Stack using Array

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

struct Stack{

   int size;
   int top;
   int *s;
};

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


int main()
{
   struct Stack st;
   create(&st);

   push(&st,5);
```

```
    push(&st,6);
    push(&st,7);
    push(&st,8);

    display(st);

    int peekedValue = peek(&st);
    printf("Peeked value = %d \n", peekedValue);

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


    return 0;
}

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

}

void push(struct Stack *st ,int x){

  if(st->top == st->size-1){
```

```
    printf("Stack is full  ");

    }else {

    st->top++;
    st->s[st->top] = x;
    }
}

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

}

int pop(struct Stack *st){

    int x =-1;
    if(st->top == -1){
        printf("stack is empty\n");
    } else {

        x = st->s[st->top];
        st->top--;
```

```
    }
    return x;
}


int isEmpty(struct Stack st){

    if(st.top == -1){
        return 1;
    }
    return 0;
}

int isFull(struct Stack st){

    if(st.top == st.size-1){
        return 1;
    }
    return 0;
}

int stackTop(struct Stack st){

    if(!isEmpty){
        return st.s[st.top];
    }
    return -1;
}
```

```
int peek(struct Stack *st) {
    if (isEmpty) {
        printf("Stack is empty\n");
        return -1;
    } else {
        return st->s[st->top];
    }
}
```

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

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
int isEmpty(struct Stack);
```

```
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char path[100];
    char *poppedPath;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new flight path: ");
                scanf(" %s", path);
                push(&st, path);
                break;
            case 2:
                poppedPath = pop(&st);
```

```
            if (poppedPath != NULL)
                printf("Undone path: %s\n", poppedPath);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the flight path to search for: ");
            scanf(" %s", path);
            foundIndex = search(st, path);
            if (foundIndex != -1)
                printf("Path found at position: %d\n", foundIndex);
            else
                printf("Path not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

```
void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->paths = (char *)malloc(st->size * 100 * sizeof(char));  // Allocate memory for all paths
}

void push(struct Stack *st, char *path) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new path.\n");
    } else {
        st->top++;
        strcpy(&st->paths[st->top * 100], path);
        printf("Path added: %s\n", path);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to undo.\n");
        return NULL;
    } else {
        return &st->paths[st->top-- * 100];
    }
}

int isEmpty(struct Stack st) {
```

```
    return st.top == -1;
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.paths[st.top * 100];
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Top path: %s\n", &st.paths[st.top * 100]);
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No paths to display.\n");
    } else {
        printf("Current Flight Path Stack:\n");
        for (int i = st.top; i >= 0; i--) {
```

```
            printf("%d: %s\n", i, &st.paths[i * 100]);
        }
    }
}

int search(struct Stack st, char *path) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.paths[i * 100], path) == 0) {
            return i;
        }
    }
    return -1;
}
void 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");
}
```

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

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
int isEmpty(struct Stack);
```

```
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char deployment[100];
    char *poppedDeployment;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new satellite deployment: ");
                scanf(" %s", deployment);
                push(&st, deployment);
                break;
            case 2:
                poppedDeployment = pop(&st);
```

```
            if (poppedDeployment != NULL)
                printf("Undone deployment: %s\n", poppedDeployment);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the satellite deployment to search for: ");
            scanf(" %s", deployment);
            foundIndex = search(st, deployment);
            if (foundIndex != -1)
                printf("Deployment found at position: %d\n", foundIndex);
            else
                printf("Deployment not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

```
void create(struct Stack *st) {
   printf("Enter the size: ");
   scanf("%d", &st->size);
   st->top = -1;
   st->deployments = (char *)malloc(st->size * 100 * sizeof(char));  // Allocate memory for all deployments
}

void push(struct Stack *st, char *deployment) {
   if (isFull(*st)) {
      printf("Stack is full. Cannot add a new deployment.\n");
   } else {
      st->top++;
      strcpy(&st->deployments[st->top * 100], deployment);  // Store each deployment in its allocated position
      printf("Deployment added: %s\n", deployment);
   }
}

char* pop(struct Stack *st) {
   if (isEmpty(*st)) {
      printf("Stack is empty. Nothing to undo.\n");
      return NULL;
   } else {
      return &st->deployments[st->top-- * 100];
   }
}
```

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

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

char* stackTop(struct Stack st) {
   if (!isEmpty(st)) {
      return &st.deployments[st.top * 100];
   }
   return NULL;
}

void peek(struct Stack st) {
   if (!isEmpty(st)) {
      printf("Top deployment: %s\n", &st.deployments[st.top * 100]);  // Print the top
deployment
   } else {
      printf("Stack is empty. Nothing to peek.\n");
   }
}

void display(struct Stack st) {
   if (isEmpty(st)) {
      printf("Stack is empty. No deployments to display.\n");
   } else {
```

```
        printf("Current Deployment Sequence:\n");
        for (int i = st.top; i >= 0; i--) {
            printf("%d: %s\n", i, &st.deployments[i * 100]);  // Print each deployment
        }
    }
}

int search(struct Stack st, char *deployment) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.deployments[i * 100], deployment) == 0) {
            return i;
        }
    }
    return -1;
}

void menu() {
    printf("\nSatellite Deployment Sequence\n");
    printf("1: Push a new satellite deployment\n");
    printf("2: Pop the last deployment\n");
    printf("3: View the deployment sequence\n");
    printf("4: Peek at the latest deployment\n");
    printf("5: Search for a specific deployment\n");
    printf("6: Exit\n");
}
```

```
/*********************************************************************
****
```

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

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
```

```
int isEmpty(struct Stack);
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char item[100];
    char *poppedItem;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new checklist item: ");
                scanf(" %[^\n]s", item);
                push(&st, item);
                break;
            case 2:
```

```
            poppedItem = pop(&st);
            if (poppedItem != NULL)
                printf("Removed item: %s\n", poppedItem);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the checklist item to search for: ");
            scanf(" %[^\n]s", item);
            foundIndex = search(st, item);
            if (foundIndex != -1)
                printf("Item found at position: %d\n", foundIndex);
            else
                printf("Item not found in the checklist.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
}

return 0;
```

```
}

void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->checklist = (char *)malloc(st->size * 100 * sizeof(char));  // Allocate memory for all checklist items
}

void push(struct Stack *st, char *item) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new item.\n");
    } else {
        st->top++;
        strcpy(&st->checklist[st->top * 100], item);  // Store each item in its allocated position
        printf("Item added: %s\n", item);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to remove.\n");
        return NULL;
    } else {
        return &st->checklist[st->top-- * 100];  // Return the top item
    }
}
```

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

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.checklist[st.top * 100];  // Return the top item
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Top checklist item: %s\n", &st.checklist[st.top * 100]);  // Print the top item
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No items to display.\n");
    } else {
```

```
        printf("Current Checklist:\n");
        for (int i = st.top; i >= 0; i--) {
            printf("%d: %s\n", i, &st.checklist[i * 100]);  // Print each item
        }
    }
}

int search(struct Stack st, char *item) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.checklist[i * 100], item) == 0) {
            return i;  // Return the index if found
        }
    }
    return -1;  // Return -1 if not found
}

void menu() {
    printf("\nRocket Launch Checklist\n");
    printf("1: Add a checklist item (push)\n");
    printf("2: Remove the last item (pop)\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");
}
```

/**
****

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>

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
```

```
int isEmpty(struct Stack);
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char telemetryData[100];
    char *poppedData;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new telemetry data: ");
                scanf(" %s", telemetryData);
                push(&st, telemetryData);
                break;
            case 2:
```

