|  |
| --- |
| **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**

**DATE : 27.08.2024**

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

1.Implement a first in first out (FIFO) queue using only two stacks. The

implemented queue should support all the functions of a normal queue (push,

peek, pop, and empty).

Implement the MyQueue class:

1. void push(int x) Pushes element x to the back of the queue.

2. int pop() Removes the element from the front of the queue and returns it.

3. int peek() Returns the element at the front of the queue.

4. boolean empty() Returns true if the queue is empty, false otherwise.

Input

["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

Output

[null, null, null, 1, 1, false]

Explanation

MyQueue myQueue = new MyQueue();

myQueue.push(1); // queue is: [1]

myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)

myQueue.peek(); // return 1

myQueue.pop(); // return 1, queue is [2]

myQueue.empty(); // return false

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX\_SIZE 1000

// Define a stack structure

typedef struct {

int data[MAX\_SIZE];

int top;

} Stack;

// Initialize a stack

void initStack(Stack \*stack) {

stack->top = -1;

}

// Check if a stack is empty

bool isEmpty(Stack \*stack) {

return stack->top == -1;

}

// Check if a stack is full

bool isFull(Stack \*stack) {

return stack->top == MAX\_SIZE - 1;

}

// Push an element onto the stack

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

if (isFull(stack)) {

printf("Stack overflow\n");

return;

}

stack->data[++stack->top] = value;

}

// Pop an element from the stack

int pop(Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack underflow\n");

return -1; // Indicate error

}

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

}

// Peek the top element of the stack

int peek(Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack is empty\n");

return -1; // Indicate error

}

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

}

// Define the MyQueue structure

typedef struct {

Stack stack1;

Stack stack2;

} MyQueue;

// Initialize MyQueue

void initQueue(MyQueue \*queue) {

initStack(&queue->stack1);

initStack(&queue->stack2);

}

// Push an element to the queue

void push(MyQueue \*queue, int x) {

push(&queue->stack1, x);

}

// Transfer elements from stack1 to stack2

void transfer(Stack \*src, Stack \*dest) {

while (!isEmpty(src)) {

push(dest, pop(src));

}

}

// Pop an element from the queue

int pop(MyQueue \*queue) {

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

transfer(&queue->stack1, &queue->stack2);

}

return pop(&queue->stack2);

}

// Peek the front element of the queue

int peek(MyQueue \*queue) {

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

transfer(&queue->stack1, &queue->stack2);

}

return peek(&queue->stack2);

}

// Check if the queue is empty

bool empty(MyQueue \*queue) {

return isEmpty(&queue->stack1) && isEmpty(&queue->stack2);

}

// Test the implementation

int main() {

MyQueue queue;

initQueue(&queue);

push(&queue, 1);

push(&queue, 2);

printf("Peek: %d\n", peek(&queue)); // Should print 1

printf("Pop: %d\n", pop(&queue)); // Should print 1

printf("Empty: %d\n", empty(&queue)); // Should print 0 (false)

return 0;

}

2. Given an array arr, sort the elements in descending order using bubblesort.

Arr=[9,10,-9,23,67,-90]

Output:[67,23,10,9,-9,-90]

Answer:

#include <stdio.h>

// Function to perform Bubble Sort in descending order

void bubbleSortDescending(int arr[], int n) {

int i, j, temp;

int swapped;

// Traverse through all array elements

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

swapped = 0;

// Last i elements are already in place

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

// Compare adjacent elements

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

// Swap if the element is smaller than the next element

temp = arr[j];

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

arr[j + 1] = temp;

swapped = 1;

}

}

// If no two elements were swapped by inner loop, then break

if (swapped == 0) {

break;

}

}

}

// Function to print the array

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

int i;

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

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

}

printf("\n");

}

int main() {

int arr[] = {9, 10, -9, 23, 67, -90};

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

printf("Original array: \n");

printArray(arr, n);

bubbleSortDescending(arr, n);

printf("Sorted array in descending order: \n");

printArray(arr, n);

return 0;

}

3. You have been given a positive integer N. You need to find and print the

Factorial of this number without using recursion. The Factorial of a positive

integer N refers to the product of all number in the range from 1 to N.

Input: N=2

Output: 2

Input: N=4

Output: 24

Answer:

#include <stdio.h>

// Function to calculate factorial iteratively

unsigned long long factorial(int N) {

unsigned long long result = 1; // Use unsigned long long to handle larger results

for (int i = 1; i <= N; i++) {

result \*= i;

}

return result;

}

int main() {

int N;

// Read input from user

printf("Enter a positive integer: ");

scanf("%d", &N);

// Check if the input is valid

if (N < 0) {

printf("Please enter a positive integer.\n");

return 1;

}

// Calculate and print the factorial

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

return 0;

}

4. Given an array arr, sort the elements in ascending order using Bubble sort.

Arr=[9,10,-9,23,67,-90]

Output:[-90,-9,9,10,23,67]

Answer:

#include <stdio.h>

#include <stdbool.h>

// Function to perform Bubble Sort in ascending order

void bubbleSortAscending(int arr[], int n) {

int i, j, temp;

bool swapped;

// Traverse through all array elements

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

swapped = false;

// Last i elements are already in place

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

// Compare adjacent elements

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

// Swap if the element is greater than the next element

temp = arr[j];

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

arr[j + 1] = temp;

swapped = true;

}

}

// If no two elements were swapped by inner loop, then break

if (!swapped) {

break;

}

}

}

// Function to print the array

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

int i;

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

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

}

printf("\n");

}

int main() {

int arr[] = {9, 10, -9, 23, 67, -90};

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

printf("Original array: \n");

printArray(arr, n);

bubbleSortAscending(arr, n);

printf("Sorted array in ascending order: \n");

printArray(arr, n);

return 0;

}

5. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

1. MinStack() initializes the stack object.

2. void push(int val) pushes the element val onto the stack.

