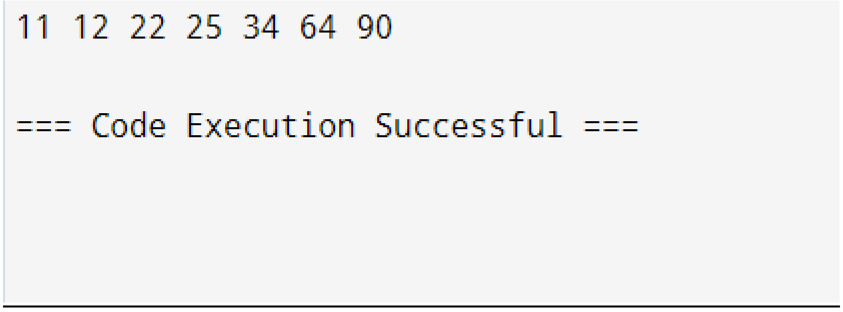
30/8/24

1.

```
#include <stdio.h>
void sort(int arr[], int n) {
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    sort(arr, n);
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT:

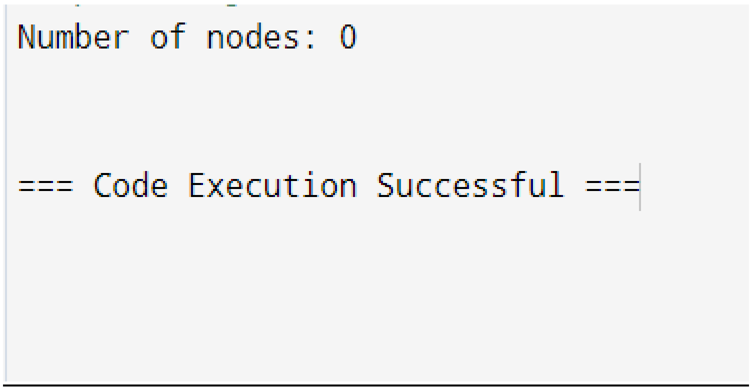


```
11 12 22 25 34 64 90

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

2.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
int countNodes(struct Node* head) {
    int count = 0;
3.
        count++;
        head = head->next;
    }
    return count;
}
int main() {
    struct Node* head = NULL; // Assume linked list is created and populated
    printf("Number of nodes: %d\n", countNodes(head));
    return 0;
}
```

OUTPUT:

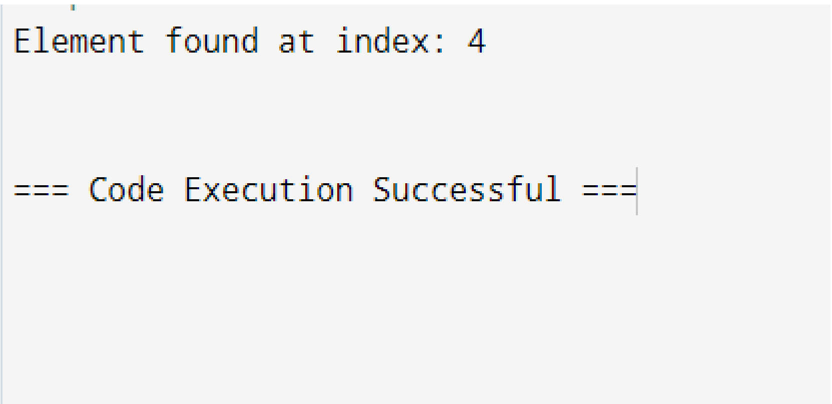


```
Number of nodes: 0


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

3.

```
#include <stdio.h>
int binarySearch(int arr[], int size, int target) {
    int left = 0, right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) return mid;
        if (arr[mid] < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 5;
    int result = binarySearch(arr, size, target);
    printf("Element found at index: %d\n", result);
    return 0;
}
```

OUTPUT:

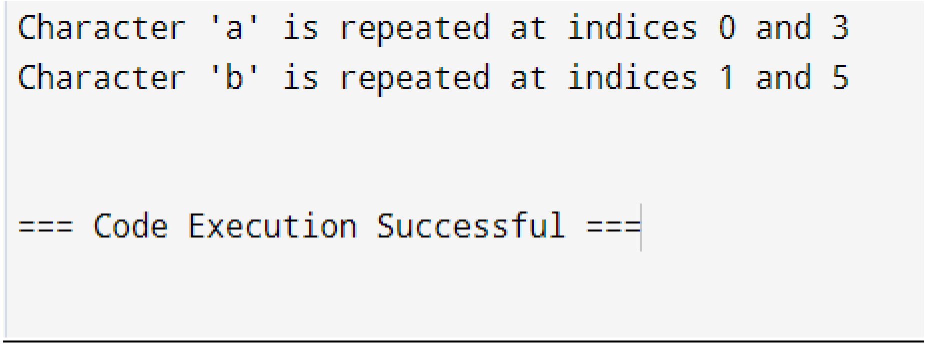


```
Element found at index: 4


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

4.

