**17-01-2025**

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
2. #include<stdio.h>
3. #include<stdlib.h>
4. #include<string.h>
5. struct Stack {
6. int size;
7. int top;
8. char \*\*paths;
9. };
10. void create(struct Stack \*st);
11. void push(struct Stack \*st, char \*path);
12. char \*pop(struct Stack \*st);
13. void display(struct Stack st);
14. char \*peek(struct Stack st);
15. int search(struct Stack st, char \*path);
16. int main() {
17. struct Stack st;
18. create(&st);
19. int choice;
20. char path[100];
21. do {
22. printf("\n1. Add path\n2. Undo last path\n3. Display paths\n4. Peek top path\n5. Search path\n6. Exit\n");
23. printf("Enter choice: ");
24. scanf("%d", &choice);
25. getchar();  // To consume the newline
26. switch (choice) {
27. case 1:
28. printf("Enter path: ");
29. fgets(path, 100, stdin);
30. path[strlen(path) - 1] = '\0';  // Remove newline
31. push(&st, path);
32. break;
33. case 2:
34. printf("Undone: %s\n", pop(&st));
35. break;
36. case 3:
37. display(st);
38. break;
39. case 4:
40. printf("Top path: %s\n", peek(st));
41. break;
42. case 5:
43. printf("Enter path to search: ");
44. fgets(path, 100, stdin);
45. path[strlen(path) - 1] = '\0';  // Remove newline
46. int index = search(st, path);
47. if (index != -1) printf("Found at position %d\n", index + 1);
48. else printf("Path not found\n");
49. break;
50. }
51. } while (choice != 6);
52. return 0;
53. }
54. void create(struct Stack \*st) {
55. printf("Enter stack size: ");
56. scanf("%d", &st->size);
57. st->top = -1;
58. st->paths = (char \*\*)malloc(st->size \* sizeof(char \*));
59. }
60. void push(struct Stack \*st, char \*path) {
61. if (st->top < st->size - 1) {
62. st->top++;
63. st->paths[st->top] = (char \*)malloc((strlen(path) + 1) \* sizeof(char));
64. strcpy(st->paths[st->top], path);
65. } else {
66. printf("Stack full\n");
67. }
68. }
69. char \*pop(struct Stack \*st) {
70. if (st->top >= 0) {
71. char \*path = st->paths[st->top];
72. st->top--;
73. return path;
74. }
75. return "Stack empty";
76. }
77. void display(struct Stack st) {
78. if (st.top == -1) {
79. printf("Stack empty\n");
80. return;
81. }
82. for (int i = st.top; i >= 0; i--) {
83. printf("%s\n", st.paths[i]);
84. }
85. }
86. char \*peek(struct Stack st) {
87. return st.top >= 0 ? st.paths[st.top] : "Stack empty";
88. }
89. int search(struct Stack st, char \*path) {
90. for (int i = 0; i <= st.top; i++) {
91. if (strcmp(st.paths[i], path) == 0) return i;
92. }
93. return -1;
94. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*deployments;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*deployment);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*deployment);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char deployment[100];
20. do {
21. printf("\n1. Push Deployment\n2. Pop Deployment\n3. View Deployments\n4. Peek Top Deployment\n5. Search Deployment\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();
25. switch (choice) {
26. case 1:
27. printf("Enter deployment name: ");
28. scanf("%s", deployment);
29. push(&st, deployment);
30. break;
31. case 2:
32. printf("Popped: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Latest deployment: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter deployment to search: ");
42. scanf("%s", deployment);  // Read input without a newline
43. int index = search(st, deployment);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Deployment not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->deployments = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*deployment) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->deployments[st->top] = (char \*)malloc((strlen(deployment) + 1) \* sizeof(char));
61. strcpy(st->deployments[st->top], deployment);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*deployment = st->deployments[st->top];
69. st->top--;
70. return deployment;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("Stack empty\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.deployments[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.deployments[st.top] : "Stack empty";
85. }
86. int search(struct Stack st, char \*deployment) {
87. for (int i = 0; i <= st.top; i++) {
88. if (strcmp(st.deployments[i], deployment) == 0) return i;
89. }
90. return -1;
91. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*checklist;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*item);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*item);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char item[100];
20. do {
21. printf("\n1. Add Checklist Item\n2. Remove Last Item\n3. Display Checklist\n4. Peek at Top Item\n5. Search for Item\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();  // To consume the newline character after entering the choice
25. switch (choice) {
26. case 1:
27. printf("Enter checklist item: ");
28. scanf("%99[^\n]", item);  // Read a line until newline is encountered
29. push(&st, item);
30. break;
31. case 2:
32. printf("Removed: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Top checklist item: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter item to search: ");
42. scanf("%99[^\n]", item);  // Read a line until newline is encountered
43. int index = search(st, item);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Item not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->checklist = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*item) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->checklist[st->top] = (char \*)malloc((strlen(item) + 1) \* sizeof(char));
61. strcpy(st->checklist[st->top], item);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*item = st->checklist[st->top];
69. st->top--;
70. return item;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("Stack empty\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.checklist[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.checklist[st.top] : "Stack empty";
85. }
86. int search(struct Stack st, char \*item) {
87. for (int i = 0; i <= st.top; i++) {
88. if (strcmp(st.checklist[i], item) == 0) return i;
89. }
90. return -1;
91. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*data;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*entry);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*entry);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char entry[100];
20. do {
21. printf("\n1. Push Telemetry Data\n2. Pop Data Entry\n3. View Stored Data\n4. Peek at Most Recent Data\n5. Search for Data\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();  // To consume the newline character after entering the choice
25. switch (choice) {
26. case 1:
27. printf("Enter telemetry data: ");
28. scanf("%99[^\n]", entry);  // Read the data entry
29. push(&st, entry);
30. break;
31. case 2:
32. printf("Popped: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Most recent data: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter data to search: ");
42. scanf("%99[^\n]", entry);  // Read the data entry to search
43. int index = search(st, entry);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Data not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->data = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*entry) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->data[st->top] = (char \*)malloc((strlen(entry) + 1) \* sizeof(char));
61. strcpy(st->data[st->top], entry);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*entry = st->data[st->top];
69. st->top--;
70. return entry;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("Stack empty\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.data[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.data[st.top] : "Stack empty";
85. }
86. int search(struct Stack st, char \*entry) {
87. for (int i = 0; i <= st.top; i++) {
88. if (strcmp(st.data[i], entry) == 0) return i;
89. }
90. return -1;
91. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*tasks;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*task);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*task);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char task[100];
20. do {
21. printf("\n1. Add Task\n2. Mark Last Task as Completed\n3. List Pending Tasks\n4. Peek at Most Recent Task\n5. Search for Task\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();  // To consume the newline character after entering the choice
25. switch (choice) {
26. case 1:
27. printf("Enter task: ");
28. scanf("%99[^\n]", task);  // Read the task
29. push(&st, task);
30. break;
31. case 2:
32. printf("Completed task: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Most recent task: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter task to search: ");
42. scanf("%99[^\n]", task);  // Read the task to search
