

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];

Selection sort: from pseudocode to program

For each position i :

- Find the index of the maximum in positions i through $n-1$.
- Swap that element with $a[i]$.

Here `max` stores an **index**, not the maximum value. The result is in **decreasing order**.

Selection sort: from pseudocode to program

For each position i :

- Find the index of the maximum in positions i through $n-1$.
- Swap that element with $a[i]$.

Here `max` stores an **index**, not the maximum value. The result is in **decreasing order**.

```
for (i=0; i<n; i++)
{
    max = i;
    for (j=i+1; j<n; j++)
    {
        if (a[j] > a[max])
            max = j;
    }
    temp = a[i];
    a[i] = a[max];
    a[max] = temp;
}
```

Source: `sel-sort-slide.c`; the supplied loop and swap are unchanged. Assume `i`, `j`, `max`, `temp` are declared as integers.

Mini-quiz 2 | T1: the first selection-sort pass

Sort in decreasing order, exactly as in the preceding slides.

Start with:

4 9 2 7 5

In one pass, find the maximum in the unsorted part, then swap it with the first element of that part.

What is the entire array after the first pass?

Poll response: Type all five values, separated by spaces.

Give the intermediate array, not the fully sorted array.

1 mark

Mini-quiz 2 | T2: the second selection-sort pass

Continue sorting in decreasing order.

After the first pass, the array is:

9 | 4 2 7 5

The first element is already in its final position. The remaining four elements form the unsorted part.

What is the entire array after one more pass?

Poll response: Type all five values, separated by spaces.

Select the maximum of the unsorted part, then swap once.

1 mark

Mini-quiz 2 | T3: complete the maximum search

Which condition finds the maximum for decreasing-order sorting?

```
int max = i;
for (int j = i + 1; j < n; j++) {
    if ( /* condition */ )
        max = j;
}
int temp = a[i];
a[i] = a[max];
a[max] = temp;
```

Poll response: Type only the missing comparison expression.

Compare values, not indices.

1 mark

Mini-quiz 2 | Answers and reasoning

T1: 9 4 2 7 5 The maximum is 9; swap it with 4 at index 0.

T2: 9 7 2 4 5 In 4 2 7 5, the maximum is 7; swap it with 4 at index 1.

T3: $a[j] > a[\max]$ Update the index when a larger value is found. Accept the equivalent comparison with reversed operands; $\geq$ also sorts correctly, choosing the later equal maximum.

What each pass achieves

One more element reaches its final position at the front. The remaining suffix need not be sorted yet.

Total: 3 marks

Multi-dimensional arrays in C

Declaration:

```
int matrix[MAXROW][MAXCOL];  
/* For example: */  
int matrix[10][10];
```

A two-dimensional array has rows and columns.

How is it stored in memory?

Multi-dimensional arrays in C

Declaration:

```
int matrix[MAXROW][MAXCOL];  
/* For example: */  
int matrix[10][10];
```

A two-dimensional array has rows and columns.

How is it stored in memory?

Initializer from `arrayinit.c`; the surrounding `main` is omitted and the missing semicolon after the initializer is added.

Initialization:

```
int matrix[3][4] = {  
    {1, 2, 3, 4},  
    {5, 6, 7, 8},  
    {9, 10, 11, 12}  
};
```

This declares 3 rows and 4 columns.

Multi-dimensional arrays: row-major storage

$A[i][j]$ is the element in row i , column j . Both indices start at 0.

	Column 0	Column 1	Column 2	Column 3
Row 0	1	2	3	4
Row 1	5	6	7	8
Row 2	9	10	11	12

Multi-dimensional arrays: row-major storage

$A[i][j]$ is the element in row i , column j . Both indices start at 0.

	Column 0	Column 1	Column 2	Column 3
Row 0	1	2	3	4
Row 1	5	6	7	8
Row 2	9	10	11	12

Storage order: all of row 0, then row 1, then row 2.

1	2	3	4	5	6	7	8	9	10	11	12
row 0				row 1				row 2			

Values from `arrayinit.c`. The missing external figure is replaced by this typeset view of the same initializer.

Initializing multi-dimensional arrays

```
int matrix1[3][4] = {  
    1, 2, 3, 4,  
    5, 6, 7, 8,  
    9, 10, 11, 12};
```

```
int matrix2[][4] = {  
    1, 2, 3, 4,  
    5, 6, 7, 8,  
    9, 10, 11, 12};
```

Initializing multi-dimensional arrays

```
int matrix1[3][4] = {  
    1, 2, 3, 4,  
    5, 6, 7, 8,  
    9, 10, 11, 12};
```

```
int matrix2[][4] = {  
    1, 2, 3, 4,  
    5, 6, 7, 8,  
    9, 10, 11, 12};
```

The initializer fills elements in row-major order.

