----------programs----------

1.flight path logging system

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

#define MAX 100

struct FlightPathStack {
   int top;
   char paths[MAX][100];
};

void initializeStack(struct FlightPathStack *stack);
void push(struct FlightPathStack *stack, const char *path);
void pop(struct FlightPathStack *stack);
void display(struct FlightPathStack stack);
void peek(struct FlightPathStack stack);
int search(struct FlightPathStack stack, const char *path);
int isEmpty(struct FlightPathStack stack);
int isFull(struct FlightPathStack stack);

int main() {
   struct FlightPathStack stack;
   initializeStack(&stack);
   int choice;
   char path[100];
   int numPaths;
```

```
printf("Enter the number of flight paths you want to add initially: ");
scanf("%d", &numPaths);
getchar();
for (int i = 0; i < numPaths; i++) {
   printf("Enter flight path #%d: ", i + 1);
   fgets(path, sizeof(path), stdin);
   path[strcspn(path, "\n")] = '\0';
   push(&stack, path);
}
do {
   printf("\nFlight Path Logging System\n");
   printf("1: Add a new path\n");
   printf("2: Undo the last path\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);
   getchar();
   switch (choice) {
      case 1:
         printf("Enter the flight path: ");
         fgets(path, sizeof(path), stdin);
         path[strcspn(path, "\n")] = '\0';
         push(&stack, path);
         break;
      case 2:
         pop(&stack);
```

```
            break;
        case 3:
            display(stack);
            break;
        case 4:
            peek(stack);
            break;
        case 5:
            printf("Enter the path to search: ");
            fgets(path, sizeof(path), stdin);
            path[strcspn(path, "\n")] = '\0';
            if (search(stack, path)) {
                printf("Path found in the stack.\n");
            } else {
                printf("Path not found in the stack.\n");
            }
            break;
        case 6:
            printf("Exiting the system.\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
  } while (choice != 6);

  return 0;
}

void initializeStack(struct FlightPathStack *stack) {
```

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

void push(struct FlightPathStack *stack, const char *path) {
    if (isFull(*stack)) {
        printf("Stack is full, cannot add more paths.\n");
    } else {
        stack->top++;
        strcpy(stack->paths[stack->top], path);
        printf("Path added: %s\n", path);
    }
}

void pop(struct FlightPathStack *stack) {
    if (isEmpty(*stack)) {
        printf("Stack is empty, no path to undo.\n");
    } else {
        printf("Undoing path: %s\n", stack->paths[stack->top]);
        stack->top--;
    }
}

void display(struct FlightPathStack stack) {
    if (isEmpty(stack)) {
        printf("No flight paths in the stack.\n");
    } else {
        printf("Current Flight Path Stack:\n");
        for (int i = stack.top; i >= 0; i--) {
            printf("%d: %s\n", stack.top - i + 1, stack.paths[i]);
```

```
        }
    }
}

void peek(struct FlightPathStack stack) {
    if (isEmpty(stack)) {
        printf("No paths available to peek at.\n");
    } else {
        printf("Top path: %s\n", stack.paths[stack.top]);
    }
}

int search(struct FlightPathStack stack, const char *path) {
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.paths[i], path) == 0) {
            return 1;
        }
    }
    return 0;
}

int isEmpty(struct FlightPathStack stack) {
    return stack.top == -1;
}

int isFull(struct FlightPathStack stack) {
    return stack.top == MAX - 1;
}
```

2. satellite deployment sequence

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

#define MAX 50

struct SatelliteDeployment {
    int top;
    char deployments[MAX][100];
};

void init(struct SatelliteDeployment *stack) {
    stack->top = -1;
}

int isFull(struct SatelliteDeployment *stack) {
    return stack->top == MAX - 1;
}

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

void push(struct SatelliteDeployment *stack, const char *deployment) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
    } else {
        stack->top++;
```

```
        strcpy(stack->deployments[stack->top], deployment);
        printf("Deployment added!\n");
    }
}

void pop(struct SatelliteDeployment *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped deployment: %s\n", stack->deployments[stack->top]);
        stack->top--;
    }
}

void display(struct SatelliteDeployment stack) {
    if (isEmpty(&stack)) {
        printf("No deployments!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("%s\n", stack.deployments[i]);
        }
    }
}

void peek(struct SatelliteDeployment stack) {
    if (isEmpty(&stack)) {
        printf("No deployments to peek at!\n");
    } else {
        printf("Latest deployment: %s\n", stack.deployments[stack.top]);
```

```
    }
}

int search(struct SatelliteDeployment stack, const char *deployment) {
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.deployments[i], deployment) == 0) {
            return 1;
        }
    }
    return 0;
}

int main() {
    struct SatelliteDeployment stack;
    init(&stack);
    int choice;
    char deployment[100];

    while (1) {
        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);
        getchar();
```

```
switch (choice) {
    case 1:
        printf("Enter satellite deployment: ");
        fgets(deployment, sizeof(deployment), stdin);
        deployment[strcspn(deployment, "\n")] = '\0';
        push(&stack, deployment);
        break;
    case 2:
        pop(&stack);
        break;
    case 3:
        display(stack);
        break;
    case 4:
        peek(stack);
        break;
    case 5:
        printf("Enter deployment to search: ");
        fgets(deployment, sizeof(deployment), stdin);
        deployment[strcspn(deployment, "\n")] = '\0';
        if (search(stack, deployment)) {
            printf("Deployment found!\n");
        } else {
            printf("Deployment not found!\n");
        }
        break;
    case 6:
        exit(0);
    default:
```

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

    return 0;
}
```

3. rocket launch checklist

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

#define MAX 50

struct SatelliteDeployment {
    int top;
    char deployments[MAX][100];
};

void init(struct SatelliteDeployment *stack) {
    stack->top = -1;
}

int isFull(struct SatelliteDeployment *stack) {
    return stack->top == MAX - 1;
}

int isEmpty(struct SatelliteDeployment *stack) {
    return stack->top == -1;
```

```
}

void push(struct SatelliteDeployment *stack, const char *deployment) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
    } else {
        stack->top++;
        strcpy(stack->deployments[stack->top], deployment);
        printf("Deployment added!\n");
    }
}

void pop(struct SatelliteDeployment *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped deployment: %s\n", stack->deployments[stack->top]);
        stack->top--;
    }
}

void display(struct SatelliteDeployment stack) {
    if (isEmpty(&stack)) {
        printf("No deployments!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("%s\n", stack.deployments[i]);
        }
    }
```