43. int index = search(st, task);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Task not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->tasks = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*task) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->tasks[st->top] = (char \*)malloc((strlen(task) + 1) \* sizeof(char));
61. strcpy(st->tasks[st->top], task);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*task = st->tasks[st->top];
69. st->top--;
70. return task;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("No pending tasks\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.tasks[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.tasks[st.top] : "No pending tasks";
85. }
86. int search(struct Stack st, char \*task) {
87. for (int i = 0; i <= st.top; i++) {
88. if (strncmp(st.tasks[i], task, strlen(task)) == 0) {
89. return i;
90. }
91. }
92. return -1;
93. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*countdownSteps;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*step);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*step);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char step[100];
20. do {
21. printf("\n1. Add Countdown Step\n2. Remove Last Step\n3. Display Countdown Steps\n4. Peek at Next Step\n5. Search for Step\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();  // To consume the newline character after entering the choice
25. switch (choice) {
26. case 1:
27. printf("Enter countdown step: ");
28. scanf("%99[^\n]", step);  // Read the countdown step
29. push(&st, step);
30. break;
31. case 2:
32. printf("Removed step: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Next countdown step: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter step to search: ");
42. scanf("%99[^\n]", step);  // Read the step to search
43. int index = search(st, step);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Step not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->countdownSteps = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*step) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->countdownSteps[st->top] = (char \*)malloc((strlen(step) + 1) \* sizeof(char));
61. strcpy(st->countdownSteps[st->top], step);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*step = st->countdownSteps[st->top];
69. st->top--;
70. return step;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("No countdown steps\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.countdownSteps[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.countdownSteps[st.top] : "No pending countdown step";
85. }
86. int search(struct Stack st, char \*step) {
87. for (int i = 0; i <= st.top; i++) {
88. // Using strncmp to compare the countdown steps
89. if (strncmp(st.countdownSteps[i], step, strlen(step)) == 0) {
90. return i;
91. }
92. }
93. return -1;
94. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*logs;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*log);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*log);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char log[100];
20. do {
21. printf("\n1. Add Maintenance Log\n2. Remove Last Log\n3. View All Logs\n4. Peek at Latest Log\n5. Search for Log\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();
25. switch (choice) {
26. case 1:
27. printf("Enter maintenance log: ");
28. scanf("%99[^\n]", log);  // Read the log
29. push(&st, log);
30. break;
31. case 2:
32. printf("Removed log: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Latest log: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter log to search: ");
42. scanf("%99[^\n]", log);  // Read the log to search
43. int index = search(st, log);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Log not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->logs = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*log) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->logs[st->top] = (char \*)malloc((strlen(log) + 1) \* sizeof(char));
61. strcpy(st->logs[st->top], log);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*log = st->logs[st->top];
69. st->top--;
70. return log;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("No maintenance logs\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.logs[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.logs[st.top] : "No maintenance log";
85. }
86. int search(struct Stack st, char \*log) {
87. for (int i = 0; i <= st.top; i++) {
88. // Using strncmp to compare the logs
89. if (strncmp(st.logs[i], log, strlen(log)) == 0) {
90. return i;
91. }
92. }
93. return -1;
94. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*steps;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*step);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*step);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char step[100];
20. do {
21. printf("\n1. Push New Step\n2. Pop Last Step\n3. Display Procedure Steps\n4. Peek Next Step\n5. Search for Step\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();  // To consume the newline character after entering the choice
25. switch (choice) {
26. case 1:
27. printf("Enter docking procedure step: ");
28. scanf("%99[^\n]", step);  // Read the step
29. push(&st, step);
30. break;
31. case 2:
32. printf("Removed step: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Next step: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter step to search: ");
42. scanf("%99[^\n]", step);  // Read the step to search
43. int index = search(st, step);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Step not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->steps = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*step) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->steps[st->top] = (char \*)malloc((strlen(step) + 1) \* sizeof(char));
61. strcpy(st->steps[st->top], step);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*step = st->steps[st->top];
69. st->top--;
70. return step;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("No procedure steps\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.steps[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.steps[st.top] : "No steps left";
85. }
86. int search(struct Stack st, char \*step) {
87. for (int i = 0; i <= st.top; i++) {
88. if (strncmp(st.steps[i], step, strlen(step)) == 0) {
89. return i;
90. }
91. }
92. return -1;
93. }
94. **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
95. #include<stdio.h>
96. #include<stdlib.h>
97. #include<string.h>
98. struct Stack {
99. int size;
100. int top;
101. char \*\*commands;
102. };
103. void create(struct Stack \*st);
104. void push(struct Stack \*st, char \*command);
105. char \*pop(struct Stack \*st);
106. void display(struct Stack st);
107. char \*peek(struct Stack st);
108. int search(struct Stack st, char \*command);
109. int main() {
110. struct Stack st;
111. create(&st);
112. int choice;
113. char command[100];
114. do {
115. printf("\n1. Add Command\n2. Undo Last Command\n3. View Command History\n4. Peek Recent Command\n5. Search for Command\n6. Exit\n");
116. printf("Enter choice: ");
117. scanf("%d", &choice);
118. getchar();
119. switch (choice) {
120. case 1:
121. printf("Enter command: ");
122. scanf("%99[^\n]", command);  // Read the command
123. push(&st, command);
124. break;
125. case 2:
126. printf("Undone command: %s\n", pop(&st));
127. break;
128. case 3:
129. display(st);
130. break;
131. case 4:
132. printf("Most recent command: %s\n", peek(st));
133. break;
134. case 5:
135. printf("Enter command to search: ");
136. scanf("%99[^\n]", command);  // Read the command to search
137. int index = search(st, command);
138. if (index != -1) printf("Found at position %d\n", index + 1);
139. else printf("Command not found\n");
140. break;
141. }
142. } while (choice != 6);
143. return 0;
144. }
145. void create(struct Stack \*st) {
146. printf("Enter stack size: ");
147. scanf("%d", &st->size);
148. st->top = -1;
149. st->commands = (char \*\*)malloc(st->size \* sizeof(char \*));
150. }
151. void push(struct Stack \*st, char \*command) {
152. if (st->top < st->size - 1) {
153. st->top++;
154. st->commands[st->top] = (char \*)malloc((strlen(command) + 1) \* sizeof(char));
155. strcpy(st->commands[st->top], command);
156. } else {
157. printf("Stack full\n");
158. }
159. }
160. char \*pop(struct Stack \*st) {
161. if (st->top >= 0) {
162. char \*command = st->commands[st->top];
163. st->top--;
164. return command;
165. }
166. return "Stack empty";
167. }
168. void display(struct Stack st) {
169. if (st.top == -1) {
170. printf("No commands in history\n");
171. return;
172. }
173. for (int i = st.top; i >= 0; i--) {
174. printf("%s\n", st.commands[i]);
175. }
176. }
177. char \*peek(struct Stack st) {
178. return st.top >= 0 ? st.commands[st.top] : "No commands available";
179. }
180. int search(struct Stack st, char \*command) {
181. for (int i = 0; i <= st.top; i++) {
182. if (strncmp(st.commands[i], command, strlen(command)) == 0) {
183. return i;
184. }
185. }
186. return -1;
187. }

