STACK
==========

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

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

};

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

    display(st);

    return 0;
}
void create(struct Stack *st){
    printf("enter the size");
    scanf("%d",&st->size);
    st->top=-1;
    st->s=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");
    }
}
```

ALL FUNCTIONS

```
==============================

#include <stdlib.h>
#include <stdio.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 isTop(struct Stack);
void 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("poped value=%d\n",popedvalue);
    display(st);
    peek(&st,2);
    return 0;
}

void create(struct Stack *st){
    printf("enter the size");
    scanf("%d",&st->size);
    st->top=-1;
    st->s=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 isTop(struct Stack st){
    if(!isEmpty){
        return st.s[st.top];
    }
    return -1;
}
void peek(struct Stack *st,int n)
{
    if(st->top!=-1)
    {
        printf("value at position %d is %d\n",n,st->s[n]);
    }
}
```

SET OF PROBLEMS

==========

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

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

```
#include <string.h>

#define MAX_PATHS 100
#define MAX_PATH_LENGTH 100

// Stack structure to hold flight paths
typedef struct {
    char stack[MAX_PATHS][MAX_PATH_LENGTH];
    int top;
} FlightPathStack;

// Function declarations
void push(FlightPathStack* stack, const char* path);
void pop(FlightPathStack* stack);
void displayStack(FlightPathStack* stack);
void peek(FlightPathStack* stack);
int searchPath(FlightPathStack* stack, const char* path);

int main() {
    FlightPathStack stack = { .top = -1 }; // Initialize stack with no elements
    char path[MAX_PATH_LENGTH];
    int choice;

    while (1) {
        // Display the menu
        printf("\nFlight Path Logging System\n");
        printf("1: Add a new path (push)\n");
        printf("2: Undo the last path (pop)\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: ");

        // Read the choice from the user
        if (scanf("%d", &choice) != 1) {
            printf("Invalid input! Please enter a valid option.\n");
            // Clear the input buffer
            while (getchar() != '\n');
            continue;
        }

        switch (choice) {
            case 1:
                // Add a new path
                printf("Enter the flight path: ");
                scanf(" %99[^\n]", path);  // Read string with spaces
                push(&stack, path);
                break;
            case 2:
                // Undo the last path
                pop(&stack);
                break;
            case 3:
                // Display the current stack
```

```
            displayStack(&stack);
            break;
        case 4:
            // Peek at the top path
            peek(&stack);
            break;
        case 5:
            // Search for a specific path
            printf("Enter the path to search: ");
            scanf(" %99[^\n]", path);
            if (searchPath(&stack, path)) {
                printf("Path found in the stack.\n");
            } else {
                printf("Path not found in the stack.\n");
            }
            break;
        case 6:
            // Exit the program
            printf("Exiting the system.\n");
            return 0;
        default:
            printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}

// Push a path onto the stack
void push(FlightPathStack* stack, const char* path) {
  if (stack->top < MAX_PATHS - 1) {
    stack->top++;
    strcpy(stack->stack[stack->top], path);
    printf("Path added to the stack.\n");
  } else {
    printf("Stack is full! Cannot add more paths.\n");
  }
}

// Pop a path from the stack
void pop(FlightPathStack* stack) {
  if (stack->top >= 0) {
    printf("Undoing the last path: %s\n", stack->stack[stack->top]);
    stack->top--;
  } else {
    printf("Stack is empty! No paths to undo.\n");
  }
}

// Display the current flight path stack
void displayStack(FlightPathStack* stack) {
  if (stack->top >= 0) {
    printf("Current flight path stack:\n");
    for (int i = stack->top; i >= 0; i--) {
        printf("%d: %s\n", stack->top - i + 1, stack->stack[i]);
```

```
        }
    } else {
        printf("The stack is empty!\n");
    }
}

// Peek at the top path in the stack
void peek(FlightPathStack* stack) {
    if (stack->top >= 0) {
        printf("Top path: %s\n", stack->stack[stack->top]);
    } else {
        printf("Stack is empty! No top path to view.\n");
    }
}

// Search for a specific path in the stack
int searchPath(FlightPathStack* stack, const char* path) {
    for (int i = 0; i <= stack->top; i++) {
        if (strcmp(stack->stack[i], path) == 0) {
            return 1; // Path found
        }
    }
    return 0; // Path not found
}
```

2.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

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

#define MAX_SIZE 100

// Define a structure for managing the satellite deployments
typedef struct {
    char deployments[MAX_SIZE][100];  // Stack to hold deployment names
    int top;  // Index of the top of the stack
} Stack;

// Function to initialize the stack
void initStack(Stack *stack) {
    stack->top = -1;  // Stack is empty initially
}

// Function to check if the stack is full
int isFull(Stack *stack) {
    return stack->top == MAX_SIZE - 1;
}
```

```
// Function to check if the stack is empty
int isEmpty(Stack *stack) {
    return stack->top == -1;
}

// Function to push a new deployment onto the stack
void push(Stack *stack) {
    if (isFull(stack)) {
        printf("Error: Stack is full. Cannot push new deployment.\n");
    } else {
        char deployment[100];
        printf("Enter the satellite deployment name: ");
        scanf("%s", deployment);
        stack->top++;
        strcpy(stack->deployments[stack->top], deployment);
        printf("Deployment ’%s’ added to the stack.\n", deployment);
    }
}

// Function to pop the last deployment from the stack
void pop(Stack *stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack is empty. No deployment to pop.\n");
    } else {
        printf("Deployment ’%s’ popped from the stack.\n", stack->deployments[stack->top]);
        stack->top--;
    }
}

// Function to view the deployment sequence (print all deployments in the stack)
void viewDeployments(Stack *stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack is empty. No deployments to view.\n");
    } else {
        printf("Deployment Sequence:\n");
        for (int i = stack->top; i >= 0; i--) {
            printf("%d. %s\n", stack->top - i + 1, stack->deployments[i]);
        }
    }
}

// Function to peek at the latest deployment without removing it
void peek(Stack *stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack is empty. No deployment to peek at.\n");
    } else {
        printf("Latest deployment: %s\n", stack->deployments[stack->top]);
    }
}

// Function to search for a specific deployment in the stack
void search(Stack *stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack is empty. No deployments to search.\n");
    } else {
```

```
        char deployment[100];
        printf("Enter the satellite deployment name to search for: ");
        scanf("%s", deployment);
        int found = 0;
        for (int i = 0; i <= stack->top; i++) {
            if (strcmp(stack->deployments[i], deployment) == 0) {
                printf("Deployment '%s' found at position %d.\n", deployment, i + 1);
                found = 1;
                break;
            }
        }
        if (!found) {
            printf("Deployment '%s' not found in the sequence.\n", deployment);
        }
    }
}

// Main function with switch-case menu
int main() {
    Stack stack;
    initStack(&stack);

    int choice;
    do {
        printf("\nSatellite Deployment Management\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:
                push(&stack);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                viewDeployments(&stack);
                break;
            case 4:
                peek(&stack);
                break;
            case 5:
                search(&stack);
                break;
            case 6:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
```

```
        }
    } while (choice != 6);

    return 0;
}
```

3.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

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

#define MAX_SIZE 10
#define ITEM_LENGTH 100

// Define a stack to hold the checklist items
char checklist[MAX_SIZE][ITEM_LENGTH];
int top = -1;

// Function to add a checklist item
void addItem() {
    if (top < MAX_SIZE - 1) {
        top++;
        printf("Enter checklist item: ");
        scanf(" %99[^\n]", checklist[top]);
        printf("Item added to the checklist.\n");
    } else {
        printf("Checklist is full. Cannot add more items.\n");
    }
}

// Function to remove the last item
void removeItem() {
    if (top >= 0) {
        printf("Removed item: %s\n", checklist[top]);
        top--;
    } else {
        printf("Checklist is empty. Nothing to remove.\n");
    }
}

// Function to display the current checklist
void displayChecklist() {
    if (top >= 0) {
        printf("Current checklist:\n");
        for (int i = 0; i <= top; i++) {
            printf("%d. %s\n", i + 1, checklist[i]);
        }
    } else {
```

```
        printf("Checklist is empty.\n");
    }
}

// Function to peek at the top checklist item
void peekItem() {
    if (top >= 0) {
        printf("Top checklist item: %s\n", checklist[top]);
    } else {
        printf("Checklist is empty.\n");
    }
}

// Function to search for a specific checklist item
void searchItem() {
    char search[ITEM_LENGTH];
    printf("Enter checklist item to search for: ");
    scanf(" %99[^\n]", search);

    for (int i = 0; i <= top; i++) {
        if (strcmp(checklist[i], search) == 0) {
            printf("Item found at position %d: %s\n", i + 1, checklist[i]);
            return;
        }
    }
    printf("Item not found in the checklist.\n");
}

int main() {
    int choice;

    do {
        // Display menu
        printf("\nRocket Launch Checklist Menu:\n");
        printf("1. Add a checklist item\n");
        printf("2. Remove the last item\n");
        printf("3. Display 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: ");

        // Read choice from the user
        scanf("%d", &choice);

        // Switch-case for menu options
        switch (choice) {
            case 1:
                addItem();
                break;
            case 2:
                removeItem();
                break;
            case 3:
                displayChecklist();
```

```
            break;
        case 4:
            peekItem();
            break;
        case 5:
            searchItem();
            break;
        case 6:
            printf("Exiting program...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }

  } while (choice != 6);

  return 0;
}
```

4.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

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

#define MAX_SIZE 10
#define MAX_DATA_LENGTH 100

// Stack structure to hold telemetry data
typedef struct {
  char data[MAX_SIZE][MAX_DATA_LENGTH]; // array to hold the telemetry data strings
  int top; // index of the top of the stack
} Stack;

// Function prototypes
void initStack(Stack *stack);
int isFull(Stack *stack);
int isEmpty(Stack *stack);
void push(Stack *stack, char *data);
void pop(Stack *stack);
void peek(Stack *stack);
void viewData(Stack *stack);
void searchData(Stack *stack, char *searchTerm);

// Main function with menu interface
int main() {
  Stack stack;
  initStack(&stack);
```

```
int choice;
char input[MAX_DATA_LENGTH];

while (1) {
    // Menu display
    printf("\nTelemetry 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);
    while(getchar() != '\n');  // Clear the input buffer

    switch (choice) {
        case 1: // Push new telemetry data
            if (isFull(&stack)) {
                printf("Error: Stack is full. Cannot push more data.\n");
            } else {
                printf("Enter telemetry data (max %d characters): ", MAX_DATA_LENGTH - 1);
                // Using scanf to capture the data string
                scanf("%99[^\n]", input);  // Limit input size to prevent overflow
                while(getchar() != '\n');  // Clear the input buffer
                push(&stack, input);
            }
            break;

        case 2: // Pop the last data entry
            pop(&stack);
            break;

        case 3: // View the stored telemetry data
            viewData(&stack);
            break;

        case 4: // Peek at the most recent data entry
            peek(&stack);
            break;

        case 5: // Search for specific telemetry data
            printf("Enter search term: ");
            scanf("%99[^\n]", input);
            while(getchar() != '\n');  // Clear the input buffer
            searchData(&stack, input);
            break;

        case 6: // Exit
            printf("Exiting program.\n");
            return 0;

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

```
        }
    }

    return 0;
}

// Initialize the stack
void initStack(Stack *stack) {
    stack->top = -1; // Set the stack to be empty
}

