**12/08/2024**

**22. 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]**

Sol

#include <stdio.h>

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

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

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

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

int temp = arr[j];

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

arr[j+1] = temp;

}

}

}

}

int main() {

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

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

bubbleSortDescending(arr, n);

printf("Output: [");

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

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

if (i < n - 1) {

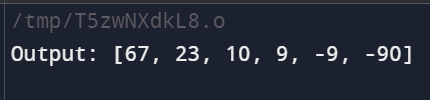
printf(", ");

}

} printf("]\n");

return 0;

}



**23.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.**

sol

#include <stdio.h>

int main() {

int N;

long long factorial = 1;

// Input

printf("Enter a positive integer: ");

scanf("%d", &N);

// Calculate factorial

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

factorial \*= i;

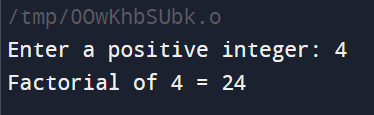
}

// Output

printf("Factorial of %d = %lld\n", N, factorial);

return 0;

}



**24.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**]

#include <stdio.h>

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

int i, j, temp;

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

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

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

temp = arr[j];

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

arr[j+1] = temp;

}

}

}

}

int main() {

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

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

bubbleSort(arr, n);

printf("Output: [");

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

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

if (i < n - 1) {

printf(", ");

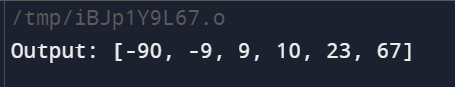
}

}

printf("]\n");

return 0;

}



25. **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","g**

**etMin"]**

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

**Sol**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

typedef struct {

int \*stack;

int \*minStack;

int topIndex;

int minIndex;

int capacity;

} MinStack;

// Function implementations (as provided in your code)

MinStack\* minStackCreate() {

MinStack \*minStack = (MinStack \*)malloc(sizeof(MinStack));

minStack->capacity = 1000;

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

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

minStack->topIndex = -1;

minStack->minIndex = -1;

return minStack;

}

void minStackPush(MinStack\* obj, int val) {

obj->stack[++(obj->topIndex)] = val;

if (obj->minIndex == -1 || val <= obj->minStack[obj->minIndex]) {

obj->minStack[++(obj->minIndex)] = val;

}

}

void minStackPop(MinStack\* obj) {

if (obj->stack[obj->topIndex] == obj->minStack[obj->minIndex]) {

obj->minIndex--;

}

obj->topIndex--;

}

int minStackTop(MinStack\* obj) {

return obj->stack[obj->topIndex];

}

int minStackGetMin(MinStack\* obj) {

return obj->minStack[obj->minIndex];

}

void minStackFree(MinStack\* obj) {

free(obj->stack);

free(obj->minStack);

free(obj);

}

int main() {

MinStack\* minStack = minStackCreate();

minStackPush(minStack, 3);

minStackPush(minStack, 5);

printf("Current Min: %d\n", minStackGetMin(minStack)); // returns 3

minStackPush(minStack, 2);

minStackPush(minStack, 1);

printf("Current Min: %d\n", minStackGetMin(minStack)); // returns 1

minStackPop(minStack);

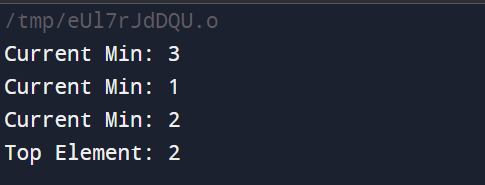
printf("Current Min: %d\n", minStackGetMin(minStack)); // returns 2

printf("Top Element: %d\n", minStackTop(minStack)); // returns 2

minStackFree(minStack);

return 0;

**}**



**26.find the factorial of a number using iterative**

**procedure Input : 3**

**sol**

#include <stdio.h>

int main() {

int number = 3;

int factorial = 1;

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

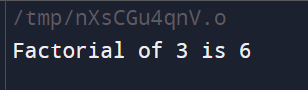
factorial \*= i;

}

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

return 0;

}



**27**. **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]**

Sol.

