Assignment -21

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

struct Stack{
    int size;
    int top;
    int *s;
};

void create(struct Stack *);
void display(struct Stack );
void push(struct Stack *, int);
int pop(struct Stack *);
int isEmpty(struct Stack);
int isFull(struct Stack);
int stackTop(struct Stack);

int main(){
    struct Stack st;
    create(&st);
    push(&st, 5);
    push(&st, 6);
    push(&st, 7);
    push(&st, 8);
    display(st);
    int popedValue = pop(&st);
    printf("popedValue = %d \n",popedValue);
    display(st);
    return 0;
}
void create(struct Stack *st){
    printf("Enter The Size");
    scanf("%d",&st->size);
    st->top = -1;
    st->s = (int *)malloc((st->size) * sizeof(int));
}


void push(struct Stack *st, int x){
    if(st->top == st->size-1){
        printf("Stack is FUll \n");
    }else{
        st->top++;
        st->s[st->top] = x;
    }
}
void display(struct Stack st){
    int i;
    for(i= st.top;i>=0;i--){
        printf("%d ",st.s[i]);
        printf("\n");
    }
}

int pop(struct Stack *st){
    int x = -1;
    if(st->top == -1){
        printf("Stack is Empty\n");
    }
    else{
        x = st->s[st->top];
        st->top--;
    }
    return x;
}

int isEmpty(struct Stack st){
    if(st.top == -1){
        return 1;
    }
    return 0;
}

int isFull(struct Stack st){
    if(st.top == st.size-1){
        return 1;
    }
    return 0;
}

int stackTop(struct Stack st){
    if(!isEmpty){
        return st.s[st.top];
    }
    return -1;
}

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

struct Stack{
    int size;
    int top;
    int *s;
};

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, int);
int pop(struct Stack *);
int isEmpty(struct Stack);
int isFull(struct Stack);
int stackTop(struct Stack);
int peek(struct Stack, int);

int main(){
    struct Stack st;
    create(&st);
    push(&st, 5);
    push(&st, 6);
    push(&st, 7);
    push(&st, 8);
    display(st);

    int popedValue = pop(&st);
    printf("popedValue = %d \n", popedValue);
    display(st);

    int position = 2;  // Example position to peek
    int peekedValue = peek(st, position);
    if (peekedValue != -1) {
        printf("Element at position %d from top is %d\n", position, peekedValue);
    }

    return 0;
}

void create(struct Stack *st){
    printf("Enter The Size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->s = (int *)malloc((st->size) * sizeof(int));
}

void push(struct Stack *st, int x){
    if(st->top == st->size-1){
        printf("Stack is Full\n");
    } else {
        st->top++;
        st->s[st->top] = x;
    }
}

void display(struct Stack st){
    int i;
    for(i = st.top; i >= 0; i--){
        printf("%d\n", st.s[i]);
    }
}

int pop(struct Stack *st){
    int x = -1;
    if(st->top == -1){
        printf("Stack is Empty\n");
    } else {
        x = st->s[st->top];
        st->top--;
    }
    return x;
}

int isEmpty(struct Stack st){
    return st.top == -1;
}

int isFull(struct Stack st){
    return st.top == st.size-1;
```

```
}


int stackTop(struct Stack st){
    if(!isEmpty(st)){
        return st.s[st.top];
    }
    return -1;
}


int peek(struct Stack st, int position){
    if(position <= 0 || position > st.top + 1){
        printf("Invalid position\n");
        return -1;
    } else {
        return st.s[st.top - position + 1];
    }
}
```

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



struct Node{
    int data;
    struct Node *next;
}*top = NULL;



void push(int);
int pop();
void display();
```

```
int main(){
    push(20);
    push(30);
    push(40);
    display();
    int poopedValue=pop();
    printf("%d \n",poopedValue);
    printf("\n");
    display();
    return 0;
}



void push(int x){
    struct Node *t;
    t = (struct Node*)malloc(sizeof(struct Node));

    if(t == NULL){
        printf("Stack is Full \n");
    }
    else{
        t->data = x;
        t->next = top;
        top = t;
    }
}
void display(){
    struct Node *p;
    p = top;
    while(p != NULL){
        printf("%d ",p->data);
        printf("\n");
        p = p->next;
    }
    printf("\n");
}



int pop(){
    struct Node *t;
    int x = -1;

    if (top == NULL){
        printf("Stack is Empty");
    }
    else{
        t = top;
        top = top->next;
        x = t->data;
        free(t);
    }
    return x;
}
```

1. **Flight Path Logging System**: Implement a stack-based system using arrays to record the sequence of flight paths an aircraft takes. Use a switch-case menu with options:
   - 1: Add a new path (push)
   - 2: Undo the last path (pop)
   - 3: Display the current flight path stack
   - 4: Peek at the top path
   - 5: Search for a specific path
   - 6: Exit

Satellite Deployment Sequence: Develop a stack using arrays to manage the sequence of satellite deployments from a spacecraft. Include a switch-case menu with options:

- 1: Push a new satellite deployment
- 2: Pop the last deployment
- 3: View the deployment sequence
- 4: Peek at the latest deployment
- 5: Search for a specific deployment
- 6: Exit

Rocket Launch Checklist: Create a stack for a rocket launch checklist using arrays. Implement a switch-case menu with options:

- 1: Add a checklist item (push)
- 2: Remove the last item (pop)
- 3: Display the current checklist
- 4: Peek at the top checklist item
- 5: Search for a specific checklist item
- 6: Exit

Telemetry Data Storage: Implement a stack to store telemetry data from an aerospace vehicle. Use a switch-case menu with options:

- 1: Push new telemetry data
- 2: Pop the last data entry

- 3: View the stored telemetry data
- 4: Peek at the most recent data entry
- 5: Search for specific telemetry data
- 6: Exit

Space Mission Task Manager: Design a stack-based task manager for space missions using arrays. Include a switch-case menu with options:

- 1: Add a task (push)
- 2: Mark the last task as completed (pop)
- 3: List all pending tasks
- 4: Peek at the most recent task
- 5: Search for a specific task
- 6: Exit

Launch Countdown Management: Use a stack to manage the countdown sequence for a rocket launch. Implement a switch-case menu with options:

- 1: Add a countdown step (push)
- 2: Remove the last step (pop)
- 3: Display the current countdown
- 4: Peek at the next countdown step
- 5: Search for a specific countdown step
- 6: Exit

Aircraft Maintenance Logs: Implement a stack to keep track of maintenance logs for an aircraft. Use a switch-case menu with options:

- 1: Add a new log (push)
- 2: Remove the last log (pop)
- 3: View all maintenance logs
- 4: Peek at the latest maintenance log
- 5: Search for a specific maintenance log
- 6: Exit
-

Spacecraft Docking Procedure: Develop a stack for the sequence of steps in a spacecraft docking procedure. Implement a switch-case menu with options:

- 1: Push a new step
- 2: Pop the last step
- 3: Display the procedure steps
- 4: Peek at the next step in the procedure
- 5: Search for a specific step
- 6: Exit

Mission Control Command History: Create a stack to record the command history sent from mission control. Use a switch-case menu with options:

- 1: Add a command (push)
- 2: Undo the last command (pop)
- 3: View the command history
- 4: Peek at the most recent command
- 5: Search for a specific command
- 6: Exit

Aerospace Simulation Events: Implement a stack to handle events in an aerospace simulation. Include a switch-case menu with options:

- 1: Push a new event
- 2: Pop the last event
- 3: Display all events
- 4: Peek at the most recent event
- 5: Search for a specific event
- 6: Exit

2. **Pilot Training Maneuver Stack**: Use a stack to keep track of training maneuvers for pilots. Implement a switch-case menu with options:
    - 1: Add a maneuver (push)
    - 2: Remove the last maneuver (pop)
    - 3: View all maneuvers
    - 4: Peek at the most recent maneuver
    - 5: Search for a specific maneuver
    - 6: Exit
3. **Satellite Operation Commands**: Design a stack to manage operation commands for a satellite. Use a switch-case menu with options:
    - 1: Push a new command
    - 2: Pop the last command
    - 3: View the operation commands
    - 4: Peek at the most recent command
    - 5: Search for a specific command
    - 6: Exit
4. **Emergency Procedures for Spacecraft**: Create a stack-based system for handling emergency procedures in a spacecraft. Implement a switch-case menu with options:
    - 1: Add a procedure (push)
    - 2: Remove the last procedure (pop)
    - 3: View all procedures
    - 4: Peek at the next procedure
    - 5: Search for a specific procedure
    - 6: Exit
5. **Astronaut Activity Log**: Implement a stack for logging astronaut activities during a mission. Use a switch-case menu with options:
    - 1: Add a new activity (push)

- 2: Remove the last activity (pop)
- 3: Display the activity log
- 4: Peek at the most recent activity
- 5: Search for a specific activity
- 6: Exit

