Day 18 programs

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>


typedef struct Stack {

   int top;

   int size;

   char **data; // Dynamic array for storing paths

} Stack;


// Function prototypes

void initializeStack(Stack *stack, int size);

void push(Stack *stack, const char *path);

void pop(Stack *stack);

void display(Stack stack);

void peek(Stack stack);

void search(Stack stack, const char *path);
```

```
void freeStack(Stack *stack);

int main() {
    Stack stack;
    int size;

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

    int choice;
    char path[100];

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

        switch (choice) {
            case 1:
                printf("Enter the flight path: ");
                scanf("%s", path);
```

```
            push(&stack, path);
            break;

        case 2:
            pop(&stack);
            break;

        case 3:
            display(stack);
            break;

        case 4:
            peek(stack);
            break;

        case 5:
            printf("Enter the flight path to search: ");
            scanf("%s", path);
            search(stack, path);
            break;

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

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

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

    freeStack(&stack);
    return 0;
}

// Initialize the stack
void initializeStack(Stack *stack, int size) {
    stack->top = -1;
    stack->size = size;
    stack->data = (char **)malloc(size * sizeof(char *));
    for (int i = 0; i < size; i++) {
        stack->data[i] = (char *)malloc(100 * sizeof(char));
    }
}

// Push a new flight path onto the stack
void push(Stack *stack, const char *path) {
    if (stack->top == stack->size - 1) {
        printf("Stack is full. Cannot add more paths.\n");
    } else {
        stack->top++;
        strcpy(stack->data[stack->top], path);
        printf("Path added successfully.\n");
    }
}

// Pop the last flight path from the stack
```

```
void pop(Stack *stack) {
    if (stack->top == -1) {
        printf("Stack is empty. Nothing to undo.\n");
    } else {
        printf("Path '%s' removed from the stack.\n", stack->data[stack->top]);
        stack->top--;
    }
}

// Display all flight paths in the stack
void display(Stack stack) {
    if (stack.top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Flight paths in the stack:\n");
        for (int i = stack.top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack.data[i]);
        }
    }
}

// Peek at the top flight path in the stack
void peek(Stack stack) {
    if (stack.top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Top flight path: %s\n", stack.data[stack.top]);
    }
```

```
}

// Search for a specific flight path in the stack
void search(Stack stack, const char *path) {
    int found = 0;
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.data[i], path) == 0) {
            printf("Flight path '%s' found at position %d.\n", path, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Flight path  not found in the stack.\n");
    }
}

// Free dynamically allocated memory for the stack
void freeStack(Stack *stack) {
    for (int i = 0; i < stack->size; i++) {
        free(stack->data[i]);
    }
    free(stack->data);
}
```

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

1: Push a new satellite deployment

2: Pop the last deployment

3: View the deployment sequence

4: Peek at the latest deployment

5: Search for a specific deployment

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *deployment) {
    if (top == SIZE - 1) {
        printf("Stack is full. Cannot add more deployments.\n");
    } else {
        strcpy(stack[++top], deployment);
        printf("Deployment added successfully.\n");
    }
}

void pop() {
    if (top == -1) {
        printf("Stack is empty. No deployment to remove.\n");
    } else {
        printf("Deployment '%s' removed.\n", stack[top--]);
```

```
    }
}

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

void peek() {
    if (top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Latest Deployment: %s\n", stack[top]);
    }
}

void search(const char *deployment) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], deployment) == 0) {
            printf("Deployment '%s' found at position %d.\n", deployment, i + 1);
            found = 1;
```

```
            break;
        }
    }
    if (!found) {
        printf("Deployment '%s' not found in the stack.\n", deployment);
    }
}

int main() {
    int choice;
    char deployment[100];

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

        switch (choice) {
            case 1:
                printf("Enter the deployment name: ");
                scanf("%s", deployment);
                push(deployment);
```

```
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            printf("Enter the deployment to search: ");
            scanf("%s", deployment);
            search(deployment);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   } while (choice != 6);

   return 0;
}
```

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

1: Add a checklist item (push)

2: Remove the last item (pop)

3: Display the current checklist

4: Peek at the top checklist item

5: Search for a specific checklist item

6: Exit

```
#include <stdio.h>

#include <string.h>


#define SIZE 100


char stack[SIZE][100];

int top = -1;


void push(const char *item) {

   if (top == SIZE - 1) {

      printf("Checklist is full. Cannot add more items.\n");

   } else {

      strcpy(stack[++top], item);

      printf("Checklist item added successfully.\n");

   }

}


void pop() {

   if (top == -1) {

      printf("Checklist is empty. No item to remove.\n");
```

```
    } else {
        printf("Item '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Checklist is empty.\n");
    } else {
        printf("Rocket Launch Checklist:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
    if (top == -1) {
        printf("Checklist is empty.\n");
    } else {
        printf("Latest Item: %s\n", stack[top]);
    }
}

void search(const char *item) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], item) == 0) {
```

```
            printf("Item '%s' found at position %d.\n", item, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Item '%s' not found in the checklist.\n", item);
    }
}

int main() {
    int choice;
    char item[100];

    do {
        printf("\n--- Rocket Launch Checklist ---\n");
        printf("1. Add a checklist item\n");
        printf("2. Remove the last item\n");
        printf("3. Display the current checklist\n");
        printf("4. Peek at the top checklist item\n");
        printf("5. Search for a specific checklist item\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the checklist item: ");
```

```
            scanf("%s", item);
            push(item);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            printf("Enter the checklist item to search: ");
            scanf("%s", item);
            search(item);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  } while (choice != 6);

  return 0;
}
```