The first dimension can be inferred from the initializer.

The column size cannot be omitted.

Source: arrayinit2.c; both declarations are unchanged, with the surrounding main omitted.

Multi-dimensional arrays: what is printed?

Assume $N1 = 3$ and $N2 = 4$. Trace the output.

```
int matrix[N1][N2] = {
    1, 2, 3, 4,
    5, 6, 7, 8,
    9, 10, 11, 12};
int i;
for (i = 0; i < N1; i++) {
    printf("%d\n", matrix[i][2]++);
}
for (i = 0; i < N2; i++) {
    printf("%d\t", matrix[2][i]);
}
printf("\n");
```

Source: `matrixexample.c`; body excerpt, with no changes to the initializer or loops. The first loop uses post-increment.

Reading and writing matrices

mat: the matrix; rows and cols: its dimensions.

Reading the elements:

```
for (int i = 0; i < rows; i++)  
    for (int j = 0; j < cols; j++)  
        scanf("%d", &mat[i][j]);
```

Reading and writing matrices

mat: the matrix; rows and cols: its dimensions.

Reading the elements:

```
for (int i = 0; i < rows; i++)
    for (int j = 0; j < cols; j++)
        scanf("%d", &mat[i][j]);
```

Writing one row per line:

```
for (int i = 0; i < rows; i++)
{
    for (int j = 0; j < cols; j++)    /* print a row */
        { printf ("%d ", mat[i][j]); }    /* notice missing \n */
    printf ("\n");                    /* print a newline at the end a row */
}
```

Sources: matrix-read.c and matrix-write.c; supplied loop fragments, unchanged.

What exactly is printed?

```
int mat[][3] = {2, 4, 6, 8, 10, 12};  
printf("%d %d", mat[0][2], mat[1][1]);
```

The number of rows is inferred from the initializer.

Poll response: Type the two printed values, separated by a space.

Discuss in your group, then submit one answer.

1 mark

What exactly is printed by these two loops?

```
int mat[2][3] = {{1, 2, 3}, {4, 5, 6}};  
for (int i = 0; i < 2; i++)  
    printf("%d ", mat[i][1]++);  
printf("| ");  
for (int j = 0; j < 3; j++)  
    printf("%d ", mat[1][j]);
```

Poll response: Type the output on one line, including |. Ignore the final space.

Keep track of the stored values after the first loop.

1 mark

What is the order of the printed values?

```
int mat[2][3] = {{1, 2, 3}, {4, 5, 6}};  
for (int j = 0; j < 3; j++)  
    for (int i = 0; i < 2; i++)  
        printf("%d ", mat[i][j]);
```

Poll response: Type all six values, separated by spaces.

Trace the nested loops in the order shown.

1 mark

Mini-quiz 3 | Answers and reasoning

M1: 6 10 The rows are 2 4 6 and 8 10 12. Thus `mat[0][2]` is 6 and `mat[1][1]` is 10.

M2: 2 5 | 4 6 6 Post-increment prints the old values 2 and 5, then changes them to 3 and 6.
The second row is now 4 6 6.

M3: 1 4 2 5 3 6 The outer loop fixes a column; the inner loop visits its two rows. The output is column by column, although storage is still row-major.

Marking

One mark per correct output. Ignore extra spaces. Total: 3 marks.

Matrix operations: addition

Task: Add two matrices of the same dimensions.

In the supplied code, A is the result: $A = B + C$.

Assume A, B, and C have dimensions N1 by N2; B and C are already initialized.

Matrix operations: addition

Task: Add two matrices of the same dimensions.

In the supplied code, A is the result: $A = B + C$.

Assume A, B, and C have dimensions N1 by N2; B and C are already initialized.

```
for (i = 0; i<N1; i++) {  
    for (j = 0; j<N2; j++) {  
        A[i][j] = B[i][j] + C[i][j];  
    }  
}
```

Source: matrixadd.c; loop excerpt, unchanged. Its $A = B + C$ convention is retained rather than renaming the matrices.

Matrix operations: multiplication

Task: Multiply square matrices A and B of size N1.

One output entry uses a row of A and a column of B.

$$C[i][j] = \sum_{k=0}^{N1-1} A[i][k] B[k][j]$$

Matrix operations: multiplication

Task: Multiply square matrices A and B of size N1.

One output entry uses a row of A and a column of B.

$$C[i][j] = \sum_{k=0}^{N1-1} A[i][k] B[k][j]$$

Keep fixed	Vary
Row i of A	Column k of A
Column j of B	Row k of B

Reset sum to zero for each $C[i][j]$, then accumulate products.

Multiplication rule from `matrixmult-run.c`, specialized to the square-matrix exercise in `matrixmult.c`; replaces the missing figure.