**[1A]**

#include <stdio.h>

void reverseArray(int arr[], int size) {

int start = 0;

int end = size - 1;

int temp;

while (start < end) {

temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

}}

int main() {

int arr[100], n, i;

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

printf("Enter %d elements:\n", n);

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

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

}

reverseArray(arr, n);

printf("Reversed array:\n");

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

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

}

return 0;}

**[2A]**

#include<stdio.h>

int linearSearch(int arr[], int n, int key) {

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

if (arr[i] == key)

return i;

}

return -1;

}

int main() {

int arr[] = {10, 25, 30, 45, 50, 60};

int n = sizeof(arr) / sizeof(arr[0]);

int key, result;

printf("Array elements: ");

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

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

}

printf("\nEnter the element to search: ");

scanf("%d", &key);

result = linearSearch(arr, n, key);

if (result != -1)

printf("Element %d found at index %d.\n", key, result);

else

printf("Element %d not found in the array.\n", key);

return 0;

}

[1B]

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* left;

struct Node\* right; };

struct Node\* createNode(int value) {

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

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode; }

struct Node\* insert(struct Node\* root, int value) {

if (root == NULL) {

return createNode(value); }

if (value < root->data)

root->left = insert(root->left, value);

else if (value > root->data)

root->right = insert(root->right, value);

return root; }

void inorderTraversal(struct Node\* root) {

if (root != NULL) {

inorderTraversal(root->left);

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

inorderTraversal(root->right); } }

struct Node\* search(struct Node\* root, int key) {

if (root == NULL || root->data == key)

return root;

if (key < root->data)

return search(root->left, key);

return search(root->right, key); }

int main() {

struct Node\* root = NULL;

int choice, value, key;

[1B….]

while (1) {

printf("\n--- Binary Search Tree Menu ---\n");

printf("1. Insert\n");

printf("2. In-order Traversal\n");

printf("3. Search\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

root = insert(root, value);

break;

case 2:

printf("In-order Traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 3:

printf("Enter value to search: ");

scanf("%d", &key);

struct Node\* result = search(root, key);

if (result != NULL)

printf("Value %d found in the BST.\n", key);

else

printf("Value %d not found in the BST.\n", key);

break;

case 4:

exit(0);

default:

printf("Invalid choice! Try again.\n");

} } return 0; }

**[2B]**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* top = NULL;

void push(int value) {

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

if (newNode == NULL) {

printf("Stack Overflow\n");

return;

}

newNode->data = value;

newNode->next = top;

top = newNode;

printf("%d pushed to stack\n", value);

}

void pop() {

if (top == NULL) {

printf("Stack Underflow\n");

return;

}

struct Node\* temp = top;

printf("Popped element: %d\n", top->data);

top = top->next;

free(temp);

}

void display() {

if (top == NULL) {

printf("Stack is empty\n");

return;

} **[2B….]**

struct Node\* temp = top;

printf("Stack elements:\n");

while (temp != NULL) {

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

temp = temp->next;

} }

int main() {

int choice, value;

while (1) {

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

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(value);

break;

case 2: case 3:

pop(); display();

break; break;

case 4:

exit(0);

default:

printf("Invalid choice! Try again.\n");

} }

return 0;

}

**[3A]**

#include <stdio.h>

int linearSearchRecursive(int arr[], int index, int size, int key) {

if (index >= size)

return -1;

if (arr[index] == key)

return index;

return linearSearchRecursive(arr, index + 1, size, key);

}

int main() {

int arr[] = {5, 10, 15, 20, 25, 30};

int size = sizeof(arr) / sizeof(arr[0]);

int key;

printf("Array elements: ");

for (int i = 0; i < size; i++) {

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

}

printf("\nEnter the element to search: ");

scanf("%d", &key);

int result = linearSearchRecursive(arr, 0, size, key);

if (result != -1)

printf("Element %d found at index %d.\n", key, result);

else

printf("Element %d not found in the array.\n", key);

return 0;

}

**[3B]**

#include <stdio.h>

#define SIZE 100

int queue[SIZE];

int front = -1, rear = -1;

void enqueue(int value) {

if (rear == SIZE - 1) {

printf("Queue Overflow\n");

return;

}

if (front == -1) front = 0;

rear++;

queue[rear] = value;

printf("%d enqueued to queue\n", value);

}

void dequeue() {

if (front == -1 || front > rear) {

printf("Queue Underflow\n");

return;

}

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

front++;

}

void display() {

if (front == -1 || front > rear) {

printf("Queue is empty\n");

return;

}

printf("Queue elements: ");

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

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

}

printf("\n");