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

1: Push new telemetry data

2: Pop the last data entry

3: View the stored telemetry data

4: Peek at the most recent data entry

5: Search for specific telemetry data

6: Exit

```
#include <stdio.h>

#include <string.h>


#define SIZE 100


char stack[SIZE][100];

int top = -1;


void push(const char *data) {

   if (top == SIZE - 1) {

      printf("Stack is full. Cannot add more telemetry data.\n");

   } else {

      strcpy(stack[++top], data);

      printf("Telemetry data added successfully.\n");

   }

}


void pop() {

   if (top == -1) {
```

```
        printf("Stack is empty. No data to remove.\n");
    } else {
        printf("Telemetry data '%s' removed.\n", stack[top--]);
    }
}

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

void peek() {
    if (top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Most Recent Data: %s\n", stack[top]);
    }
}

void search(const char *data) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
```

```
        if (strcmp(stack[i], data) == 0) {
            printf("Telemetry data '%s' found at position %d.\n", data, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Telemetry data '%s' not found.\n", data);
    }
}

int main() {
    int choice;
    char data[100];

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

        switch (choice) {
            case 1:
```

```
            printf("Enter telemetry data: ");
            scanf("%s", data);
            push(data);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            printf("Enter telemetry data to search: ");
            scanf("%s", data);
            search(data);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
      }
   } while (choice != 6);

   return 0;
}
```

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 <string.h>


#define SIZE 100


char stack[SIZE][100];

int top = -1;


void push(const char *task) {

   if (top == SIZE - 1) {

      printf("Task list is full. Cannot add more tasks.\n");

   } else {

      strcpy(stack[++top], task);

      printf("Task added successfully.\n");

   }

}


void pop() {
```

```
    if (top == -1) {
        printf("Task list is empty. No task to remove.\n");
    } else {
        printf("Task '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Task list is empty.\n");
    } else {
        printf("Pending Tasks:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
    if (top == -1) {
        printf("Task list is empty.\n");
    } else {
        printf("Most Recent Task: %s\n", stack[top]);
    }
}

void search(const char *task) {
    int found = 0;
```

```
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], task) == 0) {
            printf("Task '%s' found at position %d.\n", task, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Task '%s' not found in the list.\n", task);
    }
}

int main() {
    int choice;
    char task[100];

    do {
        printf("\n--- Space Mission Task Manager ---\n");
        printf("1. Add a task\n");
        printf("2. Mark the last task as completed\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 task: ");
            scanf("%s", task);
            push(task);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            printf("Enter task to search: ");
            scanf("%s", task);
            search(task);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 6);

return 0;
```

}

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

1: Add a countdown step (push)

2: Remove the last step (pop)

3: Display the current countdown

4: Peek at the next countdown step

5: Search for a specific countdown step

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *countdownStep) {
    if (top == SIZE - 1) {
        printf("Countdown sequence is full. Cannot add more steps.\n");
    } else {
        strcpy(stack[++top], countdownStep);
        printf("Countdown step added successfully.\n");
    }
}
```

```
void pop() {
    if (top == -1) {
        printf("Countdown sequence is empty. No step to remove.\n");
    } else {
        printf("Countdown step '%s' removed.\n", stack[top--]);
    }
}

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

void peek() {
    if (top == -1) {
        printf("Countdown sequence is empty.\n");
    } else {
        printf("Next countdown step: %s\n", stack[top]);
    }
}

void search(const char *countdownStep) {
```

```
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], countdownStep) == 0) {
            printf("Countdown step '%s' found at position %d.\n", countdownStep, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Countdown step '%s' not found.\n", countdownStep);
    }
}

int main() {
    int choice;
    char countdownStep[100];

    do {
        printf("\n--- Launch Countdown Management ---\n");
        printf("1. Add a countdown step\n");
        printf("2. Remove the last step\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 countdown step: ");
            scanf("%s", countdownStep);
            push(countdownStep);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            printf("Enter countdown step to search: ");
            scanf("%s", countdownStep);
            search(countdownStep);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 6);
```

```
    return 0;
}
```

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

1: Add a new log (push)

2: Remove the last log (pop)

3: View all maintenance logs

4: Peek at the latest maintenance log

5: Search for a specific maintenance log

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *log) {
    if (top == SIZE - 1) {
        printf("Maintenance log is full. Cannot add more logs.\n");
    } else {
        strcpy(stack[++top], log);
        printf("Maintenance log added successfully.\n");
    }
}
```

