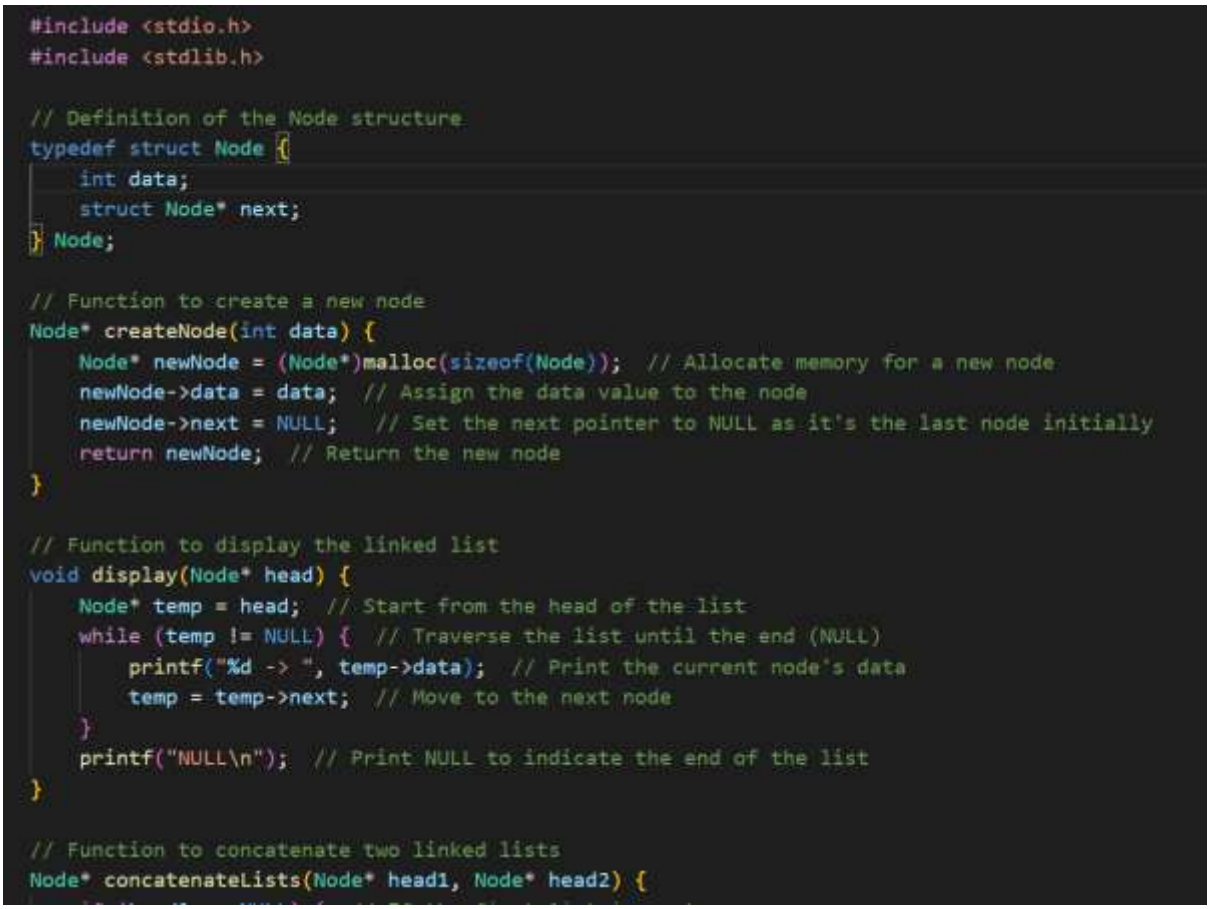
DAY 22-DAIILY ASSIGNMENTS

ANSU MARIUM SHIBU

05-12-2024

1. create two linked list in one linked {1,2,3,4}and in the 2nd linked list will have value{7,8,9}.COncatenate both the linked list and display the concatenated linked list.