6. **Fuel Management System**: Develop a stack to monitor fuel usage in an aerospace vehicle. Implement a switch-case menu with options:
   - 1: Add a fuel usage entry (push)
   - 2: Remove the last entry (pop)
   - 3: View all fuel usage data
   - 4: Peek at the latest fuel usage entry
   - 5: Search for a specific fuel usage entry
   - 6: Exit
7. **Order Processing System**: Implement a stack-based system using a linked list to manage order processing. Use a switch-case menu with options:
   - 1: Add a new order (push)
   - 2: Process the last order (pop)
   - 3: Display all pending orders
   - 4: Peek at the next order to be processed
   - 5: Search for a specific order
   - 6: Exit
8. **Customer Support Ticketing**: Create a stack using a linked list to handle customer support tickets. Include a switch-case menu with options:
   - 1: Add a new ticket (push)
   - 2: Resolve the latest ticket (pop)
   - 3: View all pending tickets
   - 4: Peek at the latest ticket
   - 5: Search for a specific ticket
   - 6: Exit
9. **Product Return Management**: Develop a stack to manage product returns using a linked list. Implement a switch-case menu with options:
   - 1: Add a new return request (push)
   - 2: Process the last return (pop)
   - 3: Display all return requests
   - 4: Peek at the next return to process
   - 5: Search for a specific return request
   - 6: Exit
10. **Inventory Restock System**: Implement a stack to manage inventory restocking using a linked list. Use a switch-case menu with options:
    - 1: Add a restock entry (push)
    - 2: Process the last restock (pop)
    - 3: View all restock entries
    - 4: Peek at the latest restock entry
    - 5: Search for a specific restock entry
    - 6: Exit

11. **Flash Sale Deal Management**: Create a stack for managing flash sale deals using a linked list. Include a switch-case menu with options:
    - 1: Add a new deal (push)
    - 2: Remove the last deal (pop)
    - 3: View all active deals
    - 4: Peek at the latest deal
    - 5: Search for a specific deal
    - 6: Exit
12. **User Session History**: Use a stack to track user session history in an e-commerce site using a linked list. Implement a switch-case menu with options:
    - 1: Add a session (push)
    - 2: End the last session (pop)
    - 3: Display all sessions
    - 4: Peek at the most recent session
    - 5: Search for a specific session
    - 6: Exit
13. **Wishlist Management**: Develop a stack to manage user wishlists using a linked list. Use a switch-case menu with options:
    - 1: Add a product to wishlist (push)
    - 2: Remove the last added product (pop)
    - 3: View all wishlist items
    - 4: Peek at the most recent wishlist item
    - 5: Search for a specific product in wishlist
    - 6: Exit
14. **Checkout Process Steps**: Implement a stack to manage steps in the checkout process using a linked list. Include a switch-case menu with options:
    - 1: Add a checkout step (push)
    - 2: Remove the last step (pop)
    - 3: Display all checkout steps
    - 4: Peek at the current step
    - 5: Search for a specific step
    - 6: Exit
15. **Coupon Code Management**: Create a stack for managing coupon codes using a linked list. Use a switch-case menu with options:
    - 1: Add a new coupon code (push)
    - 2: Remove the last coupon code (pop)
    - 3: View all available coupon codes
    - 4: Peek at the latest coupon code
    - 5: Search for a specific coupon code
    - 6: Exit
16. **Shipping Status Tracker**: Develop a stack to track shipping status updates using a linked list. Implement a switch-case menu with options:
    - 1: Add a shipping status update (push)
    - 2: Remove the last update (pop)
    - 3: View all shipping status updates
    - 4: Peek at the latest update

   - 5: Search for a specific update
   - 6: Exit
17. **User Review Management**: Use a stack to manage user reviews for products using a linked list. Include a switch-case menu with options:
   - 1: Add a new review (push)
   - 2: Remove the last review (pop)
   - 3: Display all reviews
   - 4: Peek at the latest review
   - 5: Search for a specific review
   - 6: Exit
18. **Promotion Notification System**: Create a stack for managing promotional notifications using a linked list. Use a switch-case menu with options:
   - 1: Add a new notification (push)
   - 2: Remove the last notification (pop)
   - 3: View all notifications
   - 4: Peek at the latest notification
   - 5: Search for a specific notification
   - 6: Exit
19. **Product Viewing History**: Implement a stack to track the viewing history of products using a linked list. Include a switch-case menu with options:
   - 1: Add a product to viewing history (push)
   - 2: Remove the last viewed product (pop)
   - 3: Display all viewed products
   - 4: Peek at the most recent product viewed
   - 5: Search for a specific product
   - 6: Exit
20. **Cart Item Management**: Develop a stack to manage items in a shopping cart using a linked list. Use a switch-case menu with options:
   - 1: Add an item to the cart (push)
   - 2: Remove the last item (pop)
   - 3: View all cart items
   - 4: Peek at the last added item
   - 5: Search for a specific item in the cart
   - 6: Exit
21. **Payment History**: Implement a stack to record payment history using a linked list. Include a switch-case menu with options:
   - 1: Add a new payment record (push)
   - 2: Remove the last payment record (pop)
   - 3: View all payment records
   - 4: Peek at the latest payment record
   - 5: Search for a specific payment record
   - 6: Exit

Programs:

//1.

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

struct Stack {
    char **flightPath;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice, size;
```

```
char item[100];

printf("Enter the maximum size of the stack: ");
scanf("%d", &size);

create(&st, size);

while (1) {
    printf("\n--- Flight Path Logging System ---\n");
    printf("1: Add a new path\n");
    printf("2: Undo the last path\n");
    printf("3: Display the current flight path stack\n");
    printf("4: Peek at the top path\n");
    printf("5: Search for a specific path\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the flight path: ");
            getchar();
            scanf("%[^\n]", item);
            push(&st, item);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the path to search for: ");
            getchar();
            scanf("%[^\n]", item);
            search(&st, item);
            break;
        case 6:
            printf("Exiting...\n");
            free(st.flightPath);
            exit(0);
        default:
            printf("Invalid choice\n");
```

```
        }
    }

    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->flightPath = (char **)malloc(size * sizeof(char *));
    if (st->flightPath == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

```
void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->flightPath[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->flightPath[st->top], item);
    printf("Flight path '%s' added to the stack\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Flight path '%s' removed from the stack\n", st->flightPath[st->top]);
    free(st->flightPath[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {

        printf("Stack is empty\n");

        return;

    }

    printf("Current flight path stack:\n");

    for (int i = st->top; i >= 0; i--) {

        printf("%d: %s\n", i + 1, st->flightPath[i]);

    }

}


void peek(struct Stack *st) {

    if (isEmpty(st)) {

        printf("Stack is empty\n");

        return;

    }

    printf("Top flight path: %s\n", st->flightPath[st->top]);

}


void search(struct Stack *st, char item[]) {

    for (int i = st->top; i >= 0; i--) {

        if (strcmp(st->flightPath[i], item) == 0) {

            printf("Flight path '%s' found at position %d\n", item, i + 1);

            return;
        }
    }
    printf("Flight path '%s' not found in the stack\n", item);
}



//2.

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

struct Stack {
    char **deployment;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice, size;
    char item[100];

    printf("Enter the maximum size of the stack: ");
    scanf("%d", &size);

    create(&st, size);

    while (1) {
        printf("\n--- Satellite Deployment Sequence ---\n");
        printf("1: Push a new satellite deployment\n");
        printf("2: Pop the last deployment\n");
        printf("3: View the deployment sequence\n");
        printf("4: Peek at the latest deployment\n");
        printf("5: Search for a specific deployment\n");
        printf("6: Exit\n");
        printf("Enter your choice: ");
```

```
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter the satellite deployment: ");
        getchar();
        scanf("%[^\n]", item);
        push(&st, item);
        break;
    case 2:
        pop(&st);
        break;
    case 3:
        display(&st);
        break;
    case 4:
        peek(&st);
        break;
    case 5:
        printf("Enter the satellite deployment to search for: ");
        getchar();
        scanf("%[^\n]", item);
        search(&st, item);
```

```
                break;
            case 6:
                printf("Exiting...\n");
                free(st.deployment);
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->deployment = (char **)malloc(size * sizeof(char *));
    if (st->deployment == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->deployment[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->deployment[st->top], item);
    printf("Deployment '%s' added to stack\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Deployment '%s' removed from stack\n", st->deployment[st->top]);

    free(st->deployment[st->top]);

    st->top--;

}


void display(struct Stack *st) {

    if (isEmpty(st)) {

        printf("Stack is empty\n");

        return;

    }

    printf("Current deployment sequence:\n");

    for (int i = st->top; i >= 0; i--) {

        printf("%d: %s\n", i + 1, st->deployment[i]);

    }

}


void peek(struct Stack *st) {

    if (isEmpty(st)) {

        printf("Stack is empty\n");

        return;

    }

    printf("Latest deployment: %s\n", st->deployment[st->top]);

}
```

```
void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->deployment[i], item) == 0) {
            printf("Deployment '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Deployment '%s' not found in the stack\n", item);
}

//3.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Stack {
	char **checklist;
	int top;
	int size;
};

void create(struct Stack *st, int size);
```

```
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice, size;
    char item[100];

    printf("Enter the maximum size of the stack: ");
    scanf("%d", &size);

    create(&st, size);

    while (1) {
        printf("\n--- Rocket Launch Checklist ---\n");
        printf("1: Add a checklist item (push)\n");
        printf("2: Remove the last item (pop)\n");
        printf("3: View the current checklist\n");
        printf("4: Peek at the top checklist item\n");

        printf("5: Search for a specific checklist item\n");

        printf("6: Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);


        switch (choice) {

        case 1:

                printf("Enter the checklist item: ");

                getchar();

                scanf("%[^\n]", item);

                push(&st, item);

                break;

        case 2:

                pop(&st);

                break;

        case 3:

                display(&st);

                break;

        case 4:

                peek(&st);

                break;

        case 5:
```

```
                printf("Enter the checklist item to search for: ");
                getchar();
                scanf("%[^\n]", item);
                search(&st, item);
                break;
            case 6:
                printf("Exiting...\n");
                free(st.checklist);
                exit(0);
            default:
                printf("Invalid choice\n");
            }
        }
        return 0;
}

void create(struct Stack *st, int size) {
        st->size = size;
        st->top = -1;
        st->checklist = (char **)malloc(size * sizeof(char *));
        if (st->checklist == NULL) {
            printf("Memory allocation failed\n");
            exit(1);
        }
}

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

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

void push(struct Stack *st, char item[]) {
        if (isFull(st)) {
                printf("Stack Overflow\n");
                return;
        }
        st->top++;
        st->checklist[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
        strcpy(st->checklist[st->top], item);
        printf("Checklist item '%s' added to stack\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Checklist item '%s' removed from stack\n", st->checklist[st->top]);
    free(st->checklist[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Current checklist:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->checklist[i]);
    }
}

void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Top checklist item: %s\n", st->checklist[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->checklist[i], item) == 0) {
            printf("Checklist item '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Checklist item '%s' not found in the stack\n", item);
}

//4.

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

struct Stack {
    char **data;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice, size;
    char item[100];

    printf("Enter the maximum size of the stack: ");
    scanf("%d", &size);

    create(&st, size);
```

```
while (1) {
    printf("\n--- Telemetry Data Storage ---\n");
    printf("1: Push new telemetry data\n");
    printf("2: Pop the last data entry\n");
    printf("3: View the stored telemetry data\n");
    printf("4: Peek at the most recent data entry\n");
    printf("5: Search for specific telemetry data\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
    case 1:
        printf("Enter telemetry data: ");
        getchar();
        scanf("%[^\n]", item);
        push(&st, item);
        break;
    case 2:
        pop(&st);
        break;
    case 3:
        display(&st);
```

```
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter telemetry data to search for: ");
            getchar();
            scanf("%[^\n]", item);
            search(&st, item);
            break;
        case 6:
            printf("Exiting...\n");
            free(st.data);
            exit(0);
        default:
            printf("Invalid choice\n");
        }
    }
    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->data = (char **)malloc(size * sizeof(char *));
    if (st->data == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->data[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->data[st->top], item);
    printf("Telemetry data '%s' added to stack\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Telemetry data '%s' removed from stack\n", st->data[st->top]);
    free(st->data[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Stored telemetry data:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->data[i]);
    }
```

```
}


void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Most recent telemetry data: %s\n", st->data[st->top]);
}


void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->data[i], item) == 0) {
            printf("Telemetry data '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Telemetry data '%s' not found in the stack\n", item);
}



//5.
```

