ASSIGNMENT - 1 (BRIDGE COURSE) Date: 24 July 2022

Roll number: m22cs007 Name: A.Jyothi

**1(a). Single linked list**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* next;

}\*head = NULL;

void displayLinkedList(struct node\*);

void addBegin(struct node\*\*,int);

void addAfter(struct node\*,int,int);

void addEnd(struct node\*\*,int);

void deleteNode(struct node\*\*,int);

int main(){

struct node\* head = NULL;

while(1){

int c;

printf("Single Linked List.");

printf("1.Insert Begin\n 2.Insert After\n 3.Insert End.\n 4.Delete\n 5.Display.\n");

scanf("%d",&c);

switch(c){

case 1: int data;

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

scanf("%d",&data);

addBegin(&head,data);

break;

case 2: printf("Enter the data to be inserted:\n");

scanf("%d",&data);

int key;

printf("Please enter the key after which to be inserted:\n");

scanf("%d",&key);

addAfter(head,key,data);

break;

case 3: printf("Enter the data to be inserted:\n");

scanf("%d",&data);

addEnd(&head,data);

break;

case 4: printf("Enter the data to be deleted.\n");

scanf("%d",&data);

deleteNode(&head,data);

break;

case 5: displayLinkedList(head);

break;

default: exit(1);

}

}

}

//This method accepts double pointer as the head reference is going to change.

void addBegin(struct node\*\* head, int data){

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

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

}

//Given a key where the new node should be inserted after.

void addAfter(struct node\* head, int key, int data){

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

newNode->data = data;

struct node\* temp= head;

//traverse along the linked list until the key is reached.

while(temp!= NULL){

if(temp->data == key){//If key is reached, break out of while loop.

break;

}

temp = temp->next;

}

if(temp==NULL){//IF the list doesn't have the given key, temp reaches the end and loop breaks.

printf("Cannot insert a new node after %d as there is no %d in linkedlist\n",key,key);

return;

}

//new node next should be the given key node's next.

newNode->next = temp->next;

//given node's next should be new node.

temp->next = newNode;

}

//Accepts double pointer, as sometimes the list might be empty. Then adding an item changes head ref.

void addEnd(struct node\*\* head, int data){

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

newNode->data = data;

newNode->next = NULL;

struct node\* temp = \*head;

if(temp==NULL){//If the linkedlist is empty, head should be newnode and return.

\*head = newNode;

return;

}

while(temp->next!= NULL){//Traverse until temp reaches last node of the linked list.

temp = temp->next;

}

temp->next = newNode;

}

//Accepting double pointer because, we may delete the first node and head reference would change.

void deleteNode(struct node\*\* head, int key){

struct node\* temp = \*head;

if(temp== NULL){

printf("Linked List is empty.");

return;

}

if(temp->data == key){//Deleting at begin case.if directly head data is key, then should change head.

\*head = temp->next;

temp->next = NULL;

free(temp);

return;

}

struct node\* prev = NULL;

while(temp!= NULL && temp->data != key ){//Traverse till the temp data not equal to key.

prev = temp;

temp =temp->next;

}

if(temp==NULL){//If key is not in linked list, temp goes till end breaks.

printf("The given key doesn't exists. Please enter valid.\n");

return;

}

// Temp data is key, which is to be deleted.

prev->next = temp->next;

temp->next = NULL;

free(temp);

}

void displayLinkedList(struct node\* head){

struct node\* temp = head;

printf("The LinkedList is as follows\n");

//Traversing the linkedlist till the end.

while(temp!=NULL){

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

temp = temp->next;

}

printf("NULL\n");

}

**1(b). Double linked list**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* prev;

struct node\* next;

}\*head = NULL;

void addBegin(struct node\*\*,int);

void addAfter(struct node\*,int,int);

void addEnd(struct node\*\*,int);

void deleteNode(struct node\*\*,int);

void displayDoubleLinkedList(struct node\*);

int main(){

struct node\* head = NULL;

while(1){

int c;

printf("Double Linked List.");

printf("1.Insert Begin\n 2.Insert After\n 3.Insert End.\n 4.Delete\n 5.Display.\n");

scanf("%d",&c);

switch(c){

case 1: int data;

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

scanf("%d",&data);

addBegin(&head,data);

break;

case 2: printf("Enter the data to be inserted:\n");

scanf("%d",&data);

int key;

printf("Please enter the key after which to be inserted:\n");

scanf("%d",&key);

addAfter(head,key,data);

break;

case 3: printf("Enter the data to be inserted:\n");

scanf("%d",&data);

addEnd(&head,data);

break;

case 4: printf("Enter the data to be deleted.\n");

scanf("%d",&data);

deleteNode(&head,data);

break;

case 5: displayDoubleLinkedList(head);

break;

default: exit(1);

}

}

}

//This method accepts double pointer as the head reference is going to change.

void addBegin(struct node\*\* head, int data){

struct node\* temp = \*head;

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

newNode->data = data;

newNode->next = temp;

newNode->prev = NULL;//Created a newnode and put next as head and prev as null;

if(temp!=NULL){//When linkedlist is empty, temp will be null.

temp->prev = newNode;

}

\*head = newNode;

}

//Given a key where the new node should be inserted after.

void addAfter(struct node\* head, int key, int data){

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

newNode->data = data;

struct node\* temp= head;

//traverse along the linked list until the key is reached.

while(temp!= NULL){

if(temp->data == key){//If key is reached, break out of while loop.

break;

}

temp = temp->next;

}

if(temp==NULL){//IF the list doesn't have the given key, temp reaches the end and loop breaks.

printf("Cannot insert a new node after %d as there is no %d in linkedlist\n",key,key);

return;

}

//new node next should be the given key node's next.

newNode->next = temp->next;

//temp node's next node's prev should be newnode.

temp->next->prev = newNode;

//given node's next should be new node.

temp->next = newNode;

//new node prev should be temp.

newNode->prev = temp;

}

//Accepts double pointer, as sometimes the list might be empty. Then adding an item changes head ref.

void addEnd(struct node\*\* head, int data){

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

newNode->data = data;

newNode->next = NULL;

struct node\* temp = \*head;

if(temp==NULL){//If the linkedlist is empty, head should be newnode and return.

\*head = newNode;

return;

}

while(temp->next!= NULL){//Traverse until temp reaches last node of the linked list.

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

//Accepting double pointer because, we may delete the first node and head reference would change.

void deleteNode(struct node\*\* head, int key){

struct node\* temp = \*head;

if(temp== NULL){

printf("Linked List is empty.");

return;

}

if(temp->data == key){//Deleting at begin case.if directly head data is key, then should change head.

struct node\* deletingNode = temp;

temp = temp->next;

deletingNode->next = NULL;

temp->prev = NULL;

\*head = temp;

free(temp);

return;

}

struct node\* prev = NULL;

while(temp!= NULL && temp->data != key ){//Traverse till the temp data not equal to key.

prev = temp;

temp =temp->next;

}

if(temp==NULL){//If key is not in linked list, temp goes till end breaks.

printf("The given key doesn't exists. Please enter valid.\n");

return;

}

//Deleting node is temp. Hence temp's next's prev should be previous node.

temp->next->prev = prev;

// Temp data is key, which is to be deleted.

prev->next = temp->next;

temp->next = NULL;

free(temp);

}

void displayDoubleLinkedList(struct node\* head){

struct node\* temp = head;

struct node\* prev = NULL;

printf("The Double LinkedList is as follows\n");

//Traversing the linkedlist till the end.

while(temp!=NULL){

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

prev = temp;

temp = temp->next;

}

printf("NULL\n");

printf("Reverse Traversal:\n");

while(prev!= NULL){

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

prev = prev->prev;

}

printf("NULL\n");

}

**1(C). Circular linked list**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* next;

}\*head = NULL;

void addBegin(struct node\*\*,int);

void addAfter(struct node\*\*,int,int);

void addEnd(struct node\*\*,int);

void deleteNode(struct node\*\*,int);

void displayCircularLinkedList(struct node\*);

int main(){

struct node\* head = NULL;

while(1){

int c;

printf("Circlular Linked List.");

printf("1.Insert \n 2.Insert After\n 3.Insert End.\n 4.Delete\n 5.Display.\n");

scanf("%d",&c);

switch(c){

case 1: int data;

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

scanf("%d",&data);

addBegin(&head,data);

break;

case 2: printf("Enter the data to be inserted:\n");

scanf("%d",&data);

int key;

printf("Please enter the key after which to be inserted:\n");

scanf("%d",&key);

addAfter(&head,key,data);

break;

case 3: printf("Enter the data to be inserted:\n");

scanf("%d",&data);

addEnd(&head,data);

break;

case 4: printf("Enter the data to be deleted.\n");

scanf("%d",&data);

deleteNode(&head,data);

break;

case 5: displayCircularLinkedList(head);

break;

default: exit(1);