```
            poppedData = pop(&st);
            if (poppedData != NULL)
                printf("Removed data: %s\n", poppedData);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the telemetry data to search for: ");
            scanf(" %s", telemetryData);
            foundIndex = search(st, telemetryData);
            if (foundIndex != -1)
                printf("Data found at position: %d\n", foundIndex);
            else
                printf("Data not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
}

return 0;
```

```
}

void create(struct Stack *st) {
   printf("Enter the size: ");
   scanf("%d", &st->size);
   st->top = -1;
   st->data = (char *)malloc(st->size * 100 * sizeof(char));  // Allocate memory for all telemetry data
}

void push(struct Stack *st, char *telemetryData) {
   if (isFull(*st)) {
      printf("Stack is full. Cannot add new telemetry data.\n");
   } else {
      st->top++;
      strcpy(&st->data[st->top * 100], telemetryData);  // Store each data entry in its allocated position
      printf("Data added: %s\n", telemetryData);
   }
}

char* pop(struct Stack *st) {
   if (isEmpty(*st)) {
      printf("Stack is empty. Nothing to remove.\n");
      return NULL;
   } else {
      return &st->data[st->top-- * 100];  // Return the top data entry
   }
}
```

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

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.data[st.top * 100];  // Return the top data entry
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Top telemetry data: %s\n", &st.data[st.top * 100]);  // Print the top data entry
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No data to display.\n");
    } else {
```

```
        printf("Stored Telemetry Data:\n");
        for (int i = st.top; i >= 0; i--) {
            printf("%d: %s\n", i, &st.data[i * 100]);
        }
    }
}

int search(struct Stack st, char *telemetryData) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.data[i * 100], telemetryData) == 0) {
            return i;  // Return the index if found
        }
    }
    return -1;
}

void menu() {
    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");
}
```

```
/*********************************************************************
****
```

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

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
int isEmpty(struct Stack);
```

```
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char task[100];
    char *poppedTask;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new task: ");
                scanf(" %s", task);
                push(&st, task);
                break;
            case 2:
                poppedTask = pop(&st);
```

```
            if (poppedTask != NULL)
                printf("Completed task: %s\n", poppedTask);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the task to search for: ");
            scanf(" %s", task);
            foundIndex = search(st, task);
            if (foundIndex != -1)
                printf("Task found at position: %d\n", foundIndex);
            else
                printf("Task not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

```
void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->tasks = (char *)malloc(st->size * 100 * sizeof(char));  // Allocate memory for all tasks
}

void push(struct Stack *st, char *task) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new task.\n");
    } else {
        st->top++;
        strcpy(&st->tasks[st->top * 100], task);  // Store each task in its allocated position
        printf("Task added: %s\n", task);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to complete.\n");
        return NULL;
    } else {
        return &st->tasks[st->top-- * 100];  // Return the top task
    }
}

int isEmpty(struct Stack st) {
```

```
    return st.top == -1;
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.tasks[st.top * 100];  // Return the top task
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Top task: %s\n", &st.tasks[st.top * 100]);
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No tasks to display.\n");
    } else {
        printf("Pending Tasks:\n");
        for (int i = st.top; i >= 0; i--) {
```

```
            printf("%d: %s\n", i, &st.tasks[i * 100]);
        }
    }
}

int search(struct Stack st, char *task) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.tasks[i * 100], task) == 0) {
            return i;  // Return the index if found
        }
    }
    return -1;  // Return -1 if not found
}

void 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");
}
```

```
/**********************************************************************
****
```

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

```
**********************************************************************
****/

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

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
int isEmpty(struct Stack);
```

```
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char step[100];
    char *poppedStep;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new countdown step: ");
                scanf(" %[^\n]s", step);
                push(&st, step);
                break;
            case 2:
                poppedStep = pop(&st);
```

```
            if (poppedStep != NULL)
                printf("Removed step: %s\n", poppedStep);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the countdown step to search for: ");
            scanf(" %[^\n]s", step);
            foundIndex = search(st, step);
            if (foundIndex != -1)
                printf("Step found at position: %d\n", foundIndex);
            else
                printf("Step not found in the countdown.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

```
void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->steps = (char *)malloc(st->size * 100 * sizeof(char));
}

void push(struct Stack *st, char *step) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new step.\n");
    } else {
        st->top++;
        strcpy(&st->steps[st->top * 100], step);
        printf("Step added: %s\n", step);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to remove.\n");
        return NULL;
    } else {
        return &st->steps[st->top-- * 100];
    }
}

int isEmpty(struct Stack st) {
```

```
    return st.top == -1;
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.steps[st.top * 100];
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Next countdown step: %s\n", &st.steps[st.top * 100]);  // Print the top step
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No steps to display.\n");
    } else {
        printf("Current Countdown:\n");
        for (int i = st.top; i >= 0; i--) {
```

```
            printf("%d: %s\n", i, &st.steps[i * 100]);
        }
    }
}

int search(struct Stack st, char *step) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.steps[i * 100], step) == 0) {
            return i;
        }
    }
    return -1;
}

void menu() {
    printf("\nLaunch Countdown Management\n");
    printf("1: Add a countdown step (push)\n");
    printf("2: Remove the last step (pop)\n");
    printf("3: Display the current countdown\n");
    printf("4: Peek at the next countdown step\n");
    printf("5: Search for a specific countdown step\n");
    printf("6: Exit\n");
}
```

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

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
```

```
int isEmpty(struct Stack);
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char log[100];
    char *poppedLog;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new maintenance log: ");
                scanf(" %s", log);
                push(&st, log);
                break;
            case 2:
```

```
            poppedLog = pop(&st);
            if (poppedLog != NULL)
                printf("Removed log: %s\n", poppedLog);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the maintenance log to search for: ");
            scanf(" %s", log);
            foundIndex = search(st, log);
            if (foundIndex != -1)
                printf("Log found at position: %d\n", foundIndex);
            else
                printf("Log not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
}

return 0;
```

```
}

void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->logs = (char *)malloc(st->size * 100 * sizeof(char));
}

void push(struct Stack *st, char *log) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new log.\n");
    } else {
        st->top++;
        strcpy(&st->logs[st->top * 100], log);
        printf("Log added: %s\n", log);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to remove.\n");
        return NULL;
    } else {
        return &st->logs[st->top-- * 100];
    }
}
```

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

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.logs[st.top * 100];
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Latest maintenance log: %s\n", &st.logs[st.top * 100]);  // Print the top log
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No logs to display.\n");
    } else {
        printf("Current Maintenance Logs:\n");
```

```
        for (int i = st.top; i >= 0; i--) {
            printf("%d: %s\n", i, &st.logs[i * 100]);
        }
    }
}

int search(struct Stack st, char *log) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.logs[i * 100], log) == 0) {
            return i;
        }
    }
    return -1;
}

void menu() {
    printf("\nAircraft Maintenance Logs\n");
    printf("1: Add a new log (push)\n");
    printf("2: Remove the last log (pop)\n");
    printf("3: View all maintenance logs\n");
    printf("4: Peek at the latest maintenance log\n");
    printf("5: Search for a specific maintenance log\n");
    printf("6: Exit\n");
}
```

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

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
int isEmpty(struct Stack);
```

```
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char step[100];
    char *poppedStep;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new docking step: ");
                scanf(" %s", step);
                push(&st, step);
                break;
            case 2:
                poppedStep = pop(&st);
```

```
            if (poppedStep != NULL)
                printf("Removed step: %s\n", poppedStep);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the docking step to search for: ");
            scanf(" %s", step);
            foundIndex = search(st, step);
            if (foundIndex != -1)
                printf("Step found at position: %d\n", foundIndex);
            else
                printf("Step not found in the procedure.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

```
void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->steps = (char *)malloc(st->size * 100 * sizeof(char));  // Allocate memory for all steps
}

void push(struct Stack *st, char *step) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new step.\n");
    } else {
        st->top++;
        strcpy(&st->steps[st->top * 100], step);  // Store each step in its allocated position
        printf("Step added: %s\n", step);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to remove.\n");
        return NULL;
    } else {
        return &st->steps[st->top-- * 100];  // Return the top step
    }
}

int isEmpty(struct Stack st) {
```

```
    return st.top == -1;
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.steps[st.top * 100];  // Return the top step
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Next docking step: %s\n", &st.steps[st.top * 100]);
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No steps to display.\n");
    } else {
        printf("Current Docking Procedure Steps:\n");
        for (int i = st.top; i >= 0; i--) {
```

```
            printf("%d: %s\n", i, &st.steps[i * 100]);
        }
    }
}

int search(struct Stack st, char *step) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.steps[i * 100], step) == 0) {
            return i;
        }
    }
    return -1;
}

void menu() {
    printf("\nSpacecraft Docking Procedure\n");
    printf("1: Push a new step\n");
    printf("2: Pop the last step\n");
    printf("3: Display the procedure steps\n");
    printf("4: Peek at the next step in the procedure\n");
    printf("5: Search for a specific step\n");
    printf("6: Exit\n");
}
```

```
/**********************************************************************
****
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>

