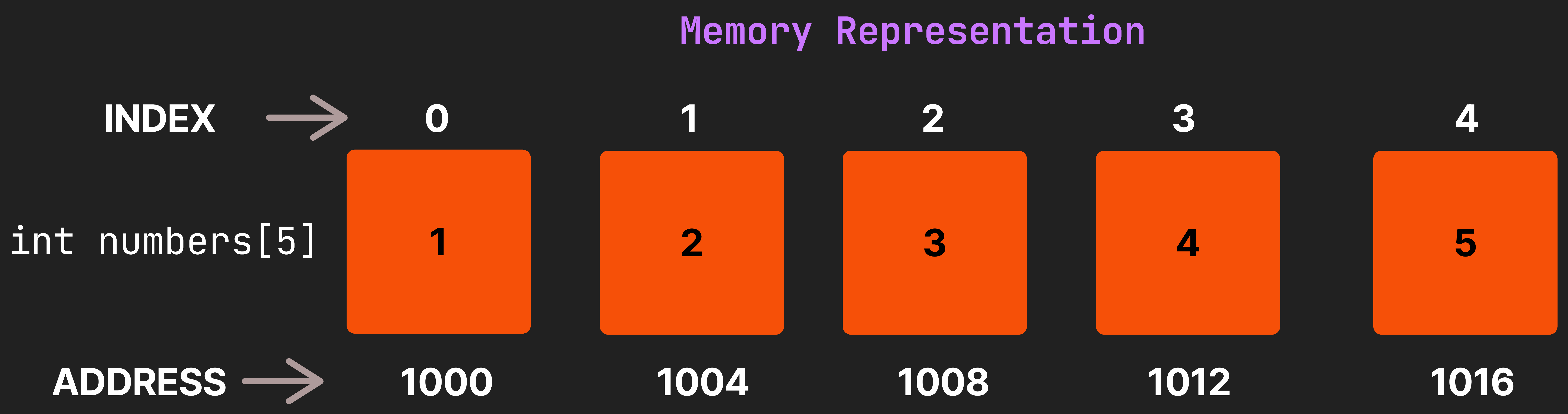
Introduction to an array

- Collection of elements of the same type
- Stored in contiguous memory
- Stores multiple values using one variable name
- Fixed size (cannot change at runtime)

    - Syntax: `type arrayName[size];`
    - Example: `int numbers[5] = {1, 2, 3, 4, 5};`



One-dimensional array

- A one-dimensional array stores elements in a linear form.

Example:

Declaration

```
int a[5];
```

Declaration

```
int a[5] = {10, 20, 30, 40, 50};
```

Partial Initialization

```
int a[5] = {1, 2};
```

- Remaining elements are set to 0.