3. void pop() removes the element on the top of the stack.

4. int top() gets the top element of the stack.

5. int getMin() retrieves the minimum element in the stack.

Input

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

Output

[null,null,null,null,-3,null,0,-2]

Explanation

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top(); // return 0

minStack.getMin(); // return -2

Answer:

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX\_SIZE 1000

// Define a stack structure

typedef struct {

int data[MAX\_SIZE];

int top;

} Stack;

// Initialize a stack

void initStack(Stack \*stack) {

stack->top = -1;

}

// Check if a stack is empty

int isEmpty(Stack \*stack) {

return stack->top == -1;

}

// Check if a stack is full

int isFull(Stack \*stack) {

return stack->top == MAX\_SIZE - 1;

}

// Push an element onto the stack

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

if (isFull(stack)) {

printf("Stack overflow\n");

return;

}

stack->data[++stack->top] = value;

}

// Pop an element from the stack

int pop(Stack \*stac

6.Find the factorial of a number using iterative procedure

Input : 3

Output: 6

Answer:

#include <stdio.h>

// Function to compute factorial iteratively

unsigned long long factorial(int n) {

unsigned long long result = 1; // Use unsigned long long to handle large results

// Multiply result by each number from 1 to n

for (int i = 1; i <= n; i++) {

result \*= i;

}

return result;

}

int main() {

int number;

// Read the input number

printf("Enter a positive integer: ");

scanf("%d", &number);

// Check for valid input

if (number < 0) {

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

return 1;

}

// Compute and print the factorial

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

return 0;

}

7.Given the head of a linked list, insert the node in nth place and return its head.

Input: head = [1,3,2,3,4,5], p=3 n = 2

Output: [1,3,2,3,4,5]

Input: head = [1], p = 0, n = 1

Output: [0,1]

Input: head = [1,2], p=3, n = 3

Output: [1,2,3]

Answer:

#include <stdio.h>

#include <stdlib.h>

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

typedef struct Node {

int value;

struct Node \*next;

} Node;

// Function to create a new node

Node\* createNode(int value) {

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

newNode->value = value;

newNode->next = NULL;

return newNode;

}

// Function to insert a node at a specific position

Node\* insertNode(Node\* head, int p, int n) {

Node \*newNode = createNode(n);

// Special case: Inserting at the head

8. Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return the reversed list.

Input: head = [1, 2, 3, 4, 5], left = 2, right = 4

Output: [1, 4, 3, 2, 5]

Input: head = [5], left = 1, right = 1

Output: [5]

Input : [10,20,30,40,50,60,70], left = 3, right = 6

Output : [10,20,60,50,40,30,70]

Answer:

#include <stdio.h>

#include <stdlib.h>

// Definition for singly-linked list node

typedef struct Node {

int value;

struct Node \*next;

} Node;

// Function to create a new node

Node\* createNode(int value) {

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

newNode->value = value;

newNode->next = NULL;

return newNode;

}

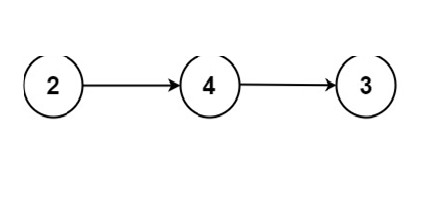
// Function to print the linked list

void printList(Node \*head) {

Node \*current = head;

while

9. You are given with the following linked list



The digits are stored in the above order, you are asked to print the list in reverse order.

Answer:

#include <stdio.h>

#include <stdlib.h>

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

typedef struct Node {

int value;

struct Node \*next;

} Node;

// Define the structure for a stack

typedef struct Stack {

int top;

int capacity;

int \*array;

} Stack;

// Function to create a new node

Node\* createNode(int value) {

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

newNode->value = value;

newNode->next = NULL;

return newNode;

}

// Function to create a stack

Stack\* createStack(int capacity) {

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

stack->capacity = capacity;

stack->top = -1;

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

return stack;

}

// Function to check if the stack is full

int isFull(Stack \*stack) {

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

}

10.Given two sorted arrays nums1 and nums2 of size m and n respectively, return the sum of these two arrays

Example 1:

Input: nums1 = [1,3], nums2 = [2]

Output: 6

Example 2:

Input: nums1 = [1,2], nums2 = [3,4]

Output: 10

Answer:

#include <stdio.h>

// Function to calculate the sum of elements in an array

int sumArray(int \*arr, int size) {

int sum = 0;

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

sum += arr[i];

}

return sum;

}

// Function to calculate the sum of two sorted arrays

int sumOfTwoArrays(int \*nums1, int m, int \*nums2, int n) {

// Calculate the sum of both arrays

int sum1 = sumArray(nums1, m);

int sum2 = sumArray(nums2, n);