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
```

```
int isEmpty(struct Stack);
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char command[100];
    char *poppedCommand;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
            poppedCommand = pop(&st);
            if (poppedCommand != NULL)
                printf("Undone command: %s\n", poppedCommand);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the command to search for: ");
            scanf(" %[^\n]s", command);
            foundIndex = search(st, command);
            if (foundIndex != -1)
                printf("Command found at position: %d\n", foundIndex);
            else
                printf("Command not found in the history.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
}

return 0;
```

```
}

void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->commands = (char *)malloc(st->size * 100 * sizeof(char));
}

void push(struct Stack *st, char *command) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new command.\n");
    } else {
        st->top++;
        strcpy(&st->commands[st->top * 100], command);
        printf("Command added: %s\n", command);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to undo.\n");
        return NULL;
    } else {
        return &st->commands[st->top-- * 100];
    }
}
```

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

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.commands[st.top * 100];
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Most recent command: %s\n", &st.commands[st.top * 100]);
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No commands to display.\n");
    } else {
        printf("Command History:\n");
```

```
        for (int i = st.top; i >= 0; i--) {
            printf("%d: %s\n", i, &st.commands[i * 100]);  // Print each command
        }
    }
}

int search(struct Stack st, char *command) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.commands[i * 100], command) == 0) {
            return i;
        }
    }
    return -1;
}

void menu() {
    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");
}
```

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

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
int isEmpty(struct Stack);
```

```
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char event[100];
    char *poppedEvent;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new event: ");
                scanf(" %s", event);
                push(&st, event);
                break;
            case 2:
                poppedEvent = pop(&st);
```

```
            if (poppedEvent != NULL)
                printf("Removed event: %s\n", poppedEvent);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the event to search for: ");
            scanf(" %s", event);
            foundIndex = search(st, event);
            if (foundIndex != -1)
                printf("Event found at position: %d\n", foundIndex);
            else
                printf("Event not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

```
void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->events = (char *)malloc(st->size * 100 * sizeof(char));
}

void push(struct Stack *st, char *event) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new event.\n");
    } else {
        st->top++;
        strcpy(&st->events[st->top * 100], event);
        printf("Event added: %s\n", event);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to remove.\n");
        return NULL;
    } else {
        return &st->events[st->top-- * 100];
    }
}

int isEmpty(struct Stack st) {
```

```
    return st.top == -1;
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.events[st.top * 100];
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Most recent event: %s\n", &st.events[st.top * 100]);
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No events to display.\n");
    } else {
        printf("Event History:\n");
        for (int i = st.top; i >= 0; i--) {
```

```
            printf("%d: %s\n", i, &st.events[i * 100]);
        }
    }
}

int search(struct Stack st, char *event) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.events[i * 100], event) == 0) {
            return i;
        }
    }
    return -1;
}

void menu() {
    printf("\nAerospace Simulation Events\n");
    printf("1: Push a new event\n");
    printf("2: Pop the last event\n");
    printf("3: Display all events\n");
    printf("4: Peek at the most recent event\n");
    printf("5: Search for a specific event\n");
    printf("6: Exit\n");
}
```

```
/*************************************************************************
****
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>

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
int isEmpty(struct Stack);
```

```
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char maneuver[100];
    char *poppedManeuver;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new maneuver: ");
                scanf(" %s", maneuver);
                push(&st, maneuver);
                break;
            case 2:
                poppedManeuver = pop(&st);
```

```
            if (poppedManeuver != NULL)
                printf("Removed maneuver: %s\n", poppedManeuver);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the maneuver to search for: ");
            scanf(" %s", maneuver);
            foundIndex = search(st, maneuver);
            if (foundIndex != -1)
                printf("Maneuver found at position: %d\n", foundIndex);
            else
                printf("Maneuver not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

```
void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->maneuvers = (char *)malloc(st->size * 100 * sizeof(char));
}

void push(struct Stack *st, char *maneuver) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new maneuver.\n");
    } else {
        st->top++;
        strcpy(&st->maneuvers[st->top * 100], maneuver);
        printf("Maneuver added: %s\n", maneuver);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to remove.\n");
        return NULL;
    } else {
        return &st->maneuvers[st->top-- * 100];
    }
}

int isEmpty(struct Stack st) {
```

```
    return st.top == -1;
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.maneuvers[st.top * 100];
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Most recent maneuver: %s\n", &st.maneuvers[st.top * 100]);
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No maneuvers to display.\n");
    } else {
        printf("Current Maneuvers:\n");
        for (int i = st.top; i >= 0; i--) {
```

```
            printf("%d: %s\n", i, &st.maneuvers[i * 100]);
        }
    }
}

int search(struct Stack st, char *maneuver) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.maneuvers[i * 100], maneuver) == 0) {
            return i;
        }
    }
    return -1;
}

void menu() {
    printf("\nPilot Training Maneuver Stack\n");
    printf("1: Add a maneuver (push)\n");
    printf("2: Remove the last maneuver (pop)\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");
}
```

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

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

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

```
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
   struct Stack st;
   create(&st);

   int choice;
   char command[100];
   char *poppedCommand;
   int foundIndex;

   while (1) {
      menu();
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the new command: ");
            scanf(" %[^\n]s", command);
            push(&st, command);
            break;
         case 2:
            poppedCommand = pop(&st);
            if (poppedCommand != NULL)
```

```
            printf("Undone command: %s\n", poppedCommand);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the command to search for: ");
            scanf(" %[^\n]s", command);
            foundIndex = search(st, command);
            if (foundIndex != -1)
                printf("Command found at position: %d\n", foundIndex);
            else
                printf("Command not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

```
void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->commands = (char *)malloc(st->size * 100 * sizeof(char));
}

void push(struct Stack *st, char *command) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new command.\n");
    } else {
        st->top++;
        strcpy(&st->commands[st->top * 100], command);
        printf("Command added: %s\n", command);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to undo.\n");
        return NULL;
    } else {
        return &st->commands[st->top-- * 100];
    }
}

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

```
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.commands[st.top * 100];
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Most recent command: %s\n", &st.commands[st.top * 100]);
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No commands to display.\n");
    } else {
        printf("Operation Commands:\n");
        for (int i = st.top; i >= 0; i--) {
            printf("%d: %s\n", i, &st.commands[i * 100]);
```

```
        }
    }
}

int search(struct Stack st, char *command) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.commands[i * 100], command) == 0) {
            return i;
        }
    }
    return -1;
}

void menu() {
    printf("\nSatellite Operation Commands\n");
    printf("1: Push a new command\n");
    printf("2: Pop the last command\n");
    printf("3: View the operation commands\n");
    printf("4: Peek at the most recent command\n");
    printf("5: Search for a specific command\n");
    printf("6: Exit\n");
}
```

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

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

void create(struct Stack *);
void display(struct Stack);
void push(struct Stack *, char *);
char* pop(struct Stack *);
int isEmpty(struct Stack);
```

```
int isFull(struct Stack);
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char procedure[100];
    char *poppedProcedure;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new procedure: ");
                scanf(" %s", procedure);
                push(&st, procedure);
                break;
            case 2:
                poppedProcedure = pop(&st);
```

```
            if (poppedProcedure != NULL)
                printf("Removed procedure: %s\n", poppedProcedure);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the procedure to search for: ");
            scanf(" %s", procedure);
            foundIndex = search(st, procedure);
            if (foundIndex != -1)
                printf("Procedure found at position: %d\n", foundIndex);
            else
                printf("Procedure not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}
```

```
void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->procedures = (char *)malloc(st->size * 100 * sizeof(char));
}

void push(struct Stack *st, char *procedure) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new procedure.\n");
    } else {
        st->top++;
        strcpy(&st->procedures[st->top * 100], procedure);
        printf("Procedure added: %s\n", procedure);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to remove.\n");
        return NULL;
    } else {
        return &st->procedures[st->top-- * 100];
    }
}

int isEmpty(struct Stack st) {
```

```
    return st.top == -1;
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.procedures[st.top * 100];
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Next procedure: %s\n", &st.procedures[st.top * 100]);
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No procedures to display.\n");
    } else {
        printf("Current Procedures:\n");
        for (int i = st.top; i >= 0; i--) {
```

```
            printf("%d: %s\n", i, &st.procedures[i * 100]);
        }
    }
}

int search(struct Stack st, char *procedure) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.procedures[i * 100], procedure) == 0) {
            return i;
        }
    }
    return -1;
}

void menu() {
    printf("\nEmergency Procedures for Spacecraft\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");
}
```

```
/*************************************************************************
****
```