// Check if the stack is full
int isFull(Stack *stack) {
    return stack->top == MAX_SIZE - 1;
}

// Check if the stack is empty
int isEmpty(Stack *stack) {
    return stack->top == -1;
}

// Push new telemetry data onto the stack
void push(Stack *stack, char *data) {
    if (isFull(stack)) {
        printf("Stack is full, cannot push data.\n");
    } else {
        stack->top++;
        strncpy(stack->data[stack->top], data, MAX_DATA_LENGTH);
        printf("Data pushed: %s\n", data);
    }
}

// Pop the last telemetry data entry
void pop(Stack *stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack is empty. No data to pop.\n");
    } else {
        printf("Data popped: %s\n", stack->data[stack->top]);
        stack->top--;
    }
}

// Peek at the most recent data entry
void peek(Stack *stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack is empty. No data to peek.\n");
    } else {
        printf("Most recent data: %s\n", stack->data[stack->top]);
    }
}

// View all the stored telemetry data
void viewData(Stack *stack) {
    if (isEmpty(stack)) {
        printf("No data in the stack.\n");
```

```
    } else {
        printf("Stored telemetry data:\n");
        for (int i = stack->top; i >= 0; i--) {
            printf("%d: %s\n", i + 1, stack->data[i]);
        }
    }
}

// Search for specific telemetry data
void searchData(Stack *stack, char *searchTerm) {
    int found = 0;
    for (int i = stack->top; i >= 0; i--) {
        if (strstr(stack->data[i], searchTerm) != NULL) {
            printf("Found matching data: %s (Index: %d)\n", stack->data[i], i);
            found = 1;
        }
    }
    if (!found) {
        printf("No matching data found.\n");
    }
}
```

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

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

#define MAX_TASKS 10
#define MAX_TASK_LENGTH 100

// Stack to hold tasks
char tasks[MAX_TASKS][MAX_TASK_LENGTH];
int top = -1; // Points to the top of the stack

// Function to add a task to the stack (push)
void addTask(char task[]) {
    if (top == MAX_TASKS - 1) {
        printf("Stack is full. Cannot add more tasks.\n");
    } else {
        top++;
        strcpy(tasks[top], task);
        printf("Task '%s' added.\n", task);
    }
}

// Function to mark the last task as completed (pop)
void markTaskCompleted() {
    if (top == -1) {
```

```
        printf("No tasks to complete.\n");
    } else {
        printf("Task '%s' completed.\n", tasks[top]);
        top--;
    }
}

// Function to list all pending tasks
void listTasks() {
    if (top == -1) {
        printf("No tasks pending.\n");
    } else {
        printf("Pending tasks:\n");
        for (int i = 0; i <= top; i++) {
            printf("%d. %s\n", i + 1, tasks[i]);
        }
    }
}

// Function to peek at the most recent task
void peekTask() {
    if (top == -1) {
        printf("No tasks to view.\n");
    } else {
        printf("Most recent task: %s\n", tasks[top]);
    }
}

// Function to search for a specific task
void searchTask(char task[]) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(tasks[i], task) == 0) {
            printf("Task '%s' found at position %d.\n", task, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Task '%s' not found.\n", task);
    }
}

int main() {
    int choice;
    char task[MAX_TASK_LENGTH];

    do {
        // Display the menu
        printf("\nSpace 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");

        // Get user choice
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();  // To consume the newline character after the choice input

        switch (choice) {
            case 1: // Add a task
                printf("Enter task description: ");
                // We use scanf to get a string without spaces
                scanf("%s", task);
                addTask(task);
                break;

            case 2: // Mark the last task as completed
                markTaskCompleted();
                break;

            case 3: // List all pending tasks
                listTasks();
                break;

            case 4: // Peek at the most recent task
                peekTask();
                break;

            case 5: // Search for a specific task
                printf("Enter task description to search: ");
                // Again, scanf for a single word task
                scanf("%s", task);
                searchTask(task);
                break;

            case 6: // Exit
                printf("Exiting program...\n");
                break;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

6.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

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

#define MAX_SIZE 10

// Stack structure
typedef struct {
    int items[MAX_SIZE];
    int top;
} Stack;

// Function declarations
void initializeStack(Stack *stack);
int isFull(Stack *stack);
int isEmpty(Stack *stack);
void push(Stack *stack, int step);
int pop(Stack *stack);
void display(Stack *stack);
int peek(Stack *stack);
int search(Stack *stack, int step);
void printMenu();

int main() {
    Stack countdownStack;
    int choice, step, index;

    // Initialize stack
    initializeStack(&countdownStack);

    while (1) {
        // Print menu
        printMenu();

        // Get user choice
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:  // Add a countdown step
                if (isFull(&countdownStack)) {
                    printf("Stack is full. Cannot add more countdown steps.\n");
                } else {
                    printf("Enter countdown step number: ");
                    scanf("%d", &step);
                    push(&countdownStack, step);
                    printf("Step %d added to countdown.\n", step);
                }
                break;

            case 2:  // Remove the last countdown step
                if (isEmpty(&countdownStack)) {
                    printf("Stack is empty. No steps to remove.\n");
                } else {
                    step = pop(&countdownStack);
```

```
                printf("Removed step %d from countdown.\n", step);
            }
            break;

        case 3:  // Display the current countdown
            display(&countdownStack);
            break;

        case 4:  // Peek at the next countdown step
            step = peek(&countdownStack);
            if (step == -1) {
                printf("No steps left in the countdown.\n");
            } else {
                printf("Next countdown step: %d\n", step);
            }
            break;

        case 5:  // Search for a specific countdown step
            printf("Enter the countdown step number to search for: ");
            scanf("%d", &step);
            index = search(&countdownStack, step);
            if (index == -1) {
                printf("Step %d not found in countdown.\n", step);
            } else {
                printf("Step %d found at position %d.\n", step, index);
            }
            break;

        case 6:  // Exit
            printf("Exiting the program.\n");
            exit(0);

        default:
            printf("Invalid choice. Please try again.\n");
      }
   }
   return 0;
}

// Function to initialize the stack
void initializeStack(Stack *stack) {
   stack->top = -1;
}

// Function to check if stack is full
int isFull(Stack *stack) {
   return stack->top == MAX_SIZE - 1;
}

// Function to check if stack is empty
int isEmpty(Stack *stack) {
   return stack->top == -1;
}

// Function to push an element onto the stack
```

```
void push(Stack *stack, int step) {
    if (isFull(stack)) {
        printf("Stack is full, cannot add more steps.\n");
    } else {
        stack->items[++(stack->top)] = step;
    }
}

// Function to pop an element from the stack
int pop(Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty, nothing to pop.\n");
        return -1;
    } else {
        return stack->items[(stack->top)--];
    }
}

// Function to display the current countdown
void display(Stack *stack) {
    if (isEmpty(stack)) {
        printf("The countdown stack is empty.\n");
    } else {
        printf("Current countdown steps: ");
        for (int i = stack->top; i >= 0; i--) {
            printf("%d ", stack->items[i]);
        }
        printf("\n");
    }
}

// Function to peek at the next countdown step
int peek(Stack *stack) {
    if (isEmpty(stack)) {
        return -1;
    } else {
        return stack->items[stack->top];
    }
}

// Function to search for a specific countdown step
int search(Stack *stack, int step) {
    for (int i = 0; i <= stack->top; i++) {
        if (stack->items[i] == step) {
            return i;  // return the index where the step is found
        }
    }
    return -1;  // return -1 if the step is not found
}

// Function to print the menu options
void printMenu() {
    printf("\n=== Rocket Launch Countdown Management ===\n");
    printf("1: Add a countdown step (push)\n");
    printf("2: Remove the last countdown 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");
}
```

7.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

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

#define MAX_LOGS 100
#define LOG_LENGTH 256

// Define the stack structure
struct Stack {
    char logs[MAX_LOGS][LOG_LENGTH]; // Array to store logs
    int top; // Top of the stack
};

// Function to initialize the stack
void initStack(struct Stack *s) {
    s->top = -1;
}

// Function to check if the stack is full
int isFull(struct Stack *s) {
    return s->top == MAX_LOGS - 1;
}

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

// Function to push a log onto the stack
void push(struct Stack *s, const char *log) {
    if (isFull(s)) {
        printf("Error: Stack is full. Cannot add more logs.\n");
    } else {
        s->top++;
        strncpy(s->logs[s->top], log, LOG_LENGTH);
        printf("Log added successfully.\n");
    }
}

// Function to pop a log from the stack
void pop(struct Stack *s) {
```

```
    if (isEmpty(s)) {
        printf("Error: No logs to remove. The stack is empty.\n");
    } else {
        printf("Removed log: %s\n", s->logs[s->top]);
        s->top--;
    }
}

// Function to display all logs
void viewLogs(struct Stack *s) {
    if (isEmpty(s)) {
        printf("No logs to display. The stack is empty.\n");
    } else {
        printf("Maintenance Logs:\n");
        for (int i = s->top; i >= 0; i--) {
            printf("%d: %s\n", i + 1, s->logs[i]);
        }
    }
}

// Function to peek at the latest log
void peek(struct Stack *s) {
    if (isEmpty(s)) {
        printf("Error: No logs to display. The stack is empty.\n");
    } else {
        printf("Latest log: %s\n", s->logs[s->top]);
    }
}

// Function to search for a specific log
void searchLog(struct Stack *s, const char *searchTerm) {
    if (isEmpty(s)) {
        printf("No logs to search. The stack is empty.\n");
    } else {
        int found = 0;
        for (int i = s->top; i >= 0; i--) {
            if (strstr(s->logs[i], searchTerm) != NULL) {
                printf("Found log: %d: %s\n", i + 1, s->logs[i]);
                found = 1;
            }
        }
        if (!found) {
            printf("No logs found matching '%s'.\n", searchTerm);
        }
    }
}

int main() {
    struct Stack maintenanceLogs;
    initStack(&maintenanceLogs);

    int choice;
    char log[LOG_LENGTH];
    char searchTerm[LOG_LENGTH];
```

```
    while (1) {
        // Display the menu
        printf("\nAircraft Maintenance Log Menu:\n");
        printf("1. Add a new log\n");
        printf("2. Remove the last log\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: ");
                // Read a string (no spaces) for the log
                scanf(" %[^\n]", log); // Read the entire line until a newline
                push(&maintenanceLogs, log);
                break;
            case 2:
                pop(&maintenanceLogs);
                break;
            case 3:
                viewLogs(&maintenanceLogs);
                break;
            case 4:
                peek(&maintenanceLogs);
                break;
            case 5:
                printf("Enter the search term: ");
                // Read a string for the search term
                scanf(" %[^\n]", searchTerm); // Read the entire line until a newline
                searchLog(&maintenanceLogs, searchTerm);
                break;
            case 6:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

