**LAB CYCLE -1**

**SUBMITTED BY**

**ANUSHREE**

**MCA139**

**PROGRAM -1**

**Q1:** Merge two sorted arrays and store in a third array.

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 50

void merge(int arr[], int p, int q, int r)

{

int i, j, k;

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], R[n2];

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

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

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

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

i = 0;

j = 0;

k = p;

while (i < n1 && j < n2)

{

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

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

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

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int arr[], int n)

{

int i;

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

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

printf("\n");

}

int main()

{

int n , i;

int arr[MAX];

printf("Size of the Array:");

scanf("%d",&n);

printf("enter elements:");

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

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

mergeSort(arr, 0, n - 1);

printf("\nSorted array is \n");

printArray(arr,n);

}

**PROGRAM-2**

**Q2:** Circular Queue

**PROGRAM CODE:**

#include<stdio.h>

#define SIZE 10

void enQueue(int);

void deQueue();

void display();

void exit();

int queue[SIZE],front=-1,rear=-1;

int main()

{

int value,choice;

while(1)

{

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

printf("\n1.Insertion\n2.Deletion\n3.Display\n4.Exit");

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

scanf("%d",&choice);

switch(choice)

{

case 1: printf("enter value to be inserted:");

scanf("%d",&value);

enQueue(value);

break;

case 2: deQueue();

break;

case 3: display();

break;

case 4: exit(0);

break;

default:

printf("\n wrong selection");

}

}

return 0;

}

void enQueue(int value)

{

if(rear==SIZE-1)

printf("\n Queue is full!!!Insertion not possible!!!");

else

{

if(front==-1)

front=0;

rear++;

queue[rear]=value;

printf("\n Insertion success!!!");

}

}

void deQueue()

{

if(front==-1)

printf("\n Queue is empty!!!Deletion not possible!!!");

else

{

printf("\n Deleted:%d",queue[front]);

front++;

if(front>rear)

front=rear=-1;

}

}

void display()

{

if(rear==-1)

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

else

{

int i;

printf("\n Queue elements are:\n");

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

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

}

}

**PROGRAM-3**

**Q3:** Singly Linked Stack

**PROGRAM CODE:**

#include<stdio.h>

#include<stdlib.h>

void push();

void pop();

void display();

struct node

{

int value;

struct node\*next;

};

struct node\*head;

int main()

{

int choice=0;

printf("\n\*\*\*\*\*\*\*\*\*Linked Stack\*\*\*\*\*\*\*\*\*\n");

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

while(choice!=4)

{

printf("\n\n Choose one from the below options...\n");

printf("\n1.push\n2.pop\n3.display\n4.exit\n");

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

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("Existing...");

break;

}

default:

{

printf("Please enter valid choice:");

}

};

}

}

void push ()

{

int value;

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

if(ptr==NULL)

{

printf("not able to push the element");

}

else

{

printf("Enter the value:");

scanf("%d",&value);

if(head==NULL)

{

ptr->value=value;

ptr->next=NULL;

head=ptr;

}

else

{

ptr->value=value;

ptr->next=head;

head=ptr;

}

printf("item pushed");

}

}

void pop()

{

int item;

struct node\*ptr;

if(head==NULL)

{

printf("Underflow");

}

else

{

item=head->value;

ptr=head;

head=head->next;

free(ptr);

printf("item popped");

}

}

void display()

{

int i;

struct node\*ptr;

ptr=head;

if(ptr==NULL)

{

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

}

else

{

printf("printing stack elements:\n");

while(ptr!=NULL)

{

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

ptr=ptr->next;

}

}

}

**PROGRAM-4**

**Q4:** Doubly Linked List Implementation

**PROGRAM CODE:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

struct node \*prev;

int n;

struct node \*next;

}\*h,\*temp,\*temp1,\*temp2,\*temp4;

void insert1();

void insert2();

void insert3();

void traversebeg();

void traverseend(int);

void sort();

void search();

void update();

void delete();

int count = 0;

int main()