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

* + 1: Push a new event
  + 2: Pop the last event
  + 3: Display all events
  + 4: Peek at the most recent event
  + 5: Search for a specific event
  + 6: Exit

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*events;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*event);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*event);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char event[100];
20. do {
21. printf("\n1. Push New Event\n2. Pop Last Event\n3. Display All Events\n4. Peek Recent Event\n5. Search for Event\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();  // To consume the newline character after entering the choice
25. switch (choice) {
26. case 1:
27. printf("Enter event: ");
28. scanf("%99[^\n]", event);  // Read the event
29. push(&st, event);
30. break;
31. case 2:
32. printf("Popped event: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Most recent event: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter event to search: ");
42. scanf("%99[^\n]", event);  // Read the event to search
43. int index = search(st, event);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Event not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->events = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*event) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->events[st->top] = (char \*)malloc((strlen(event) + 1) \* sizeof(char));
61. strcpy(st->events[st->top], event);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*event = st->events[st->top];
69. st->top--;
70. return event;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("No events recorded\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.events[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.events[st.top] : "No events available";
85. }
86. int search(struct Stack st, char \*event) {
87. for (int i = 0; i <= st.top; i++) {
88. if (strncmp(st.events[i], event, strlen(event)) == 0) {
89. return i;
90. }
91. }
92. return -1;
93. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*maneuvers;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*maneuver);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*maneuver);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char maneuver[100];
20. do {
21. printf("\n1. Add Maneuver\n2. Remove Last Maneuver\n3. View All Maneuvers\n4. Peek Recent Maneuver\n5. Search for Maneuver\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();
25. switch (choice) {
26. case 1:
27. printf("Enter maneuver: ");
28. scanf("%99[^\n]", maneuver);  // Read the maneuver
29. push(&st, maneuver);
30. break;
31. case 2:
32. printf("Removed maneuver: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Most recent maneuver: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter maneuver to search: ");
42. scanf("%99[^\n]", maneuver);  // Read the maneuver to search
43. int index = search(st, maneuver);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Maneuver not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->maneuvers = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*maneuver) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->maneuvers[st->top] = (char \*)malloc((strlen(maneuver) + 1) \* sizeof(char));
61. strcpy(st->maneuvers[st->top], maneuver);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*maneuver = st->maneuvers[st->top];
69. st->top--;
70. return maneuver;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("No maneuvers recorded\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.maneuvers[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.maneuvers[st.top] : "No maneuvers available";
85. }
86. int search(struct Stack st, char \*maneuver) {
87. for (int i = 0; i <= st.top; i++) {
88. if (strncmp(st.maneuvers[i], maneuver, strlen(maneuver)) == 0) {
89. return i;
90. }
91. }
92. return -1;
93. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*commands;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*command);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*command);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char command[100];
20. do {
21. printf("\n1. Push New Command\n2. Pop Last Command\n3. View All Commands\n4. Peek Recent Command\n5. Search for Command\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();
25. switch (choice) {
26. case 1:
27. printf("Enter command: ");
28. scanf("%99[^\n]", command);  // Read the command
29. push(&st, command);
30. break;
31. case 2:
32. printf("Popped command: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Most recent command: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter command to search: ");
42. scanf("%99[^\n]", command);  // Read the command to search
43. int index = search(st, command);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Command not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->commands = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*command) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->commands[st->top] = (char \*)malloc((strlen(command) + 1) \* sizeof(char));
61. strcpy(st->commands[st->top], command);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*command = st->commands[st->top];
69. st->top--;
70. return command;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("No commands recorded\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.commands[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.commands[st.top] : "No commands available";
85. }
86. int search(struct Stack st, char \*command) {
87. for (int i = 0; i <= st.top; i++) {
88. if (strncmp(st.commands[i], command, strlen(command)) == 0) {
89. return i;
90. }
91. }
92. return -1;
93. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. struct Stack {
5. int size;
6. int top;
7. char \*\*procedures;
8. };
9. void create(struct Stack \*st);
10. void push(struct Stack \*st, char \*procedure);
11. char \*pop(struct Stack \*st);
12. void display(struct Stack st);
13. char \*peek(struct Stack st);
14. int search(struct Stack st, char \*procedure);
15. int main() {
16. struct Stack st;
17. create(&st);
18. int choice;
19. char procedure[100];
20. do {
21. printf("\n1. Add Procedure\n2. Remove Last Procedure\n3. View All Procedures\n4. Peek Next Procedure\n5. Search for Procedure\n6. Exit\n");
22. printf("Enter choice: ");
23. scanf("%d", &choice);
24. getchar();  // To consume the newline character after entering the choice
25. switch (choice) {
26. case 1:
27. printf("Enter procedure: ");
28. scanf("%99[^\n]", procedure);  // Read the procedure
29. push(&st, procedure);
30. break;
31. case 2:
32. printf("Removed procedure: %s\n", pop(&st));
33. break;
34. case 3:
35. display(st);
36. break;
37. case 4:
38. printf("Next procedure: %s\n", peek(st));
39. break;
40. case 5:
41. printf("Enter procedure to search: ");
42. scanf("%99[^\n]", procedure);  // Read the procedure to search
43. int index = search(st, procedure);
44. if (index != -1) printf("Found at position %d\n", index + 1);
45. else printf("Procedure not found\n");
46. break;
47. }
48. } while (choice != 6);
49. return 0;
50. }
51. void create(struct Stack \*st) {
52. printf("Enter stack size: ");
53. scanf("%d", &st->size);
54. st->top = -1;
55. st->procedures = (char \*\*)malloc(st->size \* sizeof(char \*));
56. }
57. void push(struct Stack \*st, char \*procedure) {
58. if (st->top < st->size - 1) {
59. st->top++;
60. st->procedures[st->top] = (char \*)malloc((strlen(procedure) + 1) \* sizeof(char));
61. strcpy(st->procedures[st->top], procedure);
62. } else {
63. printf("Stack full\n");
64. }
65. }
66. char \*pop(struct Stack \*st) {
67. if (st->top >= 0) {
68. char \*procedure = st->procedures[st->top];
69. st->top--;
70. return procedure;
71. }
72. return "Stack empty";
73. }
74. void display(struct Stack st) {
75. if (st.top == -1) {
76. printf("No procedures recorded\n");
77. return;
78. }
79. for (int i = st.top; i >= 0; i--) {
80. printf("%s\n", st.procedures[i]);
81. }
82. }
83. char \*peek(struct Stack st) {
84. return st.top >= 0 ? st.procedures[st.top] : "No procedures available";
85. }
86. int search(struct Stack st, char \*procedure) {
87. for (int i = 0; i <= st.top; i++) {
88. if (strncmp(st.procedures[i], procedure, strlen(procedure)) == 0) {
89. return i;
90. }
91. }
92. return -1;
93. }
94. **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
95. #include<stdio.h>
96. #include<stdlib.h>
97. #include<string.h>
98. struct Stack {
99. int size;
100. int top;
101. char \*\*activities;
102. };
103. void create(struct Stack \*st);
104. void push(struct Stack \*st, char \*activity);
105. char \*pop(struct Stack \*st);
106. void display(struct Stack st);
107. char \*peek(struct Stack st);
108. int search(struct Stack st, char \*activity);
109. int main() {
110. struct Stack st;
111. create(&st);
112. int choice;
113. char activity[100];
114. do {
