1. .

```c
#include <stdio.h>

// Function to perform linear search
int linear_search(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Return the index of the target
        }
    }
    return -1; // Target not found
}

int main() {
    // Test case 1: Target present
    int arr1[] = {10, 25, 8, 14, 3, 23};
    int size1 = sizeof(arr1) / sizeof(arr1[0]);
    int target1 = 14;
    int result1 = linear_search(arr1, size1, target1);
    if (result1 != -1) {
        printf("Target %d found at index: %d\n", target1, result1);
    } else {
        printf("Target %d not found in the array.\n", target1);
    }

    // Test case 2: Target absent
    int arr2[] = {5, 7, 1, 2, 9};
    int size2 = sizeof(arr2) / sizeof(arr2[0]);
    int target2 = 4;
    int result2 = linear_search(arr2, size2, target2);
    if (result2 != -1) {
        printf("Target %d found at index: %d\n", target2, result2);
    } else {
        printf("Target %d not found in the array.\n", target2);
    }
    return 0;
}
```

input

```
Target 14 found at index: 3
Target 4 not found in the array.
```

2. .

```c
#include <stdio.h>

// Function to perform binary search
int binary_search(int arr[], int left, int right, int target) {
    while (left <= right) {
        int mid = left + (right - left) / 2; // Calculate the middle index

        if (arr[mid] == target) {
            return mid; // Target found at mid index
        } else if (arr[mid] < target) {
            left = mid + 1; // Continue search in the right half
        } else {
            right = mid - 1; // Continue search in the left half
        }
    }
    return -1; // Target not present in the array
}

int main() {
    // Test case 1: Searching for a value in the middle of the array
    int arr1[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
    int target1 = 10;
    int result1 = binary_search(arr1, 0, sizeof(arr1)/sizeof(arr1[0]) - 1, target1);
    printf("Searching for %d... ", target1);
    result1 != -1 ? printf("found at index %d\n", result1) : printf("not found\n");
```

```c
    result1 != -1 ? printf("found at index %d\n", result1) : printf("not found\n");

    // Test case 2: Searching for values at the edges of the array
    int target2 = 2;
    int result2 = binary_search(arr1, 0, sizeof(arr1)/sizeof(arr1[0]) - 1, target2);
    printf("Searching for %d... ", target2);
    result2 != -1 ? printf("found at index %d\n", result2) : printf("not found\n");

    int target3 = 20;
    int result3 = binary_search(arr1, 0, sizeof(arr1)/sizeof(arr1[0]) - 1, target3);
    printf("Searching for %d... ", target3);
    result3 != -1 ? printf("found at index %d\n", result3) : printf("not found\n");

    // Test case 3: Searching for a value not present in the array
    int target4 = 11;
    int result4 = binary_search(arr1, 0, sizeof(arr1)/sizeof(arr1[0]) - 1, target4);
    printf("Searching for %d... ", target4);
    result4 != -1 ? printf("found at index %d\n", result4) : printf("not found\n");

    // Test case 4: Performing a search in an empty array
    int arr2[] = {};
    int target5 = 3;
    int result5 = binary_search(arr2, 0, sizeof(arr2)/sizeof(arr2[0]) - 1, target5);
    printf("Searching for %d in an empty array... ", target5);
    result5 != -1 ? printf("found at index %d\n", result5) : printf("not found\n");
```

```
    printf("Searching for %d in an empty array... ", target5);
    result5 != -1 ? printf("found at index %d\n", result5) : printf("not found\n");

    return 0;
}
```

input

```
main.c: In function 'main':
main.c:55:42: warning: overflow in conversion from 'long unsigned int' to 'int' changes value from '18446744073
709551615' to '-1' [-Woverflow]
   55 |     int result5 = binary_search(arr2, 0, sizeof(arr2)/sizeof(arr2[0]) - 1, target5);
      |                                          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
main.c:55:19: warning: 'binary_search' accessing 4 bytes in a region of size 0 [-Wstringop-overflow=]
   55 |     int result5 = binary_search(arr2, 0, sizeof(arr2)/sizeof(arr2[0]) - 1, target5);
      |                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
main.c:55:19: note: referencing argument 1 of type 'int *'
main.c:12:5: note: in a call to function 'binary_search'
   12 | int binary_search(int arr[], int left, int right, int target) {
      |     ^~~~~~~~~~~~~
Searching for 10... found at index 4
Searching for 2... found at index 0
Searching for 20... found at index 9
Searching for 11... not found
Searching for 3 in an empty array... not found
```