```
void pop() {
    if (top == -1) {
        printf("Maintenance log is empty. No log to remove.\n");
    } else {
        printf("Maintenance log '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Maintenance log is empty.\n");
    } else {
        printf("Maintenance Logs:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
    if (top == -1) {
        printf("Maintenance log is empty.\n");
    } else {
        printf("Most recent maintenance log: %s\n", stack[top]);
    }
}
```

```
void search(const char *log) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], log) == 0) {
            printf("Maintenance log '%s' found at position %d.\n", log, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Maintenance log '%s' not found.\n", log);
    }
}

int main() {
    int choice;
    char log[100];

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

```
    switch (choice) {
        case 1:
            printf("Enter maintenance log: ");
            scanf("%s", log);
            push(log);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            peek();
            break;
        case 5:
            printf("Enter maintenance log to search: ");
            scanf("%s", log);
            search(log);
            break;
        case 6:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 6);
```

```
    return 0;
}
```

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

1: Push a new step

2: Pop the last step

3: Display the procedure steps

4: Peek at the next step in the procedure

5: Search for a specific step

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *step) {
    if (top == SIZE - 1) {
        printf("Docking procedure is full. Cannot add more steps.\n");
    } else {
        strcpy(stack[++top], step);
        printf("Docking step added successfully.\n");
    }
```

```
}

void pop() {
    if (top == -1) {
        printf("Docking procedure is empty. No step to remove.\n");
    } else {
        printf("Docking step '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Docking procedure is empty.\n");
    } else {
        printf("Docking Procedure Steps:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
    if (top == -1) {
        printf("Docking procedure is empty.\n");
    } else {
        printf("Next docking step: %s\n", stack[top]);
    }
}
```

```
void search(const char *step) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], step) == 0) {
            printf("Docking step '%s' found at position %d.\n", step, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Docking step '%s' not found.\n", step);
    }
}

int main() {
    int choice;
    char step[100];

    do {
        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 docking step: ");
        scanf("%s", step);
        push(step);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        peek();
        break;
    case 5:
        printf("Enter docking step to search: ");
        scanf("%s", step);
        search(step);
        break;
    case 6:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
}
```

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

    return 0;
}
```

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

1: Add a command (push)

2: Undo the last command (pop)

3: View the command history

4: Peek at the most recent command

5: Search for a specific command

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *command) {
    if (top == SIZE - 1) {
        printf("Command history is full. Cannot add more commands.\n");
    } else {
        strcpy(stack[++top], command);
        printf("Command added successfully.\n");
```

```
    }
}

void pop() {
    if (top == -1) {
        printf("Command history is empty. No command to remove.\n");
    } else {
        printf("Command '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Command history is empty.\n");
    } else {
        printf("Command History:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
    if (top == -1) {
        printf("Command history is empty.\n");
    } else {
        printf("Most recent command: %s\n", stack[top]);
    }
```

```
}

void search(const char *command) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], command) == 0) {
            printf("Command '%s' found at position %d.\n", command, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Command '%s' not found.\n", command);
    }
}

int main() {
    int choice;
    char command[100];

    do {
        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 command: ");
        scanf("%s", command);
        push(command);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        peek();
        break;
    case 5:
        printf("Enter command to search: ");
        scanf("%s", command);
        search(command);
        break;
    case 6:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
```

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

    return 0;
}
```

10Aerospace Simulation Events: Implement a stack to handle events in an aerospace simulation. Include a switch-case menu with options:

1: Push a new event

2: Pop the last event

3: Display all events

4: Peek at the most recent event

5: Search for a specific event

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *event) {
    if (top == SIZE - 1) {
        printf("Event stack is full. Cannot add more events.\n");
    } else {
        strcpy(stack[++top], event);
```

```
        printf("Event added successfully.\n");
    }
}

void pop() {
    if (top == -1) {
        printf("Event stack is empty. No event to remove.\n");
    } else {
        printf("Event '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Event stack is empty.\n");
    } else {
        printf("Aerospace Simulation Events:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
    if (top == -1) {
        printf("Event stack is empty.\n");
    } else {
        printf("Most recent event: %s\n", stack[top]);
```

```
    }
}

void search(const char *event) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], event) == 0) {
            printf("Event '%s' found at position %d.\n", event, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Event '%s' not found.\n", event);
    }
}

int main() {
    int choice;
    char event[100];

    do {
        printf("\n--- Aerospace Simulation Events ---\n");
        printf("1. Push a new event\n");
        printf("2. Pop the last event\n");
        printf("3. Display all events\n");
        printf("4. Peek at the most recent event\n");
        printf("5. Search for a specific event\n");
```

```
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter event: ");
        scanf("%s", event);
        push(event);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        peek();
        break;
    case 5:
        printf("Enter event to search: ");
        scanf("%s", event);
        search(event);
        break;
    case 6:
        printf("Exiting...\n");
        break;
    default:
```

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

    return 0;
}
```

11.Pilot Training Maneuver Stack: Use a stack to keep track of training maneuvers for pilots. Implement a switch-case menu with options:

1: Add a maneuver (push)

2: Remove the last maneuver (pop)

3: View all maneuvers

4: Peek at the most recent maneuver

5: Search for a specific maneuver

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *maneuver) {
    if (top == SIZE - 1) {
        printf("Maneuver stack is full. Cannot add more maneuvers.\n");
    } else {
```

