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

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

#define MAX_PATH_LENGTH 100

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

void create(struct Stack *st);
void push(struct Stack *st, const char *path);
int pop(struct Stack *st);
```

```
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *path);

int main()
{
   struct Stack st;
   create(&st);
   int option;
   char path[MAX_PATH_LENGTH];
   int index;
   do
   {
      printf("\n--- Flight Path Logging System ---\n");
      printf("1: Add a new path (push)\n");
      printf("2: Undo the last path (pop)\n");
      printf("3: Display the current flight path stack\n");
      printf("4: Peek at the top path\n");
      printf("5: Search for a specific path\n");
      printf("6: Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
      case 1: printf("Enter the flight path: ");
            scanf(" %[^\n]", path);
            push(&st, path);
```

```
            break;
        case 2: pop(&st);
            break;
        case 3: display(st);
            break;
        case 4: printf("Enter the index to peek: ");
            scanf("%d", &index);
            if (peek(st, index) != -1)
               printf("Path at index %d: %s\n", index, st.s[st.top - index +
1]);
            break;
        case 5: printf("Enter the path to search: ");
            scanf(" %[^\n]", path);
            int position = search(st, path);
            if (position != -1)
               printf("Path found at index %d\n", position);
            else
               printf("Path not found\n");
            break;
        case 6: printf("Exiting the system\n");
            for (int i = 0; i <= st.top; i++)
               free(st.s[i]);
            free(st.s);
            break;
        default: printf("Invalid option\n");
        }
    } while(option != 6);
    return 0;
```

```
}

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

void push(struct Stack *st, const char *path)
{
   if (st->top == st->size - 1)
      printf("Stack Overflow\n");
   else
   {
      st->top++;
      st->s[st->top] = (char *)malloc(MAX_PATH_LENGTH
                                      * sizeof(char));
      strcpy(st->s[st->top], path);
      printf("Path added: %s\n", path);
   }
}

int pop(struct Stack *st)
{
   if (st->top == -1)
```

```
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
        printf("Undoing path: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if (st.top == -1)
        printf("Stack is empty\n");
    else
    {
        printf("Current flight path stack:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

int peek(struct Stack st, int index)
{
```

```
    if (index < 1 || index > st.top + 1)
    {
        printf("Invalid index\n");
        return -1;
    }
    return 0;
}

int search(struct Stack st, const char *path)
{
    for (int i = st.top; i >= 0; i--)
    {
        if (strcmp(st.s[i], path) == 0)
            return i + 1;
    }
    return -1;
}
```

2. Satellite Deployment Sequence: Develop a stack using arrays to manage the sequence of satellite deployments from a spacecraft. Include a switch-case menu with options:
    - 1: Push a new satellite deployment
    - 2: Pop the last deployment
    - 3: View the deployment sequence
    - 4: Peek at the latest deployment
    - 5: Search for a specific deployment
    - 6: Exit

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

#define MAX_SAT_NAME_LENGTH 100

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

void create(struct Stack *st);
void push(struct Stack *st, const char *satellite);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *satellite);

int main()
{
   struct Stack st;
   create(&st);
   int option;
   char satellite[MAX_SAT_NAME_LENGTH];
   int index;
```

```
    do
    {
        printf("\n--- Satellite Deployment Sequence ---\n");
        printf("1: Push a new satellite deployment\n");
        printf("2: Pop the last deployment\n");
        printf("3: View the deployment sequence\n");
        printf("4: Peek at the latest deployment\n");
        printf("5: Search for a specific deployment\n");
        printf("6: Exit\n");
        printf("Enter your option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the satellite name: ");
                    scanf(" %[^\n]", satellite);
                    push(&st, satellite);
                    break;
            case 2: pop(&st);
                    break;
            case 3: display(st);
                    break;
            case 4: printf("Enter the index to peek: ");
                    scanf("%d", &index);
                    if (peek(st, index) != -1)
                        printf("Satellite at index %d: %s\n", index, st.s[st.top -
index + 1]);
                    break;
            case 5: printf("Enter the satellite name to search: ");
```

```
                scanf(" %[^\n]", satellite);
                int position = search(st, satellite);
                if (position != -1)
                    printf("Satellite found at index %d\n", position);
                else
                    printf("Satellite not found\n");
                break;
            case 6: printf("Exiting the sequence\n");
                for (int i = 0; i <= st.top; i++)
                    free(st.s[i]);
                free(st.s);
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

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

void push(struct Stack *st, const char *satellite)
```

```
{
   if (st->top == st->size - 1)
      printf("Stack Overflow\n");
   else
   {
      st->top++;
      st->s[st->top] = (char *)malloc(MAX_SAT_NAME_LENGTH
                                                * sizeof(char));
      strcpy(st->s[st->top], satellite);
      printf("Satellite deployed: %s\n", satellite);
   }
}

int pop(struct Stack *st)
{
   if (st->top == -1)
   {
      printf("Stack Underflow\n");
      return -1;
   }
   else
   {
      printf("Removing satellite deployment: %s\n", st->s[st->top]);
      free(st->s[st->top]);
      st->top--;
      return 0;
   }
```

```
}

void display(struct Stack st)
{
   if (st.top == -1)
      printf("Deployment stack is empty\n");
   else
   {
      printf("Current satellite deployment sequence:\n");
      for (int i = st.top; i >= 0; i--)
         printf("%d: %s\n", i + 1, st.s[i]);
   }
}

int peek(struct Stack st, int index)
{
   if (index < 1 || index > st.top + 1)
   {
      printf("Invalid index\n");
      return -1;
   }
   return 0;
}

int search(struct Stack st, const char *satellite)
{
   for (int i = st.top; i >= 0; i--)
```

```
    {
        if (strcmp(st.s[i], satellite) == 0)
            return i + 1;
    }
    return -1;
}
```

3. Rocket Launch Checklist: Create a stack for a rocket launch checklist using arrays. Implement a switch-case menu with options:
    - 1: Add a checklist item (push)
    - 2: Remove the last item (pop)
    - 3: Display the current checklist
    - 4: Peek at the top checklist item
    - 5: Search for a specific checklist item
    - 6: Exit

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

#define MAX_ITEM_LENGTH 100

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

```
};

void create(struct Stack *st);
void push(struct Stack *st, const char *item);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *item);

int main()
{
    struct Stack st;
    create(&st);
    int option;
    char item[MAX_ITEM_LENGTH];
    int index;
    do
    {
        printf("\n--- Rocket 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");
        printf("Enter your option: ");
        scanf("%d", &option);
```

```
        switch (option)
        {
            case 1: printf("Enter the checklist item: ");
                    scanf(" %[^\n]", item);
                    push(&st, item);
                    break;
            case 2: pop(&st);
                    break;
            case 3: display(st);
                    break;
            case 4: printf("Enter the index to peek: ");
                    scanf("%d", &index);
                    if (peek(st, index) != -1)
                            printf("Checklist item at index %d: %s\n", index,
st.s[st.top - index + 1]);
                    break;
            case 5: printf("Enter the checklist item to search: ");
                    scanf(" %[^\n]", item);
                    int position = search(st, item);
                    if (position != -1)
                       printf("Checklist item found at index %d\n", position);
                    else
                       printf("Checklist item not found\n");
                    break;
            case 6: printf("Exiting the checklist\n");
                    for (int i = 0; i <= st.top; i++)
                       free(st.s[i]);
                    free(st.s);
```

```
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

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

void push(struct Stack *st, const char *item)
{
    if (st->top == st->size - 1)
        printf("Stack Overflow\n");
    else
    {
        st->top++;
            st->s[st->top] = (char *)malloc(MAX_ITEM_LENGTH *
sizeof(char));
        strcpy(st->s[st->top], item);
        printf("Checklist item added: %s\n", item);
    }
}
```

```
int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
        printf("Removing checklist item: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

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

```
}

int peek(struct Stack st, int index)
{
   if (index < 1 || index > st.top + 1)
   {
      printf("Invalid index\n");
      return -1;
   }
   return 0;
}

int search(struct Stack st, const char *item)
{
   for (int i = st.top; i >= 0; i--)
   {
      if (strcmp(st.s[i], item) == 0)
         return i + 1;
   }
   return -1;
}
```

4. Telemetry Data Storage: Implement a stack to store telemetry data from an aerospace vehicle. Use a switch-case menu with options:
    - 1: Push new telemetry data
    - 2: Pop the last data entry
    - 3: View the stored telemetry data

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

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

#define MAX_DATA_LENGTH 100

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

void create(struct Stack *st);
void push(struct Stack *st, const char *entry);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *entry);

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

```
create(&st);
int option;
char entry[MAX_DATA_LENGTH];
int index;
do
{
   printf("\n--- Telemetry Data Storage ---\n");
   printf("1: Push new telemetry data\n");
   printf("2: Pop the last data entry\n");
   printf("3: View the stored telemetry data\n");
   printf("4: Peek at the most recent data entry\n");
   printf("5: Search for specific telemetry data\n");
   printf("6: Exit\n");
   printf("Enter your option: ");
   scanf("%d", &option);
   switch (option)
   {
      case 1: printf("Enter telemetry data: ");
            scanf(" %[^\n]", entry);
            push(&st, entry);
            break;
      case 2: pop(&st);
            break;
      case 3: display(st);
            break;
      case 4: printf("Enter the index to peek: ");
            scanf("%d", &index);
```