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

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

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

```
char* stackTop(struct Stack);
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char activity[100];
    char *poppedActivity;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new activity: ");
                scanf(" %[^\n]s", activity);
                push(&st, activity);
                break;
            case 2:
                poppedActivity = pop(&st);
                if (poppedActivity != NULL)
```

```
            printf("Removed activity: %s\n", poppedActivity);
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the activity to search for: ");
            scanf(" %s", activity);
            foundIndex = search(st, activity);
            if (foundIndex != -1)
                printf("Activity found at position: %d\n", foundIndex);
            else
                printf("Activity not found in the log.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}
```

```
void create(struct Stack *st) {
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->activities = (char *)malloc(st->size * 100 * sizeof(char));  // Allocate memory for all activities
}

void push(struct Stack *st, char *activity) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new activity.\n");
    } else {
        st->top++;
        strcpy(&st->activities[st->top * 100], activity);  // Store each activity in its allocated position
        printf("Activity added: %s\n", activity);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to remove.\n");
        return NULL;
    } else {
        return &st->activities[st->top-- * 100];  // Return the top activity
    }
}

int isEmpty(struct Stack st) {
```

```
    return st.top == -1;
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.activities[st.top * 100];  // Return the top activity
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Most recent activity: %s\n", &st.activities[st.top * 100]);  // Print the top activity
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No activities to display.\n");
    } else {
        printf("Activity Log:\n");
        for (int i = st.top; i >= 0; i--) {
```

```
            printf("%d: %s\n", i, &st.activities[i * 100]);  // Print each activity
        }
    }
}

int search(struct Stack st, char *activity) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.activities[i * 100], activity) == 0) {
            return i;  // Return the index if found
        }
    }
    return -1;  // Return -1 if not found
}

void menu() {
    printf("\nAstronaut Activity Log\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");
}
```

```
/*********************************************************************
****
```

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>

#include <string.h>


struct Stack {

   int size;

   int top;

   char *fuelUsageEntries;

};


void create(struct Stack *);

void display(struct Stack);

void push(struct Stack *, char *);

char* pop(struct Stack *);

int isEmpty(struct Stack);

int isFull(struct Stack);

char* stackTop(struct Stack);
```

```
void peek(struct Stack);
int search(struct Stack, char *);
void menu();

int main() {
    struct Stack st;
    create(&st);

    int choice;
    char fuelUsageEntry[100];
    char *poppedEntry;
    int foundIndex;

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the new fuel usage entry: ");
                scanf(" %s", fuelUsageEntry);
                push(&st, fuelUsageEntry);
                break;
            case 2:
                poppedEntry = pop(&st);
                if (poppedEntry != NULL)
                    printf("Removed entry: %s\n", poppedEntry);
```

```
            break;
        case 3:
            display(st);
            break;
        case 4:
            peek(st);
            break;
        case 5:
            printf("Enter the fuel usage entry to search for: ");
            scanf(" %s", fuelUsageEntry);
            foundIndex = search(st, fuelUsageEntry);
            if (foundIndex != -1)
                printf("Entry found at position: %d\n", foundIndex);
            else
                printf("Entry not found in the stack.\n");
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

void create(struct Stack *st) {
```

```
    printf("Enter the size: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->fuelUsageEntries = (char *)malloc(st->size * 100 * sizeof(char));  // Allocate memory for all entries
}

void push(struct Stack *st, char *fuelUsageEntry) {
    if (isFull(*st)) {
        printf("Stack is full. Cannot add a new entry.\n");
    } else {
        st->top++;
        strcpy(&st->fuelUsageEntries[st->top * 100], fuelUsageEntry);  // Store each entry in its allocated position
        printf("Entry added: %s\n", fuelUsageEntry);
    }
}

char* pop(struct Stack *st) {
    if (isEmpty(*st)) {
        printf("Stack is empty. Nothing to remove.\n");
        return NULL;
    } else {
        return &st->fuelUsageEntries[st->top-- * 100];  // Return the top entry
    }
}

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

```
}

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

char* stackTop(struct Stack st) {
    if (!isEmpty(st)) {
        return &st.fuelUsageEntries[st.top * 100];  // Return the top entry
    }
    return NULL;
}

void peek(struct Stack st) {
    if (!isEmpty(st)) {
        printf("Latest fuel usage entry: %s\n", &st.fuelUsageEntries[st.top * 100]);  // Print the top entry
    } else {
        printf("Stack is empty. Nothing to peek.\n");
    }
}

void display(struct Stack st) {
    if (isEmpty(st)) {
        printf("Stack is empty. No entries to display.\n");
    } else {
        printf("Fuel Usage Data:\n");
        for (int i = st.top; i >= 0; i--) {
```

```
            printf("%d: %s\n", i, &st.fuelUsageEntries[i * 100]);  // Print each entry
        }
    }
}

int search(struct Stack st, char *fuelUsageEntry) {
    for (int i = st.top; i >= 0; i--) {
        if (strcmp(&st.fuelUsageEntries[i * 100], fuelUsageEntry) == 0) {
            return i;  // Return the index if found
        }
    }
    return -1;  // Return -1 if not found
}

void menu() {
    printf("\nFuel Management System\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");
}
```

STACK usinf linkedlist

```
#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;
```

```
}

int isEmpty(){
   return top == NULL;
}
```

//////////////////////

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

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

void push(int);
int pop();
void display();
int isEmpty();
int isFull();
int peek();

int main() {
   push(20);
```

```
    push(30);
    push(40);

    display();

    int poppedValue = pop();
    printf("Popped value = %d\n", poppedValue);
    printf("\n");

    display();

    int topValue = peek();
    if (topValue != -1) {
        printf("Top value = %d\n", topValue);
    }
    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\n", p->data);
        p = p->next;
    }
    printf("\n");
}

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

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

```
int isEmpty() {
   return top == NULL;
}

int isFull() {
   struct Node *t = (struct Node *)malloc(sizeof(struct Node));
   int full = t == NULL;
   free(t);
   return full;
}

int peek() {
   if (!isEmpty()) {
      return top->data;
   } else {
      printf("Stack is Empty\n");
      return -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>

struct Node {
   int orderID;
   char description[100];
   struct Node *next;
} *top = NULL;

void push(int, char *);

int pop();

void display();
```

```
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, orderID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the order ID: ");
                scanf("%d", &orderID);
                printf("Enter the order description: ");
                scanf(" %s", description);
                push(orderID, description);
                break;
            case 2:
                if (!isEmpty()) {
                    int processedOrder = pop();
                    printf("Processed order ID: %d\n", processedOrder);
                } else {
                    printf("No orders to process.\n");
```

```
    }
    break;
case 3:
    display();
    break;
case 4:
    if (!isEmpty()) {
        int nextOrder = peek();
        printf("Next order to be processed ID: %d\n", nextOrder);
    } else {
        printf("No orders to process.\n");
    }
    break;
case 5:
    printf("Enter the order ID to search for: ");
    scanf("%d", &orderID);
    foundIndex = search(orderID);
    if (foundIndex != -1) {
        printf("Order ID %d found at position: %d\n", orderID, foundIndex);
    } else {
        printf("Order ID %d not found.\n", orderID);
    }
    break;
case 6:
    printf("Exiting the system...\n");
    exit(0);
default:
    printf("Invalid choice! Please try again.\n");
```

```
        }
    }

    return 0;
}

// Function to add a new order
void push(int orderID, char *description) {
    struct Node *t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL) {
        printf("Stack is Full\n");
    } else {
        t->orderID = orderID;
        strcpy(t->description, description);
        t->next = top;
        top = t;
        printf("Order added: ID = %d, Description = %s\n", orderID, description);
    }
}

// Function to remove and return the last order
int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int orderID = t->orderID;
```

```
        top = top->next;
        free(t);
        return orderID;
    }
}

// Function to display all pending orders
void display() {
    if (isEmpty()) {
        printf("No pending orders.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Pending Orders:\n");
        while (p != NULL) {
            printf("Position: %d, Order ID: %d, Description: %s\n", position, p->orderID, p->description);
            p = p->next;
            position++;
        }
    }
}

// Function to check if the stack is empty
int isEmpty() {
    return top == NULL;
}
```

```
// Function to peek at the next order to be processed
int peek() {
    if (!isEmpty()) {
        return top->orderID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}

// Function to search for a specific order by its ID
int search(int orderID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->orderID == orderID) {
            return position;
        }
        p = p->next;
        position++;
    }
    return -1;
}

// Function to display the menu options
void menu() {

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