8.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

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

```
#include <string.h>

#define MAX_STEPS 10
#define MAX_STEP_LENGTH 100

// Stack structure to hold the procedure steps
typedef struct {
    char steps[MAX_STEPS][MAX_STEP_LENGTH]; // Array to hold the steps
    int top;  // Index of the top of the stack
} Stack;

// Initialize the stack
void initializeStack(Stack* stack) {
    stack->top = -1;  // Stack is empty initially
}

// Check if the stack is full
int isFull(Stack* stack) {
    return stack->top == MAX_STEPS - 1;
}

// Check if the stack is empty
int isEmpty(Stack* stack) {
    return stack->top == -1;
}

// Push a new step onto the stack
void push(Stack* stack, const char* step) {
    if (isFull(stack)) {
        printf("Stack is full. Cannot add more steps.\n");
    } else {
        stack->top++;
        strncpy(stack->steps[stack->top], step, MAX_STEP_LENGTH - 1);
        stack->steps[stack->top][MAX_STEP_LENGTH - 1] = '\0';  // Null-terminate the string
        printf("Step added: %s\n", stack->steps[stack->top]);
    }
}

// Pop the last step from the stack
void pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. No steps to remove.\n");
    } else {
        printf("Removing step: %s\n", stack->steps[stack->top]);
        stack->top--;
    }
}

// Display the procedure steps
void display(Stack* stack) {
    if (isEmpty(stack)) {
        printf("No steps to display. The stack is empty.\n");
    } else {
        printf("Docking procedure steps:\n");
        for (int i = 0; i <= stack->top; i++) {
```

```
            printf("%d. %s\n", i + 1, stack->steps[i]);
        }
    }
}

// Peek at the next step in the procedure
void peek(Stack* stack) {
    if (isEmpty(stack)) {
        printf("No steps available to peek at.\n");
    } else {
        printf("Next step: %s\n", stack->steps[stack->top]);
    }
}

// Search for a specific step in the procedure
void search(Stack* stack, const char* step) {
    int found = 0;
    for (int i = 0; i <= stack->top; i++) {
        if (strncmp(stack->steps[i], step, MAX_STEP_LENGTH) == 0) {
            printf("Step found at position %d: %s\n", i + 1, stack->steps[i]);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Step not found.\n");
    }
}

int main() {
    Stack stack;
    initializeStack(&stack);

    int choice;
    char step[MAX_STEP_LENGTH];

    do {
        printf("\nSpacecraft Docking Procedure Menu:\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\n");
        printf("5. Search for a specific step\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        // Clear the newline character left in the buffer by scanf
        while (getchar() != '\n');

        switch (choice) {
            case 1:
                printf("Enter the step to push: ");
                scanf("%99[^\n]", step);  // Read a line of text, allowing spaces in between
                push(&stack, step);
```

```
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(&stack);
            break;
        case 4:
            peek(&stack);
            break;
        case 5:
            printf("Enter the step to search for: ");
            scanf("%99[^\n]", step);  // Read a line of text
            search(&stack, step);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice, please try again.\n");
      }
   } while (choice != 6);

   return 0;
}
```

9.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

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

#define MAX_COMMAND_LENGTH 100
#define MAX_STACK_SIZE 10

// Stack structure to hold command history
typedef struct {
   char commands[MAX_STACK_SIZE][MAX_COMMAND_LENGTH];
   int top;
} CommandStack;

// Function to initialize the stack
void initializeStack(CommandStack *stack) {
   stack->top = -1;
}

// Function to check if the stack is full
int isFull(CommandStack *stack) {
```

```
    return stack->top == MAX_STACK_SIZE - 1;
}

// Function to check if the stack is empty
int isEmpty(CommandStack *stack) {
    return stack->top == -1;
}

// Function to add a command to the stack
void push(CommandStack *stack, const char *command) {
    if (isFull(stack)) {
        printf("Stack is full. Cannot add more commands.\n");
    } else {
        stack->top++;
        strncpy(stack->commands[stack->top], command, MAX_COMMAND_LENGTH);
        stack->commands[stack->top][MAX_COMMAND_LENGTH - 1] = '\0'; // Ensure null termination
        printf("Command added: %s\n", command);
    }
}

// Function to remove the last command from the stack
void pop(CommandStack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. No command to undo.\n");
    } else {
        printf("Undoing command: %s\n", stack->commands[stack->top]);
        stack->top--;
    }
}

// Function to view the command history
void viewHistory(CommandStack *stack) {
    if (isEmpty(stack)) {
        printf("No command history available.\n");
    } else {
        printf("Command History:\n");
        for (int i = 0; i <= stack->top; i++) {
            printf("%d: %s\n", i + 1, stack->commands[i]);
        }
    }
}

// Function to peek at the most recent command
void peek(CommandStack *stack) {
    if (isEmpty(stack)) {
        printf("No commands to peek.\n");
    } else {
        printf("Most recent command: %s\n", stack->commands[stack->top]);
    }
}

// Function to search for a specific command
void searchCommand(CommandStack *stack, const char *command) {
    int found = 0;
    for (int i = 0; i <= stack->top; i++) {
```

```
        if (strncmp(stack->commands[i], command, MAX_COMMAND_LENGTH) == 0) {
            printf("Command found at position %d: %s\n", i + 1, stack->commands[i]);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Command not found in history.\n");
    }
}

// Main function to handle user input
int main() {
    CommandStack stack;
    initializeStack(&stack);

    int choice;
    char command[MAX_COMMAND_LENGTH];

    do {
        printf("\nMission Control Command History\n");
        printf("1: Add a command (push)\n");
        printf("2: Undo the last command (pop)\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 command to add: ");
                scanf(" %[^\n]%*c", command);  // Read a full line of input (up to MAX_COMMAND_LENGTH)
                push(&stack, command);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                viewHistory(&stack);
                break;
            case 4:
                peek(&stack);
                break;
            case 5:
                printf("Enter command to search for: ");
                scanf(" %[^\n]%*c", command);  // Read a full line of input (up to MAX_COMMAND_LENGTH)
                searchCommand(&stack, command);
                break;
            case 6:
                printf("Exiting program...\n");
                break;
            default:
                printf("Invalid choice, please try again.\n");
```

```
        }
    } while (choice != 6);

    return 0;
}
```

10.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

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

#define MAX_EVENTS 10
#define EVENT_NAME_LENGTH 100

// Define the structure for an event
typedef struct {
    char name[EVENT_NAME_LENGTH];
} Event;

// Stack to hold events
Event eventStack[MAX_EVENTS];
int top = -1;  // Initialize stack as empty

// Function to push a new event onto the stack
void pushEvent() {
    if (top == MAX_EVENTS - 1) {
        printf("Error: Stack is full, cannot add more events.\n");
    } else {
        top++;
        printf("Enter the name of the event: ");
        scanf(" %[^\n]", eventStack[top].name);  // Read event name, space allowed
        printf("Event '%s' added successfully.\n", eventStack[top].name);
    }
}

// Function to pop the last event from the stack
void popEvent() {
    if (top == -1) {
        printf("Error: No events to pop.\n");
    } else {
        printf("Event '%s' removed from the stack.\n", eventStack[top].name);
        top--;
    }
}

// Function to display all events in the stack
void displayEvents() {
    if (top == -1) {
        printf("No events to display.\n");
```

```
    } else {
        printf("Events in the stack:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d: %s\n", i + 1, eventStack[i].name);
        }
    }
}

// Function to peek at the most recent event
void peekEvent() {
    if (top == -1) {
        printf("No events to peek.\n");
    } else {
        printf("Most recent event: %s\n", eventStack[top].name);
    }
}

// Function to search for a specific event
void searchEvent() {
    char searchName[EVENT_NAME_LENGTH];
    int found = 0;
    printf("Enter the name of the event to search: ");
    scanf(" %[^\n]", searchName);  // Read event name, space allowed

    for (int i = 0; i <= top; i++) {
        if (strcmp(eventStack[i].name, searchName) == 0) {
            printf("Event '%s' found at position %d in the stack.\n", searchName, i + 1);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Event '%s' not found in the stack.\n", searchName);
    }
}

int main() {
    int choice;

    do {
        printf("\nAerospace Simulation Event Stack Menu:\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:
                pushEvent();
                break;
```

```
            case 2:
                popEvent();
                break;
            case 3:
                displayEvents();
                break;
            case 4:
                peekEvent();
                break;
            case 5:
                searchEvent();
                break;
            case 6:
                printf("Exiting the program...\n");
                break;
            default:
                printf("Invalid choice, please try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

11.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

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

#define MAX_SIZE 100
#define MAX_LENGTH 100

// Define the stack structure
typedef struct {
    char maneuvers[MAX_SIZE][MAX_LENGTH];
    int top;
} ManeuverStack;

// Function to initialize the stack
void initStack(ManeuverStack* stack) {
    stack->top = -1;
}

// Function to check if the stack is empty
int isEmpty(ManeuverStack* stack) {
    return stack->top == -1;
}
```

```
// Function to check if the stack is full
int isFull(ManeuverStack* stack) {
    return stack->top == MAX_SIZE - 1;
}

// Function to push a maneuver onto the stack
void push(ManeuverStack* stack, char* maneuver) {
    if (isFull(stack)) {
        printf("Stack is full. Cannot add more maneuvers.\n");
    } else {
        stack->top++;
        strcpy(stack->maneuvers[stack->top], maneuver);
        printf("Maneuver added: %s\n", maneuver);
    }
}

// Function to pop the last maneuver from the stack
void pop(ManeuverStack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. No maneuvers to remove.\n");
    } else {
        printf("Removed maneuver: %s\n", stack->maneuvers[stack->top]);
        stack->top--;
    }
}

// Function to view all maneuvers in the stack
void viewAllManeuvers(ManeuverStack* stack) {
    if (isEmpty(stack)) {
        printf("No maneuvers to display.\n");
    } else {
        printf("Maneuvers in the stack:\n");
        for (int i = stack->top; i >= 0; i--) {
            printf("%d: %s\n", stack->top - i + 1, stack->maneuvers[i]);
        }
    }
}

// Function to peek at the most recent maneuver
void peek(ManeuverStack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. No maneuvers to peek at.\n");
    } else {
        printf("Most recent maneuver: %s\n", stack->maneuvers[stack->top]);
    }
}

// Function to search for a specific maneuver in the stack
void search(ManeuverStack* stack, char* maneuver) {
    int found = 0;
    for (int i = 0; i <= stack->top; i++) {
        if (strcmp(stack->maneuvers[i], maneuver) == 0) {
            printf("Maneuver '%s' found at position %d.\n", maneuver, i + 1);
            found = 1;
            break;
```

```
        }
    }
    if (!found) {
        printf("Maneuver ’%s’ not found in the stack.\n", maneuver);
    }
}

int main() {
    ManeuverStack stack;
    initStack(&stack);
    int choice;
    char maneuver[MAX_LENGTH];

    while (1) {
        // Menu
        printf("\nPilot Training Maneuver Stack Menu:\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);
        getchar(); // To consume the newline character left by scanf()

        switch (choice) {
            case 1:
                printf("Enter the maneuver to add: ");
                scanf("%99[^\n]", maneuver);  // Read a line of input up to 99 characters
                push(&stack, maneuver);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                viewAllManeuvers(&stack);
                break;
            case 4:
                peek(&stack);
                break;
            case 5:
                printf("Enter the maneuver to search for: ");
                scanf("%99[^\n]", maneuver);  // Read a line of input up to 99 characters
                search(&stack, maneuver);
                break;
            case 6:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
```

}

12.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

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

#define MAX_COMMANDS 100
#define MAX_COMMAND_LENGTH 100

// Stack data structure
typedef struct {
    char commands[MAX_COMMANDS][MAX_COMMAND_LENGTH];
    int top;
} CommandStack;

