

1 // Write a program in c to search the element using binary search.

2  
3 #include <stdio.h>

4  
5 // Function to perform binary search

6 int binarySearch(int arr[], int size, int key) {  
7 int left = 0;  
8 int right = size - 1;  
9  
10 while (left <= right) {  
11 int mid = left + (right - left) / 2;  
12  
13 if (arr[mid] == key) {  
14 return mid; // Return the index where the key is found  
15 }  
16  
17 if (arr[mid] < key) {  
18 left = mid + 1; // Search in the right half  
19 } else {  
20 right = mid - 1; // Search in the left half  
21 }  
22 }  
23  
24 return -1; // Return -1 if the key is not found  
25 }

26  
27 int main() {  
28 int arr[] = {11, 12, 22, 25, 34, 64, 90};  
29 int size = sizeof(arr) / sizeof(arr[0]);  
30  
31 int key;  
32 printf("Enter the element to search for: ");  
33 scanf("%d", &key);  
34  
35 int result = binarySearch(arr, size, key);  
36  
37 if (result != -1) {  
38 printf("%d found at index %d\n", key, result);  
39 } else {  
40 printf("%d not found in the array\n", key);  
41 }  
42  
43 return 0;  
44 }

45 /\*

46 Output

47 Enter the element to search for: 25

48 25 found at index 3

49 \*/

```

1 // Write a program in c to implement bubble sort.
2 #include <stdio.h>
3 void bubbleSort(int arr[], int size) {
4     int temp;
5     int swapped;
6     for (int i = 0; i < size - 1; i++) {
7         swapped = 0; // Flag to check if any swaps were made in this pass
8
9         for (int j = 0; j < size - 1 - i; j++) {
10             // If the current element is greater than the next element, swap them
11             if (arr[j] > arr[j + 1]) {
12                 temp = arr[j];
13                 arr[j] = arr[j + 1];
14                 arr[j + 1] = temp;
15                 swapped = 1; // Set the swapped flag to 1
16             }
17         }
18
19         // If no two elements were swapped in this pass, the array is already sorted
20         if (swapped == 0) {
21             break;
22         }
23     }
24 }
25
26 int main() {
27     int arr[] = {64, 34, 25, 12, 22, 11, 90};
28     int size = sizeof(arr) / sizeof(arr[0]);
29
30     printf("Original array: ");
31     for (int i = 0; i < size; i++) {
32         printf("%d ", arr[i]);
33     }
34
35     // Perform bubble sort
36     bubbleSort(arr, size);
37
38     printf("\nSorted array: ");
39     for (int i = 0; i < size; i++) {
40         printf("%d ", arr[i]);
41     }
42
43     return 0;
44 }
45 /*
46 Output
47 Original array: 64 34 25 12 22 11 90
48 Sorted array: 11 12 22 25 34 64 90
49 */

```

```

1 // Que:- Write a program in c to create double linked list and display the elements in reverse order.
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 // Define a structure for a node in the doubly linked list
6 struct Node {
7     int data;
8     struct Node* prev;
9     struct Node* next;
10 };
11
12 // Function to insert a new node at the end of the doubly linked list
13 void insertAtEnd(struct Node** head, int data) {
14     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
15     newNode->data = data;
16     newNode->next = NULL;
17
18     if (*head == NULL) {
19         newNode->prev = NULL;
20         *head = newNode;
21     } else {
22         struct Node* current = *head;
23         while (current->next != NULL) {
24             current = current->next;
25         }
26         current->next = newNode;
27         newNode->prev = current;
28     }
29 }
30
31 // Function to display the doubly linked list in reverse order
32 void displayReverse(struct Node* head) {
33     if (head == NULL) {
34         printf("The list is empty.\n");

```

```

35     return;
36 }
37
38 struct Node* current = head;
39 while (current->next != NULL) {
40     current = current->next;
41 }
42
43 printf("Doubly Linked List (in reverse order): ");
44 while (current != NULL) {
45     printf("%d -> ", current->data);
46     current = current->prev;
47 }
48 printf("NULL\n");
49 }
50
51 int main() {
52     struct Node* head = NULL;
53     int n, data;
54
55     printf("Enter the number of elements in the doubly linked list: ");
56     scanf("%d", &n);
57
58     printf("Enter the elements of the doubly linked list:\n");
59     for (int i = 0; i < n; i++) {
60         scanf("%d", &data);
61         insertAtEnd(&head, data);
62     }
63
64     displayReverse(head);
65
66     return 0;
67 }
68
69 /*
70 Output
71 Enter the number of elements in the doubly linked list: 3
72 Enter the elements of the doubly linked list:
73 3
74 2
75 1
76 Doubly Linked List (in reverse order): 1 -> 2 -> 3 -> NULL
77
78 */

