21) #include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct {

int \*stack1;

int \*stack2;

int top1;

int top2;

int size;

} MyQueue;

MyQueue\* myQueueCreate(int size) {

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

queue->stack1 = (int\*)malloc(size \* sizeof(int));

queue->stack2 = (int\*)malloc(size \* sizeof(int));

queue->top1 = -1;

queue->top2 = -1;

queue->size = size;

return queue;

}

void myQueuePush(MyQueue\* obj, int x) {

obj->stack1[++(obj->top1)] = x;

}

int myQueuePop(MyQueue\* obj) {

if (obj->top2 == -1) {

while (obj->top1 != -1) {

obj->stack2[++(obj->top2)] = obj->stack1[(obj->top1)--];

}

}

return obj->stack2[(obj->top2)--];

}

int myQueuePeek(MyQueue\* obj) {

if (obj->top2 == -1) {

while (obj->top1 != -1) {

obj->stack2[++(obj->top2)] = obj->stack1[(obj->top1)--];

}

}

return obj->stack2[obj->top2];

}

bool myQueueEmpty(MyQueue\* obj) {

return obj->top1 == -1 && obj->top2 == -1;

}

void myQueueFree(MyQueue\* obj) {

free(obj->stack1);

free(obj->stack2);

free(obj);

}

int main() {

MyQueue\* myQueue = myQueueCreate(100);

myQueuePush(myQueue, 1);

myQueuePush(myQueue, 2);

printf("Peek: %d\n", myQueuePeek(myQueue));

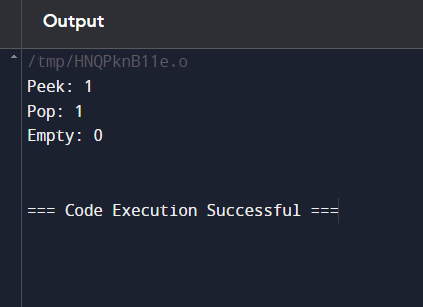
printf("Pop: %d\n", myQueuePop(myQueue));

printf("Empty: %d\n", myQueueEmpty(myQueue));

myQueueFree(myQueue);

return 0;

}



22) #include <stdio.h>

void bubbleSortDescending(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]) {

// Swap arr[j] and 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]);

bubbleSortDescending(arr, n);

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

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

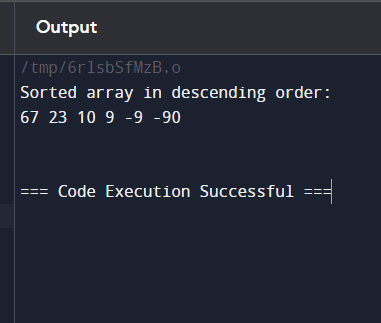
printf("%d ", arr[i]);

}

printf("\n");

return 0;

}



23) #include <stdio.h>

int main() {

int N, i;

unsigned long long factorial = 1;

printf("Enter a positive integer: ");

scanf("%d", &N);

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

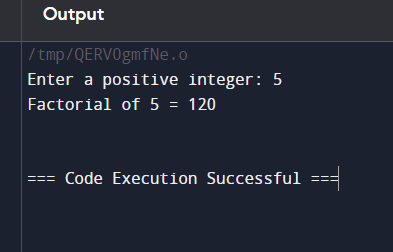
factorial \*= i;

}

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

return 0;

}



24) #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]);

int i;

bubbleSort(arr, n);

printf("Sorted array: [");

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

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

if (i < n-1) {

printf(", ");

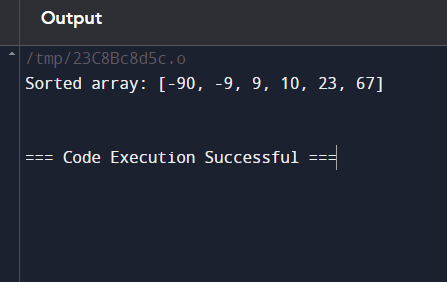
}

}

printf("]\n");

return 0;

}



25) #include <stdio.h>

#include <stdlib.h>

#include <limits.h>

typedef struct {

int \*data;

int \*minData;

int top;

int capacity;

} MinStack;

MinStack\* minStackCreate() {

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

stack->capacity = 10000;

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

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

stack->top = -1;

return stack;

}

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

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

if (stack->top == 0) {

stack->minData[stack->top] = val;

} else {

stack->minData[stack->top] = (val < stack->minData[stack->top - 1]) ? val : stack->minData[stack->top - 1];

}

}

void minStackPop(MinStack\* stack) {

if (stack->top >= 0) {

stack->top--;

}

}

int minStackTop(MinStack\* stack) {

if (stack->top >= 0) {

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

}

return INT\_MIN;

}

int minStackGetMin(MinStack\* stack) {

if (stack->top >= 0) {

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

}

return INT\_MIN;

}

void minStackFree(MinStack\* stack) {

free(stack->data);

free(stack->minData);

free(stack);

}

int main() {

MinStack\* minStack = minStackCreate();

minStackPush(minStack, -2);

minStackPush(minStack, 0);

minStackPush(minStack, -3);

printf("Minimum: %d\n", minStackGetMin(minStack));

minStackPop(minStack);