}

}

}

void addBegin(struct node\*\* last, int data){

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

newNode->data = data;

//Empty case.

if((\*last) == NULL){

newNode->next = newNode;

\*last = newNode;

return;

}

//Newnode should be after last node.

newNode->next = (\*last)->next;

(\*last)->next = newNode;

}

//Given a key where the new node should be inserted after.

void addAfter(struct node\*\* last, int key, int data){

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

newNode->data = data;

struct node\* temp= (\*last)->next;

//traverse along the linked list until the key is reached.

do{

if(temp->data == key){//If key is reached, break out of while loop.

break;

}

temp = temp->next;

}while(temp!=(\*last)->next);

//new node next should be the given key node's next.

newNode->next = temp->next;

//given node's next should be new node.

temp->next = newNode;

}

//Accepts double pointer, as sometimes the list might be empty. Then adding an item changes head ref.

void addEnd(struct node\*\* last, int data){

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

newNode->data = data;

newNode->next = NULL;

if(\*(last)==NULL){//If the linkedlist is empty, head should be newnode and return.

newNode->next = newNode;

\*last = newNode;

return;

}

newNode->next = (\*last)->next;

(\*last)->next = newNode;

(\*last)= newNode;

}

void deleteNode(struct node\*\* head, int key){

struct node\* temp = \*head;

if(temp== NULL){

printf("Linked List is empty.");

return;

}

struct node\* prev = NULL;

while(temp!= NULL && temp->data != key ){//Traverse till the temp data not equal to key.

if(temp->next == \*(head)){

printf("Node not found\n");

return;

}

prev = temp;

temp =temp->next;

}

if(temp->next == \*(head)){//Single node case.

\*head = NULL;

free(temp);

return;

}

if(temp == \*head){//If first node to be deleted case.

prev = \*head;

while(prev->next!= \*head){

prev = prev->next;

}

\*head = temp->next;

prev->next = \*head;

free(temp);

return;

}

else if(temp->next == \*head && temp == \*head){

prev->next = \*head;

free(temp);

}

else{

prev->next = temp->next;

free(temp);

}

// Temp data is key, which is to be deleted.

prev->next = temp->next;

temp->next = NULL;

free(temp);

}

void displayCircularLinkedList(struct node\* head){

struct node\* temp = head;

printf("The LinkedList is as follows\n");

if(temp==NULL){

printf("Linked List is empty\n");

return;

}

//Traversing the linkedlist till the end.

do{

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

temp = temp->next;

}while(temp!=head);

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

}

**2(a) Reverse a list**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* next;

}\*head = NULL;

void displayLinkedList(struct node\*);

void reverseofaLinkedList(struct node\*\*);

int main()

{

int n;

printf("Reversing a Single Linked List\n");

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

printf("Please enter the number of nodes you want to create in the single linkedlist.\n");

scanf("%d",&n);

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

int d;

printf("Enter the data for first Node:\n");

scanf("%d",&d);

head->data = d;

head->next = NULL;

struct node\* temp = head;

int i;

for(i = 2; i<=n; i++){

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

int newData;

printf("Enter the data part for the %d th node\n",i);

scanf("%d",&newData);

newNode->data = newData;

newNode->next = NULL;

temp->next = newNode;

temp = newNode;

}

displayLinkedList(head);

reverseofaLinkedList(&head);

printf("LinkedList after reversing is:\n");

displayLinkedList(head);

}

void displayLinkedList(struct node\* head)

{

struct node\* temp = head;

printf("The LinkedList is as follows\n");

while(temp!=NULL){

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

temp = temp->next;

}

printf("NULL\n");

}

void reverseofaLinkedList(struct node\*\* head)

{

struct node\* current = \*head;

struct node\* prev = NULL;

struct node\* next = NULL;

while(current!=NULL){

next = current->next;

current->next = prev;

prev = current;

current =next;

}

\*head = prev;

}

**2(b) Delete nth node**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* next;

}\*head = NULL;

void displayLinkedList(struct node\*);

void deleteNthNode(struct node\*\*);

int main(){

int n;

printf("Deleting nth node from Single Linked List\n");

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

printf("Please enter the number of nodes you want to create in the single linkedlist.\n");

scanf("%d",&n);

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

int d;

printf("Enter the data for first Node:\n");

scanf("%d",&d);

head->data = d;

head->next = NULL;

struct node\* temp = head;

int i;

for(i = 2; i<=n; i++){

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

int newData;

printf("Enter the data part for the %d th node\n",i);

scanf("%d",&newData);

newNode->data = newData;

newNode->next = NULL;

temp->next = newNode;

temp = newNode;

