}

else

{

printf("\n The Queue is empty");

}

}

int main()

{

do

{

printf("\n\t Queue OPERATIONS USING LINKED LIST");

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

printf("\n\t 1. Create a list\n\t 2. Traverse the whole list\n\t 3. Delete first node\n\t 4. Delete last node\n\t 5. Delete a node before specified data\n\t 6. Insert at first position\n\t 7. Insert at last position\n\t 8. Insert a node before specified data\n\t 9. Insert a node at specified position\n\t 10. Count\n\t 11. Copy (NOT DONE YET)\n\t 12. Merge two list (NOT DONE YET)\n\t 13. Reverse\n\t 14. Search\n\t 15. Sort\n\t 16. Save\n\t 17. Exit");

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

create();

break;

}

case 2:

{

traverse();

break;

}

case 3:

{

delete\_first();

break;

}

case 4:

{

delete\_last();

break;

}

case 5:

{

delete\_before();

break;

}

case 6:

{

insert\_first();

break;

}

case 7:

{

insert\_last();

break;

}

case 8:

{

insert\_before();

break;

}

case 9:

{

insert\_specified();

break;

}

case 10:

{

count();

break;

}

case 11:

{

break;

}

case 12:

{

break;

}

case 13:

{

reverse\_list();

break;

}

case 14:

{

search();

break;

}

case 15:

{

sort\_list();

break;

}

case 16:

{

save();

break;

}

case 17:

{

printf("\nExit point...\n");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1 to 17)");

}

}

}

while(choice!=17);

return 0;

}

**OUTPUT :-**

**Practical : 17**

**Write a program to represent an undirected graph using the adjacency matrix to implement the graph and your program be menu driven allowing the user the following options:**

**1. Create graph**

**2. Insert an edge**

**3. Print Adjacency Matrix**

**4. List all vertices that are adjacent to a specified vertex.**

**5. Print out vertices using depth first search**

**6. Print out vertices using breadth first search**

**7. Exit program**

**CODE**

//... A Program to represent a Graph by using an Adjacency Matrix method

#include <stdio.h>

#include <stdlib.h>

int dir\_graph();

int undir\_graph();

int read\_graph(int adj\_mat[50][50], int n );

void main()

{

   int option;

   do

   {

        printf("\n A Program to represent a Graph by using an ");

  printf("Adjacency Matrix method \n ");

  printf("\n 1. Directed Graph ");

  printf("\n 2. Un-Directed Graph ");

  printf("\n 3. Exit ");

  printf("\n\n Select a proper option : ");

  scanf("%d", &option);

  switch(option)

  {

    case 1 : dir\_graph();

       break;

    case 2 : undir\_graph();

       break;

    case 3 : exit(0);

  } // switch

    }while(1);

}

int dir\_graph()

{

    int adj\_mat[50][50];

    int n;

    int in\_deg, out\_deg, i, j;

    printf("\n How Many Vertices ? : ");

    scanf("%d", &n);

    read\_graph(adj\_mat, n);

    printf("\n Vertex \t In\_Degree \t Out\_Degree \t Total\_Degree ");

    for (i = 1; i <= n ; i++ )

    {

        in\_deg = out\_deg = 0;

 for ( j = 1 ; j <= n ; j++ )

 {

            if ( adj\_mat[j][i] == 1 )

                in\_deg++;

 }

        for ( j = 1 ; j <= n ; j++ )

            if (adj\_mat[i][j] == 1 )

                out\_deg++;

            printf("\n\n %5d\t\t\t%d\t\t%d\t\t%d\n\n",i,in\_deg,out\_deg,in\_deg+out\_deg);

    }

    return;

}

int undir\_graph()

{

    int adj\_mat[50][50];

    int deg, i, j, n;

    printf("\n How Many Vertices ? : ");

    scanf("%d", &n);

    read\_graph(adj\_mat, n);

    printf("\n Vertex \t Degree ");

    for ( i = 1 ; i <= n ; i++ )

    {

        deg = 0;

        for ( j = 1 ; j <= n ; j++ )

            if ( adj\_mat[i][j] == 1)

                deg++;

        printf("\n\n %5d \t\t %d\n\n", i, deg);

    }

    return;

}

int read\_graph ( int adj\_mat[50][50], int n )

{

    int i, j;

    char reply;

    for ( i = 1 ; i <= n ; i++ )

    {

        for ( j = 1 ; j <= n ; j++ )

        {

            if ( i == j )

            {

                adj\_mat[i][j] = 0;

  continue;

            }

            printf("\n Vertices %d & %d are Adjacent ? (Y/N) :",i,j);

            scanf("%c", &reply);

            if ( reply == 'y' || reply == 'Y' )

                adj\_mat[i][j] = 1;

            else

                adj\_mat[i][j] = 0;

 }

    }

    return;

}

</stdlib.h></stdio.h>

**OUTPUT**