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

// Definition of the Node structure
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));  // Allocate memory for a new node
    newNode->data = data;  // Assign the data value to the node
    newNode->next = NULL;   // Set the next pointer to NULL as it's the last node initially
    return newNode;  // Return the new node
}

// Function to display the linked list
void display(Node* head) {
    Node* temp = head;  // Start from the head of the list
    while (temp != NULL) {  // Traverse the list until the end (NULL)
        printf("%d -> ", temp->data);  // Print the current node's data
        temp = temp->next;  // Move to the next node
    }
    printf("NULL\n");  // Print NULL to indicate the end of the list
}

// Function to concatenate two linked lists
Node* concatenateLists(Node* head1, Node* head2) {
```

```c
// Function to concatenate two linked lists
Node* concatenateLists(Node* head1, Node* head2) {
    if (head1 == NULL) {  // If the first list is empty
        return head2;  // Return the second list
    }

    Node* temp = head1;  // Start from the first list
    while (temp->next != NULL) {  // Traverse to the end of the first list
        temp = temp->next;  // Move to the next node
    }

    temp->next = head2;  // Link the last node of the first list to the head of the second list
    return head1;  // Return the head of the combined list
}

int main() {
    // Creating the first list: 1 -> 2 -> 3 -> 4
    Node* head1 = createNode(1);  // Create the first node with data 1
    head1->next = createNode(2);  // Link the second node with data 2
    head1->next->next = createNode(3);  // Link the third node with data 3
    head1->next->next->next = createNode(4);  // Link the fourth node with data 4

    // Creating the second list: 7 -> 8 -> 9
    Node* head2 = createNode(7);  // Create the first node with data 7
    head2->next = createNode(8);  // Link the second node with data 8
    head2->next->next = createNode(9);  // Link the third node with data 9

    // Display the first list
    printf("First List: ");
    head1->next->next = createNode(3);  // Link the third node with data 3
    head1->next->next->next = createNode(4);  // Link the fourth node with data 4

    // Creating the second list: 7 -> 8 -> 9
    Node* head2 = createNode(7);  // Create the first node with data 7
    head2->next = createNode(8);  // Link the second node with data 8
    head2->next->next = createNode(9);  // Link the third node with data 9

    // Display the first list
    printf("First List: ");
    display(head1);

    // Display the second list
    printf("Second List: ");
    display(head2);

    // Concatenate the lists
    Node* mergedHead = concatenateLists(head1, head2);

    // Display the concatenated list
    printf("Concatenated List: ");
    display(mergedHead);

