**1**.**Queue implementation in C**

#include <stdio.h>

#define SIZE 5

void enQueue(int);

void deQueue();

void display();

int items[SIZE], front = -1, rear = -1;

int main() {

//deQueue is not possible on empty queue

deQueue();

//enQueue 5 elements

enQueue(1);

enQueue(2);

enQueue(3);

enQueue(4);

enQueue(5);

// 6th element can't be added to because the queue is full

enQueue(6);

display();

//deQueue removes element entered first i.e. 1

deQueue();

//Now we have just 4 elements

display();

return 0;

}

void enQueue(int value) {

if (rear == SIZE - 1)

printf("\nQueue is Full!!");

else {

if (front == -1)

front = 0;

rear++;

items[rear] = value;

printf("\nInserted -> %d", value);

}

}

void deQueue() {

if (front == -1)

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

else {

printf("\nDeleted : %d", items[front]);

front++;

if (front > rear)

front = rear = -1;

}

}

// Function to print the queue

void display() {

if (rear == -1)

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

else {

int i;

printf("\nQueue elements are:\n");

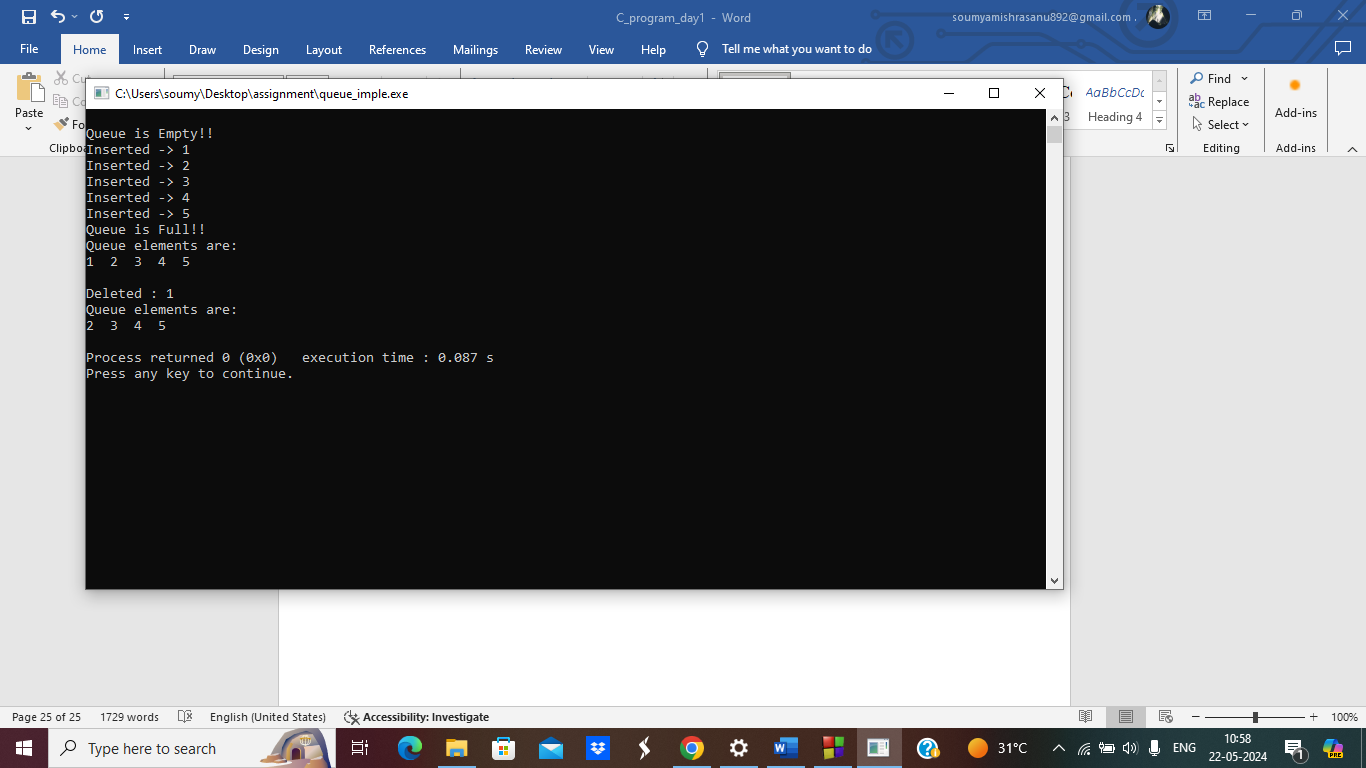
for (i = front; i <= rear; i++)

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

}

printf("\n");

}



**2.Stack**

// Stack implementation in C

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

int count = 0;

// Creating a stack

struct stack {

int items[MAX];

int top;

};

typedef struct stack st;

void createEmptyStack(st \*s) {

s->top = -1;

}

// Check if the stack is full

int isfull(st \*s) {

if (s->top == MAX - 1)

return 1;

else

return 0;

}

// Check if the stack is empty

int isempty(st \*s) {

if (s->top == -1)

return 1;

else

return 0;

}

// Add elements into stack

void push(st \*s, int newitem) {

if (isfull(s)) {

printf("STACK FULL");

} else {

s->top++;

s->items[s->top] = newitem;

}

count++;

}

// Remove element from stack

void pop(st \*s) {

if (isempty(s)) {

printf("\n STACK EMPTY \n");