Accessing Elements

```
a[0] = 100;
printf("%d", a[2]);
```

One-dimensional array

Traversing an Array - Input/Output

Input

```
for(int i = 0; i < 5; i++)
    scanf("%d", &a[i]);
```

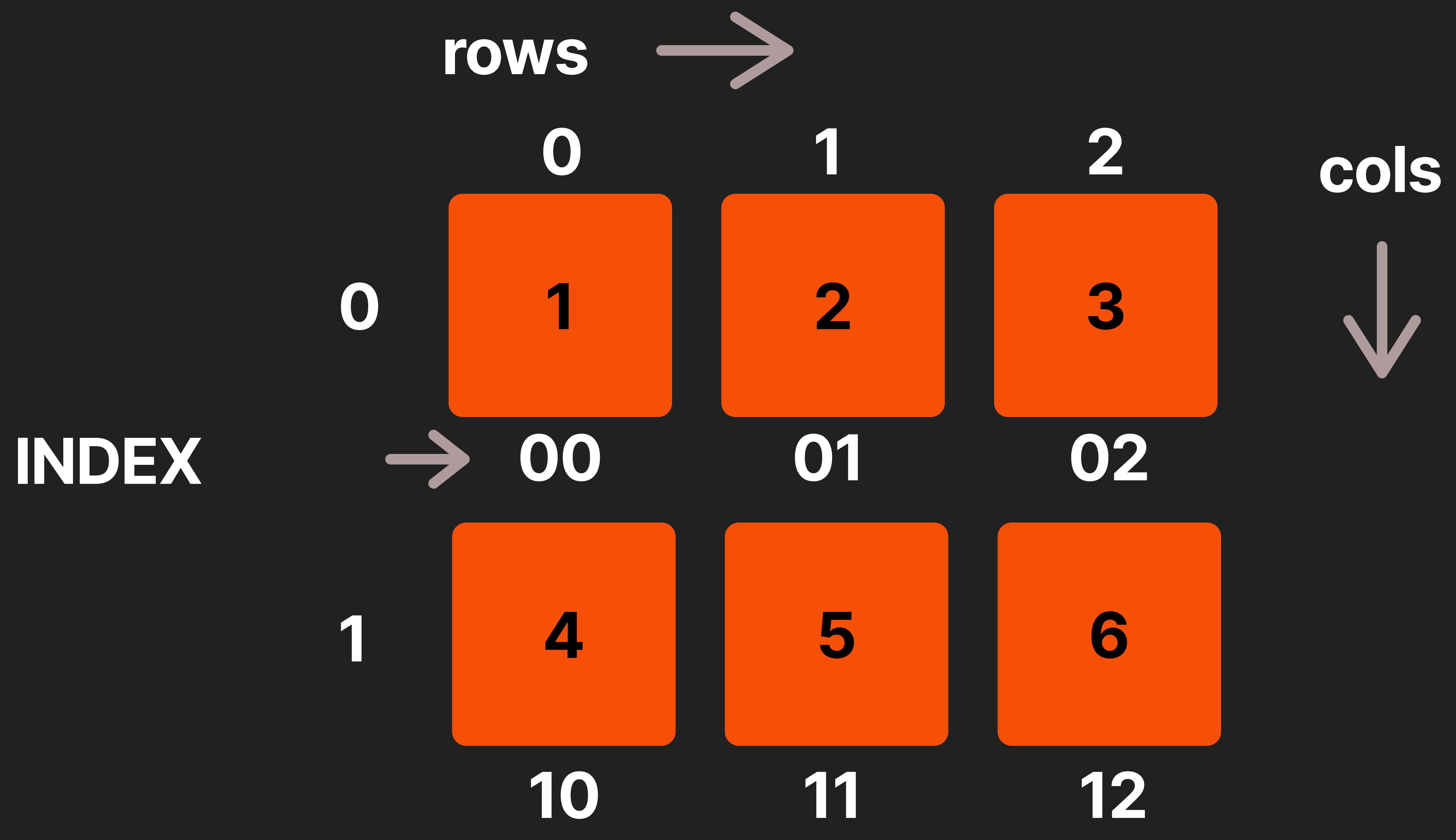
Output

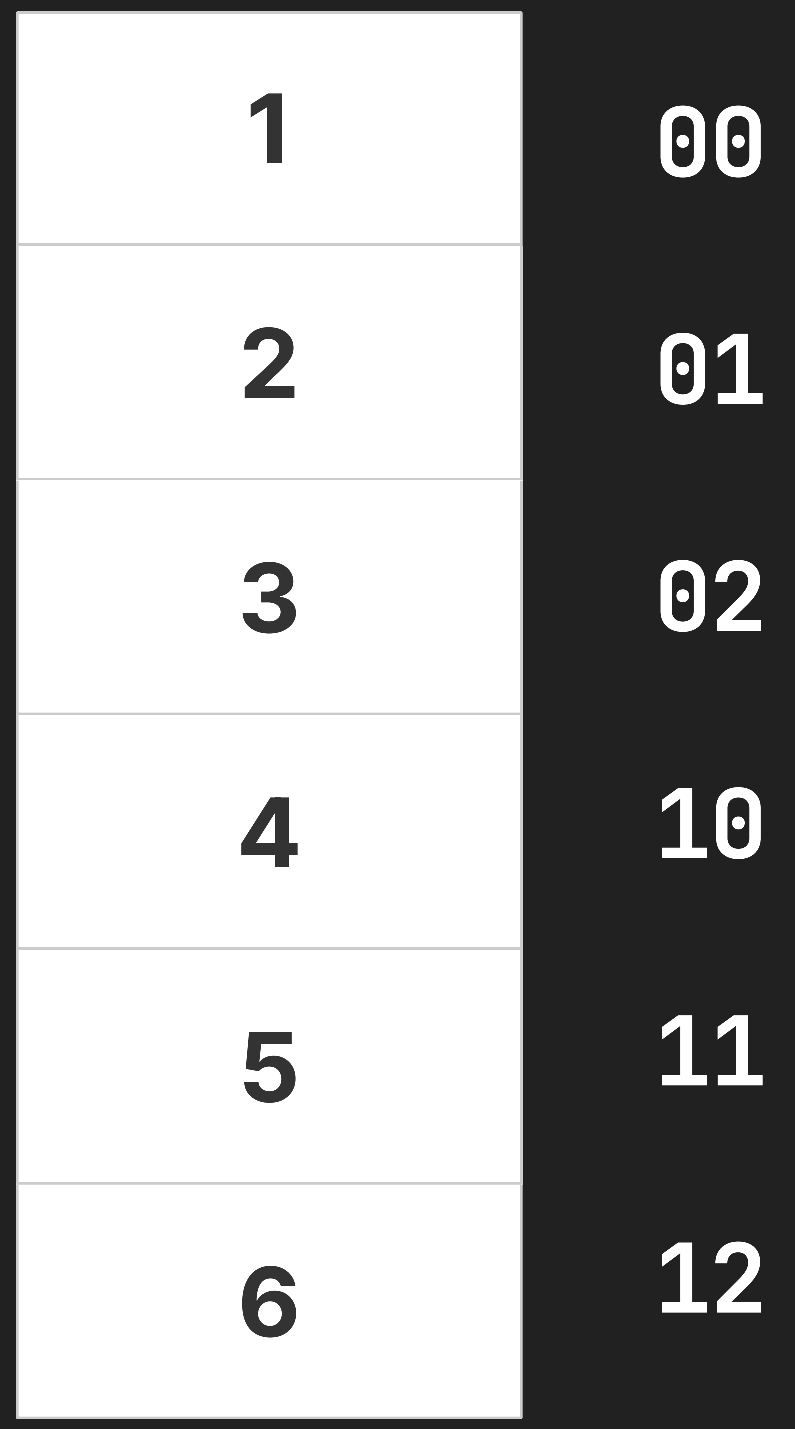
```
for(int i = 0; i < 5; i++)
    printf("%d ", a[i]);
```