115. printf("\n1. Add Activity\n2. Remove Last Activity\n3. View All Activities\n4. Peek Next Activity\n5. Search for Activity\n6. Exit\n");
116. printf("Enter choice: ");
117. scanf("%d", &choice);
118. getchar();
119. switch (choice) {
120. case 1:
121. printf("Enter activity: ");
122. scanf("%99[^\n]", activity);  // Read the activity
123. push(&st, activity);
124. break;
125. case 2:
126. printf("Removed activity: %s\n", pop(&st));
127. break;
128. case 3:
129. display(st);
130. break;
131. case 4:
132. printf("Next activity: %s\n", peek(st));
133. break;
134. case 5:
135. printf("Enter activity to search: ");
136. scanf("%99[^\n]", activity);  // Read the activity to search
137. int index = search(st, activity);
138. if (index != -1) printf("Found at position %d\n", index + 1);
139. else printf("Activity not found\n");
140. break;
141. }
142. } while (choice != 6);
143. return 0;
144. }
145. void create(struct Stack \*st) {
146. printf("Enter stack size: ");
147. scanf("%d", &st->size);
148. st->top = -1;
149. st->activities = (char \*\*)malloc(st->size \* sizeof(char \*));
150. }
151. void push(struct Stack \*st, char \*activity) {
152. if (st->top < st->size - 1) {
153. st->top++;
154. st->activities[st->top] = (char \*)malloc((strlen(activity) + 1) \* sizeof(char));
155. strcpy(st->activities[st->top], activity);
156. } else {
157. printf("Stack full\n");
158. }
159. }
160. char \*pop(struct Stack \*st) {
161. if (st->top >= 0) {
162. char \*activity = st->activities[st->top];
163. st->top--;
164. return activity;
165. }
166. return "Stack empty";
167. }
168. void display(struct Stack st) {
169. if (st.top == -1) {
170. printf("No activities logged\n");
171. return;
172. }
173. for (int i = st.top; i >= 0; i--) {
174. printf("%s\n", st.activities[i]);
175. }
176. }
177. char \*peek(struct Stack st) {
178. return st.top >= 0 ? st.activities[st.top] : "No activities available";
179. }
180. int search(struct Stack st, char \*activity) {
181. for (int i = 0; i <= st.top; i++) {
182. if (strncmp(st.activities[i], activity, strlen(activity)) == 0) {
183. return i;
184. }
185. }
186. return -1;
187. }

**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

1. #include<stdio.h>
2. #include<stdlib.h>
3. struct Stack {
4. int size;
5. int top;
6. float \*fuelUsage;
7. };
8. void create(struct Stack \*st);
9. void push(struct Stack \*st, float fuel);
10. float pop(struct Stack \*st);
11. void display(struct Stack st);
12. float peek(struct Stack st);
13. int search(struct Stack st, float fuel);
14. int main() {
15. struct Stack st;
16. create(&st);
17. int choice;
18. float fuel;
19. do {
20. printf("\n1. Add Fuel Usage Entry\n2. Remove Last Entry\n3. View All Fuel Usage\n4. Peek Latest Entry\n5. Search for Entry\n6. Exit\n");
21. printf("Enter choice: ");
22. scanf("%d", &choice);
23. switch (choice) {
24. case 1:
25. printf("Enter fuel usage: ");
26. scanf("%f", &fuel);
27. push(&st, fuel);
28. break;
29. case 2:
30. printf("Removed fuel usage: %.2f\n", pop(&st));
31. break;
32. case 3:
33. display(st);
34. break;
35. case 4:
36. printf("Latest fuel usage: %.2f\n", peek(st));
37. break;
38. case 5:
39. printf("Enter fuel usage to search: ");
40. scanf("%f", &fuel);
41. int index = search(st, fuel);
42. if (index != -1) printf("Found at position %d\n", index + 1);
43. else printf("Fuel usage not found\n");
44. break;
45. }
46. } while (choice != 6);
47. return 0;
48. }
49. void create(struct Stack \*st) {
50. printf("Enter stack size: ");
51. scanf("%d", &st->size);
52. st->top = -1;
53. st->fuelUsage = (float \*)malloc(st->size \* sizeof(float));
54. }
55. void push(struct Stack \*st, float fuel) {
56. if (st->top < st->size - 1) {
57. st->top++;
58. st->fuelUsage[st->top] = fuel;
59. } else {
60. printf("Stack full\n");
61. }
62. }
63. float pop(struct Stack \*st) {
64. if (st->top >= 0) {
65. return st->fuelUsage[st->top--];
66. }
67. return -1.0;
68. }
69. void display(struct Stack st) {
70. if (st.top == -1) {
71. printf("No fuel usage data\n");
72. return;
73. }
74. for (int i = st.top; i >= 0; i--) {
75. printf("%.2f\n", st.fuelUsage[i]);
76. }
77. }
78. float peek(struct Stack st) {
79. return st.top >= 0 ? st.fuelUsage[st.top] : -1.0;
80. }
81. int search(struct Stack st, float fuel) {
82. for (int i = 0; i <= st.top; i++) {
83. if (st.fuelUsage[i] == fuel) {
84. return i;
85. }
86. }
87. return -1;
88. }
89. **Order Processing System: Implement a stack-based system using a linked list to manage order processing. Use a switch-case menu with options:**

* 1: Add a new order (push)
* 2: Process the last order (pop)
* 3: Display all pending orders
* 4: Peek at the next order to be processed
* 5: Search for a specific order
* 6: Exit
* #include <stdio.h>
* #include <stdlib.h>
* #include <string.h>
* // Node structure for the stack
* struct Node {
* char order[50];
* struct Node \*next;
* };
* // Function prototypes
* void push(struct Node \*\*top, char \*order);
* char \*pop(struct Node \*\*top);
* void display(struct Node \*top);
* char \*peek(struct Node \*top);
* int search(struct Node \*top, char \*order);
* int main() {
* struct Node \*top = NULL;
* int choice;
* char order[50];
* do {
* printf("\n1: Add a New Order (Push)\n2: Process the Last Order (Pop)\n3: Display All Pending Orders\n4: Peek at the Next Order\n5: Search for a Specific Order\n6: Exit\n");
* printf("Enter your choice: ");
* scanf("%d", &choice);
* switch (choice) {
* case 1:
* printf("Enter order details: ");
* scanf(" %[^\n]s", order);
* push(&top, order);
* break;
* case 2:
* if (top) {
* printf("Processed order: %s\n", pop(&top));
* } else {
* printf("No orders to process\n");
* }
* break;
* case 3:
* display(top);
* break;
* case 4:
* if (top) {
* printf("Next order to process: %s\n", peek(top));
* } else {
* printf("No pending orders\n");
* }
* break;
* case 5:
* printf("Enter order to search: ");
* scanf(" %[^\n]s", order);
* if (search(top, order)) {
* printf("Order found\n");
* } else {
* printf("Order not found\n");
* }
* break;
* case 6:
* printf("Exiting...\n");
* break;
* default:
* printf("Invalid choice. Try again.\n");
* }
* } while (choice != 6);
* return 0;
* }
* // Push operation
* void push(struct Node \*\*top, char \*order) {
* struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
* strcpy(newNode->order, order);
* newNode->next = \*top;
* \*top = newNode;
* }
* // Pop operation
* char \*pop(struct Node \*\*top) {
* static char order[50];
* struct Node \*temp = \*top;
* strcpy(order, temp->order);
* \*top = temp->next;
* free(temp);
* return order;
* }
* // Display operation
* void display(struct Node \*top) {
* if (!top) {
* printf("No pending orders\n");
* return;
* }
* printf("Pending orders:\n");
* while (top) {
* printf("%s\n", top->order);
* top = top->next;
* }
* }
* // Peek operation
* char \*peek(struct Node \*top) {
* if (top) {
* return top->order;
* }
* return "No pending orders";
* }
* // Search operation
* int search(struct Node \*top, char \*order) {
* while (top) {
* if (strcmp(top->order, order) == 0) {
* return 1;  // Order found
* }
* top = top->next;
* }
* return 0;  // Order not found
* }

**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>
* // Node structure for the stack
* struct Node {
* char ticket[100];
* struct Node \*next;
* };
* // Function prototypes
* void push(struct Node \*\*top, char \*ticket);
* char \*pop(struct Node \*\*top);
* void display(struct Node \*top);
* char \*peek(struct Node \*top);
* int search(struct Node \*top, char \*ticket);
* int main() {
* struct Node \*top = NULL;
* int choice;
* char ticket[100];
* do {
* printf("\n1: Add a New Ticket (Push)\n");
* printf("2: Resolve the Latest Ticket (Pop)\n");
* printf("3: View All Pending Tickets\n");
* printf("4: Peek at the Latest Ticket\n");
* printf("5: Search for a Specific Ticket\n");
* printf("6: Exit\n");
* printf("Enter your choice: ");
* scanf("%d", &choice);
* switch (choice) {
* case 1:
* printf("Enter ticket details: ");
* scanf(" %[^\n]s", ticket);
* push(&top, ticket);
* break;
* case 2:
* if (top) {
* printf("Resolved ticket: %s\n", pop(&top));
* } else {
* printf("No tickets to resolve.\n");
* }
* break;
* case 3:
* display(top);
* break;
* case 4:
* if (top) {
* printf("Latest ticket: %s\n", peek(top));
* } else {
* printf("No pending tickets.\n");
* }
* break;
* case 5:
* printf("Enter ticket to search: ");
* scanf(" %[^\n]s", ticket);
* if (search(top, ticket)) {
* printf("Ticket found.\n");
* } else {
* printf("Ticket not found.\n");
* }
* break;
* case 6:
* printf("Exiting...\n");
* break;
* default:
* printf("Invalid choice. Please try again.\n");
* }
* } while (choice != 6);
* return 0;
* }
* // Push operation
* void push(struct Node \*\*top, char \*ticket) {
* struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
* strcpy(newNode->ticket, ticket);
* newNode->next = \*top;
* \*top = newNode;
* }
* // Pop operation
* char \*pop(struct Node \*\*top) {