    return 0;
}
```

```
PS D:\c progrms coding> gcc linkconcatass1.c
PS D:\c progrms coding> ./a
First List: 1 -> 2 -> 3 -> 4 -> NULL
Second List: 7 -> 8 -> 9 -> NULL
Concatenated List: 1 -> 2 -> 3 -> 4 -> 7 -> 8 -> 9 -> NULL
PS D:\c progrms coding>
```

2. Problem Statement: Automotive Manufacturing Plant Management System
Objective:
Develop a program to manage an automotive manufacturing plant's operations using a linked list in C programming. The system will allow creation, insertion, deletion, and searching operations for managing assembly lines and their details.

Requirements
Data Representation
Node Structure:
Each node in the linked list represents an assembly line.
Fields:
lineID (integer): Unique identifier for the assembly line.
lineName (string): Name of the assembly line (e.g., "Chassis Assembly").
capacity (integer): Maximum production capacity of the line per shift.
status (string): Current status of the line (e.g., "Active", "Under Maintenance").
next (pointer to the next node): Link to the next assembly line in the list.
Linked List:
The linked list will store a dynamic number of assembly lines, allowing for additions and removals as needed.

Features to Implement
Creation:
Initialize the linked list with a specified number of assembly lines.
Insertion:
Add a new assembly line to the list either at the beginning, end, or at a specific position.
Deletion:
Remove an assembly line from the list by its lineID or position.
Searching:
Search for an assembly line by lineID or lineName and display its details.
Display:
Display all assembly lines in the list along with their details.
Update Status:
Update the status of an assembly line (e.g., from "Active" to "Under Maintenance").

Example Program Flow
Menu Options:
Provide a menu-driven interface with the following operations:
Create Linked List of Assembly Lines
Insert New Assembly Line
Delete Assembly Line

Search for Assembly Line
Update Assembly Line Status
Display All Assembly Lines
Exit
Sample Input/Output:
Input:
Number of lines: 3
Line 1: ID = 101, Name = "Chassis Assembly", Capacity = 50, Status = "Active".
Line 2: ID = 102, Name = "Engine Assembly", Capacity = 40, Status = "Under Maintenance".
Output:
Assembly Lines:
Line 101: Chassis Assembly, Capacity: 50, Status: Active
Line 102: Engine Assembly, Capacity: 40, Status: Under Maintenance

Linked List Node Structure in C

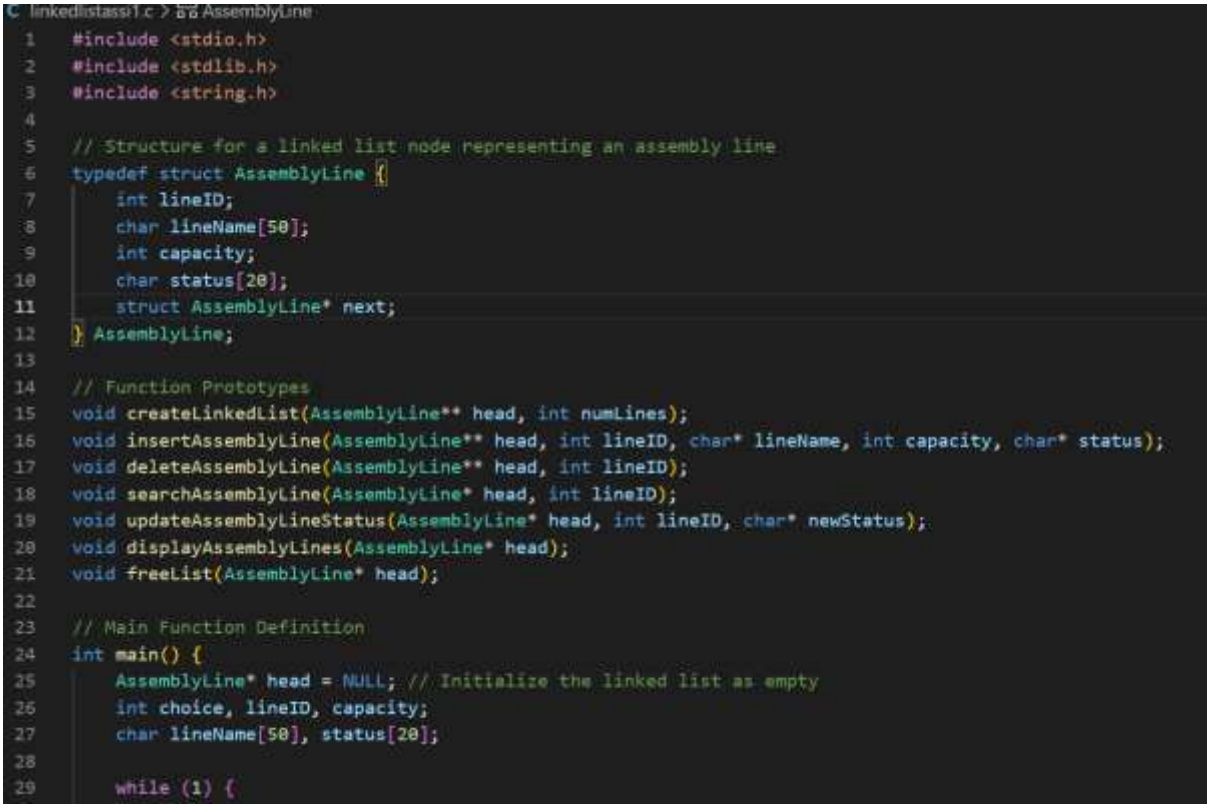
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Structure for a linked list node
typedef struct AssemblyLine {
  int lineID;                 // Unique line ID
  char lineName[50];          // Name of the assembly line
  int capacity;               // Production capacity per shift
  char status[20];            // Current status of the line
  struct AssemblyLine* next;  // Pointer to the next node
} AssemblyLine;
```

Operations Implementation
1. Create Linked List
Allocate memory dynamically for AssemblyLine nodes.
Initialize each node with details such as lineID, lineName, capacity, and status.
2. Insert New Assembly Line
Dynamically allocate a new node and insert it at the desired position in the list.
3. Delete Assembly Line
Locate the node to delete by lineID or position and adjust the next pointers of adjacent nodes.
4. Search for Assembly Line
Traverse the list to find a node by its lineID or lineName and display its details.
5. Update Assembly Line Status
Locate the node by lineID and update its status field.
6. Display All Assembly Lines
Traverse the list and print the details of each node.