```
    printf("2: Process the last order (pop)\n");
    printf("3: Display all pending orders\n");
    printf("4: Peek at the next order to be processed\n");
    printf("5: Search for a specific order\n");
    printf("6: Exit\n");

}
```

```
/*****************************************

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>



struct Node {

   int ticketID;

   char description[100];

   struct Node *next;

} *top = NULL;



void push(int, char *);

int pop();

void display();
```

```
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, ticketID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the ticket ID: ");
                scanf("%d", &ticketID);
                printf("Enter the ticket description: ");
                scanf(" %[^\n]s", description);
                push(ticketID, description);
                break;
            case 2:
                if (!isEmpty()) {
                    int resolvedTicket = pop();
                    printf("Resolved ticket ID: %d\n", resolvedTicket);
                } else {
                    printf("No tickets to resolve.\n");
```

```
    }
    break;
case 3:
    display();
    break;
case 4:
    if (!isEmpty()) {
        int latestTicket = peek();
        printf("Latest ticket ID: %d\n", latestTicket);
    } else {
        printf("No tickets to peek.\n");
    }
    break;
case 5:
    printf("Enter the ticket ID to search for: ");
    scanf("%d", &ticketID);
    foundIndex = search(ticketID);
    if (foundIndex != -1) {
        printf("Ticket ID %d found at position: %d\n", ticketID, foundIndex);
    } else {
        printf("Ticket ID %d not found.\n", ticketID);
    }
    break;
case 6:
    printf("Exiting the system...\n");
    exit(0);
default:
    printf("Invalid choice! Please try again.\n");
```

```
        }
    }

    return 0;
}

// Function to add a new ticket to the stack
void push(int ticketID, char *description) {
    struct Node *t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL) {
        printf("Stack is Full\n");
    } else {
        t->ticketID = ticketID;
        strcpy(t->description, description);
        t->next = top;
        top = t;
        printf("Ticket added: ID = %d, Description = %s\n", ticketID, description);
    }
}

// Function to remove and return the last ticket
int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int ticketID = t->ticketID;
```

```
        top = top->next;
        free(t);
        return ticketID;
    }
}

// Function to display all pending tickets
void display() {
    if (isEmpty()) {
        printf("No pending tickets.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Pending Tickets:\n");
        while (p != NULL) {
            printf("Position: %d, Ticket ID: %d, Description: %s\n", position, p->ticketID, p->description);
            p = p->next;
            position++;
        }
    }
}

// Function to check if the stack is empty
int isEmpty() {
    return top == NULL;
}
```

```
// Function to peek at the latest ticket
int peek() {
    if (!isEmpty()) {
        return top->ticketID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}

// Function to search for a specific ticket by its ID
int search(int ticketID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->ticketID == ticketID) {
            return position;
        }
        p = p->next;
        position++;
    }
    return -1;
}

// Function to display the menu options
void menu() {

    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");
}
```

```
/*****************************************
```

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>
```

```
struct Node {
    int returnID;
    char description[100];
    struct Node *next;
} *top = NULL;


void push(int, char *);
int pop();
void display();
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, returnID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the return ID: ");
```

```
    scanf("%d", &returnID);
    printf("Enter the return description: ");
    scanf(" %[^\n]s", description);
    push(returnID, description);
    break;
case 2:
    if (!isEmpty()) {
        int processedReturn = pop();
        printf("Processed return ID: %d\n", processedReturn);
    } else {
        printf("No returns to process.\n");
    }
    break;
case 3:
    display();
    break;
case 4:
    if (!isEmpty()) {
        int nextReturn = peek();
        printf("Next return to process ID: %d\n", nextReturn);
    } else {
        printf("No returns to process.\n");
    }
    break;
case 5:
    printf("Enter the return ID to search for: ");
    scanf("%d", &returnID);
    foundIndex = search(returnID);
```

```
            if (foundIndex != -1) {
                printf("Return ID %d found at position: %d\n", returnID, foundIndex);
            } else {
                printf("Return ID %d not found.\n", returnID);
            }
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void push(int returnID, char *description) {
   struct Node *t = (struct Node *)malloc(sizeof(struct Node));
   if (t == NULL) {
      printf("Stack is Full\n");
   } else {
      t->returnID = returnID;
      strcpy(t->description, description);
      t->next = top;
      top = t;
      printf("Return request added: ID = %d, Description = %s\n", returnID, description);
```

```
    }
}



int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int returnID = t->returnID;
        top = top->next;
        free(t);
        return returnID;
    }
}



void display() {
    if (isEmpty()) {
        printf("No pending return requests.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Pending Return Requests:\n");
        while (p != NULL) {
            printf("Position: %d, Return ID: %d, Description: %s\n", position, p->returnID, p-
>description);
```

```
            p = p->next;
            position++;
        }
    }
}


int isEmpty() {
    return top == NULL;
}


int peek() {
    if (!isEmpty()) {
        return top->returnID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int returnID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->returnID == returnID) {
            return position;
```

```
        }
        p = p->next;
        position++;
    }
    return -1;
}



void menu() {

    printf("1: Add a new return request (push)\n");
    printf("2: Process the last return (pop)\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");
}
```

```
/*****************************************

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>



struct Node {

   int restockID;

   char description[100];

   struct Node *next;

} *top = NULL;



void push(int, char *);

int pop();

void display();
```

```
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, restockID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the restock ID: ");
                scanf("%d", &restockID);
                printf("Enter the restock description: ");
                scanf(" %[^\n]s", description);
                push(restockID, description);
                break;
            case 2:
                if (!isEmpty()) {
                    int processedRestock = pop();
                    printf("Processed restock ID: %d\n", processedRestock);
                } else {
                    printf("No restocks to process.\n");
```

```
    }
    break;
case 3:
    display();
    break;
case 4:
    if (!isEmpty()) {
        int latestRestock = peek();
        printf("Latest restock ID: %d\n", latestRestock);
    } else {
        printf("No restocks to peek.\n");
    }
    break;
case 5:
    printf("Enter the restock ID to search for: ");
    scanf("%d", &restockID);
    foundIndex = search(restockID);
    if (foundIndex != -1) {
        printf("Restock ID %d found at position: %d\n", restockID, foundIndex);
    } else {
        printf("Restock ID %d not found.\n", restockID);
    }
    break;
case 6:
    printf("Exiting the system...\n");
    exit(0);
default:
    printf("Invalid choice! Please try again.\n");
```

```
        }
    }

    return 0;
}


void push(int restockID, char *description) {
    struct Node *t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL) {
        printf("Stack is Full\n");
    } else {
        t->restockID = restockID;
        strcpy(t->description, description);
        t->next = top;
        top = t;
        printf("Restock entry added: ID = %d, Description = %s\n", restockID, description);
    }
}


int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int restockID = t->restockID;
```

```
        top = top->next;
        free(t);
        return restockID;
    }
}


void display() {
    if (isEmpty()) {
        printf("No pending restock entries.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Pending Restock Entries:\n");
        while (p != NULL) {
            printf("Position: %d, Restock ID: %d, Description: %s\n", position, p->restockID, p->description);
            p = p->next;
            position++;
        }
    }
}


int isEmpty() {
    return top == NULL;
}
```

```
int peek() {
    if (!isEmpty()) {
        return top->restockID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int restockID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->restockID == restockID) {
            return position;
        }
        p = p->next;
        position++;
    }
    return -1;
}


void menu() {

    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");
}



/*******************************************

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>



struct Node {

    int dealID;
```

```
    char description[100];
    struct Node *next;
} *top = NULL;



void push(int, char *);
int pop();
void display();
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, dealID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the deal ID: ");
                scanf("%d", &dealID);
                printf("Enter the deal description: ");
                scanf(" %[^\n]s", description);
```

```
        push(dealID, description);
        break;
    case 2:
        if (!isEmpty()) {
            int removedDeal = pop();
            printf("Removed deal ID: %d\n", removedDeal);
        } else {
            printf("No deals to remove.\n");
        }
        break;
    case 3:
        display();
        break;
    case 4:
        if (!isEmpty()) {
            int latestDeal = peek();
            printf("Latest deal ID: %d\n", latestDeal);
        } else {
            printf("No deals to peek.\n");
        }
        break;
    case 5:
        printf("Enter the deal ID to search for: ");
        scanf("%d", &dealID);
        foundIndex = search(dealID);
        if (foundIndex != -1) {
            printf("Deal ID %d found at position: %d\n", dealID, foundIndex);
        } else {
```

```
                printf("Deal ID %d not found.\n", dealID);
            }
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void push(int dealID, char *description) {
   struct Node *t = (struct Node *)malloc(sizeof(struct Node));
   if (t == NULL) {
      printf("Stack is Full\n");
   } else {
      t->dealID = dealID;
      strcpy(t->description, description);
      t->next = top;
      top = t;
      printf("Deal added: ID = %d, Description = %s\n", dealID, description);
   }
}
```

```
int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int dealID = t->dealID;
        top = top->next;
        free(t);
        return dealID;
    }
}



void display() {
    if (isEmpty()) {
        printf("No active deals.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Active Deals:\n");
        while (p != NULL) {
            printf("Position: %d, Deal ID: %d, Description: %s\n", position, p->dealID, p-
>description);
            p = p->next;
            position++;
        }
```

```
    }
}


int isEmpty() {
    return top == NULL;
}

int peek() {
    if (!isEmpty()) {
        return top->dealID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int dealID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->dealID == dealID) {
            return position;
        }
        p = p->next;
        position++;
    }
```

```
    return -1;
}



void menu() {

    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");
}
```

```
/*****************************************

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>



struct Node {

   int sessionID;

   char description[100];

   struct Node *next;

} *top = NULL;



void push(int, char *);

int pop();

void display();

int isEmpty();
```

```
int peek();
int search(int);
void menu();

int main() {
    int choice, sessionID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the session ID: ");
                scanf("%d", &sessionID);
                printf("Enter the session description: ");
                scanf(" %[^\n]s", description);
                push(sessionID, description);
                break;
            case 2:
                if (!isEmpty()) {
                    int endedSession = pop();
                    printf("Ended session ID: %d\n", endedSession);
                } else {
                    printf("No sessions to end.\n");
                }
```

```
            break;
        case 3:
            display();
            break;
        case 4:
            if (!isEmpty()) {
                int recentSession = peek();
                printf("Most recent session ID: %d\n", recentSession);
            } else {
                printf("No sessions to peek.\n");
            }
            break;
        case 5:
            printf("Enter the session ID to search for: ");
            scanf("%d", &sessionID);
            foundIndex = search(sessionID);
            if (foundIndex != -1) {
                printf("Session ID %d found at position: %d\n", sessionID, foundIndex);
            } else {
                printf("Session ID %d not found.\n", sessionID);
            }
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
```

```
    }

    return 0;
}

void push(int sessionID, char *description) {
    struct Node *t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL) {
        printf("Stack is Full\n");
    } else {
        t->sessionID = sessionID;
        strcpy(t->description, description);
        t->next = top;
        top = t;
        printf("Session added: ID = %d, Description = %s\n", sessionID, description);
    }
}


int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int sessionID = t->sessionID;
        top = top->next;
        free(t);
```

```
        return sessionID;
    }
}



void display() {
    if (isEmpty()) {
        printf("No active sessions.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Active Sessions:\n");
        while (p != NULL) {
            printf("Position: %d, Session ID: %d, Description: %s\n", position, p->sessionID, p->description);
            p = p->next;
            position++;
        }
    }
}



int isEmpty() {
    return top == NULL;
}



int peek() {
```

```
    if (!isEmpty()) {
        return top->sessionID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int sessionID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->sessionID == sessionID) {
            return position;
        }
        p = p->next;
        position++;
    }
    return -1;
}


void menu() {

    printf("\nUser Session History\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");
}
```

```
/*****************************************
```