* static char ticket[100];
* struct Node \*temp = \*top;
* strcpy(ticket, temp->ticket);
* \*top = temp->next;
* free(temp);
* return ticket;
* }
* // Display operation
* void display(struct Node \*top) {
* if (!top) {
* printf("No pending tickets.\n");
* return;
* }
* printf("Pending tickets:\n");
* while (top) {
* printf("%s\n", top->ticket);
* top = top->next;
* }
* }
* // Peek operation
* char \*peek(struct Node \*top) {
* if (top) {
* return top->ticket;
* }
* return "No pending tickets.";
* }
* // Search operation
* int search(struct Node \*top, char \*ticket) {
* while (top) {
* if (strcmp(top->ticket, ticket) == 0) {
* return 1; // Ticket found
* }
* top = top->next;
* }
* return 0; // Ticket not found
* }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char returnRequest[100];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*request);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*request);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char request[100];
25. do {
26. printf("\n1: Add a New Return Request (Push)\n");
27. printf("2: Process the Last Return (Pop)\n");
28. printf("3: Display All Return Requests\n");
29. printf("4: Peek at the Next Return to Process\n");
30. printf("5: Search for a Specific Return Request\n");
31. printf("6: Exit\n");
32. printf("Enter your choice: ");
33. scanf("%d", &choice);
34. switch (choice) {
35. case 1:
36. printf("Enter return request details: ");
37. scanf(" %[^\n]s", request); // Reads input including spaces
38. push(&top, request);
39. break;
40. case 2:
41. if (top) {
42. printf("Processed return request: %s\n", pop(&top));
43. } else {
44. printf("No return requests to process.\n");
45. }
46. break;
47. case 3:
48. display(top);
49. break;
50. case 4:
51. if (top) {
52. printf("Next return to process: %s\n", peek(top));
53. } else {
54. printf("No return requests pending.\n");
55. }
56. break;
57. case 5:
58. printf("Enter return request to search: ");
59. scanf(" %[^\n]s", request);
60. if (search(top, request)) {
61. printf("Return request found.\n");
62. } else {
63. printf("Return request not found.\n");
64. }
65. break;
66. case 6:
67. printf("Exiting...\n");
68. break;
69. default:
70. printf("Invalid choice. Please try again.\n");
71. }
72. } while (choice != 6);
73. return 0;
74. }
75. // Push operation
76. void push(struct Node \*\*top, char \*request) {
77. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
78. strcpy(newNode->returnRequest, request);
79. newNode->next = \*top;
80. \*top = newNode;
81. }
82. // Pop operation
83. char \*pop(struct Node \*\*top) {
84. static char request[100];
85. struct Node \*temp = \*top;
86. strcpy(request, temp->returnRequest);
87. \*top = temp->next;
88. free(temp);
89. return request;
90. }
91. // Display operation
92. void display(struct Node \*top) {
93. if (!top) {
94. printf("No return requests.\n");
95. return;
96. }
97. printf("Pending return requests:\n");
98. while (top) {
99. printf("%s\n", top->returnRequest);
100. top = top->next;
101. }
102. }
103. // Peek operation
104. char \*peek(struct Node \*top) {
105. if (top) {
106. return top->returnRequest;
107. }
108. return "No return requests.";
109. }
110. // Search operation
111. int search(struct Node \*top, char \*request) {
112. while (top) {
113. if (strcmp(top->returnRequest, request) == 0) {
114. return 1; // Request found
115. }
116. top = top->next;
117. }
118. return 0; // Request not found
119. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char restockEntry[100];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*entry);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*entry);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char entry[100];
25. do {
26. printf("\n1: Add a Restock Entry (Push)\n");
27. printf("2: Process the Last Restock (Pop)\n");
28. printf("3: View All Restock Entries\n");
29. printf("4: Peek at the Latest Restock Entry\n");
30. printf("5: Search for a Specific Restock Entry\n");
31. printf("6: Exit\n");
32. printf("Enter your choice: ");
33. scanf("%d", &choice);
34. switch (choice) {
35. case 1:
36. printf("Enter restock entry details: ");
37. scanf(" %[^\n]s", entry); // Reads input including spaces
38. push(&top, entry);
39. break;
40. case 2:
41. if (top) {
42. printf("Processed restock entry: %s\n", pop(&top));
43. } else {
44. printf("No restock entries to process.\n");
45. }
46. break;
47. case 3:
48. display(top);
49. break;
50. case 4:
51. if (top) {
52. printf("Latest restock entry: %s\n", peek(top));
53. } else {
54. printf("No restock entries available.\n");
55. }
56. break;
57. case 5:
58. printf("Enter restock entry to search: ");
59. scanf(" %[^\n]s", entry);
60. if (search(top, entry)) {
61. printf("Restock entry found.\n");
62. } else {
63. printf("Restock entry not found.\n");
64. }
65. break;
66. case 6:
67. printf("Exiting...\n");
68. break;
69. default:
70. printf("Invalid choice. Please try again.\n");
71. }
72. } while (choice != 6);
73. return 0;
74. }
75. // Push operation
76. void push(struct Node \*\*top, char \*entry) {
77. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
78. strcpy(newNode->restockEntry, entry);
79. newNode->next = \*top;
80. \*top = newNode;
81. }
82. // Pop operation
83. char \*pop(struct Node \*\*top) {
84. static char entry[100];
85. struct Node \*temp = \*top;
86. strcpy(entry, temp->restockEntry);
87. \*top = temp->next;
88. free(temp);
89. return entry;
90. }
91. // Display operation
92. void display(struct Node \*top) {
93. if (!top) {
94. printf("No restock entries.\n");
95. return;
96. }
97. printf("Restock entries:\n");
98. while (top) {
99. printf("%s\n", top->restockEntry);
100. top = top->next;
101. }
102. }
103. // Peek operation
104. char \*peek(struct Node \*top) {
105. if (top) {
106. return top->restockEntry;
107. }
108. return "No restock entries.";
109. }
110. // Search operation
111. int search(struct Node \*top, char \*entry) {
112. while (top) {
113. if (strcmp(top->restockEntry, entry) == 0) {
114. return 1; // Entry found
115. }
116. top = top->next;
117. }
118. return 0; // Entry not found
119. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char deal[100];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*deal);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*deal);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char deal[100];
25. do {
26. printf("\n1: Add a New Deal (Push)\n");
27. printf("2: Remove the Last Deal (Pop)\n");
28. printf("3: View All Active Deals\n");
29. printf("4: Peek at the Latest Deal\n");
30. printf("5: Search for a Specific Deal\n");
31. printf("6: Exit\n");
32. printf("Enter your choice: ");
33. scanf("%d", &choice);
34. switch (choice) {
35. case 1:
36. printf("Enter deal details: ");
37. scanf(" %[^\n]s", deal); // Reads input including spaces
38. push(&top, deal);
39. break;
40. case 2:
41. if (top) {
42. printf("Removed deal: %s\n", pop(&top));
43. } else {
44. printf("No active deals to remove.\n");
45. }
46. break;
47. case 3:
48. display(top);
49. break;
50. case 4:
51. if (top) {
52. printf("Latest deal: %s\n", peek(top));
53. } else {
54. printf("No active deals available.\n");
55. }
56. break;
57. case 5:
58. printf("Enter deal to search: ");
59. scanf(" %[^\n]s", deal);
60. if (search(top, deal)) {
61. printf("Deal found.\n");
62. } else {
63. printf("Deal not found.\n");
64. }
65. break;
66. case 6:
67. printf("Exiting...\n");
68. break;
69. default:
70. printf("Invalid choice. Please try again.\n");
71. }
72. } while (choice != 6);
73. return 0;
74. }
75. // Push operation
76. void push(struct Node \*\*top, char \*deal) {
77. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
78. strcpy(newNode->deal, deal);
79. newNode->next = \*top;
80. \*top = newNode;
81. }
82. // Pop operation
83. char \*pop(struct Node \*\*top) {
84. static char deal[100];
85. struct Node \*temp = \*top;
86. strcpy(deal, temp->deal);
87. \*top = temp->next;
88. free(temp);
89. return deal;
90. }
91. // Display operation
92. void display(struct Node \*top) {
93. if (!top) {
94. printf("No active deals.\n");
95. return;
96. }
97. printf("Active deals:\n");
98. while (top) {
99. printf("%s\n", top->deal);
100. top = top->next;
101. }
102. }
103. // Peek operation
104. char \*peek(struct Node \*top) {
105. if (top) {
106. return top->deal;
107. }
108. return "No active deals.";
109. }
110. // Search operation
111. int search(struct Node \*top, char \*deal) {
112. while (top) {
113. if (strcmp(top->deal, deal) == 0) {
114. return 1; // Deal found
115. }
116. top = top->next;
117. }
118. return 0; // Deal not found
119. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char session[100];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*session);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*session);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char session[100];
25. do {
26. printf("\n1: Add a session (Push)\n");