```
                if (peek(st, index) != -1)
                        printf("Telemetry data at index %d: %s\n", index,
st.s[st.top - index + 1]);
                break;
        case 5: printf("Enter telemetry data to search: ");
                scanf(" %[^\n]", entry);
                int position = search(st, entry);
                if (position != -1)
                   printf("Telemetry data found at index %d\n", position);
                else
                   printf("Telemetry data not found\n");
                break;
        case 6: printf("Exiting the system\n");
                for (int i = 0; i <= st.top; i++)
                   free(st.s[i]);
                free(st.s);
                break;
        default: printf("Invalid option\n");
      }
   } while (option != 6);
   return 0;
}

void create(struct Stack *st)
{
   printf("Enter the size of the telemetry data stack: ");
   scanf("%d", &st->size);
   st->top = -1;
```

```
    st->s = (char **)malloc(st->size * sizeof(char *));
}

void push(struct Stack *st, const char *entry)
{
    if (st->top == st->size - 1)
        printf("Stack Overflow\n");
    else
    {
        st->top++;
            st->s[st->top] = (char *)malloc(MAX_DATA_LENGTH *
sizeof(char));
        strcpy(st->s[st->top], entry);
        printf("Telemetry data added: %s\n", entry);
    }
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
        printf("Removing telemetry data: %s\n", st->s[st->top]);
        free(st->s[st->top]);
```

```
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if (st.top == -1)
        printf("No telemetry data stored\n");
    else
    {
        printf("Stored telemetry data:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

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

```
int search(struct Stack st, const char *entry)
{
   for (int i = st.top; i >= 0; i--)
   {
      if (strcmp(st.s[i], entry) == 0)
         return i + 1;
   }
   return -1;
}
```

5. 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>

#define MAX_TASK_LENGTH 100

struct Stack
{
```

```
    int size;
    int top;
    char **s;
};

void create(struct Stack *st);
void push(struct Stack *st, const char *task);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *task);

int main()
{
    struct Stack st;
    create(&st);
    int option;
    char task[MAX_TASK_LENGTH];
    int index;
    do
    {
        printf("\n--- Space Mission Task Manager ---\n");
        printf("1: Add a task (push)\n");
        printf("2: Mark the last task as completed (pop)\n");
        printf("3: List all pending tasks\n");
        printf("4: Peek at the most recent task\n");
        printf("5: Search for a specific task\n");
```

```
        printf("6: Exit\n");
        printf("Enter your option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the task: ");
                    scanf(" %[^\n]", task);
                    push(&st, task);
                    break;
            case 2: pop(&st);
                    break;
            case 3: display(st);
                    break;
            case 4: printf("Enter the index to peek: ");
                    scanf("%d", &index);
                    if (peek(st, index) != -1)
                      printf("Task at index %d: %s\n", index, st.s[st.top - index
+ 1]);
                    break;
            case 5:
                    printf("Enter the task to search: ");
                    scanf(" %[^\n]", task);
                    int position = search(st, task);
                    if (position != -1)
                       printf("Task found at index %d\n", position);
                    else
                       printf("Task not found\n");
                    break;
```

```
            case 6: printf("Exiting the task manager\n");
                    for (int i = 0; i <= st.top; i++)
                        free(st.s[i]);
                    free(st.s);
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

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

void push(struct Stack *st, const char *task)
{
    if (st->top == st->size - 1)
        printf("Stack Overflow\n");
    else
    {
        st->top++;
        st->s[st->top] = (char *)malloc(MAX_TASK_LENGTH
```

```
                                                * sizeof(char));
        strcpy(st->s[st->top], task);
        printf("Task added: %s\n", task);
    }
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
        printf("Task completed: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if (st.top == -1)
        printf("No pending tasks\n");
    else
```

```
    {
        printf("Pending tasks:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

int peek(struct Stack st, int index)
{
    if (index < 1 || index > st.top + 1)
    {
        printf("Invalid index\n");
        return -1;
    }
    return 0;
}

int search(struct Stack st, const char *task)
{
    for (int i = st.top; i >= 0; i--)
    {
        if (strcmp(st.s[i], task) == 0)
            return i + 1;
    }
    return -1;
}
```

6. Launch Countdown Management: Use a stack to manage the countdown sequence for a rocket launch. Implement a switch-case menu with options:
    - 1: Add a countdown step (push)
    - 2: Remove the last step (pop)
    - 3: Display the current countdown
    - 4: Peek at the next countdown step
    - 5: Search for a specific countdown step
    - 6: Exit

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

#define MAX_STEP_LENGTH 100

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

void create(struct Stack *st);
void push(struct Stack *st, const char *step);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *step);
```

```
int main()
{
    struct Stack st;
    create(&st);
    int option;
    char step[MAX_STEP_LENGTH];
    int index;
    do
    {
        printf("\n--- Launch Countdown Management ---\n");
        printf("1: Add a countdown step (push)\n");
        printf("2: Remove the last step (pop)\n");
        printf("3: Display the current countdown\n");
        printf("4: Peek at the next countdown step\n");
        printf("5: Search for a specific countdown step\n");
        printf("6: Exit\n");
        printf("Enter your option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the countdown step: ");
                    scanf(" %[^\n]", step);
                    push(&st, step);
                    break;
            case 2: pop(&st);
                    break;
```

```
            case 3: display(st);
                    break;
            case 4: printf("Enter the index to peek: ");
                    scanf("%d", &index);
                    if (peek(st, index) != -1)
                            printf("Countdown step at index %d: %s\n", index,
st.s[st.top - index + 1]);
                    break;
            case 5: printf("Enter the countdown step to search: ");
                    scanf(" %[^\n]", step);
                    int position = search(st, step);
                    if (position != -1)
                       printf("Countdown step found at index %d\n", position);
                    else
                        printf("Countdown step not found\n");
                    break;
            case 6: printf("Exiting the management\n");
                    for (int i = 0; i <= st.top; i++)
                        free(st.s[i]);
                    free(st.s);
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void create(struct Stack *st)
```

```
{
    printf("Enter the maximum number of countdown steps: ");
    scanf("%d", &st->size);
    st->top = -1;
    st->s = (char **)malloc(st->size * sizeof(char *));
}

void push(struct Stack *st, const char *step)
{
    if (st->top == st->size - 1)
        printf("Stack Overflow\n");
    else
    {
        st->top++;
            st->s[st->top] = (char *)malloc(MAX_STEP_LENGTH *
sizeof(char));
        strcpy(st->s[st->top], step);
        printf("Step added: %s\n", step);
    }
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
```

```
    else
    {
        printf("Removing step: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if (st.top == -1)
        printf("No countdown steps available\n");
    else
    {
        printf("Current countdown sequence:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

int peek(struct Stack st, int index)
{
    if (index < 1 || index > st.top + 1)
    {
        printf("Invalid index\n");
        return -1;
```

```
    }
    return 0;
}

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

7. Aircraft Maintenance Logs: Implement a stack to keep track of maintenance logs for an aircraft. Use a switch-case menu with options:
    - 1: Add a new log (push)
    - 2: Remove the last log (pop)
    - 3: View all maintenance logs
    - 4: Peek at the latest maintenance log
    - 5: Search for a specific maintenance log
    - 6: Exit

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

```
#define MAX_LOG_LENGTH 100

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

void create(struct Stack *st);
void push(struct Stack *st, const char *log);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *log);

int main()
{
   struct Stack st;
   create(&st);
   int option;
   char log[MAX_LOG_LENGTH];
   int index;
   do
   {
      printf("\n--- Aircraft Maintenance Logs ---\n");
      printf("1: Add a new log (push)\n");
```

```
        printf("2: Remove the last log (pop)\n");
        printf("3: View all maintenance logs\n");
        printf("4: Peek at the latest maintenance log\n");
        printf("5: Search for a specific maintenance log\n");
        printf("6: Exit\n");
        printf("Enter your option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the maintenance log: ");
                    scanf(" %[^\n]", log);
                    push(&st, log);
                    break;
            case 2: pop(&st);
                    break;
            case 3: display(st);
                    break;
            case 4: printf("Enter the index to peek: ");
                    scanf("%d", &index);
                    if (peek(st, index) != -1)
                            printf("Maintenance log at index %d: %s\n", index,
st.s[st.top - index + 1]);
                    break;
            case 5: printf("Enter the maintenance log to search: ");
                    scanf(" %[^\n]", log);
                    int position = search(st, log);
                    if (position != -1)
                      printf("Maintenance log found at index %d\n", position);
```

```
            else
                printf("Maintenance log not found\n");
            break;
        case 6: printf("Exiting the maintenance log system.\n");
            for (int i = 0; i <= st.top; i++)
                free(st.s[i]);
            free(st.s);
            break;
        default: printf("Invalid option\n");
      }
   } while (option != 6);
   return 0;
}

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

void push(struct Stack *st, const char *log)
{
   if (st->top == st->size - 1)
      printf("Stack Overflow\n");
   else
```

```
    {
        st->top++;
        st->s[st->top] = (char *)malloc(MAX_LOG_LENGTH
                                                    * sizeof(char));
        strcpy(st->s[st->top], log);
        printf("Log added: %s\n", log);
    }
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
        printf("Removing log: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
```