#include <stdio.h>

#include <stdlib.h>

struct ListNode {

int val;

struct ListNode \*next;

};

struct ListNode\* insertNode(struct ListNode\* head, int p, int n) {

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

newNode->val = p;

newNode->next = NULL;

if (n == 0) {

newNode->next = head;

return newNode;

}

struct ListNode\* current = head;

for (int i = 0; i < n - 1 && current != NULL; i++) {

current = current->next;

}

if (current != NULL) {

newNode->next = current->next;

current->next = newNode;

}

return head;

}

void printList(struct ListNode\* head) {

struct ListNode\* current = head;

while (current != NULL) {

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

current = current->next;

}

printf("\n");

}

int main() {

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

head->val = 1;

head->next = (struct ListNode\*)malloc(sizeof(struct ListNode));

head->next->val = 3;

head->next->next = (struct ListNode\*)malloc(sizeof(struct ListNode));

head->next->next->val = 2;

head->next->next->next = (struct ListNode\*)malloc(sizeof(struct ListNode));

head->next->next->next->val = 3;

head->next->next->next->next = (struct ListNode\*)malloc(sizeof(struct ListNode));

head->next->next->next->next->val = 4;

head->next->next->next->next->next = (struct ListNode\*)malloc(sizeof(struct ListNode));

head->next->next->next->next->next->val = 5;

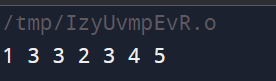
head->next->next->next->next->next->next = NULL;

head = insertNode(head, 3, 2);

printList(head);

return 0;

}



**28.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]**

Sol.

struct ListNode {

int val;

struct ListNode \*next;

};

struct ListNode\* reverseBetween(struct ListNode\* head, int left, int right) {

if (!head || left == right) return head;

struct ListNode dummy;

dummy.next = head;

struct ListNode\* prev = &dummy;

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

prev = prev->next;

}

struct ListNode\* curr = prev->next;

struct ListNode\* tail = curr;

for (int i = 0; i < right - left; i++) {

struct ListNode\* temp = curr->next;

curr->next = temp->next;

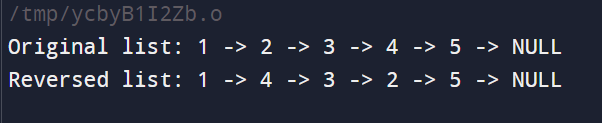
temp->next = prev->next;

prev->next = temp;

}

return dummy.next;

}



**29.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.**

**Sol.**

#include <stdio.h>

#include <stdlib.h>

struct ListNode {

int val;

struct ListNode \*next;

};

struct ListNode\* createNode(int val) {

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

newNode->val = val;

newNode->next = NULL;

return newNode;

}

void printReverse(struct ListNode\* head) {

if (head == NULL) {

return; // Base case: if the list is empty, return

}

printReverse(head->next); // Recursive call with the next node

printf("%d -> ", head->val); // Print the current node's value after returning from recursion

}

void freeList(struct ListNode\* head) {

while (head != NULL) {

struct ListNode\* temp = head;

head = head->next;

free(temp);

}

}

int main() {

// Create the linked list: 1 -> 2 -> 3 -> 4 -> 5

struct ListNode\* head = createNode(1);

head->next = createNode(2);

head->next->next = createNode(3);

head->next->next->next = createNode(4);

head->next->next->next->next = createNode(5);

// Print the linked list in reverse order

printf("Linked list in reverse order: ");

printReverse(head);

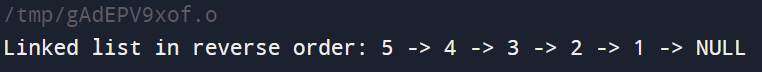
printf("NULL\n"); // Indicate the end of the list

// Free the allocated memory

freeList(head);

return 0;

}



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

**Sol.**

#include <stdio.h>

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

int sum = 0;

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

sum += nums1[i];

}

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

sum += nums2[j];

}

return sum;

}

int main() {

int nums1[] = {1, 2, 3};

int nums2[] = {4, 5, 6};

int m = sizeof(nums1) / sizeof(nums1[0]);

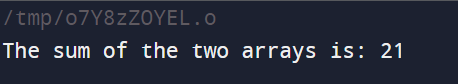
int n = sizeof(nums2) / sizeof(nums2[0]);

int result = sumSortedArrays(nums1, m, nums2, n);

printf("The sum of the two arrays is: %d\n", result);

return 0;

}



**21. 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**

Sol.