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

struct Stack {
	char **task;
	int top;
	int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
	struct Stack st;
	int choice, size;
	char item[100];
```

```
printf("Enter the maximum size of the stack: ");
scanf("%d", &size);

create(&st, size);

while (1) {
    printf("\n--- Space Mission Task Manager ---\n");
    printf("1: Add a task (push)\n");
    printf("2: Mark the last task as completed (pop)\n");
    printf("3: List all pending tasks\n");
    printf("4: Peek at the most recent task\n");
    printf("5: Search for a specific task\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
    case 1:
        printf("Enter the task: ");
        getchar();
        scanf("%[^\n]", item);
        push(&st, item);
        break;
    case 2:
        pop(&st);
        break;
    case 3:
        display(&st);
        break;
    case 4:
        peek(&st);
        break;
    case 5:
        printf("Enter the task to search for: ");
        getchar();
        scanf("%[^\n]", item);
        search(&st, item);
        break;
    case 6:
        printf("Exiting...\n");
        free(st.task);
        exit(0);
    default:
        printf("Invalid choice\n");
    }
```

```
        }
        return 0;
}

void create(struct Stack *st, int size) {
        st->size = size;
        st->top = -1;
        st->task = (char **)malloc(size * sizeof(char *));
        if (st->task == NULL) {
                printf("Memory allocation failed\n");
                exit(1);
        }
}

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

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->task[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->task[st->top], item);
    printf("Task '%s' added to stack\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Task '%s' marked as completed\n", st->task[st->top]);
    free(st->task[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Pending tasks:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->task[i]);
    }
}

void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Most recent task: %s\n", st->task[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->task[i], item) == 0) {
            printf("Task '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Task '%s' not found in the stack\n", item);
}

//6.

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

struct Stack {
    char **countdownStep;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);
```

```
int main() {
    struct Stack st;
    int choice, size;
    char item[100];

    printf("Enter the maximum size of the stack: ");
    scanf("%d", &size);

    create(&st, size);

    while (1) {
        printf("\n--- Launch Countdown Management ---\n");
        printf("1: Add a countdown step (push)\n");
        printf("2: Remove the last step (pop)\n");
        printf("3: Display the current countdown\n");
        printf("4: Peek at the next countdown step\n");
        printf("5: Search for a specific countdown step\n");
        printf("6: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
```

```
case 1:
        printf("Enter the countdown step: ");
        getchar();
        scanf("%[^\n]", item);
        push(&st, item);
        break;
case 2:
        pop(&st);
        break;
case 3:
        display(&st);
        break;
case 4:
        peek(&st);
        break;
case 5:
        printf("Enter the countdown step to search for: ");
        getchar();
        scanf("%[^\n]", item);
        search(&st, item);
        break;
case 6:
        printf("Exiting...\n");
```

```
                        free(st.countdownStep);
                        exit(0);
                default:
                        printf("Invalid choice\n");
                }
        }
        return 0;
}

void create(struct Stack *st, int size) {
        st->size = size;
        st->top = -1;
        st->countdownStep = (char **)malloc(size * sizeof(char *));
        if (st->countdownStep == NULL) {
                printf("Memory allocation failed\n");
                exit(1);
        }
}

int isFull(struct Stack *st) {
        return st->top == st->size - 1;
}
```

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->countdownStep[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->countdownStep[st->top], item);
    printf("Countdown step '%s' added to stack\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Countdown step '%s' removed from stack\n", st->countdownStep[st->top]);
    free(st->countdownStep[st->top]);
    st->top--;
```

```
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Current countdown sequence:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->countdownStep[i]);
    }
}

void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Next countdown step: %s\n", st->countdownStep[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
```

```
		if (strcmp(st->countdownStep[i], item) == 0) {
			printf("Countdown step '%s' found at position %d\n", item, i + 1);
			return;
		}
	}
	printf("Countdown step '%s' not found in the stack\n", item);
}

//7.

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

struct Stack {
	char **log;
	int top;
	int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
```

```
void push(struct Stack *st, char item[]);

void pop(struct Stack *st);

void display(struct Stack *st);

void peek(struct Stack *st);

void search(struct Stack *st, char item[]);


int main() {

        struct Stack st;

        int choice, size;

        char item[100];


        printf("Enter the maximum size of the stack: ");

        scanf("%d", &size);


        create(&st, size);


        while (1) {

                printf("\n--- Aircraft Maintenance Logs ---\n");

                printf("1: Add a new log (push)\n");

                printf("2: Remove the last log (pop)\n");

                printf("3: View all maintenance logs\n");

                printf("4: Peek at the latest maintenance log\n");

                printf("5: Search for a specific maintenance log\n");
                printf("6: Exit\n");
                printf("Enter your choice: ");
                scanf("%d", &choice);

                switch (choice) {
                case 1:
                        printf("Enter the maintenance log: ");
                        getchar();
                        scanf("%[^\n]", item);
                        push(&st, item);
                        break;
                case 2:
                        pop(&st);
                        break;
                case 3:
                        display(&st);
                        break;
                case 4:
                        peek(&st);
                        break;
                case 5:
                        printf("Enter the log to search for: ");
                        getchar();
        }

                        scanf("%[^\n]", item);

                        search(&st, item);

                        break;

                case 6:

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

                        free(st.log);

                        exit(0);

                default:

                        printf("Invalid choice\n");

                }

            }

            return 0;

        }


        void create(struct Stack *st, int size) {

            st->size = size;

            st->top = -1;

            st->log = (char **)malloc(size * sizeof(char *));

            if (st->log == NULL) {

                printf("Memory allocation failed\n");

                exit(1);

            }

        }
```

```
int isFull(struct Stack *st) {

        return st->top == st->size - 1;

}


int isEmpty(struct Stack *st) {

        return st->top == -1;

}


void push(struct Stack *st, char item[]) {

        if (isFull(st)) {

                printf("Stack Overflow\n");

                return;

        }

        st->top++;

        st->log[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));

        strcpy(st->log[st->top], item);

        printf("Maintenance log '%s' added to stack\n", item);

}


void pop(struct Stack *st) {

        if (isEmpty(st)) {

                printf("Stack is empty\n");
        return;
    }
    printf("Maintenance log '%s' removed from stack\n", st->log[st->top]);
    free(st->log[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("All maintenance logs:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->log[i]);
    }
}

void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
```

```
        printf("Latest maintenance log: %s\n", st->log[st->top]);
}


void search(struct Stack *st, char item[]) {
        for (int i = st->top; i >= 0; i--) {
                if (strcmp(st->log[i], item) == 0) {
                        printf("Maintenance log '%s' found at position %d\n", item, i + 1);
                        return;
                }
        }
        printf("Maintenance log '%s' not found in the stack\n", item);
}



//8.


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


struct Stack {
    char **procedureStep;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice, size;
    char item[100];

    printf("Enter the maximum size of the stack: ");
    scanf("%d", &size);

    create(&st, size);

    while (1) {
```

```
printf("\n--- Spacecraft Docking Procedure ---\n");

printf("1: Push a new step\n");

printf("2: Pop the last step\n");

printf("3: Display the procedure steps\n");

printf("4: Peek at the next step in the procedure\n");

printf("5: Search for a specific step\n");

printf("6: Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);


switch (choice) {

    case 1:

        printf("Enter the procedure step: ");

        getchar();

        scanf("%[^\n]", item);

        push(&st, item);

        break;

    case 2:

        pop(&st);

        break;

    case 3:

        display(&st);

        break;
```

```
            case 4:
                peek(&st);
                break;
            case 5:
                printf("Enter the procedure step to search for: ");
                getchar();
                scanf("%[^\n]", item);
                search(&st, item);
                break;
            case 6:
                printf("Exiting...\n");
                free(st.procedureStep);
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->procedureStep = (char **)malloc(size * sizeof(char *));
    if (st->procedureStep == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->procedureStep[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->procedureStep[st->top], item);
    printf("Procedure step '%s' added to stack\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Procedure step '%s' removed from stack\n", st->procedureStep[st->top]);
    free(st->procedureStep[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Procedure steps:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->procedureStep[i]);
    }
}
```

```
void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Next procedure step: %s\n", st->procedureStep[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->procedureStep[i], item) == 0) {
            printf("Procedure step '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Procedure step '%s' not found in the stack\n", item);
}

//9.

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

