# Module 3

**1.Insert at head**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*next;

};

void push(struct Node\*\* head\_ref, int new\_data) {

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

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void insertAfter(struct Node\* prev\_node, int new\_data) {

if (prev\_node == NULL) {

printf("the given previous node cannot be NULL");

return;

}

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

new\_node->data = new\_data;

new\_node->next = prev\_node->next;

prev\_node->next = new\_node;

}

void append(struct Node\*\* head\_ref, int new\_data) {

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

struct Node \*last = \*head\_ref;

new\_node->data = new\_data;

new\_node->next = NULL;

if (\*head\_ref == NULL) {

\*head\_ref = new\_node;

return;

}

while (last->next != NULL)

last = last->next;

last->next = new\_node;

}

void printList(struct Node \*node) {

while (node != NULL) {

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

node = node->next;

}

}

int main() {

struct Node\* head = NULL;

append(&head, 6);

push(&head, 7);

push(&head, 1);

append(&head, 4);

insertAfter(head->next, 8);

printf("Created Linked list is: ");

printList(head);

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2.Insert at tail**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*next;

};

void insertAtTail(struct Node \*\*head, int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed!\n");

return;

}

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node \*current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

void printList(struct Node \*head) {

struct Node \*current = head;

while (current != NULL) {

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

current = current->next;

}

printf("NULL\n");

}

int main() {

struct Node \*head = NULL;

insertAtTail(&head, 10);

insertAtTail(&head, 20);

insertAtTail(&head, 30);

printf("Linked List: ");

printList(head);

struct Node \*current = head;

while (current != NULL) {

struct Node \*temp = current;

current = current->next;

free(temp);

}

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3.Insert specific**

#include <stdio.h>

#include <stdlib.h>

struct slinklist {

int data;

struct slinklist \*next;

};

typedef struct slinklist node;

node \*start = NULL;

int menu() {

int ch;

printf("\n 1.Create a list ");

printf("\n 2.Insert a node at specified position");

printf("\n 3.Display");

printf("\n 4. Exit ");

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

scanf("%d", &ch);

return ch;

}

node\* getnode() {

node \*newnode;

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

printf("\n Enter data: ");

scanf("%d", &newnode->data);

newnode->next = NULL;

return newnode;

}

void createlist(int n) {

int i;

node \*newnode;

node \*temp;

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

newnode = getnode();

if (start == NULL) {

start = newnode;

} else {

temp = start;

while (temp->next != NULL)

temp = temp->next;

temp->next = newnode;

}

}

}

int countnode(node \*ptr) {

int count = 0;

while (ptr != NULL) {

count++;

ptr = ptr->next;

}

return count;

}

void display() {

node \*temp;

temp = start;

printf("\n The contents of List (Left to Right): \n");

if (start == NULL) {

printf("\n Empty List");

return;

} else {

while (temp != NULL) {

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

temp = temp->next;

}

}

printf(" X ");

}

void insert\_at\_pos() {

node \*newnode, \*temp, \*prev;

int pos, nodectr, ctr = 1;

newnode = getnode();

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

scanf("%d", &pos);

nodectr = countnode(start);

if (pos < 1 || pos > nodectr + 1) {

printf("Position %d is invalid\n", pos);

free(newnode);

return;

}

if (pos == 1) {

newnode->next = start;

start = newnode;

} else {

temp = start;

while (ctr < pos - 1) {

temp = temp->next;

ctr++;

}

newnode->next = temp->next;

temp->next = newnode;

}

}

void main(void) {

int ch, n;

while (1) {

ch = menu();

switch (ch) {

case 1:

if (start == NULL) {

printf("\n Number of nodes you want to create: ");

scanf("%d", &n);

createlist(n);

printf("\n List created..");

} else

printf("\n List is already created..");

break;

case 2:

insert\_at\_pos();

break;

case 3:

display();

break;

default:

exit(0);

}

}

}

# Module 4

**4.POISONOUS PLANT**

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int pesticide;

int days;

} Plant;

int poisonousPlants(int n, int\* p) {

Plant\* stack = (Plant\*)malloc(n \* sizeof(Plant));

int top = -1, max\_days = 0;

int i;

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

int days = 0;

while (top >= 0 && stack[top].pesticide >= p[i]) {

days = days > stack[top].days ? days : stack[top].days;

top--;

