

## 1) Write a c program for linked list.

```
Programiz C Online Compiler
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node { int data; struct Node* next; };
4 struct Node* insert(struct Node* h, int d) {
5     struct Node* n = malloc(sizeof(struct Node));
6     n->data = d; n->next = h; return n;
7 }
8 struct Node* delete(struct Node* h, int d) {
9     struct Node *c = h, *p = NULL;
10    if (c && c->data == d) {
11        h = c->next; free(c); return h;
12    }
13    while (c && c->data != d) p = c, c = c->next;
14    if (c) p->next = c->next; free(c); return h;
15 }
16 int search(struct Node* h, int d) {
17     while (h && h->data != d) h = h->next; return h ? 1 : 0;
18 }
19 void display(struct Node* h) {
20     while (h) printf("%d -> ", h->data), h = h->next; printf("NULL\n");
21 }
22 int main() {
23     struct Node* h = NULL;
24     h = insert(h, 4); h = insert(h, 3);
25     h = insert(h, 2); h = insert(h, 1);
26     printf("Linked List: "); display(h);
27     h = delete(h, 3); printf("After deleting 3: ");
28     display(h); printf(search(h, 2) ? "2 found.\n" : "2 not found.\n");
29     return 0;
30 }
```

```
/tmp/nPJJoDPbuR.o
Linked List: 1 -> 2 -> 3 -> 4 -> NULL
After deleting 3: 1 -> 2 -> 4 -> NULL
2 found.

=== Code Execution Successful ===
```

## 2) Write a c program for single linked list.

```
Programiz C Online Compiler
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node { int data; struct Node* next; };
4 struct Node* insert(struct Node* h, int d) {
5     struct Node* n = malloc(sizeof(struct Node));
6     n->data = d; n->next = h; return n;
7 }
8 struct Node* delete(struct Node* h, int d) {
9     struct Node *c = h, *p = NULL;
10    if (c && c->data == d) {
11        h = c->next; free(c); return h;
12    }
13    while (c && c->data != d) p = c, c = c->next;
14    if (c) p->next = c->next; free(c); return h;
15 }
16 int search(struct Node* h, int d) {
17     while (h && h->data != d) h = h->next; return h ? 1 : 0;
18 }
19 void display(struct Node* h) {
20     while (h) printf("%d -> ", h->data), h = h->next; printf("NULL\n");
21 }
22 int main() {
23     struct Node* h = NULL;
24     h = insert(h, 4); h = insert(h, 3);
25     h = insert(h, 2); h = insert(h, 1);
26     printf("Linked List: "); display(h);
27     h = delete(h, 3); printf("After deleting 3: ");
28     display(h); printf(search(h, 2) ? "2 found.\n" : "2 not found.\n");
29     return 0;
30 }
```

```
/tmp/nPJJoDPbuR.o
Linked List: 1 -> 2 -> 3 -> 4 -> NULL
After deleting 3: 1 -> 2 -> 4 -> NULL
2 found.

=== Code Execution Successful ===
```

## 3) Write a c program for double linked list.

C Online Compiler

main.c

Share

Run

Output

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node { int data; struct Node *p, *n; };
4 struct Node* insert(struct Node* h, int d) {
5     struct Node* n = malloc(sizeof(struct Node));
6     n->data = d; n->p = NULL; n->n = h;
7     if (h) h->p = n; return n;
8 }
9 struct Node* delete(struct Node* h, int d) {
10    struct Node *c = h;
11    while (c && c->data != d) c = c->n;
12    if (c->p) c->p->n = c->n;
13    if (c->n) c->n->p = c->p;
14    if (c == h) h = c->n;
15    free(c); return h;
16 }
17 int search(struct Node* h, int d) {
18    while (h && h->data != d) h = h->n; return h ? 1 : 0;
19 }
20 void display(struct Node* h) {
21    while (h) printf("%d -> ", h->data), h = h->n; printf("NULL\n");
22 }
23 int main() {
24    struct Node* h = NULL;
25    h = insert(h, 4); h = insert(h, 3);
26    h = insert(h, 2); h = insert(h, 1);
27    printf("Doubly Linked List: "); display(h);
28    h = delete(h, 3); printf("After deleting 3: ");
29    display(h); printf(search(h, 2) ? "2 found.\n" : "2 not found.\n");
30    return 0;

```

```

/tmp/WowHcOPcjg.o
Doubly Linked List: 1 -> 2 -> 3 -> 4 -> NULL
After deleting 3: 1 -> 2 -> 4 -> NULL
2 found.

