PHYS11 CH: Two Dimensional Arrays

Data Structures and Memory Organization

Mr. Gullo

Computer Science Department

February 25, 2025

Outline

Table of Contents

Learning Objectives

By the end of this lesson, you will be able to:

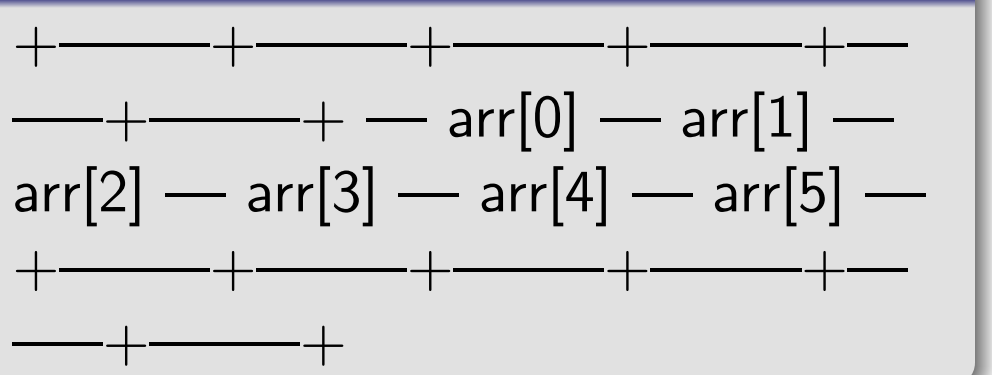
- Define and declare two-dimensional arrays in C++
- Understand how 2D arrays are represented in memory
- Access and manipulate elements in a 2D array
- Implement functions that work with 2D arrays
- Solve problems using 2D arrays

One Dimensional Arrays - Review

Declaration

```
int arr[6];
```

Memory Representation



- A one-dimensional array stores elements in a single row
- Each element is accessed with a single index
- Elements are stored contiguously in memory

Two Dimensional Arrays

Declaration

```
int arr[3][6];
```

We can think of this as:

- 3 rows of arrays
- Each row contains 6 elements
- Total of $3 \times 6 = 18$ elements

Memory Representation

```
[basicstyle=] Memory layout
(row-major order):  arr[0][0]
arr[0][1] arr[0][2] arr[0][3]
arr[0][4] arr[0][5] arr[1][0]
arr[1][1] arr[1][2] arr[1][3]
arr[1][4] arr[1][5] arr[2][0]
arr[2][1] arr[2][2] arr[2][3]
arr[2][4] arr[2][5]
```

Accessing 2D Array Elements

- Elements are accessed using two indices: [row][column]
- The first index (row) ranges from 0 to rows-1
- The second index (column) ranges from 0 to columns-1

Example

For `arr[3][6]`:

- Valid indices range from [0][0] to [2][5]
- `arr[1][4]` accesses the element in the 2nd row, 5th column
- `arr[0][2] = 42;` assigns the value 42 to the element in the 1st row, 3rd column

Be Careful!

`arr[3][6]` is an out-of-bounds access and will lead to undefined behavior!

Table of Contents

Limitations of 2D Arrays in C++

Important Limitation

When passing 2D arrays to functions, you **must** specify the number of columns!

Invalid

```
int someFunction(int arr[][]);
```

```
int someFunction(int arr[][COLS]);
```

- The number of columns must be a constant value
- The first dimension (rows) can be unspecified
- Typically, use a global constant or #define for COLS

Function Parameter Syntax

Function Declaration Examples

// Prints a 2D array void printArr2d(int arr[][NUM$_C$OLS], *introws*);
// Calculates the sum of elements in a specific row int rowSum(int arr[][NUM$_C$OLS], *introwNum*);
// Finds the maximum value in a specific row int rowMax(int arr[][NUM$_C$OLS], *introwNum*);
// Finds the row with the largest sum int maxRowSum(int arr[][NUM$_C$OLS], *introws*);

Table of Contents

Printing a 2D Array

Function Declaration

[basicstyle=] `void printArr2d(int arr[][NUM_COLS], int rows);` /* *Parameters* : *arr*, *a 2D array of size rows x cols Behavior* : *uses cout to output the array columns as a comma separated list for each row* */

Implementation

[basicstyle=] `void printArr2d(int arr[][NUM_COLS], int rows) for(int i = 0; i < ro`

Calculating Row Sum

Function Declaration

[basicstyle=] `int rowSum(int arr[][NUM_C OLS], int rowNum); /* Finds the sum of the elements of row Num Parame arr[][NUM_C OLS] - array with at least row Num + 1 rows row Num - the row to find the sum from 0 to number of rows - 1 */`

Implementation

[basicstyle=] `int rowSum(int arr[][NUM_C OLS], int rowNum) int sum = 0; for (int j = 0; j < NUM_C OLS; j++) sum += arr[rowNum][j]; return sum;`

Finding Row Maximum

Function Declaration

[basicstyle=] `int rowMax(int arr[][NUM_COLS], int rowNum);` /* Finds the largest element in the rowNum row Para
arr[][NUM_COLS] - array with at least rowNum+1 rows rowNum - the row to find largest value in the rowNum row No
rows are from 0 to number of rows - 1 */

Implementation

[basicstyle=] `int rowMax(int arr[][NUM_COLS], int rowNum) int max = arr[r
for (int j = 1; j < NUM_COLS; j + +) if (arr[rowNum][j] > max) max = arr[rowN
return max;`