}

displayLinkedList(head);

deleteNthNode(&head);

printf("LinkedList After deleting:\n");

displayLinkedList(head);

}

void displayLinkedList(struct node\* head){

struct node\* temp = head;

printf("The LinkedList is as follows\n");

while(temp!=NULL){

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

temp = temp->next;

}

printf("NULL\n");

}

void deleteNthNode(struct node\*\* head){

int n;

printf("Please enter the position where you want to delete:\n");

scanf("%d",&n);

struct node\* temp = \*head;

if(n == 1){

\*head = temp->next;

temp->next = NULL;

free(temp);

printf("LinkedList After deleting:\n");

displayLinkedList(head);

return;

}

else{

int i;

for(i=2; i<=n-1;i++){

if(temp==NULL){

printf("N exceeds the length of the linkedList,Cannot be deleted.");

return;

}

temp =temp->next;

}

if(temp->next == NULL){

printf("There are only %d elements in the list and hence cannot delete %dth element",n-1,n);

return;

}

struct node\* delNode = temp->next;

temp->next = temp->next->next;

delNode->next = NULL;

free(delNode);

}

}

**3(A) To remove duplicate elements from a sorted linked List**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* next;

}\*head = NULL;

void displayLinkedList(struct node\*);

void removeDuplicates(struct node\*\*);

int main(){

int n;

printf("Single Linked List\n");

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

printf("Please enter the number of nodes you want to create in the single linkedlist.\n");

scanf("%d",&n);

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

int d;

printf("Enter the data for first Node:\n");

scanf("%d",&d);

head->data = d;

head->next = NULL;

struct node\* temp = head;

int i;

for(i = 2; i<=n; i++){

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

int newData;

printf("Enter the data part for the %d th node\n",i);

scanf("%d",&newData);

newNode->data = newData;

newNode->next = NULL;

temp->next = newNode;

temp = newNode;

}

displayLinkedList(head);

removeDuplicates(&head);

printf("LinkedList After removing duplicates:\n");

displayLinkedList(head);

}

void displayLinkedList(struct node\* head){

struct node\* temp = head;

printf("The LinkedList is as follows\n");

while(temp!=NULL){

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

temp = temp->next;

}

printf("NULL\n");

}

void removeDuplicates(struct node\*\* head){

struct node\* current = \*head;

struct node\* next = NULL;

while(current->next != NULL){

if(current->data == current->next->data)

{

next = current->next->next;

free(current->next);

current->next = next;

}

else{

current = current->next;

}

}

}

**3(b) To delete k nodes after n nodes of a linked list**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* next;

}\*head = NULL;

void displayLinkedList(struct node\*);

void deleteKNodesAfterNNodes(struct node\*\*);

int main(){

int n;

printf("Single Linked List\n");

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

printf("Please enter the number of nodes you want to create in the single linkedlist.\n");

scanf("%d",&n);

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

int d;

printf("Enter the data for first Node:\n");

scanf("%d",&d);

head->data = d;

head->next = NULL;

struct node\* temp = head;

int i;

for(i = 2; i<=n; i++){

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

int newData;

printf("Enter the data part for the %d th node\n",i);

scanf("%d",&newData);

newNode->data = newData;

newNode->next = NULL;

temp->next = newNode;

temp = newNode;

}

displayLinkedList(head);

deleteKNodesAfterNNodes(&head);

printf("LinkedList After removing duplicates:\n");

displayLinkedList(head);

}

void displayLinkedList(struct node\* head){

struct node\* temp = head;

printf("The LinkedList is as follows\n");

while(temp!=NULL){

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

temp = temp->next;

}

printf("NULL\n");

}

void deleteKNodesAfterNNodes(struct node\*\* head){

int n;

int k;

printf("Please enter the value of K:\n");

scanf("%d",&k);

printf("Please enter the value of N:\n");

scanf("%d",&n);

struct node\* current = \*head;

int i;

for(i = 2; i<= n; i++){

if(current == NULL){

return;

}

current = current->next;

}

struct node\* nextNode = current->next;//we will store the node after k nodes in this.

int j;

for(int j =1; j<=k; j++){

if(nextNode == NULL){

current->next = NULL;

return;

}

nextNode = nextNode->next;

}

current->next = nextNode;