Sample Menu
Menu:
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line

3. Delete Assembly Line
4. Search for Assembly Line
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit



```c
// Main Function Definition
int main() {
    AssemblyLine* head = NULL; // Initialize the linked list as empty
    int choice, lineID, capacity;
    char lineName[50], status[20];

    while (1) {
        // Display menu
        printf("\nMenu:\n");
        printf("1. Create Linked List of Assembly Lines\n");
        printf("2. Insert New Assembly Line\n");
        printf("3. Delete Assembly Line\n");
        printf("4. Search for Assembly Line by Line ID\n");
        printf("5. Update Assembly Line Status\n");
        printf("6. Display All Assembly Lines\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the number of assembly lines: ");
                int numLines;
                scanf("%d", &numLines);
                createLinkedList(&head, numLines);
                break;
            case 2:
                printf("Enter Line ID: ");
                scanf("%d", &lineID);
                printf("Enter Line Name: ");
                scanf(" %49[^\n]", lineName);  // Reads input with spaces
                printf("Enter Capacity: ");
                scanf("%d", &capacity);
                printf("Enter Status: ");
                scanf(" %19[^\n]", status);  // Reads input with spaces
                insertAssemblyLine(&head, lineID, lineName, capacity, status);
                break;
            case 3:
                printf("Enter Line ID to delete: ");
                scanf("%d", &lineID);
                deleteAssemblyLine(&head, lineID);
                break;
            case 4:
                printf("Enter Line ID to search: ");
                scanf("%d", &lineID);
                searchAssemblyLine(head, lineID);
                break;
            case 5:
                printf("Enter Line ID to update status: ");
                scanf("%d", &lineID);
                printf("Enter new status: ");
                scanf(" %19[^\n]", status);  // Reads input with spaces
                updateAssemblyLineStatus(head, lineID, status);
                break;
```

```
        switch (choice) {
                updateAssemblyLineStatus(head, lineID, status);
                break;
            case 6:
                displayAssemblyLines(head);
                break;
            case 7:
                freeList(head);  // Free dynamically allocated memory
                printf("Exiting program...\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
}

// Function Definitions

// Function to create linked list with a specified number of lines
void createLinkedList(AssemblyLine** head, int numLines) {
    int lineID, capacity;
    char lineName[50], status[20];

    for (int i = 0; i < numLines; i++) {
        printf("Enter details for line %d\n", i + 1);
        printf("Line ID: ");
        scanf("%d", &lineID);
```

```
    for (int i = 0; i < numLines; i++) {
        scanf(" %19[^\n]", status);  // Reads input with spaces
        insertAssemblyLine(head, lineID, lineName, capacity, status);
    }
}

// Function to insert a new assembly line at the end of the list
void insertAssemblyLine(AssemblyLine** head, int lineID, char* lineName, int capacity, char* status) {
    AssemblyLine* newLine = (AssemblyLine*)malloc(sizeof(AssemblyLine));  // Create a new node
    newLine->lineID = lineID;
    strcpy(newLine->lineName, lineName);
    newLine->capacity = capacity;
    strcpy(newLine->status, status);
    newLine->next = NULL;

    if (*head == NULL) {
        *head = newLine;  // If the list is empty, make the new node the head
    } else {
        AssemblyLine* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;  // Traverse to the last node
        }
        temp->next = newLine;  // Link the new node at the end
    }
}

// Function to delete an assembly line by line ID
void deleteAssemblyLine(AssemblyLine** head, int lineID) {
```

```c
// Function to delete an assembly line by line ID
void deleteAssemblyLine(AssemblyLine** head, int lineID) {
    AssemblyLine *temp = *head, *prev = NULL;

    if (temp != NULL && temp->lineID == lineID) {
        *head = temp->next;  // Move head to the next node
        free(temp);  // Free the node
        return;
    }

    while (temp != NULL && temp->lineID != lineID) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Line ID not found.\n");
        return;
    }

    prev->next = temp->next;
    free(temp);  // Free the node
}

// Function to search an assembly line by line ID
void searchAssemblyLine(AssemblyLine* head, int lineID) {
    AssemblyLine* temp = head;
```

```c
// Function to search an assembly line by line ID
void searchAssemblyLine(AssemblyLine* head, int lineID) {
    AssemblyLine* temp = head;

    while (temp != NULL) {
        if (temp->lineID == lineID) {
            printf("Assembly Line found:\n");
            printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n",
                   temp->lineID, temp->lineName, temp->capacity, temp->status);
            return;
        }
        temp = temp->next;
    }
    printf("Assembly Line with ID %d not found.\n", lineID);
}

// Function to update the status of an assembly line
void updateAssemblyLineStatus(AssemblyLine* head, int lineID, char* newStatus) {
    AssemblyLine* temp = head;

    while (temp != NULL) {
        if (temp->lineID == lineID) {
            strcpy(temp->status, newStatus);  // Update the status
            printf("Status updated successfully.\n");
            return;
        }
        temp = temp->next;
    }
```

```c
void updateAssemblyLineStatus(AssemblyLine* head, int lineID, char* newStatus) {
    while (temp != NULL) {
    }
    printf("Assembly Line with ID %d not found.\n", lineID);
}

// Function to display all assembly lines
void displayAssemblyLines(AssemblyLine* head) {
    if (head == NULL) {
        printf("No assembly lines to display.\n");
        return;
    }
    AssemblyLine* temp = head;
    while (temp != NULL) {
        printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n",
               temp->lineID, temp->lineName, temp->capacity, temp->status);
        temp = temp->next;
    }
}

// Function to free the dynamically allocated memory of the linked list
void freeList(AssemblyLine* head) {
    AssemblyLine* temp = head;
    while (temp != NULL) {
        AssemblyLine* next = temp->next;
        free(temp);
        temp = next;
    }
}
```

```
PS D:\c progrms coding> ./a

Menu:
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line
3. Delete Assembly Line
4. Search for Assembly Line by Line ID
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit
Enter your choice: 1
Enter the number of assembly lines: 2
Enter details for line 1
Line ID: 1
Line Name: assim
Capacity: 100
Status: active
Enter details for line 2
Line ID: 23
Line Name: engine assim
Capacity: 500
Status: under maintenance

Menu:
```

```
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line
3. Delete Assembly Line
4. Search for Assembly Line by Line ID
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit
Enter your choice: 2
Enter Line ID: 45
Enter Line Name: skelton
Enter Capacity: 45
Enter Status: active

Menu:
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line
3. Delete Assembly Line
4. Search for Assembly Line by Line ID
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit
Enter your choice:
```

```
2. Insert New Assembly Line
3. Delete Assembly Line
4. Search for Assembly Line by Line ID
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit
Enter your choice: 3
Enter Line ID to delete: 45

Menu:
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line
3. Delete Assembly Line
4. Search for Assembly Line by Line ID
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit
Enter your choice: 6
Line ID: 1, Name: assim, Capacity: 100, Status: active
Line ID: 23, Name: engine assim, Capacity: 500, Status: under maintenance

Menu:
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line
```

```
Line ID: 23, Name: engine assim, Capacity: 500, Status: under maintenance

Menu:
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line
3. Delete Assembly Line
4. Search for Assembly Line by Line ID
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit
Enter your choice: 4
Enter Line ID to search: 1
Assembly Line found:
Line ID: 1, Name: assim, Capacity: 100, Status: active

Menu:
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line
3. Delete Assembly Line
4. Search for Assembly Line by Line ID
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit
Enter your choice:
```

```
 Menu:
 1. Create Linked List of Assembly Lines
 2. Insert New Assembly Line
 3. Delete Assembly Line
 4. Search for Assembly Line by Line ID
 5. Update Assembly Line Status
 6. Display All Assembly Lines
 7. Exit
 Enter your choice: 5
 Enter Line ID to update status: 23
 Enter new status: active
 Status updated successfully.

 Menu:
 1. Create Linked List of Assembly Lines
 2. Insert New Assembly Line
 3. Delete Assembly Line
 4. Search for Assembly Line by Line ID
 5. Update Assembly Line Status
 6. Display All Assembly Lines
 7. Exit
 Enter your choice: 6
```

```
Enter new status: active
Status updated successfully.

Menu:
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line
3. Delete Assembly Line
4. Search for Assembly Line by Line ID
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit
Enter your choice: 6
Line ID: 1, Name: assim, Capacity: 100, Status: active
Line ID: 23, Name: engine assim, Capacity: 500, Status: active

Menu:
1. Create Linked List of Assembly Lines
2. Insert New Assembly Line
3. Delete Assembly Line
4. Search for Assembly Line by Line ID
5. Update Assembly Line Status
6. Display All Assembly Lines
7. Exit
Enter your choice:
```