=== Code Execution Successful ===

```

4) Write a c program circular linked list.

C Online Compiler

main.c

Share

Run

Output

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Node { int data; struct Node* next; };
5
6 int main() {
7     struct Node* head = malloc(sizeof(struct Node));
8     head->data = 1; head->next = head;
9     head->next = malloc(sizeof(struct Node));
10    head->next->data = 2; head->next->next = head;
11    free(head->next); head = NULL;
12    return 0;
13 }
14

```

```

/tmp/CtuG9onyHn.o
Circular Linked List: 1 -> 4 -> 3 -> 2 -> NULL
After deleting 3: 1 -> 4 -> 2 -> NULL
2 found.

=== Code Execution Successful ===

```

5) Write a c program stack using all operations push, pop, peek, full, empty.

```
main.c
1 #include <stdio.h>
2 #include <stdbool.h>
3 #define MAX_SIZE 100
4 typedef struct {
5     int items[MAX_SIZE];
6     int top;
7 } Stack;
8 void initStack(Stack *stack) {
9     stack->top = -1;
10 }
11 bool isFull(Stack *stack) {
12     return stack->top == MAX_SIZE - 1;
13 }
14 bool isEmpty(Stack *stack) {
15     return stack->top == -1;
16 }
17 void push(Stack *stack, int value) {
18     if (isFull(stack)) {
19         printf("Stack overflow! Cannot push %d\n", value);
20     } else {
21         stack->items[++stack->top] = value;
22     }
23 }
24 int pop(Stack *stack) {
25     if (isEmpty(stack)) {
26         printf("Stack underflow! Cannot pop\n");
27         return -1;
28     } else {
29         return stack->items[stack->top--];
30     }
31 }
32 int peek(Stack *stack) {
33     if (isEmpty(stack)) {
34         printf("Stack is empty\n");
35         return -1;
36     } else {
37         return stack->items[stack->top];
38     }
39 }
40 int main() {
41     Stack stack;
42     initStack(&stack);
43     push(&stack, 1);
44     push(&stack, 2);
45     push(&stack, 3);
46     printf("Top element: %d\n", peek(&stack));
47     printf("Popped element: %d\n", pop(&stack));
48     printf("Popped element: %d\n", pop(&stack));
49     printf("Top element: %d\n", peek(&stack));
50     push(&stack, 4);
51     printf("Top element: %d\n", peek(&stack));
52     return 0;
53 }
54
```

Output

```
/tmp/hPXDmM21Vp.o
Top element: 3
Popped element: 3
Popped element: 2
Top element: 1
Top element: 4

=== Code Execution Successful ===

/tmp/hPXDmM21Vp.o
Top element: 3
Popped element: 3
Popped element: 2
Top element: 1
Top element: 4

=== Code Execution Successful ===
```

6) Write a c program for array implementation of stack.