3. .

```c
#include <stdio.h>

// Function to perform insertion sort
void insertion_sort(int arr[], int length) {
    int i, key, j;
    for (i = 1; i < length; i++) {
        key = arr[i]; // The element to be inserted
        j = i - 1;

        // Move elements of arr[0..i-1], that are greater than key,
        // to one position ahead of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key; // Insert the key at after the element just smaller than it
    }
}

int main() {
    // Example of declaring an array and calling the insertion_sort function
    int arr[] = {9, 5, 1, 4, 3};
    int length = sizeof(arr) / sizeof(arr[0]);

    insertion_sort(arr, length);

    // Code to print the sorted array
    printf("Sorted array: ");
    for (int i = 0; i < length; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Additional tests
    // Test with an already sorted array
```

```c
    // Test with an already sorted array
    int sorted_arr[] = {1, 2, 3, 4, 5};
    insertion_sort(sorted_arr, sizeof(sorted_arr) / sizeof(sorted_arr[0]));
    // Test with a reverse sorted array
    int reverse_arr[] = {5, 4, 3, 2, 1};
    insertion_sort(reverse_arr, sizeof(reverse_arr) / sizeof(reverse_arr[0]));
    // Test with an array with duplicate elements
    int duplicate_arr[] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
    insertion_sort(duplicate_arr, sizeof(duplicate_arr) / sizeof(duplicate_arr[0]));

    // Print additional test results
    printf("Already sorted array: ");
    for (int i = 0; i < sizeof(sorted_arr) / sizeof(sorted_arr[0]); i++) {
        printf("%d ", sorted_arr[i]);
    }
    printf("\n");

    printf("Reverse sorted array: ");
    for (int i = 0; i < sizeof(reverse_arr) / sizeof(reverse_arr[0]); i++) {
        printf("%d ", reverse_arr[i]);
    }
    printf("\n");

    printf("Array with duplicates: ");
    for (int i = 0; i < sizeof(duplicate_arr) / sizeof(duplicate_arr[0]); i++) {
        printf("%d ", duplicate_arr[i]);
    }
    printf("\n");

    return 0;
}
```

```
Sorted array: 1 3 4 5 9
Already sorted array: 1 2 3 4 5
Reverse sorted array: 1 2 3 4 5
Array with duplicates: 1 1 2 3 3 4 5 5 5 6 9
```

4. .

```c
#include <stdio.h>

// Function to perform selection sort
void selection_sort(int arr[], int length) {
    int i, j, min_idx, temp;

    // Move the boundary of the unsorted subarray one by one
    for (i = 0; i < length-1; i++) {
        // Find the minimum element in the unsorted array
        min_idx = i;
        for (j = i+1; j < length; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }

        // Swap the found minimum element with the first element
        temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    // Example of declaring an array and calling the selection_sort function
    int arr[] = {64, 25, 12, 22, 11};
    int length = sizeof(arr) / sizeof(arr[0]);

    selection_sort(arr, length);

    // Code to print the sorted array
    printf("Sorted array: ");
    for (int i = 0; i < length; i++) {
        printf("%d ", arr[i]);
```

```
42          printf("%d ", arr[i]);
43      }
44      printf("\n");
45
46      // Additional tests
47      // Test with an already sorted array
48      int sorted_arr[] = {1, 2, 3, 4, 5};
49      selection_sort(sorted_arr, sizeof(sorted_arr) / sizeof(sorted_arr[0]));
50      // Test with a reverse sorted array
51      int reverse_arr[] = {5, 4, 3, 2, 1};
52      selection_sort(reverse_arr, sizeof(reverse_arr) / sizeof(reverse_arr[0]));
53      // Test with an array with duplicate elements
54      int duplicate_arr[] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
55      selection_sort(duplicate_arr, sizeof(duplicate_arr) / sizeof(duplicate_arr[0]));
56
57      // Print additional test results
58      printf("Already sorted array: ");
59      for (int i = 0; i < sizeof(sorted_arr) / sizeof(sorted_arr[0]); i++) {
60          printf("%d ", sorted_arr[i]);
61      }
62      printf("\n");
63
64      printf("Reverse sorted array: ");
65      for (int i = 0; i < sizeof(reverse_arr) / sizeof(reverse_arr[0]); i++) {
66          printf("%d ", reverse_arr[i]);
67      }
68      printf("\n");
69
70      printf("Array with duplicates: ");
71      for (int i = 0; i < sizeof(duplicate_arr) / sizeof(duplicate_arr[0]); i++) {
72          printf("%d ", duplicate_arr[i]);
73      }
74      printf("\n");
75
```

```
75
76      return 0;
77  }
```

input

```
Sorted array: 11 12 22 25 64
Already sorted array: 1 2 3 4 5
Reverse sorted array: 1 2 3 4 5
Array with duplicates: 1 1 2 3 3 4 5 5 5 6 9
```