```
#include <string.h>


struct Stack {

    char **command;

    int top;

    int size;

};


void create(struct Stack *st, int size);

int isFull(struct Stack *st);

int isEmpty(struct Stack *st);

void push(struct Stack *st, char item[]);

void pop(struct Stack *st);

void display(struct Stack *st);

void peek(struct Stack *st);

void search(struct Stack *st, char item[]);


int main() {

    struct Stack st;

    int choice, size;

    char item[100];


    printf("Enter the maximum size of the stack: ");
```

```
scanf("%d", &size);

create(&st, size);

while (1) {
    printf("\n--- Mission Control Command History ---\n");
    printf("1: Add a command\n");
    printf("2: Undo the last command\n");
    printf("3: View the command history\n");
    printf("4: Peek at the most recent command\n");
    printf("5: Search for a specific command\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the command: ");
            getchar();
            scanf("%[^\n]", item);
            push(&st, item);
            break;
        case 2:
            pop(&st);

            break;

        case 3:

            display(&st);

            break;

        case 4:

            peek(&st);

            break;

        case 5:

            printf("Enter the command to search for: ");

            getchar();

            scanf("%[^\n]", item);

            search(&st, item);

            break;

        case 6:

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

            free(st.command);

            exit(0);

        default:

            printf("Invalid choice\n");

    }

}

return 0;
```

```
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->command = (char **)malloc(size * sizeof(char *));
    if (st->command == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->command[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->command[st->top], item);
    printf("Command '%s' added\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Command '%s' removed\n", st->command[st->top]);
    free(st->command[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
```

```
}

    printf("Command history:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->command[i]);
    }
}

void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Most recent command: %s\n", st->command[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->command[i], item) == 0) {
            printf("Command '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Command '%s' not found\n", item);
}
```

```
//10.

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

struct Stack {
    char **event;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
```

```
struct Stack st;
int choice, size;
char item[100];

printf("Enter the maximum size of the stack: ");
scanf("%d", &size);

create(&st, size);

while (1) {
    printf("\n--- Aerospace Simulation Events ---\n");
    printf("1: Push a new event\n");
    printf("2: Pop the last event\n");
    printf("3: Display all events\n");
    printf("4: Peek at the most recent event\n");
    printf("5: Search for a specific event\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the simulation event: ");
            getchar();
            scanf("%[^\n]", item);
            push(&st, item);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the event to search for: ");
            getchar();
            scanf("%[^\n]", item);
            search(&st, item);
            break;
        case 6:
            printf("Exiting...\n");
            free(st.event);
            exit(0);
```

```
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->event = (char **)malloc(size * sizeof(char *));
    if (st->event == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

int isEmpty(struct Stack *st) {
    return st->top == -1;
```

```
}

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->event[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->event[st->top], item);
    printf("Event '%s' added\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Event '%s' removed\n", st->event[st->top]);
    free(st->event[st->top]);
    st->top--;
}
```

```
void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Simulation events:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->event[i]);
    }
}

void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Most recent event: %s\n", st->event[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->event[i], item) == 0) {
            printf("Event '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Event '%s' not found\n", item);
}

//11.

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

struct Stack {
    char **maneuver;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice, size;
    char item[100];

    printf("Enter the maximum size of the stack: ");
    scanf("%d", &size);

    create(&st, size);

    while (1) {
        printf("\n--- Pilot Training Maneuver Stack ---\n");
        printf("1: Add a maneuver\n");
        printf("2: Remove the last maneuver\n");
        printf("3: View all maneuvers\n");
        printf("4: Peek at the most recent maneuver\n");
        printf("5: Search for a specific maneuver\n");
        printf("6: Exit\n");
        printf("Enter your choice: ");
```

```
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter the maneuver: ");
        getchar();
        scanf("%[^\n]", item);
        push(&st, item);
        break;
    case 2:
        pop(&st);
        break;
    case 3:
        display(&st);
        break;
    case 4:
        peek(&st);
        break;
    case 5:
        printf("Enter the maneuver to search for: ");
        getchar();
        scanf("%[^\n]", item);
        search(&st, item);
```

```
                break;
            case 6:
                printf("Exiting...\n");
                free(st.maneuver);
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->maneuver = (char **)malloc(size * sizeof(char *));
    if (st->maneuver == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->maneuver[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->maneuver[st->top], item);
    printf("Maneuver '%s' added\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Maneuver '%s' removed\n", st->maneuver[st->top]);

    free(st->maneuver[st->top]);

    st->top--;

}


void display(struct Stack *st) {

    if (isEmpty(st)) {

        printf("Stack is empty\n");

        return;

    }

    printf("Training maneuvers:\n");

    for (int i = st->top; i >= 0; i--) {

        printf("%d: %s\n", i + 1, st->maneuver[i]);

    }

}


void peek(struct Stack *st) {

    if (isEmpty(st)) {

        printf("Stack is empty\n");

        return;

    }

    printf("Most recent maneuver: %s\n", st->maneuver[st->top]);

}
```

```
void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->maneuver[i], item) == 0) {
            printf("Maneuver '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Maneuver '%s' not found\n", item);
}

//12.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Stack {
    char **command;
    int top;
    int size;
};

void create(struct Stack *st, int size);
```

```
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice, size;
    char item[100];

    printf("Enter the maximum size of the stack: ");
    scanf("%d", &size);

    create(&st, size);

    while (1) {
        printf("\n--- Satellite Operation Commands ---\n");
        printf("1: Push a new command\n");
        printf("2: Pop the last command\n");
        printf("3: View the operation commands\n");
        printf("4: Peek at the most recent command\n");

        printf("5: Search for a specific command\n");

        printf("6: Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);


        switch (choice) {

            case 1:

                printf("Enter the operation command: ");

                getchar();

                scanf("%[^\n]", item);

                push(&st, item);

                break;

            case 2:

                pop(&st);

                break;

            case 3:

                display(&st);

                break;

            case 4:

                peek(&st);

                break;

            case 5:
```

```
                printf("Enter the command to search for: ");
                getchar();
                scanf("%[^\n]", item);
                search(&st, item);
                break;
            case 6:
                printf("Exiting...\n");
                free(st.command);
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->command = (char **)malloc(size * sizeof(char *));
    if (st->command == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->command[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->command[st->top], item);
    printf("Operation command '%s' added\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Operation command '%s' removed\n", st->command[st->top]);
    free(st->command[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Operation commands:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->command[i]);
    }
}

void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Most recent operation command: %s\n", st->command[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->command[i], item) == 0) {
            printf("Command '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Command '%s' not found\n", item);
}
//13.

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

struct Stack {
    char **procedure;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice, size;
    char item[100];

    printf("Enter the maximum size of the stack: ");
    scanf("%d", &size);

    create(&st, size);

    while (1) {
```

```
printf("\n--- Emergency Procedures for Spacecraft ---\n");

printf("1: Add a procedure\n");

printf("2: Remove the last procedure\n");

printf("3: View all procedures\n");

printf("4: Peek at the next procedure\n");

printf("5: Search for a specific procedure\n");

printf("6: Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);


switch (choice) {

    case 1:

        printf("Enter the emergency procedure: ");

        getchar();

        scanf("%[^\n]", item);

        push(&st, item);

        break;

    case 2:

        pop(&st);

        break;

    case 3:

        display(&st);

        break;
```

```
            case 4:
                peek(&st);
                break;
            case 5:
                printf("Enter the procedure to search for: ");
                getchar();
                scanf("%[^\n]", item);
                search(&st, item);
                break;
            case 6:
                printf("Exiting...\n");
                free(st.procedure);
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->procedure = (char **)malloc(size * sizeof(char *));
    if (st->procedure == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->procedure[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->procedure[st->top], item);
    printf("Emergency procedure '%s' added\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Emergency procedure '%s' removed\n", st->procedure[st->top]);
    free(st->procedure[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Emergency procedures:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->procedure[i]);
    }
}
```

```
void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Next emergency procedure: %s\n", st->procedure[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->procedure[i], item) == 0) {
            printf("Procedure '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Procedure '%s' not found\n", item);
}

//14.