}

**[3B….]**

int main() {

int choice, value;

while (1) {

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

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

return 0;

default:

printf("Invalid choice! Try again.\n");

}

}

return 0;

}

**[4A]**

#include <stdio.h>

int binarySearchRecursive(int arr[], int low, int high, int key) {

if (low > high)

return -1;

int mid = (low + high) / 2;

if (arr[mid] == key)

return mid;

else if (key < arr[mid])

return binarySearchRecursive(arr, low, mid - 1, key);

else

return binarySearchRecursive(arr, mid + 1, high, key);

}

int main() {

int arr[] = {5, 10, 15, 20, 25, 30, 35};

int size = sizeof(arr) / sizeof(arr[0]);

int key;

printf("Array elements: ");

for (int i = 0; i < size; i++) {

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

}

printf("\nEnter the element to search: ");

scanf("%d", &key);

int result = binarySearchRecursive(arr, 0, size - 1, key);

if (result != -1)

printf("Element %d found at index %d.\n", key, result);

else

printf("Element %d not found in the array.\n", key);

return 0;

}

#include <stdio.h> **[4B]**

#define SIZE 100

int queue[SIZE];

int front = -1, rear = -1;

void enqueue(int value) {

if (rear == SIZE - 1) {

printf("Queue Overflow\n");

return; }

if (front == -1) front = 0;

rear++;

queue[rear] = value;

printf("%d enqueued to queue\n", value); }

void dequeue() {

if (front == -1 || front > rear) {

printf("Queue Underflow\n");

return; }

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

front++; } case 1:

void display() { printf("Enter value to enqueue: ");

if (front == -1 || front > rear) { scanf("%d", &value);

printf("Queue is empty\n"); enqueue(value);

return; } break;

printf("Queue elements: "); case 2:

for (int i = front; i <= rear; i++) { dequeue();

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

printf("\n"); } case 3:

int main() { display();

int choice, value; break;

while (1) { case 4:

printf("\n--- Queue Menu ---\n"); return 0;

printf("1. Enqueue\n"); default:

printf("2. Dequeue\n"); printf("Invalid choice! Try again.\n"); } }

printf("3. Display\n"); return 0;

printf("4. Exit\n"); }

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

**[5A]**

#include <stdio.h>

int binarySearchIterative(int arr[], int size, int key) {

int low = 0, high = size - 1, mid;

while (low <= high) {

mid = (low + high) / 2;

if (arr[mid] == key)

return mid;

else if (key < arr[mid])

high = mid - 1;

else

low = mid + 1;

}

return -1;

}

int main() {

int arr[] = {2, 5, 8, 12, 16, 23, 38, 45, 56, 72};

int size = sizeof(arr) / sizeof(arr[0]);

int key;

printf("Array elements: ");

for (int i = 0; i < size; i++) {

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

}

printf("\nEnter the element to search: ");

scanf("%d", &key);

int result = binarySearchIterative(arr, size, key);

if (result != -1)

printf("Element %d found at index %d.\n", key, result);

else

printf("Element %d not found in the array.\n", key);

return 0;

}

**[5B]**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* front = NULL;

Node\* rear = NULL;

void enqueue(int value) {

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

if (!newNode) {

printf("Memory allocation failed\n");

return;

}

newNode->data = value;

newNode->next = NULL;

if (front == NULL) {

front = rear = newNode;

rear->next = front;

} else {

rear->next = newNode;

rear = newNode;

rear->next = front; }

printf("%d enqueued to queue\n", value);}

void dequeue() {

if (front == NULL) {

printf("Queue is empty\n");

return; }

if (front == rear) {

printf("Dequeued: %d\n", front->data);

free(front);

front = rear = NULL;

} else {

Node\* temp = front;

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

front = front->next;

rear->next = front;

free(temp);

**[5B…..]**

} }

void display() {

if (front == NULL) {

printf("Queue is empty\n");

return;

}

Node\* temp = front;

printf("Queue elements: ");

do {

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

temp = temp->next;

} while (temp != front);

printf("\n"); }

int main() {

int choice, value;

while (1) {

printf("\n--- Circular Queue using Linked List ---\n");

printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to enqueue: ");

scanf("%d", &value);

enqueue(value);

break;

case 2: case 3:

dequeue(); display();

break; break;

case 4:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice\n");

}

}

}

[6A]