```
        strcpy(stack[++top], maneuver);
        printf("Maneuver added successfully.\n");
    }
}

void pop() {
    if (top == -1) {
        printf("Maneuver stack is empty. No maneuver to remove.\n");
    } else {
        printf("Maneuver '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Maneuver stack is empty.\n");
    } else {
        printf("Pilot Training Maneuvers:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
    if (top == -1) {
        printf("Maneuver stack is empty.\n");
    } else {
```

```
        printf("Most recent maneuver: %s\n", stack[top]);
    }
}

void search(const char *maneuver) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], maneuver) == 0) {
            printf("Maneuver '%s' found at position %d.\n", maneuver, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Maneuver '%s' not found.\n", maneuver);
    }
}

int main() {
    int choice;
    char maneuver[100];

    do {
        printf("\n--- Pilot Training Maneuver Stack ---\n");
        printf("1. Add a maneuver\n");
        printf("2. Remove the last maneuver\n");
        printf("3. View all maneuvers\n");
        printf("4. Peek at the most recent maneuver\n");
```

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

switch (choice) {
    case 1:
        printf("Enter maneuver: ");
        scanf("%s", maneuver);
        push(maneuver);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        peek();
        break;
    case 5:
        printf("Enter maneuver to search: ");
        scanf("%s", maneuver);
        search(maneuver);
        break;
    case 6:
        printf("Exiting...\n");
        break;
```

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

    return 0;
}
```

12.Satellite Operation Commands: Design a stack to manage operation commands for a satellite. Use a switch-case menu with options:

1: Push a new command

2: Pop the last command

3: View the operation commands

4: Peek at the most recent command

5: Search for a specific command

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *command) {
    if (top == SIZE - 1) {
        printf("Command stack is full. Cannot add more commands.\n");
```

```
    } else {
        strcpy(stack[++top], command);
        printf("Command added successfully.\n");
    }
}

void pop() {
    if (top == -1) {
        printf("Command stack is empty. No command to remove.\n");
    } else {
        printf("Command '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Command stack is empty.\n");
    } else {
        printf("Satellite Operation Commands:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
    if (top == -1) {
        printf("Command stack is empty.\n");
```

```
    } else {
        printf("Most recent command: %s\n", stack[top]);
    }
}

void search(const char *command) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], command) == 0) {
            printf("Command '%s' found at position %d.\n", command, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Command '%s' not found.\n", command);
    }
}

int main() {
    int choice;
    char command[100];

    do {
        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 command: ");
      scanf("%s", command);
      push(command);
      break;
   case 2:
      pop();
      break;
   case 3:
      display();
      break;
   case 4:
      peek();
      break;
   case 5:
      printf("Enter command to search: ");
      scanf("%s", command);
      search(command);
      break;
   case 6:
      printf("Exiting...\n");
```

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

  return 0;
}
```

13.Emergency Procedures for Spacecraft: Create a stack-based system for handling emergency procedures in a spacecraft. Implement a switch-case menu with options:

1: Add a procedure (push)

2: Remove the last procedure (pop)

3: View all procedures

4: Peek at the next procedure

5: Search for a specific procedure

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *procedure) {
  if (top == SIZE - 1) {
```

```
        printf("Procedure stack is full. Cannot add more procedures.\n");
    } else {
        strcpy(stack[++top], procedure);
        printf("Procedure added successfully.\n");
    }
}

void pop() {
    if (top == -1) {
        printf("Procedure stack is empty. No procedure to remove.\n");
    } else {
        printf("Procedure '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Procedure stack is empty.\n");
    } else {
        printf("Emergency Procedures:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
    if (top == -1) {
```

```
        printf("Procedure stack is empty.\n");
    } else {
        printf("Next procedure: %s\n", stack[top]);
    }
}

void search(const char *procedure) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], procedure) == 0) {
            printf("Procedure '%s' found at position %d.\n", procedure, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Procedure '%s' not found.\n", procedure);
    }
}

int main() {
    int choice;
    char procedure[100];

    do {
        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 procedure: ");
      scanf("%s", procedure);
      push(procedure);
      break;
   case 2:
      pop();
      break;
   case 3:
      display();
      break;
   case 4:
      peek();
      break;
   case 5:
      printf("Enter procedure to search: ");
      scanf("%s", procedure);
      search(procedure);
      break;
   case 6:
```

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

  return 0;
}
```

14.Astronaut Activity Log: Implement a stack for logging astronaut activities during a mission. Use a switch-case menu with options:

1: Add a new activity (push)

2: Remove the last activity (pop)

3: Display the activity log

4: Peek at the most recent activity

5: Search for a specific activity

6: Exit

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

#define SIZE 100

char stack[SIZE][100];
int top = -1;

void push(const char *activity) {
```

```
    if (top == SIZE - 1) {
        printf("Activity stack is full. Cannot add more activities.\n");
    } else {
        strcpy(stack[++top], activity);
        printf("Activity added successfully.\n");
    }
}

void pop() {
    if (top == -1) {
        printf("Activity stack is empty. No activity to remove.\n");
    } else {
        printf("Activity '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Activity stack is empty.\n");
    } else {
        printf("Astronaut Activity Log:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}

void peek() {
```

```
    if (top == -1) {
        printf("Activity stack is empty.\n");
    } else {
        printf("Most recent activity: %s\n", stack[top]);
    }
}

void search(const char *activity) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], activity) == 0) {
            printf("Activity '%s' found at position %d.\n", activity, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Activity '%s' not found.\n", activity);
    }
}

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

    do {
        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 activity: ");
        scanf("%s", activity);
        push(activity);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        peek();
        break;
    case 5:
        printf("Enter activity to search: ");
        scanf("%s", activity);
        search(activity);
        break;
```