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

```
#include <string.h>

struct Stack {
    char **activity;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice, size;
    char item[100];

    printf("Enter the maximum size of the stack: ");
```

```
scanf("%d", &size);

create(&st, size);

while (1) {
    printf("\n--- Astronaut Activity Log ---\n");
    printf("1: Add a new activity\n");
    printf("2: Remove the last activity\n");
    printf("3: View all activity log\n");
    printf("4: Peek at the most recent activity\n");
    printf("5: Search for a specific activity\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the astronaut activity: ");
            getchar();
            scanf("%[^\n]", item);
            push(&st, item);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the activity to search for: ");
            getchar();
            scanf("%[^\n]", item);
            search(&st, item);
            break;
        case 6:
            printf("Exiting...\n");
            free(st.activity);
            exit(0);
        default:
            printf("Invalid choice\n");
    }
}
return 0;
```

```
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->activity = (char **)malloc(size * sizeof(char *));
    if (st->activity == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->activity[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->activity[st->top], item);
    printf("Activity '%s' added\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Activity '%s' removed\n", st->activity[st->top]);
    free(st->activity[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Astronaut activity log:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->activity[i]);
    }
}

void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Most recent activity: %s\n", st->activity[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->activity[i], item) == 0) {
            printf("Activity '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Activity '%s' not found\n", item);
}
```

```
//15.

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

struct Stack {
    char **fuelUsage;
    int top;
    int size;
};

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
```

```
struct Stack st;
int choice, size;
char item[100];

printf("Enter the maximum size of the stack: ");
scanf("%d", &size);

create(&st, size);

while (1) {
    printf("\n--- Fuel Management System ---\n");
    printf("1: Add a fuel usage entry\n");
    printf("2: Remove the last entry\n");
    printf("3: View all fuel usage data\n");
    printf("4: Peek at the latest fuel usage entry\n");
    printf("5: Search for a specific fuel usage entry\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the fuel usage data: ");
            getchar();
            scanf("%[^\n]", item);
            push(&st, item);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the fuel usage entry to search for: ");
            getchar();
            scanf("%[^\n]", item);
            search(&st, item);
            break;
        case 6:
            printf("Exiting...\n");
            free(st.fuelUsage);
            exit(0);
```

```
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->fuelUsage = (char **)malloc(size * sizeof(char *));
    if (st->fuelUsage == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

int isEmpty(struct Stack *st) {
    return st->top == -1;
```

```
}

void push(struct Stack *st, char item[]) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->fuelUsage[st->top] = (char *)malloc((strlen(item) + 1) * sizeof(char));
    strcpy(st->fuelUsage[st->top], item);
    printf("Fuel usage data '%s' added\n", item);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Fuel usage data '%s' removed\n", st->fuelUsage[st->top]);
    free(st->fuelUsage[st->top]);
    st->top--;
}
```

```
void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Fuel usage data:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d: %s\n", i + 1, st->fuelUsage[i]);
    }
}

void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Latest fuel usage data: %s\n", st->fuelUsage[st->top]);
}

void search(struct Stack *st, char item[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->fuelUsage[i], item) == 0) {
            printf("Fuel usage entry '%s' found at position %d\n", item, i + 1);
            return;
        }
    }
    printf("Fuel usage entry '%s' not found\n", item);
}
//1 #include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Order {
    int orderID;
    char customerName[50];
    struct Order* next;
};

struct Stack {
    struct Order* top;
};

void initialize(struct Stack* stack) {
    stack->top = NULL;
}
```

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

void push(struct Stack* stack, int id, char* name) {
    struct Order* newOrder = (struct Order*)malloc(sizeof(struct Order));
    if (!newOrder) {
        printf("Memory allocation failed.\n");
        return;
    }
    newOrder->orderID = id;
    strcpy(newOrder->customerName, name);
    newOrder->next = stack->top;
    stack->top = newOrder;
    printf("Order %d for %s added.\n", id, name);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No orders to process.\n");
        return;
    }
    struct Order* temp = stack->top;
    printf("Processing order %d for %s.\n", temp->orderID, temp->customerName);
    stack->top = stack->top->next;
    free(temp);
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No pending orders.\n");
        return;
    }
    printf("Pending orders:\n");
    struct Order* current = stack->top;
    while (current) {
        printf("Order ID: %d, Customer: %s\n", current->orderID, current->customerName);
        current = current->next;
    }
}

void peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No orders to process.\n");
        return;
    }
    printf("Next order to process: Order ID %d, Customer: %s\n", stack->top->orderID, stack->top->customerName);
}

void search(struct Stack* stack, int id) {
    struct Order* current = stack->top;
    while (current) {
        if (current->orderID == id) {
            printf("Order found: Order ID %d, Customer: %s\n", current->orderID, current->customerName);
            return;
        }
        current = current->next;
    }
    printf("Order ID %d not found.\n", id);
}

int main() {
    struct Stack stack;
    initialize(&stack);
    int choice, id;
    char name[50];

    do {
```

```
printf("\nOrder Processing System\n");
printf("1. Add a new order (push)\n");
printf("2. Process the last order (pop)\n");
printf("3. Display all pending orders\n");
printf("4. Peek at the next order to be processed\n");
printf("5. Search for a specific order\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter Order ID: ");
        scanf("%d", &id);
        printf("Enter Customer Name: ");
        scanf("%s", name);
        push(&stack, id, name);
        break;
    case 2:
        pop(&stack);
        break;
    case 3:
        display(&stack);
```

```
                break;
            case 4:
                peek(&stack);
                break;
            case 5:
                printf("Enter Order ID to search: ");
                scanf("%d", &id);
                search(&stack, id);
                break;
            case 6:
                printf("Exiting the system.\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 6);


    return 0;
}
```

2. #include <stdio.h>

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

```
struct Node {
    char ticket[100];
    struct Node* next;
};

void push(struct Node** top, char ticket[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    strcpy(newNode->ticket, ticket);
    newNode->next = *top;
    *top = newNode;
    printf("Ticket '%s' added.\n", ticket);
}

void pop(struct Node** top) {
    if (*top == NULL) {
        printf("No tickets to resolve.\n");
        return;
    }
    struct Node* temp = *top;
    printf("Ticket '%s' resolved.\n", temp->ticket);

    *top = (*top)->next;

    free(temp);

}


void display(struct Node* top) {

    if (top == NULL) {

        printf("No pending tickets.\n");

        return;

    }

    printf("Pending tickets:\n");

    while (top != NULL) {

        printf("- %s\n", top->ticket);

        top = top->next;

    }

}


void peek(struct Node* top) {

    if (top == NULL) {

        printf("No tickets to peek.\n");

        return;

    }

    printf("Latest ticket: %s\n", top->ticket);
```

```
}

void search(struct Node* top, char ticket[]) {
    int position = 1;
    while (top != NULL) {
        if (strcmp(top->ticket, ticket) == 0) {
            printf("Ticket '%s' found at position %d.\n", ticket, position);
            return;
        }
        top = top->next;
        position++;
    }
    printf("Ticket '%s' not found.\n", ticket);
}

int main() {
    struct Node* stack = NULL;
    int choice;
    char ticket[100];

    do {
        printf("\nCustomer Support Ticketing System\n");
        printf("1. Add a new ticket\n");
        printf("2. Resolve the latest ticket\n");
        printf("3. View all pending tickets\n");
        printf("4. Peek at the latest ticket\n");
        printf("5. Search for a specific ticket\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();  // Consume newline

        switch (choice) {
            case 1:
                printf("Enter ticket: ");
                scanf("%[^\n]", ticket);
                push(&stack, ticket);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                display(stack);
                break;
            case 4:
                peek(stack);
```

```
                break;
            case 5:
                printf("Enter ticket to search: ");
                scanf("%[^\n]", ticket);
                search(stack, ticket);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}
3. #include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Node {
    char product[100];
    struct Node* next;
```

```
};


void push(struct Node** top, char product[]) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    if (!newNode) {

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

        return;

    }

    strcpy(newNode->product, product);

    newNode->next = *top;

    *top = newNode;

    printf("Return request for '%s' added.\n", product);

}


void pop(struct Node** top) {

    if (*top == NULL) {

        printf("No return requests to process.\n");

        return;

    }

    struct Node* temp = *top;

    printf("Processing return for '%s'.\n", temp->product);

    *top = (*top)->next;

    free(temp);
```

```
}


void display(struct Node* top) {
    if (top == NULL) {
        printf("No pending return requests.\n");
        return;
    }
    printf("Pending return requests:\n");
    while (top != NULL) {
        printf("- %s\n", top->product);
        top = top->next;
    }
}


void peek(struct Node* top) {
    if (top == NULL) {
        printf("No return requests to peek.\n");
        return;
    }
    printf("Next return to process: %s\n", top->product);
}


void search(struct Node* top, char product[]) {
    int position = 1;
    while (top != NULL) {
        if (strcmp(top->product, product) == 0) {
            printf("Return request for '%s' found at position %d.\n", product, position);
            return;
        }
        top = top->next;
        position++;
    }
    printf("Return request for '%s' not found.\n", product);
}

int main() {
    struct Node* stack = NULL;
    int choice;
    char product[100];

    do {
        printf("\nProduct Return Management System\n");
        printf("1. Add a new return request\n");
        printf("2. Process the last return\n");
        printf("3. Display all return requests\n");
        printf("4. Peek at the next return to process\n");
        printf("5. Search for a specific return request\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();  // Consume newline

        switch (choice) {
            case 1:
                printf("Enter product name: ");
                scanf("%[^\n]", product);
                push(&stack, product);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                display(stack);
                break;
            case 4:
                peek(stack);
                break;
            case 5:
                printf("Enter product to search: ");
```