#include <stdio.h>

#include <stdlib.h>

struct Node {

int coeff;

int exp;

struct Node\* next; };

struct Node\* createNode(int coeff, int exp) {

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

newNode->coeff = coeff;

newNode->exp = exp;

newNode->next = NULL;

return newNode; }

struct Node\* insertTerm(struct Node\* head, int coeff, int exp) {

struct Node\* newNode = createNode(coeff, exp);

if (head == NULL) {

return newNode; }

struct Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next; }

temp->next = newNode;

return head; }

void displayPolynomial(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

printf("%dx^%d", temp->coeff, temp->exp);

if (temp->next != NULL) {

printf(" + "); }

temp = temp->next; }

printf("\n"); } while (head != NULL) {

int main() { temp = head;

struct Node\* head = NULL; head = head->next;

head = insertTerm(head, 3, 2); free(temp);

head = insertTerm(head, 5, 1); }

head = insertTerm(head, 2, 0); return 0;

printf("Polynomial Expression: "); }

displayPolynomial(head);

struct Node\* temp;

**[6B]**

#include <stdio.h>

void merge(int arr[], int left, int right) {

if (left >= right) {

return; }

int mid = left + (right - left) / 2;

merge(arr, left, mid);

merge(arr, mid + 1, right);

int n1 = mid - left + 1;

int n2 = right - mid;

int leftArr[n1], rightArr[n2];

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

leftArr[i] = arr[left + i]; }

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

rightArr[j] = arr[mid + 1 + j]; }

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (leftArr[i] <= rightArr[j]) {

arr[k] = leftArr[i];

i++;

} else {

arr[k] = rightArr[j]; printf("\n"); }

j++; } int main() {

k++; } int arr[] = {38, 27, 43,

while (i < n1) { 3, 9, 82, 10};

arr[k] = leftArr[i]; int size = sizeof(arr) /

i++; sizeof(arr[0]);

k++; } printf("Original Array: ");

while (j < n2) { printArray(arr, size);

arr[k] = rightArr[j]; merge(arr, 0, size - 1);

j++; printf("Sorted Array: ");

k++; } } printArray(arr, size);

void printArray(int arr[], int size) { return 0;

for (int i = 0; i < size; i++) { }

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

**[7A]**

#include <stdio.h>

void selectionSort(int arr[], int size) {

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

int minIndex = i;

for (int j = i + 1; j < size; j++) {

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

minIndex = j;

}

}

if (minIndex != i) {

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

}

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++) {

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

}

printf("\n");

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

int size = sizeof(arr) / sizeof(arr[0]);

printf("Original Array: ");

printArray(arr, size);

selectionSort(arr, size);

printf("Sorted Array: ");

printArray(arr, size);

return 0;

}

**[7B]**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

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

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

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

printf("%d inserted at the beginning.\n", data);

}

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

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

struct Node\* temp = \*head;

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

\*head = newNode; // If the list is empty, make the new node the head

printf("%d inserted at the end.\n", data);

return;

}

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode; // Insert the new node at the end

printf("%d inserted at the end.\n", data);

}

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

struct Node\* temp = head;

while (temp != NULL && temp->data != prevData) {

temp = temp->next; **[7B…..]**

}

if (temp != NULL) {

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

newNode->data = data;

newNode->next = temp->next;

temp->next = newNode;

printf("%d inserted after %d.\n", data, prevData);

} else {

printf("Node with data %d not found.\n", prevData);

}

}

void displayList(struct Node\* head) {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

printf("Linked List: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

insertAtBeginning(&head, 10);

insertAtEnd(&head, 20);

insertAtEnd(&head, 30);

insertAtBeginning(&head, 5);

insertAfter(head, 10, 15);

displayList(head);

return 0;

}

**[8A]**

#include <stdio.h>

void bubbleSort(int arr[], int size) {

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

int swapped = 0;

for (int j = 0; j < size - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

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

arr[j + 1] = temp;

swapped = 1;

}

}

if (swapped == 0) {

break;

}

}

}

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++) {

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

}

printf("\n");

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

int size = sizeof(arr) / sizeof(arr[0]);

printf("Original Array: ");

printArray(arr, size);

bubbleSort(arr, size);

printf("Sorted Array: ");

printArray(arr, size);

return 0;

}

**[8B]**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

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

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

newNode->data = data;

newNode->prev = NULL;

newNode->next = \*head;

if (\*head != NULL) {

(\*head)->prev = newNode; }

\*head = newNode;

printf("%d inserted at the beginning.\n", data); }

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

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

struct Node\* temp = \*head;

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

newNode->prev = NULL;

\*head = newNode;

printf("%d inserted at the end.\n", data);

return; }

while (temp->next != NULL) {

temp = temp->next; }

temp->next = newNode;

newNode->prev = temp;

printf("%d inserted at the end.\n", data); }

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

struct Node\* temp = head;

while (temp != NULL && temp->data != prevData) {

temp = temp->next;