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX\_SIZE 100

typedef struct {

int data[MAX\_SIZE];

int top;

} Stack;

typedef struct {

Stack stack1;

Stack stack2;

} MyQueue;

// Function to initialize a stack

void stack\_init(Stack \*stack) {

stack->top = -1;

}

// Function to check if a stack is empty

bool stack\_is\_empty(Stack \*stack) {

return stack->top == -1;

}

// Function to push an element onto the stack

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

if (stack->top < MAX\_SIZE - 1) {

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

} else {

printf("Stack overflow\n");

}

}

// Function to pop an element from the stack

int stack\_pop(Stack \*stack) {

if (!stack\_is\_empty(stack)) {

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

} else {

printf("Stack underflow\n");

return -1;

}

}

// Function to get the top element of the stack without popping it

int stack\_peek(Stack \*stack) {

if (!stack\_is\_empty(stack)) {

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

} else {

printf("Stack is empty\n");

return -1;

}

}

// Function to initialize the queue

void myQueue\_init(MyQueue \*queue) {

stack\_init(&queue->stack1);

stack\_init(&queue->stack2);

}

// Function to push an element onto the queue

void myQueue\_push(MyQueue \*queue, int x) {

stack\_push(&queue->stack1, x);

}

// Function to pop an element from the queue

int myQueue\_pop(MyQueue \*queue) {

if (stack\_is\_empty(&queue->stack2)) {

while (!stack\_is\_empty(&queue->stack1)) {

stack\_push(&queue->stack2, stack\_pop(&queue->stack1));

}

}

return stack\_pop(&queue->stack2);

}

// Function to get the front element of the queue

int myQueue\_peek(MyQueue \*queue) {

if (stack\_is\_empty(&queue->stack2)) {

while (!stack\_is\_empty(&queue->stack1)) {

stack\_push(&queue->stack2, stack\_pop(&queue->stack1));

}

}

return stack\_peek(&queue->stack2);

}

// Function to check if the queue is empty

bool myQueue\_empty(MyQueue \*queue) {

return stack\_is\_empty(&queue->stack1) && stack\_is\_empty(&queue->stack2);

}

// Main function to test the MyQueue implementation

int main() {

MyQueue queue;

myQueue\_init(&queue);

myQueue\_push(&queue, 1);

myQueue\_push(&queue, 2);

myQueue\_push(&queue, 3);

printf("Front element: %d\n", myQueue\_peek(&queue));

printf("Popped element: %d\n", myQueue\_pop(&queue));

printf("Is queue empty? %s\n", myQueue\_empty(&queue) ? "Yes" : "No");

printf("Front element: %d\n", myQueue\_peek(&queue));

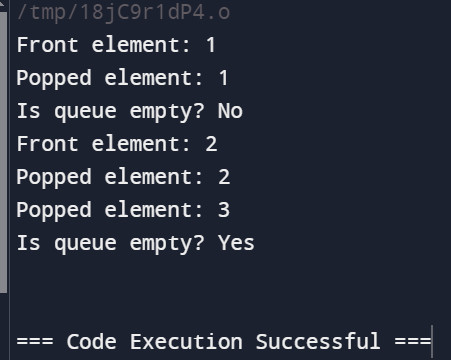
printf("Popped element: %d\n", myQueue\_pop(&queue));

printf("Popped element: %d\n", myQueue\_pop(&queue));

printf("Is queue empty? %s\n", myQueue\_empty(&queue) ? "Yes" : "No");

return 0;

}



**14/08/2024**

**10.Given a string s, find the frequency of**

**characters Example 1:**

**Input: s = "tree"**

**Sol.**

#include <stdio.h>

#include <string.h>

void characterFrequency(char \*s) {

int freq[256] = {0};

int length = strlen(s);

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

freq[(int)s[i]]++;

}

for (int i = 0; i < 256; i++) {

if (freq[i] > 0) {

printf("%c->%d, ", i, freq[i]);