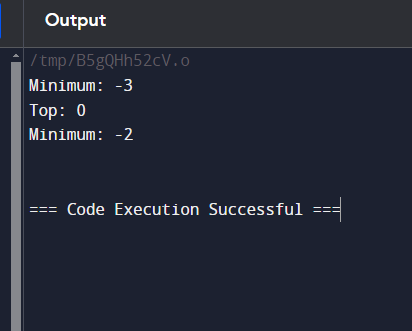
printf("Top: %d\n", minStackTop(minStack));

printf("Minimum: %d\n", minStackGetMin(minStack));

minStackFree(minStack);

return 0;

}



26) ) #include <stdio.h>

int main() {

int N, i;

unsigned long long factorial = 1;

printf("Enter a positive integer: ");

scanf("%d", &N);

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

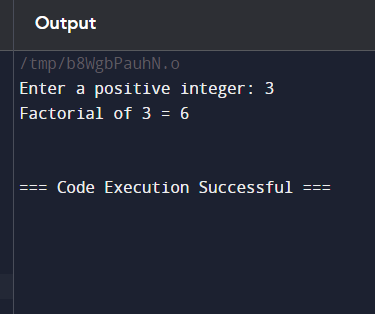
factorial \*= i;

}

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

return 0;

}



27) #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;

}

struct ListNode\* insertAtNthPosition(struct ListNode\* head, int val, int n) {

struct ListNode\* newNode = createNode(val);

if (n == 1) {

newNode->next = head;

return newNode;

}

struct ListNode\* current = head;

struct ListNode\* previous = NULL;

int i = 1;

while (current != NULL && i < n) {

previous = current;

current = current->next;

i++;

}

previous->next = newNode;

newNode->next = current;

return head;

}

void printList(struct ListNode\* head) {

struct ListNode\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

struct ListNode\* head = createNode(1);

head->next = createNode(3);

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

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

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

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

int p = 3, n = 2;

head = insertAtNthPosition(head, p, n);

printList(head);

head = createNode(1);

p = 0, n = 1;

head = insertAtNthPosition(head, p, n);

printList(head);

head = createNode(1);

head->next = createNode(2);

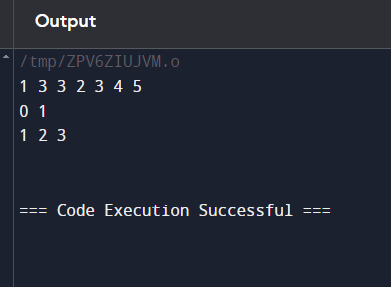
p = 3, n = 3;

head = insertAtNthPosition(head, p, n);

printList(head);

return 0;

}



28) #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;

}

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

if (head == NULL || left == right) {

return head;

}

struct ListNode\* dummy = createNode(0);

dummy->next = head;

struct ListNode\* prev = dummy;

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

prev = prev->next;

}

struct ListNode\* start = prev->next;

struct ListNode\* then = start->next;

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

start->next = then->next;

then->next = prev->next;

prev->next = then;

then = start->next;

}

head = dummy->next;

free(dummy);

return head;

}

void printList(struct ListNode\* head) {

struct ListNode\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

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);

int left = 2, right = 4;

head = reverseBetween(head, left, right);

printList(head);

head = createNode(5);

left = 1, right = 1;

head = reverseBetween(head, left, right);

printList(head);

head = createNode(10);

head->next = createNode(20);

head->next->next = createNode(30);

head->next->next->next = createNode(40);

head->next->next->next->next = createNode(50); head->next->next->next->next->next = createNode(60);

head->next->next->next->next->next->next = createNode(70);

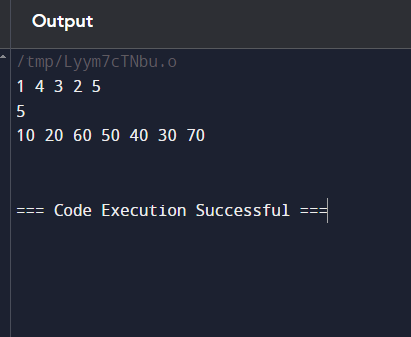
left = 3, right = 6;

head = reverseBetween(head, left, right);

printList(head);

return 0;

}



29) #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;

}

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

struct ListNode\* prev = NULL;

struct ListNode\* current = head;

struct ListNode\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current

current = next;

}

return prev;

}

void printList(struct ListNode\* head) {

struct ListNode\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

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);

printf("Original List: ");

printList(head);

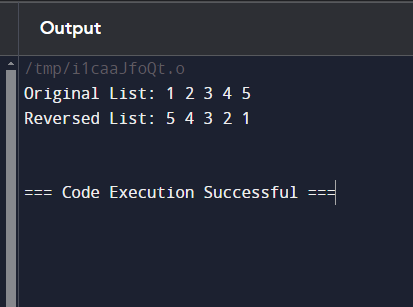
head = reverseList(head);

printf("Reversed List: ");

printList(head);

return 0;

}



30) #include <stdio.h>

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

int sum = 0;

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

sum += nums1[i];

}

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

sum += nums2[i];

}

return sum;

}

int main() {

int nums1\_1[] = {1, 3};

int nums2\_1[] = {2};

int result1 = sumOfArrays(nums1\_1, 2, nums2\_1, 1);

printf("Output: %d\n", result1);

int nums1\_2[] = {1, 2};

int nums2\_2[] = {3, 4};

int result2 = sumOfArrays(nums1\_2, 2, nums2\_2, 2);

printf("Output: %d\n", result2);

return 0;

}

