27/8/24

1.

```
#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 maxSize) {
    MyQueue* queue = (MyQueue*)malloc(sizeof(MyQueue));
    queue->stack1 = (int*)malloc(maxSize * sizeof(int));
    queue->stack2 = (int*)malloc(maxSize * sizeof(int));
    queue->top1 = -1;
    queue->top2 = -1;
    queue->size = maxSize;
    return queue;
}

void push(MyQueue* obj, int x) {
    obj->top1++;
```

```
    obj->stack1[obj->top1] = x;
}

int pop(MyQueue* obj) {
    if (obj->top2 == -1) {
        while (obj->top1 != -1) {
            obj->top2++;
            obj->stack2[obj->top2] = obj->stack1[obj->top1];
            obj->top1--;
        }
    }
    int frontElement = obj->stack2[obj->top2];
    obj->top2--;
    return frontElement;
}

int peek(MyQueue* obj) {
    if (obj->top2 == -1) {
        while (obj->top1 != -1) {
            obj->top2++;
            obj->stack2[obj->top2] = obj->stack1[obj->top1];
            obj->top1--;
        }
    }
    return obj->stack2[obj->top2];
```

```
}

bool empty(MyQueue* obj) {
    return obj->top1 == -1 && obj->top2 == -1;
}

void myQueueFree(MyQueue* obj) {
    free(obj->stack1);
    free(obj->stack2);
    free(obj);
}
int main() {
    MyQueue* queue = myQueueCreate(10);
    push(queue, 1);
    push(queue, 2);
    printf("Peek: %d\n", peek(queue));
    printf("Pop: %d\n", pop(queue));
    printf("Is empty: %d\n", empty(queue));
    myQueueFree(queue);
    return 0;
}
```

OUTPUT:

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


=== Code Execution Successful ===
```

2.

```
#include <stdio.h>

void bubbleSort(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]);
    bubbleSort(arr, n);
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT:

```
67 23 10 9 -9 -90

=== Code Execution Successful ===
```

3.

```
#include <stdio.h>
int main() {
    int N = 4, factorial = 1;
    for (int i = 1; i <= N; i++) factorial *= i;
    printf("%d\n", factorial);
    return 0;
}
```

OUTPUT:

```
24


=== Code Execution Successful ===
```

4.

```
#include <stdio.h>


void bubbleSort(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]);
    bubbleSort(arr, n);
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT:

```
-90 -9 9 10 23 67

=== Code Execution Successful ===
```

5.

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
typedef struct {
    int *stack;
    int *minStack;
    int top;
} MinStack;
MinStack* minStackCreate() {
```

```
    MinStack *ms = malloc(sizeof(MinStack));
    ms->stack = malloc(100 * sizeof(int));
    ms->minStack = malloc(100 * sizeof(int));
    ms->top = -1;
    return ms;
}
void minStackPush(MinStack* obj, int val) {
    obj->stack[++obj->top] = val;
    if (obj->top == 0 || val < obj->minStack[obj->top - 1]) {
        obj->minStack[obj->top] = val;
    } else {
        obj->minStack[obj->top] = obj->minStack[obj->top - 1];
    }
}
void minStackPop(MinStack* obj) {
    obj->top--;
}
int minStackTop(MinStack* obj) {
    return obj->stack[obj->top];
}
int minStackGetMin(MinStack* obj) {
    return obj->minStack[obj->top];
}
void minStackFree(MinStack* obj) {
    free(obj->stack);
    free(obj->minStack);
    free(obj);
```

}

OUTPUT:

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

6.

```
#include <stdio.h>


int main() {
    int n = 3, factorial = 1;
    for(int i = 1; i <= n; i++) factorial *= i;
    printf("%d\n", factorial);
    return 0;
}
```

OUTPUT:

```
6


=== Code Execution Successful ===
```

7.

```
#include <stdio.h>
#include <stdlib.h>


struct Node {
    int data;
    struct Node* next;
```

```
};

struct Node* insertNth(struct Node* head, int n, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    if (n == 0) {
        newNode->next = head;
        return newNode;
    }
    struct Node* current = head;
    for (int i = 0; i < n - 1 && current; i++)
        current = current->next;
    newNode->next = current;
    if (current) {
        struct Node* prev = head;
        for (int i = 0; i < n - 1; i++)
            prev = prev->next;
        prev->next = newNode;
    }
    return head;
}

int main() {
    struct Node* head = NULL;
    return 0;
}
```

OUTPUT:

```
[0,1]

=== Code Execution Successful ===
```

8.

```
#include <stdio.h>
#include <stdlib.h>

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

    for (int i = 1; i < left; i++) prev = prev->next;

    struct ListNode *curr = prev->next, *tail = curr;
    for (int i = 0; i < right - left; i++) {
        struct ListNode *next = curr->next;
        curr->next = next->next;
```

```
            next->next = prev->next;
            prev->next = next;
        }

        return dummy->next;
}

struct ListNode* createNode(int val) {
        struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
        newNode->val = val;
        newNode->next = NULL;
        return newNode;
}

void printList(struct ListNode* head) {
        struct ListNode* current = head;
        while (current) {
            printf("%d -> ", current->val);
            current = current->next;
        }
        printf("NULL\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 = reverseBetween(head, 2, 4);

    printf("Reversed List between 2 and 4: ");
    printList(head);

    return 0;
}
```

OUTPUT:

```
Original List: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
Reversed List between 2 and 4: 1 -> 4 -> 3 -> 2 -> 5 -> NULL


=== Code Execution Successful ===
```

9.

```
#include <stdio.h>

int sumArrays(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, 3}, nums2[] = {2};
    printf("%d\n", sumArrays(nums1, 2, nums2, 1));
    return 0;
}
```

OUTPUT:

```
6


=== Code Execution Successful ===
```

10.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

void printReverse(struct Node* head) {
    if (head == NULL) return;
    printReverse(head->next);
    printf("%d ", head->data);
}

int main() {
```

```
    struct Node* head = malloc(sizeof(struct Node));
    head->data = 1;
    head->next = malloc(sizeof(struct Node));
    head->next->data = 2;
    head->next->next = malloc(sizeof(struct Node));
    head->next->next->data = 3;
    head->next->next->next = NULL;

    printReverse(head);
    return 0;
}
```

OUTPUT:

```
3 2 1

=== Code Execution Successful ===
```