```

```

1 // Write a program in c to search the element using sequential/linear search.
2
3 #include <stdio.h>
4
5 // Function to perform sequential (linear) search
6 int linearSearch(int arr[], int size, int key) {
7     for (int i = 0; i < size; i++) {
8         if (arr[i] == key) {
9             return i; // Return the index where the key is found
10        }
11    }
12    return -1; // Return -1 if the key is not found
13 }
14
15 int main() {
16     int arr[] = {64, 34, 25, 12, 22, 11, 90};
17     int size = sizeof(arr) / sizeof(arr[0]);
18
19     int key;
20     printf("Enter the element to search for: ");
21     scanf("%d", &key);
22
23     int result = linearSearch(arr, size, key);
24
25     if (result != -1) {
26         printf("%d found at index %d\n", key, result);
27     } else {
28         printf("%d not found in the array\n", key);
29     }
30
31     return 0;
32 }
33
34
35 /*
36 Output
37 Enter the element to search for: 34
38 34 found at index 1
39 */

```

```

1 // Que:- Write a program in c to implement the concept of Circular Queue
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #define MAX_SIZE 5
6
7 // Structure to represent the Circular Queue
8 struct CircularQueue {
9     int items[MAX_SIZE];
10    int front, rear;
11 };
12
13 // Function to enqueue an element into the Circular Queue
14 void enqueue(struct CircularQueue* queue, int data) {
15     if (((queue->rear + 1) % MAX_SIZE) == queue->front) {
16         printf("Queue is full. Cannot enqueue %d\n", data);
17     } else {
18         if (queue->front == -1) {
19             queue->front = 0;
20         }
21         queue->rear = (queue->rear + 1) % MAX_SIZE;
22         queue->items[queue->rear] = data;
23         printf("%d enqueued to the queue\n", data);
24     }
25 }
26
27 // Function to dequeue an element from the Circular Queue
28 int dequeue(struct CircularQueue* queue) {
29     int data = -1;
30     if (queue->front == -1) {
31         printf("Queue is empty. Cannot dequeue\n");
32     } else {
33         data = queue->items[queue->front];
34         if (queue->front == queue->rear) {

```

```

35         // Queue has only one element, reset front and rear
36         queue->front = -1;
37         queue->rear = -1;
38     } else {
39         queue->front = (queue->front + 1) % MAX_SIZE;
40     }
41 }
42 return data;
43 }
44
45 // Function to display the elements in the Circular Queue
46 void display(struct CircularQueue* queue) {
47     if (queue->front == -1) {
48         printf("Queue is empty\n");
49     } else {
50         int i = queue->front;
51         printf("Queue elements: ");
52         while (1) {
53             printf("%d ", queue->items[i]);
54             if (i == queue->rear) {
55                 break;
56             }
57             i = (i + 1) % MAX_SIZE;
58         }
59         printf("\n");
60     }
61 }
62
63 int main() {
64     struct CircularQueue queue;
65     queue.front = -1;
66     queue.rear = -1;

```



```

68 int choice, data;
69
70 do {
71     printf("\nCircular Queue Menu:\n");
72     printf("1. Enqueue\n");
73     printf("2. Dequeue\n");
74     printf("3. Display\n");
75     printf("4. Quit\n");
76     printf("Enter your choice: ");
77     scanf("%d", &choice);
78
79     switch (choice) {
80         case 1:
81             printf("Enter data to enqueue: ");
82             scanf("%d", &data);
83             enqueue(&queue, data);
84             break;
85         case 2:
86             data = dequeue(&queue);
87             if (data != -1) {
88                 printf("Dequeued element: %d\n", data);
89             }
90             break;
91         case 3:
92             display(&queue);
93             break;
94         case 4:
95             printf("Exiting the program.\n");
96             break;
97         default:
98             printf("Invalid choice. Please try again.\n");
99     }
100 } while (choice != 4);