```
                scanf("%[^\n]", product);
                search(stack, product);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}
```

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

struct Node {
    char item[100];
    struct Node* next;
};

void push(struct Node** top, char item[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    strcpy(newNode->item, item);
    newNode->next = *top;
    *top = newNode;
    printf("Restock entry for '%s' added.\n", item);
}

void pop(struct Node** top) {
    if (*top == NULL) {
        printf("No restock entries to process.\n");
        return;
    }
    struct Node* temp = *top;
    printf("Processing restock for '%s'.\n", temp->item);
    *top = (*top)->next;
    free(temp);
}

void display(struct Node* top) {
    if (top == NULL) {
        printf("No pending restock entries.\n");
        return;
    }
    printf("Pending restock entries:\n");
    while (top != NULL) {
        printf("- %s\n", top->item);
        top = top->next;
    }
}

void peek(struct Node* top) {
    if (top == NULL) {
        printf("No restock entries to peek.\n");
        return;
    }
    printf("Next restock to process: %s\n", top->item);
}

void search(struct Node* top, char item[]) {
    int position = 1;
    while (top != NULL) {
        if (strcmp(top->item, item) == 0) {
            printf("Restock entry for '%s' found at position %d.\n", item, position);

            return;

        }

        top = top->next;

        position++;

    }

    printf("Restock entry for '%s' not found.\n", item);

}


int main() {

    struct Node* stack = NULL;

    int choice;

    char item[100];


    do {

        printf("\nInventory Restock System\n");

        printf("1. Add a restock entry\n");

        printf("2. Process the last restock\n");

        printf("3. View all restock entries\n");

        printf("4. Peek at the latest restock entry\n");

        printf("5. Search for a specific restock entry\n");

        printf("6. Exit\n");

        printf("Enter your choice: ");
```

```
scanf("%d", &choice);
getchar();  // Consume newline

switch (choice) {
    case 1:
        printf("Enter item name: ");
        scanf("%[^\n]", item);
        push(&stack, item);
        break;
    case 2:
        pop(&stack);
        break;
    case 3:
        display(stack);
        break;
    case 4:
        peek(stack);
        break;
    case 5:
        printf("Enter item to search: ");
        scanf("%[^\n]", item);
        search(stack, item);
        break;
```

```
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}
```

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

struct Node {
    char deal[100];
    struct Node* next;
};

void push(struct Node** top, char deal[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    strcpy(newNode->deal, deal);
    newNode->next = *top;
    *top = newNode;
    printf("Deal '%s' added.\n", deal);
}

void pop(struct Node** top) {
    if (*top == NULL) {
        printf("No deals to remove.\n");
        return;
    }
    struct Node* temp = *top;
    printf("Removing deal: %s\n", temp->deal);
    *top = (*top)->next;
    free(temp);
}

void display(struct Node* top) {
    if (top == NULL) {
        printf("No active deals.\n");
        return;
    }
    printf("Active deals:\n");
    while (top != NULL) {
        printf("- %s\n", top->deal);
        top = top->next;
    }
}

void peek(struct Node* top) {
    if (top == NULL) {
        printf("No active deals to peek.\n");
        return;
    }
    printf("Latest deal: %s\n", top->deal);
}

void search(struct Node* top, char deal[]) {
    int position = 1;
    while (top != NULL) {
        if (strcmp(top->deal, deal) == 0) {
            printf("Deal '%s' found at position %d.\n", deal, position);
            return;
        }
        top = top->next;
        position++;
    }
    printf("Deal '%s' not found.\n", deal);
}

int main() {
    struct Node* stack = NULL;
    int choice;
    char deal[100];

    do {
        printf("\nFlash Sale Deal Management\n");
        printf("1. Add a new deal\n");
        printf("2. Remove the last deal\n");
        printf("3. View all active deals\n");
        printf("4. Peek at the latest deal\n");
        printf("5. Search for a specific deal\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        if (choice == 1 || choice == 5) {
            printf("Enter deal: ");
            scanf(" %[^\n]", deal);
        }

        switch (choice) {
            case 1:
                push(&stack, deal);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                display(stack);
                break;
            case 4:
                peek(stack);
                break;
            case 5:
                search(stack, deal);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
```

```
                printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}
```

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

struct Node {
    char session[100];
    struct Node* next;
};

void push(struct Node** top, char session[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    strcpy(newNode->session, session);
    newNode->next = *top;
    *top = newNode;
    printf("Session '%s' added.\n", session);
}

void pop(struct Node** top) {
    if (*top == NULL) {
        printf("No sessions to end.\n");
        return;
    }
    struct Node* temp = *top;
    printf("Ending session: %s\n", temp->session);
    *top = (*top)->next;
    free(temp);
}

void display(struct Node* top) {
    if (top == NULL) {
        printf("No sessions recorded.\n");
        return;
    }
    printf("User sessions:\n");
    while (top != NULL) {
        printf("- %s\n", top->session);
        top = top->next;
    }
}

void peek(struct Node* top) {
    if (top == NULL) {
        printf("No sessions to peek.\n");
        return;
    }
    printf("Most recent session: %s\n", top->session);
}

void search(struct Node* top, char session[]) {
    int position = 1;
    while (top != NULL) {
        if (strcmp(top->session, session) == 0) {
            printf("Session '%s' found at position %d.\n", session, position);
            return;
        }
        top = top->next;
        position++;
    }
    printf("Session '%s' not found.\n", session);
```

```
}


int main() {
    struct Node* stack = NULL;
    int choice;
    char session[100];

    do {
        printf("\nUser Session History\n");
        printf("1. Add a session\n");
        printf("2. End the last session\n");
        printf("3. Display all sessions\n");
        printf("4. Peek at the most recent session\n");
        printf("5. Search for a specific session\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        if (choice == 1 || choice == 5) {
            printf("Enter session: ");
            scanf(" %[^\n]", session);
        }

        switch (choice) {
            case 1:
                push(&stack, session);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                display(stack);
                break;
            case 4:
                peek(stack);
                break;
            case 5:
                search(stack, session);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 6);
    return 0;
}
7.  #include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Node {
    char product[100];
    struct Node* next;
};

void push(struct Node** top, char product[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    strcpy(newNode->product, product);
    newNode->next = *top;
    *top = newNode;
    printf("Product '%s' added to wishlist.\n", product);
}
```

```
void pop(struct Node** top) {
    if (*top == NULL) {
        printf("Wishlist is empty.\n");
        return;
    }
    struct Node* temp = *top;
    printf("Removing product: %s\n", temp->product);
    *top = (*top)->next;
    free(temp);
}

void display(struct Node* top) {
    if (top == NULL) {
        printf("Wishlist is empty.\n");
        return;
    }
    printf("Wishlist items:\n");
    while (top != NULL) {
        printf("- %s\n", top->product);
        top = top->next;
    }
}
```

```c
void peek(struct Node* top) {
    if (top == NULL) {
        printf("Wishlist is empty.\n");
        return;
    }
    printf("Most recent wishlist item: %s\n", top->product);
}

void search(struct Node* top, char product[]) {
    int position = 1;
    while (top != NULL) {
        if (strcmp(top->product, product) == 0) {
            printf("Product '%s' found at position %d.\n", product, position);
            return;
        }
        top = top->next;
        position++;
    }
    printf("Product '%s' not found.\n", product);
}

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

```
int choice;

char product[100];


do {

    printf("\nWishlist Management\n");

    printf("1. Add a product to wishlist\n");

    printf("2. Remove the last added product\n");

    printf("3. View all wishlist items\n");

    printf("4. Peek at the most recent wishlist item\n");

    printf("5. Search for a specific product\n");

    printf("6. Exit\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);

    if (choice == 1 || choice == 5) {

        printf("Enter product: ");

        scanf(" %[^\n]", product);

    }


    switch (choice) {

        case 1:

            push(&stack, product);

            break;

        case 2:
                pop(&stack);
                break;
            case 3:
                display(stack);
                break;
            case 4:
                peek(stack);
                break;
            case 5:
                search(stack, product);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}
```

8\.

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

```
#include <string.h>

struct Node {
    char step[100];
    struct Node* next;
};

void push(struct Node** top, char step[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    strcpy(newNode->step, step);
    newNode->next = *top;
    *top = newNode;
    printf("Step '%s' added to checkout process.\n", step);
}

void pop(struct Node** top) {
    if (*top == NULL) {
        printf("No steps in checkout process.\n");
        return;
    }
    struct Node* temp = *top;
    printf("Removing step: %s\n", temp->step);
    *top = (*top)->next;
    free(temp);
}

void display(struct Node* top) {
    if (top == NULL) {
        printf("No steps in checkout process.\n");
        return;
    }
    printf("Checkout process steps:\n");
    while (top != NULL) {
        printf("- %s\n", top->step);
        top = top->next;
    }
}

void peek(struct Node* top) {
    if (top == NULL) {
        printf("No steps in checkout process.\n");
        return;
    }
    printf("Current step: %s\n", top->step);
}

void search(struct Node* top, char step[]) {
    int position = 1;
    while (top != NULL) {
        if (strcmp(top->step, step) == 0) {
            printf("Step '%s' found at position %d.\n", step, position);
            return;
        }
        top = top->next;
        position++;
    }
    printf("Step '%s' not found.\n", step);
}

int main() {
    struct Node* stack = NULL;
    int choice;
    char step[100];

    do {
```

```
printf("\nCheckout Process Management\n");
printf("1. Add a checkout step\n");
printf("2. Remove the last step\n");
printf("3. Display all checkout steps\n");
printf("4. Peek at the current step\n");
printf("5. Search for a specific step\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
if (choice == 1 || choice == 5) {
    printf("Enter checkout step: ");
    scanf(" %[^\n]", step);
}

switch (choice) {
    case 1:
        push(&stack, step);
        break;
    case 2:
        pop(&stack);
        break;
    case 3:
        display(stack);
```