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

    return 0;
}
```

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 <string.h>

#define SIZE 100

char stack[SIZE][100];
int top = -1;
```

```
void push(const char *entry) {
    if (top == SIZE - 1) {
        printf("Fuel usage stack is full. Cannot add more entries.\n");
    } else {
        strcpy(stack[++top], entry);
        printf("Fuel usage entry added successfully.\n");
    }
}

void pop() {
    if (top == -1) {
        printf("Fuel usage stack is empty. No entry to remove.\n");
    } else {
        printf("Fuel usage entry '%s' removed.\n", stack[top--]);
    }
}

void display() {
    if (top == -1) {
        printf("Fuel usage stack is empty.\n");
    } else {
        printf("Fuel Usage Entries:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d. %s\n", i + 1, stack[i]);
        }
    }
}
```

```
void peek() {
    if (top == -1) {
        printf("Fuel usage stack is empty.\n");
    } else {
        printf("Most recent fuel usage entry: %s\n", stack[top]);
    }
}

void search(const char *entry) {
    int found = 0;
    for (int i = 0; i <= top; i++) {
        if (strcmp(stack[i], entry) == 0) {
            printf("Fuel usage entry '%s' found at position %d.\n", entry, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Fuel usage entry '%s' not found.\n", entry);
    }
}

int main() {
    int choice;
    char entry[100];

    do {
        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 fuel usage entry: ");
        scanf("%s", entry);
        push(entry);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        peek();
        break;
    case 5:
        printf("Enter fuel usage entry to search: ");
        scanf("%s", entry);
        search(entry);
```

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

   return 0;
}
```

Stack using linked list programs

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>

struct Node {
   int data;
   struct Node *next;
```

```
} *top = NULL;

void push(int x) {
   struct Node *t = (struct Node *)malloc(sizeof(struct Node));
   if (!t) {
      printf("Memory Error\n");
      return;
   }
   t->data = x;
   t->next = top;
   top = t;
}

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

void display() {
   struct Node *p = top;
   if (p == NULL) {
```

```
        printf("No orders\n");
        return;
    }
    while (p) {
        printf("%d ", p->data);
        p = p->next;
    }
    printf("\n");
}

void peek() {
    if (top == NULL) {
        printf("Stack is Empty\n");
    } else {
        printf("Next order to be processed: %d\n", top->data);
    }
}

int search(int x) {
    struct Node *p = top;
    while (p) {
        if (p->data == x) {
            return 1;
        }
        p = p->next;
    }
    return 0;
}
```

```
int main() {
    int choice, order;
    while (1) {
        printf("\n1. Add Order\n2. Process Order\n3. Display Orders\n4. Peek Next Order\n5. Search Order\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter order number: ");
                scanf("%d", &order);
                push(order);
                break;
            case 2:
                order = pop();
                if (order != -1) {
                    printf("Processed order %d\n", order);
                }
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
```

```
            printf("Enter order number to search: ");
            scanf("%d", &order);
            if (search(order)) {
                printf("Order found\n");
            } else {
                printf("Order not found\n");
            }
            break;
        case 6:
            exit(0);
    }
  }
}
```

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 Node {
    int ticketID;
    struct Node *next;
};

struct Node *top = NULL;

void push(int ticketID) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    newNode->ticketID = ticketID;
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return -1;
    }
    struct Node *temp = top;
    int ticketID = temp->ticketID;
    top = top->next;
    free(temp);
    return ticketID;
```

```
}

void display() {
    if (!top) {
        printf("No tickets in the system\n");
        return;
    }
    struct Node *temp = top;
    while (temp) {
        printf("Ticket ID: %d\n", temp->ticketID);
        temp = temp->next;
    }
}

int main() {
    int choice, ticketID;

    do {
        printf("\n1. Add Ticket (Push)\n");
        printf("2. Resolve Ticket (Pop)\n");
        printf("3. View Pending Tickets\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

```
            scanf("%d", &ticketID);
            push(ticketID);
            break;
        case 2:
            ticketID = pop();
            if (ticketID != -1)
                printf("Resolved Ticket ID: %d\n", ticketID);
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

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

1: Add a new return request (push)

2: Process the last return (pop)

3: Display all return requests

4: Peek at the next return to process

5: Search for a specific return request

6: Exit

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

struct Node {
    int returnID;
    struct Node *next;
};

struct Node *top = NULL;

void push(int returnID) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    newNode->returnID = returnID;
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return -1;
```

```
    }
    struct Node *temp = top;
    int returnID = temp->returnID;
    top = top->next;
    free(temp);
    return returnID;
}

void display() {
    if (!top) {
        printf("No return requests\n");
        return;
    }
    struct Node *temp = top;
    while (temp) {
        printf("Return ID: %d\n", temp->returnID);
        temp = temp->next;
    }
}

int main() {
    int choice, returnID;

    do {
        printf("\n1. Add Return Request (Push)\n");
        printf("2. Process Return (Pop)\n");
        printf("3. View Return Requests\n");
        printf("4. Exit\n");
```

