31/08/2024.

1)

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    i = 0;
    j = 0;
    k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
```

```
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
```

```
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

Output:

```
/tmp/s0pY45mxJP.o
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13


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

2)

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

struct Stack {
    int top;
    unsigned capacity;
    int* array;
```

```
};

struct Queue {
    struct Stack* stack1;
    struct Stack* stack2;
};

struct Stack* createStack(unsigned capacity) {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (int*)malloc(stack->capacity * sizeof(int));
    return stack;
}

struct Queue* createQueue(unsigned capacity) {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->stack1 = createStack(capacity);
    queue->stack2 = createStack(capacity);
    return queue;
}

int isFull(struct Stack* stack) {
    return stack->top == stack->capacity - 1;
}

int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

void push(struct Stack* stack, int item) {
```

```
    if (isFull(stack)) return;
    stack->array[++stack->top] = item;
}

int pop(struct Stack* stack) {
    if (isEmpty(stack)) return -1;
    return stack->array[stack->top--];
}

void enqueue(struct Queue* queue, int item) {
    push(queue->stack1, item);
}

int dequeue(struct Queue* queue) {
    if (isEmpty(queue->stack2)) {
        while (!isEmpty(queue->stack1)) {
            push(queue->stack2, pop(queue->stack1));
        }
    }
    return pop(queue->stack2);
}

int main() {
    struct Queue* queue = createQueue(100);
    enqueue(queue, 1);
    enqueue(queue, 2);
    enqueue(queue, 3);
    printf("%d dequeued from queue\n", dequeue(queue));
    printf("%d dequeued from queue\n", dequeue(queue));
    return 0;
}
```

Output:

```
/tmp/7OIDVR5Zox.o
1 dequeued from queue
2 dequeued from queue


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

3)

```
#include <stdio.h>

void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

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

```
int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    insertionSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

Output:

```
/tmp/9iqukQgHRw.o
5 6 11 12 13


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

4)

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

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

void insert(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
```

```
}

struct Node* intersect(struct Node* head1, struct Node* head2) {
    struct Node* result = NULL;
    struct Node* temp1 = head1;
    struct Node* temp2;

    while (temp1 != NULL) {
        temp2 = head2;
        while (temp2 != NULL) {
            if (temp1->data == temp2->data) {
                insert(&result, temp1->data);
                break;
            }
            temp2 = temp2->next;
        }
        temp1 = temp1->next;
    }
    return result;
}

void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head1 = NULL;
```

```
    struct Node* head2 = NULL;
    struct Node* intersection = NULL;

    insert(&head1, 3);
    insert(&head1, 1);
    insert(&head1, 5);
    insert(&head1, 7);

    insert(&head2, 2);
    insert(&head2, 1);
    insert(&head2, 5);
    insert(&head2, 8);

    intersection = intersect(head1, head2);

    printf("Intersection of Linked Lists: ");
    printList(intersection);

    return 0;
}
```

Output:

```
/tmp/S8qN4ZdNsh.o
Intersection of Linked Lists: 1 -> 5 -> NULL


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

5)

```
#include <stdio.h>

void mergeArrays(int arr1[], int arr2[], int merged[], int size) {
```

```
    int i = 0, j = 0, k = 0;

    while (i < size && j < size) {
        if (arr1[i] >= arr2[j]) {
            merged[k++] = arr1[i++];
        } else {
            merged[k++] = arr2[j++];
        }
    }

    while (i < size) {
        merged[k++] = arr1[i++];
    }

    while (j < size) {
        merged[k++] = arr2[j++];
    }
}

int main() {
    int size;
    printf("Enter the size of the arrays: ");
    scanf("%d", &size);

    int arr1[size], arr2[size], merged[2 * size];

    printf("Enter elements of the first array in descending order:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr1[i]);
    }
```

```
    printf("Enter elements of the second array in descending order:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr2[i]);
    }

    mergeArrays(arr1, arr2, merged, size);

    printf("Merged array in descending order:\n");
    for (int i = 0; i < 2 * size; i++) {
        printf("%d ", merged[i]);
    }

    return 0;
}
```

Output:

```
/tmp/Jr7QP4bJ2d.o
Enter the size of the arrays: 5
Enter elements of the first array in descending order:
67
59
23
21
17
Enter elements of the second array in descending order:
84
76
46
37
24
Merged array in descending order:
84 76 67 59 46 37 24 23 21 17

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

6)

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
```

```
    node->left = NULL;
    node->right = NULL;
    return node;
}

void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

int main() {
    struct Node* root = newNode(1);
```

```
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");

    printf("Preorder traversal: ");
    preorder(root);
    printf("\n");

    printf("Postorder traversal: ");
    postorder(root);
    printf("\n");

    return 0;
}
```

Output:

```
/tmp/JjAzLa1Gmr.o
Inorder traversal: 4 2 5 1 3
Preorder traversal: 1 2 4 5 3
Postorder traversal: 4 5 2 3 1


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

7)

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

```
int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Return the index of the found element
        }
    }
    return -1; // Return -1 if the element is not found
}

int main() {
    int arr[] = {34, 7, 23, 32, 5, 62};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 23;

    int result = linearSearch(arr, size, target);

    if (result != -1) {
        printf("Element found at index: %d\n", result);
    } else {
        printf("Element not found in the array.\n");
    }

    return 0;
}
```

Output:

```
/tmp/dHDenyCDNW.o
Element found at index: 2


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

8)

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

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
    int count; // Count of nodes in the subtree
};

struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    node->count = 1;
    return node;
}

void updateCount(struct Node* node) {
    if (node) {
        node->count = 1 + (node->left ? node->left->count : 0) + (node->right ? node->right->count : 0);
    }
}
```

```
struct Node* insert(struct Node* node, int data) {
    if (node == NULL) return newNode(data);
    if (data < node->data) {
        node->left = insert(node->left, data);
    } else {
        node->right = insert(node->right, data);
    }
    updateCount(node);
    return node;
}

int kthMin(struct Node* root, int k) {
    if (root == NULL) return -1;

    int leftCount = (root->left ? root->left->count : 0);

    if (k <= leftCount) {
        return kthMin(root->left, k);
    } else if (k == leftCount + 1) {
        return root->data;
    } else {
        return kthMin(root->right, k - leftCount - 1);
    }
}

int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
```

```
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    int k = 3;
    printf("The %dth minimum value in the BST is: %d\n", k, kthMin(root, k));

    return 0;
}
```

Output:

```
/tmp/sYGjNogsZW.o
The 3th minimum value in the BST is: 40


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

9)

```
#include <stdio.h>

#define ROWS 3
#define COLS 3

int main() {
    int matrix[ROWS][COLS] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    int largest = matrix[0][0];
```

```
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            if (matrix[i][j] > largest) {
                largest = matrix[i][j];
            }
        }
    }

    printf("The largest element in the matrix is: %d\n", largest);
    return 0;
}
```

Output:

```
/tmp/8WwCDE6eCh.o
The largest element in the matrix is: 9


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

10)

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

// Define the structure for a node in the linked list
struct Node {
    int data;
    struct Node* next;
};

// Function to insert a new node at the beginning of the linked list
```

```
void insertAtBeginning(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

// Function to print the linked list
void printList(struct Node* node) {
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

// Main function to demonstrate the insertion
int main() {
    struct Node* head = NULL;

    insertAtBeginning(&head, 1);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 3);

    printf("Linked list after insertion: ");
    printList(head);

    return 0;
}
```

Output:

```
/tmp/pR4AUekbZq.o
Linked list after insertion: 3 -> 2 -> 1 -> NULL


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