// Function prototypes
void initStack(CommandStack* stack);
int isFull(CommandStack* stack);
int isEmpty(CommandStack* stack);
void pushCommand(CommandStack* stack, const char* command);
void popCommand(CommandStack* stack);
void viewCommands(CommandStack* stack);
void peekCommand(CommandStack* stack);
int searchCommand(CommandStack* stack, const char* command);

int main() {
    CommandStack stack;
    initStack(&stack);

    int choice;
    char command[MAX_COMMAND_LENGTH];

    do {
        printf("\nSatellite Operation Command Menu:\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);

        // Clear the remaining newline character in the buffer
        while (getchar() != '\n'); // Discard the newline character

        switch (choice) {
```

```
            case 1:
                if (isFull(&stack)) {
                    printf("Error: Stack is full. Cannot push a new command.\n");
                } else {
                    printf("Enter the command: ");
                    scanf("%99[^\n]", command);  // Read a line until a newline is encountered
                    pushCommand(&stack, command);
                    printf("Command pushed: %s\n", command);
                }
                break;

            case 2:
                popCommand(&stack);
                break;

            case 3:
                viewCommands(&stack);
                break;

            case 4:
                peekCommand(&stack);
                break;

            case 5:
                printf("Enter the command to search for: ");
                scanf("%99[^\n]", command);  // Read a line until a newline is encountered
                if (searchCommand(&stack, command)) {
                    printf("Command found: %s\n", command);
                } else {
                    printf("Command not found: %s\n", command);
                }
                break;

            case 6:
                printf("Exiting...\n");
                break;

            default:
                printf("Invalid choice, please try again.\n");
        }
    } while (choice != 6);

    return 0;
}

// Function definitions

void initStack(CommandStack* stack) {
    stack->top = -1; // Initialize top to -1 to indicate an empty stack
}

int isFull(CommandStack* stack) {
    return stack->top == MAX_COMMANDS - 1;
}
```

```
int isEmpty(CommandStack* stack) {
    return stack->top == -1;
}

void pushCommand(CommandStack* stack, const char* command) {
    if (!isFull(stack)) {
        stack->top++;
        strcpy(stack->commands[stack->top], command);
    }
}

void popCommand(CommandStack* stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack is empty. Cannot pop a command.\n");
    } else {
        printf("Command popped: %s\n", stack->commands[stack->top]);
        stack->top--;
    }
}

void viewCommands(CommandStack* stack) {
    if (isEmpty(stack)) {
        printf("No commands in the stack.\n");
    } else {
        printf("Commands in the stack:\n");
        for (int i = stack->top; i >= 0; i--) {
            printf("%s\n", stack->commands[i]);
        }
    }
}

void peekCommand(CommandStack* stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack is empty. No command to peek.\n");
    } else {
        printf("Most recent command: %s\n", stack->commands[stack->top]);
    }
}

int searchCommand(CommandStack* stack, const char* command) {
    for (int i = 0; i <= stack->top; i++) {
        if (strcmp(stack->commands[i], command) == 0) {
            return 1; // Command found
        }
    }
    return 0; // Command not found
}
```

13.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

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

#define MAX_STACK_SIZE 10
#define PROCEDURE_LENGTH 100

// Stack structure to hold the procedures
char stack[MAX_STACK_SIZE][PROCEDURE_LENGTH];
int top = -1;

// Function to add a procedure to the stack
void push() {
    if (top == MAX_STACK_SIZE - 1) {
        printf("Stack Overflow! Unable to add more procedures.\n");
    } else {
        top++;
        printf("Enter procedure name: ");
        scanf("%s", stack[top]);
        printf("Procedure '%s' added.\n", stack[top]);
    }
}

// Function to remove the last procedure from the stack
void pop() {
    if (top == -1) {
        printf("Stack Underflow! No procedures to remove.\n");
    } else {
        printf("Procedure '%s' removed.\n", stack[top]);
        top--;
    }
}

// Function to view all procedures in the stack
void viewAll() {
    if (top == -1) {
        printf("No procedures to display.\n");
    } else {
        printf("All procedures in stack:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d: %s\n", i + 1, stack[i]);
        }
    }
}

// Function to peek at the next procedure
void peek() {
    if (top == -1) {
        printf("No procedures in stack to peek.\n");
    } else {
        printf("Next procedure: %s\n", stack[top]);
    }
}
```

```
// Function to search for a specific procedure
void search() {
    char procedure[PROCEDURE_LENGTH];
    printf("Enter the procedure to search for: ");
    scanf("%s", procedure);

    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], procedure) == 0) {
            printf("Procedure '%s' found at position %d.\n", procedure, i + 1);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("Procedure '%s' not found.\n", procedure);
    }
}

// Main menu with switch-case structure for user interaction
int main() {
    int choice;

    do {
        printf("\n--- Emergency Procedures Menu ---\n");
        printf("1: Add a procedure (push)\n");
        printf("2: Remove the last procedure (pop)\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:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                viewAll();
                break;
            case 4:
                peek();
                break;
            case 5:
                search();
                break;
            case 6:
                printf("Exiting program...\n");
                break;
            default:
```

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

    return 0;
}
```

14.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

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

#define MAX 100

// Define a stack structure to hold astronaut activities
struct Stack {
    char activities[MAX][100];  // Array to hold activity descriptions
    int top;                    // Index for the top of the stack
};

// Function prototypes
void initStack(struct Stack* stack);
int isFull(struct Stack* stack);
int isEmpty(struct Stack* stack);
void push(struct Stack* stack, char* activity);
void pop(struct Stack* stack);
void display(struct Stack* stack);
void peek(struct Stack* stack);
int search(struct Stack* stack, char* activity);

int main() {
    struct Stack stack;
    char activity[100];
    int choice;

    // Initialize the stack
    initStack(&stack);

    while (1) {
        // Display menu
        printf("\nAstronaut Activity Log Menu:\n");
        printf("1. Add a new activity (push)\n");
        printf("2. Remove the last activity (pop)\n");
        printf("3. Display the 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:
                // Add a new activity (push)
                if (isFull(&stack)) {
                    printf("Stack is full! Cannot add more activities.\n");
                } else {
                    printf("Enter the activity: ");
                    // We use scanf to take the input and handle a single word activity
                    scanf(" %[^\n]s", activity);  // The " %[^\n]s" format allows to take space-separated input till
newline
                    push(&stack, activity);
                    printf("Activity added: %s\n", activity);
                }
                break;

            case 2:
                // Remove the last activity (pop)
                if (isEmpty(&stack)) {
                    printf("No activities to remove.\n");
                } else {
                    pop(&stack);
                }
                break;

            case 3:
                // Display the activity log
                display(&stack);
                break;

            case 4:
                // Peek at the most recent activity
                peek(&stack);
                break;

            case 5:
                // Search for a specific activity
                printf("Enter the activity to search for: ");
                scanf(" %[^\n]s", activity);  // Using same format specifier to allow space-separated input
                int found = search(&stack, activity);
                if (found != -1) {
                    printf("Activity '%s' found at position %d in the log.\n", activity, found);
                } else {
                    printf("Activity '%s' not found in the log.\n", activity);
                }
                break;

            case 6:
                // Exit
                printf("Exiting the program...\n");
                return 0;

            default:
                printf("Invalid choice! Please try again.\n");
```

```
        }
    }

    return 0;
}

// Initialize stack
void initStack(struct Stack* stack) {
    stack->top = -1;  // Stack is empty initially
}

// Check if the stack is full
int isFull(struct Stack* stack) {
    return stack->top == MAX - 1;
}

// Check if the stack is empty
int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

// Add a new activity (push)
void push(struct Stack* stack, char* activity) {
    if (isFull(stack)) {
        printf("Stack overflow! Cannot add more activities.\n");
    } else {
        stack->top++;
        strcpy(stack->activities[stack->top], activity);
    }
}

// Remove the last activity (pop)
void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No activities to remove.\n");
    } else {
        printf("Removing activity: %s\n", stack->activities[stack->top]);
        stack->top--;
    }
}

// Display all activities
void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No activities logged yet.\n");
    } else {
        printf("Activity Log:\n");
        for (int i = stack->top; i >= 0; i--) {
            printf("%d. %s\n", stack->top - i + 1, stack->activities[i]);
        }
    }
}

// Peek at the most recent activity
void peek(struct Stack* stack) {
```

```
    if (isEmpty(stack)) {
        printf("No activities to peek.\n");
    } else {
        printf("Most recent activity: %s\n", stack->activities[stack->top]);
    }
}

// Search for a specific activity
int search(struct Stack* stack, char* activity) {
    for (int i = stack->top; i >= 0; i--) {
        if (strcmp(stack->activities[i], activity) == 0) {
            return i + 1;  // Return position (1-based index)
        }
    }
    return -1;  // Activity not found
}
```

15.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

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

#define MAX_STACK_SIZE 100

typedef struct {
    float fuelAmount;
    char timestamp[20]; // You can store a timestamp as a string (format: YYYY-MM-DD HH:MM:SS)
} FuelEntry;

typedef struct {
    FuelEntry stack[MAX_STACK_SIZE];
    int top;
} FuelStack;

void initStack(FuelStack *stack) {
    stack->top = -1; // Stack is initially empty
}

int isFull(FuelStack *stack) {
    return stack->top == MAX_STACK_SIZE - 1;
}

int isEmpty(FuelStack *stack) {
    return stack->top == -1;
}

void push(FuelStack *stack, float fuelAmount, const char *timestamp) {
    if (isFull(stack)) {
```

```
        printf("Stack is full! Cannot add more fuel entries.\n");
        return;
    }
    stack->top++;
    stack->stack[stack->top].fuelAmount = fuelAmount;
    snprintf(stack->stack[stack->top].timestamp, sizeof(stack->stack[stack->top].timestamp), "%s",
timestamp);
    printf("Fuel usage entry added: %.2f at %s\n", fuelAmount, timestamp);
}

void pop(FuelStack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! No entries to remove.\n");
        return;
    }
    printf("Removing fuel usage entry: %.2f at %s\n", stack->stack[stack->top].fuelAmount,
stack->stack[stack->top].timestamp);
    stack->top--;
}

void peek(FuelStack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! No entries to peek.\n");
        return;
    }
    printf("Latest fuel usage entry: %.2f at %s\n", stack->stack[stack->top].fuelAmount,
stack->stack[stack->top].timestamp);
}

void viewAll(FuelStack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! No entries to view.\n");
        return;
    }
    printf("Fuel usage entries:\n");
    for (int i = 0; i <= stack->top; i++) {
        printf("Entry %d: %.2f at %s\n", i + 1, stack->stack[i].fuelAmount, stack->stack[i].timestamp);
    }
}

void search(FuelStack *stack, float fuelAmount) {
    int found = 0;
    for (int i = 0; i <= stack->top; i++) {
        if (stack->stack[i].fuelAmount == fuelAmount) {
            printf("Found fuel usage entry: %.2f at %s\n", stack->stack[i].fuelAmount,
stack->stack[i].timestamp);
            found = 1;
        }
    }
    if (!found) {
        printf("No fuel usage entry with %.2f found.\n", fuelAmount);
    }
}

int main() {
```

```
FuelStack stack;
initStack(&stack);

int choice;
float fuelAmount;
char timestamp[20];

while (1) {
    // Menu
    printf("\nFuel Management System Menu:\n");
    printf("1: Add a fuel usage entry (push)\n");
    printf("2: Remove the last entry (pop)\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: ");