```



```

102     return 0;
103 }
104 /*
105  Output
106
107  Circular Queue Menu:
108  1. Enqueue
109  2. Dequeue
110  3. Display
111  4. Quit
112  Enter your choice: 1
113  Enter data to enqueue: 1
114  1 enqueued to the queue
115
116  Circular Queue Menu:
117  1. Enqueue
118  2. Dequeue
119  3. Display
120  4. Quit
121  Enter your choice: 3
122  Queue elements: 1
123
124  Circular Queue Menu:
125  1. Enqueue
126  2. Dequeue
127  3. Display
128  4. Quit
129  Enter your choice: 1
130  Enter data to enqueue: 2
131  2 enqueued to the queue
132
133  Circular Queue Menu:
134  1. Enqueue
135  2. Dequeue
136  3. Display
137  4. Quit
138  Enter your choice: 3
139  Queue elements: 1 2
140
141  Circular Queue Menu:
142  1. Enqueue
143  2. Dequeue
144  3. Display
145  4. Quit
146  Enter your choice: 2
147  Dequeued element: 1
148
149  Circular Queue Menu:
150  1. Enqueue
151  2. Dequeue
152  3. Display
153  4. Quit
154  Enter your choice: 3
155  Queue elements: 2
156
157  Circular Queue Menu:
158  1. Enqueue
159  2. Dequeue
160  3. Display
161  4. Quit
162  Enter your choice:

```

```

1 // Write a program in c to implement quick sort.
2
3 #include <stdio.h>
4
5 // Function to swap two elements in an array
6 void swap(int* a, int* b) {
7     int temp = *a;
8     *a = *b;
9     *b = temp;
10 }
11
12 // Function to partition the array and return the pivot index
13 int partition(int arr[], int low, int high) {
14     int pivot = arr[high]; // Choose the last element as the pivot
15     int i = (low - 1); // Index of the smaller element
16
17     for (int j = low; j <= high - 1; j++) {
18         // If the current element is smaller than or equal to the pivot
19         if (arr[j] <= pivot) {
20             i++; // Increment the index of the smaller element
21             swap(&arr[i], &arr[j]);
22         }
23     }
24
25     // Swap the pivot element with the element at (i+1), placing the pivot in its correct position
26     swap(&arr[i + 1], &arr[high]);
27     return (i + 1); // Return the pivot index
28 }
29
30 // Function to perform quick sort
31 void quickSort(int arr[], int low, int high) {
32     if (low < high) {
33         // Find the pivot element such that
34         // elements smaller than the pivot are on the left
35         // elements greater than the pivot are on the right
36         int pi = partition(arr, low, high);
37
38         // Recursively sort elements before and after the pivot
39         quickSort(arr, low, pi - 1);
40         quickSort(arr, pi + 1, high);
41     }
42 }
43
44 int main() {
45     int arr[] = {64, 34, 25, 12, 22, 11, 98};
46     int size = sizeof(arr) / sizeof(arr[0]);
47
48     printf("Original array: ");
49     for (int i = 0; i < size; i++) {
50         printf("%d ", arr[i]);
51     }
52
53     // Perform quick sort
54     quickSort(arr, 0, size - 1);
55
56     printf("\nSorted array: ");
57     for (int i = 0; i < size; i++) {
58         printf("%d ", arr[i]);
59     }
60
61     return 0;
62 }
63 /*
64 Output
65 Original array: 64 34 25 12 22 11 98
66 Sorted array: 11 12 22 25 34 64 98
67 */

```

```

1 // Que:- Write a program in c to search the elements in the linked list and display the position
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 struct Node {
7     int data;
8     struct Node* next;
9 };
10
11 // Function to search for an element in the linked list and return its position
12 int searchElement(struct Node* head, int key) {
13     int position = 1;
14     while (head != NULL) {
15         if (head->data == key) {
16             return position;
17         }
18         head = head->next;
19         position++;
20     }
21     return -1; // Element not found
22 }
23
24 // Function to insert a new node at the beginning of the linked list
25 void insertAtBeginning(struct Node** head, int data) {
26     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
27     newNode->data = data;
28     newNode->next = *head;
29     *head = newNode;
30 }
31
32 int main() {
33     struct Node* head = NULL;
34     int n, key;
35
36     printf("Enter the number of elements in the linked list: ");
37     scanf("%d", &n);
38
39     printf("Enter the elements of the linked list:\n");
40     for (int i = 0; i < n; i++) {
41         int data;
42         scanf("%d", &data);
43         insertAtBeginning(&head, data);
44     }
45
46     printf("Enter the element to search: ");
47     scanf("%d", &key);
48
49     int position = searchElement(head, key);