```
}

void peek(struct SatelliteDeployment stack) {
    if (isEmpty(&stack)) {
        printf("No deployments to peek at!\n");
    } else {
        printf("Latest deployment: %s\n", stack.deployments[stack.top]);
    }
}

int search(struct SatelliteDeployment stack, const char *deployment) {
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.deployments[i], deployment) == 0) {
            return 1;
        }
    }
    return 0;
}

int main() {
    struct SatelliteDeployment stack;
    init(&stack);
    int choice;
    char deployment[100];

    while (1) {
        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);
getchar();

switch (choice) {
   case 1:
      printf("Enter satellite deployment: ");
      fgets(deployment, sizeof(deployment), stdin);
      deployment[strcspn(deployment, "\n")] = '\0';
      push(&stack, deployment);
      break;
   case 2:
      pop(&stack);
      break;
   case 3:
      display(stack);
      break;
   case 4:
      peek(stack);
      break;
   case 5:
      printf("Enter deployment to search: ");
      fgets(deployment, sizeof(deployment), stdin);
      deployment[strcspn(deployment, "\n")] = '\0';
      if (search(stack, deployment)) {
         printf("Deployment found!\n");
```

```
            } else {
                printf("Deployment not found!\n");
            }
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
      }
   }

   return 0;
}
```

4.telementry data storage

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

#define MAX 50

struct SatelliteDeployment {
   int top;
   char deployments[MAX][100];
};

void init(struct SatelliteDeployment *stack) {
   stack->top = -1;
}
```

```
int isFull(struct SatelliteDeployment *stack) {
    return stack->top == MAX - 1;
}

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

void push(struct SatelliteDeployment *stack, const char *deployment) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
    } else {
        stack->top++;
        strcpy(stack->deployments[stack->top], deployment);
        printf("Deployment added!\n");
    }
}

void pop(struct SatelliteDeployment *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped deployment: %s\n", stack->deployments[stack->top]);
        stack->top--;
    }
}

void display(struct SatelliteDeployment stack) {
```

```
    if (isEmpty(&stack)) {
        printf("No deployments!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("%s\n", stack.deployments[i]);
        }
    }
}

void peek(struct SatelliteDeployment stack) {
    if (isEmpty(&stack)) {
        printf("No deployments to peek at!\n");
    } else {
        printf("Latest deployment: %s\n", stack.deployments[stack.top]);
    }
}

int search(struct SatelliteDeployment stack, const char *deployment) {
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.deployments[i], deployment) == 0) {
            return 1;
        }
    }
    return 0;
}

int main() {
    struct SatelliteDeployment stack;
    init(&stack);
```

```
int choice;
char deployment[100];

while (1) {
   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);
   getchar();

   switch (choice) {
      case 1:
         printf("Enter satellite deployment: ");
         fgets(deployment, sizeof(deployment), stdin);
         deployment[strcspn(deployment, "\n")] = '\0';
         push(&stack, deployment);
         break;
      case 2:
         pop(&stack);
         break;
      case 3:
         display(stack);
         break;
      case 4:
         peek(stack);
```

```
            break;
        case 5:
            printf("Enter deployment to search: ");
            fgets(deployment, sizeof(deployment), stdin);
            deployment[strcspn(deployment, "\n")] = '\0';
            if (search(stack, deployment)) {
                printf("Deployment found!\n");
            } else {
                printf("Deployment not found!\n");
            }
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
      }
   }

   return 0;
}
```

5.space mission task manager

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

#define MAX 50

struct SatelliteDeployment {
```

```
    int top;
    char deployments[MAX][100];
};

void init(struct SatelliteDeployment *stack) {
    stack->top = -1;
}

int isFull(struct SatelliteDeployment *stack) {
    return stack->top == MAX - 1;
}

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

void push(struct SatelliteDeployment *stack, const char *deployment) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
    } else {
        stack->top++;
        strcpy(stack->deployments[stack->top], deployment);
        printf("Deployment added!\n");
    }
}

void pop(struct SatelliteDeployment *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
```

```
    } else {
        printf("Popped deployment: %s\n", stack->deployments[stack->top]);
        stack->top--;
    }
}

void display(struct SatelliteDeployment stack) {
    if (isEmpty(&stack)) {
        printf("No deployments!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("%s\n", stack.deployments[i]);
        }
    }
}

void peek(struct SatelliteDeployment stack) {
    if (isEmpty(&stack)) {
        printf("No deployments to peek at!\n");
    } else {
        printf("Latest deployment: %s\n", stack.deployments[stack.top]);
    }
}

int search(struct SatelliteDeployment stack, const char *deployment) {
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.deployments[i], deployment) == 0) {
            return 1;
        }
```

```
    }
    return 0;
}

int main() {
    struct SatelliteDeployment stack;
    init(&stack);
    int choice;
    char deployment[100];

    while (1) {
        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);
        getchar();

        switch (choice) {
            case 1:
                printf("Enter satellite deployment: ");
                fgets(deployment, sizeof(deployment), stdin);
                deployment[strcspn(deployment, "\n")] = '\0';
                push(&stack, deployment);
                break;
            case 2:
```

```
            pop(&stack);
            break;
        case 3:
            display(stack);
            break;
        case 4:
            peek(stack);
            break;
        case 5:
            printf("Enter deployment to search: ");
            fgets(deployment, sizeof(deployment), stdin);
            deployment[strcspn(deployment, "\n")] = '\0';
            if (search(stack, deployment)) {
                printf("Deployment found!\n");
            } else {
                printf("Deployment not found!\n");
            }
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
      }
   }

   return 0;
}
```

6.rocket countdown management

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

#define MAX 50

struct Countdown {
    int top;
    int steps[MAX];
};

void init(struct Countdown *stack) {
    stack->top = -1;
}

int isFull(struct Countdown *stack) {
    return stack->top == MAX - 1;
}

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

void push(struct Countdown *stack, int step) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
    } else {
        stack->top++;
        stack->steps[stack->top] = step;
        printf("Countdown step added!\n");
```

```
    }
}

void pop(struct Countdown *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped countdown step: %d\n", stack->steps[stack->top]);
        stack->top--;
    }
}

void display(struct Countdown stack) {
    if (isEmpty(&stack)) {
        printf("No countdown steps!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("Step: %d\n", stack.steps[i]);
        }
    }
}

void peek(struct Countdown stack) {
    if (isEmpty(&stack)) {
        printf("No steps to peek at!\n");
    } else {
        printf("Latest countdown step: %d\n", stack.steps[stack.top]);
    }
}
```

```
int search(struct Countdown stack, int step) {
    for (int i = 0; i <= stack.top; i++) {
        if (stack.steps[i] == step) {
            return 1;
        }
    }
    return 0;
}

int main() {
    struct Countdown stack;
    init(&stack);
    int choice, step;

    while (1) {
        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("%d", &step);
```

```
            push(&stack, step);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(stack);
            break;
        case 4:
            peek(stack);
            break;
        case 5:
            printf("Enter step to search: ");
            scanf("%d", &step);
            if (search(stack, step)) {
                printf("Step found!\n");
            } else {
                printf("Step not found!\n");
            }
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
      }
   }

   return 0;
}
```