27. printf("2: End the last session (Pop)\n");
28. printf("3: Display all sessions\n");
29. printf("4: Peek at the most recent session\n");
30. printf("5: Search for a specific session\n");
31. printf("6: Exit\n");
32. printf("Enter your choice: ");
33. scanf("%d", &choice);
34. switch (choice) {
35. case 1:
36. printf("Enter session name: ");
37. scanf(" %[^\n]s", session); // Read session name with spaces
38. push(&top, session);
39. break;
40. case 2:
41. if (top) {
42. printf("Ended session: %s\n", pop(&top));
43. } else {
44. printf("No active sessions to end.\n");
45. }
46. break;
47. case 3:
48. display(top);
49. break;
50. case 4:
51. if (top) {
52. printf("Most recent session: %s\n", peek(top));
53. } else {
54. printf("No active sessions.\n");
55. }
56. break;
57. case 5:
58. printf("Enter session name to search: ");
59. scanf(" %[^\n]s", session);
60. if (search(top, session)) {
61. printf("Session found.\n");
62. } else {
63. printf("Session not found.\n");
64. }
65. break;
66. case 6:
67. printf("Exiting...\n");
68. break;
69. default:
70. printf("Invalid choice. Please try again.\n");
71. }
72. } while (choice != 6);
73. return 0;
74. }
75. // Push operation
76. void push(struct Node \*\*top, char \*session) {
77. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
78. strcpy(newNode->session, session);
79. newNode->next = \*top;
80. \*top = newNode;
81. }
82. // Pop operation
83. char \*pop(struct Node \*\*top) {
84. static char session[100];
85. struct Node \*temp = \*top;
86. strcpy(session, temp->session);
87. \*top = temp->next;
88. free(temp);
89. return session;
90. }
91. // Display operation
92. void display(struct Node \*top) {
93. if (!top) {
94. printf("No active sessions.\n");
95. return;
96. }
97. printf("Active sessions:\n");
98. while (top) {
99. printf("%s\n", top->session);
100. top = top->next;
101. }
102. }
103. // Peek operation
104. char \*peek(struct Node \*top) {
105. if (top) {
106. return top->session;
107. }
108. return "No active sessions.";
109. }
110. // Search operation
111. int search(struct Node \*top, char \*session) {
112. while (top) {
113. if (strcmp(top->session, session) == 0) {
114. return 1; // Session found
115. }
116. top = top->next;
117. }
118. return 0; // Session not found
119. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char product[100];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*product);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*product);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char product[100];
25. do {
26. printf("\n1: Add a product to wishlist (Push)\n");
27. printf("2: Remove the last added product (Pop)\n");
28. printf("3: View all wishlist items\n");
29. printf("4: Peek at the most recent wishlist item\n");
30. printf("5: Search for a specific product in wishlist\n");
31. printf("6: Exit\n");
32. printf("Enter your choice: ");
33. scanf("%d", &choice);
34. switch (choice) {
35. case 1:
36. printf("Enter product name: ");
37. scanf(" %[^\n]s", product); // Read product name with spaces
38. push(&top, product);
39. break;
40. case 2:
41. if (top) {
42. printf("Removed product: %s\n", pop(&top));
43. } else {
44. printf("Wishlist is empty.\n");
45. }
46. break;
47. case 3:
48. display(top);
49. break;
50. case 4:
51. if (top) {
52. printf("Most recent product: %s\n", peek(top));
53. } else {
54. printf("Wishlist is empty.\n");
55. }
56. break;
57. case 5:
58. printf("Enter product name to search: ");
59. scanf(" %[^\n]s", product);
60. if (search(top, product)) {
61. printf("Product found in wishlist.\n");
62. } else {
63. printf("Product not found in wishlist.\n");
64. }
65. break;
66. case 6:
67. printf("Exiting...\n");
68. break;
69. default:
70. printf("Invalid choice. Please try again.\n");
71. }
72. } while (choice != 6);
73. return 0;
74. }
75. // Push operation
76. void push(struct Node \*\*top, char \*product) {
77. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
78. strcpy(newNode->product, product);
79. newNode->next = \*top;
80. \*top = newNode;
81. }
82. // Pop operation
83. char \*pop(struct Node \*\*top) {
84. static char product[100];
85. struct Node \*temp = \*top;
86. strcpy(product, temp->product);
87. \*top = temp->next;
88. free(temp);
89. return product;
90. }
91. // Display operation
92. void display(struct Node \*top) {
93. if (!top) {
94. printf("Wishlist is empty.\n");
95. return;
96. }
97. printf("Wishlist items:\n");
98. while (top) {
99. printf("%s\n", top->product);
100. top = top->next;
101. }
102. }
103. // Peek operation
104. char \*peek(struct Node \*top) {
105. if (top) {
106. return top->product;
107. }
108. return "Wishlist is empty.";
109. }
110. // Search operation
111. int search(struct Node \*top, char \*product) {
112. while (top) {
113. if (strcmp(top->product, product) == 0) {
114. return 1; // Product found
115. }
116. top = top->next;
117. }
118. return 0; // Product not found
119. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char step[100];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*step);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*step);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char step[100];
25. do {
26. printf("\n1: Add a checkout step (Push)\n");
27. printf("2: Remove the last step (Pop)\n");
28. printf("3: Display all checkout steps\n");
29. printf("4: Peek at the current step\n");
30. printf("5: Search for a specific step\n");
31. printf("6: Exit\n");
32. printf("Enter your choice: ");
33. scanf("%d", &choice);
34. switch (choice) {
35. case 1:
36. printf("Enter checkout step: ");
37. scanf(" %[^\n]s", step); // Read step with spaces
38. push(&top, step);
39. break;
40. case 2:
41. if (top) {
42. printf("Removed step: %s\n", pop(&top));
43. } else {
44. printf("No steps to remove.\n");
45. }
46. break;
47. case 3:
48. display(top);
49. break;
50. case 4:
51. if (top) {
52. printf("Current step: %s\n", peek(top));
53. } else {
54. printf("No steps available.\n");
55. }
56. break;
57. case 5:
58. printf("Enter step to search: ");
59. scanf(" %[^\n]s", step);
60. if (search(top, step)) {
61. printf("Step found.\n");
62. } else {
63. printf("Step not found.\n");
64. }
65. break;
66. case 6:
67. printf("Exiting...\n");
68. break;
69. default:
70. printf("Invalid choice. Please try again.\n");
71. }
72. } while (choice != 6);
73. return 0;
74. }
75. // Push operation
76. void push(struct Node \*\*top, char \*step) {
77. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
78. strcpy(newNode->step, step);
79. newNode->next = \*top;
80. \*top = newNode;
81. }
82. // Pop operation
83. char \*pop(struct Node \*\*top) {
84. static char step[100];
85. struct Node \*temp = \*top;
86. strcpy(step, temp->step);
87. \*top = temp->next;
88. free(temp);
89. return step;
90. }
91. // Display operation
92. void display(struct Node \*top) {
93. if (!top) {
94. printf("No checkout steps.\n");
95. return;
96. }
97. printf("Checkout steps:\n");
98. while (top) {
99. printf("%s\n", top->step);
100. top = top->next;
101. }
102. }
103. // Peek operation
104. char \*peek(struct Node \*top) {
105. if (top) {
106. return top->step;
107. }
108. return "No steps available.";
109. }
110. // Search operation
111. int search(struct Node \*top, char \*step) {
112. while (top) {
113. if (strcmp(top->step, step) == 0) {
114. return 1; // Step found
115. }
116. top = top->next;
117. }
118. return 0; // Step not found
119. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char coupon[50];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*coupon);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*coupon);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char coupon[50];
25. do {
26. printf("\nCoupon Code Management Menu:\n");
27. printf("1: Add a new coupon code (Push)\n");
28. printf("2: Remove the last coupon code (Pop)\n");
29. printf("3: View all available coupon codes\n");
30. printf("4: Peek at the latest coupon code\n");
31. printf("5: Search for a specific coupon code\n");
32. printf("6: Exit\n");
33. printf("Enter your choice: ");
34. scanf("%d", &choice);
35. switch (choice) {
36. case 1:
37. printf("Enter coupon code: ");
38. scanf(" %[^\n]s", coupon); // Read coupon code with spaces
39. push(&top, coupon);
40. break;
41. case 2:
42. if (top) {
43. printf("Removed coupon code: %s\n", pop(&top));
44. } else {
45. printf("No coupon codes to remove.\n");
46. }
47. break;
48. case 3:
49. display(top);
50. break;
51. case 4:
52. if (top) {
53. printf("Latest coupon code: %s\n", peek(top));
54. } else {
55. printf("No coupon codes available.\n");
56. }
57. break;
58. case 5:
59. printf("Enter coupon code to search: ");
60. scanf(" %[^\n]s", coupon);
61. if (search(top, coupon)) {
62. printf("Coupon code found.\n");
63. } else {
64. printf("Coupon code not found.\n");
65. }
66. break;
67. case 6:
68. printf("Exiting...\n");
69. break;
70. default:
71. printf("Invalid choice. Please try again.\n");
