DAA ASSIGNMENT-03

QUE01.A.

#include <stdio.h>

#include <stdlib.h>

struct QueueElement {

    int data;

    int priority;

    int insertionOrder;

};

struct PriorityQueue {

    struct QueueElement\* elements;

    int size;

    int capacity;

};

struct PriorityQueue\* createPriorityQueue(int capacity) {

    struct PriorityQueue\* pq = (struct PriorityQueue\*)malloc(sizeof(struct PriorityQueue));

    pq->elements = (struct QueueElement\*)malloc(sizeof(struct QueueElement) \* capacity);

    pq->size = 0;

    pq->capacity = capacity;

    return pq;

}

void swap(struct QueueElement\* a, struct QueueElement\* b) {

    struct QueueElement temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void minHeapify(struct PriorityQueue\* pq, int index) {

    int left = 2 \* index + 1;

    int right = 2 \* index + 2;

    int smallest = index;

    if (left < pq->size && pq->elements[left].priority < pq->elements[smallest].priority) {

        smallest = left;

    }

    if (right < pq->size && pq->elements[right].priority < pq->elements[smallest].priority) {

        smallest = right;

    }

    if (smallest != index) {

        swap(&pq->elements[index], &pq->elements[smallest]);

        minHeapify(pq, smallest);

    }

}

void enqueue(struct PriorityQueue\* pq, int data, int priority) {

    if (pq->size >= pq->capacity) {

        printf("Queue overflow\n");

        return;

    }

    struct QueueElement newElement;

    newElement.data = data;

    newElement.priority = priority;

    newElement.insertionOrder = pq->size;

    pq->elements[pq->size] = newElement;

    pq->size++;

    int i = pq->size - 1;

    while (i > 0 && pq->elements[(i - 1) / 2].priority > pq->elements[i].priority) {

        swap(&pq->elements[(i - 1) / 2], &pq->elements[i]);

        i = (i - 1) / 2;

    }

}

int dequeue(struct PriorityQueue\* pq) {

    if (pq->size <= 0) {

        printf("Queue underflow\n");

        return -1;

    }

    int data = pq->elements[0].data;

    pq->size--;

    pq->elements[0] = pq->elements[pq->size];

    minHeapify(pq, 0);

    return data;

}

void destroyPriorityQueue(struct PriorityQueue\* pq) {

    free(pq->elements);

    free(pq);

}

int main() {

    struct PriorityQueue\* pq = createPriorityQueue(10);

    enqueue(pq, 1, 2);

    enqueue(pq, 2, 1);

    enqueue(pq, 3, 3);

    enqueue(pq, 4, 1);

    printf("Dequeued elements: ");

    while (pq->size > 0) {

        printf("%d ", dequeue(pq));

    }

    printf("\n");

    destroyPriorityQueue(pq);

    return 0;

}

OUTPUT:-



QUE01.B

#include <stdio.h>

int arr[100];

int size = -1;

void upHeapify(int i) {

    while (i > 0) {

        int p = i / 2;

        if (arr[i] > arr[p]) {

            // swap

            int temp = arr[i];

            arr[i] = arr[p];

            arr[p] = temp;

            i = p;

        } else

            break;

    }

}

void insert(int val) {

    arr[++size] = val;

    upHeapify(size);

}

int findMax() {

    return arr[0];

}

int extractMax() {

    int maxele = arr[0];

    int temp = arr[0];

    arr[0] = arr[size];

    arr[size] = temp;

    size--;

    upHeapify(size);

    return maxele;

}

void display() {

    for (int i = 0; i <= size; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

int main() {

    printf("press -1 to exist\n");

    printf("press 1 to insert\n");

    printf("press 2 to find max\n");

    printf("press 3 to extract max\n");

    printf("press 4 to display queue\n");

    int choice;

    do {

        printf("Enter choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1: {

                int val;

                printf("Enter value to insert: ");

                scanf("%d", &val);

                insert(val);

                break;

            }

            case 2: {

                int maxele = findMax();

                printf("Max element is: %d\n", maxele);

                break;

            }

            case 3: {

                int maxele = extractMax();

                printf("Max element extracted is: %d\n", maxele);

                break;

            }

            case 4: {

                display();

                break;

            }

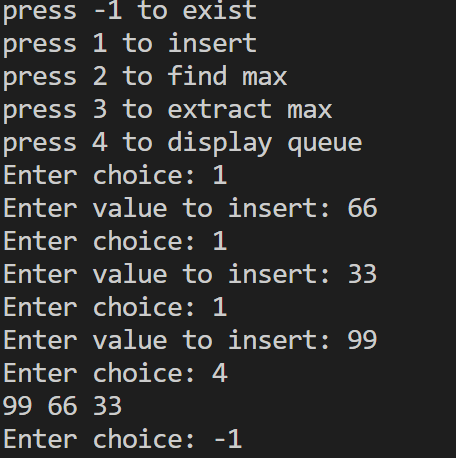
        }

    } while (choice != -1);

    return 0;

}

Output:-



QUE02.

#include <stdio.h>

#include <stdlib.h>

struct StackElement {

    int data;

    int priority;

};

struct PriorityQueueStack {

    struct StackElement\* elements;

    int size;

    int capacity;

    int priorityCounter;

};

struct PriorityQueueStack\* createPriorityQueueStack(int capacity) {

    struct PriorityQueueStack\* stack = (struct PriorityQueueStack\*)malloc(sizeof(struct PriorityQueueStack));

    stack->elements = (struct StackElement\*)malloc(sizeof(struct StackElement) \* capacity);

    stack->size = 0;

    stack->capacity = capacity;

    stack->priorityCounter = 0;

    return stack;

}

void push(struct PriorityQueueStack\* stack, int data) {

    if (stack->size >= stack->capacity) {

        printf("Stack overflow\n");

        return;

    }

    struct StackElement newElement;

    newElement.data = data;

    newElement.priority = stack->priorityCounter++;

    stack->elements[stack->size++] = newElement;

}

int pop(struct PriorityQueueStack\* stack) {

    if (stack->size <= 0) {

        printf("Stack underflow\n");

        return -1;

    }

    int maxPriority = stack->elements[0].priority;

    int maxPriorityIndex = 0;

    for (int i = 1; i < stack->size; ++i) {

        if (stack->elements[i].priority > maxPriority) {

            maxPriority = stack->elements[i].priority;

            maxPriorityIndex = i;

        }

    }

    int data = stack->elements[maxPriorityIndex].data;

    stack->size--;

    stack->elements[maxPriorityIndex] = stack->elements[stack->size];

    return data;

}

void destroyPriorityQueueStack(struct PriorityQueueStack\* stack) {

    free(stack->elements);

    free(stack);

}

int main() {

    struct PriorityQueueStack\* stack = createPriorityQueueStack(10);

    push(stack, 1);

    push(stack, 2);

    push(stack, 3);

    printf("Popped elements: ");

    while (stack->size > 0) {

        printf("%d ", pop(stack));

    }

    printf("\n");

    destroyPriorityQueueStack(stack);

    return 0;

}

Output:-