}

}

}

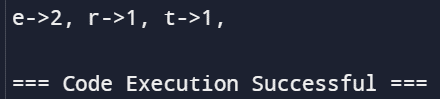
int main() {

char s[] = "tree";

characterFrequency(s);

return 0;

}



**11. Given an unsorted array arr[] with both positive and negative elements,**

**the task is to find the smallest positive number missing from the array.**

**Input: arr[] = {2, 3, 7, 6, 8, -1, -10, 15}**

**Output: 1**

**Input: arr[] = { 2, 3, -7, 6, 8, 1, -10, 15 }**

**Output: 4**

**Input: arr[] = {1, 1, 0, -1, -2}**

**Output: 2**

**Sol.**

#include <stdio.h>

int findMissingPositive(int arr[], int size) {

int i;

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

while (arr[i] > 0 && arr[i] <= size && arr[arr[i] - 1] != arr[i]) {

int temp = arr[i];

arr[i] = arr[temp - 1];

arr[temp - 1] = temp;

}

}

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

if (arr[i] != i + 1) {

return i + 1;

}

}

return size + 1;

}

int main() {

int arr1[] = {2, 3, 7, 6, 8, -1, -10, 15};

int size1 = sizeof(arr1) / sizeof(arr1[0]);

printf("Output: %d\n", findMissingPositive(arr1, size1)); // Output: 1

int arr2[] = {2, 3, -7, 6, 8, 1, -10, 15};

int size2 = sizeof(arr2) / sizeof(arr2[0]);

printf("Output: %d\n", findMissingPositive(arr2, size2)); // Output: 4

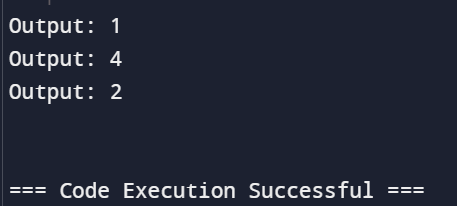
int arr3[] = {1, 1, 0, -1, -2};

int size3 = sizeof(arr3) / sizeof(arr3[0]);

printf("Output: %d\n", findMissingPositive(arr3, size3)); // Output: 2

return 0;

}



**12.Given two integer arrays preorder and inorder where preorder is the**

**preorder traversal of a binary tree and inorder is the inorder traversal**

**of the same tree,**

**construct and return the binary tree. Input: preorder = [3,9,20,15,7], inorder =**

**[9,3,15,20,7] Output: [3,9,20,null,null,15,7]**

**Sol.**

#include <stdio.h>

#include <stdlib.h>

struct TreeNode {

int val;

struct TreeNode \*left;

struct TreeNode \*right;

};

// Function to create a new tree node

struct TreeNode\* createNode(int val) {

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

node->val = val;

node->left = NULL;

node->right = NULL;

return node;

}

// Function to find the index of a value in an array

int findIndex(int\* array, int start, int end, int value) {

for (int i = start; i <= end; i++) {

if (array[i] == value) {

return i;

}

}

return -1;

}

// Recursive function to build the binary tree

struct TreeNode\* buildTreeHelper(int\* preorder, int\* inorder, int inorderStart, int inorderEnd, int\* preorderIndex) {

if (inorderStart > inorderEnd) {

return NULL;

}

// The next element in preorder[] is the root node for this subtree

int rootVal = preorder[\*preorderIndex];

(\*preorderIndex)++;

// Create the root node

struct TreeNode\* root = createNode(rootVal);

// If the tree has only one node, return it

if (inorderStart == inorderEnd) {

return root;

}

// Find the index of the root in inorder[]

int inorderIndex = findIndex(inorder, inorderStart, inorderEnd, rootVal);

// Recursively build the left and right subtrees

root->left = buildTreeHelper(preorder, inorder, inorderStart, inorderIndex - 1, preorderIndex);

root->right = buildTreeHelper(preorder, inorder, inorderIndex + 1, inorderEnd, preorderIndex);

return root;

}

// Function to build the binary tree from preorder and inorder arrays

struct TreeNode\* buildTree(int\* preorder, int preorderSize, int\* inorder, int inorderSize) {

int preorderIndex = 0;

return buildTreeHelper(preorder, inorder, 0, inorderSize - 1, &preorderIndex);

