Assignment 7

DSA LAB

2029196

Adarsh Kumar

Q1. WAP to implement a stack which will support three additional operations in addition to push and pop:

a) peekLowestElement - return the lowest element in the stack without removing it from the stack

b) peekHighestElement - return the highest element in the stack without removing it from the stack

c) peekMiddleElement - return the (size/2+1)th lowest element in the stack without removing it from the stack.

#include <stdio.h>

#define MAX 15

typedef struct

{

    int data[MAX];

    int top;

} stack;

void push(stack \**s*, int *value*);

void pop(stack \**s*, int \**temp*);

void display(stack \**s*);

int isEmpty(stack \**s*);

void peek(stack \**s*);

int input();

void peekLowestElement(stack \**s*);

void peekHighestElement(stack \**s*);

void peekMiddleElement(stack \**s*);

void init(stack \**s*)

{

*s*->top = -1; // initially we take top to be -1

}

int main(void)

{

    int cont = 1, value, b;

    stack s1;

    init(&s1);

    do

    {

        printf("\n1.PUSH\n2.POP\n3.CHECK IF EMPTY\n4.PEEK\n5.DISPLAY STACK:\n");

        int choice = input();

        switch (choice)

        {

        case 1:

        {

            printf("\nEnter element in stack: ");

            scanf("%d", &value);

            push(&s1, value);

            break;

        }

        case 2:

        {

            printf("\nPopping an item: ");

            pop(&s1, &b);

            break;

        }

        case 3:

        {

            printf("\nChecking if the stack is empty: ");

            int resp = isEmpty(&s1);

            if (resp != 1)

            {

                printf("NOT EMPTY\n");

            }

            break;

        }

        case 4:

        {

            printf("\nSelect peek option:n\n1.peekLowestElement\n2. peekHighestElement\n3. peekMiddleElement: ");

            int select = input();

            switch (select)

            {

            case 1:

            {

                peekLowestElement(&s1);

                break;

            }

            case 2:

            {

                peekHighestElement(&s1);

                break;

            }

            default:

            {

                peekMiddleElement(&s1);

                break;

            }

            }

            break;

        }

        default:

        {

            printf("\nStack entered: top-> ");

            display(&s1);

            break;

        }

        }

        printf("\nDo you wish to continue?\n1.YES\n2.No: ");

        int r;

        scanf("%d", &r);

        if (r != 1)

        {

            cont++;

        }

    } while (cont == 1);

    printf("\nDo you wish to go again?\n1.YES\n2.No: ");

    int r;

    scanf("%d", &r);

    if (r != 1)

    {

        cont == 1;

    }

}

void push(stack \**s*, int *val*)

{

    if (*s*->top == MAX - 1)

    {

        printf("\nOVERFLOW. STACK IS ALREADY FULL.");

        return;

    }

    else

    {

*s*->top++;

*s*->data[*s*->top] = *val*;

        return;

    }

}

void pop(stack \**s*, int \**temp*)

{

    if (*s*->top == -1)

    {

        printf("\nUNDERFLOW");

        return;

    }

    else

    {

        \**temp* = *s*->data[*s*->top];

        printf("\nPopped item: %d\n", \**temp*);

*s*->top--;

        return;

    }

}

int isEmpty(stack \**s*)

{

    if (*s*->top == -1)

    {

        printf("\nStack is empty");

        return 1;

    }

    else

    {

        return 0;

    }

}

void display(stack \**s*)

{

    if (*s*->top == -1)

    {

        printf("\nStack is empty");

        return;

    }

    else

    {

        int i;

        for (i = *s*->top; i >= 0; i--)

        {

            printf("\t%d\t", *s*->data[i]);

        }

    }

}

int input()

{

    int num;

    scanf("%d", &num);

    return num;

}

void peek(stack \**s*)

{

    if (*s*->top == -1)

    {

        printf("\nStack is empty");

        return;

    }

    else

    {

        printf("\ntop-> %d", *s*->data[*s*->top]);

    }

}

void peekLowestElement(stack \**s*)

{

    int min;

    if (*s*->top == -1)

    {

        printf("\nStack is empty");

        return;

    }

    else

    {

        min = *s*->data[*s*->top];

        int i;

        for (i = *s*->top; i >= 0; i--)

        {

            if (*s*->data[i] < min)

            {

                min = *s*->data[i];

            }

        }

        printf("\nLowest element of stack: %d", min);

    }

}

void peekMiddleElement(stack \**s*)

{

    if (*s*->top == -1)

    {

        printf("\nStack is empty");

        return;

    }

    else

    {

        int size = *s*->top + 1, pos = ((size / 2)), value, i;

        for (i = *s*->top; i >= 0; i--)

        {

            if (i == pos)

            {

                value = *s*->data[i];

            }

        }

        printf("\nRequired middle element: %d", value);

    }

}

void peekHighestElement(stack \**s*)

{

    int max;

    if (*s*->top == -1)

    {

        printf("\nStack is empty");

        return;

    }

    else

    {

        max = 0;

        int i;

        for (i = *s*->top; i >= 0; i--)

        {

            if (*s*->data[i] > max)

            {

                max = *s*->data[i];

            }

        }

        printf("\nHighest element of stack: %d", max);

    }

}

OUTPUT:-

Text

Description automatically generated

Q2. Write a menu driven program to implement queue operations such as Enqueue, Dequeue, Peek, Display of elements, IsEmpty, IsFull using static array.