}

**3(c) merging two lists alternatively**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* next;

}\*head = NULL, \*head2=NULL;

void displayLinkedList(struct node\*);

void MergeLinkedLists(struct node\*, struct node\*\*);

int main(){

int n1,n2;

printf("Single Linked List\n");

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

printf("Please enter the number of nodes you want to create in the 1st linkedlist.\n");

scanf("%d",&n1);

printf("Please enter the number of nodes you want to create in the 2nd linkedlist.\n");

scanf("%d",&n2);

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

int d;

printf("Enter the data for first Node of first linked list:\n");

scanf("%d",&d);

head->data = d;

head->next = NULL;

struct node\* temp = head;

int i;

for(i = 2; i<=n1; i++){

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

int newData;

printf("Enter the data part for the %d th node of first linkedlist\n",i);

scanf("%d",&newData);

newNode->data = newData;

newNode->next = NULL;

temp->next = newNode;

temp = newNode;

}

head2 = (struct node\*)malloc(sizeof(struct node\*));

int d2;

printf("Enter the value of first node of 2nd linkedlist:\n");

scanf("%d",&d2);

head2->data = d2;

head2->next = NULL;

temp = head2;

for(i = 2; i<=n2; i++){

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

int newData;

printf("Enter the data part for the %d th node of 2nd linkedlist\n",i);

scanf("%d",&newData);

newNode->data = newData;

newNode->next = NULL;

temp->next = newNode;

temp = newNode;

}

printf("First LinkedList:\n");

displayLinkedList(head);

printf("Second LinkedList:\n");

displayLinkedList(head2);

MergeLinkedLists(head,&head2);

printf("After merging:\n");

printf("First LinkedList:\n");

displayLinkedList(head);

printf("Second LinkedList:");

displayLinkedList(head2);

}

void MergeLinkedLists(struct node\* head1, struct node\*\* head2)

{

struct node\* head1\_curr = head1;

struct node\* head2\_curr = \*head2;

struct node\* head1\_next;

struct node\* head2\_next;

while(head1\_curr!=NULL && head2\_curr!=NULL){

head1\_next = head1\_curr->next;

head2\_next = head2\_curr->next;

head2\_curr->next = head1\_next;

head1\_curr->next = head2\_curr;

head1\_curr = head1\_next;

head2\_curr = head2\_next;

}

\*head2 = head2\_curr;

}

void displayLinkedList(struct node\* head){

struct node\* temp = head;

printf("The LinkedList is as follows\n");

while(temp!=NULL){

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

temp = temp->next;

}

printf("NULL\n");

}

**4. Write a program to implement a Stack (using Structures and Pointers).**

#include<stdio.h>

#include<stdlib.h>

struct stack{

int maxsize;

int top;

int\* elements;

};

struct stack\* initializeStack(int);

int size(struct stack\*);

int isEmpty(struct stack\*);

int isFull(struct stack\*);

void push(struct stack\*,int);

int peek(struct stack\*);

int pop(struct stack\*);

void display(struct stack\*);

int main(){

struct stack\* stack;

int n;

printf("Please enter the maxsize of the stack:\n");

scanf("%d",&n);

stack = initializeStack(n);

while(1){

int c;

printf("Stack Menu\n Enter 1-> push\n Enter 2->pop\n Enter 3->peek\n Enter 4->Display stack\n Enter 5 -> Exit\n");

scanf("%d",&c);

switch(c){

case 1: int d;

printf("Please enter the value of element to be pushed.\n");

scanf("%d",&d);

push(stack,d);

break;

case 2: int ele = pop(stack);

printf("Popped %d\n",ele);

break;

case 3: int ele2 = peek(stack);

printf("Peek Elemenet %d\n",ele2);

break;

case 4: display(stack);

break;

case 5: exit(1);

break;

default:exit(1);

break;

}

}

}

struct stack\* initializeStack(int size){

struct stack\* s = (struct stack\*)malloc(sizeof(struct stack));

s->maxsize = size;

s->top = -1;

s->elements = (int\*)malloc(sizeof(int)\*size);

return s;

}

int size(struct stack\* s){

return s->top + 1;

}

int isEmpty(struct stack\* s){

return s->top == -1;

}

int isFull(struct stack\* s){

return s->top == s->maxsize - 1;

}

void push(struct stack\* s, int data){

if(isFull(s)){

printf("Stack is Full, please delete some to add it.");

return;

}

s->top++;

s->elements[s->top] = data;

}

int peek(struct stack\* s){

if(isEmpty(s)){

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

exit(1);