}

// Function to print the tree in level order to verify the result

void printLevelOrder(struct TreeNode\* root) {

if (root == NULL) return;

struct TreeNode\* queue[100];

int front = 0;

int rear = 0;

queue[rear++] = root

while (front < rear) {

struct TreeNode\* node = queue[front++];

if (node) {

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

queue[rear++] = node->left;

queue[rear++] = node->right;

} else {

printf("null ");

}

}

// Main function to test the buildTree function

int main() {

int preorder[] = {3, 9, 20, 15, 7};

int inorder[] = {9, 3, 15, 20, 7};

int preorderSize = sizeof(preorder) / sizeof(preorder[0]);

int inorderSize = sizeof(inorder) / sizeof(inorder[0]);

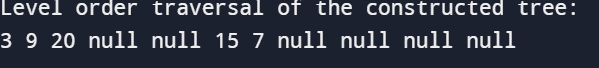
struct TreeNode\* root = buildTree(preorder, preorderSize, inorder, inorderSize);

printf("Level order traversal of the constructed tree: \n");

printLevelOrder(root);

return 0;

}



**13. Write a program to create and display a**

**linked list Example 1:**

**Nodes : 6,7,8,9**

**Output: 6->7->8->9**

**Sol.**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void displayList(struct Node\* node) {

while (node != NULL) {

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

if (node->next != NULL) {

printf("->");

}

node = node->next;

}

printf("\n");

}

int main() {

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

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

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

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

head->data = 6;

head->next = second;

second->data = 7;

second->next = third;

third->data = 8;

third->next = fourth;

fourth->data = 9;

fourth->next = NULL;

displayList(head);

free(head);

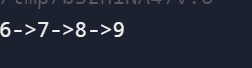
free(second);

free(third);

free(fourth);

return 0;

}



**14.Write a program to sort the below numbers in descending order using**

**bubble sort Input 4,7,9,1,2**

**Output:9,7,4,2,1**

**Sol.**

#include <stdio.h>

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

int i, j, temp;

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

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

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

temp = arr[j];

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

arr[j+1] = temp;

}

}

}

}

int main() {

int arr[] = {4, 7, 9, 1, 2};

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

bubbleSort(arr, n);

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

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

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

if (i < n - 1) {

printf(",");

}

}

return 0;

}



**15. Given an array of size N-1 such that it only contains distinct**

**integers in the range of 1 to N. Find the missing element.**

**Input:**

**N = 5**

**A[] = {1,2,3,5}**

**Output:4**

**Input N = 10**

**A[] = {6,1,2,8,3,4,7,10,5}**

**Output: 9**

**Sol.**

#include <stdio.h>

int findMissing(int A[], int N) {

int total = (N \* (N + 1)) / 2;

int sum = 0;

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

sum += A[i];

}

return total - sum;

}

int main() {

int A1[] = {1, 2, 3, 5};

int N1 = 5;

printf("%d\n", findMissing(A1, N1)); // Output: 4

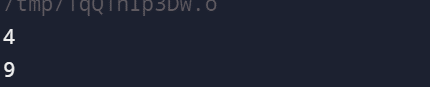
int A2[] = {6, 1, 2, 8, 3, 4, 7, 10, 5};

int N2 = 10;

printf("%d\n", findMissing(A2, N2)); // Output: 9

return 0;

}



**16. Write a program to find odd number present in the data part of**

**a node Example Linked List 1->2->3->7**

**Output: 1,3,7**

**Sol**.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void findOddNumbers(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

if (current->data % 2 != 0) {

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

}

current = current->next;

}

}

int main() {

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

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

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

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

head->data = 1;

head->next = second;

second->data = 2;

second->next = third;

third->data = 3;

third->next = fourth;

fourth->data = 7;

fourth->next = NULL;

printf("Odd numbers in the linked list: ");

findOddNumbers(head);

free(head);

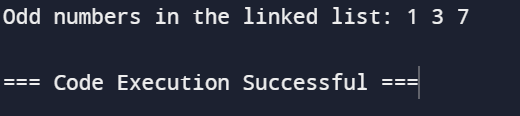
free(second);

free(third);

free(fourth);

return 0;