```
    if (st.top == -1)
        printf("No maintenance logs available\n");
    else
    {
        printf("Current maintenance logs:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

int peek(struct Stack st, int index)
{
    if (index < 1 || index > st.top + 1)
    {
        printf("Invalid index\n");
        return -1;
    }
    return 0;
}

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

```
    return -1;
}
```

8. Spacecraft Docking Procedure: Develop a stack for the sequence of steps in a spacecraft docking procedure. Implement a switch-case menu with options:
    - 1: Push a new step
    - 2: Pop the last step
    - 3: Display the procedure steps
    - 4: Peek at the next step in the procedure
    - 5: Search for a specific step
    - 6: Exit

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

#define MAX_STEP_LENGTH 100

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

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

```
void push(struct Stack *st, const char *step);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *step);

int main()
{
    struct Stack st;
    create(&st);
    int option;
    char step[MAX_STEP_LENGTH];
    int index;
    do
    {
        printf("\n--- Spacecraft Docking Procedure ---\n");
        printf("1: Add a new step (push)\n");
        printf("2: Remove the last step (pop)\n");
        printf("3: Display the procedure steps\n");
        printf("4: Peek at the next step in the procedure\n");
        printf("5: Search for a specific step\n");
        printf("6: Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the procedure step: ");
```

```
            scanf(" %[^\n]", step);
            push(&st, step);
            break;
        case 2: pop(&st);
            break;
        case 3: display(st);
            break;
        case 4: printf("Enter the index to peek: ");
            scanf("%d", &index);
            if (peek(st, index) != -1)
              printf("Step at index %d: %s\n", index, st.s[st.top - index
+ 1]);
            break;
        case 5: printf("Enter the step to search: ");
            scanf(" %[^\n]", step);
            int position = search(st, step);
            if (position != -1)
               printf("Step found at index %d\n", position);
            else
               printf("Step not found\n");
            break;

        case 6: printf("Exiting the docking procedure system.\n");
            for (int i = 0; i <= st.top; i++)
               free(st.s[i]);
            free(st.s);
            break;
        default: printf("Invalid option\n");
```

```
        }
    } while (option != 6);
    return 0;
}

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

void push(struct Stack *st, const char *step)
{
    if (st->top == st->size - 1)
        printf("Stack Overflow\n");
    else
    {
        st->top++;
        st->s[st->top] = (char *)malloc(MAX_STEP_LENGTH
                                        * sizeof(char));
        strcpy(st->s[st->top], step);
        printf("Step added: %s\n", step);
    }
}
```

```
int pop(struct Stack *st)
{
   if (st->top == -1)
   {
      printf("Stack Underflow\n");
      return -1;
   }
   else
   {
      printf("Removing step: %s\n", st->s[st->top]);
      free(st->s[st->top]);
      st->top--;
      return 0;
   }
}

void display(struct Stack st)
{
   if (st.top == -1)
      printf("No procedure steps available\n");
   else
   {
      printf("Current procedure steps:\n");
      for (int i = st.top; i >= 0; i--)
         printf("%d: %s\n", i + 1, st.s[i]);
   }
}
```

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

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

9. Mission Control Command History: Create a stack to record the command history sent from mission control. Use a switch-case menu with options:
    - 1: Add a command (push)
    - 2: Undo the last command (pop)
    - 3: View the command history
    - 4: Peek at the most recent command

- 5: Search for a specific command
- 6: Exit

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

#define MAX_COMMAND_LENGTH 100

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

void create(struct Stack *st);
void push(struct Stack *st, const char *command);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *command);

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

```
int option;
char command[MAX_COMMAND_LENGTH];
int index;
do
{
   printf("\n--- Mission Control Command History ---\n");
   printf("1: Add a command (push)\n");
   printf("2: Undo the last command (pop)\n");
   printf("3: View the command history\n");
   printf("4: Peek at the most recent command\n");
   printf("5: Search for a specific command\n");
   printf("6: Exit\n");
   printf("Enter your option: ");
   scanf("%d", &option);
   switch (option)
   {
      case 1: printf("Enter the command: ");
              scanf(" %[^\n]", command);
              push(&st, command);
              break;
      case 2: pop(&st);
              break;
      case 3: display(st);
              break;
      case 4: printf("Enter the index to peek: ");
              scanf("%d", &index);
              if (peek(st, index) != -1)
```

```
                printf("Command at index %d: %s\n", index, st.s[st.top
- index + 1]);
                break;
            case 5: printf("Enter the command to search: ");
                scanf(" %[^\n]", command);
                int position = search(st, command);
                if (position != -1)
                    printf("Command found at index %d\n", position);
                else
                    printf("Command not found\n");
                break;
            case 6: printf("Exiting the system\n");
                for (int i = 0; i <= st.top; i++)
                    free(st.s[i]);
                free(st.s);
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

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

```
}

void push(struct Stack *st, const char *command)
{
   if (st->top == st->size - 1)
      printf("Stack Overflow\n");
   else
   {
      st->top++;
      st->s[st->top] = (char *)malloc(MAX_COMMAND_LENGTH
                                                  * sizeof(char));
      strcpy(st->s[st->top], command);
      printf("Command added: %s\n", command);
   }
}

int pop(struct Stack *st)
{
   if (st->top == -1)
   {
      printf("Stack Underflow\n");
      return -1;
   }
   else
   {
      printf("Undoing command: %s\n", st->s[st->top]);
      free(st->s[st->top]);
```

```
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if (st.top == -1)
        printf("No commands in history\n");
    else
    {
        printf("Current command history:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

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

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

10. Aerospace Simulation Events: Implement a stack to handle events in an aerospace simulation. Include a switch-case menu with options:
    - 1: Push a new event
    - 2: Pop the last event
    - 3: Display all events
    - 4: Peek at the most recent event
    - 5: Search for a specific event
    - 6: Exit

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

#define MAX_EVENT_LENGTH 100

struct Stack
{
```

```
    int size;
    int top;
    char **s;
};

void create(struct Stack *st);
void push(struct Stack *st, const char *event);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *event);

int main()
{
    struct Stack st;
    create(&st);
    int option;
    char event[MAX_EVENT_LENGTH];
    int index;
    do
    {
        printf("\n--- Aerospace Simulation Events ---\n");
        printf("1: Push a new event\n");
        printf("2: Pop the last event\n");
        printf("3: Display all events\n");
        printf("4: Peek at the most recent event\n");
        printf("5: Search for a specific event\n");
```

```
        printf("6: Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the event: ");
                    scanf(" %[^\n]", event);
                    push(&st, event);
                    break;
            case 2: pop(&st);
                    break;
            case 3: display(st);
                    break;
            case 4: printf("Enter the index to peek: ");
                    scanf("%d", &index);
                    if (peek(st, index) != -1)
                        printf("Event at index %d: %s\n", index, st.s[st.top -
index + 1]);
                    break;
            case 5: printf("Enter the event to search: ");
                    scanf(" %[^\n]", event);
                    int position = search(st, event);
                    if (position != -1)
                       printf("Event found at index %d\n", position);
                    else
                       printf("Event not found\n");
                    break;
            case 6: printf("Exiting the simulation\n");
```

```
                for (int i = 0; i <= st.top; i++)
                    free(st.s[i]);
                free(st.s);
                break;
            default: printf("Invalid option\n");
            }
    } while (option != 6);
    return 0;
}

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

void push(struct Stack *st, const char *event)
{
    if (st->top == st->size - 1)
        printf("Stack Overflow\n");
    else
    {
        st->top++;
        st->s[st->top] = (char *)malloc(MAX_EVENT_LENGTH
                                        * sizeof(char));
```

```
        strcpy(st->s[st->top], event);
        printf("Event added: %s\n", event);
    }
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
        printf("Removing event: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if (st.top == -1)
        printf("No events in the stack\n");
    else
    {
```

```
        printf("Current aerospace simulation events:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

int peek(struct Stack st, int index)
{
    if (index < 1 || index > st.top + 1)
    {
        printf("Invalid index\n");
        return -1;
    }
    return 0;
}

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

11. Pilot Training Maneuver Stack: Use a stack to keep track of training maneuvers for pilots. Implement a switch-case menu with options:
    - 1: Add a maneuver (push)
    - 2: Remove the last maneuver (pop)
    - 3: View all maneuvers
    - 4: Peek at the most recent maneuver
    - 5: Search for a specific maneuver
    - 6: Exit

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

#define MAX_MANEUVERS 100
#define MAX_MANEUVER_LENGTH 100

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

void create(struct Stack *st);
void push(struct Stack *st, const char *maneuver);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
```

```
int search(struct Stack st, const char *maneuver);

int main()
{
   struct Stack st;
   create(&st);
   int option;
   char maneuver[MAX_MANEUVER_LENGTH];
   int index;
   do
   {
      printf("\n--- Pilot 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");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: printf("Enter the maneuver: ");
                 scanf(" %[^\n]", maneuver);
                 push(&st, maneuver);
                 break;
         case 2: pop(&st);
```