    // Input handling (no fgets or getchar)
    if (scanf("%d", &choice) != 1) {
        printf("Invalid input! Please enter an integer.\n");
        while(getchar() != '\n'); // clear buffer
        continue;
    }

    switch (choice) {
        case 1: // Add a fuel usage entry
            printf("Enter fuel usage amount: ");
            if (scanf("%f", &fuelAmount) != 1) {
                printf("Invalid input! Please enter a valid number.\n");
                while(getchar() != '\n'); // clear buffer
                continue;
            }
            printf("Enter timestamp (YYYY-MM-DD HH:MM:SS): ");
            if (scanf("%s", timestamp) != 1) {
                printf("Invalid input! Please enter a valid timestamp.\n");
                while(getchar() != '\n'); // clear buffer
                continue;
            }
            push(&stack, fuelAmount, timestamp);
            break;

        case 2: // Remove the last entry
            pop(&stack);
            break;

        case 3: // View all fuel usage data
            viewAll(&stack);
            break;

        case 4: // Peek at the latest fuel usage entry
            peek(&stack);
            break;

        case 5: // Search for a specific fuel usage entry
```

```
            printf("Enter fuel usage amount to search for: ");
            if (scanf("%f", &fuelAmount) != 1) {
                printf("Invalid input! Please enter a valid number.\n");
                while(getchar() != '\n'); // clear buffer
                continue;
            }
            search(&stack, fuelAmount);
            break;

        case 6: // Exit
            printf("Exiting Fuel Management System...\n");
            return 0;

        default:
            printf("Invalid choice! Please choose a valid option.\n");
      }
   }

   return 0;
}
```

****

STACK USING LINKED LIST

==========

CODE

==========

```
#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;
}
```

SET OF PROBLEMS
==========
1.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

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

// Structure for an order node
struct Order {
    int orderId;
    char description[100];
    struct Order *next;
};
```

```
// Function prototypes
void push(struct Order *top, int id, const char *description);
void pop(struct Order *top);
void display(struct Order *top);
void peek(struct Order *top);
void search(struct Order *top, int id);

int main() {
    struct Order *stackTop = NULL;
    int choice, orderId;
    char description[100];

    while (1) {
        // Display menu
        printf("\nOrder Processing System\n");
        printf("1. Add a new order\n");
        printf("2. Process the last order\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:
                // Add a new order
                printf("Enter Order ID: ");
                scanf("%d", &orderId);
                // Clear the input buffer
                while(getchar() != '\n'); // To consume the newline character left by scanf
                printf("Enter Order Description: ");
                scanf("%[^\n]", description);  // Reads until newline
                push(stackTop, orderId, description);
                break;

            case 2:
                // Process the last order
                pop(stackTop);
                break;

            case 3:
                // Display all pending orders
                display(stackTop);
                break;

            case 4:
                // Peek at the next order to be processed
                peek(stackTop);
                break;

            case 5:
                // Search for a specific order
                printf("Enter Order ID to search for: ");
```

```
            scanf("%d", &orderId);
            search(stackTop, orderId);
            break;

        case 6:
            // Exit
            printf("Exiting the program.\n");
            exit(0);

        default:
            printf("Invalid choice, please try again.\n");
      }
   }

   return 0;
}

// Pushes a new order onto the stack
void push(struct Order *top, int id, const char *description) {
   struct Order *newOrder = (struct Order *)malloc(sizeof(struct Order));
   newOrder->orderId = id;
   strcpy(newOrder->description, description);
   newOrder->next = top;  // Point new order to the current top
   top = newOrder;        // Update the top to the new order
   printf("Order %d added to the stack.\n", id);
}

// Pops the last order from the stack
void pop(struct Order *top) {
   if (top == NULL) {
      printf("No orders to process.\n");
      return;
   }

   struct Order *temp = top;
   top = top->next;  // Move the top to the next order
   printf("Order %d processed: %s\n", temp->orderId, temp->description);
   free(temp);  // Free the memory of the processed order
}

// Displays all pending orders
void display(struct Order *top) {
   if (top == NULL) {
      printf("No pending orders.\n");
      return;
   }

   printf("Pending Orders:\n");
   struct Order *current = top;
   while (current != NULL) {
      printf("Order ID: %d, Description: %s\n", current->orderId, current->description);
      current = current->next;
   }
}
```

```
// Peeks at the next order to be processed
void peek(struct Order *top) {
    if (top == NULL) {
        printf("No orders to peek at.\n");
        return;
    }

    printf("Next order to process: Order ID: %d, Description: %s\n", top->orderId, top->description);
}

// Searches for a specific order by ID
void search(struct Order *top, int id) {
    struct Order *current = top;
    while (current != NULL) {
        if (current->orderId == id) {
            printf("Order found: Order ID: %d, Description: %s\n", current->orderId, current->description);
            return;
        }
        current = current->next;
    }
    printf("Order ID %d not found.\n", id);
}
```

2.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

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

typedef struct Ticket {
    int ticketID;
    char description[256];
    struct Ticket* next;
} Ticket;

Ticket* stack = NULL; // Top of the stack

// Function to create a new ticket
Ticket* createTicket(int ticketID, const char* description) {
    Ticket* newTicket = (Ticket*)malloc(sizeof(Ticket));
    if (newTicket == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newTicket->ticketID = ticketID;
    strcpy(newTicket->description, description);
    newTicket->next = NULL;
    return newTicket;
```

```
}

// Push a new ticket onto the stack
void addTicket() {
    int ticketID;
    char description[256];

    printf("Enter ticket ID: ");
    scanf("%d", &ticketID);
    printf("Enter ticket description: ");
    getchar();  // To consume newline character from previous input
    fgets(description, sizeof(description), stdin);

    // Remove newline character at the end of the description
    description[strcspn(description, "\n")] = '\0';

    Ticket* newTicket = createTicket(ticketID, description);
    newTicket->next = stack;
    stack = newTicket;

    printf("Ticket added successfully!\n");
}

// Pop the latest ticket from the stack
void resolveTicket() {
    if (stack == NULL) {
        printf("No pending tickets to resolve.\n");
        return;
    }

    Ticket* resolvedTicket = stack;
    stack = stack->next;
    printf("Ticket resolved: ID %d, Description: %s\n", resolvedTicket->ticketID,
resolvedTicket->description);
    free(resolvedTicket);
}

// View all pending tickets
void viewAllTickets() {
    if (stack == NULL) {
        printf("No pending tickets.\n");
        return;
    }

    Ticket* current = stack;
    while (current != NULL) {
        printf("Ticket ID: %d, Description: %s\n", current->ticketID, current->description);
        current = current->next;
    }
}

// Peek at the latest ticket
void peekTicket() {
    if (stack == NULL) {
        printf("No pending tickets.\n");
```

```
        return;
    }

    printf("Latest Ticket ID: %d, Description: %s\n", stack->ticketID, stack->description);
}

// Search for a specific ticket
void searchTicket() {
    int ticketID;
    printf("Enter ticket ID to search for: ");
    scanf("%d", &ticketID);

    Ticket* current = stack;
    while (current != NULL) {
        if (current->ticketID == ticketID) {
            printf("Ticket found: ID %d, Description: %s\n", current->ticketID, current->description);
            return;
        }
        current = current->next;
    }
    printf("Ticket with ID %d not found.\n", ticketID);
}

int main() {
    int choice;

    do {
        printf("\nCustomer Support Ticketing System\n");
        printf("1. Add a new ticket (push)\n");
        printf("2. Resolve the latest ticket (pop)\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);

        switch (choice) {
            case 1:
                addTicket();
                break;
            case 2:
                resolveTicket();
                break;
            case 3:
                viewAllTickets();
                break;
            case 4:
                peekTicket();
                break;
            case 5:
                searchTicket();
                break;
            case 6:
                printf("Exiting...\n");
```

```
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 6);

// Free any remaining tickets before exiting
while (stack != NULL) {
    Ticket* temp = stack;
    stack = stack->next;
    free(temp);
}