}



**17. Write a program to perform insert and delete operations in a**

**queue Example : 12,34,56,78**

**After insertion of 60 content of the queue is**

**12,34,56,78,60 After deletion of 12 , the contents of the**

**queue : 34,56,78,60**

**\sol.**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

struct Queue {

int items[MAX];

int front;

int rear;

};

struct Queue\* createQueue() {

struct Queue\* q = (struct Queue\*)malloc(sizeof(struct Queue));

q->front = -1;

q->rear = -1;

return q;

}

int isFull(struct Queue\* q) {

return q->rear == MAX - 1;

}

int isEmpty(struct Queue\* q) {

return q->front == -1 || q->front > q->rear;

}

void enqueue(struct Queue\* q, int value) {

if (isFull(q)) {

printf("Queue is full\n");

return;

}

if (isEmpty(q)) {

q->front = 0;

}

q->rear++;

q->items[q->rear] = value;

}

int dequeue(struct Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return -1;

}

int item = q->items[q->front];

q->front++;

return item;

}

void display(struct Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return;

}

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

printf("%d ", q->items[i]);

}

printf("\n");

}

int main() {

struct Queue\* q = createQueue();

enqueue(q, 12);

enqueue(q, 34);

enqueue(q, 56);

enqueue(q, 78);

printf("After insertion of 60, contents of the queue: ");

enqueue(q, 60);

display(q);

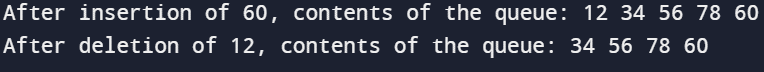
printf("After deletion of %d, contents of the queue: ", dequeue(q));

display(q)

free(q);

return 0;

}



**18. Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if**

**the input string is valid.**

**Sol.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

typedef struct {

char items[MAX];

int top;

} Stack;

void initStack(Stack\* s) {

s->top = -1;

}

int isFull(Stack\* s) {

return s->top == MAX - 1;

}

int isEmpty(Stack\* s) {

return s->top == -1;

}

void push(Stack\* s, char item) {

if (!isFull(s)) {

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

}

}

char pop(Stack\* s) {

if (!isEmpty(s)) {

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

}

return '\0';

}

int isValid(char\* s) {

Stack stack;

initStack(&stack);

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

if (s[i] == '(' || s[i] == '{' || s[i] == '[') {

push(&stack, s[i]);

} else {

if (isEmpty(&stack)) return 0;

char top = pop(&stack);

if ((s[i] == ')' && top != '(') ||

(s[i] == '}' && top != '{') ||

(s[i] == ']' && top != '[')) {

return 0;

}

}

}

return isEmpty(&stack);

}

int main() {

char s[MAX];

printf("Enter a string of parentheses: ");

scanf("%s", s);

if (isValid(s)) {

printf("The string is valid.\n");

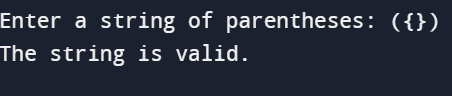
} else {

printf("The string is not valid.\n");

}

return 0;

}



**19.Given a number n, the task is to print the Fibonacci series and the sum of the**

**series using Iterative procedure.**

**input n=10**

**output Fibonacci series 0, 1, 1, 2, 3, 5, 8, 13, 21, 34**

**Sum: 88**

#include <stdio.h>

int main() {

int n = 10;

int a = 0, b = 1, sum = 0;

printf("Fibonacci series:\n");

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

printf("%d", a);

if (i != n) printf(", "); // For formatting the output

sum += a;

int next = a + b;

a = b;

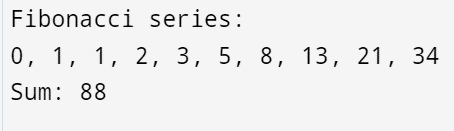
b = next;

}’

printf("\nSum: %d\n", sum);

return 0;

}



**Monday problems**

**3.Given the head of a singly linked list, return number of nodes present in**

**a linked Example 1:**

**1->2->3->5->8**

**Output 5**

**Sol**

#include <stdio.h>

#include <stdlib.h>

struct ListNode {

int val;

struct ListNode\* next;

