

Example 3: Palindromes

Task: Given a string, check if it is a palindrome.

Pseudo-code:

- Start at the first and last characters.
- Compare the characters, then move inward.
- If a pair differs, declare NOT PALINDROME.
- If all pairs match, declare PALINDROME.

Does this program implement that last step correctly?

From `palindrome-run.c`:

```
int l = 0;
char str[] = "abcba";
int h = strlen(str) - 1;
while (h > l) {
    if (str[l++] != str[h--]) {
        printf("NOT PALINDROME");
        break;
    }
}
if (h == l)
    printf("PALINDROME");
```

Main-body excerpt; unused declaration omitted.

Why the supplied test happens to work

For `str = "abcba"`, every compared pair matches.

Comparison	Before: (l, h)	Characters	After: (l, h)
1	$(0, 4)$	'a' and 'a'	$(1, 3)$
2	$(1, 3)$	'b' and 'b'	$(2, 2)$

```
if (str[l++] != str[h--]) { /* ... */ }
```

The comparison uses the old indices. Both updates are complete before entering the `if`-body, even when the characters differ.

The final test succeeds here

The indices meet at the middle character: `h == l` is true.

But passing one odd-length example does not establish correctness.

Error 1: an even-length palindrome prints nothing

Change only the string to "abba".

Comparison	Before: (l, h)	Characters	After: (l, h)
1	(0, 3)	'a' and 'a'	(1, 2)
2	(1, 2)	'b' and 'b'	(2, 1)

Every pair matches, so no failure message is printed. The loop ends because $h > l$ is false.

```
if (h == l)                /* 1 == 2 is false */
    printf("PALINDROME");
```

Meeting is not the only successful ending

For an odd length, the indices **meet**; for an even length, they **cross**. This program prints nothing for abba.

Error 2: a mismatch can print both messages

Change only the string to "abc".

Initially, $l = 0$, $h = 2$.

```
if (str[l++] != str[h--]) {  
    printf("NOT PALINDROME");  
    break;  
}
```

/ After the loop: */*

```
if (h == l)  
    printf("PALINDROME");
```

'a' differs from 'c', but the updates have already made $l = 1$ and $h = 1$.

The failure message is printed. `break` leaves only the loop; the final test still runs and prints the success message too.

Exact output: no newline separates the messages

NOT PALINDROME
PALINDROME

Merely changing $h == l$ to $h <= l$ does not fix this: abc would still print both messages.

Fix 1: remember a mismatch with a flag

Inside main; include `<stdio.h>` and `<string.h>`.

```
char str[] = "abba";
int l = 0, h = (int)strlen(str) - 1;
int is_palindrome = 1;
while (h > l) {
    if (str[l++] != str[h--]) {
        is_palindrome = 0;
        break;
    }
}
if (is_palindrome)
    printf("PALINDROME\n");
else
    printf("NOT PALINDROME\n");
```

The flag records the result.

Start by assuming a palindrome. A mismatch changes the flag to 0.

After the loop, print based on the flag, not on whether the indices meet.

Exactly one message is printed, for either odd or even length.

Fix 2: return immediately on a mismatch

Inside main; include `<stdio.h>` and `<string.h>`.

```
char str[] = "abba";
int l = 0, h = (int)strlen(str) - 1;
while (h > l) {
    if (str[l++] != str[h--]) {
        printf("NOT PALINDROME\n");
        return 0;
    }
}
printf("PALINDROME\n");
return 0;
```

The control flow records the result.

return exits main, so a mismatch cannot reach the success message.

Reaching the end of the loop means all compared pairs matched.

No flag or final `h == l` test is needed.

Check both repairs on small inputs

Change `str` and predict the result before running the code.

str	Original program	Either repaired program
a	PALINDROME	PALINDROME
ab	NOT PALINDROME	NOT PALINDROME
abc	Both messages	NOT PALINDROME
abba	No output	PALINDROME
abca	NOT PALINDROME	NOT PALINDROME
abcba	PALINDROME	PALINDROME

Testing lesson: Include odd/even lengths and both outcomes.