return 0;
}
```

3.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

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

#define MAX_NAME_LENGTH 100

// Define the structure for a return request
typedef struct ReturnRequest {
    int id;                 // Unique return ID
    char productName[MAX_NAME_LENGTH];  // Product name
    struct ReturnRequest* next;  // Pointer to the next return request
} ReturnRequest;

// Function to create a new return request
ReturnRequest* createRequest(int id, const char* productName) {
    ReturnRequest* newRequest = (ReturnRequest*)malloc(sizeof(ReturnRequest));
    if (newRequest == NULL) {
        printf("Memory allocation failed.\n");
        return NULL;
    }
    newRequest->id = id;
    strncpy(newRequest->productName, productName, MAX_NAME_LENGTH - 1);
    newRequest->productName[MAX_NAME_LENGTH - 1] = '\0';
    newRequest->next = NULL;
    return newRequest;
}

// Function to push a return request onto the stack (linked list)
void push(ReturnRequest** stack, int id, const char* productName) {
    ReturnRequest* newRequest = createRequest(id, productName);
```

```
    if (newRequest == NULL) {
        return;
    }
    newRequest->next = *stack;
    *stack = newRequest;
    printf("Return request added: ID = %d, Product = %s\n", id, productName);
}

// Function to pop the top return request from the stack
void pop(ReturnRequest** stack) {
    if (*stack == NULL) {
        printf("No return requests to process.\n");
        return;
    }
    ReturnRequest* temp = *stack;
    *stack = (*stack)->next;
    printf("Processing return request: ID = %d, Product = %s\n", temp->id, temp->productName);
    free(temp);
}

// Function to display all return requests
void display(ReturnRequest* stack) {
    if (stack == NULL) {
        printf("No return requests to display.\n");
        return;
    }
    printf("All return requests:\n");
    ReturnRequest* current = stack;
    while (current != NULL) {
        printf("ID = %d, Product = %s\n", current->id, current->productName);
        current = current->next;
    }
}

// Function to peek at the top return request (next to process)
void peek(ReturnRequest* stack) {
    if (stack == NULL) {
        printf("No return requests to peek at.\n");
        return;
    }
    printf("Next return to process: ID = %d, Product = %s\n", stack->id, stack->productName);
}

// Function to search for a specific return request by ID
void search(ReturnRequest* stack, int id) {
    if (stack == NULL) {
        printf("No return requests to search through.\n");
        return;
    }
    ReturnRequest* current = stack;
    while (current != NULL) {
        if (current->id == id) {
            printf("Found return request: ID = %d, Product = %s\n", current->id, current->productName);
            return;
        }
```

```
        current = current->next;
    }
    printf("Return request with ID = %d not found.\n", id);
}

int main() {
    ReturnRequest* stack = NULL;
    int choice, id;
    char productName[MAX_NAME_LENGTH];

    while (1) {
        printf("\nProduct Return Management\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);

        switch (choice) {
            case 1:
                // Add a new return request
                printf("Enter the return request ID: ");
                scanf("%d", &id);
                printf("Enter the product name: ");
                // Use fgets to read product name, and discard newline character
                getchar();  // to consume the newline from previous input
                fgets(productName, MAX_NAME_LENGTH, stdin);
                productName[strcspn(productName, "\n")] = '\0';  // Remove newline
                push(&stack, id, productName);
                break;

            case 2:
                // Process the last return (pop)
                pop(&stack);
                break;

            case 3:
                // Display all return requests
                display(stack);
                break;

            case 4:
                // Peek at the next return to process
                peek(stack);
                break;

            case 5:
                // Search for a specific return request by ID
                printf("Enter the return request ID to search: ");
                scanf("%d", &id);
                search(stack, id);
                break;
```

```
            case 6:
                // Exit the program
                printf("Exiting the program...\n");
                // Free remaining allocated memory
                while (stack != NULL) {
                    pop(&stack);
                }
                return 0;

            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

4.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

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

// Define the structure for a restock entry
struct Restock {
    int item_id;
    int quantity;
    char date[20];
    struct Restock* next; // Pointer to the next entry in the stack
};

// Function to create a new restock entry
struct Restock* createRestock(int item_id, int quantity, const char* date) {
    struct Restock* newRestock = (struct Restock*)malloc(sizeof(struct Restock));
    newRestock->item_id = item_id;
    newRestock->quantity = quantity;
    strcpy(newRestock->date, date);
    newRestock->next = NULL;
    return newRestock;
}

// Function to add a restock entry (Push to the stack)
void push(struct Restock** stack, int item_id, int quantity, const char* date) {
    struct Restock* newRestock = createRestock(item_id, quantity, date);
    newRestock->next = *stack;
    *stack = newRestock;
    printf("Restock entry added successfully.\n");
```

```
}

// Function to process the last restock entry (Pop from the stack)
void pop(struct Restock** stack) {
    if (*stack == NULL) {
        printf("The stack is empty. No restocks to process.\n");
        return;
    }

    struct Restock* temp = *stack;
    *stack = (*stack)->next;
    printf("Processed restock entry (Item ID: %d, Quantity: %d, Date: %s)\n",
           temp->item_id, temp->quantity, temp->date);
    free(temp);
}

// Function to view all restock entries
void viewAllRestocks(struct Restock* stack) {
    if (stack == NULL) {
        printf("No restock entries to display.\n");
        return;
    }

    printf("All restock entries:\n");
    struct Restock* current = stack;
    while (current != NULL) {
        printf("Item ID: %d, Quantity: %d, Date: %s\n", current->item_id, current->quantity, current->date);
        current = current->next;
    }
}

// Function to peek at the latest restock entry
void peek(struct Restock* stack) {
    if (stack == NULL) {
        printf("No restock entries to peek at.\n");
        return;
    }

    printf("Latest restock entry (Item ID: %d, Quantity: %d, Date: %s)\n",
           stack->item_id, stack->quantity, stack->date);
}

// Function to search for a specific restock entry by Item ID
void searchRestock(struct Restock* stack, int item_id) {
    struct Restock* current = stack;
    while (current != NULL) {
        if (current->item_id == item_id) {
            printf("Found restock entry (Item ID: %d, Quantity: %d, Date: %s)\n",
                   current->item_id, current->quantity, current->date);
            return;
        }
        current = current->next;
    }
    printf("No restock entry found for Item ID: %d\n", item_id);
}
```

```
int main() {
    struct Restock* stack = NULL; // Initialize the stack as empty
    int choice, item_id, quantity;
    char date[20];

    do {
        printf("\nInventory Restock System\n");
        printf("1: Add a restock entry (Push)\n");
        printf("2: Process the last restock (Pop)\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);

        switch (choice) {
            case 1: // Add a restock entry
                printf("Enter item ID: ");
                scanf("%d", &item_id);
                printf("Enter quantity: ");
                scanf("%d", &quantity);
                printf("Enter date (YYYY-MM-DD): ");
                scanf("%s", date);
                push(&stack, item_id, quantity, date);
                break;

            case 2: // Process the last restock entry
                pop(&stack);
                break;

            case 3: // View all restock entries
                viewAllRestocks(stack);
                break;

            case 4: // Peek at the latest restock entry
                peek(stack);
                break;

            case 5: // Search for a specific restock entry
                printf("Enter item ID to search: ");
                scanf("%d", &item_id);
                searchRestock(stack, item_id);
                break;

            case 6: // Exit
                printf("Exiting the system.\n");
                break;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 6);
```

```
    // Free all remaining stack entries before exit
    while (stack != NULL) {
        pop(&stack);
    }

    return 0;
}
```

5.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

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

// Define the structure of a flash sale deal
typedef struct Deal {
    char name[50];
    float discount;
    struct Deal* next;
} Deal;

// Function prototypes
void push(Deal** head, char name[], float discount);
void pop(Deal** head);
void viewDeals(Deal* head);
void peek(Deal* head);
Deal* searchDeal(Deal* head, char name[]);

// Main function with switch-case menu
int main() {
    Deal* head = NULL;
    int choice;
    char name[50];
    float discount;

    while(1) {
        printf("\nFlash Sale Deal Management Menu:\n");
        printf("1: Add a new deal (push)\n");
        printf("2: Remove the last deal (pop)\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);

        switch(choice) {
            case 1:
```

```
            printf("Enter deal name: ");
            scanf(" %[^\n]", name);  // Scan input including spaces
            printf("Enter discount: ");
            scanf("%f", &discount);
            push(&head, name, discount);
            break;

        case 2:
            pop(&head);
            break;

        case 3:
            viewDeals(head);
            break;

        case 4:
            peek(head);
            break;

        case 5:
            printf("Enter the deal name to search for: ");
            scanf(" %[^\n]", name);  // Scan input including spaces
            Deal* deal = searchDeal(head, name);
            if (deal != NULL) {
                printf("Deal found: %s, Discount: %.2f%%\n", deal->name, deal->discount);
            } else {
                printf("Deal not found!\n");
            }
            break;

        case 6:
            printf("Exiting...\n");
            return 0;

        default:
            printf("Invalid choice! Please try again.\n");
      }
   }
}

// Function to add a new deal (push)
void push(Deal** head, char name[], float discount) {
   Deal* newDeal = (Deal*)malloc(sizeof(Deal));
   if (!newDeal) {
      printf("Memory allocation failed!\n");
      return;
   }
   strcpy(newDeal->name, name);
   newDeal->discount = discount;
   newDeal->next = *head;
   *head = newDeal;
   printf("Deal added successfully: %s, Discount: %.2f%%\n", name, discount);
}

// Function to remove the last deal (pop)
```

```
void pop(Deal** head) {
    if (*head == NULL) {
        printf("No deals to remove!\n");
        return;
    }
    Deal* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("Last deal removed.\n");
}

// Function to view all active deals
void viewDeals(Deal* head) {
    if (head == NULL) {
        printf("No active deals.\n");
        return;
    }
    Deal* current = head;
    while (current != NULL) {
        printf("Deal: %s, Discount: %.2f%%\n", current->name, current->discount);
        current = current->next;
    }
}

// Function to peek at the latest deal
void peek(Deal* head) {
    if (head == NULL) {
        printf("No deals available to peek.\n");
        return;
    }
    printf("Latest deal: %s, Discount: %.2f%%\n", head->name, head->discount);
}

// Function to search for a specific deal
Deal* searchDeal(Deal* head, char name[]) {
    Deal* current = head;
    while (current != NULL) {
        if (strcmp(current->name, name) == 0) {
            return current;
        }
        current = current->next;
    }
    return NULL;  // Deal not found
}
```

6.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

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

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

// Define a structure for the linked list node
struct Session {
    int sessionID;
    char sessionDetails[100];  // Store session details (could be more complex in real applications)
    struct Session* next;
};

// Define the stack (linked list)
struct Session* top = NULL;

// Function to create a new session node
struct Session* createSession(int sessionID, const char* details) {
    struct Session* newSession = (struct Session*)malloc(sizeof(struct Session));
    if (newSession == NULL) {
        printf("Memory allocation failed!\n");
        return NULL;
    }
    newSession->sessionID = sessionID;
    strcpy(newSession->sessionDetails, details);
    newSession->next = NULL;
    return newSession;
}

// Push function to add a session to the stack
void push(int sessionID, const char* details) {
    struct Session* newSession = createSession(sessionID, details);
    if (newSession == NULL) return;
    newSession->next = top;
    top = newSession;
    printf("Session added: ID = %d, Details = %s\n", sessionID, details);
}

// Pop function to remove the most recent session
void pop() {
    if (top == NULL) {
        printf("No session to end. Stack is empty.\n");
        return;
    }
    struct Session* temp = top;
    top = top->next;
    printf("Session ended: ID = %d, Details = %s\n", temp->sessionID, temp->sessionDetails);
    free(temp);
}

// Display all sessions in the stack
void displaySessions() {
    if (top == NULL) {
        printf("No sessions in the stack.\n");
        return;
    }
    struct Session* current = top;
    printf("Current session stack:\n");
```

```
    while (current != NULL) {
        printf("Session ID: %d, Details: %s\n", current->sessionID, current->sessionDetails);
        current = current->next;
    }
}

// Peek at the most recent session
void peek() {
    if (top == NULL) {
        printf("No sessions in the stack.\n");
        return;
    }
    printf("Most recent session: ID = %d, Details = %s\n", top->sessionID, top->sessionDetails);
}

// Search for a specific session by ID
void searchSession(int sessionID) {
    struct Session* current = top;
    while (current != NULL) {
        if (current->sessionID == sessionID) {
            printf("Session found: ID = %d, Details = %s\n", current->sessionID, current->sessionDetails);
            return;
        }
        current = current->next;
    }
    printf("Session with ID %d not found.\n", sessionID);
}

// Main function with switch-case menu
int main() {
    int choice, sessionID;
    char sessionDetails[100];

    while (1) {
        printf("\nSession History Menu:\n");
        printf("1: Add a session (push)\n");
        printf("2: End the last session (pop)\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);

        switch (choice) {
            case 1:
                printf("Enter session ID: ");
                scanf("%d", &sessionID);
                getchar();  // Consume the newline character left by scanf
                printf("Enter session details: ");
                fgets(sessionDetails, sizeof(sessionDetails), stdin);
                sessionDetails[strcspn(sessionDetails, "\n")] = 0; // Remove newline at the end
                push(sessionID, sessionDetails);
                break;
```

```
            case 2:
                pop();
                break;

            case 3:
                displaySessions();
                break;

            case 4:
                peek();
                break;

            case 5:
                printf("Enter session ID to search: ");
                scanf("%d", &sessionID);
                searchSession(sessionID);
                break;

            case 6:
                printf("Exiting the program.\n");
                return 0;

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

    return 0;
}
```

7.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

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

typedef struct Product {
    char name[100];           // Product name
    struct Product* next;     // Pointer to the next product
} Product;

// Function to create a new product node
Product* createProduct(const char* name) {
    Product* newProduct = (Product*)malloc(sizeof(Product));
    if (newProduct != NULL) {
        strncpy(newProduct->name, name, sizeof(newProduct->name) - 1);
        newProduct->name[sizeof(newProduct->name) - 1] = '\0';
        newProduct->next = NULL;
```

```
    }
    return newProduct;
}

// Function to push a product onto the wishlist
void push(Product** head, const char* name) {
    Product* newProduct = createProduct(name);
    if (newProduct == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }
    newProduct->next = *head;
    *head = newProduct;
    printf("Product '%s' added to wishlist.\n", name);
}

// Function to pop the last added product from the wishlist
void pop(Product** head) {
    if (*head == NULL) {
        printf("Wishlist is empty.\n");
        return;
    }
    Product* temp = *head;
    *head = (*head)->next;
    printf("Product '%s' removed from wishlist.\n", temp->name);
    free(temp);
}