7.aircraft manaitenace logs

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

#define MAX 50

struct Countdown {
    int top;
    int steps[MAX];
};

void init(struct Countdown *stack) {
    stack->top = -1;
}

int isFull(struct Countdown *stack) {
    return stack->top == MAX - 1;
}

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

void push(struct Countdown *stack, int step) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
    } else {
        stack->top++;
```

```
        stack->steps[stack->top] = step;
        printf("Countdown step added!\n");
    }
}

void pop(struct Countdown *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped countdown step: %d\n", stack->steps[stack->top]);
        stack->top--;
    }
}

void display(struct Countdown stack) {
    if (isEmpty(&stack)) {
        printf("No countdown steps!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("Step: %d\n", stack.steps[i]);
        }
    }
}

void peek(struct Countdown stack) {
    if (isEmpty(&stack)) {
        printf("No steps to peek at!\n");
    } else {
        printf("Latest countdown step: %d\n", stack.steps[stack.top]);
```

```
    }
}

int search(struct Countdown stack, int step) {
    for (int i = 0; i <= stack.top; i++) {
        if (stack.steps[i] == step) {
            return 1;
        }
    }
    return 0;
}

int main() {
    struct Countdown stack;
    init(&stack);
    int choice, step;

    while (1) {
        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("%d", &step);
            push(&stack, step);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(stack);
            break;
        case 4:
            peek(stack);
            break;
        case 5:
            printf("Enter step to search: ");
            scanf("%d", &step);
            if (search(stack, step)) {
                printf("Step found!\n");
            } else {
                printf("Step not found!\n");
            }
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
    }
}
```

```
    return 0;
}
```

8.spacecraft docking procedure

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

#define MAX 50

struct CommandHistory {
    int top;
    char commands[MAX][100];
};

void init(struct CommandHistory *stack) {
    stack->top = -1;
}

int isFull(struct CommandHistory *stack) {
    return stack->top == MAX - 1;
}

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

void push(struct CommandHistory *stack, const char *command) {
    if (isFull(stack)) {
```

```
        printf("Stack is full!\n");
    } else {
        stack->top++;
        strcpy(stack->commands[stack->top], command);
        printf("Command added!\n");
    }
}

void pop(struct CommandHistory *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped command: %s\n", stack->commands[stack->top]);
        stack->top--;
    }
}

void display(struct CommandHistory stack) {
    if (isEmpty(&stack)) {
        printf("No commands in history!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("Command: %s\n", stack.commands[i]);
        }
    }
}

void peek(struct CommandHistory stack) {
    if (isEmpty(&stack)) {
```

```
        printf("No commands to peek at!\n");
    } else {
        printf("Latest command: %s\n", stack.commands[stack.top]);
    }
}

int search(struct CommandHistory stack, const char *command) {
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.commands[i], command) == 0) {
            return 1;
        }
    }
    return 0;
}

int main() {
    struct CommandHistory stack;
    init(&stack);
    int choice;
    char command[100];

    while (1) {
        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);
getchar();

switch (choice) {
    case 1:
        printf("Enter command: ");
        fgets(command, sizeof(command), stdin);
        command[strcspn(command, "\n")] = '\0';
        push(&stack, command);
        break;
    case 2:
        pop(&stack);
        break;
    case 3:
        display(stack);
        break;
    case 4:
        peek(stack);
        break;
    case 5:
        printf("Enter command to search: ");
        fgets(command, sizeof(command), stdin);
        command[strcspn(command, "\n")] = '\0';
        if (search(stack, command)) {
            printf("Command found!\n");
        } else {
            printf("Command not found!\n");
        }
        break;
```

```
            case 6:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }

    return 0;
}
```

9.aerospace simulation events

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

#define MAX 50

struct SimulationEvents {
    int top;
    char events[MAX][100];
};

void init(struct SimulationEvents *stack) {
    stack->top = -1;
}

int isFull(struct SimulationEvents *stack) {
    return stack->top == MAX - 1;
}
```

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

void push(struct SimulationEvents *stack, const char *event) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
    } else {
        stack->top++;
        strcpy(stack->events[stack->top], event);
        printf("Event added!\n");
    }
}

void pop(struct SimulationEvents *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped event: %s\n", stack->events[stack->top]);
        stack->top--;
    }
}

void display(struct SimulationEvents stack) {
    if (isEmpty(&stack)) {
        printf("No simulation events!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("Event: %s\n", stack.events[i]);
```

```
        }
    }
}

void peek(struct SimulationEvents stack) {
    if (isEmpty(&stack)) {
        printf("No events to peek at!\n");
    } else {
        printf("Latest event: %s\n", stack.events[stack.top]);
    }
}

int search(struct SimulationEvents stack, const char *event) {
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.events[i], event) == 0) {
            return 1;
        }
    }
    return 0;
}

int main() {
    struct SimulationEvents stack;
    init(&stack);
    int choice;
    char event[100];

    while (1) {
        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);
getchar();

switch (choice) {
   case 1:
      printf("Enter event: ");
      fgets(event, sizeof(event), stdin);
      event[strcspn(event, "\n")] = '\0';
      push(&stack, event);
      break;
   case 2:
      pop(&stack);
      break;
   case 3:
      display(stack);
      break;
   case 4:
      peek(stack);
      break;
   case 5:
      printf("Enter event to search: ");
      fgets(event, sizeof(event), stdin);
      event[strcspn(event, "\n")] = '\0';
```

```
            if (search(stack, event)) {
                printf("Event found!\n");
            } else {
                printf("Event not found!\n");
            }
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
      }
   }

   return 0;
}
```

10.pilot training manuever

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

#define MAX 50

struct ManeuverStack {
   int top;
   char maneuvers[MAX][100];
};
```

```
void init(struct ManeuverStack *stack) {
    stack->top = -1;
}

int isFull(struct ManeuverStack *stack) {
    return stack->top == MAX - 1;
}

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

void push(struct ManeuverStack *stack, const char *maneuver) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
    } else {
        stack->top++;
        strcpy(stack->maneuvers[stack->top], maneuver);
        printf("Maneuver added!\n");
    }
}

void pop(struct ManeuverStack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped maneuver: %s\n", stack->maneuvers[stack->top]);
        stack->top--;
    }
```

