21) #include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct {

int \*data;

int top;

int capacity;

} Stack;

Stack\* createStack(int capacity) {

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

stack->capacity = capacity;

stack->top = -1;

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

return stack;

}

bool isEmpty(Stack \*stack) {

return stack->top == -1;

}

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

if (stack->top < stack->capacity - 1) {

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

}

}

int pop(Stack \*stack) {

if (!isEmpty(stack)) {

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

}

return -1;

}

int peek(Stack \*stack) {

if (!isEmpty(stack)) {

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

}

return -1;

}

typedef struct {

Stack \*stack\_in;

Stack \*stack\_out;

} MyQueue;

MyQueue\* myQueueCreate() {

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

queue->stack\_in = createStack(100);

queue->stack\_out = createStack(100);

return queue;

}

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

push(queue->stack\_in, x);

}

void transferStacks(MyQueue \*queue) {

if (isEmpty(queue->stack\_out)) {

while (!isEmpty(queue->stack\_in)) {

push(queue->stack\_out, pop(queue->stack\_in));

}

}

}

int myQueuePop(MyQueue \*queue) {

transferStacks(queue);

return pop(queue->stack\_out);

}

int myQueuePeek(MyQueue \*queue) {

transferStacks(queue);

return peek(queue->stack\_out);

}

bool myQueueEmpty(MyQueue \*queue) {

return isEmpty(queue->stack\_in) && isEmpty(queue->stack\_out);

}

void myQueueFree(MyQueue \*queue) {

free(queue->stack\_in->data);

free(queue->stack\_in);

free(queue->stack\_out->data);

free(queue->stack\_out);

free(queue);

}

int main() {

MyQueue\* myQueue = myQueueCreate();

myQueuePush(myQueue, 1);

myQueuePush(myQueue, 2);

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

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

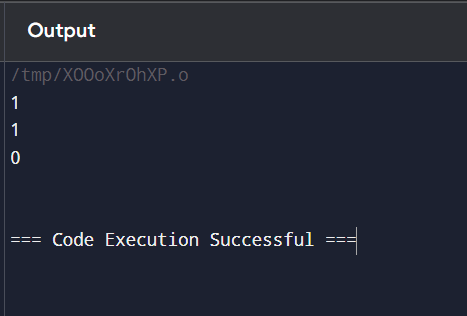
printf("%d\n", myQueueEmpty(myQueue));

myQueueFree(myQueue);

return 0;

}

Output:



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]) {

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: ");

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

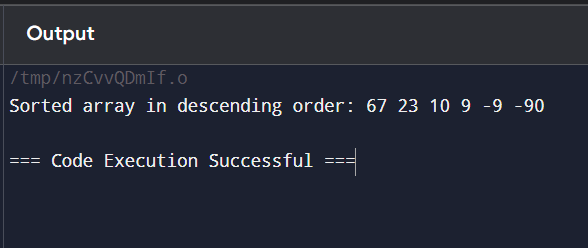
printf("%d ", arr[i]);

}

return 0;

}

Output:



23) #include <stdio.h>

int main() {

int N;

unsigned long long factorial = 1;

printf("Enter a positive integer: ");

scanf("%d", &N);

if (N < 0) {

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

} else {

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

factorial \*= i;

}

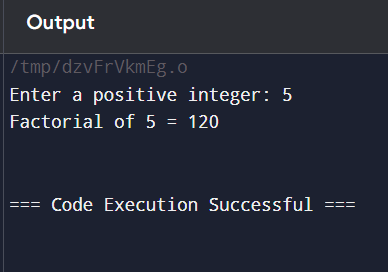
printf("Factorial of %d = %llu\n", N, factorial);

}

return 0;

}

Output:



24) #include <stdio.h>

void bubbleSortAscending(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]);

bubbleSortAscending(arr, n);

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

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

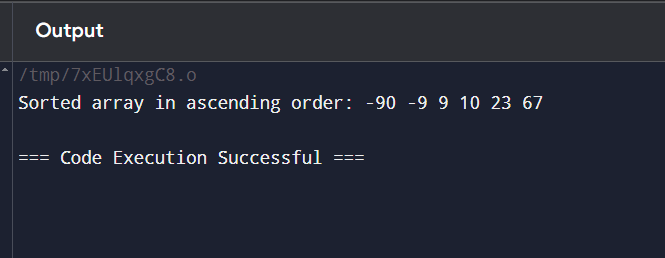
printf("%d ", arr[i]);

}

return 0;

}

Output:



25) #include <stdio.h>

#include <stdlib.h>

#include <limits.h>

typedef struct {

int \*stack;

int \*minStack;

int top;

int minTop;

int capacity;

} MinStack;

MinStack\* minStackCreate(int capacity) {

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

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

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

obj->top = -1;

obj->minTop = -1;

obj->capacity = capacity;

return obj;

}

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

if (obj->top < obj->capacity - 1) {

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

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

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

}

}

}

void minStackPop(MinStack\* obj) {

if (obj->top != -1) {

int poppedValue = obj->stack[obj->top--];

if (poppedValue == obj->minStack[obj->minTop]) {

obj->minTop--;

}

}

}

int minStackTop(MinStack\* obj) {

if (obj->top != -1) {

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

}

return INT\_MIN;

}

int minStackGetMin(MinStack\* obj) {

if (obj->minTop != -1) {

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

}

return INT\_MIN;

}

void minStackFree(MinStack\* obj) {

free(obj->stack);

free(obj->minStack);

free(obj);

}

int main() {

MinStack\* minStack = minStackCreate(100);

minStackPush(minStack, -2);

minStackPush(minStack, 0);

minStackPush(minStack, -3);

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

minStackPop(minStack);