```
            break;
        case 3: display(st);
            break;
        case 4: printf("Enter the index to peek: ");
            scanf("%d", &index);
            if (peek(st, index) != -1)
                printf("Maneuver at index %d: %s\n", index, st.s[st.top
- index + 1]);
            break;
        case 5: printf("Enter the maneuver to search: ");
            scanf(" %[^\n]", maneuver);
            int position = search(st, maneuver);
            if (position != -1)
               printf("Maneuver found at index %d\n", position);
            else
               printf("Maneuver not found\n");
            break;
        case 6: printf("Exiting the stack\n");
            for (int i = 0; i <= st.top; i++)
               free(st.s[i]);
            free(st.s);
            break;
        default: printf("Invalid option\n");
      }
   } while (option != 6);
   return 0;
}
```

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

void push(struct Stack *st, const char *maneuver)
{
   if (st->top == st->size - 1)
      printf("Stack Overflow\n");
   else
   {
      st->top++;
      st->s[st->top] = (char *)malloc(MAX_MANEUVER_LENGTH
                                          * sizeof(char));
      strcpy(st->s[st->top], maneuver);
      printf("Maneuver added: %s\n", maneuver);
   }
}

int pop(struct Stack *st)
{
   if (st->top == -1)
   {
      printf("Stack Underflow\n");
```

```
        return -1;
    }
    else
    {
        printf("Removing maneuver: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if (st.top == -1)
        printf("No maneuvers in the stack\n");
    else
    {
        printf("Current pilot training maneuvers:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

int peek(struct Stack st, int index)
{
    if (index < 1 || index > st.top + 1)
    {
```

```
        printf("Invalid index\n");
        return -1;
    }
    return 0;
}

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

12. Satellite Operation Commands: Design a stack to manage operation commands for a satellite. Use a switch-case menu with options:
    - 1: Push a new command
    - 2: Pop the last command
    - 3: View the operation commands
    - 4: Peek at the most recent command
    - 5: Search for a specific command
    - 6: Exit

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

```
#include <string.h>

#define MAX_COMMAND_LENGTH 100

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

void create(struct Stack *st);
void push(struct Stack *st, const char *command);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *command);

int main()
{
   struct Stack st;
   create(&st);
   int option;
   char command[MAX_COMMAND_LENGTH];
   int index;
   do
   {
```

```
        printf("\n--- Satellite Operation Commands Stack ---\n");
        printf("1: Push a new command\n");
        printf("2: Pop the last command\n");
        printf("3: View the operation commands\n");
        printf("4: Peek at the most recent command\n");
        printf("5: Search for a specific command\n");
        printf("6: Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
           case 1: printf("Enter the satellite operation command: ");
                   scanf(" %[^\n]", command);
                   push(&st, command);
                   break;
           case 2: pop(&st);
                   break;
           case 3: display(st);
                   break;
           case 4: printf("Enter the index to peek: ");
                   scanf("%d", &index);
                   if (peek(st, index) != -1)
                       printf("Command at index %d: %s\n", index, st.s[st.top
- index + 1]);
                   break;
           case 5: printf("Enter the command to search: ");
                   scanf(" %[^\n]", command);
                   int position = search(st, command);
```

```
            if (position != -1)
                printf("Command found at index %d\n", position);
            else
                printf("Command not found\n");
            break;
        case 6: printf("Exiting the stack\n");
            for (int i = 0; i <= st.top; i++)
                free(st.s[i]);
            free(st.s);
            break;
        default: printf("Invalid option\n");
      }
   } while (option != 6);
   return 0;
}

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

void push(struct Stack *st, const char *command)
{
   if (st->top == st->size - 1)
```

```
        printf("Stack Overflow\n");
    else
    {
        st->top++;
        st->s[st->top] = (char *)malloc(MAX_COMMAND_LENGTH
                                                * sizeof(char));
        strcpy(st->s[st->top], command);
        printf("Command added: %s\n", command);
    }
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
        printf("Removing command: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}
```

```
void display(struct Stack st)
{
   if (st.top == -1)
      printf("No commands in the stack\n");
   else
   {
      printf("Current satellite operation commands:\n");
      for (int i = st.top; i >= 0; i--)
         printf("%d: %s\n", i + 1, st.s[i]);
   }
}

int peek(struct Stack st, int index)
{
   if (index < 1 || index > st.top + 1)
   {
      printf("Invalid index\n");
      return -1;
   }
   return 0;
}

int search(struct Stack st, const char *command)
{
   for (int i = st.top; i >= 0; i--)
   {
      if (strcmp(st.s[i], command) == 0)
```

```
            return i + 1;
        }
        return -1;
    }
```

13. Emergency Procedures for Spacecraft: Create a stack-based system for handling emergency procedures in a spacecraft. Implement a switch-case menu with options:
    - 1: Add a procedure (push)
    - 2: Remove the last procedure (pop)
    - 3: View all procedures
    - 4: Peek at the next procedure
    - 5: Search for a specific procedure
    - 6: Exit

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

#define MAX_PROCEDURE_LENGTH 100

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

```
void create(struct Stack *st);
void push(struct Stack *st, const char *procedure);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *procedure);

int main()
{
   struct Stack st;
   create(&st);
   int option;
   char procedure[MAX_PROCEDURE_LENGTH];
   int index;
   do
   {
      printf("\n--- Emergency 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");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
```

```
        {
            case 1: printf("Enter the emergency procedure: ");
                    scanf(" %[^\n]", procedure);
                    push(&st, procedure);
                    break;
            case 2: pop(&st);
                    break;
            case 3: display(st);
                    break;

            case 4: printf("Enter the index to peek: ");
                    scanf("%d", &index);
                    if (peek(st, index) != -1)
                        printf("Procedure at index %d: %s\n", index, st.s[st.top
- index + 1]);
                    break;
            case 5: printf("Enter the procedure to search: ");
                    scanf(" %[^\n]", procedure);
                    int position = search(st, procedure);
                    if (position != -1)
                       printf("Procedure found at index %d\n", position);
                    else
                       printf("Procedure not found\n");
                    break;
            case 6: printf("Exiting the system\n");
                    for (int i = 0; i <= st.top; i++)
                       free(st.s[i]);
                    free(st.s);
```

```
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

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

void push(struct Stack *st, const char *procedure)
{
    if (st->top == st->size - 1)
        printf("Stack Overflow\n");
    else
    {
        st->top++;
        st->s[st->top] = (char *)malloc(MAX_PROCEDURE_LENGTH
                                        * sizeof(char));
        strcpy(st->s[st->top], procedure);
        printf("Procedure added: %s\n", procedure);
    }
```

```
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
        printf("Removing procedure: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if (st.top == -1)
        printf("No procedures in the stack\n");
    else
    {
        printf("Current emergency procedures:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
```

```
    }
}

int peek(struct Stack st, int index)
{
    if (index < 1 || index > st.top + 1)
    {
        printf("Invalid index\n");
        return -1;
    }
    return 0;
}

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

14. Astronaut Activity Log: Implement a stack for logging astronaut activities during a mission. Use a switch-case menu with options:
    - 1: Add a new activity (push)
    - 2: Remove the last activity (pop)

- 3: Display the activity log
- 4: Peek at the most recent activity
- 5: Search for a specific activity
- 6: Exit

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

#define MAX_ACTIVITY_LENGTH 100

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

void create(struct Stack *st);
void push(struct Stack *st, const char *activity);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *activity);

int main()
{
```

```
struct Stack st;
create(&st);
int option;
char activity[MAX_ACTIVITY_LENGTH];
int index;
do
{
   printf("\n--- Astronaut 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");
   printf("Enter the option: ");
   scanf("%d", &option);
   switch (option)
   {
      case 1: printf("Enter the astronaut activity: ");
              scanf(" %[^\n]", activity);
              push(&st, activity);
              break;
      case 2: pop(&st);
              break;
      case 3: display(st);
              break;
      case 4: printf("Enter the index to peek: ");
```

```
                scanf("%d", &index);
                if (peek(st, index) != -1)
                    printf("Activity at index %d: %s\n", index, st.s[st.top -
index + 1]);
                break;
            case 5: printf("Enter the activity to search: ");
                scanf(" %[^\n]", activity);
                int position = search(st, activity);
                if (position != -1)
                   printf("Activity found at index %d\n", position);
                else
                   printf("Activity not found\n");
                break;
            case 6: printf("Exiting the system\n");
                for (int i = 0; i <= st.top; i++)
                   free(st.s[i]);
                free(st.s);
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void create(struct Stack *st)
{
    printf("Enter the maximum number of activities: ");
    scanf("%d", &st->size);
```

```
    st->top = -1;
    st->s = (char **)malloc(st->size * sizeof(char *));
}

void push(struct Stack *st, const char *activity)
{
    if (st->top == st->size - 1)
        printf("Stack Overflow\n");
    else
    {
        st->top++;
        st->s[st->top] = (char *)malloc(MAX_ACTIVITY_LENGTH
                                        * sizeof(char));
        strcpy(st->s[st->top], activity);
        printf("Activity added: %s\n", activity);
    }
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
```

```
        printf("Removing activity: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if (st.top == -1)
        printf("No activities in the log\n");
    else
    {
        printf("Current astronaut activity log:\n");
        for (int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

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

```
}

int search(struct Stack st, const char *activity)
{
   for (int i = st.top; i >= 0; i--)
   {
      if (strcmp(st.s[i], activity) == 0)
         return i + 1;
   }
   return -1;
}
```