```
                break;
            case 4:
                peek(stack);
                break;
            case 5:
                search(stack, step);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}
```

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

struct Node {
    char code[100];
    struct Node* next;

};


void push(struct Node** top, char code[]) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    if (!newNode) {

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

        return;

    }

    strcpy(newNode->code, code);

    newNode->next = *top;

    *top = newNode;

    printf("Coupon code '%s' added.\n", code);

}


void pop(struct Node** top) {

    if (*top == NULL) {

        printf("No coupon codes available.\n");

        return;

    }

    struct Node* temp = *top;

    printf("Removing coupon code: %s\n", temp->code);

    *top = (*top)->next;
    free(temp);
}

void display(struct Node* top) {
    if (top == NULL) {
        printf("No coupon codes available.\n");
        return;
    }
    printf("Available coupon codes:\n");
    while (top != NULL) {
        printf("- %s\n", top->code);
        top = top->next;
    }
}

void peek(struct Node* top) {
    if (top == NULL) {
        printf("No coupon codes available.\n");
        return;
    }
    printf("Latest coupon code: %s\n", top->code);
}
```

```
void search(struct Node* top, char code[]) {
    int position = 1;
    while (top != NULL) {
        if (strcmp(top->code, code) == 0) {
            printf("Coupon code '%s' found at position %d.\n", code, position);
            return;
        }
        top = top->next;
        position++;
    }
    printf("Coupon code '%s' not found.\n", code);
}

int main() {
    struct Node* stack = NULL;
    int choice;
    char code[100];

    do {
        printf("\nCoupon Code Management\n");
        printf("1. Add a new coupon code\n");
        printf("2. Remove the last coupon code\n");
        printf("3. View all available coupon codes\n");
        printf("4. Peek at the latest coupon code\n");

        printf("5. Search for a specific coupon code\n");

        printf("6. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        if (choice == 1 || choice == 5) {

            printf("Enter coupon code: ");

            scanf(" %[^\n]", code); // Space before % ensures it skips any newline character

        }


        switch (choice) {

            case 1:

                push(&stack, code);

                break;

            case 2:

                pop(&stack);

                break;

            case 3:

                display(stack);

                break;

            case 4:

                peek(stack);

                break;
```

```
            case 5:
                search(stack, code);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 6);

    return 0;
}
```

10.

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

// Define the structure for a linked list node
struct Node {
    char status[100];  // To hold the shipping status
    struct Node* next; // Pointer to the next node
};
```

```
// Function to create a new node
struct Node* createNode(char* status) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->status, status);
    newNode->next = NULL;
    return newNode;
}

// Function to add a new shipping status update (push)
void push(struct Node** top, char* status) {
    struct Node* newNode = createNode(status);
    newNode->next = *top;
    *top = newNode;
    printf("Shipping status added: %s\n", status);
}

// Function to remove the last update (pop)
void pop(struct Node** top) {
    if (*top == NULL) {
        printf("No shipping status to remove.\n");
        return;
    }
    struct Node* temp = *top;
    *top = (*top)->next;
    printf("Shipping status removed: %s\n", temp->status);
    free(temp);
}

// Function to view all shipping status updates
void viewAll(struct Node* top) {
    if (top == NULL) {
        printf("No shipping statuses available.\n");
        return;
    }
    printf("Shipping status updates:\n");
    struct Node* current = top;
    while (current != NULL) {
        printf("- %s\n", current->status);
        current = current->next;
    }
}

// Function to peek at the latest update
void peek(struct Node* top) {
    if (top == NULL) {
        printf("No shipping status available.\n");
        return;
    }
    printf("Latest shipping status: %s\n", top->status);
}

// Function to search for a specific update
void search(struct Node* top, char* status) {
    struct Node* current = top;
    while (current != NULL) {
        if (strcmp(current->status, status) == 0) {
            printf("Found status: %s\n", current->status);
            return;
        }
        current = current->next;
    }
    printf("Status not found.\n");
}

int main() {
    struct Node* top = NULL;
    int choice;
    char status[100];
```

```
do {
    // Display the menu
    printf("\nShipping Status Tracker Menu:\n");
    printf("1. Add a shipping status update (push)\n");
    printf("2. Remove the last update (pop)\n");
    printf("3. View all shipping status updates\n");
    printf("4. Peek at the latest update\n");
    printf("5. Search for a specific update\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the shipping status: ");
            getchar();  // To consume the newline left by scanf
            fgets(status, 100, stdin);
            status[strcspn(status, "\n")] = '\0';  // Remove the trailing newline character
            push(&top, status);
            break;
        case 2:
            pop(&top);
            break;
            case 3:
                viewAll(top);
                break;
            case 4:
                peek(top);
                break;
            case 5:
                printf("Enter the shipping status to search for: ");
                getchar();  // To consume the newline left by scanf
                fgets(status, 100, stdin);
                status[strcspn(status, "\n")] = '\0';  // Remove the trailing newline character
                search(top, status);
                break;
            case 6:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

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

// Define the structure for a linked list node
struct Node {
    char review[255];  // To hold the user review text
    struct Node* next; // Pointer to the next node
};

// Function to create a new node
struct Node* createNode(char* review) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->review, review);
    newNode->next = NULL;
    return newNode;
}

// Function to add a new review (push)
void push(struct Node** top, char* review) {
    struct Node* newNode = createNode(review);
    newNode->next = *top;
    *top = newNode;
```

```c
    printf("Review added: %s\n", review);
}

// Function to remove the last review (pop)
void pop(struct Node** top) {
    if (*top == NULL) {
        printf("No reviews to remove.\n");
        return;
    }
    struct Node* temp = *top;
    *top = (*top)->next;
    printf("Review removed: %s\n", temp->review);
    free(temp);
}

// Function to display all reviews
void displayAll(struct Node* top) {
    if (top == NULL) {
        printf("No reviews available.\n");
        return;
    }
    printf("All Reviews:\n");
    struct Node* current = top;
    while (current != NULL) {
        printf("- %s\n", current->review);
        current = current->next;
    }
}

// Function to peek at the latest review
void peek(struct Node* top) {
    if (top == NULL) {
        printf("No reviews available.\n");
        return;
    }
    printf("Latest review: %s\n", top->review);
}

// Function to search for a specific review
void search(struct Node* top, char* review) {
    struct Node* current = top;
    while (current != NULL) {
        if (strcmp(current->review, review) == 0) {
            printf("Found review: %s\n", current->review);
            return;
        }
        current = current->next;
    }
    printf("Review not found.\n");
}

int main() {
    struct Node* top = NULL;
    int choice;
    char review[255];

    do {
        // Display the menu
        printf("\nUser Review Management Menu:\n");
        printf("1. Add a new review (push)\n");
        printf("2. Remove the last review (pop)\n");
        printf("3. Display all reviews\n");
        printf("4. Peek at the latest review\n");
        printf("5. Search for a specific review\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        // Clear the input buffer by consuming the remaining newline
```

```
while (getchar() != '\n');

switch (choice) {
    case 1:
        printf("Enter the review text: ");
        fgets(review, 255, stdin);
        review[strcspn(review, "\n")] = '\0';  // Remove the newline character
        push(&top, review);
        break;
    case 2:
        pop(&top);
        break;
    case 3:
        displayAll(top);
        break;
    case 4:
        peek(top);
        break;
    case 5:
        printf("Enter the review text to search for: ");
        fgets(review, 255, stdin);
        review[strcspn(review, "\n")] = '\0';  // Remove the newline character
        search(top, review);
```

```
                break;

            case 6:

                printf("Exiting program.\n");

                break;

            default:

                printf("Invalid choice. Please try again.\n");

        }

    } while (choice != 6);


    return 0;

}

12.  #include <stdio.h>

#include <stdlib.h>

#include <string.h>


// Define the structure for a linked list node (Stack Node)

struct Node {

    char notification[255];  // To hold the promotional notification text

    struct Node* next;       // Pointer to the next node

};


// Define the stack structure

struct Stack {
    struct Node* top;  // Pointer to the top node of the stack
};

// Function declarations
void create(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice;
    char item[255];

    // Initialize the stack
    create(&st);

    while (1) {
        printf("\n--- Promotion Notification System ---\n");
        printf("1: Add a new notification (push)\n");
        printf("2: Remove the last notification (pop)\n");
        printf("3: View all notifications\n");
        printf("4: Peek at the latest notification\n");
        printf("5: Search for a specific notification\n");
        printf("6: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();  // To consume the leftover newline character after scanf

        switch (choice) {
            case 1:
                printf("Enter the notification: ");
                scanf("%[^\n]", item);
                push(&st, item);
                break;
            case 2:
                pop(&st);
                break;
            case 3:
                display(&st);
                break;
            case 4:
                peek(&st);
```

```
                break;
            case 5:
                printf("Enter the notification to search for: ");
                scanf("%[^\n]", item);
                search(&st, item);
                break;
            case 6:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}


// Function to initialize the stack
void create(struct Stack *st) {
    st->top = NULL;  // Initialize the stack as empty
}