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

        switch (choice) {
        case 1:
            printf("Enter Return ID: ");
            scanf("%d", &returnID);
            push(returnID);
            break;
        case 2:
            returnID = pop();
            if (returnID != -1)
                printf("Processed Return ID: %d\n", returnID);
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

4.Inventory Restock System: Implement a stack to manage inventory restocking using a linked list. Use a switch-case menu with options:

1: Add a restock entry (push)

2: Process the last restock (pop)

3: View all restock entries

4: Peek at the latest restock entry

5: Search for a specific restock entry

6: Exit

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int restockID;

   struct Node *next;

};


struct Node *top = NULL;


void push(int restockID) {

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

   if (!newNode) {

      printf("Stack Overflow\n");

      return;

   }

   newNode->restockID = restockID;

   newNode->next = top;
```

```
    top = newNode;
}

int pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return -1;
    }
    struct Node *temp = top;
    int restockID = temp->restockID;
    top = top->next;
    free(temp);
    return restockID;
}

void display() {
    if (!top) {
        printf("No restock entries available\n");
        return;
    }
    struct Node *temp = top;
    while (temp) {
        printf("Restock ID: %d\n", temp->restockID);
        temp = temp->next;
    }
}

int main() {
```

```
int choice, restockID;

do {
    printf("\n1. Add Restock Entry (Push)\n");
    printf("2. Process Restock (Pop)\n");
    printf("3. View All Restock Entries\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
    case 1:
        printf("Enter Restock ID: ");
        scanf("%d", &restockID);
        push(restockID);
        break;
    case 2:
        restockID = pop();
        if (restockID != -1)
            printf("Processed Restock ID: %d\n", restockID);
        break;
    case 3:
        display();
        break;
    case 4:
        printf("Exiting...\n");
        break;
    default:
```

```
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

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

1: Add a new deal (push)

2: Remove the last deal (pop)

3: View all active deals

4: Peek at the latest deal

5: Search for a specific deal

6: Exit

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

struct Node {
    int dealID;
    struct Node *next;
};

struct Node *top = NULL;

void push(int dealID) {
```

```
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    newNode->dealID = dealID;
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return -1;
    }
    struct Node *temp = top;
    int dealID = temp->dealID;
    top = top->next;
    free(temp);
    return dealID;
}

void display() {
    if (!top) {
        printf("No active deals\n");
        return;
    }
    struct Node *temp = top;
```

```
    while (temp) {
        printf("Deal ID: %d\n", temp->dealID);
        temp = temp->next;
    }
}

int main() {
    int choice, dealID;

    do {
        printf("\n1. Add Deal (Push)\n");
        printf("2. Remove Deal (Pop)\n");
        printf("3. View All Deals\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter Deal ID: ");
            scanf("%d", &dealID);
            push(dealID);
            break;
        case 2:
            dealID = pop();
            if (dealID != -1)
                printf("Removed Deal ID: %d\n", dealID);
            break;
```

```
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

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>

struct Node {
    int sessionID;
```

```
    struct Node *next;
};

struct Node *top = NULL;

void push(int sessionID) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    newNode->sessionID = sessionID;
    newNode->next = top;
    top = newNode;
}

int pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return -1;
    }
    struct Node *temp = top;
    int sessionID = temp->sessionID;
    top = top->next;
    free(temp);
    return sessionID;
}
```

```
void display() {
    if (!top) {
        printf("No sessions available\n");
        return;
    }
    struct Node *temp = top;
    while (temp) {
        printf("Session ID: %d\n", temp->sessionID);
        temp = temp->next;
    }
}

int main() {
    int choice, sessionID;

    do {
        printf("\n1. Add Session (Push)\n");
        printf("2. End Session (Pop)\n");
        printf("3. View All Sessions\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter Session ID: ");
            scanf("%d", &sessionID);
            push(sessionID);
```

```
            break;
        case 2:
            sessionID = pop();
            if (sessionID != -1)
                printf("Ended Session ID: %d\n", sessionID);
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

7.Wishlist Management: Develop a stack to manage user wishlists using a linked list. Use a switch-case menu with options:

1: Add a product to wishlist (push)

2: Remove the last added product (pop)

3: View all wishlist items

4: Peek at the most recent wishlist item

5: Search for a specific product in wishlist

6: Exit

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

struct Node {
    int productID;
    struct Node *next;
};

struct Node *top = NULL;

void push(int productID) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    newNode->productID = productID;
    newNode->next = top;
    top = newNode;
}