15. Fuel Management System: Develop a stack to monitor fuel usage in an aerospace vehicle. Implement a switch-case menu with options:
    - 1: Add a fuel usage entry (push)
    - 2: Remove the last entry (pop)
    - 3: View all fuel usage data
    - 4: Peek at the latest fuel usage entry
    - 5: Search for a specific fuel usage entry
    - 6: Exit

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

#define MAX_ENTRY_LENGTH 100
```

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

void create(struct Stack *st);
void push(struct Stack *st, const char *entry);
int pop(struct Stack *st);
void display(struct Stack st);
int peek(struct Stack st, int index);
int search(struct Stack st, const char *entry);

int main()
{
   struct Stack st;
   create(&st);
   int option;
   char entry[MAX_ENTRY_LENGTH];
   int index;
   do
   {
      printf("\n--- Fuel 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 all latest fuel usage entry\n");
        printf("5. Search for a specific fuel usage entry\n");
        printf("Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the fuel usage entry: ");
                    scanf(" %[^\n]", entry);
                    push(&st, entry);
                    break;
            case 2: pop(&st);
                    break;
            case 3: display(st);
                    break;
            case 4: printf("Enter the index to peek: ");
                    scanf(" %d", &index);
                    if(peek(st, index) != -1)
                        printf("Fuel usage at index %d: %s\n", index, st.s[st.top
- index + 1]);
                    break;
            case 5: printf("Enter the fuwl usage entry to search: ");
                    scanf(" %[^\n]", entry);
                    int position = search(st, entry);
                    if(position != -1)
                       printf("fuel usage entry found at index %d\n", position);
                    else
                       printf("Fuel usage entry not found\n");
```

```
            break;
        case 6: printf("Exiting the system\n");
            for(int i = 0; i <= st.top; i++)
                free(st.s[i]);
            free(st.s);
            break;
        default: printf("Invalid oprion\n");
      }
   } while( option != 6);
   return 0;
}

void create(struct Stack *st)
{
   printf("Enter the maximum number of fuel usage entries: ");
   scanf("%d", &st->size);
   st->top = -1;
   st->s = (char **)malloc(st->size * sizeof(char *));
}

void push(struct Stack *st, const char *entry)
{
   if(st->top == st->size - 1)
      printf("Stack Overflow\n");
   else
   {
      st->top++;
```

```
        st->s[st->top] = (char *)malloc(MAX_ENTRY_LENGTH
                                                * sizeof(char));
        strcpy(st->s[st->top], entry);
        printf("Fuel usage entry added: %s\n", entry);
    }
}

int pop(struct Stack *st)
{
    if(st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    else
    {
        printf("Removing fuel usage entry: %s\n", st->s[st->top]);
        free(st->s[st->top]);
        st->top--;
        return 0;
    }
}

void display(struct Stack st)
{
    if(st.top == -1)
        printf("No fuel usage data available\n");
```

```
    else
    {
        printf("Current fuel usage data:\n");
        for(int i = st.top; i >= 0; i--)
            printf("%d: %s\n", i + 1, st.s[i]);
    }
}

int peek(struct Stack st, int index)
{
    if(index < 1 || index > st.top + 1)
    {
        printf("Invalid index\n");
        return -1;
    }
    return 0;
}

int search(struct Stack st, const char *entry)\
{
    for(int i = st.top; i >= 0; i--)
    {
        if(strcmp(st.s[i], entry) == 0)
            return i + 1;
    }
    return -1;
}
```

Stacks using linked list

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>

#define MAX_ORDER_LENGTH 100

struct Node
{
   char order[MAX_ORDER_LENGTH];
   struct Node* next;
} *top = NULL;

void push(const char* order);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* order);
```

```
int main()
{
   int option, index;
   char order[MAX_ORDER_LENGTH];
   do
   {
      printf("\n--- Order Processing System ---\n");
      printf("1: Add a new order (push)\n");
      printf("2: Process the last order (pop)\n");
      printf("3: Display all pending orders\n");
      printf("4: Peek at the next order to be processed\n");
      printf("5: Search for a specific order\n");
      printf("6: Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: printf("Enter the order: ");
                 scanf(" %[^\n]", order);
                 push(order);
                 break;
         case 2: pop();
                 break;
         case 3: display();
                 break;
         case 4: printf("Enter the index to peek: ");
                 scanf("%d", &index);
```

```
                peek(*top, index);
                break;
            case 5: printf("Enter the order to search for: ");
                scanf(" %[^\n]", order);
                int position = search(order);
                if (position != -1)
                    printf("Order found at position %d\n", position);
                else
                    printf("Order not found\n");
                break;
            case 6: printf("Exiting the system\n");
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void push(const char* order)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->order, order);
        t->next = top;
```

```
        top = t;
        printf("Order added: %s\n", order);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No orders to process (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Processing order: %s\n", temp->order);
    free(temp);
    return 0;
}

void display()
{
    if (top == NULL)
    {
        printf("No pending orders\n");
        return;
    }
    struct Node* current = top;
```

```
    printf("Pending orders:\n");
    while (current != NULL)
    {
        printf("%s\n", current->order);
        current = current->next;
    }
}

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
        {
            printf("Order at index %d: %s\n", position, current->order);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* order)
```

```
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->order, order) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

2. Customer Support Ticketing: Create a stack using a linked list to handle customer support tickets. Include a switch-case menu with options:
    - 1: Add a new ticket (push)
    - 2: Resolve the latest ticket (pop)
    - 3: View all pending tickets
    - 4: Peek at the latest ticket
    - 5: Search for a specific ticket
    - 6: Exit

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

#define MAX_TICKET_LENGTH 100
```

```
struct Node
{
   char ticket[MAX_TICKET_LENGTH];
   struct Node* next;
} *top = NULL;

void push(const char* ticket);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* ticket);

int main()
{
   int option, index;
   char ticket[MAX_TICKET_LENGTH];
   do
   {
      printf("\n--- Customer Support Ticketing System ---\n");
      printf("1: Add a new ticket (push)\n");
      printf("2: Resolve the latest ticket (pop)\n");
      printf("3: View all pending tickets\n");
      printf("4: Peek at the latest ticket\n");
      printf("5: Search for a specific ticket\n");
      printf("6: Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
```

```
    switch (option)
    {
        case 1: printf("Enter the ticket description: ");
                scanf(" %[^\n]", ticket);
                push(ticket);
                break;
        case 2: pop();
                break;
        case 3: display();
                break;
        case 4: printf("Enter the index to peek: ");
                scanf("%d", &index);
                peek(*top, index);
                break;
        case 5: printf("Enter the ticket description to search for: ");
                scanf(" %[^\n]", ticket);
                int position = search(ticket);
                if (position != -1)
                    printf("Ticket found at position %d\n", position);
                else
                    printf("Ticket not found\n");
                break;
        case 6: printf("Exiting the system\n");
                break;
        default: printf("Invalid option\n");
    }
} while (option != 6);
```

```
    return 0;
}

void push(const char* ticket)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->ticket, ticket);
        t->next = top;
        top = t;
        printf("Ticket added: %s\n", ticket);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No tickets to resolve (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Resolving ticket: %s\n", temp->ticket);
```

```
    free(temp);
    return 0;
}

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

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
```

```
        {
            printf("Ticket at index %d: %s\n", position, current->ticket);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* ticket)
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->ticket, ticket) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

3. Product Return Management: Develop a stack to manage product returns using a linked list. Implement a switch-case menu with options:

- 1: Add a new return request (push)
- 2: Process the last return (pop)
- 3: Display all return requests
- 4: Peek at the next return to process
- 5: Search for a specific return request
- 6: Exit

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

#define MAX_RETURN_LENGTH 100

struct Node
{
    char returnRequest[MAX_RETURN_LENGTH];
    struct Node* next;
} *top = NULL;

void push(const char* returnRequest);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* returnRequest);

int main()
{
```

```
int option, index;
char returnRequest[MAX_RETURN_LENGTH];
do
{
   printf("\n--- Product Return Management System ---\n");
   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");
   printf("Enter the option: ");
   scanf("%d", &option);
   switch (option)
   {
      case 1: printf("Enter the return request description: ");
            scanf(" %[^\n]", returnRequest);
            push(returnRequest);
            break;
      case 2: pop();
            break;
      case 3: display();
            break;
      case 4: printf("Enter the index to peek: ");
            scanf("%d", &index);
            peek(*top, index);
            break;
```

```
            case 5: printf("Enter the return request description to search for:
");
                    scanf(" %[^\n]", returnRequest);
                    int position = search(returnRequest);
                    if (position != -1)
                      printf("Return request found at position %d\n", position);
                    else
                       printf("Return request not found\n");
                    break;
            case 6: printf("Exiting the management\n");
                    break;
            default:printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void push(const char* returnRequest)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->returnRequest, returnRequest);
        t->next = top;
        top = t;
        printf("Return request added: %s\n", returnRequest);
```

```
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No return requests to process (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Processing return request: %s\n", temp->returnRequest);
    free(temp);
    return 0;
}

void display()
{
    if (top == NULL)
    {
        printf("No pending return requests\n");
        return;
    }
    struct Node* current = top;
    printf("Pending return requests:\n");
    while (current != NULL)
```

```
    {
        printf("%s\n", current->returnRequest);
        current = current->next;
    }
}

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
        {
            printf("Return request at index %d: %s\n", position, current-
>returnRequest);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* returnRequest)
{
    struct Node* current = top;
```