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

// Function to add a new notification (push)
void push(struct Stack *st, char item[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    strcpy(newNode->notification, item);
    newNode->next = st->top;
    st->top = newNode;
    printf("Notification '%s' added to stack\n", item);
}

// Function to remove the last notification (pop)
void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("No notifications to remove\n");
        return;
    }
    struct Node* temp = st->top;
    st->top = st->top->next;

    printf("Notification '%s' removed from stack\n", temp->notification);

    free(temp);

}


// Function to display all notifications

void display(struct Stack *st) {

    if (isEmpty(st)) {

        printf("No notifications available\n");

        return;

    }

    struct Node* current = st->top;

    printf("Current Notifications:\n");

    int i = 1;

    while (current != NULL) {

        printf("%d: %s\n", i++, current->notification);

        current = current->next;

    }

}


// Function to peek at the latest notification

void peek(struct Stack *st) {

    if (isEmpty(st)) {
        printf("No notifications available\n");
        return;
    }
    printf("Latest notification: %s\n", st->top->notification);
}

// Function to search for a specific notification
void search(struct Stack *st, char item[]) {
    struct Node* current = st->top;
    int position = 1;
    while (current != NULL) {
        if (strcmp(current->notification, item) == 0) {
            printf("Notification '%s' found at position %d\n", item, position);
            return;
        }
        current = current->next;
        position++;
    }
    printf("Notification '%s' not found\n", item);
}
```

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

```
// Define the structure for a linked list node (Stack Node)
struct Node {
    char product[255];  // To hold the product name
    struct Node* next;  // Pointer to the next node
};

// Define the stack structure
struct Stack {
    struct Node* top;  // Pointer to the top node of the stack
};

// Function declarations
void create(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
```

```
int choice;
char item[255];


// Initialize the stack
create(&st);


while (1) {
    printf("\n--- Product Viewing History ---\n");
    printf("1: Add a product to viewing history (push)\n");
    printf("2: Remove the last viewed product (pop)\n");
    printf("3: View all viewed products\n");
    printf("4: Peek at the most recent product viewed\n");
    printf("5: Search for a specific product\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    getchar();  // To consume the leftover newline character after scanf


    switch (choice) {
        case 1:
            printf("Enter the product name: ");
            scanf("%[^\n]", item);
            push(&st, item);
```

```
                break;
            case 2:
                pop(&st);
                break;
            case 3:
                display(&st);
                break;
            case 4:
                peek(&st);
                break;
            case 5:
                printf("Enter the product to search for: ");
                scanf("%[^\n]", item);
                search(&st, item);
                break;
            case 6:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
```

```
}

// Function to initialize the stack
void create(struct Stack *st) {
    st->top = NULL;  // Initialize the stack as empty
}

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

// Function to add a product to the viewing history (push)
void push(struct Stack *st, char item[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    strcpy(newNode->product, item);
    newNode->next = st->top;
    st->top = newNode;
    printf("Product '%s' added to viewing history\n", item);
```

```
}


// Function to remove the last viewed product (pop)
void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("No products in the viewing history to remove\n");
        return;
    }
    struct Node* temp = st->top;
    st->top = st->top->next;
    printf("Product '%s' removed from viewing history\n", temp->product);
    free(temp);
}


// Function to display all viewed products
void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("No products in the viewing history\n");
        return;
    }
    struct Node* current = st->top;
    printf("Viewed Products:\n");
    int i = 1;
    while (current != NULL) {
        printf("%d: %s\n", i++, current->product);
        current = current->next;
    }
}

// Function to peek at the most recent product viewed
void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("No products in the viewing history\n");
        return;
    }
    printf("Most recent product viewed: %s\n", st->top->product);
}

// Function to search for a specific product in the viewing history
void search(struct Stack *st, char item[]) {
    struct Node* current = st->top;
    int position = 1;
    while (current != NULL) {
        if (strcmp(current->product, item) == 0) {
            printf("Product '%s' found at position %d\n", item, position);
            return;
        }
        current = current->next;
        position++;
    }
    printf("Product '%s' not found in the viewing history\n", item);
}
```

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

// Define the structure for a linked list node (Stack Node)
struct Node {
    char item[255];  // To hold the item name
    struct Node* next;  // Pointer to the next node
};

// Define the stack structure
struct Stack {
    struct Node* top;  // Pointer to the top node of the stack
};

// Function declarations
void create(struct Stack *st);
```

```
int isEmpty(struct Stack *st);
void push(struct Stack *st, char item[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char item[]);

int main() {
    struct Stack st;
    int choice;
    char item[255];

    // Initialize the stack
    create(&st);

    while (1) {
        printf("\n--- Cart Item Management ---\n");
        printf("1: Add an item to the cart (push)\n");
        printf("2: Remove the last item (pop)\n");
        printf("3: View all cart items\n");
        printf("4: Peek at the last added item\n");
        printf("5: Search for a specific item in the cart\n");
        printf("6: Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
getchar();  // To consume the leftover newline character after scanf

switch (choice) {
    case 1:
        printf("Enter the item name: ");
        scanf("%[^\n]", item);
        push(&st, item);
        break;
    case 2:
        pop(&st);
        break;
    case 3:
        display(&st);
        break;
    case 4:
        peek(&st);
        break;
    case 5:
        printf("Enter the item to search for: ");
        scanf("%[^\n]", item);
        search(&st, item);
```

```
                break;
            case 6:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}

// Function to initialize the stack
void create(struct Stack *st) {
    st->top = NULL;  // Initialize the stack as empty
}

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

// Function to add an item to the cart (push)
void push(struct Stack *st, char item[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    if (newNode == NULL) {

        printf("Memory allocation failed\n");

        return;

    }

    strcpy(newNode->item, item);

    newNode->next = st->top;

    st->top = newNode;

    printf("Item '%s' added to cart\n", item);

}


// Function to remove the last item from the cart (pop)

void pop(struct Stack *st) {

    if (isEmpty(st)) {

        printf("No items in the cart to remove\n");

        return;

    }

    struct Node* temp = st->top;

    st->top = st->top->next;

    printf("Item '%s' removed from cart\n", temp->item);

    free(temp);

}
```

```
}


// Function to display all cart items
void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("No items in the cart\n");
        return;
    }
    struct Node* current = st->top;
    printf("Items in the cart:\n");
    int i = 1;
    while (current != NULL) {
        printf("%d: %s\n", i++, current->item);
        current = current->next;
    }
}


// Function to peek at the last added item in the cart
void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("No items in the cart\n");
        return;
    }
    printf("Last added item: %s\n", st->top->item);
}
```

```
// Function to search for a specific item in the cart
void search(struct Stack *st, char item[]) {
    struct Node* current = st->top;
    int position = 1;
    while (current != NULL) {
        if (strcmp(current->item, item) == 0) {
            printf("Item '%s' found at position %d\n", item, position);
            return;
        }
        current = current->next;
        position++;
    }
    printf("Item '%s' not found in the cart\n", item);
}
```

15.

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

// Define the structure for a linked list node (Stack Node)
struct Node {
    char paymentDetails[255];  // To hold the payment details
    struct Node* next;  // Pointer to the next node
```

```
};

// Define the stack structure
struct Stack {
    struct Node* top;  // Pointer to the top node of the stack
};

// Function declarations
void create(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, char details[]);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, char details[]);

int main() {
    struct Stack st;
    int choice;
    char details[255];

    // Initialize the stack
    create(&st);
```

```
while (1) {
    printf("\n--- Payment History ---\n");
    printf("1: Add a new payment record (push)\n");
    printf("2: Remove the last payment record (pop)\n");
    printf("3: View all payment records\n");
    printf("4: Peek at the latest payment record\n");
    printf("5: Search for a specific payment record\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    getchar();  // To consume the leftover newline character after scanf

    switch (choice) {
        case 1:
            printf("Enter payment details: ");
            scanf("%[^\n]", details);
            push(&st, details);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
                display(&st);
                break;
            case 4:
                peek(&st);
                break;
            case 5:
                printf("Enter the payment details to search for: ");
                scanf("%[^\n]", details);
                search(&st, details);
                break;
            case 6:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}

// Function to initialize the stack
void create(struct Stack *st) {
    st->top = NULL;  // Initialize the stack as empty
```

```
}

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

// Function to add a payment record (push)
void push(struct Stack *st, char details[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    strcpy(newNode->paymentDetails, details);
    newNode->next = st->top;
    st->top = newNode;
    printf("Payment record '%s' added\n", details);
}

// Function to remove the last payment record (pop)
void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("No payment records to remove\n");
        return;
    }
    struct Node* temp = st->top;
    st->top = st->top->next;
    printf("Payment record '%s' removed\n", temp->paymentDetails);
    free(temp);
}

// Function to display all payment records
void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("No payment records\n");
        return;
    }
    struct Node* current = st->top;
    printf("Payment Records:\n");
    int i = 1;
    while (current != NULL) {
        printf("%d: %s\n", i++, current->paymentDetails);
        current = current->next;
    }
}
```

```
// Function to peek at the latest payment record
void peek(struct Stack *st) {
    if (isEmpty(st)) {
        printf("No payment records\n");
        return;
    }
    printf("Latest payment record: %s\n", st->top->paymentDetails);
}

// Function to search for a specific payment record
void search(struct Stack *st, char details[]) {
    struct Node* current = st->top;
    int position = 1;
    while (current != NULL) {
        if (strcmp(current->paymentDetails, details) == 0) {
            printf("Payment record '%s' found at position %d\n", details, position);
            return;
        }
        current = current->next;
        position++;
    }
    printf("Payment record '%s' not found\n", details);
}
```