```
}

void display(struct ManeuverStack stack) {
    if (isEmpty(&stack)) {
        printf("No training maneuvers!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("Maneuver: %s\n", stack.maneuvers[i]);
        }
    }
}

void peek(struct ManeuverStack stack) {
    if (isEmpty(&stack)) {
        printf("No maneuvers to peek at!\n");
    } else {
        printf("Latest maneuver: %s\n", stack.maneuvers[stack.top]);
    }
}

int search(struct ManeuverStack stack, const char *maneuver) {
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.maneuvers[i], maneuver) == 0) {
            return 1;
        }
    }
    return 0;
}
```

```
int main() {
    struct ManeuverStack stack;
    init(&stack);
    int choice;
    char maneuver[100];

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

        switch (choice) {
            case 1:
                printf("Enter maneuver: ");
                fgets(maneuver, sizeof(maneuver), stdin);
                maneuver[strcspn(maneuver, "\n")] = '\0';
                push(&stack, maneuver);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                display(stack);
```

```
            break;
        case 4:
            peek(stack);
            break;
        case 5:
            printf("Enter maneuver to search: ");
            fgets(maneuver, sizeof(maneuver), stdin);
            maneuver[strcspn(maneuver, "\n")] = '\0';
            if (search(stack, maneuver)) {
                printf("Maneuver found!\n");
            } else {
                printf("Maneuver not found!\n");
            }
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
      }
   }

   return 0;
}
```

11.satellite operation commands

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

```
#define MAX 50

struct SatelliteCommands {
    int top;
    char commands[MAX][100];
};

void init(struct SatelliteCommands *stack) {
    stack->top = -1;
}

int isFull(struct SatelliteCommands *stack) {
    return stack->top == MAX - 1;
}

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

void push(struct SatelliteCommands *stack, const char *command) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
    } else {
        stack->top++;
        strcpy(stack->commands[stack->top], command);
        printf("Command added!\n");
    }
}
```

```
void pop(struct SatelliteCommands *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped command: %s\n", stack->commands[stack->top]);
        stack->top--;
    }
}

void display(struct SatelliteCommands stack) {
    if (isEmpty(&stack)) {
        printf("No operation commands!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("Command: %s\n", stack.commands[i]);
        }
    }
}

void peek(struct SatelliteCommands stack) {
    if (isEmpty(&stack)) {
        printf("No commands to peek at!\n");
    } else {
        printf("Latest command: %s\n", stack.commands[stack.top]);
    }
}

int search(struct SatelliteCommands stack, const char *command) {
    for (int i = 0; i <= stack.top; i++) {
```

```
        if (strcmp(stack.commands[i], command) == 0) {
            return 1;
        }
    }
    return 0;
}

int main() {
    struct SatelliteCommands stack;
    init(&stack);
    int choice;
    char command[100];

    while (1) {
        printf("1: Add a new command\n");
        printf("2: Remove 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);
        getchar();

        switch (choice) {
            case 1:
                printf("Enter command: ");
                fgets(command, sizeof(command), stdin);
                command[strcspn(command, "\n")] = '\0';
```

```
            push(&stack, command);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(stack);
            break;
        case 4:
            peek(stack);
            break;
        case 5:
            printf("Enter command to search: ");
            fgets(command, sizeof(command), stdin);
            command[strcspn(command, "\n")] = '\0';
            if (search(stack, command)) {
                printf("Command found!\n");
            } else {
                printf("Command not found!\n");
            }
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
    }
}

return 0;
```

```
}
```

12.emergency procedure for aircrafts

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

#define MAX 50

struct EmergencyProcedures {
    int top;
    char procedures[MAX][100];
};

void init(struct EmergencyProcedures *stack) {
    stack->top = -1;
}

int isFull(struct EmergencyProcedures *stack) {
    return stack->top == MAX - 1;
}

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

void push(struct EmergencyProcedures *stack, const char *procedure) {
    if (isFull(stack)) {
        printf("Stack is full!\n");
```

```
    } else {
        stack->top++;
        strcpy(stack->procedures[stack->top], procedure);
        printf("Procedure added!\n");
    }
}

void pop(struct EmergencyProcedures *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Popped procedure: %s\n", stack->procedures[stack->top]);
        stack->top--;
    }
}

void display(struct EmergencyProcedures stack) {
    if (isEmpty(&stack)) {
        printf("No emergency procedures!\n");
    } else {
        for (int i = stack.top; i >= 0; i--) {
            printf("Procedure: %s\n", stack.procedures[i]);
        }
    }
}

void peek(struct EmergencyProcedures stack) {
    if (isEmpty(&stack)) {
        printf("No procedures to peek at!\n");
```

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

int search(struct EmergencyProcedures stack, const char *procedure) {
    for (int i = 0; i <= stack.top; i++) {
        if (strcmp(stack.procedures[i], procedure) == 0) {
            return 1;
        }
    }
    return 0;
}

int main() {
    struct EmergencyProcedures stack;
    init(&stack);
    int choice;
    char procedure[100];

    while (1) {
        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);
```

```
getchar();

switch (choice) {
   case 1:
      printf("Enter procedure: ");
      fgets(procedure, sizeof(procedure), stdin);
      procedure[strcspn(procedure, "\n")] = '\0';
      push(&stack, procedure);
      break;
   case 2:
      pop(&stack);
      break;
   case 3:
      display(stack);
      break;
   case 4:
      peek(stack);
      break;
   case 5:
      printf("Enter procedure to search: ");
      fgets(procedure, sizeof(procedure), stdin);
      procedure[strcspn(procedure, "\n")] = '\0';
      if (search(stack, procedure)) {
         printf("Procedure found!\n");
      } else {
         printf("Procedure not found!\n");
      }
      break;
   case 6:
```

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

   return 0;
}
```

15.fuel management system

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

#define MAX 100
struct FuelStack {
   int top;
   int fuelUsage[MAX];
};

void initializeStack(struct FuelStack* stack);
int isFull(struct FuelStack stack);
int isEmpty(struct FuelStack stack);
void push(struct FuelStack* stack, int usage);
void pop(struct FuelStack* stack);
void display(struct FuelStack stack);
void peek(struct FuelStack stack);
int search(struct FuelStack stack, int usage);

int main() {
```

```
struct FuelStack stack;
initializeStack(&stack);
int choice, usage;

while (1) {
    printf("\nFuel 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 (in liters): ");
            scanf("%d", &usage);
            push(&stack, usage);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(stack);
            break;
        case 4:
            peek(stack);
```