```
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->returnRequest, returnRequest) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

4. Inventory Restock System: Implement a stack to manage inventory restocking using a linked list. Use a switch-case menu with options:
    - 1: Add a restock entry (push)
    - 2: Process the last restock (pop)
    - 3: View all restock entries
    - 4: Peek at the latest restock entry
    - 5: Search for a specific restock entry
    - 6: Exit

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

#define MAX_ENTRY_LENGTH 100

struct Node
```

```
{
    char restockEntry[MAX_ENTRY_LENGTH];
    struct Node* next;
} *top = NULL;

void push(const char* restockEntry);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* restockEntry);

int main()
{
    int option, index;
    char restockEntry[MAX_ENTRY_LENGTH];
    do
    {
        printf("\n--- Inventory Restock System ---\n");
        printf("1: Add a restock entry (push)\n");
        printf("2: Process the last restock (pop)\n");
        printf("3: View all restock entries\n");
        printf("4: Peek at the latest restock entry\n");
        printf("5: Search for a specific restock entry\n");
        printf("6: Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
```

```
        {
            case 1: printf("Enter the restock entry description: ");
                    scanf(" %[^\n]", restockEntry);
                    push(restockEntry);
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: printf("Enter the index to peek: ");
                    scanf("%d", &index);
                    peek(*top, index);
                    break;
            case 5: printf("Enter the restock entry description to search for:
");
                    scanf(" %[^\n]", restockEntry);
                    int position = search(restockEntry);
                    if (position != -1)
                       printf("Restock entry found at position %d\n", position);
                    else
                       printf("Restock entry not found\n");
                    break;
            case 6: printf("Exiting the system\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
```

```
}

void push(const char* restockEntry)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->restockEntry, restockEntry);
        t->next = top;
        top = t;
        printf("Restock entry added: %s\n", restockEntry);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No restock entries to process (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Processing restock entry: %s\n", temp->restockEntry);
    free(temp);
```

```
    return 0;
}

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

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
        {
```

```
            printf("Restock entry at index %d: %s\n", position, current->restockEntry);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* restockEntry)
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->restockEntry, restockEntry) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

5. Flash Sale Deal Management: Create a stack for managing flash sale deals using a linked list. Include a switch-case menu with options:

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

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

#define MAX_DEAL_DESCRIPTION_LENGTH 100

struct Node
{
   char dealDescription[MAX_DEAL_DESCRIPTION_LENGTH];
   struct Node* next;
} *top = NULL;

void push(const char* dealDescription);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* dealDescription);

int main()
{
```

```
int option, index;
char dealDescription[MAX_DEAL_DESCRIPTION_LENGTH];
do
{
   printf("\n--- Flash Sale Deal Management ---\n");
   printf("1: Add a new deal (push)\n");
   printf("2: Remove the last deal (pop)\n");
   printf("3: View all active deals\n");
   printf("4: Peek at the latest deal\n");
   printf("5: Search for a specific deal\n");
   printf("6: Exit\n");
   printf("Enter the option: ");
   scanf("%d", &option);
   switch (option)
   {
      case 1: printf("Enter the deal description: ");
              scanf(" %[^\n]", dealDescription);
              push(dealDescription);
              break;
      case 2: pop();
              break;
      case 3: display();
              break;
      case 4: printf("Enter the index to peek: ");
              scanf("%d", &index);
              peek(*top, index);
              break;
```

```
            case 5: printf("Enter the deal description to search for: ");
                    scanf(" %[^\n]", dealDescription);
                    int position = search(dealDescription);
                    if (position != -1)
                        printf("Deal found at position %d\n", position);
                    else
                        printf("Deal not found\n");
                    break;
            case 6: printf("Exiting the management\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void push(const char* dealDescription)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->dealDescription, dealDescription);
        t->next = top;
        top = t;
        printf("Deal added: %s\n", dealDescription);
```

```
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No active deals to remove (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Removed deal: %s\n", temp->dealDescription);
    free(temp);
    return 0;
}

void display()
{
    if (top == NULL)
    {
        printf("No active deals\n");
        return;
    }
    struct Node* current = top;
    printf("Active deals:\n");
    while (current != NULL)
```

```
    {
        printf("%s\n", current->dealDescription);
        current = current->next;
    }
}

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
        {
                printf("Deal at index %d: %s\n", position, current-
>dealDescription);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* dealDescription)
{
    struct Node* current = top;
```

```
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->dealDescription, dealDescription) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

6. User Session History: Use a stack to track user session history in an e-commerce site using a linked list. Implement a switch-case menu with options:
    - 1: Add a session (push)
    - 2: End the last session (pop)
    - 3: Display all sessions
    - 4: Peek at the most recent session
    - 5: Search for a specific session
    - 6: Exit

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

#define MAX_SESSION_INFO_LENGTH 100
```

```
struct Node
{
   char sessionInfo[MAX_SESSION_INFO_LENGTH];
   struct Node* next;
} *top = NULL;

void push(const char* sessionInfo);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* sessionInfo);

int main()
{
   int option, index;
   char sessionInfo[MAX_SESSION_INFO_LENGTH];
   do
   {
      printf("\n--- User 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");
      printf("Enter the option: ");
      scanf("%d", &option);
```

```
    switch (option)
    {
        case 1: printf("Enter the session info: ");
                scanf(" %[^\n]", sessionInfo);  // Read session info
                push(sessionInfo);
                break;
        case 2: pop();
                break;
        case 3: display();
                break;
        case 4: printf("Enter the index to peek: ");
                scanf("%d", &index);
                peek(*top, index);
                break;
        case 5: printf("Enter the session info to search for: ");
                scanf(" %[^\n]", sessionInfo);
                int position = search(sessionInfo);
                if (position != -1)
                    printf("Session found at position %d\n", position);
                else
                    printf("Session not found\n");
                break;
        case 6: printf("Exiting the history\n");
                break;
        default: printf("Invalid option\n");
    }
} while (option != 6);
```

```
    return 0;
}

void push(const char* sessionInfo)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->sessionInfo, sessionInfo);
        t->next = top;
        top = t;
        printf("Session added: %s\n", sessionInfo);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No active sessions to end (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Ended session: %s\n", temp->sessionInfo);
```

```
    free(temp);
    return 0;
}

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

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
```

```
        {
          printf("Session at index %d: %s\n", position,
                                                  current->sessionInfo);
           return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* sessionInfo)
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->sessionInfo, sessionInfo) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

7. Wishlist Management: Develop a stack to manage user wishlists using a linked list. Use a switch-case menu with options:
    - 1: Add a product to wishlist (push)
    - 2: Remove the last added product (pop)
    - 3: View all wishlist items
    - 4: Peek at the most recent wishlist item
    - 5: Search for a specific product in wishlist
    - 6: Exit

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

#define MAX_PRODUCT_NAME_LENGTH 100

struct Node
{
   char productName[MAX_PRODUCT_NAME_LENGTH];
   struct Node* next;
} *top = NULL;

void push(const char* productName);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* productName);

int main()
```

```
{
    int option, index;
    char productName[MAX_PRODUCT_NAME_LENGTH];
    do
    {
        printf("\n--- Wishlist Management ---\n");
        printf("1: Add a product to wishlist (push)\n");
        printf("2: Remove the last added product (pop)\n");
        printf("3: View all wishlist items\n");
        printf("4: Peek at the most recent wishlist item\n");
        printf("5: Search for a specific product in wishlist\n");
        printf("6: Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1:
                printf("Enter the product name: ");
                scanf(" %[^\n]", productName);
                push(productName);
                break;
            case 2: pop();
                break;
            case 3: display();
                break;
            case 4: printf("Enter the index to peek: ");
                scanf("%d", &index);
```

```
                peek(*top, index);
                break;
            case 5: printf("Enter the product name to search for: ");
                scanf(" %[^\n]", productName);
                int position = search(productName);
                if (position != -1)
                    printf("Product found at position %d\n", position);
                else
                    printf("Product not found\n");
                break;
            case 6: printf("Exiting the management\n");
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void push(const char* productName)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->productName, productName);
        t->next = top;
```

```
        top = t;
        printf("Product added to wishlist: %s\n", productName);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No products in the wishlist to remove (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Product removed from wishlist: %s\n", temp->productName);
    free(temp);
    return 0;
}

void display()
{
    if (top == NULL)
    {
        printf("No products in the wishlist\n");
        return;
    }
    struct Node* current = top;
```

```
    printf("Wishlist items:\n");
    while (current != NULL)
    {
        printf("%s\n", current->productName);
        current = current->next;
    }
}

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
        {
                printf("Product at index %d: %s\n", position, current->productName);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* productName)
```

```
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->productName, productName) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

8. Checkout Process Steps: Implement a stack to manage steps in the checkout process using a linked list. Include a switch-case menu with options:
    - 1: Add a checkout step (push)
    - 2: Remove the last step (pop)
    - 3: Display all checkout steps
    - 4: Peek at the current step
    - 5: Search for a specific step
    - 6: Exit