```

```

38
39     printf("Enter the elements of the linked list:\n");
40     for (int i = 0; i < n; i++) {
41         int data;
42         scanf("%d", &data);
43         insertAtBeginning(&head, data);
44     }
45
46     printf("Enter the element to search: ");
47     scanf("%d", &key);
48
49     int position = searchElement(head, key);
50     if (position != -1) {
51         printf("Element found at position %d\n", position);
52     } else {
53         printf("Element not found in the linked list\n");
54     }
55
56     return 0;
57 }

```

```

58
59 /*

```

```

60 OutPut

```

```

61 Enter the number of elements in the linked list: 8

```

```

62 Enter the elements of the linked list:

```

```

63 1

```

```

64 2

```

```

65 3

```

```

66 5

```

```

67 6

```

```

68 4

```

```

69 9

```

```

70 7

```

```

71 Enter the element to search: 5

```

```

72 Element found at position 5

```

```

73 */

```

stack\_push\_pop.c / m

```
1 // Write a program in c to implement the Stack with Push, Pop and Display operations.
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #define MAX_SIZE 100
6
7 struct Stack {
8     int items[MAX_SIZE];
9     int top;
10 };
11
12 void initialize(struct Stack *stack) {
13     stack->top = -1;
14 }
15
16 int isEmpty(struct Stack *stack) {
17     return (stack->top == -1);
18 }
19
20 int isFull(struct Stack *stack) {
21     return (stack->top == MAX_SIZE - 1);
22 }
23
24 void push(struct Stack *stack, int value) {
25     if (isFull(stack)) {
26         printf("Stack is full. Cannot push %d\n", value);
27     } else {
28         stack->top++;
29         stack->items[stack->top] = value;
30         printf("%d pushed onto the stack\n", value);
31     }
32 }
33
34 int pop(struct Stack *stack) {
35     if (isEmpty(stack)) {
36         printf("Stack is empty. Cannot pop.\n");
37         return -1;
38     } else {
39         int poppedValue = stack->items[stack->top];
40         stack->top--;
41         return poppedValue;
42     }
43 }
```

```

42     }
43 }
44
45 void display(struct Stack *stack) {
46     if (isEmpty(stack)) {
47         printf("Stack is empty.\n");
48     } else {
49         printf("Stack elements: ");
50         for (int i = 0; i <= stack->top; i++) {
51             printf("%d ", stack->items[i]);
52         }
53         printf("\n");
54     }
55 }
56
57 int main() {
58     struct Stack stack;
59     initialize(&stack);
60
61     int choice, value;
62
63     while (1) {
64         printf("\nStack Menu:\n");
65         printf("1. Push\n");
66         printf("2. Pop\n");
67         printf("3. Display\n");
68         printf("4. Exit\n");
69         printf("Enter your choice: ");
70         scanf("%d", &choice);
71
72         switch (choice) {
73             case 1:
74                 printf("Enter the value to push: ");
75                 scanf("%d", &value);
76                 push(&stack, value);
77                 break;
78             case 2:
79                 printf("Popped element: %d\n", pop(&stack));
80                 break;
81             case 3:
82                 display(&stack);

```



```

82         display(&stack);
83         break;
84     case 4:
85         exit(0);
86     default:
87         printf("Invalid choice. Please try again.\n");
88     }
89 }
90
91 return 0;
92 }
93
94 /*
95 OutPut
96
97 Stack Menu:
98 1. Push
99 2. Pop
100 3. Display
101 4. Exit
102 Enter your choice: 3
103 Stack is empty.
104
105 Stack Menu:
106 1. Push
107 2. Pop
108 3. Display
109 4. Exit
110 Enter your choice: 1
111 Enter the value to push: 1
112 1 pushed onto the stack
113
114 Stack Menu:
115 1. Push
116 2. Pop
117 3. Display
118 4. Exit
119 Enter your choice: 1
120 Enter the value to push: 2
121 2 pushed onto the stack
122
123 Stack Menu:
124 1. Push
125 2. Pop
126 3. Display
127 4. Exit
128 Enter your choice: 1
129 Enter the value to push: 3
130 3 pushed onto the stack
131
132 Stack Menu:
133 1. Push
134 2. Pop
135 3. Display
136 4. Exit
137 Enter your choice: 3
138 Stack elements: 1 2 3