```
            break;
        case 5:
            printf("Enter the fuel usage to search: ");
            scanf("%d", &usage);
            if (search(stack, usage)) {
                printf("Fuel usage found in the stack.\n");
            } else {
                printf("Fuel usage not found in the stack.\n");
            }
            break;
        case 6:
            printf("Exiting the system.\n");
            return 0;
        default:
            printf("Invalid option! Please try again.\n");
      }
   }

   return 0;
}

void initializeStack(struct FuelStack* stack) {
   stack->top = -1;
}

// Check if the stack is full
int isFull(struct FuelStack stack) {
   return stack.top == MAX - 1;
}
```

```
int isEmpty(struct FuelStack stack) {
    return stack.top == -1;
}

void push(struct FuelStack* stack, int usage) {
    if (isFull(*stack)) {
        printf("Stack is full, cannot add more entries.\n");
    } else {
        stack->top++;
        stack->fuelUsage[stack->top] = usage;
        printf("Fuel usage of %d liters added.\n", usage);
    }
}
void pop(struct FuelStack* stack) {
    if (isEmpty(*stack)) {
        printf("Stack is empty, no entry to remove.\n");
    } else {
        printf("Removed fuel usage entry: %d liters.\n", stack->fuelUsage[stack->top]);
        stack->top--;
    }
}

void display(struct FuelStack stack) {
    if (isEmpty(stack)) {
        printf("No fuel usage data to display.\n");
    } else {
        printf("Current fuel usage entries:\n");
```

```
        for (int i = stack.top; i >= 0; i--) {
            printf("Entry %d: %d liters\n", stack.top - i + 1, stack.fuelUsage[i]);
        }
    }
}

void peek(struct FuelStack stack) {
    if (isEmpty(stack)) {
        printf("No fuel usage entries available.\n");
    } else {
        printf("Latest fuel usage entry: %d liters\n", stack.fuelUsage[stack.top]);
    }
}



int search(struct FuelStack stack, int usage) {
    for (int i = 0; i <= stack.top; i++) {
        if (stack.fuelUsage[i] == usage) {
            return 1;
        }
    }
    return 0;
}
*/
```

----------using linkedlists----------

```
/*
```

1.order processing system

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

struct Order {
    int orderID;
    char description[100];
    struct Order* next;
};

struct Stack {
    struct Order* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int orderID, const char* description) {
    struct Order* newOrder = (struct Order*)malloc(sizeof(struct Order));
    newOrder->orderID = orderID;
    strcpy(newOrder->description, description);
    newOrder->next = stack->top;
    stack->top = newOrder;
```

```
    printf("Order %d added!\n", orderID);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No pending orders!\n");
    } else {
        struct Order* temp = stack->top;
        stack->top = stack->top->next;
        printf("Order %d processed: %s\n", temp->orderID, temp->description);
        free(temp);
    }
}

void display(struct Stack stack) {
    if (isEmpty(&stack)) {
        printf("No pending orders!\n");
    } else {
        struct Order* temp = stack.top;
        while (temp != NULL) {
            printf("Order ID: %d, Description: %s\n", temp->orderID, temp->description);
            temp = temp->next;
        }
    }
}

void peek(struct Stack stack) {
    if (isEmpty(&stack)) {
        printf("No orders to peek at!\n");
```

```
    } else {
        printf("Next order to be processed: Order ID: %d, Description: %s\n", stack.top->orderID, stack.top->description);
    }
}

int search(struct Stack stack, int orderID) {
    struct Order* temp = stack.top;
    while (temp != NULL) {
        if (temp->orderID == orderID) {
            return 1;
        }
        temp = temp->next;
    }
    return 0;
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, orderID;
    char description[100];

    while (1) {
        printf("\nOrder 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);
getchar();  // To consume newline character left by scanf

switch (choice) {
    case 1:
        printf("Enter order ID: ");
        scanf("%d", &orderID);
        getchar(); // To consume newline character
        printf("Enter order description: ");
        fgets(description, sizeof(description), stdin);
        description[strcspn(description, "\n")] = '\0'; // Removing the newline character
        push(&stack, orderID, description);
        break;

    case 2:
        pop(&stack);
        break;

    case 3:
        display(stack);
        break;

    case 4:
        peek(stack);
        break;
```

```
            case 5:
                printf("Enter order ID to search: ");
                scanf("%d", &orderID);
                if (search(stack, orderID)) {
                    printf("Order ID %d found!\n", orderID);
                } else {
                    printf("Order ID %d not found!\n", orderID);
                }
                break;

            case 6:
                exit(0);

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

    return 0;
}
```

2.customer support tracking system

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

struct Ticket {
    int ticketID;
    char description[100];
```

```
    struct Ticket* next;
};

struct Stack {
    struct Ticket* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int ticketID, const char* description) {
    struct Ticket* newTicket = (struct Ticket*)malloc(sizeof(struct Ticket));
    newTicket->ticketID = ticketID;
    strcpy(newTicket->description, description);
    newTicket->next = stack->top;
    stack->top = newTicket;
    printf("Ticket %d added!\n", ticketID);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No pending tickets!\n");
    } else {
        struct Ticket* temp = stack->top;
```

```
        stack->top = stack->top->next;
        printf("Ticket %d resolved: %s\n", temp->ticketID, temp->description);
        free(temp);
    }
}

void display(struct Stack stack) {
    if (isEmpty(&stack)) {
        printf("No pending tickets!\n");
    } else {
        struct Ticket* temp = stack.top;
        while (temp != NULL) {
            printf("Ticket ID: %d, Description: %s\n", temp->ticketID, temp->description);
            temp = temp->next;
        }
    }
}

void peek(struct Stack stack) {
    if (isEmpty(&stack)) {
        printf("No tickets to peek at!\n");
    } else {
        printf("Next ticket to resolve: Ticket ID: %d, Description: %s\n", stack.top->ticketID, sta
```

3.product return management

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

```
struct Return {
    int returnID;
    char productName[100];
    struct Return* next;
};

struct Stack {
    struct Return* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int returnID, const char* productName) {
    struct Return* newReturn = (struct Return*)malloc(sizeof(struct Return));
    newReturn->returnID = returnID;
    strcpy(newReturn->productName, productName);
    newReturn->next = stack->top;
    stack->top = newReturn;
    printf("Return %d added!\n", returnID);
}

void pop(struct Stack* stack) {
```

```
    if (isEmpty(stack)) {
        printf("No pending returns!\n");
    } else {
        struct Return* temp = stack->top;
        stack->top = stack->top->next;
        printf("Return %d processed for product: %s\n", temp->returnID, temp->productName);
        free(temp);
    }
}

void display(struct Stack stack) {
    if (isEmpty(&stack)) {
        printf("No pending returns!\n");
    } else {
        struct Return* temp = stack.top;
        while (temp != NULL) {
            printf("Return ID: %d, Product: %s\n", temp->returnID, temp->productName);
            temp = temp->next;
        }
    }
}

void peek(struct Stack stack) {
    if (isEmpty(&stack)) {
        printf("No returns to peek at!\n");
    } else {
        printf("Next return to process: Return ID: %d, Product: %s\n", stack.top->returnID, stack.top->productName);
    }
```

```
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, returnID;
    char productName[100];

    while (1) {
        printf("\nProduct Return Management\n");
        printf("1: Add a new return request (push)\n");
        printf("2: Process the last return (pop)\n");
        printf("3: View 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);
        getchar();

        switch (choice) {
            case 1:
                printf("Enter return ID: ");
                scanf("%d", &returnID);
                getchar(); // To consume newline character
                printf("Enter product name: ");
                fgets(productName, sizeof(productName), stdin);
                productName[strcspn(productName, "\n")] = '\0'; // Removing newline character
                push(&stack, returnID, productName);
```

```
            break;

        case 2:
            pop(&stack);
            break;

        case 3:
            display(stack);
            break;

        case 4:
            peek(stack);
            break;

        case 5:
            printf("Enter return ID to search: ");
            scanf("%d", &returnID);
            // Implement search functionality here if needed
            break;

        case 6:
            exit(0);

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

return 0;
```

```
}
```

4.inventory restock system

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

struct Restock {
    int restockID;
    char itemName[100];
    struct Restock* next;
};

struct Stack {
    struct Restock* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int restockID, const char* itemName) {
    struct Restock* newRestock = (struct Restock*)malloc(sizeof(struct Restock));
    newRestock->restockID = restockID;
```

```
    strcpy(newRestock->itemName, itemName);
    newRestock->next = stack->top;
    stack->top = newRestock;
    printf("Restock ID %d added!\n", restockID);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No pending restocks!\n");
    } else {
        struct Restock* temp = stack->top;
        stack->top = stack->top->next;
        printf("Restock ID %d processed for item: %s\n", temp->restockID, temp->itemName);
        free(temp);
    }
}

void display(struct Stack stack) {
    if (isEmpty(&stack)) {
        printf("No pending restocks!\n");
    } else {
        struct Restock* temp = stack.top;
        while (temp != NULL) {
            printf("Restock ID: %d, Item: %s\n", temp->restockID, temp->itemName);
            temp = temp->next;
        }
    }
}
```

```
void peek(struct Stack stack) {
    if (isEmpty(&stack)) {
        printf("No restocks to peek at!\n");
    } else {
        printf("Next restock to process: Restock ID: %d, Item: %s\n", stack.top->restockID, stack.top->itemName);
    }
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, restockID;
    char itemName[100];

    while (1) {
        printf("\nInventory Restock System\n");
        printf("1: Add a restock entry (push)\n");
        printf("2: Process the last restock (pop)\n");
        printf("3: View all restock entries\n");
        printf("4: Peek at the latest restock entry\n");
        printf("5: Search for a specific restock entry\n");
        printf("6: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();

        switch (choice) {
            case 1:
```

```
        printf("Enter restock ID: ");
        scanf("%d", &restockID);
        getchar();
        printf("Enter item name: ");
        fgets(itemName, sizeof(itemName), stdin);
        itemName[strcspn(itemName, "\n")] = '\0';
        push(&stack, restockID, itemName);
        break;

    case 2:
        pop(&stack);
        break;

    case 3:
        display(stack);
        break;

    case 4:
        peek(stack);
        break;

    case 5:
        printf("Enter restock ID to search: ");
        scanf("%d", &restockID);
        // Implement search functionality here if needed
        break;

    case 6:
        exit(0);
```

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

    return 0;
}
```

5. product return management

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

struct ReturnRequest {
    int returnID;
    char productName[100];
    struct ReturnRequest* next;
};

struct Stack {
    struct ReturnRequest* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

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

```
    return stack->top == NULL;
}

void push(struct Stack* stack, int returnID, const char* productName) {
    struct ReturnRequest* newRequest = (struct ReturnRequest*)malloc(sizeof(struct ReturnRequest));
    newRequest->returnID = returnID;
    strcpy(newRequest->productName, productName);
    newRequest->next = stack->top;
    stack->top = newRequest;
    printf("Return request %d added!\n", returnID);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No pending return requests!\n");
    } else {
        struct ReturnRequest* temp = stack->top;
        stack->top = stack->top->next;
        printf("Processed return request %d: %s\n", temp->returnID, temp->productName);
        free(temp);
    }
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No return requests to display!\n");
    } else {
        struct ReturnRequest* temp = stack->top;
        while (temp != NULL) {
```

```
            printf("Return ID: %d, Product: %s\n", temp->returnID, temp->productName);
            temp = temp->next;
        }
    }
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, returnID;
    char productName[100];

    while (1) {
        printf("\nProduct Return Management\n");
        printf("1: Add a new return request\n");
        printf("2: Process the last return\n");
        printf("3: View all 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);
                getchar();
                printf("Enter product name: ");
                fgets(productName, sizeof(productName), stdin);
                productName[strcspn(productName, "\n")] = '\0';
                push(&stack, returnID, productName);
```

```
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(&stack);
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice, try again.\n");
      }
   }
}
```

6.inventory restock system

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

struct Restock {
   int restockID;
   char productName[100];
   struct Restock* next;
};

struct Stack {
   struct Restock* top;
```

```
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int restockID, const char* productName) {
    struct Restock* newRestock = (struct Restock*)malloc(sizeof(struct Restock));
    newRestock->restockID = restockID;
    strcpy(newRestock->productName, productName);
    newRestock->next = stack->top;
    stack->top = newRestock;
    printf("Restock request %d added!\n", restockID);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No restock requests to process!\n");
    } else {
        struct Restock* temp = stack->top;
        stack->top = stack->top->next;
        printf("Processed restock request %d: %s\n", temp->restockID, temp->productName);
        free(temp);
    }
}
```

```
void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No restock requests to display!\n");
    } else {
        struct Restock* temp = stack->top;
        while (temp != NULL) {
            printf("Restock ID: %d, Product: %s\n", temp->restockID, temp->productName);
            temp = temp->next;
        }
    }
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, restockID;
    char productName[100];

    while (1) {
        printf("\nInventory Restock System\n");
        printf("1: Add a restock entry\n");
        printf("2: Process the last restock\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);
            getchar();
            printf("Enter product name: ");
            fgets(productName, sizeof(productName), stdin);
            productName[strcspn(productName, "\n")] = '\0';
            push(&stack, restockID, productName);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(&stack);
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice, try again.\n");
      }
   }
}
```

7.