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

```
#define MAX_STEP_NAME_LENGTH 100

struct Node
{
   char stepName[MAX_STEP_NAME_LENGTH];
   struct Node* next;
} *top = NULL;

void push(const char* stepName);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* stepName);

int main()
{
   int option, index;
   char stepName[MAX_STEP_NAME_LENGTH];
   do
   {
      printf("\n--- Checkout Process Steps ---\n");
      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");
```

```
printf("Enter the option: ");
scanf("%d", &option);
switch (option)
{
   case 1: printf("Enter the checkout step name: ");
           scanf(" %[^\n]", stepName);
           push(stepName);
           break;
   case 2: pop();
           break;
   case 3: display();
           break;
   case 4: printf("Enter the index to peek: ");
           scanf("%d", &index);
           peek(*top, index);
           break;
   case 5: printf("Enter the step name to search for: ");
           scanf(" %[^\n]", stepName);
           int position = search(stepName);
           if (position != -1)
              printf("Step found at position %d\n", position);
           else
              printf("Step not found\n");
           break;
   case 6: printf("Exiting\n");
           break;
   default:printf("Invalid option\n");
```

```
        }
    } while (option != 6);
    return 0;
}

void push(const char* stepName)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->stepName, stepName);
        t->next = top;
        top = t;
        printf("Step added to checkout process: %s\n", stepName);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No steps to process (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
```

```
    top = top->next;
        printf("Step removed from checkout process: %s\n", temp->stepName);
    free(temp);
    return 0;
}

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

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
```

```
    {
        if (position == index)
        {
            printf("Step at index %d: %s\n", position, current->stepName);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* stepName)
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->stepName, stepName) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

9. Coupon Code Management: Create a stack for managing coupon codes using a linked list. Use a switch-case menu with options:
    - 1: Add a new coupon code (push)
    - 2: Remove the last coupon code (pop)
    - 3: View all available coupon codes
    - 4: Peek at the latest coupon code
    - 5: Search for a specific coupon code
    - 6: Exit

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

#define MAX_COUPON_CODE_LENGTH 100

struct Node
{
   char couponCode[MAX_COUPON_CODE_LENGTH];
   struct Node* next;
} *top = NULL;

void push(const char* couponCode);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* couponCode);

int main()
```

```
{
    int option;
    char couponCode[MAX_COUPON_CODE_LENGTH];
    do
    {
        printf("\n--- Coupon Code Management ---\n");
        printf("1: Add a new coupon code (push)\n");
        printf("2: Remove the last coupon code (pop)\n");
        printf("3: View all available coupon codes\n");
        printf("4: Peek at the latest coupon code\n");
        printf("5: Search for a specific coupon code\n");
        printf("6: Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the coupon code: ");
                    scanf(" %[^\n]", couponCode);
                    push(couponCode);
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: peek(*top, 1);
                    break;
            case 5: printf("Enter the coupon code to search for: ");
```

```
                scanf(" %[^\n]", couponCode);
                int position = search(couponCode);
                if (position != -1)
                    printf("Coupon code found at position %d\n", position);
                else
                    printf("Coupon code not found\n");
                break;
            case 6: printf("Exiting the management\n");
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void push(const char* couponCode)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->couponCode, couponCode);
        t->next = top;
        top = t;
        printf("Coupon code added: %s\n", couponCode);
    }
```

```
}

int pop()
{
    if (top == NULL)
    {
        printf("No coupon codes to remove (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Removed coupon code: %s\n", temp->couponCode);
    free(temp);
    return 0;
}

void display()
{
    if (top == NULL)
    {
        printf("No available coupon codes\n");
        return;
    }
    struct Node* current = top;
    printf("Available coupon codes:\n");
    while (current != NULL)
    {
```

```
        printf("%s\n", current->couponCode);
        current = current->next;
    }
}

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
        {
            printf("Coupon code at index %d: %s\n", position, current->couponCode);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* couponCode)
{
    struct Node* current = top;
    int position = 1;
```

```
    while (current != NULL)
    {
        if (strcmp(current->couponCode, couponCode) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

10. Shipping Status Tracker: Develop a stack to track shipping status updates using a linked list. Implement a switch-case menu with options:
    - 1: Add a shipping status update (push)
    - 2: Remove the last update (pop)
    - 3: View all shipping status updates
    - 4: Peek at the latest update
    - 5: Search for a specific update
    - 6: Exit

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

#define MAX_STATUS_LENGTH 100

struct Node
{
```

```
    char status[MAX_STATUS_LENGTH];
    struct Node* next;
} *top = NULL;

void push(const char* status);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* status);

int main()
{
    int option;
    char status[MAX_STATUS_LENGTH];
    do
    {
        printf("\n--- Shipping Status Tracker ---\n");
        printf("1: Add a shipping status update (push)\n");
        printf("2: Remove the last update (pop)\n");
        printf("3: View all shipping status updates\n");
        printf("4: Peek at the latest shipping status update\n");
        printf("5: Search for a specific status update\n");
        printf("6: Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
```

```
            case 1: printf("Enter the shipping status update: ");
                    scanf(" %[^\n]", status);
                    push(status);
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: peek(*top, 1);
                    break;
            case 5: printf("Enter the status update to search for: ");
                    scanf(" %[^\n]", status);
                    int position = search(status);
                    if (position != -1)
                       printf("Status update found at position %d\n", position);
                    else
                       printf("Status update not found\n");
                    break;
            case 6: printf("Exiting the tracker\n");
                    break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void push(const char* status)
```

```
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->status, status);
        t->next = top;
        top = t;
        printf("Status update added: %s\n", status);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No status updates to remove (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Removed status update: %s\n", temp->status);
    free(temp);
    return 0;
}
```

```
void display()
{
   if (top == NULL)
   {
      printf("No status updates available\n");
      return;
   }
   struct Node* current = top;
   printf("Shipping status updates:\n");
   while (current != NULL)
   {
      printf("%s\n", current->status);
      current = current->next;
   }
}

int peek(struct Node st, int index)
{
   struct Node* current = &st;
   int position = 1;
   while (current != NULL)
   {
      if (position == index)
      {
            printf("Status update at index %d: %s\n", position, current-
>status);
         return 0;
      }
```

```
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* status)
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->status, status) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

11. User Review Management: Use a stack to manage user reviews for products using a linked list. Include a switch-case menu with options:
    - 1: Add a new review (push)
    - 2: Remove the last review (pop)
    - 3: Display all reviews
    - 4: Peek at the latest review

- 5: Search for a specific review
- 6: Exit

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

#define MAX_REVIEW_LENGTH 200

struct Node
{
    char review[MAX_REVIEW_LENGTH];
    struct Node* next;
} *top = NULL;

void push(const char* review);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* review);

int main()
{
    int option;
    char review[MAX_REVIEW_LENGTH];
    do
    {
```

```
printf("\n--- User Review Management ---\n");
printf("1: Add a new review (push)\n");
printf("2: Remove the last review (pop)\n");
printf("3: Display all reviews\n");
printf("4: Peek at the latest review\n");
printf("5: Search for a specific review\n");
printf("6: Exit\n");
printf("Enter the option: ");
scanf("%d", &option);
switch (option)
{
   case 1: printf("Enter the review: ");
           scanf(" %[^\n]", review);
           push(review);
           break;
   case 2: pop();
           break;
   case 3: display();
           break;
   case 4: peek(*top, 1);
           break;
   case 5: printf("Enter the review to search for: ");
           scanf(" %[^\n]", review);
           int position = search(review);
           if (position != -1)
              printf("Review found at position %d\n", position);
           else
```

```
                printf("Review not found\n");
                break;
            case 6: printf("Exiting the management\n");
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void push(const char* review)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->review, review);
        t->next = top;
        top = t;
        printf("Review added: %s\n", review);
    }
}

int pop()
{
    if (top == NULL)
```

```
    {
        printf("No reviews to remove (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Removed review: %s\n", temp->review);
    free(temp);
    return 0;
}

void display()
{
    if (top == NULL)
    {
        printf("No reviews available\n");
        return;
    }
    struct Node* current = top;
    printf("User reviews:\n");
    while (current != NULL)
    {
        printf("%s\n", current->review);
        current = current->next;
    }
}
```

```
int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
        {
            printf("Review at index %d: %s\n", position, current->review);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* review)
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->review, review) == 0)
            return position;
        current = current->next;
```

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

12. Promotion Notification System: Create a stack for managing promotional notifications using a linked list. Use a switch-case menu with options:
    - 1: Add a new notification (push)
    - 2: Remove the last notification (pop)
    - 3: View all notifications
    - 4: Peek at the latest notification
    - 5: Search for a specific notification
    - 6: Exit

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

#define MAX_NOTIFICATION_LENGTH 200

struct Node
{
    char notification[MAX_NOTIFICATION_LENGTH];
    struct Node* next;
} *top = NULL;

void push(const char* notification);
```

```
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* notification);

int main()
{
   int option;
   char notification[MAX_NOTIFICATION_LENGTH];
   do
   {
      printf("\n--- Promotion Notification System ---\n");
      printf("1: Add a new notification (push)\n");
      printf("2: Remove the last notification (pop)\n");
      printf("3: View all notifications\n");
      printf("4: Peek at the latest notification\n");
      printf("5: Search for a specific notification\n");
      printf("6: Exit\n");
      printf("Enter the option: ");
      scanf("%d", &option);
      switch (option)
      {
         case 1: printf("Enter the notification: ");
                 scanf(" %[^\n]", notification);
                 push(notification);
                 break;
         case 2: pop();
```

```
            break;
        case 3: display();
            break;
        case 4: peek(*top, 1);
            break;
        case 5: printf("Enter the notification to search for: ");
            scanf(" %[^\n]", notification);
            int position = search(notification);
            if (position != -1)
                printf("Notification found at position %d\n", position);
            else
                printf("Notification not found\n");
            break;
        case 6: printf("Exiting the system\n");
            break;
        default: printf("Invalid option\n");
      }
   } while (option != 6);
   return 0;
}

void push(const char* notification)
{
   struct Node* t = malloc(sizeof(struct Node));
   if (t == NULL)
      printf("Stack Overflow\n");
   else
```

```
    {
        strcpy(t->notification, notification);
        t->next = top;
        top = t;
        printf("Notification added: %s\n", notification);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No notifications to remove (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Removed notification: %s\n", temp->notification);
    free(temp);
    return 0;
}

void display()
{
    if (top == NULL)
    {
        printf("No notifications available\n");
```

```
        return;
    }
    struct Node* current = top;
    printf("Promotion notifications:\n");
    while (current != NULL)
    {
        printf("%s\n", current->notification);
        current = current->next;
    }
}

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
        {
                printf("Notification at index %d: %s\n", position, current-
>notification);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
```

```
}

int search(const char* notification)
{
   struct Node* current = top;
   int position = 1;
   while (current != NULL)
   {
      if (strcmp(current->notification, notification) == 0)
         return position;
      current = current->next;
      position++;
   }
   return -1;
}
```

13. Product Viewing History: Implement a stack to track the viewing history of products using a linked list. Include a switch-case menu with options:
    - 1: Add a product to viewing history (push)
    - 2: Remove the last viewed product (pop)
    - 3: Display all viewed products
    - 4: Peek at the most recent product viewed
    - 5: Search for a specific product
    - 6: Exit

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

```
#include <string.h>

#define MAX_PRODUCT_NAME_LENGTH 100

struct Node
{
   char product[MAX_PRODUCT_NAME_LENGTH];
   struct Node* next;
} *top = NULL;

void push(const char* product);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* product);

int main()
{
   int option;
   char product[MAX_PRODUCT_NAME_LENGTH];
   do
   {
      printf("\n--- Product Viewing History ---\n");
      printf("1: Add a product to viewing history (push)\n");
      printf("2: Remove the last viewed product (pop)\n");
      printf("3: Display all viewed products\n");
      printf("4: Peek at the most recent product viewed\n");
```

```
printf("5: Search for a specific product\n");
printf("6: Exit\n");
printf("Enter the option: ");
scanf("%d", &option);
switch (option)
{
   case 1: printf("Enter the product name: ");
           scanf(" %[^\n]", product);
           push(product);
           break;
   case 2: pop();
           break;
   case 3: display();
           break;
   case 4: peek(*top, 1);
           break;
   case 5: printf("Enter the product name to search for: ");
           scanf(" %[^\n]", product);
           int position = search(product);
           if (position != -1)
              printf("Product found at position %d\n", position);
           else
              printf("Product not found\n");
           break;
   case 6: printf("Exiting the system\n");
           break;
   default: printf("Invalid option\n");
```

```
        }
    } while (option != 6);
    return 0;
}

void push(const char* product)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->product, product);
        t->next = top;
        top = t;
        printf("Product added to viewing history: %s\n", product);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No products to remove (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
```

```
    top = top->next;
      printf("Removed product from viewing history: %s\n", temp->product);
    free(temp);
    return 0;
}

void display()
{
    if (top == NULL)
    {
        printf("No products in viewing history\n");
        return;
    }
    struct Node* current = top;
    printf("Viewed products:\n");
    while (current != NULL)
    {
        printf("%s\n", current->product);
        current = current->next;
    }
}

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
```

```
    {
        if (position == index)
        {
            printf("Product at index %d: %s\n", position, current->product);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* product)
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->product, product) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

14. Cart Item Management: Develop a stack to manage items in a shopping cart using a linked list. Use a switch-case menu with options:
    - 1: Add an item to the cart (push)
    - 2: Remove the last item (pop)
    - 3: View all cart items
    - 4: Peek at the last added item
    - 5: Search for a specific item in the cart
    - 6: Exit

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

#define MAX_ITEM_NAME_LENGTH 100

struct Node
{
   char item[MAX_ITEM_NAME_LENGTH];
   struct Node* next;
} *top = NULL;

void push(const char* item);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* item);

int main()
```

```
{
    int option;
    char item[MAX_ITEM_NAME_LENGTH];
    do
    {
        printf("\n--- Cart Item Management ---\n");
        printf("1: Add an item to the cart (push)\n");
        printf("2: Remove the last item (pop)\n");
        printf("3: View all cart items\n");
        printf("4: Peek at the last added item\n");
        printf("5: Search for a specific item in the cart\n");
        printf("6: Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
            case 1: printf("Enter the item name: ");
                    scanf(" %[^\n]", item);
                    push(item);
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: peek(*top, 1);
                    break;
            case 5: printf("Enter the item name to search for: ");
```

```
                scanf(" %[^\n]", item);
                int position = search(item);
                if (position != -1)
                    printf("Item found at position %d\n", position);
                else
                    printf("Item not found\n");
                break;
            case 6: printf("Exiting the management\n");
                break;
            default: printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void push(const char* item)
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->item, item);
        t->next = top;
        top = t;
        printf("Item added to the cart: %s\n", item);
    }
```

```
}

int pop()
{
    if (top == NULL)
    {
        printf("No items to remove (cart is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Removed item from the cart: %s\n", temp->item);
    free(temp);
    return 0;
}

void display()
{
    if (top == NULL)
    {
        printf("No items in the cart\n");
        return;
    }
    struct Node* current = top;
    printf("Cart items:\n");
    while (current != NULL)
    {
```

```
        printf("%s\n", current->item);
        current = current->next;
    }
}

int peek(struct Node st, int index)
{
    struct Node* current = &st;
    int position = 1;
    while (current != NULL)
    {
        if (position == index)
        {
            printf("Item at index %d: %s\n", position, current->item);
            return 0;
        }
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* item)
{
    struct Node* current = top;
    int position = 1;
```

```
    while (current != NULL)
    {
        if (strcmp(current->item, item) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```

15. Payment History: Implement a stack to record payment history using a linked list. Include a switch-case menu with options:
    - 1: Add a new payment record (push)
    - 2: Remove the last payment record (pop)
    - 3: View all payment records
    - 4: Peek at the latest payment record
    - 5: Search for a specific payment record
    - 6: Exit

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

#define MAX_RECORD_LENGTH 100

struct Node
{
```

```
    char paymentRecord[MAX_RECORD_LENGTH];
    struct Node* next;
} *top = NULL;

void push(const char* record);
int pop();
void display();
int peek(struct Node st, int index);
int search(const char* record);

int main()
{
    int option;
    char record[MAX_RECORD_LENGTH];
    do
    {
        printf("\n--- Payment History Management ---\n");
        printf("1: Add a new payment record (push)\n");
        printf("2: Remove the last payment record (pop)\n");
        printf("3: View all payment records\n");
        printf("4: Peek at the latest payment record\n");
        printf("5: Search for a specific payment record\n");
        printf("6: Exit\n");
        printf("Enter the option: ");
        scanf("%d", &option);
        switch (option)
        {
```

```
            case 1: printf("Enter the payment record: ");
                    scanf(" %[^\n]", record);
                    push(record);
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: peek(*top, 1);
                    break;
            case 5: printf("Enter the payment record to search for: ");
                    scanf(" %[^\n]", record);
                    int position = search(record);
                    if (position != -1)
                     printf("Payment record found at position %d\n", position);
                    else
                       printf("Payment record not found\n");
                    break;
            case 6: printf("Exiting the system\n");
                    break;
            default:printf("Invalid option\n");
        }
    } while (option != 6);
    return 0;
}

void push(const char* record)
```

```
{
    struct Node* t = malloc(sizeof(struct Node));
    if (t == NULL)
        printf("Stack Overflow\n");
    else
    {
        strcpy(t->paymentRecord, record);
        t->next = top;
        top = t;
        printf("Payment record added: %s\n", record);
    }
}

int pop()
{
    if (top == NULL)
    {
        printf("No payment records to remove (stack is empty)\n");
        return -1;
    }
    struct Node* temp = top;
    top = top->next;
    printf("Removed payment record: %s\n", temp->paymentRecord);
    free(temp);
    return 0;
}
```

```
void display()
{
   if (top == NULL)
   {
      printf("No payment records available\n");
      return;
   }
   struct Node* current = top;
   printf("Payment records:\n");
   while (current != NULL)
   {
      printf("%s\n", current->paymentRecord);
      current = current->next;
   }
}

int peek(struct Node st, int index)
{
   struct Node* current = &st;
   int position = 1;
   while (current != NULL)
   {
      if (position == index)
      {
          printf("Payment record at index %d: %s\n", position, current-
>paymentRecord);
         return 0;
      }
```

```
        current = current->next;
        position++;
    }
    printf("Invalid index\n");
    return -1;
}

int search(const char* record)
{
    struct Node* current = top;
    int position = 1;
    while (current != NULL)
    {
        if (strcmp(current->paymentRecord, record) == 0)
            return position;
        current = current->next;
        position++;
    }
    return -1;
}
```