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

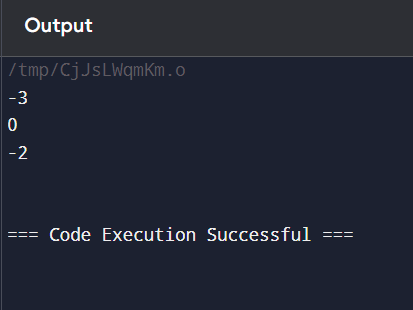
printf("%d\n", minStackGetMin(minStack));

minStackFree(minStack);

return 0;

}

Output:



26) #include <stdio.h>

int main() {

int n;

unsigned long long factorial = 1;

printf("Enter a positive integer: ");

scanf("%d", &n);

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

factorial \*= i;

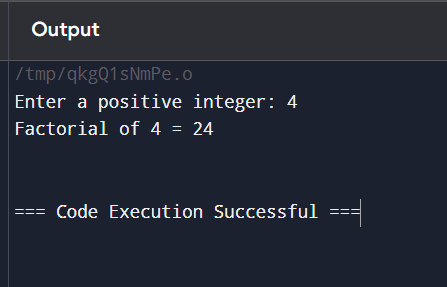
}

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

return 0;

}

Output:



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\* insertAtPosition(struct ListNode\* head, int value, int position) {

struct ListNode\* newNode = createNode(value);

if (position == 0) {

newNode->next = head;

return newNode;

}

struct ListNode\* current = head;

int currentPosition = 0;

while (current != NULL && currentPosition < position - 1) {

current = current->next;

currentPosition++;

}

if (current != NULL) {

newNode->next = current->next;

current->next = newNode;

} else {

struct ListNode\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->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\* head1 = createNode(1);

head1->next = createNode(3);

head1->next->next = createNode(2);

head1->next->next->next = createNode(3);

head1->next->next->next->next = createNode(4);

head1->next->next->next->next->next = createNode(5);

head1 = insertAtPosition(head1, 3, 2);

printList(head1);

struct ListNode\* head2 = createNode(1);

head2 = insertAtPosition(head2, 0, 0);

printList(head2);

struct ListNode\* head3 = createNode(1);

head3->next = createNode(2);

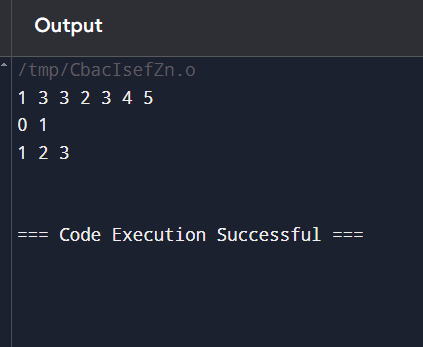
head3 = insertAtPosition(head3, 3, 3);

printList(head3);

return 0;

}

Output:



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 || 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\* curr = prev->next;

struct ListNode\* next = NULL;

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

next = curr->next;

curr->next = next->next;

next->next = prev->next;

prev->next = next;

}

return dummy->next;

}

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\* head1 = createNode(1);

head1->next = createNode(2);

head1->next->next = createNode(3);

head1->next->next->next = createNode(4);

head1->next->next->next->next = createNode(5);

head1 = reverseBetween(head1, 2, 4);

printList(head1);

struct ListNode\* head2 = createNode(5);

head2 = reverseBetween(head2, 1, 1);

printList(head2);

struct ListNode\* head3 = createNode(10);

head3->next = createNode(20);

head3->next->next = createNode(30);

head3->next->next->next = createNode(40);

head3->next->next->next->next = createNode(50);

head3->next->next->next->next->next = createNode(60);

head3->next->next->next->next->next->next = createNode(70);

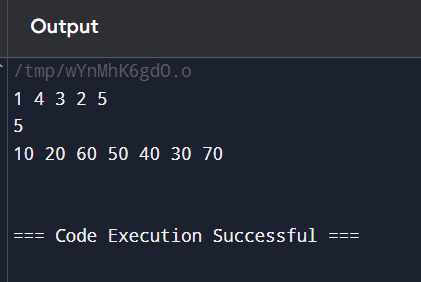
head3 = reverseBetween(head3, 3, 6);

printList(head3);

return 0;

}

Output:



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;

}

void printListInReverse(struct ListNode\* head) {

if (head == NULL) {

return;

}

printListInReverse(head->next);

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

}

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("Linked list in reverse order: ");

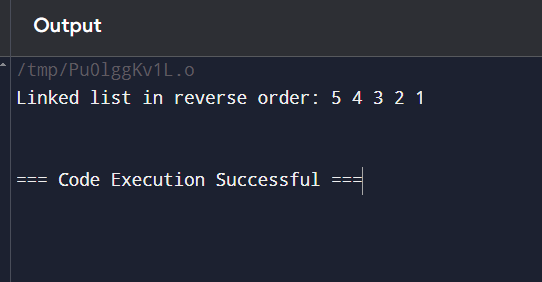
printListInReverse(head);

printf("\n");

return 0;

}

Output:



30) #include <stdio.h>

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

int sum = 0;

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

sum += array[i];

}

return sum;

}

int main() {

int nums1[] = {1, 3};

int nums2[] = {2};

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

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

int totalSum = sumArray(nums1, m) + sumArray(nums2, n);

printf("Sum of arrays: %d\n", totalSum);

int nums3[] = {1, 2};

int nums4[] = {3, 4};

int m2 = sizeof(nums3) / sizeof(nums3[0]);

int n2 = sizeof(nums4) / sizeof(nums4[0]);

totalSum = sumArray(nums3, m2) + sumArray(nums4, n2);

printf("Sum of arrays: %d\n", totalSum);

return 0;

}

Output:

