STACK

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

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

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

int main() {
    struct Stack st;
    create(&st);
    push(&st, 5);   //0
    push(&st, 10);   //1
    push(&st, 15);  //2
    push(&st, 20);  // 3
    display(st);

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


    int index =2;
    int peekValue = peek(st,index);
    if(peekValue !=-1){
        printf("The value at position %d from top = %d\n",index,peekValue);
    }

    return 0;
```

```
}

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

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

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

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

int isEmpty(struct Stack st) {
```

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

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

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


int peek(struct Stack st,int index){
    return st.s[st.top - index+1];   // 3 - 2+1 =>2  ==> st.s[2]

}
```

STACK WITH ARRAY

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>

struct Stack {
    char **path;
    int top;
```

```
    int size;
};


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

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

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

    create(&st, size);

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

        switch (choice) {
            case 1:
                printf("Enter the path to add: ");
                getchar();
                scanf("%[^\n]", item);
                push(&st, item);
                break;
```

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

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

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

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

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

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

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

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

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

```
}

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

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>

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

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

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

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

    create(&st, size);

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

        switch (choice)
        {
        case 1:
            printf("ENter the deployement:  ");
            scanf(" %[^\n]",de);
            push(&st,de);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("ENter the deployement:  ");
            scanf(" %[^\n]",de);
```

```
            search(&st,de);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.deploy);
            exit(0);

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

    }
  return 0;
}


void create(struct Stack *st,int size){
    st->size =size;
    st->top =-1;
    st->deploy = (char **)malloc(size * sizeof(char *));

    if(st->deploy == NULL){
        printf("Memory allocation failed \n");
        exit(1);
    }
}
int isFull(struct Stack *st){
    return st->top == st->size -1;
}
int isEmpty(struct Stack *st){
    return st->top == -1;
}
void push(struct Stack *st,char de[]){
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->deploy[st->top] = (char *)malloc((strlen(de) + 1) * sizeof(char));
    strcpy(st->deploy[st->top],de);
     printf("Item '%s' added to stack\n", de);
```

```
}
void pop(struct Stack *st){
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Item '%s' removed from stack \n",st->deploy[st->top]);
    free(st->deploy[st->top]);
    st->top--;
}
void display(struct Stack *st){
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Current stack \n");
    for(int i= st->top;i>=0;i--){
        printf("%d : %s\n",i+1,st->deploy[i]);
    }
}
void peek(struct Stack *st){
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Top item : %s\n",st->deploy[st->top]);
}
void search(struct Stack *st,char de[]){
    for(int i=st->top;i>=0;i--){
        if(strcmp(st->deploy[i],de)==0){
            printf("Item '%s' has found at position %d\n",de,i+1);
            return;
        }
    }
    printf("Item '%s' not found in the stack\n", de);
}
```

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>

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

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

int main() {
    struct Stack st;
    int choice, size;
    char item[100];  // Allocate enough space for input

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

    create(&st, size);

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

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

        switch (choice) {
            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:
                peek(&st);
                break;
            case 5:
                printf("Enter the checklist item to search: ");
                scanf(" %[^\n]", item);
                search(&st, item);
                break;
            case 6:
                printf("Exiting .....\n");
                free(st.checklist);
                exit(0);

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

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->checklist = (char **)malloc(size * sizeof(char *));
```

```
    if (st->checklist == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
}

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

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

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

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

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

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

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

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

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>

struct Stack {
    char **data;
```

```
    int top;
    int size;
};

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

int main() {
    struct Stack st;
    int choice, size;
    char data[100];  // Allocate enough space for telemetry data input

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

    create(&st, size);

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

        switch (choice) {
            case 1:
                printf("Enter the telemetry data: ");
                scanf(" %[^\n]", data);  // Allow spaces in input
                push(&st, data);
                break;
            case 2:
```

```
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the telemetry data to search: ");
            scanf(" %[^\n]", data);
            search(&st, data);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.data);  // Free the allocated memory for telemetry data
            exit(0);

        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}
void create(struct Stack *st,int size){
   st->size = size;
   st->top = -1;
   st->data = (char **)malloc (size *sizeof(char *));
}


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

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

void push(struct Stack *st,char data[]){
   if (isFull(st)) {
```

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

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

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

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

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

```
            printf("Telemetry data '%s' found at position %d\n", data, i + 1);
            return;
        }
    }
    printf("Telemetry data '%s' not found in the stack\n", data);
}
```

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>

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

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

int main() {
    struct Stack st;
    int choice, size;
    char task[100];
    printf("Enter the maximum size of the task stack: ");
```

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

create(&st, size);

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

    switch (choice) {
        case 1:
            printf("Enter the task description: ");
            scanf(" %[^\n]", task);  // Allow spaces in input
            push(&st, task);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the task to search: ");
            scanf(" %[^\n]", task);
            search(&st, task);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.tasks);  // Free the allocated memory for tasks
            exit(0);

        default:
```

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

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

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

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

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

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

```
    free(st->tasks[st->top]);
    st->top--;
}

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

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

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

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>

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

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

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

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

    create(&st, size);

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

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

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

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

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

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

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

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

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

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

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

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>

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

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

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

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

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

    create(&st, size);

    while (1) {
        printf("\n--- Aircraft Maintenance Logs ---\n");
        printf("1: Add a new log (push)\n");
        printf("2: Remove the last log (pop)\n");
        printf("3: View all maintenance logs\n");
        printf("4: Peek at the latest maintenance log\n");
        printf("5: Search for a specific maintenance log\n");
        printf("6: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the maintenance log: ");
                scanf(" %[^\n]", log);  // Allow spaces in input
                push(&st, log);
                break;
            case 2:
                pop(&st);
                break;
            case 3:
                display(&st);
                break;
            case 4:
                peek(&st);
```

```
            break;
        case 5:
            printf("Enter the maintenance log to search: ");
            scanf(" %[^\n]", log);
            search(&st, log);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.logs);  // Free the allocated memory for logs
            exit(0);

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

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

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

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

void push(struct Stack *st, char log[]) {
   if (isFull(st)) {
      printf("Stack Overflow\n");
      return;
   }
```

```
    st->top++;
    st->logs[st->top] = (char *)malloc((strlen(log) + 1) * sizeof(char));
    strcpy(st->logs[st->top], log);
    printf("Maintenance log '%s' added\n", log);
}

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

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

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

void search(struct Stack *st, char log[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->logs[i], log) == 0) {
            printf("Maintenance log '%s' found at position %d\n", log, i + 1);
            return;
        }
```

```
    }
    printf("Maintenance log '%s' not found\n", log);
}
```

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>

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

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

int main() {
    struct Stack st;
    int choice, size;
    char step[100];  // Allocate enough space for docking procedure steps

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

    create(&st, size);
```

```
while (1) {
    printf("\n--- Spacecraft Docking Procedure ---\n");
    printf("1: Push a new step\n");
    printf("2: Pop the last step\n");
    printf("3: Display the procedure steps\n");
    printf("4: Peek at the next step in the procedure\n");
    printf("5: Search for a specific step\n");
    printf("6: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the docking procedure step: ");
            scanf(" %[^\n]", step);  // Allow spaces in input
            push(&st, step);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the docking procedure step to search: ");
            scanf(" %[^\n]", step);
            search(&st, step);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.steps);  // Free the allocated memory for steps
            exit(0);

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

```
    return 0;
}

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

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

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

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

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

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

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

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

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>

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

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

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

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

    create(&st, size);

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

        switch (choice) {
            case 1:
```

```
            printf("Enter the command: ");
            scanf(" %[^\n]", command);  // Allow spaces in input
            push(&st, command);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the command to search: ");
            scanf(" %[^\n]", command);
            search(&st, command);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.commands);  // Free the allocated memory for commands
            exit(0);

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

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

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

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

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

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

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

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

void peek(struct Stack *st) {
    if (isEmpty(st)) {
```

```
        printf("Stack is empty\n");
        return;
    }
    printf("Most recent command: %s\n", st->commands[st->top]);
}

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

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>

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

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

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

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

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

    create(&st, size);

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

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

```
            break;
        case 5:
            printf("Enter the event to search: ");
            scanf(" %[^\n]", event);
            search(&st, event);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.events);  // Free the allocated memory for events
            exit(0);

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

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

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

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

void push(struct Stack *st, char event[]) {
   if (isFull(st)) {
      printf("Stack Overflow\n");
      return;
   }
```

```
    st->top++;
    st->events[st->top] = (char *)malloc((strlen(event) + 1) * sizeof(char));
    strcpy(st->events[st->top], event);
    printf("Event '%s' added\n", event);
}

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

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

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

void search(struct Stack *st, char event[]) {
    for (int i = st->top; i >= 0; i--) {
        if (strcmp(st->events[i], event) == 0) {
            printf("Event '%s' found at position %d\n", event, i + 1);
            return;
        }
```

```
    }
    printf("Event '%s' not found\n", event);
}
```

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>

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

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

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

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

```
create(&st, size);

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

    switch (choice) {
        case 1:
            printf("Enter the maneuver: ");
            scanf(" %[^\n]", maneuver);
            push(&st, maneuver);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the maneuver to search: ");
            scanf(" %[^\n]", maneuver);
            search(&st, maneuver);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.maneuvers);  // Free the allocated memory for maneuvers
            exit(0);

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

```
        }
    }
    return 0;
}

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

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

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

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

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

```
    st->top--;
}

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

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

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

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

o 6: Exit

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

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

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

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

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

    create(&st, size);

    while (1) {
        printf("\n--- Satellite Operation Commands ---\n");
        printf("1: Push a new command\n");
        printf("2: Pop the last command\n");
        printf("3: View the operation commands\n");
        printf("4: Peek at the most recent command\n");
        printf("5: Search for a specific command\n");
        printf("6: Exit\n");
        printf("Enter your choice: ");
```

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

        switch (choice) {
            case 1:
                printf("Enter the command: ");
                scanf(" %[^\n]", command);  // Allow spaces in input
                push(&st, command);
                break;
            case 2:
                pop(&st);
                break;
            case 3:
                display(&st);
                break;
            case 4:
                peek(&st);
                break;
            case 5:
                printf("Enter the command to search: ");
                scanf(" %[^\n]", command);
                search(&st, command);
                break;
            case 6:
                printf("Exiting .....\n");
                free(st.commands);  // Free the allocated memory for commands
                exit(0);

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

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

```
    }
}

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

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

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

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

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

```
}

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

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

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>

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

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

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

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

    create(&st, size);

    while (1) {
        printf("\n--- Emergency Procedures for Spacecraft ---\n");
        printf("1: Add a procedure\n");
        printf("2: Remove the last procedure\n");
        printf("3: View all procedures\n");
        printf("4: Peek at the next procedure\n");
        printf("5: Search for a specific procedure\n");
        printf("6: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the procedure to search: ");
            scanf(" %[^\n]", procedure);
            search(&st, procedure);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.procedures);  // Free the allocated memory for procedures
            exit(0);

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

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

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

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

void push(struct Stack *st, char procedure[]) {
   if (isFull(st)) {
```

```
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->procedures[st->top] = (char *)malloc((strlen(procedure) + 1) * sizeof(char));
    strcpy(st->procedures[st->top], procedure);
    printf("Procedure '%s' added\n", procedure);
}

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

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

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

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

```
            printf("Procedure '%s' found at position %d\n", procedure, i + 1);
            return;
        }
    }
    printf("Procedure '%s' not found\n", procedure);
}
```

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>

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

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

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

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

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

create(&st, size);

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

    switch (choice) {
        case 1:
            printf("Enter the astronaut activity: ");
            scanf(" %[^\n]", activity);  // Allow spaces in input
            push(&st, activity);
            break;
        case 2:
            pop(&st);
            break;
        case 3:
            display(&st);
            break;
        case 4:
            peek(&st);
            break;
        case 5:
            printf("Enter the activity to search: ");
            scanf(" %[^\n]", activity);
            search(&st, activity);
            break;
        case 6:
            printf("Exiting .....\n");
            free(st.activities);  // Free the allocated memory for activities
            exit(0);

        default:
```

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

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

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

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

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

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

```
    free(st->activities[st->top]);
    st->top--;
}

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

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

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

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>

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

void create(struct Stack *st, int size);
int isFull(struct Stack *st);
int isEmpty(struct Stack *st);
void push(struct Stack *st, float fuelEntry);
void pop(struct Stack *st);
void display(struct Stack *st);
void peek(struct Stack *st);
void search(struct Stack *st, float fuelEntry);

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

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

    create(&st, size);

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

```
        switch (choice) {
            case 1:
                printf("Enter the fuel usage entry: ");
                scanf("%f", &fuelEntry);
                push(&st, fuelEntry);
                break;
            case 2:
                pop(&st);
                break;
            case 3:
                display(&st);
                break;
            case 4:
                peek(&st);
                break;
            case 5:
                printf("Enter the fuel usage entry to search: ");
                scanf("%f", &fuelEntry);
                search(&st, fuelEntry);
                break;
            case 6:
                printf("Exiting .....\n");
                free(st.fuelUsage);  // Free the allocated memory for fuel usage entries
                exit(0);

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

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

```
}

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

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

void push(struct Stack *st, float fuelEntry) {
    if (isFull(st)) {
        printf("Stack Overflow\n");
        return;
    }
    st->top++;
    st->fuelUsage[st->top] = fuelEntry;
    printf("Fuel usage entry %.2f added\n", fuelEntry);
}

void pop(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Fuel usage entry %.2f removed\n", st->fuelUsage[st->top]);
    st->top--;
}

void display(struct Stack *st) {
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Current fuel usage data:\n");
    for (int i = st->top; i >= 0; i--) {
        printf("%d : %.2f\n", i + 1, st->fuelUsage[i]);
    }
}

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

```
    if (isEmpty(st)) {
        printf("Stack is empty\n");
        return;
    }
    printf("Latest fuel usage entry: %.2f\n", st->fuelUsage[st->top]);
}

void search(struct Stack *st, float fuelEntry) {
    for (int i = st->top; i >= 0; i--) {
        if (st->fuelUsage[i] == fuelEntry) {
            printf("Fuel usage entry %.2f found at position %d\n", fuelEntry, i + 1);
            return;
        }
    }
    printf("Fuel usage entry %.2f not found\n", fuelEntry);
}
```

STACK WITH LINKEDLIST

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

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

void push(int);
int pop();
void display();
int main(){
    push(20);
    push(30);
    push(40);
    display();
   int popValue = pop();
   printf("popped value : %d\n",popValue);
    display();
```

```
    return 0;
}

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

    if(t ==NULL){
        printf("Stack overflow \n");
    }
    else{
        t->data =x;
        t->next =top;
        top =t;
    }
}
int pop(){
    struct Node *t;
    int x = -1;
    if(top ==NULL){
        printf("Stack is EMpty\n");
    }
    else{
        t =top;
        top = top->next;
        x= t->data;
        free(t);
    }
    return x;
}
void display(){
    struct Node *temp;
    temp =top;
    while(temp!=NULL){
        printf("%d -> ",temp->data);

        temp = temp->next;
    }
    printf("\n");
}
```

PROBLEMS

1) Order Processing System: Implement a stack-based system using a linked list to manage order processing. Use a switch-case menu with options:
1: Add a new order (push)
2: Process the last order (pop)
3: Display all pending orders
4: Peek at the next order to be processed
5: Search for a specific order
6: Exit

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

struct Stack{
    int id;
    char order[50];
    struct Stack *next;
}*top =NULL;

void push(int id,char order[]);
void pop();
void display();
void peek();
void search(int id);
int isEmpty();

int main(){
    int choice,id;
    char order[50];
    while (1) {
        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 your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1:
                printf("ENter order ID: ");
                scanf("%d",&id);
                printf("ENter the order ");
                scanf(" %[^\n]",order);
                push(id,order);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
                printf("ENter order ID to search ");
                scanf("%d",&id);
                search(id);
                break;
            case 6:
                printf("Exiting.....\n");
                exit(0);
            default:
                printf("Invalid choice \n");

        }
    }
    return 0;
}

int isEmpty(){
    return top == NULL;
}
void push(int id,char order[]){
```

```
    struct Stack *new = (struct Stack *)malloc (sizeof(struct Stack));
    if(new ==NULL){
        printf("Memmory ALlocation failed\n");
        return;
    }
    new->id =id;
    strcpy(new->order,order);
    new->next = top;
    top = new;
    printf("Order with ID %d added: %s\n", id, order);
}
void pop(){
    if (isEmpty()) {
        printf("No orders to process (stack is empty)\n");
        return;
    }
    struct Stack *temp = top;
    top = top->next;
    printf("Order with ID %d processed: %s\n", temp->id, temp->order);
    free(temp);
}
void display(){
    if (isEmpty()) {
        printf("No orders to process (stack is empty)\n");
        return;
    }
    struct Stack *temp = top;
    while(temp!= NULL){
        printf("Order ID: %d, Order: %s\n", temp->id, temp->order);
        temp = temp->next;
    }
}
void peek(){
     if (isEmpty()) {
        printf("No orders to peek at (stack is empty)\n");
        return;
    }
    printf("Next order to process: ID %d, Order: %s\n", top->id, top->order);
}

void search(int id){
```

```
    if (isEmpty()) {
        printf("No orders to search\n");
        return;
    }
struct Stack *temp = top;
    while (temp != NULL) {
        if (temp->id == id) {
            printf("Order found: ID %d, Order: %s\n", temp->id, temp->order);
            return;
        }
        temp = temp->next;
    }
    printf("Order with ID %d not found\n", id);
}
```

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

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

struct Ticket {
    int ticket_id;
    char description[100];
    struct Ticket *next;
} *top = NULL;

void push(int ticket_id, char description[]);
```

```
void pop();
void display();
void peek();
void search(int ticket_id);
int isEmpty();

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

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

        switch (choice) {
            case 1:
                printf("Enter ticket ID: ");
                scanf("%d", &ticket_id);
                printf("Enter ticket description: ");
                scanf(" %[^\n]", description); // Allow spaces in the ticket description
                push(ticket_id, description);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
                printf("Enter the ticket ID to search for: ");
                scanf("%d", &ticket_id);
```

```
            search(ticket_id);
            break;
        case 6:
            printf("Exiting .....\n");
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }

   return 0;
}

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

void push(int ticket_id, char description[]){
   struct Ticket *new = (struct Ticket *)malloc (sizeof(struct Ticket));
   if(new == NULL){
      printf("Memory allocation failed\n");
      return;
   }
   new->ticket_id = ticket_id;
   strcpy(new->description,description);
   new->next =top;
   top =new;
   printf("Ticket with ID %d added: %s\n", ticket_id, description);
}
void pop(){
   if (isEmpty()) {
      printf("No tickets to resolve (stack is empty)\n");
      return;
   }

   struct Ticket *temp = top;
   top = top->next;
   printf("Ticket with ID %d resolved: %s\n", temp->ticket_id, temp->description);
   free(temp);
}
void display(){
```

```
    if (isEmpty()) {
        printf("No pending tickets\n");
        return;
    }
    struct Ticket *temp = top;
    printf("Pending Tickets:\n");
    while (temp != NULL) {
        printf("Ticket ID: %d, Description: %s\n", temp->ticket_id, temp->description);
        temp = temp->next;
    }
}
void peek(){
   if (isEmpty()) {
        printf("No tickets to peek at (stack is empty)\n");
        return;
    }
    printf("Latest ticket to resolve: ID %d, Description: %s\n", top->ticket_id,
top->description);
}
void search(int ticket_id){
     if (isEmpty()) {
        printf("No tickets to search\n");
        return;
    }
    struct Ticket *temp = top;
    while (temp != NULL) {
        if (temp->ticket_id == ticket_id) {
            printf("Ticket found: ID %d, Description: %s\n", temp->ticket_id,
temp->description);
            return;
        }
        temp = temp->next;
    }
    printf("Ticket with ID %d not found\n", ticket_id);
}
```

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>

struct ReturnRequest {
    int request_id;
    char product_name[100];
    struct ReturnRequest *next;
} *top = NULL;

// Function prototypes
void push(int request_id, char product_name[]);
void pop();
void display();
void peek();
void search(int request_id);
int isEmpty();

int main() {
    int choice, request_id;
    char product_name[100];

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

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

```
            scanf("%d", &request_id);
            printf("Enter product name: ");
            scanf(" %[^\n]", product_name); // Allow spaces in product name
            push(request_id, product_name);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            printf("Enter the return request ID to search for: ");
            scanf("%d", &request_id);
            search(request_id);
            break;
        case 6:
            printf("Exiting .....\n");
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }

   return 0;
}


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

void push(int request_id,char product_name[]){
   struct ReturnRequest *newRequest = (struct ReturnRequest *)malloc(sizeof(struct
ReturnRequest));
   if (newRequest == NULL) {
      printf("Memory allocation failed\n");
      return;
```

```
    }
    newRequest->request_id = request_id;
    strcpy(newRequest->product_name,product_name);
    newRequest->next = top;
    top = newRequest;
    printf("Return request with ID %d added: %s\n", request_id, product_name);
}
void pop() {
    if (isEmpty()) {
        printf("No return requests to process (stack is empty)\n");
        return;
    }
    struct ReturnRequest *temp = top;
    top = top->next;
    printf("Return request with ID %d processed: %s\n", temp->request_id,
temp->product_name);
    free(temp);
}
void display() {
    if (isEmpty()) {
        printf("No pending return requests\n");
        return;
    }
    struct ReturnRequest *temp = top;
    printf("Pending Return Requests:\n");
    while (temp != NULL) {
        printf("Request ID: %d, Product Name: %s\n", temp->request_id,
temp->product_name);
        temp = temp->next;
    }
}
void peek() {
    if (isEmpty()) {
        printf("No return requests to peek at (stack is empty)\n");
        return;
    }
    printf("Next return request to process: ID %d, Product Name: %s\n", top->request_id,
top->product_name);
}

void search(int request_id) {
```

```
    if (isEmpty()) {
        printf("No return requests to search\n");
        return;
    }
    struct ReturnRequest *temp = top;
    while (temp != NULL) {
        if (temp->request_id == request_id) {
            printf("Return request found: ID %d, Product Name: %s\n", temp->request_id,
temp->product_name);
            return;
        }
        temp = temp->next;
    }
    printf("Return request with ID %d not found\n", request_id);
}
```

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>

struct Restock{
    int id;
    char name[50];
    struct Restock *next;
}*top =NULL;

int isEmpty();
void push(int id,char name[]);
void pop();
void display();
void peek();
```

```
void search(int id);

int main(){
    int choice,id;
    char name[50];

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

        switch(choice){
            case 1:
                printf("Enter order ID: ");
                scanf("%d",&id);
                printf("ENter order name: ");
                scanf(" %[^\n]s",name);
                push(id,name);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
                printf("Enter order Id: ");
                scanf("%d",&id);
                search(id);
                break;
            case 6:
                printf("Exiting.....\n");
```

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


int isEmpty(){
   return top == NULL;
}
void push(int id,char name[]){
   struct Restock *new = (struct Restock *) malloc (sizeof(struct Restock ));
   if( new==NULL){
      printf("Memory allocation failed \n");
      return;
   }
   new->id = id;
   strcpy(new->name,name);
   new->next =top;
   top = new;
   printf("The id %d , and order name : %s is added ",id,name);
}
void pop(){
   if(isEmpty()){
      printf("Stack is empty \n");
      return;
   }
   struct Restock *temp = top;
   top = top->next;
   printf("Restock removed from id %d order %s",temp->id,temp->name);
   free(temp);
}
void display(){
   if(isEmpty()){
      printf("Stack is empty \n");
      return;
   }
   struct Restock *temp = top;
   while(temp!= NULL){
```

```
        printf("Id : %d  order Name: %s",temp->id,temp->name);
        temp = temp->next;
    }
}
void peek(){
    if(isEmpty()){
        printf("Stack is empty \n");
        return;
    }
    printf("Restock of ID: %d order name : %s",top->id,top->name);
}
void search(int id){
    if(isEmpty()){
        printf("Stack is empty \n");
        return;
    }
    struct Restock *temp = top;
    while(temp!= NULL){
        if(temp->id==id){
            printf("The id : %d and order name %s has found \n",temp->id,temp->name);
            return;
        }
        temp = temp->next;
    }
    printf("Not found\n");
}
```

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>
```

```
struct FlashSaleDeal {
    int deal_id;
    char deal_description[100];
    struct FlashSaleDeal *next;
} *top = NULL;

void push(int deal_id, char deal_description[]);
void pop();
void display();
void peek();
void search(int deal_id);
int isEmpty();

int main() {
    int choice, deal_id;
    char deal_description[100];

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

        switch (choice) {
            case 1:
                printf("Enter deal ID: ");
                scanf("%d", &deal_id);
                printf("Enter deal description: ");
                scanf(" %[^\n]", deal_description);
                push(deal_id, deal_description);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
```

```
            break;
        case 4:
            peek();
            break;
        case 5:
            printf("Enter the deal ID to search for: ");
            scanf("%d", &deal_id);
            search(deal_id);
            break;
        case 6:
            printf("Exiting .....\n");
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}

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

void push(int deal_id, char deal_description[]) {
   struct FlashSaleDeal *newDeal = (struct FlashSaleDeal *)malloc(sizeof(struct
FlashSaleDeal));
   if (newDeal == NULL) {
      printf("Memory allocation failed\n");
      return;
   }
   newDeal->deal_id = deal_id;
   strcpy(newDeal->deal_description, deal_description);
   newDeal->next = top;
   top = newDeal;
   printf("Deal with ID %d added: %s\n", deal_id, deal_description);
}

void pop() {
   if (isEmpty()) {
      printf("No deals to remove\n");
      return;
```

```
    }
    struct FlashSaleDeal *temp = top;
    top = top->next;
    printf("Deal with ID %d removed: %s\n", temp->deal_id, temp->deal_description);
    free(temp);
}

void display() {
    if (isEmpty()) {
        printf("No active deals\n");
        return;
    }
    struct FlashSaleDeal *temp = top;
    printf("Active Deals:\n");
    while (temp != NULL) {
        printf("Deal ID: %d, Description: %s\n", temp->deal_id, temp->deal_description);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No deals to peek at\n");
        return;
    }
    printf("Latest deal: ID %d, Description: %s\n", top->deal_id, top->deal_description);
}

void search(int deal_id) {
    if (isEmpty()) {
        printf("No deals to search\n");
        return;
    }
    struct FlashSaleDeal *temp = top;
    while (temp != NULL) {
        if (temp->deal_id == deal_id) {
            printf("Deal found: ID %d, Description: %s\n", temp->deal_id,
temp->deal_description);
            return;
        }
        temp = temp->next;
```

```
    }
    printf("Deal with ID %d not found\n", deal_id);
}
```

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>

struct UserSession {
    int session_id;
    char session_info[100];
    struct UserSession *next;
} *top = NULL;

void push(int session_id, char session_info[]);
void pop();
void display();
void peek();
void search(int session_id);
int isEmpty();

int main() {
    int choice, session_id;
    char session_info[100];

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

        switch (choice) {
            case 1:
                printf("Enter session ID: ");
                scanf("%d", &session_id);
                printf("Enter session info: ");
                scanf(" %[^\n]", session_info);
                push(session_id, session_info);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
                printf("Enter the session ID to search for: ");
                scanf("%d", &session_id);
                search(session_id);
                break;
            case 6:
                printf("Exiting .....\n");
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

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

void push(int session_id, char session_info[]) {
```

```
    struct UserSession *newSession = (struct UserSession *)malloc(sizeof(struct
UserSession));
    if (newSession == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newSession->session_id = session_id;
    strcpy(newSession->session_info, session_info);
    newSession->next = top;
    top = newSession;
    printf("Session with ID %d added: %s\n", session_id, session_info);
}

void pop() {
    if (isEmpty()) {
        printf("No sessions to end\n");
        return;
    }
    struct UserSession *temp = top;
    top = top->next;
    printf("Session with ID %d ended: %s\n", temp->session_id, temp->session_info);
    free(temp);
}

void display() {
    if (isEmpty()) {
        printf("No active sessions\n");
        return;
    }
    struct UserSession *temp = top;
    printf("Active Sessions:\n");
    while (temp != NULL) {
        printf("Session ID: %d, Info: %s\n", temp->session_id, temp->session_info);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No sessions to peek at\n");
        return;
```

```
    }
    printf("Most recent session: ID %d, Info: %s\n", top->session_id, top->session_info);
}

void search(int session_id) {
    if (isEmpty()) {
        printf("No sessions to search\n");
        return;
    }
    struct UserSession *temp = top;
    while (temp != NULL) {
        if (temp->session_id == session_id) {
            printf("Session found: ID %d, Info: %s\n", temp->session_id,
temp->session_info);
            return;
        }
        temp = temp->next;
    }
    printf("Session with ID %d not found\n", session_id);
}
```

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>

struct WishlistItem {
    int product_id;
    char product_name[100];
```

```
    struct WishlistItem *next;
} *top = NULL;

void push(int product_id, char product_name[]);
void pop();
void display();
void peek();
void search(int product_id);
int isEmpty();

int main() {
    int choice, product_id;
    char product_name[100];

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

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

```
            break;
        case 5:
            printf("Enter the product ID to search for: ");
            scanf("%d", &product_id);
            search(product_id);
            break;
        case 6:
            printf("Exiting .....\n");
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}

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

void push(int product_id, char product_name[]) {
   struct WishlistItem *newItem = (struct WishlistItem *)malloc(sizeof(struct
WishlistItem));
   if (newItem == NULL) {
      printf("Memory allocation failed\n");
      return;
   }
   newItem->product_id = product_id;
   strcpy(newItem->product_name, product_name);
   newItem->next = top;
   top = newItem;
   printf("Product with ID %d added to wishlist: %s\n", product_id, product_name);
}

void pop() {
   if (isEmpty()) {
      printf("No items in the wishlist to remove\n");
      return;
   }
   struct WishlistItem *temp = top;
   top = top->next;
```

```
    printf("Product with ID %d removed from wishlist: %s\n", temp->product_id,
temp->product_name);
    free(temp);
}

void display() {
    if (isEmpty()) {
        printf("No items in the wishlist\n");
        return;
    }
    struct WishlistItem *temp = top;
    printf("Wishlist Items:\n");
    while (temp != NULL) {
        printf("Product ID: %d, Product Name: %s\n", temp->product_id,
temp->product_name);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No items to peek at in the wishlist\n");
        return;
    }
    printf("Most recent wishlist item: ID %d, Name: %s\n", top->product_id,
top->product_name);
}

void search(int product_id) {
    if (isEmpty()) {
        printf("No items in the wishlist to search\n");
        return;
    }
    struct WishlistItem *temp = top;
    while (temp != NULL) {
        if (temp->product_id == product_id) {
            printf("Product found: ID %d, Name: %s\n", temp->product_id,
temp->product_name);
            return;
        }
        temp = temp->next;
```

```
    }
    printf("Product with ID %d not found in wishlist\n", product_id);
}
```

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>

struct CheckoutStep {
    int step_id;
    char step_name[100];
    struct CheckoutStep *next;
} *top = NULL;

void push(int step_id, char step_name[]);
void pop();
void display();
void peek();
void search(int step_id);
int isEmpty();

int main() {
    int choice, step_id;
    char step_name[100];

    while (1) {
        printf("\n--- Checkout Process Management ---\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 your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter step ID: ");
                scanf("%d", &step_id);
                printf("Enter step name: ");
                scanf(" %[^\n]", step_name);
                push(step_id, step_name);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
                printf("Enter the step ID to search for: ");
                scanf("%d", &step_id);
                search(step_id);
                break;
            case 6:
                printf("Exiting .....\n");
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

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

```
void push(int step_id, char step_name[]) {
    struct CheckoutStep *newStep = (struct CheckoutStep *)malloc(sizeof(struct
CheckoutStep));
    if (newStep == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newStep->step_id = step_id;
    strcpy(newStep->step_name, step_name);
    newStep->next = top;
    top = newStep;
    printf("Checkout step with ID %d added: %s\n", step_id, step_name);
}

void pop() {
    if (isEmpty()) {
        printf("No checkout steps to remove\n");
        return;
    }
    struct CheckoutStep *temp = top;
    top = top->next;
    printf("Checkout step with ID %d removed: %s\n", temp->step_id, temp->step_name);
    free(temp);
}

void display() {
    if (isEmpty()) {
        printf("No checkout steps available\n");
        return;
    }
    struct CheckoutStep *temp = top;
    printf("Checkout Steps:\n");
    while (temp != NULL) {
        printf("Step ID: %d, Step Name: %s\n", temp->step_id, temp->step_name);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
```

```
        printf("No checkout steps to peek at\n");
        return;
    }
    printf("Current checkout step: ID %d, Name: %s\n", top->step_id, top->step_name);
}

void search(int step_id) {
    if (isEmpty()) {
        printf("No checkout steps to search\n");
        return;
    }
    struct CheckoutStep *temp = top;
    while (temp != NULL) {
        if (temp->step_id == step_id) {
            printf("Checkout step found: ID %d, Name: %s\n", temp->step_id,
temp->step_name);
            return;
        }
        temp = temp->next;
    }
    printf("Checkout step with ID %d not found\n", step_id);
}
```

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>

struct CouponCode {
```

```
    int code_id;
    char code[50];
    struct CouponCode *next;
} *top = NULL;

void push(int code_id, char code[]);
void pop();
void display();
void peek();
void search(int code_id);
int isEmpty();

int main() {
    int choice, code_id;
    char code[50];

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

        switch (choice) {
            case 1:
                printf("Enter coupon code ID: ");
                scanf("%d", &code_id);
                printf("Enter coupon code: ");
                scanf(" %[^\n]", code);
                push(code_id, code);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
```

```
            case 4:
                peek();
                break;
            case 5:
                printf("Enter the coupon code ID to search for: ");
                scanf("%d", &code_id);
                search(code_id);
                break;
            case 6:
                printf("Exiting .....\n");
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

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

void push(int code_id, char code[]) {
    struct CouponCode *newCode = (struct CouponCode *)malloc(sizeof(struct
CouponCode));
    if (newCode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newCode->code_id = code_id;
    strcpy(newCode->code, code);
    newCode->next = top;
    top = newCode;
    printf("Coupon code with ID %d added: %s\n", code_id, code);
}

void pop() {
    if (isEmpty()) {
        printf("No coupon codes to remove\n");
        return;
    }
```

```
    struct CouponCode *temp = top;
    top = top->next;
    printf("Coupon code with ID %d removed: %s\n", temp->code_id, temp->code);
    free(temp);
}

void display() {
    if (isEmpty()) {
        printf("No coupon codes available\n");
        return;
    }
    struct CouponCode *temp = top;
    printf("Available Coupon Codes:\n");
    while (temp != NULL) {
        printf("Code ID: %d, Coupon Code: %s\n", temp->code_id, temp->code);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No coupon codes to peek at\n");
        return;
    }
    printf("Current coupon code: ID %d, Code: %s\n", top->code_id, top->code);
}

void search(int code_id) {
    if (isEmpty()) {
        printf("No coupon codes to search\n");
        return;
    }
    struct CouponCode *temp = top;
    while (temp != NULL) {
        if (temp->code_id == code_id) {
            printf("Coupon code found: ID %d, Code: %s\n", temp->code_id, temp->code);
            return;
        }
        temp = temp->next;
    }
    printf("Coupon code with ID %d not found\n", code_id);
```

```
}
```

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>

struct ShippingStatus {
    int status_id;
    char update[100];
    struct ShippingStatus *next;
} *top = NULL;

void push(int status_id, char update[]);
void pop();
void display();
void peek();
void search(int status_id);
int isEmpty();

int main() {
    int choice, status_id;
    char update[100];

    while (1) {
        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 update\n");
        printf("5: Search for a specific update\n");
        printf("6: Exit\n");
```

```
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter shipping status ID: ");
                scanf("%d", &status_id);
                printf("Enter shipping status update: ");
                scanf(" %[^\n]", update);
                push(status_id, update);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
                printf("Enter the shipping status ID to search for: ");
                scanf("%d", &status_id);
                search(status_id);
                break;
            case 6:
                printf("Exiting .....\n");
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}

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

void push(int status_id, char update[]) {
```

```
    struct ShippingStatus *newStatus = (struct ShippingStatus *)malloc(sizeof(struct
ShippingStatus));
    if (newStatus == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newStatus->status_id = status_id;
    strcpy(newStatus->update, update);
    newStatus->next = top;
    top = newStatus;
    printf("Shipping status with ID %d added: %s\n", status_id, update);
}

void pop() {
    if (isEmpty()) {
        printf("No shipping status updates to remove\n");
        return;
    }
    struct ShippingStatus *temp = top;
    top = top->next;
    printf("Shipping status with ID %d removed: %s\n", temp->status_id, temp->update);
    free(temp);
}

void display() {
    if (isEmpty()) {
        printf("No shipping status updates available\n");
        return;
    }
    struct ShippingStatus *temp = top;
    printf("Shipping Status Updates:\n");
    while (temp != NULL) {
        printf("Status ID: %d, Update: %s\n", temp->status_id, temp->update);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No shipping status updates to peek at\n");
        return;
```

```
    }
    printf("Current shipping status: ID %d, Update: %s\n", top->status_id, top->update);
}

void search(int status_id) {
    if (isEmpty()) {
        printf("No shipping status updates to search\n");
        return;
    }
    struct ShippingStatus *temp = top;
    while (temp != NULL) {
        if (temp->status_id == status_id) {
            printf("Shipping status found: ID %d, Update: %s\n", temp->status_id,
temp->update);
            return;
        }
        temp = temp->next;
    }
    printf("Shipping status with ID %d not found\n", status_id);
}
```

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>

struct UserReview {
    int review_id;
    char review[200];
    struct UserReview *next;
} *top = NULL;
```

```
void push(int review_id, char review[]);
void pop();
void display();
void peek();
void search(int review_id);
int isEmpty();

int main() {
    int choice, review_id;
    char review[200];

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

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

```
            scanf("%d", &review_id);
            search(review_id);
            break;
        case 6:
            printf("Exiting .....\n");
            exit(0);
        default:
            printf("Invalid choice\n");
    }
  }

  return 0;
}

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

void push(int review_id, char review[]) {
  struct UserReview *newReview = (struct UserReview *)malloc(sizeof(struct
UserReview));
  if (newReview == NULL) {
    printf("Memory allocation failed\n");
    return;
  }
  newReview->review_id = review_id;
  strcpy(newReview->review, review);
  newReview->next = top;
  top = newReview;
  printf("Review with ID %d added: %s\n", review_id, review);
}

void pop() {
  if (isEmpty()) {
    printf("No reviews to remove\n");
    return;
  }
  struct UserReview *temp = top;
  top = top->next;
  printf("Review with ID %d removed: %s\n", temp->review_id, temp->review);
  free(temp);
```

```
}

void display() {
    if (isEmpty()) {
        printf("No reviews available\n");
        return;
    }
    struct UserReview *temp = top;
    printf("User Reviews:\n");
    while (temp != NULL) {
        printf("Review ID: %d, Review: %s\n", temp->review_id, temp->review);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No reviews to peek at\n");
        return;
    }
    printf("Latest review: ID %d, Review: %s\n", top->review_id, top->review);
}

void search(int review_id) {
    if (isEmpty()) {
        printf("No reviews to search\n");
        return;
    }
    struct UserReview *temp = top;
    while (temp != NULL) {
        if (temp->review_id == review_id) {
            printf("Review found: ID %d, Review: %s\n", temp->review_id, temp->review);
            return;
        }
        temp = temp->next;
    }
    printf("Review with ID %d not found\n", review_id);
}
```

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>

struct PromotionNotification {
    int notification_id;
    char notification_message[200];
    struct PromotionNotification *next;
} *top = NULL;

void push(int notification_id, char message[]);
void pop();
void display();
void peek();
void search(int notification_id);
int isEmpty();

int main() {
    int choice, notification_id;
    char message[200];

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

        switch (choice) {
```

```
            case 1:
                printf("Enter notification ID: ");
                scanf("%d", &notification_id);
                printf("Enter notification message: ");
                scanf(" %[^\n]", message);
                push(notification_id, message);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
                printf("Enter the notification ID to search for: ");
                scanf("%d", &notification_id);
                search(notification_id);
                break;
            case 6:
                printf("Exiting .....\n");
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}

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

void push(int notification_id, char message[]) {
    struct PromotionNotification *newNotification = (struct PromotionNotification
*)malloc(sizeof(struct PromotionNotification));
    if (newNotification == NULL) {
        printf("Memory allocation failed\n");
```

```
        return;
    }
    newNotification->notification_id = notification_id;
    strcpy(newNotification->notification_message, message);
    newNotification->next = top;
    top = newNotification;
    printf("Notification with ID %d added: %s\n", notification_id, message);
}

void pop() {
    if (isEmpty()) {
        printf("No notifications to remove\n");
        return;
    }
    struct PromotionNotification *temp = top;
    top = top->next;
    printf("Notification with ID %d removed: %s\n", temp->notification_id,
temp->notification_message);
    free(temp);
}

void display() {
    if (isEmpty()) {
        printf("No notifications available\n");
        return;
    }
    struct PromotionNotification *temp = top;
    printf("Promotional Notifications:\n");
    while (temp != NULL) {
        printf("Notification ID: %d, Message: %s\n", temp->notification_id,
temp->notification_message);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No notifications to peek at\n");
        return;
    }
    printf("Latest notification: ID %d, Message: %s\n", top->notification_id,
```

```
top->notification_message);
}

void search(int notification_id) {
    if (isEmpty()) {
        printf("No notifications to search\n");
        return;
    }
    struct PromotionNotification *temp = top;
    while (temp != NULL) {
        if (temp->notification_id == notification_id) {
            printf("Notification found: ID %d, Message: %s\n", temp->notification_id,
temp->notification_message);
            return;
        }
        temp = temp->next;
    }
    printf("Notification with ID %d not found\n", notification_id);
}
```

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>

struct Product {
    int product_id;
    char product_name[100];
    struct Product *next;
} *top = NULL;

void push(int product_id, char product_name[]);
```

```
void pop();
void display();
void peek();
void search(int product_id);
int isEmpty();

int main() {
    int choice, product_id;
    char product_name[100];

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

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

```
            search(product_id);
            break;
        case 6:
            printf("Exiting .....\n");
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }

   return 0;
}

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

void push(int product_id, char product_name[]) {
   struct Product *newProduct = (struct Product *)malloc(sizeof(struct Product));
   if (newProduct == NULL) {
      printf("Memory allocation failed\n");
      return;
   }
   newProduct->product_id = product_id;
   strcpy(newProduct->product_name, product_name);
   newProduct->next = top;
   top = newProduct;
   printf("Product with ID %d added to viewing history: %s\n", product_id,
product_name);
}

void pop() {
   if (isEmpty()) {
      printf("No products to remove from viewing history\n");
      return;
   }
   struct Product *temp = top;
   top = top->next;
   printf("Product with ID %d removed from viewing history: %s\n", temp->product_id,
temp->product_name);
   free(temp);
```

```
}

void display() {
    if (isEmpty()) {
        printf("No products in viewing history\n");
        return;
    }
    struct Product *temp = top;
    printf("Viewed Products:\n");
    while (temp != NULL) {
        printf("Product ID: %d, Product Name: %s\n", temp->product_id,
temp->product_name);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No products to peek at\n");
        return;
    }
    printf("Most recent product viewed: ID %d, Product Name: %s\n", top->product_id,
top->product_name);
}

void search(int product_id) {
    if (isEmpty()) {
        printf("No products to search\n");
        return;
    }
    struct Product *temp = top;
    while (temp != NULL) {
        if (temp->product_id == product_id) {
            printf("Product found: ID %d, Product Name: %s\n", temp->product_id,
temp->product_name);
            return;
        }
        temp = temp->next;
    }
    printf("Product with ID %d not found\n", product_id);
}
```

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>

struct CartItem {
    int item_id;
    char item_name[100];
    struct CartItem *next;
} *top = NULL;

void push(int item_id, char item_name[]);
void pop();
void display();
void peek();
void search(int item_id);
int isEmpty();

int main() {
    int choice, item_id;
    char item_name[100];

    while (1) {
        printf("\n--- Cart Item Management ---\n");
        printf("1: Add an item to the cart (push)\n");
        printf("2: Remove the last item (pop)\n");
        printf("3: View all cart items\n");
        printf("4: Peek at the last added item\n");
        printf("5: Search for a specific item in the cart\n");
        printf("6: Exit\n");
```

```
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

    return 0;
}

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

void push(int item_id, char item_name[]) {
```

```
    struct CartItem *newItem = (struct CartItem *)malloc(sizeof(struct CartItem));
    if (newItem == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newItem->item_id = item_id;
    strcpy(newItem->item_name, item_name);
    newItem->next = top;
    top = newItem;
    printf("Item with ID %d added to cart: %s\n", item_id, item_name);
}

void pop() {
    if (isEmpty()) {
        printf("No items to remove from the cart\n");
        return;
    }
    struct CartItem *temp = top;
    top = top->next;
    printf("Item with ID %d removed from cart: %s\n", temp->item_id,
temp->item_name);
    free(temp);
}

void display() {
    if (isEmpty()) {
        printf("No items in the cart\n");
        return;
    }
    struct CartItem *temp = top;
    printf("Cart Items:\n");
    while (temp != NULL) {
        printf("Item ID: %d, Item Name: %s\n", temp->item_id, temp->item_name);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No items to peek at\n");
        return;
```

```
    }
    printf("Last added item: ID %d, Item Name: %s\n", top->item_id, top->item_name);
}

void search(int item_id) {
    if (isEmpty()) {
        printf("No items to search\n");
        return;
    }
    struct CartItem *temp = top;
    while (temp != NULL) {
        if (temp->item_id == item_id) {
            printf("Item found: ID %d, Item Name: %s\n", temp->item_id,
temp->item_name);
            return;
        }
        temp = temp->next;
    }
    printf("Item with ID %d not found\n", item_id);
}
```

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>

struct PaymentRecord {
    int payment_id;
    float amount;
    char payment_method[50];
    struct PaymentRecord *next;
} *top = NULL;
```

```
void push(int payment_id, float amount, char payment_method[]);
void pop();
void display();
void peek();
void search(int payment_id);
int isEmpty();

int main() {
    int choice, payment_id;
    float amount;
    char payment_method[50];

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

        switch (choice) {
            case 1:
                printf("Enter payment ID: ");
                scanf("%d", &payment_id);
                printf("Enter payment amount: ");
                scanf("%f", &amount);
                printf("Enter payment method: ");
                scanf(" %[^\n]", payment_method);
                push(payment_id, amount, payment_method);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
```

```
            peek();
            break;
        case 5:
            printf("Enter the payment ID to search for: ");
            scanf("%d", &payment_id);
            search(payment_id);
            break;
        case 6:
            printf("Exiting .....\n");
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }

   return 0;
}

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

void push(int payment_id, float amount, char payment_method[]) {
   struct PaymentRecord *newRecord = (struct PaymentRecord *)malloc(sizeof(struct
PaymentRecord));
   if (newRecord == NULL) {
      printf("Memory allocation failed\n");
      return;
   }
   newRecord->payment_id = payment_id;
   newRecord->amount = amount;
   strcpy(newRecord->payment_method, payment_method);
   newRecord->next = top;
   top = newRecord;
   printf("Payment record with ID %d added: Amount %.2f, Method: %s\n", payment_id,
amount, payment_method);
}

void pop() {
   if (isEmpty()) {
      printf("No payment records to remove\n");
```

```
        return;
    }
    struct PaymentRecord *temp = top;
    top = top->next;
    printf("Payment record with ID %d removed: Amount %.2f, Method: %s\n",
temp->payment_id, temp->amount, temp->payment_method);
    free(temp);
}

void display() {
    if (isEmpty()) {
        printf("No payment records available\n");
        return;
    }
    struct PaymentRecord *temp = top;
    printf("Payment Records:\n");
    while (temp != NULL) {
        printf("Payment ID: %d, Amount: %.2f, Method: %s\n", temp->payment_id,
temp->amount, temp->payment_method);
        temp = temp->next;
    }
}

void peek() {
    if (isEmpty()) {
        printf("No payment records to peek at\n");
        return;
    }
    printf("Latest payment record: ID %d, Amount %.2f, Method: %s\n",
top->payment_id, top->amount, top->payment_method);
}

void search(int payment_id) {
    if (isEmpty()) {
        printf("No payment records to search\n");
        return;
    }
    struct PaymentRecord *temp = top;
    while (temp != NULL) {
        if (temp->payment_id == payment_id) {
            printf("Payment record found: ID %d, Amount %.2f, Method: %s\n",
```

```
temp->payment_id, temp->amount, temp->payment_method);
            return;
        }
        temp = temp->next;
    }
    printf("Payment record with ID %d not found\n", payment_id);
}
```