```
#include <stdio.h>

void findRepeated(char arr[], int size) {
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (arr[i] == arr[j]) {
                printf("Character '%c' is repeated at indices %d and %d\n", arr[i], i, j);
            }
        }
    }
}

int main() {
    char arr[] = {'a', 'b', 'c', 'a', 'd', 'b'};
    int size = sizeof(arr) / sizeof(arr[0]);
    findRepeated(arr, size);
    return 0;
}
```

OUTPUT:

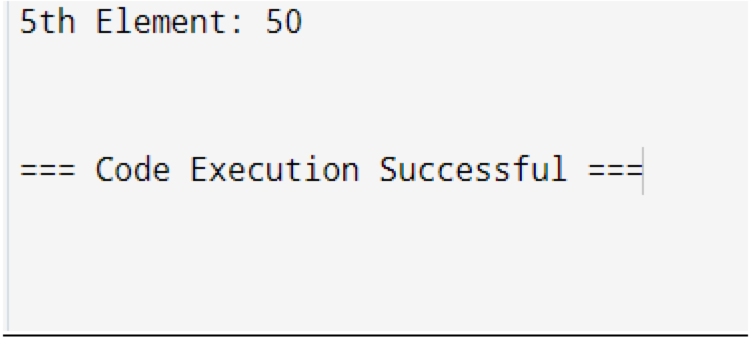


```
Character 'a' is repeated at indices 0 and 3
Character 'b' is repeated at indices 1 and 5


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

5.

```
#include <stdio.h>
int main() {
    int arr[] = {10, 20, 30, 40, 50, 60};
    printf("5th Element: %d\n", arr[4]);
    return 0;
}
```

OUTPUT:

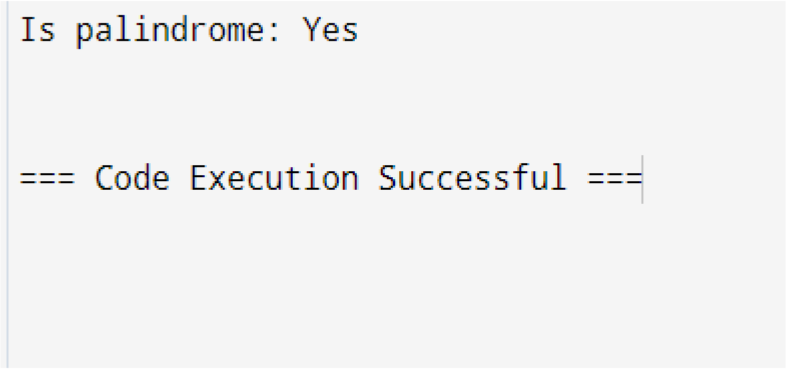


```
5th Element: 50


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

6.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node {
   char data;
   struct Node* next;
} Node;
void push(Node** head_ref, char new_data) {
   Node* new_node = (Node*)malloc(sizeof(Node));
   new_node->data = new_data;
   new_node->next = (*head_ref);
   (*head_ref) = new_node;
}
int isPalindrome(Node* head) {
   Node *slow = head, *fast = head, *prev = NULL, *temp;
   while (fast && fast->next) {
      fast = fast->next->next;
      temp = slow;
      slow = slow->next;
      temp->next = prev;
      prev = temp;
   }
   if (fast) slow = slow->next; // Skip the middle element for odd length
   while (prev && slow) {
      if (prev->data != slow->data) return 0;
      prev = prev->next;
      slow = slow->next;
   }
   return 1;
}
int main() {
   Node* head = NULL;
   char str[] = "radar";
   for (int i = 0; i < strlen(str); i++) push(&head, str[i]);
   printf("Is palindrome: %s\n", isPalindrome(head) ? "Yes" : "No");
   return 0;
}
```

OUTPUT:

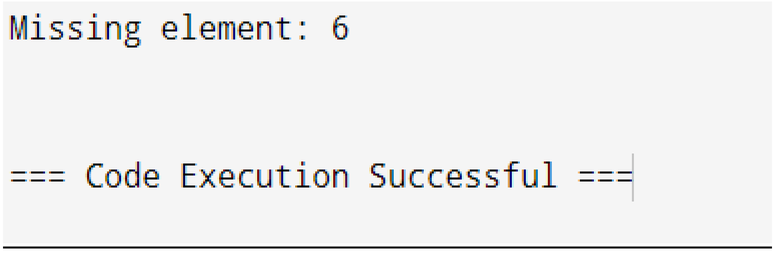


```
Is palindrome: Yes


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

7.