#include <conio.h>

#include <stdio.h>

#define MAX\_SIZE 5

int deque\_arr[MAX\_SIZE];

int Left = -1;

int Right = -1;

void InsertRight()

{

    int added\_item;

    if ((Left == 0 && Right == MAX\_SIZE - 1) || (Left == Right + 1))

    {

        printf("Queue Overflow\n");

        return;

    }

    if (Left == -1)

    {

        Left = 0;

        Right = 0;

    }

    else if (Right == MAX\_SIZE - 1)

        Right = 0;

    else

        Right = Right + 1;

    printf("Input the element for adding in queue : ");

    scanf("%d", &added\_item);

    deque\_arr[Right] = added\_item;

}

void InsertLeft()

{

    int added\_item;

    if ((Left == 0 && Right == MAX\_SIZE - 1) || (Left == Right + 1))

    {

        printf("Queue Overflow \n");

        return;

    }

    if (Left == -1)

    {

        Left = 0;

        Right = 0;

    }

    else if (Left == 0)

        Left = MAX\_SIZE - 1;

    else

        Left = Left - 1;

    printf("Input the element for adding in queue : ");

    scanf("%d", &added\_item);

    deque\_arr[Left] = added\_item;

}

void DeleteLeft()

{

    if (Left == -1)

    {

        printf("Queue Under-flow\n");

        return;

    }

    printf("Element has been deleted from queue is : %d\n", deque\_arr[Left]);

    if (Left == Right)

    {

        Left = -1;

        Right = -1;

    }

    else if (Left == MAX\_SIZE - 1)

        Left = 0;

    else

        Left = Left + 1;

}

void DeleteRight()

{

    if (Left == -1)

    {

        printf("Queue Under flow\n");

        return;

    }

    printf("Element has been deleted from queue is : %d\n", deque\_arr[Right]);

    if (Left == Right)

    {

        Left = -1;

        Right = -1;

    }

    else if (Right == 0)

        Right = MAX\_SIZE - 1;

    else

        Right = Right - 1;

}

void Display()

{

    int fpos = Left, rpos = Right;

    if (Left == -1)

    {

        printf("Queue is empty\n");

        return;

    }

    printf("Queue elements :\n");

    if (fpos <= rpos)

    {

        while (fpos <= rpos)

        {

            printf("%d ", deque\_arr[fpos]);

            fpos++;

        }

    }

    else

    {

        while (fpos <= MAX\_SIZE - 1)

        {

            printf("%d ", deque\_arr[fpos]);

            fpos++;

        }

        fpos = 0;

        while (fpos <= rpos)

        {

            printf("%d ", deque\_arr[fpos]);

            fpos++;

        }

    }

    printf("\n");

}

void Input()

{

    int Option = 0;

    do // while(Option<0 || Option>5)

    {

        printf("1.Insert at Right\n");

        printf("2.Delete from Left\n");

        printf("3.Delete from Right\n");

        printf("4.Display\n");

        printf("5.Quit\n");

        printf("Enter your choice : ");

        scanf("%d", &Option);

        switch (Option)

        {

        case 1:

            InsertRight();

            break;

        case 2:

            DeleteLeft();

            break;

        case 3:

            DeleteRight();

            break;

        case 4:

            Display();

            break;

        case 5:

            break;

        default:

            printf("Wrong Option\n");

        }

    } while (Option != 5);

}

void Output()

{

    int Option = 0;

    do // while(Option<=0 || Option>5)

    {

        printf("1.Insert at Right\n");

        printf("2.Insert at Left\n");

        printf("3.Delete from Left\n");

        printf("4.Display\n");

        printf("5.Quit\n");

        printf("Enter your choice : ");

        scanf("%d", &Option);

        switch (Option)

        {

        case 1:

            InsertRight();

            break;

        case 2:

            InsertLeft();

            break;

        case 3:

            DeleteLeft();

            break;

        case 4:

            Display();

            break;

        case 5:

            break;

        default:

            printf("Wrong Option\n");

        }

    } while (Option != 5);

}

main()

{

    int Option = 0;

    printf("1.Input restricted dequeue\n");

    printf("2.Output restricted dequeue\n");

    printf("Enter your choice : ");

    scanf("%d", &Option);

    switch (Option)

    {

    case 1:

        Input();

        break;

    case 2:

        Output();

        break;

    default:

        printf("Wrong Option\n");

    }

}

OUTPUT:-

Text

Description automatically generated

Q3 Write a menu driven program to implement queue operations such as Enqueue, Dequeue, Peek, Display of elements, IsEmpty using linked list.