}

if (top >= 0) {

days++;

} else {

days = 0;

}

max\_days = days > max\_days ? days : max\_days;

stack[++top] = (Plant){p[i], days};

}

free(stack);

return max\_days;

}

int main() {

int n,i;

scanf("%d", &n);

int\* p = (int\*)malloc(n \* sizeof(int));

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

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

}

int result = poisonousPlants(n, p);

printf("%d\n", result);

free(p);

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**5.Truck tour**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_PUMPS 100

int main() {

int n,i;

printf("Enter the number of petrol pumps: ");

scanf("%d", &n);

int petrol[MAX\_PUMPS];

int distance[MAX\_PUMPS];

printf("Enter the petrol and distance for each pump:\n");

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

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

}

int start = 0;

int deficit = 0;

int capacity = 0;

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

capacity += petrol[i] - distance[i];

if (capacity < 0) {

start = i + 1;

deficit += capacity;

capacity = 0;

}

}

if (capacity + deficit >= 0) {

printf("The tour can be completed starting from pump %d.\n", start + 1);

} else {

printf("-1\n");

}

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**6. Queue using two stacks**

#include <stdio.h>

#include <stdlib.h>

typedef struct Stack {

int top;

unsigned capacity;

int\* array;

} Stack;

Stack\* createStack(unsigned capacity) {

Stack\* stack = (Stack\*)malloc(sizeof(Stack));

stack->capacity = capacity;

stack->top = -1;

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

return stack;

}

int isFull(Stack\* stack) {

return stack->top == stack->capacity - 1;

}

int isEmpty(Stack\* stack) {

return stack->top == -1;

}

void push(Stack\* stack, int item) {

if (isFull(stack))

return;

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

}

int pop(Stack\* stack) {

if (isEmpty(stack))

return -1;

return stack->array[stack->top--];

}

typedef struct Queue {

Stack\* stack1;

Stack\* stack2;

} Queue;

Queue\* createQueue(unsigned capacity) {

Queue\* queue = (Queue\*)malloc(sizeof(Queue));

queue->stack1 = createStack(capacity);

queue->stack2 = createStack(capacity);

return queue;

}

void enqueue(Queue\* queue, int item) {

push(queue->stack1, item);

}

int dequeue(Queue\* queue) {

if (isEmpty(queue->stack2)) {

while (!isEmpty(queue->stack1)) {

push(queue->stack2, pop(queue->stack1));

}

}

return pop(queue->stack2);

}

void displayQueue(Queue\* queue) {

if (isEmpty(queue->stack1) && isEmpty(queue->stack2)) {

printf("Queue is empty\n");

return;

}

printf("Queue: ");

int i;

for (i = 0; i <= queue->stack2->top; i++) {

printf("%d ", queue->stack2->array[i]);

}

for (i = queue->stack1->top; i >= 0; i--) {

printf("%d ", queue->stack1->array[i]);

}

printf("\n");

}

int main() {

Queue\* queue = createQueue(100);

enqueue(queue, 10);

enqueue(queue, 20);

enqueue(queue, 30);

printf("Dequeued item is %d\n", dequeue(queue));

displayQueue(queue);

return 0;

}

# Module 5

**1.Lowest common ancestor**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*left, \*right;

};

struct node \*lca (struct node \*root, int n1, int n2)

{

while (root != NULL)

{

if (root->data > n1 && root->data > n2)

root = root->left;

else if (root->data < n1 && root->data < n2)

root = root->right;

else

break;

}

return root;

}

struct node \*newNode (int data)

{

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

node->data = data;

node->left = node->right = NULL;

return (node);

}

int main ()

{

struct node \*root = newNode (20);

root->left = newNode (8);

root->right = newNode (22);

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

root->left->right = newNode (12);

root->left->right->left = newNode (10);

root->left->right->right = newNode (14);

int n1 = 10, n2 = 14;

struct node \*t = lca (root, n1, n2);

printf ("LCA of %d and %d is %d \n", n1, n2, t->data);

n1 = 14, n2 = 8;

t = lca (root, n1, n2);

printf ("LCA of %d and %d is %d \n", n1, n2, t->data);

n1 = 10, n2 = 22;

t = lca (root, n1, n2);

printf ("LCA of %d and %d is %d \n", n1, n2, t->data);

getchar ();

return 0;

}