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

```
    int productID = temp->productID;
    top = top->next;
    free(temp);
    return productID;
}

void display() {
    if (!top) {
        printf("No wishlist items available\n");
        return;
    }
    struct Node *temp = top;
    while (temp) {
        printf("Product ID: %d\n", temp->productID);
        temp = temp->next;
    }
}

int main() {
    int choice, productID;

    do {
        printf("\n1. Add Product to Wishlist (Push)\n");
        printf("2. Remove Last Product (Pop)\n");
        printf("3. View Wishlist\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
        case 1:
            printf("Enter Product ID: ");
            scanf("%d", &productID);
            push(productID);
            break;
        case 2:
            productID = pop();
            if (productID != -1)
                printf("Removed Product ID: %d\n", productID);
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

8.Checkout Process Steps: Implement a stack to manage steps in the checkout process using a linked list. Include a switch-case menu with options:

1: Add a checkout step (push)

2: Remove the last step (pop)

3: Display all checkout steps

4: Peek at the current step

5: Search for a specific step

6: Exit

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

struct Node {
    char step[50];
    struct Node *next;
};

struct Node *top = NULL;

void push(char step[]) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    strcpy(newNode->step, step);
    newNode->next = top;
    top = newNode;
}
```

```
void pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return;
    }
    struct Node *temp = top;
    printf("Removed Step: %s\n", temp->step);
    top = top->next;
    free(temp);
}

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

int main() {
    int choice;
    char step[50];
```

```
do {
    printf("\n1. Add Checkout Step (Push)\n");
    printf("2. Remove Last Step (Pop)\n");
    printf("3. View Checkout Steps\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
    case 1:
        printf("Enter Checkout Step: ");
        scanf(" %[^\n]s", step);
        push(step);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice!\n");
    }
} while (choice != 4);
```

```
    return 0;
}
```

9.Coupon Code Management: Create a stack for managing coupon codes using a linked list. Use a switch-case menu with options:

1: Add a new coupon code (push)

2: Remove the last coupon code (pop)

3: View all available coupon codes

4: Peek at the latest coupon code

5: Search for a specific coupon code

6: Exit

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

struct Node {
    char coupon[30];
    struct Node *next;
};

struct Node *top = NULL;

void push(char coupon[]) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
```

```
    }
    strcpy(newNode->coupon, coupon);
    newNode->next = top;
    top = newNode;
}

void pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return;
    }
    struct Node *temp = top;
    printf("Removed Coupon Code: %s\n", temp->coupon);
    top = top->next;
    free(temp);
}

void display() {
    if (!top) {
        printf("No coupon codes available\n");
        return;
    }
    struct Node *temp = top;
    while (temp) {
        printf("Coupon Code: %s\n", temp->coupon);
        temp = temp->next;
    }
}
```

```
int main() {
    int choice;
    char coupon[30];

    do {
        printf("\n1. Add Coupon Code (Push)\n");
        printf("2. Remove Last Coupon Code (Pop)\n");
        printf("3. View All Coupon Codes\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter Coupon Code: ");
            scanf(" %[^\n]s", coupon);
            push(coupon);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
```

```
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

10.Shipping Status Tracker: Develop a stack to track shipping status updates using a linked list. Implement a switch-case menu with options:

1: Add a shipping status update (push)

2: Remove the last update (pop)

3: View all shipping status updates

4: Peek at the latest update

5: Search for a specific update

6: Exit

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

struct Node {
    char status[50];
    struct Node *next;
};

struct Node *top = NULL;
```

```
void push(char status[]) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    strcpy(newNode->status, status);
    newNode->next = top;
    top = newNode;
}

void pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return;
    }
    struct Node *temp = top;
    printf("Removed Shipping Status: %s\n", temp->status);
    top = top->next;
    free(temp);
}

void display() {
    if (!top) {
        printf("No shipping status updates available\n");
        return;
    }
    struct Node *temp = top;
```

```
    while (temp) {
        printf("Status: %s\n", temp->status);
        temp = temp->next;
    }
}

int main() {
    int choice;
    char status[50];

    do {
        printf("\n1. Add Shipping Status Update (Push)\n");
        printf("2. Remove Last Status Update (Pop)\n");
        printf("3. View All Status Updates\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter Shipping Status: ");
            scanf(" %[^\n]s", status);
            push(status);
            break;
        case 2:
            pop();
            break;
        case 3:
```

```
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

11.User Review Management: Use a stack to manage user reviews for products using a linked list. Include a switch-case menu with options:

1: Add a new review (push)

2: Remove the last review (pop)

3: Display all reviews

4: Peek at the latest review

5: Search for a specific review

6: Exit

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

struct Node {
    char review[100];
```

```
    struct Node *next;
};

struct Node *top = NULL;

void push(char review[]) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    strcpy(newNode->review, review);
    newNode->next = top;
    top = newNode;
}

void pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return;
    }
    struct Node *temp = top;
    printf("Removed Review: %s\n", temp->review);
    top = top->next;
    free(temp);
}

void display() {
```

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

int main() {
    int choice;
    char review[100];

    do {
        printf("\n1. Add User Review (Push)\n");
        printf("2. Remove Last Review (Pop)\n");
        printf("3. View All Reviews\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter User Review: ");
            scanf(" %[^\n]s", review);
            push(review);
```

```
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

12.Promotion Notification System: Create a stack for managing promotional notifications using a linked list. Use a switch-case menu with options:

1: Add a new notification (push)

2: Remove the last notification (pop)

3: View all notifications

4: Peek at the latest notification

5: Search for a specific notification

6: Exit

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

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

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

struct Node *top = NULL;

void push(char notification[]) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    strcpy(newNode->notification, notification);
    newNode->next = top;
    top = newNode;
}

void pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return;
    }
    struct Node *temp = top;
    printf("Removed Notification: %s\n", temp->notification);
```