// Function to view all products in the wishlist
void viewWishlist(Product* head) {
    if (head == NULL) {
        printf("Wishlist is empty.\n");
        return;
    }
    Product* current = head;
    printf("Wishlist items:\n");
    while (current != NULL) {
        printf("- %s\n", current->name);
        current = current->next;
    }
}

// Function to peek at the most recent product
void peek(Product* head) {
    if (head == NULL) {
        printf("Wishlist is empty.\n");
        return;
    }
    printf("Most recent product: %s\n", head->name);
}

// Function to search for a specific product in the wishlist
void search(Product* head, const char* name) {
    Product* current = head;
    while (current != NULL) {
```

```
        if (strcmp(current->name, name) == 0) {
            printf("Product '%s' found in wishlist.\n", name);
            return;
        }
        current = current->next;
    }
    printf("Product '%s' not found in wishlist.\n", name);
}

int main() {
    Product* wishlist = NULL;  // Head of the linked list
    int choice;
    char productName[100];

    while (1) {
        printf("\nWishlist Menu:\n");
        printf("1. Add a product to wishlist (push)\n");
        printf("2. Remove the last added product (pop)\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 in wishlist\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();  // To consume the newline character after entering the choice

        switch (choice) {
            case 1:
                printf("Enter product name to add: ");
                fgets(productName, sizeof(productName), stdin);
                productName[strcspn(productName, "\n")] = '\0';  // Remove the newline
                push(&wishlist, productName);
                break;
            case 2:
                pop(&wishlist);
                break;
            case 3:
                viewWishlist(wishlist);
                break;
            case 4:
                peek(wishlist);
                break;
            case 5:
                printf("Enter product name to search: ");
                fgets(productName, sizeof(productName), stdin);
                productName[strcspn(productName, "\n")] = '\0';  // Remove the newline
                search(wishlist, productName);
                break;
            case 6:
                printf("Exiting the wishlist manager.\n");
                // Clean up the memory
                while (wishlist != NULL) {
                    pop(&wishlist);
                }
                return 0;
```

```
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

8.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

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

// Define the structure for a node in the linked list
struct Node {
    char step[100]; // The checkout step (using a string)
    struct Node* next; // Pointer to the next node
};

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

// Function to initialize the stack
void initStack(struct Stack* stack) {
    stack->top = NULL; // Initially, the stack is empty
}

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

// Function to add a checkout step (push) to the stack
void push(struct Stack* stack, const char* step) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    strcpy(newNode->step, step);
    newNode->next = stack->top; // Link the new node to the previous top
    stack->top = newNode; // Update the top of the stack
    printf("Checkout step '%s' added.\n", step);
}
```

```
// Function to remove the last checkout step (pop) from the stack
void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("The stack is empty! No steps to remove.\n");
        return;
    }
    struct Node* temp = stack->top;
    stack->top = stack->top->next; // Move the top pointer to the next node
    printf("Checkout step '%s' removed.\n", temp->step);
    free(temp); // Free the memory of the removed node
}

// Function to display all checkout steps
void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("The stack is empty! No steps to display.\n");
        return;
    }
    struct Node* temp = stack->top;
    printf("Checkout steps:\n");
    while (temp != NULL) {
        printf("- %s\n", temp->step);
        temp = temp->next;
    }
}

// Function to peek at the current checkout step (the top of the stack)
void peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("The stack is empty! No current step.\n");
        return;
    }
    printf("Current checkout step: '%s'\n", stack->top->step);
}

// Function to search for a specific checkout step
void search(struct Stack* stack, const char* step) {
    if (isEmpty(stack)) {
        printf("The stack is empty! No steps to search.\n");
        return;
    }
    struct Node* temp = stack->top;
    int found = 0;
    while (temp != NULL) {
        if (strcmp(temp->step, step) == 0) {
            found = 1;
            break;
        }
        temp = temp->next;
    }
    if (found) {
        printf("Step '%s' found in the checkout process.\n", step);
    } else {
        printf("Step '%s' not found in the checkout process.\n", step);
    }
```

```
}

// Main function with a switch-case menu
int main() {
    struct Stack stack;
    initStack(&stack); // Initialize the stack
    int choice;
    char step[100];

    while (1) {
        printf("\nCheckout Process Menu:\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);
        getchar(); // Consume the newline character

        switch (choice) {
            case 1:
                printf("Enter the checkout step: ");
                fgets(step, sizeof(step), stdin);
                step[strcspn(step, "\n")] = '\0'; // Remove the newline character
                push(&stack, step);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                display(&stack);
                break;
            case 4:
                peek(&stack);
                break;
            case 5:
                printf("Enter the step to search: ");
                fgets(step, sizeof(step), stdin);
                step[strcspn(step, "\n")] = '\0'; // Remove the newline character
                search(&stack, step);
                break;
            case 6:
                printf("Exiting the checkout process.\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

9.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

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

// Define a structure for each node in the linked list
struct Node {
    char couponCode[50];
    struct Node* next;
};

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

// Push function: Add a coupon code to the stack
void push(struct Node** top, char* couponCode) {
    struct Node* newNode = createNode(couponCode);
    newNode->next = *top;
    *top = newNode;
}

// Pop function: Remove the coupon code from the stack
void pop(struct Node** top) {
    if (*top == NULL) {
        printf("The stack is empty!\n");
        return;
    }
    struct Node* temp = *top;
    *top = (*top)->next;
    printf("Removed coupon code: %s\n", temp->couponCode);
    free(temp);
}

// Peek function: View the latest coupon code in the stack
void peek(struct Node* top) {
    if (top == NULL) {
        printf("The stack is empty!\n");
        return;
    }
    printf("Latest coupon code: %s\n", top->couponCode);
}

// View all function: Display all coupon codes in the stack
```

```
void viewAll(struct Node* top) {
    if (top == NULL) {
        printf("The stack is empty!\n");
        return;
    }
    struct Node* current = top;
    while (current != NULL) {
        printf("%s\n", current->couponCode);
        current = current->next;
    }
}

// Search function: Search for a coupon code in the stack
void search(struct Node* top, char* couponCode) {
    struct Node* current = top;
    while (current != NULL) {
        if (strcmp(current->couponCode, couponCode) == 0) {
            printf("Coupon code '%s' found!\n", couponCode);
            return;
        }
        current = current->next;
    }
    printf("Coupon code '%s' not found.\n", couponCode);
}

// Main function: Menu-driven program to manage coupon codes
int main() {
    struct Node* stack = NULL;
    int choice;
    char couponCode[50];

    do {
        // Display menu
        printf("\nCoupon Code Management\n");
        printf("1: Add a new coupon code (push)\n");
        printf("2: Remove the last coupon code (pop)\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);

        switch (choice) {
            case 1:
                printf("Enter the coupon code: ");
                scanf("%s", couponCode);
                push(&stack, couponCode);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                printf("All available coupon codes:\n");
                viewAll(stack);
```

```
            break;
        case 4:
            peek(stack);
            break;
        case 5:
            printf("Enter the coupon code to search for: ");
            scanf("%s", couponCode);
            search(stack, couponCode);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
    }
  } while (choice != 6);

  // Free all the remaining nodes before exiting
  while (stack != NULL) {
    pop(&stack);
  }

  return 0;
}
```

10.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

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

typedef struct Node {
  char status[100]; // Shipping status message
  struct Node* next; // Pointer to the next node
} Node;

// Function to create a new node
Node* createNode(const char* status) {
  Node* newNode = (Node*)malloc(sizeof(Node));
  if (newNode == NULL) {
    printf("Memory allocation failed!\n");
    exit(1);
  }
  strcpy(newNode->status, status);
  newNode->next = NULL;
  return newNode;
}
```

```
// Push function to add a new status to the stack
void push(Node** top, const char* status) {
    Node* newNode = createNode(status);
    newNode->next = *top;
    *top = newNode;
    printf("Shipping status added: %s\n", status);
}

// Pop function to remove the last status from the stack
void pop(Node** top) {
    if (*top == NULL) {
        printf("No shipping status updates available!\n");
        return;
    }
    Node* temp = *top;
    *top = (*top)->next;
    printf("Removed shipping status: %s\n", temp->status);
    free(temp);
}

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

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

// Function to search for a specific shipping status
void search(Node* top, const char* status) {
    Node* temp = top;
    while (temp != NULL) {
        if (strcmp(temp->status, status) == 0) {
            printf("Shipping status found: %s\n", status);
            return;
        }
        temp = temp->next;
    }
    printf("Shipping status '%s' not found.\n", status);
}
```

```
int main() {
    Node* top = NULL;  // Initialize the stack (empty)
    int choice;
    char status[100];

    while (1) {
        printf("\nShipping Status Tracker\n");
        printf("1: Add a shipping status update\n");
        printf("2: Remove the last update\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);
        getchar();  // to clear the newline character left by scanf

        switch (choice) {
            case 1:
                printf("Enter shipping status: ");
                fgets(status, sizeof(status), stdin);
                status[strcspn(status, "\n")] = '\0';  // Remove trailing newline
                push(&top, status);
                break;
            case 2:
                pop(&top);
                break;
            case 3:
                viewAll(top);
                break;
            case 4:
                peek(top);
                break;
            case 5:
                printf("Enter shipping status to search for: ");
                fgets(status, sizeof(status), stdin);
                status[strcspn(status, "\n")] = '\0';  // Remove trailing newline
                search(top, status);
                break;
            case 6:
                printf("Exiting program...\n");
                while (top != NULL) {
                    pop(&top);  // Free memory before exit
                }
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

11.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

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

// Define the structure for a review node
struct ReviewNode {
    char review[256];  // Review text
    struct ReviewNode* next;  // Pointer to the next review
};

// Define the structure for the stack
struct Stack {
    struct ReviewNode* top;  // Top of the stack
};

// Function to create a new node for a review
struct ReviewNode* createReviewNode(const char* reviewText) {
    struct ReviewNode* newNode = (struct ReviewNode*)malloc(sizeof(struct ReviewNode));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    strcpy(newNode->review, reviewText);
    newNode->next = NULL;
    return newNode;
}

// Function to push a new review onto the stack
void push(struct Stack* stack, const char* reviewText) {
    struct ReviewNode* newNode = createReviewNode(reviewText);
    newNode->next = stack->top;
    stack->top = newNode;
    printf("Review added successfully!\n");
}

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

```
// Function to display all reviews in the stack
void displayReviews(struct Stack* stack) {
    if (stack->top == NULL) {
        printf("No reviews available!\n");
        return;
    }
    struct ReviewNode* current = stack->top;
    printf("Displaying all reviews:\n");
    while (current != NULL) {
        printf("- %s\n", current->review);
        current = current->next;
    }
}

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

// Function to search for a specific review
void searchReview(struct Stack* stack, const char* searchText) {
    if (stack->top == NULL) {
        printf("No reviews available to search!\n");
        return;
    }
    struct ReviewNode* current = stack->top;
    while (current != NULL) {
        if (strstr(current->review, searchText) != NULL) {
            printf("Found review: %s\n", current->review);
            return;
        }
        current = current->next;
    }
    printf("Review not found!\n");
}