3.stack operations

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

// Structure for Stack
struct Stack {
    int size;   // Maximum size of the stack
    int top;    // Index of the top element
    int *S;     // Pointer to array representing stack
};

// Function Prototypes
void create(struct Stack *st);
void push(struct Stack *st, int x);
void display(struct Stack *st);
int pop(struct Stack *st);
int peek(struct Stack *st, int position);
int isEmpty(struct Stack *st);
int stackTop(struct Stack *st);
int isFull(struct Stack *st);

int main() {
    struct Stack st;
    int elementPopped, peekedElement, position;

    // Create stack
```

```c
    // Perform stack operations
    push(&st, 10);
    push(&st, 20);
    push(&st, 30);
    push(&st, 40);

    // Display stack
    display(&st);

    // Check if stack is full
    if (isFull(&st)) {
        printf("Stack is full\n");
    } else {
        printf("Stack is not full\n");
    }

    // Pop an element
    elementPopped = pop(&st);
    printf("The popped element is: %d\n", elementPopped);

    // Display stack again
    display(&st);

    // Check if stack is empty
    if (isEmpty(&st)) {
```

```c
    display(&st);

    // Check if stack is empty
    if (isEmpty(&st)) {
        printf("Stack is empty\n");
    } else {
        printf("Stack is not empty\n");
    }

    // Get the top element of the stack
    int topElement = stackTop(&st);
    printf("Top element of the stack is: %d\n", topElement);

    // Peek at a specific position
    printf("Enter position to peek (1 for top element): ");
    scanf("%d", &position);
    peekedElement = peek(&st, position);
    if (peekedElement != -1) {
        printf("Element at position %d from the top is: %d\n", position, peekedElement);
    }

    return 0;
}

// Function to create the stack
```

```c
}

// Function to create the stack
void create(struct Stack *st) {
    printf("Enter size of the stack: ");
    scanf("%d", &st->size);
    st->top = -1;  // Initialize top index
    st->S = (int *)malloc(st->size * sizeof(int));  // Allocate memory for the stack array
}

// Function to push an element onto the stack
void push(struct Stack *st, int x) {
    if (st->top == st->size - 1) {
        printf("Stack Overflow\n");
    } else {
        st->top++;
        st->S[st->top] = x;
    }
}

// Function to display the stack elements
void display(struct Stack *st) {
    if (st->top == -1) {
        printf("Stack is empty\n");
```

C stackfull.c > ...

```c
82  void push(struct Stack *st, int x) {
89  }
90
91  // Function to display the stack elements
92  void display(struct Stack *st) {
93      if (st->top == -1) {
94          printf("Stack is empty\n");
95      } else {
96          printf("Stack elements:\n");
97          for (int i = st->top; i >= 0; i--) {
98              printf("%d\n", st->S[i]);
99          }
100     }
101     printf("\n");
102 }
103
104 // Function to pop an element from the stack
105 int pop(struct Stack *st) {
106     int x = -1;
107     if (st->top == -1) {
108         printf("Stack Underflow\n");
109     } else {
110         x = st->S[st->top];
111         st->top--;
112     }
```

```
int pop(struct Stack *st) {
    } else {
        return x;
    }

    // Function to peek at a specific position in the stack
    int peek(struct Stack *st, int position) {
        int index = st->top - position + 1;  // Calculate the array index of
        if (index < 0 || index > st->top) {
            printf("Invalid position\n");
            return -1;
        }
        return st->S[index];
    }

    // Function to check if the stack is empty
    int isEmpty(struct Stack *st) {
        return st->top == -1; // Returns 1 if empty, 0 otherwise
    }

    // Function to get the top element of the stack
    int stackTop(struct Stack *st) {
        if (isEmpty(st)) {
            printf("Stack is empty. No top element.\n");
            return -1;
        }
```

```
// Function to get the top element of the stack
int stackTop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No top element.\n");
        return -1;
    }
    return st->S[st->top];
}

// Function to check if the stack is full
int isFull(struct Stack *st) {
    return st->top == st->size - 1; // Returns 1 if full, 0 otherwise
}
```

```
Enter position to peek (1 for top element): 2
Element at position 2 from the top is: 20
Enter position to peek (1 for top element): 2
Element at position 2 from the top is: 20
Element at position 2 from the top is: 20
PS D:\c progrms coding> gcc stackfull.c
PS D:\c progrms coding> ./a
Enter size of the stack: 4
Stack elements:
40
30
20
10

Stack is full
The popped element is: 40
Stack elements:
30
20
10

Stack is not empty
Top element of the stack is: 30
Enter position to peek (1 for top element): 2
Element at position 2 from the top is: 20
PS D:\c progrms coding>
```