72. }
73. } while (choice != 6);
74. return 0;
75. }
76. // Push operation
77. void push(struct Node \*\*top, char \*coupon) {
78. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
79. strcpy(newNode->coupon, coupon);
80. newNode->next = \*top;
81. \*top = newNode;
82. }
83. // Pop operation
84. char \*pop(struct Node \*\*top) {
85. static char coupon[50];
86. struct Node \*temp = \*top;
87. strcpy(coupon, temp->coupon);
88. \*top = temp->next;
89. free(temp);
90. return coupon;
91. }
92. // Display operation
93. void display(struct Node \*top) {
94. if (!top) {
95. printf("No coupon codes available.\n");
96. return;
97. }
98. printf("Available coupon codes:\n");
99. while (top) {
100. printf("%s\n", top->coupon);
101. top = top->next;
102. }
103. }
104. // Peek operation
105. char \*peek(struct Node \*top) {
106. if (top) {
107. return top->coupon;
108. }
109. return "No coupon codes available.";
110. }
111. // Search operation
112. int search(struct Node \*top, char \*coupon) {
113. while (top) {
114. if (strcmp(top->coupon, coupon) == 0) {
115. return 1; // Coupon code found
116. }
117. top = top->next;
118. }
119. return 0; // Coupon code not found
120. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char status[100];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*status);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*status);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char status[100];
25. do {
26. printf("\nShipping Status Tracker Menu:\n");
27. printf("1: Add a shipping status update (Push)\n");
28. printf("2: Remove the last update (Pop)\n");
29. printf("3: View all shipping status updates\n");
30. printf("4: Peek at the latest update\n");
31. printf("5: Search for a specific update\n");
32. printf("6: Exit\n");
33. printf("Enter your choice: ");
34. scanf("%d", &choice);
35. switch (choice) {
36. case 1:
37. printf("Enter shipping status update: ");
38. scanf(" %[^\n]s", status); // Read status with spaces
39. push(&top, status);
40. break;
41. case 2:
42. if (top) {
43. printf("Removed update: %s\n", pop(&top));
44. } else {
45. printf("No shipping status updates to remove.\n");
46. }
47. break;
48. case 3:
49. display(top);
50. break;
51. case 4:
52. if (top) {
53. printf("Latest shipping status: %s\n", peek(top));
54. } else {
55. printf("No updates available.\n");
56. }
57. break;
58. case 5:
59. printf("Enter status to search: ");
60. scanf(" %[^\n]s", status);
61. if (search(top, status)) {
62. printf("Status update found.\n");
63. } else {
64. printf("Status update not found.\n");
65. }
66. break;
67. case 6:
68. printf("Exiting...\n");
69. break;
70. default:
71. printf("Invalid choice. Please try again.\n");
72. }
73. } while (choice != 6);
74. return 0;
75. }
76. // Push operation
77. void push(struct Node \*\*top, char \*status) {
78. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
79. strcpy(newNode->status, status);
80. newNode->next = \*top;
81. \*top = newNode;
82. }
83. // Pop operation
84. char \*pop(struct Node \*\*top) {
85. static char status[100];
86. struct Node \*temp = \*top;
87. strcpy(status, temp->status);
88. \*top = temp->next;
89. free(temp);
90. return status;
91. }
92. // Display operation
93. void display(struct Node \*top) {
94. if (!top) {
95. printf("No shipping status updates available.\n");
96. return;
97. }
98. printf("Shipping status updates:\n");
99. while (top) {
100. printf("%s\n", top->status);
101. top = top->next;
102. }
103. }
104. // Peek operation
105. char \*peek(struct Node \*top) {
106. if (top) {
107. return top->status;
108. }
109. return "No shipping status updates available.";
110. }
111. // Search operation
112. int search(struct Node \*top, char \*status) {
113. while (top) {
114. if (strcmp(top->status, status) == 0) {
115. return 1; // Status update found
116. }
117. top = top->next;
118. }
119. return 0; // Status update not found
120. **}**

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char review[200];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*review);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*review);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char review[200];
25. do {
26. printf("\nUser Review Management Menu:\n");
27. printf("1: Add a new review (Push)\n");
28. printf("2: Remove the last review (Pop)\n");
29. printf("3: Display all reviews\n");
30. printf("4: Peek at the latest review\n");
31. printf("5: Search for a specific review\n");
32. printf("6: Exit\n");
33. printf("Enter your choice: ");
34. scanf("%d", &choice);
35. switch (choice) {
36. case 1:
37. printf("Enter review: ");
38. scanf(" %[^\n]s", review); // Read review with spaces
39. push(&top, review);
40. break;
41. case 2:
42. if (top) {
43. printf("Removed review: %s\n", pop(&top));
44. } else {
45. printf("No reviews to remove.\n");
46. }
47. break;
48. case 3:
49. display(top);
50. break;
51. case 4:
52. if (top) {
53. printf("Latest review: %s\n", peek(top));
54. } else {
55. printf("No reviews available.\n");
56. }
57. break;
58. case 5:
59. printf("Enter review to search: ");
60. scanf(" %[^\n]s", review);
61. if (search(top, review)) {
62. printf("Review found.\n");
63. } else {
64. printf("Review not found.\n");
65. }
66. break;
67. case 6:
68. printf("Exiting...\n");
69. break;
70. default:
71. printf("Invalid choice. Please try again.\n");
72. }
73. } while (choice != 6);
74. return 0;
75. }
76. // Push operation
77. void push(struct Node \*\*top, char \*review) {
78. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
79. strcpy(newNode->review, review);
80. newNode->next = \*top;
81. \*top = newNode;
82. }
83. // Pop operation
84. char \*pop(struct Node \*\*top) {
85. static char review[200];
86. struct Node \*temp = \*top;
87. strcpy(review, temp->review);
88. \*top = temp->next;
89. free(temp);
90. return review;
91. }
92. // Display operation
93. void display(struct Node \*top) {
94. if (!top) {
95. printf("No reviews available.\n");
96. return;
97. }
98. printf("User reviews:\n");
99. while (top) {
100. printf("%s\n", top->review);
101. top = top->next;
102. }
103. }
104. // Peek operation
105. char \*peek(struct Node \*top) {
106. if (top) {
107. return top->review;
108. }
109. return "No reviews available.";
110. }
111. // Search operation
112. int search(struct Node \*top, char \*review) {
113. while (top) {
114. if (strcmp(top->review, review) == 0) {
115. return 1; // Review found
116. }
117. top = top->next;
118. }
119. return 0; // Review not found
120. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char notification[200];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*notification);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*notification);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char notification[200];
25. do {
26. printf("\nPromotion Notification System Menu:\n");
27. printf("1: Add a new notification (Push)\n");
28. printf("2: Remove the last notification (Pop)\n");
29. printf("3: View all notifications\n");
30. printf("4: Peek at the latest notification\n");
31. printf("5: Search for a specific notification\n");
32. printf("6: Exit\n");
33. printf("Enter your choice: ");
34. scanf("%d", &choice);
35. switch (choice) {
36. case 1:
37. printf("Enter notification: ");
38. scanf(" %[^\n]s", notification); // Read notification with spaces
39. push(&top, notification);
40. break;
41. case 2:
42. if (top) {
43. printf("Removed notification: %s\n", pop(&top));
44. } else {
45. printf("No notifications to remove.\n");
46. }
47. break;
48. case 3:
49. display(top);
50. break;
51. case 4:
52. if (top) {
53. printf("Latest notification: %s\n", peek(top));
54. } else {
55. printf("No notifications available.\n");
56. }
57. break;
58. case 5:
59. printf("Enter notification to search: ");
60. scanf(" %[^\n]s", notification);
61. if (search(top, notification)) {
62. printf("Notification found.\n");
63. } else {
64. printf("Notification not found.\n");
65. }
66. break;
67. case 6:
68. printf("Exiting...\n");
69. break;
70. default:
71. printf("Invalid choice. Please try again.\n");
72. }
73. } while (choice != 6);
74. return 0;
75. }
76. // Push operation
77. void push(struct Node \*\*top, char \*notification) {
78. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
79. strcpy(newNode->notification, notification);
80. newNode->next = \*top;
81. \*top = newNode;
82. }
83. // Pop operation
84. char \*pop(struct Node \*\*top) {
85. static char notification[200];
86. struct Node \*temp = \*top;
87. strcpy(notification, temp->notification);
88. \*top = temp->next;
89. free(temp);
90. return notification;
91. }
92. // Display operation
93. void display(struct Node \*top) {
94. if (!top) {
95. printf("No notifications available.\n");
96. return;
97. }
98. printf("Promotion notifications:\n");
99. while (top) {
100. printf("%s\n", top->notification);
101. top = top->next;
102. }
103. }
104. // Peek operation