flash sale deal management

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

```
struct FlashSaleDeal {
    int dealID;
    char productName[100];
    struct FlashSaleDeal* next;
};

struct Stack {
    struct FlashSaleDeal* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int dealID, const char* productName) {
    struct FlashSaleDeal* newDeal = (struct FlashSaleDeal*)malloc(sizeof(struct FlashSaleDeal));
    newDeal->dealID = dealID;
    strcpy(newDeal->productName, productName);
    newDeal->next = stack->top;
    stack->top = newDeal;
    printf("Deal %d added!\n", dealID);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
```

```
        printf("No active deals!\n");
    } else {
        struct FlashSaleDeal* temp = stack->top;
        stack->top = stack->top->next;
        printf("Deal %d removed: %s\n", temp->dealID, temp->productName);
        free(temp);
    }
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No active deals to display!\n");
    } else {
        struct FlashSaleDeal* temp = stack->top;
        while (temp != NULL) {
            printf("Deal ID: %d, Product: %s\n", temp->dealID, temp->productName);
            temp = temp->next;
        }
    }
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, dealID;
    char productName[100];

    while (1) {
        printf("\nFlash Sale Deal Management\n");
```

```
    printf("1: Add a new deal\n");
    printf("2: Remove the last deal\n");
    printf("3: View all active deals\n");
    printf("4: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            printf("Enter deal ID: ");
            scanf("%d", &dealID);
            getchar();
            printf("Enter product name: ");
            fgets(productName, sizeof(productName), stdin);
            productName[strcspn(productName, "\n")] = '\0';
            push(&stack, dealID, productName);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(&stack);
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice, try again.\n");
    }
}
```

```
}
```

8.unseen session history

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

struct UserSession {
    int sessionID;
    char userName[100];
    struct UserSession* next;
};

struct Stack {
    struct UserSession* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int sessionID, const char* userName) {
    struct UserSession* newSession = (struct UserSession*)malloc(sizeof(struct UserSession));
    newSession->sessionID = sessionID;
```

```
    strcpy(newSession->userName, userName);
    newSession->next = stack->top;
    stack->top = newSession;
    printf("Session %d added for user: %s\n", sessionID, userName);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No active sessions!\n");
    } else {
        struct UserSession* temp = stack->top;
        stack->top = stack->top->next;
        printf("Session %d ended for user: %s\n", temp->sessionID, temp->userName);
        free(temp);
    }
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No active sessions to display!\n");
    } else {
        struct UserSession* temp = stack->top;
        while (temp != NULL) {
            printf("Session ID: %d, User: %s\n", temp->sessionID, temp->userName);
            temp = temp->next;
        }
    }
}
```

```
int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, sessionID;
    char userName[100];

    while (1) {
        printf("\nUser Session History\n");
        printf("1: Add a session\n");
        printf("2: End the last session\n");
        printf("3: View all active sessions\n");
        printf("4: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter session ID: ");
                scanf("%d", &sessionID);
                getchar();
                printf("Enter user name: ");
                fgets(userName, sizeof(userName), stdin);
                userName[strcspn(userName, "\n")] = '\0';
                push(&stack, sessionID, userName);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                display(&stack);
```

```
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice, try again.\n");
      }
   }
}
```

9.wishlist management

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

struct WishlistItem {
   int itemID;
   char productName[100];
   struct WishlistItem* next;
};

struct Stack {
   struct WishlistItem* top;
};

void initStack(struct Stack* stack) {
   stack->top = NULL;
}
```

```
int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int itemID, const char* productName) {
    struct WishlistItem* newItem = (struct WishlistItem*)malloc(sizeof(struct WishlistItem));
    newItem->itemID = itemID;
    strcpy(newItem->productName, productName);
    newItem->next = stack->top;
    stack->top = newItem;
    printf("Wishlist item %d added: %s\n", itemID, productName);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No items in the wishlist!\n");
    } else {
        struct WishlistItem* temp = stack->top;
        stack->top = stack->top->next;
        printf("Wishlist item %d removed: %s\n", temp->itemID, temp->productName);
        free(temp);
    }
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No items in the wishlist!\n");
    } else {
        struct WishlistItem* temp = stack->top;
```

```
        while (temp != NULL) {
            printf("Item ID: %d, Product: %s\n", temp->itemID, temp->productName);
            temp = temp->next;
        }
    }
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, itemID;
    char productName[100];

    while (1) {
        printf("\nWishlist Management\n");
        printf("1: Add a product to wishlist\n");
        printf("2: Remove the last added product\n");
        printf("3: View all wishlist items\n");
        printf("4: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter item ID: ");
                scanf("%d", &itemID);
                getchar();
                printf("Enter product name: ");
                fgets(productName, sizeof(productName), stdin);
                productName[strcspn(productName, "\n")] = '\0';
```

```
            push(&stack, itemID, productName);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(&stack);
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice, try again.\n");
      }
   }
}
```

10.checkout process steps

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

struct CheckoutStep {
   int stepID;
   char stepDescription[100];
   struct CheckoutStep* next;
};

struct Stack {
```

```
    struct CheckoutStep* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int stepID, const char* stepDescription) {
    struct CheckoutStep* newStep = (struct CheckoutStep*)malloc(sizeof(struct CheckoutStep));
    newStep->stepID = stepID;
    strcpy(newStep->stepDescription, stepDescription);
    newStep->next = stack->top;
    stack->top = newStep;
    printf("Checkout step %d added: %s\n", stepID, stepDescription);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No checkout steps left to process!\n");
    } else {
        struct CheckoutStep* temp = stack->top;
        stack->top = stack->top->next;
        printf("Checkout step %d removed: %s\n", temp->stepID, temp->stepDescription);
        free(temp);
    }
```

```
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No checkout steps to display!\n");
    } else {
        struct CheckoutStep* temp = stack->top;
        while (temp != NULL) {
            printf("Step ID: %d, Description: %s\n", temp->stepID, temp->stepDescription);
            temp = temp->next;
        }
    }
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, stepID;
    char stepDescription[100];

    while (1) {
        printf("\nCheckout Process Steps\n");
        printf("1: Add a checkout step\n");
        printf("2: Remove the last step\n");
        printf("3: View all checkout steps\n");
        printf("4: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
```

```
            case 1:
                printf("Enter step ID: ");
                scanf("%d", &stepID);
                getchar();
                printf("Enter step description: ");
                fgets(stepDescription, sizeof(stepDescription), stdin);
                stepDescription[strcspn(stepDescription, "\n")] = '\0';
                push(&stack, stepID, stepDescription);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                display(&stack);
                break;
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice, try again.\n");
        }
    }
}
```

11.coupen code management

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