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

void display() {
    if (!top) {
        printf("No notifications available\n");
        return;
    }
    struct Node *temp = top;
    while (temp) {
        printf("Notification: %s\n", temp->notification);
        temp = temp->next;
    }
}

int main() {
    int choice;
    char notification[100];

    do {
        printf("\n1. Add Promotion Notification (Push)\n");
        printf("2. Remove Last Notification (Pop)\n");
        printf("3. View All Notifications\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
        case 1:
            printf("Enter Promotion Notification: ");
            scanf(" %[^\n]s", notification);
            push(notification);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

13.Product Viewing History: Implement a stack to track the viewing history of products using a linked list. Include a switch-case menu with options:

1: Add a product to viewing history (push)

2: Remove the last viewed product (pop)

3: Display all viewed products

4: Peek at the most recent product viewed

5: Search for a specific product

6: Exit

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

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

struct Node *top = NULL;

void push(char product[]) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Stack Overflow\n");
        return;
    }
    strcpy(newNode->product, product);
    newNode->next = top;
    top = newNode;
}

void pop() {
    if (!top) {
```

```
        printf("Stack Underflow\n");
        return;
    }
    struct Node *temp = top;
    printf("Removed Product: %s\n", temp->product);
    top = top->next;
    free(temp);
}

void display() {
    if (!top) {
        printf("No products viewed yet\n");
        return;
    }
    struct Node *temp = top;
    while (temp) {
        printf("Viewed Product: %s\n", temp->product);
        temp = temp->next;
    }
}

int main() {
    int choice;
    char product[100];

    do {
        printf("\n1. Add Product to Viewing History (Push)\n");
        printf("2. Remove Last Viewed Product (Pop)\n");
```

```
        printf("3. View All Viewed Products\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter Product Name: ");
            scanf(" %[^\n]s", product);
            push(product);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```

14.Cart Item Management: Develop a stack to manage items in a shopping cart using a linked list. Use a switch-case menu with options:

1: Add an item to the cart (push)

2: Remove the last item (pop)

3: View all cart items

4: Peek at the last added item

5: Search for a specific item in the cart

6: Exit

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Node {

   char item[50];

   struct Node *next;

};


struct Node *top = NULL;


void push(char item[]) {

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

   if (!newNode) {

      printf("Stack Overflow\n");

      return;

   }

   strcpy(newNode->item, item);

   newNode->next = top;
```

```
    top = newNode;
}

void pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return;
    }
    struct Node *temp = top;
    printf("Removed Item: %s\n", temp->item);
    top = top->next;
    free(temp);
}

void display() {
    if (!top) {
        printf("No items in the cart\n");
        return;
    }
    struct Node *temp = top;
    while (temp) {
        printf("Cart Item: %s\n", temp->item);
        temp = temp->next;
    }
}

int main() {
    int choice;
```

```
char item[50];

do {
    printf("\n1. Add Item to Cart (Push)\n");
    printf("2. Remove Last Item (Pop)\n");
    printf("3. View All Cart Items\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
    case 1:
        printf("Enter Item Name: ");
        scanf(" %[^\n]s", item);
        push(item);
        break;
    case 2:
        pop();
        break;
    case 3:
        display();
        break;
    case 4:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice!\n");
    }
```

```
    } while (choice != 4);

    return 0;
}
```

15.Payment History: Implement a stack to record payment history using a linked list. Include a switch-case menu with options:

1: Add a new payment record (push)

2: Remove the last payment record (pop)

3: View all payment records

4: Peek at the latest payment record

5: Search for a specific payment record

6: Exit

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

struct Node {
    char payment[50];
    struct Node *next;
};

struct Node *top = NULL;

void push(char payment[]) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (!newNode) {
```

```
        printf("Stack Overflow\n");
        return;
    }
    strcpy(newNode->payment, payment);
    newNode->next = top;
    top = newNode;
}

void pop() {
    if (!top) {
        printf("Stack Underflow\n");
        return;
    }
    struct Node *temp = top;
    printf("Removed Payment Record: %s\n", temp->payment);
    top = top->next;
    free(temp);
}

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

```
    }
}

int main() {
    int choice;
    char payment[50];

    do {
        printf("\n1. Add Payment Record (Push)\n");
        printf("2. Remove Last Payment Record (Pop)\n");
        printf("3. View All Payment Records\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter Payment Record: ");
            scanf(" %[^\n]s", payment);
            push(payment);
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
```

```
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
        }
    } while (choice != 4);

    return 0;
}
```