105. char \*peek(struct Node \*top) {
106. if (top) {
107. return top->notification;
108. }
109. return "No notifications available.";
110. }
111. // Search operation
112. int search(struct Node \*top, char \*notification) {
113. while (top) {
114. if (strcmp(top->notification, notification) == 0) {
115. return 1; // Notification found
116. }
117. top = top->next;
118. }
119. return 0; // Notification not found
120. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char product[200];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*product);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*product);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char product[200];
25. do {
26. printf("\nProduct Viewing History Menu:\n");
27. printf("1: Add a product to viewing history (Push)\n");
28. printf("2: Remove the last viewed product (Pop)\n");
29. printf("3: View all viewed products\n");
30. printf("4: Peek at the most recent product viewed\n");
31. printf("5: Search for a specific product\n");
32. printf("6: Exit\n");
33. printf("Enter your choice: ");
34. scanf("%d", &choice);
35. switch (choice) {
36. case 1:
37. printf("Enter product name: ");
38. scanf(" %[^\n]s", product); // Read product name with spaces
39. push(&top, product);
40. break;
41. case 2:
42. if (top) {
43. printf("Removed product: %s\n", pop(&top));
44. } else {
45. printf("No products to remove.\n");
46. }
47. break;
48. case 3:
49. display(top);
50. break;
51. case 4:
52. if (top) {
53. printf("Most recent product viewed: %s\n", peek(top));
54. } else {
55. printf("No products viewed.\n");
56. }
57. break;
58. case 5:
59. printf("Enter product name to search: ");
60. scanf(" %[^\n]s", product);
61. if (search(top, product)) {
62. printf("Product found in viewing history.\n");
63. } else {
64. printf("Product not found in viewing history.\n");
65. }
66. break;
67. case 6:
68. printf("Exiting...\n");
69. break;
70. default:
71. printf("Invalid choice. Please try again.\n");
72. }
73. } while (choice != 6);
74. return 0;
75. }
76. // Push operation
77. void push(struct Node \*\*top, char \*product) {
78. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
79. strcpy(newNode->product, product);
80. newNode->next = \*top;
81. \*top = newNode;
82. }
83. // Pop operation
84. char \*pop(struct Node \*\*top) {
85. static char product[200];
86. struct Node \*temp = \*top;
87. strcpy(product, temp->product);
88. \*top = temp->next;
89. free(temp);
90. return product;
91. }
92. // Display operation
93. void display(struct Node \*top) {
94. if (!top) {
95. printf("No products viewed.\n");
96. return;
97. }
98. printf("Viewed products:\n");
99. while (top) {
100. printf("%s\n", top->product);
101. top = top->next;
102. }
103. }
104. // Peek operation
105. char \*peek(struct Node \*top) {
106. if (top) {
107. return top->product;
108. }
109. return "No products viewed.";
110. }
111. // Search operation
112. int search(struct Node \*top, char \*product) {
113. while (top) {
114. if (strcmp(top->product, product) == 0) {
115. return 1; // Product found
116. }
117. top = top->next;
118. }
119. return 0; // Product not found
120. **}**

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char item[200];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*item);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*item);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char item[200];
25. do {
26. printf("\nCart Item Management Menu:\n");
27. printf("1: Add an item to the cart (Push)\n");
28. printf("2: Remove the last item (Pop)\n");
29. printf("3: View all cart items\n");
30. printf("4: Peek at the last added item\n");
31. printf("5: Search for a specific item in the cart\n");
32. printf("6: Exit\n");
33. printf("Enter your choice: ");
34. scanf("%d", &choice);
35. switch (choice) {
36. case 1:
37. printf("Enter item name: ");
38. scanf(" %[^\n]s", item); // Read item name with spaces
39. push(&top, item);
40. break;
41. case 2:
42. if (top) {
43. printf("Removed item: %s\n", pop(&top));
44. } else {
45. printf("No items to remove.\n");
46. }
47. break;
48. case 3:
49. display(top);
50. break;
51. case 4:
52. if (top) {
53. printf("Last added item: %s\n", peek(top));
54. } else {
55. printf("No items in the cart.\n");
56. }
57. break;
58. case 5:
59. printf("Enter item name to search: ");
60. scanf(" %[^\n]s", item);
61. if (search(top, item)) {
62. printf("Item found in the cart.\n");
63. } else {
64. printf("Item not found in the cart.\n");
65. }
66. break;
67. case 6:
68. printf("Exiting...\n");
69. break;
70. default:
71. printf("Invalid choice. Please try again.\n");
72. }
73. } while (choice != 6);
74. return 0;
75. }
76. // Push operation
77. void push(struct Node \*\*top, char \*item) {
78. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
79. strcpy(newNode->item, item);
80. newNode->next = \*top;
81. \*top = newNode;
82. }
83. // Pop operation
84. char \*pop(struct Node \*\*top) {
85. static char item[200];
86. struct Node \*temp = \*top;
87. strcpy(item, temp->item);
88. \*top = temp->next;
89. free(temp);
90. return item;
91. }
92. // Display operation
93. void display(struct Node \*top) {
94. if (!top) {
95. printf("No items in the cart.\n");
96. return;
97. }
98. printf("Cart items:\n");
99. while (top) {
100. printf("%s\n", top->item);
101. top = top->next;
102. }
103. }
104. // Peek operation
105. char \*peek(struct Node \*top) {
106. if (top) {
107. return top->item;
108. }
109. return "No items in the cart.";
110. }
111. // Search operation
112. int search(struct Node \*top, char \*item) {
113. while (top) {
114. if (strcmp(top->item, item) == 0) {
115. return 1; // Item found
116. }
117. top = top->next;
118. }
119. return 0; // Item not found
120. }

**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
7. #include <stdio.h>
8. #include <stdlib.h>
9. #include <string.h>
10. // Node structure for the stack
11. struct Node {
12. char paymentRecord[200];
13. struct Node \*next;
14. };
15. // Function prototypes
16. void push(struct Node \*\*top, char \*paymentRecord);
17. char \*pop(struct Node \*\*top);
18. void display(struct Node \*top);
19. char \*peek(struct Node \*top);
20. int search(struct Node \*top, char \*paymentRecord);
21. int main() {
22. struct Node \*top = NULL;
23. int choice;
24. char paymentRecord[200];
25. do {
26. printf("\nPayment History Menu:\n");
27. printf("1: Add a new payment record (Push)\n");
28. printf("2: Remove the last payment record (Pop)\n");
29. printf("3: View all payment records\n");
30. printf("4: Peek at the latest payment record\n");
31. printf("5: Search for a specific payment record\n");
32. printf("6: Exit\n");
33. printf("Enter your choice: ");
34. scanf("%d", &choice);
35. switch (choice) {
36. case 1:
37. printf("Enter payment record details: ");
38. scanf(" %[^\n]s", paymentRecord); // Read payment record with spaces
39. push(&top, paymentRecord);
40. break;
41. case 2:
42. if (top) {
43. printf("Removed payment record: %s\n", pop(&top));
44. } else {
45. printf("No payment records to remove.\n");
46. }
47. break;
48. case 3:
49. display(top);
50. break;
51. case 4:
52. if (top) {
53. printf("Latest payment record: %s\n", peek(top));
54. } else {
55. printf("No payment records available.\n");
56. }
57. break;
58. case 5:
59. printf("Enter payment record to search: ");
60. scanf(" %[^\n]s", paymentRecord);
61. if (search(top, paymentRecord)) {
62. printf("Payment record found.\n");
63. } else {
64. printf("Payment record not found.\n");
65. }
66. break;
67. case 6:
68. printf("Exiting...\n");
69. break;
70. default:
71. printf("Invalid choice. Please try again.\n");
72. }
73. } while (choice != 6);
74. return 0;
75. }
76. // Push operation
77. void push(struct Node \*\*top, char \*paymentRecord) {
78. struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));
79. strcpy(newNode->paymentRecord, paymentRecord);
80. newNode->next = \*top;
81. \*top = newNode;
82. }
83. // Pop operation
84. char \*pop(struct Node \*\*top) {
85. static char paymentRecord[200];
86. struct Node \*temp = \*top;
87. strcpy(paymentRecord, temp->paymentRecord);
88. \*top = temp->next;
89. free(temp);
90. return paymentRecord;
91. }
92. // Display operation
93. void display(struct Node \*top) {
94. if (!top) {
95. printf("No payment records available.\n");
96. return;
97. }
98. printf("Payment records:\n");
99. while (top) {
100. printf("%s\n", top->paymentRecord);
101. top = top->next;
102. }
103. }
104. // Peek operation
105. char \*peek(struct Node \*top) {
106. if (top) {
107. return top->paymentRecord;
108. }
109. return "No payment records available.";
110. }
111. // Search operation
112. int search(struct Node \*top, char \*paymentRecord) {
113. while (top) {
114. if (strcmp(top->paymentRecord, paymentRecord) == 0) {
115. return 1; // Payment record found
116. }
117. top = top->next;
118. }
119. return 0; // Payment record not found
120. }