```
struct CouponCode {
    int couponID;
    char code[100];
    struct CouponCode* next;
};

struct Stack {
    struct CouponCode* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int couponID, const char* code) {
    struct CouponCode* newCoupon = (struct CouponCode*)malloc(sizeof(struct CouponCode));
    newCoupon->couponID = couponID;
    strcpy(newCoupon->code, code);
    newCoupon->next = stack->top;
    stack->top = newCoupon;
    printf("Coupon code %d added: %s\n", couponID, code);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
```

```
        printf("No coupon codes to remove!\n");
    } else {
        struct CouponCode* temp = stack->top;
        stack->top = stack->top->next;
        printf("Coupon code %d removed: %s\n", temp->couponID, temp->code);
        free(temp);
    }
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No coupon codes to display!\n");
    } else {
        struct CouponCode* temp = stack->top;
        while (temp != NULL) {
            printf("Coupon ID: %d, Code: %s\n", temp->couponID, temp->code);
            temp = temp->next;
        }
    }
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, couponID;
    char code[100];

    while (1) {
        printf("\nCoupon Code Management\n");
```

```
    printf("1: Add a new coupon code\n");
    printf("2: Remove the last coupon code\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 ID: ");
            scanf("%d", &couponID);
            getchar();
            printf("Enter coupon code: ");
            fgets(code, sizeof(code), stdin);
            code[strcspn(code, "\n")] = '\0';
            push(&stack, couponID, code);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(&stack);
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice, try again.\n");
    }
}
```

```
}
```

12.shipping status tracker

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

struct ShippingStatus {
    int statusID;
    char statusDescription[100];
    struct ShippingStatus* next;
};

struct Stack {
    struct ShippingStatus* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int statusID, const char* statusDescription) {
    struct ShippingStatus* newStatus = (struct ShippingStatus*)malloc(sizeof(struct ShippingStatus));
    newStatus->statusID = statusID;
    strcpy(newStatus->statusDescription, statusDescription);
```

```
    newStatus->next = stack->top;
    stack->top = newStatus;
    printf("Shipping status %d added: %s\n", statusID, statusDescription);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No shipping status updates to remove!\n");
    } else {
        struct ShippingStatus* temp = stack->top;
        stack->top = stack->top->next;
        printf("Shipping status %d removed: %s\n", temp->statusID, temp->statusDescription);
        free(temp);
    }
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No shipping status updates to display!\n");
    } else {
        struct ShippingStatus* temp = stack->top;
        while (temp != NULL) {
            printf("Status ID: %d, Description: %s\n", temp->statusID, temp->statusDescription);
            temp = temp->next;
        }
    }
}

int main() {
```

```
struct Stack stack;
initStack(&stack);
int choice, statusID;
char statusDescription[100];

while (1) {
    printf("\nShipping Status Tracker\n");
    printf("1: Add a shipping status update\n");
    printf("2: Remove the last update\n");
    printf("3: View all shipping status updates\n");
    printf("4: Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            printf("Enter status ID: ");
            scanf("%d", &statusID);
            getchar();
            printf("Enter status description: ");
            fgets(statusDescription, sizeof(statusDescription), stdin);
            statusDescription[strcspn(statusDescription, "\n")] = '\0';
            push(&stack, statusID, statusDescription);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(&stack);
            break;
```

```
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice, try again.\n");
        }
    }
}
```

13.user review management

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

struct UserReview {
    int reviewID;
    char reviewText[200];
    struct UserReview* next;
};

struct Stack {
    struct UserReview* top;
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

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

```
    return stack->top == NULL;
}

void push(struct Stack* stack, int reviewID, const char* reviewText) {
    struct UserReview* newReview = (struct UserReview*)malloc(sizeof(struct UserReview));
    newReview->reviewID = reviewID;
    strcpy(newReview->reviewText, reviewText);
    newReview->next = stack->top;
    stack->top = newReview;
    printf("User review %d added: %s\n", reviewID, reviewText);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No reviews to remove!\n");
    } else {
        struct UserReview* temp = stack->top;
        stack->top = stack->top->next;
        printf("User review %d removed: %s\n", temp->reviewID, temp->reviewText);
        free(temp);
    }
}

void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No reviews to display!\n");
    } else {
        struct UserReview* temp = stack->top;
        while (temp != NULL) {
```

```
            printf("Review ID: %d, Review Text: %s\n", temp->reviewID, temp->reviewText);
            temp = temp->next;
        }
    }
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, reviewID;
    char reviewText[200];

    while (1) {
        printf("\nUser Review Management\n");
        printf("1: Add a new review\n");
        printf("2: Remove the last review\n");
        printf("3: View all reviews\n");
        printf("4: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter review ID: ");
                scanf("%d", &reviewID);
                getchar();
                printf("Enter review text: ");
                fgets(reviewText, sizeof(reviewText), stdin);
                reviewText[strcspn(reviewText, "\n")] = '\0';
                push(&stack, reviewID, reviewText);
```

```
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(&stack);
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice, try again.\n");
      }
   }
}
```

14.promrotion notification system

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

struct Notification {
   int notificationID;
   char message[200];
   struct Notification* next;
};

struct Stack {
   struct Notification* top;
```

```
};

void initStack(struct Stack* stack) {
    stack->top = NULL;
}

int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

void push(struct Stack* stack, int notificationID, const char* message) {
    struct Notification* newNotification = (struct Notification*)malloc(sizeof(struct Notification));
    newNotification->notificationID = notificationID;
    strcpy(newNotification->message, message);
    newNotification->next = stack->top;
    stack->top = newNotification;
    printf("Promotion notification %d added: %s\n", notificationID, message);
}

void pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No notifications to remove!\n");
    } else {
        struct Notification* temp = stack->top;
        stack->top = stack->top->next;
        printf("Promotion notification %d removed: %s\n", temp->notificationID, temp->message);
        free(temp);
    }
}
```

```
void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("No notifications to display!\n");
    } else {
        struct Notification* temp = stack->top;
        while (temp != NULL) {
            printf("Notification ID: %d, Message: %s\n", temp->notificationID, temp->message);
            temp = temp->next;
        }
    }
}

int main() {
    struct Stack stack;
    initStack(&stack);
    int choice, notificationID;
    char message[200];

    while (1) {
        printf("\nPromotion Notification System\n");
        printf("1: Add a new notification\n");
        printf("2: Remove the last notification\n");
        printf("3: View all notifications\n");
        printf("4: Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
```

```
            printf("Enter notification ID: ");
            scanf("%d", &notificationID);
            getchar();
            printf("Enter notification message: ");
            fgets(message, sizeof(message), stdin);
            message[strcspn(message, "\n")] = '\0';
            push(&stack, notificationID, message);
            break;
        case 2:
            pop(&stack);
            break;
        case 3:
            display(&stack);
            break;
        case 4:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice, try again.\n");
    }
  }
}
```