```
#include <stdio.h>
int main() {
    int arr[] = {3, 7, 1, 2, 8, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]) + 1;
    int total = n * (n + 1) / 2;
    int sum = 0;
    for (int i = 0; i < n - 1; i++) {
        sum += arr[i];
    }
    printf("Missing element: %d\n", total - sum);
    return 0;
}
```

OUTPUT:

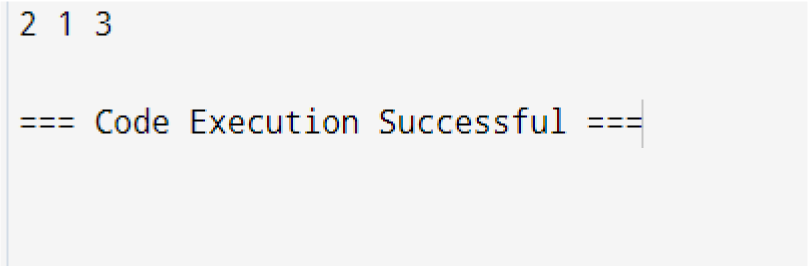


```
Missing element: 6


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

8.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}
void inorder(struct Node* root) {
    if (root) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
int main() {
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    inorder(root);
    return 0;
}
```

OUTPUT:

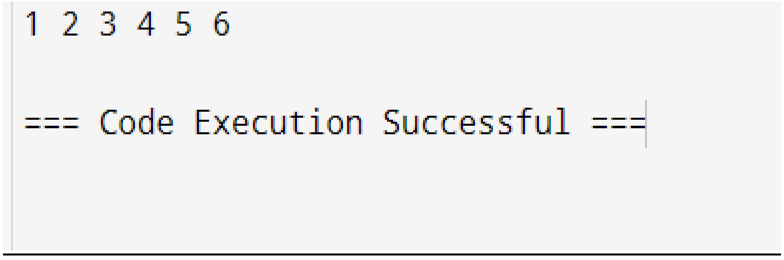


```
2 1 3

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

9.

```
#include <stdio.h>
void concatenate(int arr1[], int size1, int arr2[], int size2, int result[]) {
    for (int i = 0; i < size1; i++) result[i] = arr1[i];
    for (int i = 0; i < size2; i++) result[size1 + i] = arr2[i];
}
int main() {
    int arr1[] = {1, 2, 3};
    int arr2[] = {4, 5, 6};
    int size1 = sizeof(arr1) / sizeof(arr1[0]);
    int size2 = sizeof(arr2) / sizeof(arr2[0]);
    int result[size1 + size2];

    concatenate(arr1, size1, arr2, size2, result);

    for (int i = 0; i < size1 + size2; i++) printf("%d ", result[i]);
    return 0;
}
```

OUTPUT:



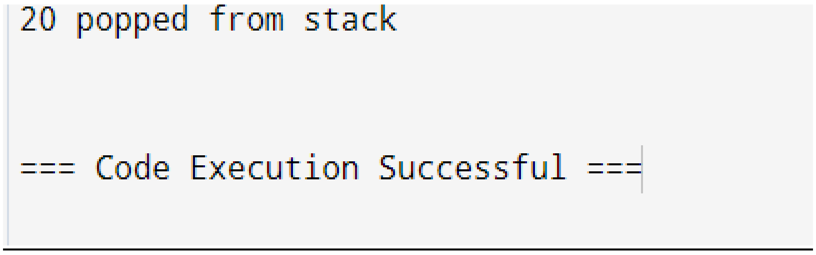
```
1 2 3 4 5 6

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

10.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Queue {
    int front, rear, size;
    unsigned capacity;
    int* array;
} Queue;
Queue* createQueue(unsigned capacity) {
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;
    queue->rear = capacity - 1;
    queue->array = (int*)malloc(queue->capacity * sizeof(int));
    return queue;
}
int isFull(Queue* queue) { return (queue->size == queue->capacity); }
int isEmpty(Queue* queue) { return (queue->size == 0); }
void enqueue(Queue* queue, int item) {
    if (isFull(queue)) return;
    queue->rear = (queue->rear + 1) % queue->capacity;
    queue->array[queue->rear] = item;
    queue->size++;
}
int dequeue(Queue* queue) {
    if (isEmpty(queue)) return -1;
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1) % queue->capacity;
    queue->size--;
    return item;
}
int front(Queue* queue) {
    if (isEmpty(queue)) return -1;
    return queue->array[queue->front];
}
typedef struct Stack {
    Queue* q1;
    Queue* q2;
} Stack;
Stack* createStack(unsigned capacity) {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->q1 = createQueue(capacity);
    stack->q2 = createQueue(capacity);
    return stack;
}
void push(Stack* stack, int item) {
    enqueue(stack->q2, item);
    while (!isEmpty(stack->q1)) {
        enqueue(stack->q2, dequeue(stack->q1));
    }
    Queue* temp = stack->q1;
    stack->q1 = stack->q2;
    stack->q2 = temp;
}
int pop(Stack* stack) {
    return dequeue(stack->q1);
```

```
}
int main() {
    Stack* stack = createStack(100);
    push(stack, 10);
    push(stack, 20);
    printf("%d popped from stack\n", pop(stack));
    return 0;
}
```

OUTPUT:



```
20 popped from stack


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