Finding Maximum Row Sum

Function Declaration

[basicstyle=] `int maxRowSum(int arr[][NUM_COLS], int rows);` /* Finds the largest row sum Parameters : arr[][NUM_COLS] – 2D array of size rows * NUM_COLS row – the total number of rows */

Implementation

[basicstyle=] `int maxRowSum(int arr[][NUM_COLS], int rows)int maxSum = row`

```
for (int i = 1; i < rows; i++)
int currentSum = rowSum(arr, i);
if (currentSum > maxSum)  maxSum =
currentSum;
return maxSum;
```

Table of Contents

Treasure Map Problem

Problem Description

Given a grid (2D array) of numbers representing treasures:

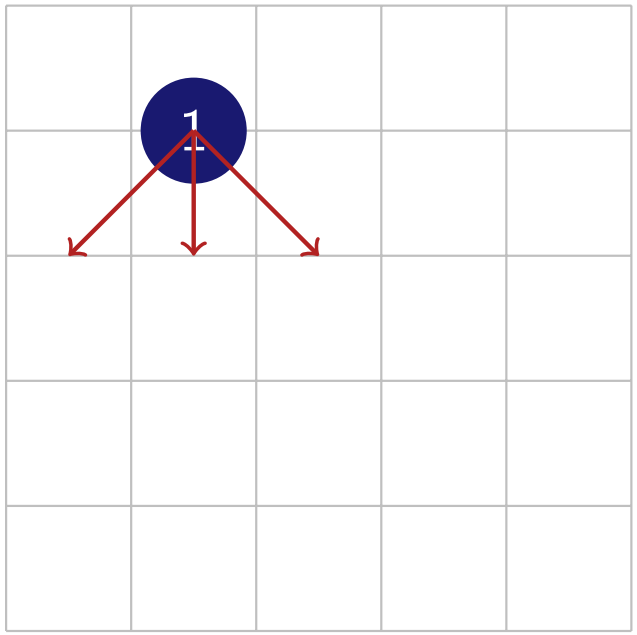
- Start at any cell in the top row
- Move down, but can only move to one of the three adjacent cells in the next row
- End at any cell in the bottom row
- Find the path with the maximum treasure sum

[basicstyle=] +---+---+---+---+---+ | 8 | 9 | 6 | 8 | 2 | +---+---+---+---+---+ | 1 | 8 | 7 | 6 | 3 | +---+---+---+---+---+ | 5 | 3 | 1 |10 | 5 | +---+---+---+---+---+ | 9 | 6 | 8 | 4 | 9 | +---+---+---+---+---+ | 7 | 1 | 9 | 7 | 7 | +---+---+---+---+---+ | 7 | 9 | 5 | 9 | 8 | +---+---+---+---+---+

Movement Rules in Treasure Map

Rules for Movement

- Start at any cell in the top row
- From a cell [i][j], can move to:
  - i+1 [j-1] (diagonal left)
  - i+1 [j] (directly below)
  - i+1 [j+1] (diagonal right)
- If at edge, can't move diagonally outside grid
- End at any cell in the bottom row



Maximum Path Solution

[basicstyle=]

```
+---+---+---+---+---+ | 8 |*9*|
6 | 8 | 2 | * = Path
+---+---+---+---+---+ | 1 | 8
|*7*| 6 | 3 |
+---+---+---+---+---+ | 5 | 3 |
1 |*10*| 5 |
+---+---+---+---+---+ | 9 | 6
|*8*| 4 | 9 |
+---+---+---+---+---+ | 7 | 1
|*9*| 7 | 7 |
+---+---+---+---+---+ | 7 |*9*|
5 | 9 | 8 |
+---+---+---+---+---+
```

Maximum Path: $9 + 7 + 10 + 8 + 9 + 9 = 52$

Dynamic Programming Approach

- Start with the top row (base case)
- For each row, calculate cumulative sums
- At each cell, choose the maximum of three possible paths from above
- Continue until we reach the bottom row
- Find the maximum value in the bottom row

Cumulative Sums Calculation

[basicstyle=]

```
+-----+-----+-----+-----+-----+ |
8 | 9 | 6 | 8 | 2 |
+-----+-----+-----+-----+-----+ |
1+9 | 8+9 | 7+9 | 6+8 | 3+8 |
+-----+-----+-----+-----+-----+ |
5+17 | 3+17 | 18 | 26 | 19 |
+-----+-----+-----+-----+-----+ |
31 | 28 | 34 | 30 | 35 |
+-----+-----+-----+-----+-----+ |
38 | 35 | 43 | 42 | 42 |
+-----+-----+-----+-----+-----+ |
45 | 52 | 48 | 52 | 50 |
+-----+-----+-----+-----+-----+
```

Calculation Method

For each cell [i][j] (except first row):

$$sum[i][j] = arr[i][j] + max(sum[i-1][j-1], sum[i-1][j], sum[i-1][j+1])$$

With boundary checks for edges.

Summary

Key Concepts

- Two-dimensional arrays store data in a grid-like structure
- Accessed using two indices: [row][column]
- When passing to functions, column size must be specified
- Common operations:
  - Traversing row by row
  - Finding row sums and maximums
  - Dynamic programming for pathfinding problems

Remember

- Valid indices range from [0][0] to [rows-1][cols-1]
- Always check array bounds to prevent errors
- Use descriptive function parameters for clarity