}

if (temp != NULL) { **[8B….]**

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

newNode->data = data;

newNode->next = temp->next;

if (temp->next != NULL) {

temp->next->prev = newNode; }

newNode->prev = temp;

temp->next = newNode;

printf("%d inserted after %d.\n", data, prevData);

} else {

printf("Node with data %d not found.\n", prevData);

} }

void displayList(struct Node\* head) {

if (head == NULL) {

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

return; }

struct Node\* temp = head;

printf("Doubly Linked List: ");

while (temp != NULL) {

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

temp = temp->next; }

printf("NULL\n"); }

int main() {

struct Node\* head = NULL;

insertAtBeginning(&head, 10);

insertAtEnd(&head, 20);

insertAtEnd(&head, 30);

insertAtBeginning(&head, 5);

insertAfter(head, 10, 15);

displayList(head);

return 0;

}

**[9A]**

#include <stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp; }

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

int pivot = arr[high];

int i = low - 1;

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

if (arr[j] <= pivot) {

i++;

swap(&arr[i], &arr[j]);

} }

swap(&arr[i + 1], &arr[high]);

return (i + 1); }

void quickSort(int arr[], int low, int high) {

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

} }

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++)

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

printf("\n"); }

int main() {

int arr[] = {34, 7, 23, 32, 5, 62};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Original Array: ");

printArray(arr, n);

quickSort(arr, 0, n - 1); return 0;

printf("Sorted Array: "); }

printArray(arr, n);

**[9 B]**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL;

void insertAtEnd(int value) {

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

newNode->data = value;

if (head == NULL) {

newNode->next = newNode;

head = newNode;

printf("%d inserted as the first node.\n", value);

return;

}

struct Node\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

printf("%d inserted at the end.\n", value);

}

void insertAtBeginning(int value) {

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

newNode->data = value;

if (head == NULL) {

newNode->next = newNode;

head = newNode;

printf("%d inserted as the first node.\n", value);

return;

} **[9 B….]**

struct Node\* temp = head;

while (temp->next != head) {

temp = temp->next;

}

newNode->next = head;

temp->next = newNode;

head = newNode;

printf("%d inserted at the beginning.\n", value);

}

void displayList() {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

printf("Circular Linked List: ");

do {

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

temp = temp->next;

} while (temp != head);

printf("(back to head)\n");

}

int main() {

insertAtEnd(10);

insertAtEnd(20);

insertAtEnd(30);

insertAtBeginning(5);

insertAtBeginning(1);

displayList();

return 0;

}

**[10 A]**

#include <stdio.h>

#define MAX 100

int stack[MAX];

int top = -1;

void push(int value) {

if (top == MAX - 1) {

printf("Stack Overflow! Cannot push %d\n", value);

} else {

top++;

stack[top] = value;

printf("%d pushed onto stack.\n", value);

} }

void pop() {

if (top == -1) {

printf("Stack Underflow! Cannot pop.\n");

} else {

printf("%d popped from stack.\n", stack[top]);

top--;} }

void display() {

if (top == -1) {

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

} else {

printf("Stack elements (top to bottom):\n");

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

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

} } }

int main() {

int choice, value;

while (1) {

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

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

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(value);

break;

case 2: case 3: case 4:

pop(); display(); printf("Exiting...\n");

break; break; return 0;

default:

printf("Invalid choice. Try again.\n");

} } }

**[10 B]**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

struct Stack {

int top;

int items[MAX];

};

void initStack(struct Stack\* s) {

s->top = -1; }

int isEmpty(struct Stack\* s) {

return s->top == -1; }

void push(struct Stack\* s, int value) {

if (s->top == MAX - 1) {

printf("Stack Overflow!\n");

return; }

s->items[++(s->top)] = value; }

int pop(struct Stack\* s) {

if (isEmpty(s)) {

**[10 B…..]**

printf("Stack Underflow!\n");

return -1; }

return s->items[(s->top)--]; }

int evaluatePostfix(char\* exp) {

struct Stack s;

initStack(&s);

for (int i = 0; exp[i] != '\0'; i++) {

if (isdigit(exp[i])) {

push(&s, exp[i] - '0'); }

else {

int op2 = pop(&s);

int op1 = pop(&s);

switch (exp[i]) {

case '+': case '-':

push(&s, op1 + op2); push(&s, op1 - op2);

break; break;

case '\*':

push(&s, op1 \* op2);

break;

case '/':

push(&s, op1 / op2);

break;

default:

printf("Invalid Operator!\n");

return -1;

} } }

return pop(&s); }

int main() {

char exp[] = "53+82-";

printf("Postfix Expression: %s\n", exp);

int result = evaluatePostfix(exp);

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

return 0; }