|  |
| --- |
| **SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES** |
| **COMPUTER SCIENCE AND ENGINEERING PROGRAMME** |

**SUB CODE: CSA0392 SUB NAME: Data Structures for Hashing Techniques**

**LIST OF PROGRAMS**

**DAY 2 : 26.07.2024**

**Lab Questions to be practiced with test cases**

1. Write a C program to implement Stack operations using array such as PUSH, POP and PEEK.

Answer:

#include <stdio.h>

#include <stdlib.h>

// Define the maximum size of the stack

#define MAX 100

// Define the stack structure

typedef struct Stack {

int arr[MAX];

int top;

} Stack;

// Function to initialize the stack

void initialize(Stack\* stack) {

stack->top = -1; // Stack is empty

}

// Function to check if the stack is full

int isFull(Stack\* stack) {

return stack->top == MAX - 1;

}

// Function to check if the stack is empty

int isEmpty(Stack\* stack) {

return stack->top == -1;

}

// Function to add an element to the stack (PUSH)

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

if (isFull(stack)) {

printf("Stack Overflow: Cannot push %d, stack is full.\n", value);

return;

}

stack->arr[++(stack->top)] = value;

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

}

// Function to remove an element from the stack (POP)

int pop(Stack\* stack) {

if (isEmpty(stack)) {

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

return -1; // Return a sentinel value to indicate an error

}

return stack->arr[(stack->top)--];

}

// Function to get the top element of the stack (PEEK)

int peek(Stack\* stack) {

if (isEmpty(stack)) {

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

return -1; // Return a sentinel value to indicate an error

}

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

}

// Function to display the elements of the stack

void display(Stack\* stack) {

if (isEmpty(stack)) {

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

return;

}

printf("Stack elements:\n");

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

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

}

printf("\n");

}

// Main function to demonstrate stack operations

int main() {

Stack stack;

initialize(&stack);

int choice, value;

while (1) {

printf("\nStack Operations Menu:\n");

printf("1. PUSH\n");

printf("2. POP\n");

printf("3. PEEK\n");

printf("4. DISPLAY\n");

printf("5. EXIT\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(&stack, value);

break;

case 2:

value = pop(&stack);

if (value != -1) {

printf("%d popped from stack.\n", value);

}

break;

case 3:

value = peek(&stack);

if (value != -1) {

printf("Top element is %d.\n", value);

}

break;

case 4:

display(&stack);

break;

case 5:

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

return 0;

default:

printf("Invalid choice. Please try again.\n");

}

}

}

1. Write a C program to implement Stack operations using linked list such as PUSH, POP and PEEK.

Answer:

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the linked list

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Define the stack structure

typedef struct Stack {

Node\* top;

} Stack;

// Function to initialize the stack

void initialize(Stack\* stack) {

stack->top = NULL; // Stack is empty

}

// Function to create a new node

Node\* createNode(int data) {

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

if (!newNode) {

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

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to check if the stack is empty

int isEmpty(Stack\* stack) {

return stack->top == NULL;

}

// Function to add an element to the stack (PUSH)

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

Node\* newNode = createNode(value);

newNode->next = stack->top;

stack

1. Write a C program for Sorting elements using a stack (e.g., sorting a stack using recursion).

Answer:

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the stack

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Function to create a new node

Node\* createNode(int data) {

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

if (!newNode) {

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

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to push an element onto the stack

void push(Node\*\* top, int data) {

Node\* newNode = createNode(data);

newNode->next = \*top;

\*top = newNode;

}

// Function to pop an element from the stack

int pop(Node\*\* top) {

if (\*top == NULL) {

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

return -1; // Return a sentinel value to indicate an error

}

Node\* temp = \*top;

int poppedValue = temp->data;

\*top = (\*top)->next;

free(temp);

return poppedValue;

}

// Function to get the top element of the stack

int peek(Node\* top) {

if (top == NULL) {

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

return -1; // Return a sentinel value to indicate an error

}

return top->data;

}

// Function to check if the stack is empty

int isEmpty(Node\* top) {

return top == NULL;

}

// Function to insert an element into a sorted stack

void sortedInsert(Node\*\* top, int element) {

// If stack is empty or the element is greater than the top element

if (isEmpty(\*top) || element > peek(\*top)) {

push(top, element);

} else {

// Remove the top element

int temp = pop(top);

// Recursively insert the element

sortedInsert(top, element);

// Push the removed element back to stack

push(top, temp);

}

}

// Function to sort the stack using recursion

void sortStack(Node\*\* top) {

// Base case: If the stack is empty or has only one element

if (!isEmpty(\*top)) {

// Remove the top element

int temp = pop(top);

// Recursively sort the remaining stack

sortStack(top);

// Insert the removed element back into the sorted stack

sortedInsert(top, temp);

}

}

// Function to display the stack elements

void displayStack(Node\* top) {

if (isEmpty(top)) {

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

return;

}

Node\* temp = top;

printf("Stack elements:\n");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

// Main function to demonstrate stack sorting

int main() {

Node\* stack = NULL;

int choice, value;

while (1) {

printf("\nStack Operations Menu:\n");

printf("1. PUSH\n");

printf("2. POP\n");

printf("3. PEEK\n");

printf("4. SORT\n");

printf("5. DISPLAY\n");

printf("6. EXIT\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to push: ");

scanf("%d", &value);

push(&stack, value);

break;

case 2:

value = pop(&stack);

if (value != -1) {

printf("%d popped from stack.\n", value);

}

break;

case 3:

value = peek(stack);

if (value != -1) {

printf("Top element is %d.\n", value);

}

break;

case 4:

sortStack(&stack);

printf("Stack sorted.\n");

break;

case 5:

displayStack(stack);

break;

case 6:

// Free all allocated memory

while (!isEmpty(stack)) {

pop(&stack);

}

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

return 0;

default:

printf("Invalid choice. Please try again.\n");

}

}

}

1. Write a C Program to Simulate Recursive Function Calls Using a Stack

Answer:

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a stack node

typedef struct StackNode {

int n; // Parameter for the factorial function

int result; // Result of the factorial computation

int state; // State to track where we are in the computation

struct StackNode\* next;

} StackNode;

// Define the stack structure

typedef struct Stack {

StackNode\* top;

} Stack;

// Function to initialize the stack

void initialize(Stack\* stack) {

stack->top = NULL;

}

// Function to create a new stack node

StackNode\* createNode(int n) {

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

if (!newNode) {

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

exit(1);

}

newNode->n = n;

newNode->result = 0; // Initialize result

newNode->state = 0; // Initialize state

newNode->next = NULL;

return newNode;

}

// Function to push a node onto the stack

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

StackNode\* newNode = createNode(n);

newNode->next = stack->top;

stack->top = newNode;

}

// Function to pop a node from the stack

StackNode\* pop(Stack\* stack) {

if (stack->top == NULL) {

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

return NULL;

}

StackNode\* temp = stack->top;

stack->top = stack->top->next;

return temp;

}

// Function to compute factorial using simulated recursion

int factorial(int n) {

Stack stack;

initialize(&stack);

int result = 1;

// Push the initial call to the stack

push(&stack, n);

while (stack.top != NULL) {

StackNode\* node = pop(&stack);

if (node->n == 0 || node->n == 1) {

node->result = 1;

} else {

// Simulate the recursive call by pushing intermediate states

push(&stack, node->n - 1); // Pushing the recursive call

node->state = 1; // Indicating that we need to process this node again after the recursive call

push(&stack, node->n); // Re-pushing the current node to process its result after the recursive call

continue;

}

// Update the result after processing

while (node->state == 1) {

StackNode\* prev = pop(&stack);

node->result \*= prev->result; // Multiply with result from the recursive call

free(prev);

node->state = 0; // Process completed

}

result = node->result;

free(node);

}

return result;

}

// Main function to demonstrate stack simulation for factorial computation

int main() {

int n;

printf("Enter a number to compute its factorial: ");

scanf("%d", &n);

if (n < 0) {

printf("Factorial is not defined for negative numbers.\n");

return 1;

}

printf("Factorial of %d is %d.\n", n, factorial(n));

return 0;

}

1. Write a C program to Implement undo and redo functionality using two stacks.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_OPERATIONS 100

// Define the structure for a stack node

typedef struct StackNode {

char operation[50]; // Store operation description

struct StackNode\* next;

} StackNode;

// Define the stack structure

typedef struct Stack {

StackNode\* top;

int size;

} Stack;

// Function to initialize the stack

void initialize(Stack\* stack) {

stack->top = NULL;

stack->size = 0;

}

// Function to create a new stack node

StackNode\* createNode(const char\* operation) {

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

if (!newNode) {

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

exit(1);

}

strcpy(newNode->operation, operation);

newNode->next = NULL;

return newNode;

}

// Function to push an operation onto the stack

void push(Stack\* stack, const char\* operation) {

StackNode\* newNode = createNode(operation);

newNode->next = stack->top;

stack->top = newNode;

stack->size++;

}

// Function to pop an operation from the stack

char\* pop(Stack\* stack) {

if (stack->top == NULL) {

return NULL; // Stack is empty

}

StackNode\* temp = stack->top;

stack->top = stack->top->next;

stack->size--;

char\* operation = strdup(temp->operation);

free(temp);

return operation;

}

// Function to check if the stack is empty

int isEmpty(Stack\* stack) {

return stack->top == NULL;

}

// Function to display stack elements

void displayStack(Stack\* stack) {

StackNode\* temp = stack->top;

while (temp != NULL) {

printf("%s -> ", temp->operation);

temp = temp->next;

}

printf("NULL\n");

}

// Function to perform undo operation

void undo(Stack\* undoStack, Stack\* redoStack) {

if (

1. Write a C program to Check if a string is a palindrome using a stack.

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100 // Define maximum stack size

// Define the stack structure

typedef struct {

char data[MAX];

int top;

} Stack;

// Function to initialize the stack

void initialize(Stack\* stack) {

stack->top = -1;

}

// Function to check if the stack is empty

int isEmpty(Stack\* stack) {

return stack->top == -1;

}

// Function to check if the stack is full

int isFull(Stack\* stack) {

return stack->top == MAX - 1;

}

// Function to push a character onto the stack

void push(Stack\* stack, char c) {

if (isFull(stack)) {

printf("Stack Overflow: Cannot push onto the stack.\n");

return;

}

stack->data[++(stack->top)] = c;

}

// Function to pop a character from the stack

char pop(Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack Underflow: Cannot pop from the stack.\n");

return '\0'; // Return null character on error

}

return stack->data[(stack->top)--];

}

// Function to check if a string is a palindrome

int isPalindrome(char\* str) {

Stack stack;

initialize(&stack);

int length = strlen(str);

// Push all characters of the string onto the stack

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

push(&stack, str[i]);

}

// Pop characters from the stack and compare with the original string

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

if (tolower(str[i]) != tolower(pop(&stack))) {

return 0; // Not a palindrome

}

}

return 1; // It is a palindrome

}

int main() {

char str[MAX];

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

// Remove trailing newline character from input

str[strcspn(str, "\n")] = '\0';

if (isPalindrome(str)) {

printf("\"%s\" is a palindrome.\n", str);

} else {

printf("\"%s\" is not a palindrome.\n", str);

}

return 0;

}