};

struct ListNode\* createNode(int val) {

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

newNode->val = val;

newNode->next = NULL;

return newNode;

}

int countNodes(struct ListNode\* head) {

int count = 0;

struct ListNode\* current = head;

while (current != NULL) {

count++;

current = current->next;

}

return count;

}

int main() {

struct ListNode\* head = createNode(1);

head->next = createNode(2);

head->next->next = createNode(3);

head->next->next->next = createNode(5);

head->next->next->next->next = createNode(8);

int nodeCount = countNodes(head);

printf("Number of nodes: %d\n", nodeCount);

return 0;

}



**4. Given a number n. the task is to print the Fibonacci series and the sum of the**

**series using recursion.**

**input: n=10**

**output: Fibonacci series**

#include <stdio.h>

int fibonacci(int n) {

if (n <= 1)

return n;

return fibonacci(n - 1) + fibonacci(n - 2);

}

int fibonacciSum(int n) {

if (n == 0)

return 0;

return fibonacci(n) + fibonacciSum(n - 1);

}

int main() {

int n = 10;

printf("Fibonacci series:\n");

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

printf("%d", fibonacci(i));

if (i != n - 1) printf(", "); // For formatting the output

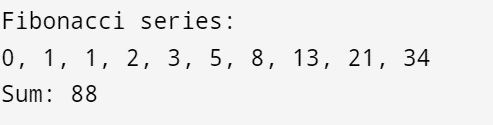
}

int sum = fibonacciSum(n - 1); // Sum of first n Fibonacci numbers

printf("\nSum: %d\n", sum);

return 0;

}



5 Given a string s, sort it in ascending order and find the starting index of repeated

character Input: s = "tree"

Output:"eert", starting index 0

Input: s = "kkj"

Sol

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

void sortDescending(char\* s) {

int n = strlen(s);

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

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

if (s[i] < s[j]) {

char temp = s[i];

s[i] = s[j];

s[j] = temp;

}

}

}

}

int findFirstRepeatIndex(char\* s) {

int n = strlen(s);

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

if (s[i] == s[i + 1]) {

return i;

}

}

return -1;

}

int main() {

char s[] = "tree";

sortDescending(s);

printf("Sorted string: \"%s\"\n", s);

int index = findFirstRepeatIndex(s);

if (index != -1) {

printf("Starting index of first repeated character: %d\n", index);

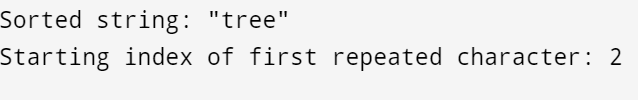
} else {

printf("No repeated characters found\n");

}

return 0;

}



**8. Given the head of a singly linked list, return true if it is a palindrome or false**

**otherwise. Example 1:**

**Input: head = [1,2,2,1]**

**Output: true**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct ListNode {

int val;

struct ListNode\* next;

};

struct ListNode\* createNode(int val) {

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

newNode->val = val;

newNode->next = NULL;

return newNode;

}

struct ListNode\* reverseList(struct ListNode\* head) {

struct ListNode\* prev = NULL;

struct ListNode\* current = head;

while (current != NULL) {

struct ListNode\* nextNode = current->next;

current->next = prev;

prev = current;

current = nextNode;

}

return prev;

}

bool isPalindrome(struct ListNode\* head) {

if (head == NULL || head->next == NULL)

return true;

struct ListNode \*slow = head, \*fast = head;

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

}

struct ListNode\* secondHalf = reverseList(slow);

struct ListNode\* firstHalf = head;

while (secondHalf != NULL) {

if (firstHalf->val != secondHalf->val)

return false;

firstHalf = firstHalf->next;

secondHalf = secondHalf->next;

}

return true;

}

int main() {

struct ListNode\* head = createNode(1);

head->next = createNode(2);

head->next->next = createNode(2);

head->next->next->next = createNode(1);

if (isPalindrome(head)) {

printf("Output: true\n");

} else {

printf("Output: false\n");

}

return 0;

}