main.c	Run	Output
<pre> 1 #include &lt;stdio.h&gt; 2 #include &lt;stdbool.h&gt; 3 #define MAX_SIZE 100 4 typedef struct { 5     int items[MAX_SIZE]; 6     int top; 7 } Stack; 8 void initStack(Stack *stack) { 9     stack-&gt;top = -1; 10 } 11 bool isFull(Stack *stack) { 12     return stack-&gt;top == MAX_SIZE - 1; 13 } 14 bool isEmpty(Stack *stack) { 15     return stack-&gt;top == -1; 16 } 17 void push(Stack *stack, int value) { 18     if (isFull(stack)) { 19         printf("Stack overflow! Cannot push %d\n", value); 20     } else { 21         stack-&gt;top++; 22         stack-&gt;items[stack-&gt;top] = value; 23         printf("%d pushed to stack\n", value); 24     } 25 } 26 int pop(Stack *stack) { 27     if (isEmpty(stack)) { 28         printf("Stack underflow! Cannot pop\n"); 29         return -1; // Return a default value indicating failure 30     } else { 31         int popped = stack-&gt;items[stack-&gt;top]; 32         stack-&gt;top--; 33         return popped; 34     } 35 } 36 int peek(Stack *stack) { 37     if (isEmpty(stack)) { 38         printf("Stack is empty\n"); 39         return -1; // Return a default value indicating failure 40     } else { 41         return stack-&gt;items[stack-&gt;top]; 42     } 43 } 44 int main() { 45     Stack stack; 46     initStack(&amp;stack); 47     push(&amp;stack, 1); 48     push(&amp;stack, 2); 49     push(&amp;stack, 3); 50     printf("Top element: %d\n", peek(&amp;stack)); 51     printf("Popped element: %d\n", pop(&amp;stack)); 52     printf("Popped element: %d\n", pop(&amp;stack)); 53     printf("Top element: %d\n", peek(&amp;stack)); 54     push(&amp;stack, 4); 55     printf("Top element: %d\n", peek(&amp;stack)); 56     return 0; 57 } 58 59 </pre>	Run	<pre> /tmp/kbHsIPkLB1.o 1 pushed to stack 2 pushed to stack 3 pushed to stack Top element: 3 Popped element: 3 Popped element: 2 Top element: 1 4 pushed to stack Top element: 4  === Code Execution Successful === </pre>
<pre> 31     int popped = stack-&gt;items[stack-&gt;top]; 32     stack-&gt;top--; 33     return popped; 34 } 35 } 36 int peek(Stack *stack) { 37     if (isEmpty(stack)) { 38         printf("Stack is empty\n"); 39         return -1; // Return a default value indicating failure 40     } else { 41         return stack-&gt;items[stack-&gt;top]; 42     } 43 } 44 int main() { 45     Stack stack; 46     initStack(&amp;stack); 47     push(&amp;stack, 1); 48     push(&amp;stack, 2); 49     push(&amp;stack, 3); 50     printf("Top element: %d\n", peek(&amp;stack)); 51     printf("Popped element: %d\n", pop(&amp;stack)); 52     printf("Popped element: %d\n", pop(&amp;stack)); 53     printf("Top element: %d\n", peek(&amp;stack)); 54     push(&amp;stack, 4); 55     printf("Top element: %d\n", peek(&amp;stack)); 56     return 0; 57 } 58 59 </pre>	Run	<pre> /tmp/kbHsIPkLB1.o 1 pushed to stack 2 pushed to stack 3 pushed to stack Top element: 3 Popped element: 3 Popped element: 2 Top element: 1 4 pushed to stack Top element: 4  === Code Execution Successful === </pre>

7) Write a c program that implements a stack using linked list.

```
main.c
1 #include <stdio.h>
2 #include <stdbool.h>
3 #define MAX_SIZE 100
4 typedef struct {
5     int items[MAX_SIZE];
6     int top;
7 } Stack;
8 void initStack(Stack *stack) {
9     stack->top = -1;
10 }
11 bool isFull(Stack *stack) {
12     return stack->top == MAX_SIZE - 1;
13 }
14 bool isEmpty(Stack *stack) {
15     return stack->top == -1;
16 }
17 void push(Stack *stack, int value) {
18     if (isFull(stack)) {
19         printf("Stack overflow! Cannot push %d\n", value);
20     } else {
21         stack->top++;
22         stack->items[stack->top] = value;
23         printf("%d pushed to stack\n", value);
24     }
25 }
26 int pop(Stack *stack) {
27     if (isEmpty(stack)) {
28         printf("Stack underflow! Cannot pop\n");
29         return -1; // Return a default value indicating failure
30     } else {
31         int popped = stack->items[stack->top];
32         stack->top--;
33     }
34 }
```

```
/tmp/kbHsIPkBL1.o
1 pushed to stack
2 pushed to stack
3 pushed to stack
Top element: 3
Popped element: 3
Popped element: 2
Top element: 1
4 pushed to stack
Top element: 4
```

```
=== Code Execution Successful ===
```

```
main.c
31     int popped = stack->items[stack->top];
32     stack->top--;
33     return popped;
34 }
35 }
36
37 int peek(Stack *stack) {
38     if (isEmpty(stack)) {
39         printf("Stack is empty\n");
40         return -1; // Return a default value indicating failure
41     } else {
42         return stack->items[stack->top];
43     }
44 }
45
46 int main() {
47     Stack stack;
48     initStack(&stack);
49     push(&stack, 1);
50     push(&stack, 2);
51     push(&stack, 3);
52     printf("Top element: %d\n", peek(&stack));
53     printf("Popped element: %d\n", pop(&stack));
54     printf("Popped element: %d\n", pop(&stack));
55     printf("Top element: %d\n", peek(&stack));
56     push(&stack, 4);
57     printf("Top element: %d\n", peek(&stack));
58     return 0;
59 }
```

```
/tmp/kbHsIPkLB1.o
1 pushed to stack
2 pushed to stack
3 pushed to stack
Top element: 3
Popped element: 3
Popped element: 2
Top element: 1
4 pushed to stack
Top element: 4
```

```
=== Code Execution Successful ===
```