} else {

printf("Item popped= %d", s->items[s->top]);

s->top--;

}

count--;

printf("\n");

}

// Print elements of stack

void printStack(st \*s) {

printf("Stack: ");

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

printf("%d ", s->items[i]);

}

printf("\n");

}

// Driver code

int main() {

int ch;

st \*s = (st \*)malloc(sizeof(st));

createEmptyStack(s);

push(s, 1);

push(s, 2);

push(s, 3);

push(s, 4);

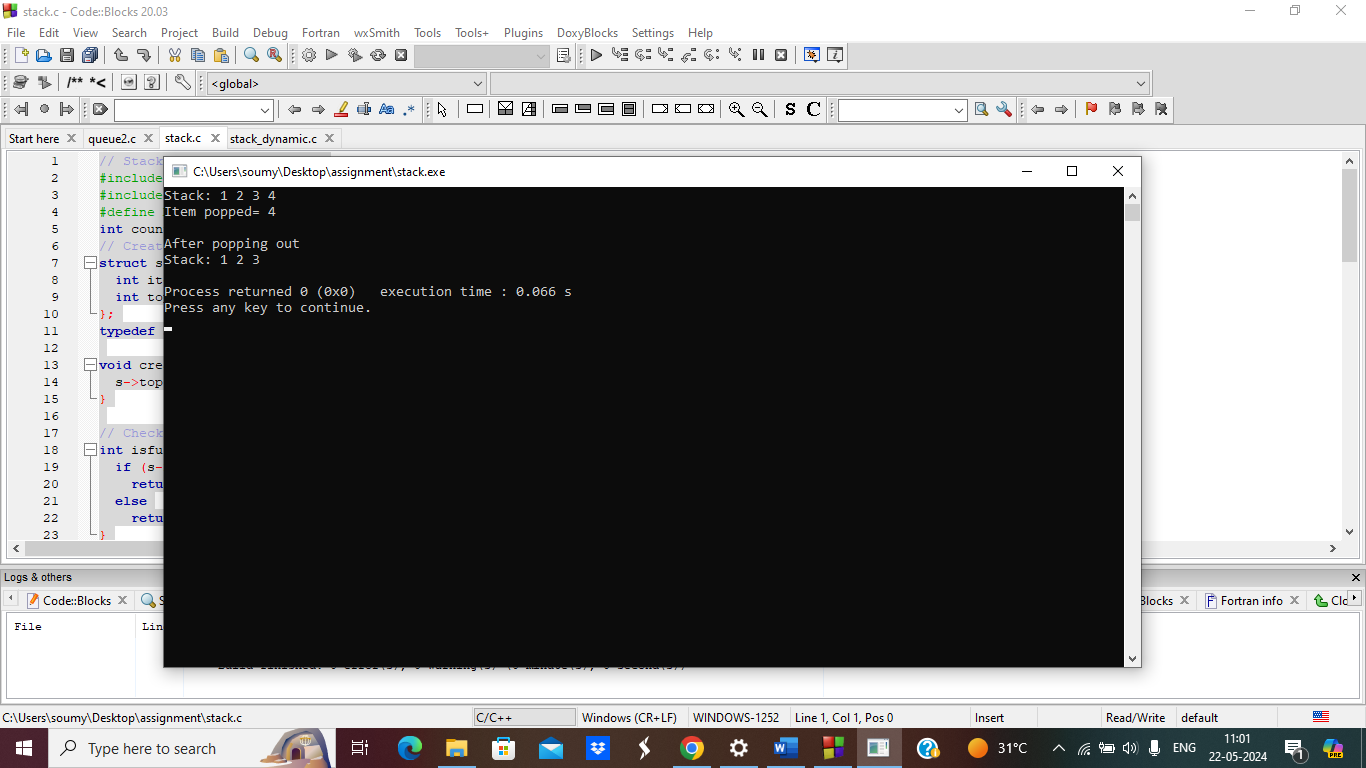
printStack(s);

pop(s);

printf("\nAfter popping out\n");

printStack(s);

}



**3. stack dynamic input**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

struct Stack {

int \*array;

int top;

int capacity;

};

struct Stack\* createStack(int capacity) {

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

if (!stack)

return NULL;

stack->capacity = capacity;

stack->top = -1;

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

if (!stack->array)

return NULL;

return stack;

}

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)) {

printf("Stack Overflow\n");

return;

}

stack->array[++stack->top] = item;

printf("%d pushed to stack\n", item);

}

int pop(struct Stack \*stack) {

if (isEmpty(stack)) {

printf("Stack Underflow\n");

return -1;

}

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

}

int peek(struct Stack \*stack) {

if (isEmpty(stack))

return -1;

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

}

int main() {

int capacity;

printf("Enter the capacity of the stack: ");

scanf("%d", &capacity);

struct Stack \*stack = createStack(capacity);

if (!stack) {

printf("Memory allocation failed.\n");

return -1;

}

int choice, element;

while (1) {

printf("\nStack Operations:\n");

printf("1. Push\n");

printf("2. Pop\n");

printf("3. Peek\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter element to push: ");

scanf("%d", &element);

push(stack, element);

break;

case 2:

printf("Popped element: %d\n", pop(stack));

break;

case 3:

printf("Top element: %d\n", peek(stack));

break;

case 4:

free(stack->array);

free(stack);

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice\n");

}

}

}