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>

struct Node {
```

```
    int productID;
    char description[100];
    struct Node *next;
} *top = NULL;


void push(int, char *);
int pop();
void display();
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, productID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the product ID: ");
                scanf("%d", &productID);
                printf("Enter the product description: ");
```

```
            scanf(" %[^\n]s", description);
            push(productID, description);
            break;
        case 2:
            if (!isEmpty()) {
                int removedProduct = pop();
                printf("Removed product ID: %d\n", removedProduct);
            } else {
                printf("No products to remove.\n");
            }
            break;
        case 3:
            display();
            break;
        case 4:
            if (!isEmpty()) {
                int recentProduct = peek();
                printf("Most recent product ID: %d\n", recentProduct);
            } else {
                printf("No products to peek.\n");
            }
            break;
        case 5:
            printf("Enter the product ID to search for: ");
            scanf("%d", &productID);
            foundIndex = search(productID);
            if (foundIndex != -1) {
                printf("Product ID %d found at position: %d\n", productID, foundIndex);
```

```
            } else {
                printf("Product ID %d not found.\n", productID);
            }
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void push(int productID, char *description) {
   struct Node *t = (struct Node *)malloc(sizeof(struct Node));
   if (t == NULL) {
      printf("Stack is Full\n");
   } else {
      t->productID = productID;
      strcpy(t->description, description);
      t->next = top;
      top = t;
      printf("Product added: ID = %d, Description = %s\n", productID, description);
   }
}
```

```
int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int productID = t->productID;
        top = top->next;
        free(t);
        return productID;
    }
}


void display() {
    if (isEmpty()) {
        printf("No items in the wishlist.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Wishlist Items:\n");
        while (p != NULL) {
            printf("Position: %d, Product ID: %d, Description: %s\n", position, p->productID, p->description);
            p = p->next;
            position++;
```

```
        }
    }
}


int isEmpty() {
    return top == NULL;
}


int peek() {
    if (!isEmpty()) {
        return top->productID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int productID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->productID == productID) {
            return position;
        }
        p = p->next;
```

```
        position++;
    }
    return -1;
}



void menu() {

    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");
}
```

```
/*****************************************

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>



struct Node {

   int stepID;

   char description[100];

   struct Node *next;

} *top = NULL;



void push(int, char *);

int pop();

void display();

int isEmpty();
```

```
int peek();
int search(int);
void menu();

int main() {
    int choice, stepID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the step ID: ");
                scanf("%d", &stepID);
                printf("Enter the step description: ");
                scanf(" %s", description);
                push(stepID, description);
                break;
            case 2:
                if (!isEmpty()) {
                    int removedStep = pop();
                    printf("Removed step ID: %d\n", removedStep);
                } else {
                    printf("No steps to remove.\n");
                }
```

```
            break;
        case 3:
            display();
            break;
        case 4:
            if (!isEmpty()) {
                int currentStep = peek();
                printf("Current step ID: %d\n", currentStep);
            } else {
                printf("No steps to peek.\n");
            }
            break;
        case 5:
            printf("Enter the step ID to search for: ");
            scanf("%d", &stepID);
            foundIndex = search(stepID);
            if (foundIndex != -1) {
                printf("Step ID %d found at position: %d\n", stepID, foundIndex);
            } else {
                printf("Step ID %d not found.\n", stepID);
            }
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
```

```
    }

    return 0;
}

// Function to add a new step to the stack
void push(int stepID, char *description) {
    struct Node *t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL) {
        printf("Stack is Full\n");
    } else {
        t->stepID = stepID;
        strcpy(t->description, description);
        t->next = top;
        top = t;
        printf("Step added: ID = %d, Description = %s\n", stepID, description);
    }
}

// Function to remove and return the last step from the stack
int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int stepID = t->stepID;
        top = top->next;
```

```
        free(t);
        return stepID;
    }
}

// Function to display all checkout steps
void display() {
    if (isEmpty()) {
        printf("No steps in the checkout process.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Checkout Process Steps:\n");
        while (p != NULL) {
            printf("Position: %d, Step ID: %d, Description: %s\n", position, p->stepID, p->description);
            p = p->next;
            position++;
        }
    }
}

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

// Function to peek at the current step
```

```
int peek() {
    if (!isEmpty()) {
        return top->stepID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}

// Function to search for a specific step by its ID
int search(int stepID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->stepID == stepID) {
            return position;
        }
        p = p->next;
        position++;
    }
    return -1;
}


void menu() {

    printf("1: Add a checkout step (push)\n");
    printf("2: Remove the last step (pop)\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");
}
```

```
/*******************************************

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>


struct Node {
    int couponID;
```

```
    char description[100];
    struct Node *next;
} *top = NULL;



void push(int, char *);
int pop();
void display();
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, couponID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the coupon ID: ");
                scanf("%d", &couponID);
                printf("Enter the coupon description: ");
                scanf(" %s", description);
```