{

int ch;

h = NULL;

temp = temp1 = NULL;

printf("\n 1 - Insert at beginning");

printf("\n 2 - Insert at end");

printf("\n 3 - Insert at position i");

printf("\n 4 - Delete at i");

printf("\n 5 - Display from beginning");

printf("\n 6 - Display from end");

printf("\n 7 - Search for element");

printf("\n 8 - Sort the list");

printf("\n 9 - Update an element");

printf("\n 10 - Exit");

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert1();

break;

case 2:

insert2();

break;

case 3:

insert3();

break;

case 4:

delete();

break;

case 5:

traversebeg();

break;

case 6:

temp2 = h;

if (temp2 == NULL)

printf("\n Error : List empty to display ");

else

{

printf("\n Reverse order of linked list is : ");

traverseend(temp2->n);

}

break;

case 7:

search();

break;

case 8:

sort();

break;

case 9:

update();

break;

case 10:

exit(0);

default:

printf("\n Wrong choice menu");

}

}

}

void create()

{

int data;

temp =(struct node \*)malloc(1\*sizeof(struct node));

temp->prev = NULL;

temp->next = NULL;

printf("\n Enter value to node : ");

scanf("%d", &data);

temp->n = data;

count++;

}

void insert1()

{

if (h == NULL)

{

create();

h = temp;

temp1 = h;

}

else

{

create();

temp->next = h;

h->prev = temp;

h = temp;

}

}

void insert2()

{

if (h == NULL)

{

create();

h = temp;

temp1 = h;

}

else

{

create();

temp1->next = temp;

temp->prev = temp1;

temp1 = temp;

}

}

void insert3()

{

int pos, i = 2;

printf("\n Enter position to be inserted : ");

scanf("%d", &pos);

temp2 = h;

if ((pos < 1) || (pos >= count + 1))

{

printf("\n Position out of range to insert");

return;

}

if ((h == NULL) && (pos != 1))

{

printf("\n Empty list cannot insert other than 1st position");

return;

}

if ((h == NULL) && (pos == 1))

{

create();

h = temp;

temp1 = h;

return;

}

else

{

while (i < pos)

{

temp2 = temp2->next;

i++;

}

create();

temp->prev = temp2;

temp->next = temp2->next;

temp2->next->prev = temp;

temp2->next = temp;

}

}

void delete()

{

int i = 1, pos;

printf("\n Enter position to be deleted : ");

scanf("%d", &pos);

temp2 = h;

if ((pos < 1) || (pos >= count + 1))

{

printf("\n Error : Position out of range to delete");

return;

}

if (h == NULL)

{

printf("\n Error : Empty list no elements to delete");

return;

}

else

{

while (i < pos)

{

temp2 = temp2->next;

i++;

}

if (i == 1)

{

if (temp2->next == NULL)

{

printf("Node deleted from list");

free(temp2);

temp2 = h = NULL;

return;

}

}

if (temp2->next == NULL)

{

temp2->prev->next = NULL;

free(temp2);

printf("Node deleted from list");

return;

}

temp2->next->prev = temp2->prev;

if (i != 1)

temp2->prev->next = temp2->next;

if (i == 1)

h = temp2->next;

printf("\n Node deleted");

free(temp2);

}

count--;

}

void traversebeg()

{

temp2 = h;

if (temp2 == NULL)

{

printf("List empty to display \n");

return;

}

printf("\n Linked list elements from begining : ");

while (temp2->next != NULL)

{

printf(" %d ", temp2->n);

temp2 = temp2->next;

}

printf(" %d ", temp2->n);

}

void traverseend(int i)

{

if (temp2 != NULL)

{

i = temp2->n;

temp2 = temp2->next;

traverseend(i);

printf(" %d ", i);

}

}

void search()

{

int data, count = 0;

temp2 = h;

if (temp2 == NULL)

{

printf("\n Error : List empty to search for data");

return;

}

printf("\n Enter value to search : ");

scanf("%d", &data);

while (temp2 != NULL)

{

if (temp2->n == data)

{

printf("\n Data found in %d position",count + 1);

return;

}

else

temp2 = temp2->next;

count++;

}

printf("\n Error : %d not found in list", data);

}

void update()

{

int data, data1;

printf("\n Enter node data to be updated : ");

scanf("%d", &data);

printf("\n Enter new data : ");

scanf("%d", &data1);

temp2 = h;

if (temp2 == NULL)

{

printf("\n Error : List empty no node to update");

return;

