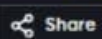




main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 typedef struct Node {
4     int data;
5     struct Node* next;
6 } Node;
7 int main() {
8     Node* head = (Node*)malloc(sizeof(Node));
9     Node* second = (Node*)malloc(sizeof(Node));
10    Node* third = (Node*)malloc(sizeof(Node));
11    head->data = 1;
12    second->data = 2;
13    third->data = 3;
14    head->next = second;
15    second->next = third;
16    third->next = head;
17    Node* current = head;
18    while (1) {
19        printf("%d ", current->data);
20        current = current->next;
21        if (current == head) break;
22    }
23    printf("\n");
24    return 0;
25 }
```

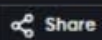
/tmp/ddsCZc0lwT.o

1 2 3

=== Code Execution Successful ===



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 typedef struct Node {
4     int data;
5     struct Node* next;
6     struct Node* prev;
7 } Node;
8 int main() {
9     Node* head = NULL;
10    Node* tail = NULL;
11    Node* node1 = (Node*) malloc(sizeof(Node));
12    node1->data = 10;
13    node1->next = NULL;
14    node1->prev = NULL;
15    head = node1;
16    tail = node1;
17    Node* node2 = (Node*) malloc(sizeof(Node));
18    node2->data = 20;
19    node2->next = NULL;
20    node2->prev = tail;
21    tail->next = node2;
22    tail = node2;
23    Node* node3 = (Node*) malloc(sizeof(Node));
24    node3->data = 30;
25    node3->next = NULL;
26    node3->prev = tail;
27    tail->next = node3;
28    tail = node3;
29    Node* temp = head;
30    while (temp != NULL) {
```

/tmp/mmSFj0jI2x.o

10 20 30

=== Code Execution Successful ===



main.c



Run

Output

Clear

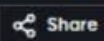
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 typedef struct Node
4 {
5     int data;
6     struct Node* next;
7 } Node;
8 void insert(Node** head, int data)
9 {
10     Node* newNode = malloc(sizeof(Node));
11     newNode->data = data;
12     newNode->next = *head;
13     *head = newNode;
14 }
15 void print(Node* head) {
16     while (head != NULL) {
17         printf("%d -> ", head->data);
18         head = head->next;
19     }
20     printf("NULL\n");
21 }
22 int main() {
23     Node* head = NULL;
24     insert(&head, 1);
25     insert(&head, 2);
26     insert(&head, 3);
27     print(head);
28     return 0;
29 }
```

```
/tmp/hv7YMOH.JMT.o
3 -> 2 -> 1 -> NULL
```

=== Code Execution Successful ===



main.c



Run

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct node {
4     int value;
5     struct node *next;
6 };
7 void printLinkedList(struct node *p) {
8     while (p != NULL) {
9         printf("%d ", p->value);
10        p = p->next;
11    }
12 }
13 int main() {
14     struct node *head;
15     struct node *one = NULL;
16     struct node *two = NULL;
17     struct node *three = NULL;
18     one = malloc(sizeof(struct node));
19     two = malloc(sizeof(struct node));
20     three = malloc(sizeof(struct node));
21     one->value = 1;
22     two->value = 2;
23     three->value = 3;
24     one->next = two;
25     two->next = three;
26     three->next = NULL;
27     head = one;
28     printLinkedList(head);
29 }
```

Output

Clear

```
/tmp/UcuT5r9wVh.o
1 2 3

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

main.c

```
1 #include <stdio.h>
2 #define MAX_SIZE 5
3 int stack[MAX_SIZE];
4 int top = -1;
5 void push(int element) {
6     if (top == MAX_SIZE - 1) {
7         printf("Stack is full.\n");
8         return;
9     }
10    stack[++top] = element;
11 }
12 int pop() {
13     if (top == -1) {
14         printf("Stack is empty.\n");
15         return -1;
16     }
17     return stack[top--];
18 }
19 int main() {
20     push(1);
21     push(2);
22     push(3);
23     printf("Popped: %d\n", pop());
24     printf("Popped: %d\n", pop());
25     return 0;
26 }
```

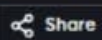
Output

/tmp/TbrLx8vaMH.o
Popped: 3
Popped: 2

=== Code Execution Successful ===



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 #define MAX_SIZE 100
4 typedef struct {
5     int data[MAX_SIZE];
6     int top;
7 } Stack;
8 void initialize(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 element) {
18     if (isFull(stack)) {
19         printf("Stack overflow: Cannot push element %d, stack is full.\n", element);
20     } else {
21         stack->top++;
22         stack->data[stack->top] = element;
23         printf("Pushed element %d onto the stack.\n", element);
24     }
25 }
26 int pop(Stack *stack) {
27     if (isEmpty(stack)) {
28         printf("Stack underflow: Cannot pop from empty stack.\n");
29         return -1;
30     } else {
```

```
/tmp/1IzyvXNS0A.o
Pushed element 1 onto the stack.
Pushed element 2 onto the stack.
Pushed element 3 onto the stack.
Top element of the stack: 3
Popped element 3 from the stack.
Popped element 2 from the stack.
Top element of the stack after pops: 1
Popped element 1 from the stack.
Stack underflow: Cannot pop from empty stack.

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