```
        push(couponID, description);
        break;
    case 2:
        if (!isEmpty()) {
            int removedCoupon = pop();
            printf("Removed coupon ID: %d\n", removedCoupon);
        } else {
            printf("No coupons to remove.\n");
        }
        break;
    case 3:
        display();
        break;
    case 4:
        if (!isEmpty()) {
            int latestCoupon = peek();
            printf("Latest coupon ID: %d\n", latestCoupon);
        } else {
            printf("No coupons to peek.\n");
        }
        break;
    case 5:
        printf("Enter the coupon ID to search for: ");
        scanf("%d", &couponID);
        foundIndex = search(couponID);
        if (foundIndex != -1) {
            printf("Coupon ID %d found at position: %d\n", couponID, foundIndex);
        } else {
```

```
                printf("Coupon ID %d not found.\n", couponID);
            }
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}

// Function to add a new coupon code
void push(int couponID, char *description) {
   struct Node *t = (struct Node *)malloc(sizeof(struct Node));
   if (t == NULL) {
      printf("Stack is Full\n");
   } else {
      t->couponID = couponID;
      strcpy(t->description, description);
      t->next = top;
      top = t;
      printf("Coupon added: ID = %d, Description = %s\n", couponID, description);
   }
}
```

```
// Function to remove and return the last coupon code
int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int couponID = t->couponID;
        top = top->next;
        free(t);
        return couponID;
    }
}



void display() {
    if (isEmpty()) {
        printf("No available coupon codes.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Available Coupon Codes:\n");
        while (p != NULL) {
            printf("Position: %d, Coupon ID: %d, Description: %s\n", position, p->couponID, p->description);
            p = p->next;
            position++;
        }
```

```
    }
}


int isEmpty() {
    return top == NULL;
}


int peek() {
    if (!isEmpty()) {
        return top->couponID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int couponID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->couponID == couponID) {
            return position;
        }
        p = p->next;
        position++;
```

```
    }
    return -1;
}



void menu() {

    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");
}
```

```
/*****************************************

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>



struct Node {
    int updateID;
    char description[100];
    struct Node *next;
} *top = NULL;



void push(int, char *);
int pop();
```

```
void display();
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, updateID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the update ID: ");
                scanf("%d", &updateID);
                printf("Enter the update description: ");
                scanf(" %s", description);
                push(updateID, description);
                break;
            case 2:
                if (!isEmpty()) {
                    int removedUpdate = pop();
                    printf("Removed update ID: %d\n", removedUpdate);
                } else {
```

```
            printf("No updates to remove.\n");
        }
        break;
    case 3:
        display();
        break;
    case 4:
        if (!isEmpty()) {
            int latestUpdate = peek();
            printf("Latest update ID: %d\n", latestUpdate);
        } else {
            printf("No updates to peek.\n");
        }
        break;
    case 5:
        printf("Enter the update ID to search for: ");
        scanf("%d", &updateID);
        foundIndex = search(updateID);
        if (foundIndex != -1) {
            printf("Update ID %d found at position: %d\n", updateID, foundIndex);
        } else {
            printf("Update ID %d not found.\n", updateID);
        }
        break;
    case 6:
        printf("Exiting the system...\n");
        exit(0);
    default:
```

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

    return 0;
}

// Function to add a new shipping status
void push(int updateID, char *description) {
    struct Node *t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL) {
        printf("Stack is Full\n");
    } else {
        t->updateID = updateID;
        strcpy(t->description, description);
        t->next = top;
        top = t;
        printf("Update added: ID = %d, Description = %s\n", updateID, description);
    }
}

// Function to remove and return the last shipping status
int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
```

```
        int updateID = t->updateID;
        top = top->next;
        free(t);
        return updateID;
    }
}

// Function to display all shipping status updates
void display() {
    if (isEmpty()) {
        printf("No shipping status updates.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Shipping Status Updates:\n");
        while (p != NULL) {
            printf("Position: %d, Update ID: %d, Description: %s\n", position, p->updateID, p->description);
            p = p->next;
            position++;
        }
    }
}

int isEmpty() {
    return top == NULL;
}
```

```
int peek() {
    if (!isEmpty()) {
        return top->updateID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int updateID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->updateID == updateID) {
            return position;
        }
        p = p->next;
        position++;
    }
    return -1;
}


void menu() {
```

```
    printf("1: Add a shipping status update (push)\n");
    printf("2: Remove the last update (pop)\n");
    printf("3: View all shipping status updates\n");
    printf("4: Peek at the latest update\n");
    printf("5: Search for a specific update\n");
    printf("6: Exit\n");
}
```

```
/*****************************************

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>
```

```
struct Node {
    int reviewID;
    char description[200];
    struct Node *next;
} *top = NULL;



void push(int, char *);
int pop();
void display();
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, reviewID, foundIndex;
    char description[200];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
```

```
        printf("Enter the review ID: ");
        scanf("%d", &reviewID);
        printf("Enter the review description: ");
        scanf(" %s", description);
        push(reviewID, description);
        break;
    case 2:
        if (!isEmpty()) {
            int removedReview = pop();
            printf("Removed review ID: %d\n", removedReview);
        } else {
            printf("No reviews to remove.\n");
        }
        break;
    case 3:
        display();
        break;
    case 4:
        if (!isEmpty()) {
            int latestReview = peek();
            printf("Latest review ID: %d\n", latestReview);
        } else {
            printf("No reviews to peek.\n");
        }
        break;
    case 5:
        printf("Enter the review ID to search for: ");
        scanf("%d", &reviewID);
```

```
            foundIndex = search(reviewID);
            if (foundIndex != -1) {
                printf("Review ID %d found at position: %d\n", reviewID, foundIndex);
            } else {
                printf("Review ID %d not found.\n", reviewID);
            }
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }

   return 0;
}



void push(int reviewID, char *description) {
   struct Node *t = (struct Node *)malloc(sizeof(struct Node));
   if (t == NULL) {
      printf("Stack is Full\n");
   } else {
      t->reviewID = reviewID;
      strcpy(t->description, description);
      t->next = top;
      top = t;
```

```
        printf("Review added: ID = %d, Description = %s\n", reviewID, description);
    }
}



int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int reviewID = t->reviewID;
        top = top->next;
        free(t);
        return reviewID;
    }
}



void display() {
    if (isEmpty()) {
        printf("No reviews available.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Reviews:\n");
        while (p != NULL) {
```

```
            printf("Position: %d, Review ID: %d, Description: %s\n", position, p->reviewID, p->description);
            p = p->next;
            position++;
        }
    }
}


int isEmpty() {
    return top == NULL;
}

int peek() {
    if (!isEmpty()) {
        return top->reviewID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int reviewID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->reviewID == reviewID) {
```

```
            return position;
        }
        p = p->next;
        position++;
    }
    return -1;
}



void menu() {

    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");
}
```

```
/*****************************************

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>



struct Node {

    int notificationID;

    char description[100];

    struct Node *next;

} *top = NULL;



void push(int, char *);

int pop();
```

```
void display();
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, notificationID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the notification ID: ");
                scanf("%d", &notificationID);
                printf("Enter the notification description: ");
                scanf(" %[^\n]s", description);
                push(notificationID, description);
                break;
            case 2:
                if (!isEmpty()) {
                    int removedNotification = pop();
                    printf("Removed notification ID: %d\n", removedNotification);
                } else {
```

```
            printf("No notifications to remove.\n");
        }
        break;
    case 3:
        display();
        break;
    case 4:
        if (!isEmpty()) {
            int latestNotification = peek();
            printf("Latest notification ID: %d\n", latestNotification);
        } else {
            printf("No notifications to peek.\n");
        }
        break;
    case 5:
        printf("Enter the notification ID to search for: ");
        scanf("%d", &notificationID);
        foundIndex = search(notificationID);
        if (foundIndex != -1) {
            printf("Notification ID %d found at position: %d\n", notificationID, foundIndex);
        } else {
            printf("Notification ID %d not found.\n", notificationID);
        }
        break;
    case 6:
        printf("Exiting the system...\n");
        exit(0);
    default:
```