}

return s->elements[s->top];

}

int pop(struct stack\* s){

if(isEmpty(s)){

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

exit(1);

}

return s->elements[s->top--];

}

void display(struct stack\* s){

int i = s->top;

printf("Stack Elements:\n");

while(i != -1){

printf("%d\n",s->elements[i--]);

}

}

**5. Queue**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* next;

}\*rear = NULL, \*front = NULL;

struct node\* createNode(int);

void dequeue();

void enqueue(int);

int peek();

void display();

int main(){

int c;

while(1){

printf("Please select \n 1. Enqueue\n 2. Dequeue\n 3. Peek\n 4. Display\n 5. Exit\n");

scanf("%d",&c);

switch(c){

case 1: int val;

printf("Please enter the data to be enqueued:\n");

scanf("%d",&val);

enqueue(val);

break;

case 2: dequeue();

break;

case 3: int val2 = peek();

printf("Peek : %d",val2);

break;

case 4: display();

break;

case 5: exit(1);

break;

}

}

}

struct node\* createNode(int val)

{

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

newNode->data = val;

newNode->next = NULL;

return newNode;

}

void enqueue(int val){

struct node\* newNode = createNode(val);

//If queue is empty, then front and rear should point to the newnode.

if(front == NULL){

front = newNode;

rear = newNode;

}

else{

//Update rear next as newNode and update the rear to newly created node.

rear->next = newNode;

rear = newNode;

}

}

void dequeue(){

if(front == NULL){

printf("Queue is Empty");

exit(1);

}

struct node\* temp = front;

//When only one node is present, front will be equal to rear. should make both of them Null

//and free temp

if(front == rear){

front = NULL;

rear = NULL;

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

free(temp);

}

else{

//Since dequeue removes element from front, moving front one position.

front= front->next;

int val = temp->data;

free(temp);

printf("Dequeued: %d",val);

}

}

int peek(){

if(front== NULL){

printf("Queue is empty");

exit(1);

}

return front->data;

}

void display(){

struct node\* temp = front;

if(front == NULL){

printf("Queue is empty\n");

return;

}

while(temp != NULL){

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

temp = temp->next;

}

printf("NULL\n");

}

6. Binary Tree

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node\* left;

struct node\* right;

}\* root = NULL;

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

void inorderDisplay(struct node\*);

void preorderDisplay(struct node\*);

void postorderDisplay(struct node\*);

struct node\* searchNode(struct node\*,int);

int main(){

int c;

while(1){

printf("Binary Search Tree:\n 1.Insert\n 2.Display Inorder\n 3.Display PreOrder\n 4. Display PostOrder\n 5. Search\n 6.Exit\n");

scanf("%d",&c);

switch(c){

case 1: int val;

printf("Please enter the data to be inserted:\n");

scanf("%d",&val);

root = insert(root,val);

break;

case 2: printf("Inorder traversal:\n");

inorderDisplay(root);

printf("\n");

break;

case 3: printf("PreOrder traversal:\n");

preorderDisplay(root);

printf("\n");

break;

case 4: printf("PostOrder traversal:\n");

postorderDisplay(root);

printf("\n");

break;

case 5: int val2;

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

scanf("%d",&val2);

struct node\* found = searchNode(root,val2);

if(found == NULL){

printf("No node found\n");

}

else{

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

}

break;

default: exit(1);

break;

}

}

}

struct node\* insert(struct node\* root, int data){

if(root == NULL){

//While traversing left or right, at one point the root becomes null, there we need to insert this.

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

else{

//Traverse left or right to the exact position where to insert and then inserts the node.

if(data < root->data){

root->left = insert(root->left,data); //Traversing to the left

}

else if( data > root->data){

root->right = insert(root->right,data); //Traversing to the right.

}

return root;

}

}

void inorderDisplay(struct node\* root){

if(root!=NULL){

inorderDisplay(root->left);

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

inorderDisplay(root->right);

}

}

void preorderDisplay(struct node\* root){

if(root!=NULL){

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

preorderDisplay(root->left);

preorderDisplay(root->right);

}

}

void postorderDisplay(struct node\* root)

{

if(root!= NULL){

postorderDisplay(root->left);

postorderDisplay(root->right);

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

}

}

struct node\* searchNode(struct node\* root, int data){

if(root == NULL){ //Not found case.

return NULL;

}

if(root->data == data){

return root;

}

if(root->data > data){

return searchNode(root->left,data);

}

else{

return searchNode(root->right,data);

}

}

7.