```

```

1 // Write a program in c for in order, post order and preorder traversal of tree
2
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 // Define the structure for a binary tree node
7 struct TreeNode {
8     int data;
9     struct TreeNode* left;
10    struct TreeNode* right;
11 };
12
13 // Function to create a new binary tree node
14 struct TreeNode* createNode(int data) {
15     struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
16     if (newNode == NULL) {
17         printf("Memory allocation failed\n");
18         exit(1);
19     }
20     newNode->data = data;
21     newNode->left = NULL;
22     newNode->right = NULL;
23     return newNode;
24 }
25
26 // Function for in-order traversal of a binary tree
27 void inorderTraversal(struct TreeNode* root) {
28     if (root == NULL) {
29         return;
30     }
31     inorderTraversal(root->left);
32     printf("%d ", root->data);
33     inorderTraversal(root->right);
34 }
35
36 // Function for post-order traversal of a binary tree
37 void postorderTraversal(struct TreeNode* root) {
38     if (root == NULL) {
39         return;
40     }
41     postorderTraversal(root->left);
42     postorderTraversal(root->right);
43     printf("%d ", root->data);
44 }
45
46 // Function for pre-order traversal of a binary tree
47 void preorderTraversal(struct TreeNode* root) {
48     if (root == NULL) {
49         return;
50     }
51     printf("%d ", root->data);
52     preorderTraversal(root->left);
53     preorderTraversal(root->right);
54 }
55
56 int main() {
57     // Create a sample binary tree
58     struct TreeNode* root = createNode(1);
59     root->left = createNode(2);
60     root->right = createNode(3);
61     root->left->left = createNode(4);
62     root->left->right = createNode(5);
63
64     printf("In-order traversal: ");
65     inorderTraversal(root);
66     printf("\n");
67
68     printf("Post-order traversal: ");
69     postorderTraversal(root);
70     printf("\n");
71
72     printf("Pre-order traversal: ");
73     preorderTraversal(root);
74     printf("\n");
75
76     return 0;
77 }
78 /*
79 Output
80
81 In-order traversal: 4 2 5 1 3
82 Post-order traversal: 4 5 2 3 1
83 Pre-order traversal: 1 2 4 5 3
84
85 */

```

```
1 //Que:- Write a program in c to read two arrays from the user and merge them and display the elements
2 #include <stdio.h>
3
4 int main() {
5     int arr1[100], arr2[100], mergeArr[200];
6     int size1, size2, mergedSize;
7
8     // Input for the first array
9     printf("Enter the size of the first array: ");
10    scanf("%d", &size1);
11
12    printf("Enter elements of the first array:\n");
13    for (int i = 0; i < size1; i++) {
14        scanf("%d", &arr1[i]);
15    }
16
17    // Input for the second array
18    printf("Enter the size of the second array: ");
19    scanf("%d", &size2);
20
21    printf("Enter elements of the second array:\n");
22    for (int i = 0; i < size2; i++) {
23        scanf("%d", &arr2[i]);
24    }
25
26    // Merging the two arrays
27    mergedSize = size1 + size2;
28    for (int i = 0; i < size1; i++) {
29        mergeArr[i] = arr1[i];
30    }
31    for (int i = 0; i < size2; i++) {
32        mergeArr[size1 + i] = arr2[i];
33    }
34
35    // Displaying the merged array
36    printf("Merged array: ");
37    for (int i = 0; i < mergedSize; i++) {
38        printf("%d ", mergeArr[i]);
39    }
40    printf("\n");
```

```
36     printf("Merged array: ");
37     for (int i = 0; i < mergedSize; i++) {
38         printf("%d ", mergeArr[i]);
39     }
40     printf("\n");
41
42     return 0;
43 }
44
45 // output
46 /*
47 Enter the size of the first array: 3
48 Enter elements of the first array:
49 1
50 2
51 3
52 Enter the size of the second array: 3
53 Enter elements of the second array:
54 9
55 8
56 7
57 Merged array: 1 2 3 9 8 7
58
59 */
```