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

    return 0;
}



void push(int notificationID, char *description) {
    struct Node *t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL) {
        printf("Stack is Full\n");
    } else {
        t->notificationID = notificationID;
        strcpy(t->description, description);
        t->next = top;
        top = t;
        printf("Notification added: ID = %d, Description = %s\n", notificationID, description);
    }
}



int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
```

```
        int notificationID = t->notificationID;
        top = top->next;
        free(t);
        return notificationID;
    }
}


void display() {
    if (isEmpty()) {
        printf("No notifications available.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Notifications:\n");
        while (p != NULL) {
            printf("Position: %d, Notification ID: %d, Description: %s\n", position, p-
>notificationID, p->description);
            p = p->next;
            position++;
        }
    }
}


int isEmpty() {
    return top == NULL;
}
```

```
int peek() {
    if (!isEmpty()) {
        return top->notificationID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int notificationID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->notificationID == notificationID) {
            return position;
        }
        p = p->next;
        position++;
    }
    return -1;
}


void menu() {
```

```
    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");
}



/*******************************************
```

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>
```

```
struct Node {
    int productID;
    char description[100];
    struct Node *next;
} *top = NULL;


void push(int, char *);
int pop();
void display();
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
    int choice, productID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the product ID: ");
                scanf("%d", &productID);
```

```
            printf("Enter the product description: ");
            scanf(" %[^\n]s", description);
            push(productID, description);
            break;
        case 2:
            if (!isEmpty()) {
                int removedProduct = pop();
                printf("Removed product ID: %d\n", removedProduct);
            } else {
                printf("No products to remove.\n");
            }
            break;
        case 3:
            display();
            break;
        case 4:
            if (!isEmpty()) {
                int recentProduct = peek();
                printf("Most recent product ID: %d\n", recentProduct);
            } else {
                printf("No products to peek.\n");
            }
            break;
        case 5:
            printf("Enter the product ID to search for: ");
            scanf("%d", &productID);
            foundIndex = search(productID);
            if (foundIndex != -1) {
```

```
            printf("Product ID %d found at position: %d\n", productID, foundIndex);
        } else {
            printf("Product ID %d not found.\n", productID);
        }
        break;
      case 6:
        printf("Exiting the system...\n");
        exit(0);
      default:
        printf("Invalid choice! Please try again.\n");
    }
  }

  return 0;
}



void push(int productID, char *description) {
  struct Node *t = (struct Node *)malloc(sizeof(struct Node));
  if (t == NULL) {
    printf("Stack is Full\n");
  } else {
    t->productID = productID;
    strcpy(t->description, description);
    t->next = top;
    top = t;
    printf("Product added: ID = %d, Description = %s\n", productID, description);
  }
```

```
}



int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int productID = t->productID;
        top = top->next;
        free(t);
        return productID;
    }
}



void display() {
    if (isEmpty()) {
        printf("No viewed products.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Viewed Products:\n");
        while (p != NULL) {
            printf("Position: %d, Product ID: %d, Description: %s\n", position, p->productID, p-
>description);
            p = p->next;
```

```
            position++;
        }
    }
}


int isEmpty() {
    return top == NULL;
}


int peek() {
    if (!isEmpty()) {
        return top->productID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int productID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->productID == productID) {
            return position;
        }
```

```
        p = p->next;
        position++;
    }
    return -1;
}



void menu() {

    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");
}
```

```
/*****************************************

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>

struct Node {

   int itemID;

   char description[100];

   struct Node *next;

} *top = NULL;



void push(int, char *);

int pop();

void display();

int isEmpty();

int peek();
```

```
int search(int);
void menu();

int main() {
    int choice, itemID, foundIndex;
    char description[100];

    while (1) {
        menu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the item ID: ");
                scanf("%d", &itemID);
                printf("Enter the item description: ");
                scanf(" %[^\n]s", description);
                push(itemID, description);
                break;
            case 2:
                if (!isEmpty()) {
                    int removedItem = pop();
                    printf("Removed item ID: %d\n", removedItem);
                } else {
                    printf("No items to remove.\n");
                }
                break;
```

```
        case 3:
            display();
            break;
        case 4:
            if (!isEmpty()) {
                int recentItem = peek();
                printf("Most recent item ID: %d\n", recentItem);
            } else {
                printf("No items to peek.\n");
            }
            break;
        case 5:
            printf("Enter the item ID to search for: ");
            scanf("%d", &itemID);
            foundIndex = search(itemID);
            if (foundIndex != -1) {
                printf("Item ID %d found at position: %d\n", itemID, foundIndex);
            } else {
                printf("Item ID %d not found.\n", itemID);
            }
            break;
        case 6:
            printf("Exiting the system...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
}
```

```
    return 0;
}



void push(int itemID, char *description) {
    struct Node *t = (struct Node *)malloc(sizeof(struct Node));
    if (t == NULL) {
        printf("Stack is Full\n");
    } else {
        t->itemID = itemID;
        strcpy(t->description, description);
        t->next = top;
        top = t;
        printf("Item added: ID = %d, Description = %s\n", itemID, description);
    }
}



int pop() {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int itemID = t->itemID;
        top = top->next;
        free(t);
```

```
        return itemID;
    }
}



void display() {
    if (isEmpty()) {
        printf("No items in the cart.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Cart Items:\n");
        while (p != NULL) {
            printf("Position: %d, Item ID: %d, Description: %s\n", position, p->itemID, p->description);
            p = p->next;
            position++;
        }
    }
}



int isEmpty() {
    return top == NULL;
}



int peek() {
```

```
    if (!isEmpty()) {
        return top->itemID;
    } else {
        printf("Stack is Empty\n");
        return -1;
    }
}


int search(int itemID) {
    struct Node *p = top;
    int position = 0;
    while (p != NULL) {
        if (p->itemID == itemID) {
            return position;
        }
        p = p->next;
        position++;
    }
    return -1;
}


void menu() {

    printf("1: Add an item to the cart (push)\n");
    printf("2: Remove the last item (pop)\n");
    printf("3: View all cart items\n");
```

```
    printf("4: Peek at the last added item\n");
    printf("5: Search for a specific item in the cart\n");
    printf("6: Exit\n");
}
```

```
/*****************************************
```

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>

struct Node {
    int paymentID;
    char description[100];
    struct Node *next;
```

```
} *top = NULL;



void push(int, char *);
int pop();
void display();
int isEmpty();
int peek();
int search(int);
void menu();

int main() {
   int choice, paymentID, foundIndex;
   char description[100];

   while (1) {
      menu();
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the payment ID: ");
            scanf("%d", &paymentID);
            printf("Enter the payment description: ");
            scanf(" %[^\n]s", description);
            push(paymentID, description);
            break;
```

```
case 2:
    if (!isEmpty()) {
        int removedPayment = pop();
        printf("Removed payment ID: %d\n", removedPayment);
    } else {
        printf("No payment records to remove.\n");
    }
    break;
case 3:
    display();
    break;
case 4:
    if (!isEmpty()) {
        int latestPayment = peek();
        printf("Latest payment ID: %d\n", latestPayment);
    } else {
        printf("No payment records to peek.\n");
    }
    break;
case 5:
    printf("Enter the payment ID to search for: ");
    scanf("%d", &paymentID);
    foundIndex = search(paymentID);
    if (foundIndex != -1) {
        printf("Payment ID %d found at position: %d\n", paymentID, foundIndex);
    } else {
        printf("Payment ID %d not found.\n", paymentID);
    }
```

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

   return 0;
}



void push(int paymentID, char *description) {
   struct Node *t = (struct Node *)malloc(sizeof(struct Node));
   if (t == NULL) {
      printf("Stack is Full\n");
   } else {
      t->paymentID = paymentID;
      strcpy(t->description, description);
      t->next = top;
      top = t;
      printf("Payment added: ID = %d, Description = %s\n", paymentID, description);
   }
}



int pop() {
```

```
    if (top == NULL) {
        printf("Stack is Empty\n");
        return -1;
    } else {
        struct Node *t = top;
        int paymentID = t->paymentID;
        top = top->next;
        free(t);
        return paymentID;
    }
}


void display() {
    if (isEmpty()) {
        printf("No payment records available.\n");
    } else {
        struct Node *p = top;
        int position = 0;
        printf("Payment Records:\n");
        while (p != NULL) {
            printf("Position: %d, Payment ID: %d, Description: %s\n", position, p->paymentID, p->description);
            p = p->next;
            position++;
        }
    }
}
```

```
int isEmpty() {
   return top == NULL;
}

int peek() {
   if (!isEmpty()) {
      return top->paymentID;
   } else {
      printf("Stack is Empty\n");
      return -1;
   }
}


int search(int paymentID) {
   struct Node *p = top;
   int position = 0;
   while (p != NULL) {
      if (p->paymentID == paymentID) {
         return position;
      }
      p = p->next;
      position++;
   }
   return -1;
}
```

```
void menu() {

    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");
}
```