A loop's final indices do not always tell you why it stopped.

What exactly is printed?

```
char s[] = "coding";  
s[0] = 'C';  
s[3] = '\\0';  
printf("%s", s);
```

Poll response: Submit only the output, without quotation marks.

Discuss in your group, then submit one answer.

1 mark

What exactly is printed after this loop?

```
char s[] = "PLANT";  
for (int i = 2; s[i] != '\0'; i++)  
    s[i] = s[i + 1];  
printf("%s", s);
```

Poll response: Submit only the final string, without quotation marks.

Trace the assignments in order, including the terminator.

1 mark

Mini-quiz 1 | S3: complete the reversal

Fill A and B so that r contains the reverse of s.

```
char s[] = "straw";  
char r[6];  
for (int i = 0; i < 5; i++)  
    r[i] = s[ /* A */ ];  
r[ /* B */ ] = '\\0';  
printf("%s", r);
```

Poll response: Type the two replacements as A = ...; B = ...

Use an expression in i for A and an index for B.

1 mark

Mini-quiz 1 | Answers and reasoning

S1: Cod The first character becomes C. Printing stops at index 3, where we inserted `'\0'`.

S2: PLNT The A at index 2 is removed: N, T, and finally `'\0'` shift left.

S3: `A = 4 - i; B = 5` Read indices 4, 3, 2, 1, 0, then terminate `r`. The printed result is
warts.

Marking

One mark each for S1 and S2; half a mark per blank in S3. Accept equivalent expressions. Total: 3 marks.

Largest element in an array

Task: Find the largest element in a non-empty array. Assume `arr` contains `n` elements and `n > 0`.

Largest element in an array

Task: Find the largest element in a non-empty array. Assume `arr` contains `n` elements and `n > 0`.

```
int i;  
  
// Initialize maximum element  
int max = arr[0];  
  
// Traverse array elements from second and  
// compare every element with current max  
for (i = 1; i < n; i++)  
    if (arr[i] > max)  
        max = arr[i];
```

Source: `largest.c`; program segment, algorithm unchanged.

Sorting an array in decreasing order

Task : Given array of n ($n \leq 1000$) numbers. Sort them in decreasing order of numbers.

15	8	3	12	30	7	9	17	32	19
----	---	---	----	----	---	---	----	----	----

Sorting an array in decreasing order

Task : Given array of n ($n \leq 1000$) numbers. Sort them in decreasing order of numbers.

15	8	3	12	30	7	9	17	32	19
----	---	---	----	----	---	---	----	----	----

One possible way: An **algorithm**

- Find max, place it at first location.
- Sort the array from second location to end.

Called **Selection Sort**.

Selection sort

15	8	3	12	30	7	9	17	32	19

Selection sort

15	8	3	12	30	7	9	17	32	19
32	8	3	12	30	7	9	17	15	19

Selection sort

15	8	3	12	30	7	9	17	32	19
32	8	3	12	30	7	9	17	15	19
32	30	3	12	8	7	9	17	15	19

Selection sort

15	8	3	12	30	7	9	17	32	19
32	8	3	12	30	7	9	17	15	19
32	30	3	12	8	7	9	17	15	19
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
32	30	19	17	15	12	9	8	7	3

Pseudo-code :

- while ($i \leq n$)
 - maxindex = index of the max element in the part of the array indexed from i to n . Find maxindex.

Selection sort

15	8	3	12	30	7	9	17	32	19
32	8	3	12	30	7	9	17	15	19
32	30	3	12	8	7	9	17	15	19
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
32	30	19	17	15	12	9	8	7	3

Pseudo-code :

- while ($i \leq n$)
 - maxindex = index of the max element in the part of the array indexed from i to n . Find maxindex.
 - swap elements array[i] and array[maxindex];