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*next;

} \* front, \*back;

void initialize()

{

    front = back = NULL;

}

int getQueueSize()

{

    struct node \*temp = front;

    int count = 0;

    if (front == NULL && back == NULL)

        return 0;

    while (temp != back)

    {

        count++;

        temp = temp->next;

    }

    if (temp == back)

        count++;

    return count;

}

int getFrontElement()

{

    return front->data;

}

int getBackElement()

{

    return back->data;

}

void isEmpty()

{

    if (front == NULL && back == NULL)

        printf("Empty Queue\n");

    else

        printf("Queue is not Empty\n");

}

void enqueue(int *num*)

{

    struct node \*temp;

    temp = (struct node \*)malloc(sizeof(struct node));

    temp->data = *num*;

    temp->next = NULL;

    if (back == NULL)

    {

        front = back = temp;

    }

    else

    {

        back->next = temp;

        back = temp;

    }

}

void dequeue()

{

    struct node \*temp;

    if (front == NULL)

    {

        printf("\nQueue is Empty \n");

        return;

    }

    else

    {

        temp = front;

        front = front->next;

        if (front == NULL)

        {

            back = NULL;

        }

        printf("Removed Element : %d\n", temp->data);

        free(temp);

    }

}

void printQueue()

{

    struct node \*temp = front;

    if ((front == NULL) && (back == NULL))

    {

        printf("Queue is Empty\n");

        return;

    }

    while (temp != NULL)

    {

        printf("%d", temp->data);

        temp = temp->next;

        if (temp != NULL)

            printf("-->");

    }

}

int main()

{

    initialize();

    enqueue(2);

    enqueue(0);

    enqueue(2);

    enqueue(9);

    enqueue(1);

    enqueue(9);

    enqueue(6);

    printQueue();

    printf("\nSize of Queue : %d\n", getQueueSize());

    printf("Front Element : %d\n", getFrontElement());

    printf("Rear Element : %d\n", getBackElement());

    dequeue();

    dequeue();

    dequeue();

    dequeue();

    dequeue();

    dequeue();

    return 0;

}

OUTPUT:-

Text

Description automatically generated

Q4. WAP using a function to reverse a queue by using stack..

Text, letter

Description automatically generated

OUTPUT:-

Text

Description automatically generated

Q5. Write a menu driven program to implement circular queue operations such as Enqueue, Dequeue, Peek, Display of elements, IsEmpty, IsFull using static array.

Text, letter

Description automatically generated

Text, letter

Description automatically generatedOUTPUT:-

Text

Description automatically generated

Q6. Write a menu driven program to implement circular queue operations such as Enqueue, Dequeue, Peek, Display of elements, IsEmpty using linked list.

#include <stdio.h>

#define max 6

int queue[max];

int front = -1;

int rear = -1;

void enqueue(int *element*)

{

    if (front == -1 && rear == -1)

    {

        front = 0;

        rear = 0;

        queue[rear] = *element*;

    }

    else if ((rear + 1) % max == front)

    {

        printf("Queue is overflow..");

    }

    else

    {

        rear = (rear + 1) % max;

        queue[rear] = *element*;

    }

}

int dequeue()

{

    if ((front == -1) && (rear == -1))

    {

        printf("\nQueue is underflow..");

    }

    else if (front == rear)

    {

        printf("\nThe dequeued element is %d", queue[front]);

        front = -1;

        rear = -1;

    }

    else

    {

        printf("\nThe dequeued element is %d", queue[front]);

        front = (front + 1) % max;

    }

}

void display()

{

    int i = front;

    if (front == -1 && rear == -1)

    {

        printf("\n Queue is empty..");

    }

    else

    {

        printf("\nElements in a Queue are : ");

        while (i <= rear)

        {

            printf("%d,", queue[i]);

            i = (i + 1) % max;

        }

    }

}

int main()

{

    int choice = 1, x;

    while (choice < 4 && choice != 0)

    {

        printf("\n Press 1: Insert an element ");

        printf("\nPress 2: Delete an element ");

        printf("\nPress 3: Display the element ");

        printf("\nEnter your choice ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            printf("Enter the element which is to be inserted ");

            scanf("%d", &*x*);

            enqueue(x);

            break;

        case 2:

            dequeue();

            break;

        case 3:

            display();

        }

    }

    return 0;

}

OUTPUT:-

Text

Description automatically generated