}

while (temp2 != NULL)

{

if (temp2->n == data)

{

temp2->n = data1;

traversebeg();

return;

}

else

temp2 = temp2->next;

}

printf("\n Error : %d not found in list to update", data);

}

void sort()

{

int i,j,x;

temp2 = h;

temp4 = h;

if (temp2 == NULL)

{

printf("\n List empty to sort");

return;

}

for (temp2 = h; temp2 != NULL; temp2 = temp2->next)

{

for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)

{

if (temp2->n > temp4->n)

{

x = temp2->n;

temp2->n = temp4->n;

temp4->n = x;

}

}

}

traversebeg();

}

**PROGRAM-5**

**Q5:** Implement Binary Search Tree.

**PROGRAM CODE:**

#include <stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*l;

struct node \*r;

}\*root=NULL,\*temp=NULL,\*t1,\*t2;

void insert();

void create();

void search(struct node \*t);

void search1(struct node \*t,int data);

void inorder(struct node \*t);

void delete();

void delete1();

int smallest(struct node \*t);

int largest(struct node \*t);

int flag = 1;

int main()

{

int ch;

printf("\nOPERATIONS ---");

printf("\n1 - Insert an element into tree\n");

printf("2 - Inorder Traversal\n");

printf("3 - Delete a node \n");

printf("4 - Exit\n");

do

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:inorder(root);

break;

case 3:delete();

break;

case 6:printf("\nInvalid option\n");

exit(0);

default :

printf("Wrong choice, Please enter correct choice ");

break;

}

}while(ch<4);

}

void insert()

{

create();

if (root == NULL)

root = temp;

else

search(root);

}

void create()

{

int data;

printf("Enter data of node to be inserted : ");

scanf("%d", &data);

temp = (struct node \*)malloc(1\*sizeof(struct node));

temp->data = data;

temp->l = temp->r = NULL;

}

void search(struct node \*t)

{

if ((temp->data > t->data) && (t->r != NULL))

search(t->r);

else if ((temp->data> t->data) && (t->r == NULL))

t->r = temp;

else if ((temp->data< t->data) && (t->l != NULL))

search(t->l);

else if ((temp->data < t->data) && (t->l == NULL))

t->l = temp;

}

void inorder(struct node \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display");

return;

}

if (t->l != NULL)

inorder(t->l);

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

if (t->r != NULL)

inorder(t->r);

}

void delete()

{

int data;

if (root == NULL)

{

printf("No elements in a tree to delete");

return;

}

printf("Enter the data to be deleted : ");

scanf("%d", &data);

t1 = root;

t2 = root;

search1(root, data);

}

void search1(struct node \*t, int data)

{

if ((data>t->data))

{

t1 = t;

search1(t->r, data);

}

else if ((data < t->data))

{

t1 = t;

search1(t->l, data);

}

else if ((data==t->data))

{

delete1(t);

}

}

void delete1(struct node \*t)

{

int k;

if ((t->l == NULL) && (t->r == NULL))

{

if (t1->l == t)

{

t1->l = NULL;

}

else

{

t1->r = NULL;

}

t = NULL;

free(t);

return;

}

else if ((t->r == NULL))

{

if (t1 == t)

{

root = t->l;

t1 = root;

}

else if (t1->l == t)

{

t1->l = t->l;

}

else

{

t1->r = t->l;

}

t = NULL;

free(t);

return;

}

else if (t->l == NULL)

{

if (t1 == t)

{

root = t->r;

t1 = root;

}

else if (t1->r == t)

t1->r = t->r;

else

t1->l = t->r;

t == NULL;

free(t);

return;

}

else if ((t->l != NULL) && (t->r != NULL))

{

t2 = root;

if (t->r != NULL)

{

k = smallest(t->r);

flag = 1;

}

else

{

k =largest(t->l);

flag = 2;

}

search1(root, k);

t->data = k;

}

}

int smallest(struct node \*t)

{

t2 = t;

if (t->l != NULL)

{

t2 = t;

return(smallest(t->l));

}

else

return (t->data);

}

int largest(struct node \*t)

{

if (t->r != NULL)

{

t2 = t;

return(largest(t->r));

}

else

return(t->data);

}