2D Arrays

- Array of arrays (rows and columns)
- commonly used to represent tables and matrices.
- Syntax: `type arrayName[rows][cols];`
- Example:

```
int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
```





Two-dimensional array

Declaration

```
int mat[3][3];
```

Initialization

```
int mat[2][2] = {{1,2},{3,4}};
```

Accessing Elements

```
printf("%d", mat[1][0]);
```

Traversing 2D Array - Input / Output

Input

```
for(int i=0;i<2;i++){
    for(int j=0;j<2;j++)
        sacnf("%d", &mat[i][j]);
}
```

Output

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

Multi-dimensional array

An array having more than two dimensions is called a multidimensional array.

Example

```
int a[2][3][4];
```

Applications

- Scientific calculations
- 3D data representation
- Advanced simulations

Memory Representation

```
mat[0][0] mat[0][1]
mat[1][0] mat[1][1]
```

Accessing Elements

```
printf("%d", mat[1][0]);
```

Traversing 2D Array

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

Accessing Array Elements

- Use index: `arrayName[index]`
- Indexing starts at 0
- **Example:**

    -
      ```
      int arr[3] = {10, 20, 30};
      printf(“%d”, arr[1]); // Outputs 20
      ```

Activity

Array Basics

- **Task:** Declare an array of 5 integers
- Initialize with values `{10, 20, 30, 40, 50}`
- Print the 3rd element

Passing 1D-Arrays to Functions

- Arrays are passed by reference (implicitly)
- Example:

```
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        printf(“%d\n”, arr[i]);
    }
}
```

- Call: `printArray(numbers, 5);`

Hands-On

Sum of Array

- Task: Write a function `sumArray` that returns the sum of an array
- Input: `int arr[], int size`
- Test with array `{1, 2, 3, 4, 5}`

Modifying Arrays in Functions

- Changes to array in function affect original
  - Example:

```
void doubleArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
    {
       arr[i] *= 2;
    }
}
```

Modify Array

- **Task:** Write a function `incrementArray` that adds 1 to each element
- **Input**: `int arr[], int size`
- Test with array `{1, 2, 3}`

Accessing 2D Array Elements

- **Use:** `arrayName[row][col]`
- **Example:**

```
int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
printf("%d\n",matrix[2][2]); // Outputs 6
```

Activity

2D Array Basics

- Task: Declare a 2×3 array
- Initialize with `{{10, 20, 30}, {40, 50, 60}}`
- Print element at row 1, column 2