// Main function with menu options
int main() {
    struct Stack stack = { NULL };  // Initialize an empty stack
    int choice;
    char reviewText[256];
    char searchText[256];

    do {
        printf("\nMenu:\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);
        getchar();  // To consume newline character after scanf

        switch (choice) {
            case 1:
                printf("Enter your review: ");
                fgets(reviewText, sizeof(reviewText), stdin);
                reviewText[strcspn(reviewText, "\n")] = 0;  // Remove newline character
                push(&stack, reviewText);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                displayReviews(&stack);
                break;
            case 4:
                peek(&stack);
                break;
            case 5:
                printf("Enter search text: ");
                fgets(searchText, sizeof(searchText), stdin);
                searchText[strcspn(searchText, "\n")] = 0;  // Remove newline character
                searchReview(&stack, searchText);
                break;
            case 6:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

12.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

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

typedef struct Notification {
    char message[256];
    struct Notification* next;
} Notification;
```

```
Notification* top = NULL; // Pointer to the top of the stack

// Function to push a new notification onto the stack
void push(char* message) {
    Notification* newNotification = (Notification*)malloc(sizeof(Notification));
    if (newNotification == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    strcpy(newNotification->message, message);
    newNotification->next = top;
    top = newNotification;
    printf("Notification added: %s\n", message);
}

// Function to pop the last notification from the stack
void pop() {
    if (top == NULL) {
        printf("No notifications to remove!\n");
        return;
    }
    Notification* temp = top;
    top = top->next;
    printf("Notification removed: %s\n", temp->message);
    free(temp);
}

// Function to view all notifications
void viewAllNotifications() {
    if (top == NULL) {
        printf("No notifications to display.\n");
        return;
    }
    Notification* current = top;
    printf("All notifications:\n");
    while (current != NULL) {
        printf("- %s\n", current->message);
        current = current->next;
    }
}

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

// Function to search for a specific notification
void search(char* searchMessage) {
    if (top == NULL) {
        printf("No notifications to search.\n");
        return;
```

```
    }
    Notification* current = top;
    while (current != NULL) {
        if (strcmp(current->message, searchMessage) == 0) {
            printf("Found notification: %s\n", current->message);
            return;
        }
        current = current->next;
    }
    printf("Notification not found: %s\n", searchMessage);
}

int main() {
    int choice;
    char message[256];

    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 newline character

        switch (choice) {
            case 1:
                printf("Enter notification message: ");
                fgets(message, sizeof(message), stdin);
                message[strcspn(message, "\n")] = '\0';  // Remove trailing newline
                push(message);
                break;
            case 2:
                pop();
                break;
            case 3:
                viewAllNotifications();
                break;
            case 4:
                peek();
                break;
            case 5:
                printf("Enter the message to search for: ");
                fgets(message, sizeof(message), stdin);
                message[strcspn(message, "\n")] = '\0';  // Remove trailing newline
                search(message);
                break;
            case 6:
                printf("Exiting the system.\n");
                exit(0);
                break;
            default:
```

```
            printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```

13.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

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

#define MAX_NAME_LENGTH 100

// Definition of the Product node
struct Product {
    char name[MAX_NAME_LENGTH];
    struct Product* next;
};

// Function to create a new product node
struct Product* createProduct(char* productName) {
    struct Product* newProduct = (struct Product*)malloc(sizeof(struct Product));
    if (newProduct == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    strcpy(newProduct->name, productName);
    newProduct->next = NULL;
    return newProduct;
}

// Function to push a product onto the stack
void push(struct Product** top, char* productName) {
    struct Product* newProduct = createProduct(productName);
    newProduct->next = *top;
    *top = newProduct;
    printf("Product '%s' added to viewing history.\n", productName);
}

// Function to pop the top product from the stack
void pop(struct Product** top) {
    if (*top == NULL) {
        printf("No products in viewing history.\n");
        return;
    }
    struct Product* temp = *top;
    *top = (*top)->next;
```

```
    printf("Product ’%s’ removed from viewing history.\n", temp->name);
    free(temp);
}

// Function to display all products in the stack
void displayHistory(struct Product* top) {
    if (top == NULL) {
        printf("No products in viewing history.\n");
        return;
    }
    printf("Viewing History:\n");
    struct Product* current = top;
    while (current != NULL) {
        printf("- %s\n", current->name);
        current = current->next;
    }
}

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

// Function to search for a specific product
void search(struct Product* top, char* productName) {
    struct Product* current = top;
    while (current != NULL) {
        if (strcmp(current->name, productName) == 0) {
            printf("Product ’%s’ found in viewing history.\n", productName);
            return;
        }
        current = current->next;
    }
    printf("Product ’%s’ not found in viewing history.\n", productName);
}

int main() {
    struct Product* top = NULL; // Initialize the stack (viewing history)

    int choice;
    char productName[MAX_NAME_LENGTH];

    do {
        // Display menu options
        printf("\nProduct Viewing History Menu:\n");
        printf("1. Add a product to viewing history (push)\n");
        printf("2. Remove the last viewed product (pop)\n");
        printf("3. Display 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 newline character after entering the choice

        switch (choice) {
            case 1:
                printf("Enter product name to add: ");
                fgets(productName, MAX_NAME_LENGTH, stdin);
                productName[strcspn(productName, "\n")] = 0; // Remove newline character
                push(&top, productName);
                break;
            case 2:
                pop(&top);
                break;
            case 3:
                displayHistory(top);
                break;
            case 4:
                peek(top);
                break;
            case 5:
                printf("Enter product name to search: ");
                fgets(productName, MAX_NAME_LENGTH, stdin);
                productName[strcspn(productName, "\n")] = 0; // Remove newline character
                search(top, productName);
                break;
            case 6:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 6);

    // Free memory allocated for stack before exiting
    while (top != NULL) {
        pop(&top);
    }

    return 0;
}
```

14.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

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

```
// Node structure for the linked list
struct Node {
    char item[50];
    struct Node* next;
};

// Stack structure (Linked List based)
struct Stack {
    struct Node* top;
};

// Function to initialize the stack
void initStack(struct Stack* stack) {
    stack->top = NULL;
}

// Function to push an item onto the stack
void push(struct Stack* stack, 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 = stack->top;
    stack->top = newNode;
    printf("Item ’%s’ added to the cart.\n", item);
}

// Function to pop an item from the stack (remove the last added item)
void pop(struct Stack* stack) {
    if (stack->top == NULL) {
        printf("Cart is empty!\n");
        return;
    }
    struct Node* temp = stack->top;
    stack->top = stack->top->next;
    printf("Item ’%s’ removed from the cart.\n", temp->item);
    free(temp);
}

// Function to display all items in the cart
void viewCart(struct Stack* stack) {
    if (stack->top == NULL) {
        printf("Cart is empty!\n");
        return;
    }
    struct Node* current = stack->top;
    printf("Items in the cart:\n");
    while (current != NULL) {
        printf("- %s\n", current->item);
        current = current->next;
    }
}
```

```
// Function to peek the last added item
void peek(struct Stack* stack) {
    if (stack->top == NULL) {
        printf("Cart is empty!\n");
        return;
    }
    printf("Last added item: %s\n", stack->top->item);
}

// Function to search for a specific item in the cart
void search(struct Stack* stack, char* item) {
    struct Node* current = stack->top;
    while (current != NULL) {
        if (strcmp(current->item, item) == 0) {
            printf("Item '%s' found in the cart.\n", item);
            return;
        }
        current = current->next;
    }
    printf("Item '%s' not found in the cart.\n", item);
}

// Main menu to interact with the cart
int main() {
    struct Stack cart;
    initStack(&cart);
    int choice;
    char item[50];

    do {
        printf("\nShopping Cart Management\n");
        printf("1: Add an item to the cart\n");
        printf("2: Remove the last item\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 newline character left by scanf

        switch (choice) {
            case 1:
                printf("Enter the name of the item to add: ");
                fgets(item, sizeof(item), stdin);
                item[strcspn(item, "\n")] = '\0';  // Remove trailing newline
                push(&cart, item);
                break;
            case 2:
                pop(&cart);
                break;
            case 3:
                viewCart(&cart);
                break;
```

```
            case 4:
                peek(&cart);
                break;
            case 5:
                printf("Enter the name of the item to search: ");
                fgets(item, sizeof(item), stdin);
                item[strcspn(item, "\n")] = '\0';  // Remove trailing newline
                search(&cart, item);
                break;
            case 6:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 6);

    return 0;
}
```

15.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

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

typedef struct Payment {
    int paymentId;
    float amount;
    char date[11]; // Format: YYYY-MM-DD
    struct Payment *next;
} Payment;

Payment *top = NULL;  // Top of the stack

// Function to create a new payment record
Payment* createPayment(int paymentId, float amount, const char* date) {
    Payment *newPayment = (Payment*)malloc(sizeof(Payment));
    newPayment->paymentId = paymentId;
    newPayment->amount = amount;
    strcpy(newPayment->date, date);
    newPayment->next = NULL;
    return newPayment;
}

// Function to add a new payment record (push)
void addPayment(int paymentId, float amount, const char* date) {
```

```
    Payment *newPayment = createPayment(paymentId, amount, date);
    newPayment->next = top;
    top = newPayment;
    printf("Payment record added successfully!\n");
}

// Function to remove the last payment record (pop)
void removePayment() {
    if (top == NULL) {
        printf("No payment records to remove.\n");
        return;
    }
    Payment *temp = top;
    top = top->next;
    free(temp);
    printf("Last payment record removed successfully!\n");
}

// Function to view all payment records
void viewPayments() {
    if (top == NULL) {
        printf("No payment records available.\n");
        return;
    }
    Payment *current = top;
    printf("Payment History:\n");
    while (current != NULL) {
        printf("Payment ID: %d, Amount: %.2f, Date: %s\n", current->paymentId, current->amount,
current->date);
        current = current->next;
    }
}

// Function to peek at the latest payment record
void peekPayment() {
    if (top == NULL) {
        printf("No payment records available to peek.\n");
        return;
    }
    printf("Latest Payment - ID: %d, Amount: %.2f, Date: %s\n", top->paymentId, top->amount, top->date);
}

// Function to search for a specific payment record by payment ID
void searchPayment(int paymentId) {
    Payment *current = top;
    while (current != NULL) {
        if (current->paymentId == paymentId) {
            printf("Payment found: ID: %d, Amount: %.2f, Date: %s\n", current->paymentId, current->amount,
current->date);
            return;
        }
        current = current->next;
    }
    printf("Payment record with ID %d not found.\n", paymentId);
}
```

```
// Main function with menu options
int main() {
    int choice;
    int paymentId;
    float amount;
    char date[11];

    while (1) {
        printf("\nPayment History Menu:\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);

        switch (choice) {
            case 1:
                // Add a new payment record
                printf("Enter payment ID: ");
                scanf("%d", &paymentId);
                printf("Enter amount: ");
                scanf("%f", &amount);
                printf("Enter payment date (YYYY-MM-DD): ");
                scanf("%s", date);
                addPayment(paymentId, amount, date);
                break;
            case 2:
                // Remove the last payment record
                removePayment();
                break;
            case 3:
                // View all payment records
                viewPayments();
                break;
            case 4:
                // Peek at the latest payment record
                peekPayment();
                break;
            case 5:
                // Search for a specific payment record
                printf("Enter payment ID to search for: ");
                scanf("%d", &paymentId);
                searchPayment(paymentId);
                break;
            case 6:
                // Exit
                printf("Exiting the program...\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
```

```
    }

    return 0;
}
```