Passing 2D Arrays to Functions

- **Syntax:** `void func(int arr[][cols], int rows)`
- **Example:**

```
 void printMatrix(int arr[][3], int rows)
{
 for (int i = 0; i < rows; i++)
    for (int j = 0; j < 3; j++)
       printf(“%d\n”,arr[i][j]);
}
```

2D Array Sum

- Task: Write a function sumMatrix to compute sum of all elements
- Input: `int arr[][3], int rows`
- Test with `{{1, 2, 3}, {4, 5, 6}}`

Pointers

Introduction to Pointers

- Variables that store memory addresses
- Syntax: `type *pointerName;`
- Example: `int *ptr;`

Pointer Declaration & Initialization

- **Decalaration** : `int* ptr;`
- **Initialization:** `int x = 10; ptr = &x; // reference`
- **Dereferencing:** `int num = *ptr;`

Activity

Pointer Basics

- Task: Declare an integer and a pointer
- Assign address of integer to pointer
- Print value using dereference (`*ptr`)

Pointer Arithmetic

Pointer Arithmetic

- Increment/decrement moves pointer to next/previous memory location
- Example:

```
int arr[3] = {10, 20, 30};
int *ptr = arr;
printf(“%d\n”, *(ptr + 1);
```

Double Pointer

```c
#include <stdio.h>
int main()
{
    int x = 100;
    int *ptr1;
    ptr1 = &x;
    int **ptr2;
    ptr2 = &ptr1;
    printf("%d %d %d", x, *ptr1, **ptr2);
    return 0;
}
```

Strings & Pointers

Strings & Pointers

- Strings are arrays of characters ending with `‘\0’`
- Can be manipulated using pointers
- Example:

```
char str[] = "Hello";
char *ptr = str;
printf(“%c\d”, *(ptr + 1);
```

Hands-On

Length of String

Write a function strLength that computes the length of a string using pointer arithmetic (without using strlen), taking char *str as input.

```c
#include <stdio.h>
#define MAX 100
int main()
{
    char str[MAX];
    int length = 0, i = 0;

    printf("Enter a string : ");
    fgets(str, MAX, stdin);
    while (str[i] != '\0')
    {
        i++;
        length++;
    }
    length--;
    printf("The length is : %d", length);

    return 0;
}
```

Reverse a string

- **Algorithm :**
    - use string.h to find length

Write a function reverseString that uses pointers to reverse a string, and test it with the input "Hello" to produce the output "olleH".

```c
char str[100], temp;
int i, length;

printf("Enter a string: ");
fgets(str, sizeof(str), stdin);

length = strlen(str);

// Remove newline if present
if (str[length - 1] == '\n') {
    str[length - 1] = '\0';
    length--;
}

for (i = 0; i < length / 2; i++) {
    temp = str[i];
    str[i] = str[length - 1 - i];
    str[length - 1 - i] = temp;
}

printf("Reversed string: %s\n", str);
```

Debug the Code

- **Guess the output:**

```
int x = 10;
int *ptr = &x;
*ptr = 20;
printf(“%d\n”, x);
```

- **Explain why?**

Pointer Pitfalls

Common Pointer Pitfalls

- Uninitialized pointers: Cause undefined behavior
- Dangling pointers: Point to freed memory
- Always initialize pointers (e.g., nullptr)

Hands-On

Safe Pointer Usages

Write a function safeSwap using pointers that checks for nullptr before swapping values, and test it with both valid and null pointers.

Arrays & Functions

Combining Functions & Arrays

Example: Function to find max in array

```
int findMax(int arr[], int size) {
    int max = arr[0];
    for (int i = 1; i < size; i++)
        if (arr[i] > max) max = arr[i];
    return max;
}
```

Hands On

Find Minimum in Array

Write a function findMin that takes int arr[] and int size as inputs and returns the minimum value, tested with the array {5, 2, 8, 1, 9}.

Find Minimum in Array

```
Write a function findMin that takes int arr[] and int size as inputs and
returns the minimum value, tested with the array {5, 2, 8, 1, 9}.
```

Pointers & Arrays

Find Minimum in Array

- **Arrays and pointers are closely related**
- **Example:**

```
int arr[3] = {10, 20, 30};
int *ptr = arr;
cout << *(ptr + 2); // Outputs 30
```

Hands On

Find Minimum in Array

Write a function to print an array using pointer arithmetic with int *arr and int size as parameters, and test it with the array {1, 2, 3, 4}.