Multidimensional Arrays
ENEE 140

Prof. Tudor Dumitraș
Assistant Professor, ECE
University of Maryland, College Park

http://ter.ps/enee140

Today's Lecture

• Where we’ve been
  – Scalar data types (int, long, float, double, char)
  – Basic control flow (while and if)
  – Functions
  – Random number generation
  – Arrays and strings

• Where we’re going today
  – Multidimensional arrays

• Where we’re going next
  – Sorting
Nested Loops

• You can nest loops

```c
for (i=1; i<=3; i++) {
    for (j=1; j<=3; j++) {
        printf("%dx%d=%2d\t", i, j, i*j);
    }
    printf("\n"); // ready for next line
}
```

• Output

1x1= 1  1x2= 2  1x3= 3
2x1= 2  2x2= 4  2x3= 6
3x1= 3  3x2= 6  3x3= 9

Multi-Dimensional Arrays

• Two-dimensional arrays

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

– int array with 3 rows and 4 columns (12 elements)

– Think of this as 3 arrays with 4 elements each

```
```

• Working with 2D arrays

```c
a[0][0] = 0;  // access element on first row and first column
a[1][2] = 0;  // access element on row 1 and column 2
a[0][4] = 0;  // error: index out of bounds
a[3][0] = 0;  // error: index out of bounds
```

– Use 2D arrays to represent matrices

• Arrays with 3, 4, 5, etc. dimensions

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

– 3D array with 24 elements
Incremental Maintenance of Aggregates

- Sometimes, you must compute values that summarize multiple numbers (aggregates)
  - Examples: count, maximum, average
  - You can compute many aggregates incrementally, by updating a variable at each iteration of a loop

```c
int a, count = 0, max = INT_MIN;  // must initialize the aggregates
while (scanf("%d", &a) > 0) {
    count++;  // increment count
    if (max < a)  // update max
        max = a;
}
```

- How should you initialize the aggregate?

Backtracking

- General problem solving strategy

- Works on problems where:
  - You must search a large space of possible solutions
  - You can build the solution incrementally
  - You can check if the current partial solution is invalid (cannot possibly lead to a complete solution)
    - Typically, because it violates some constraints of the problem
  - You can enumerate all possible values for the current level (the current stage of the partial solution)
Backtracking: Key Idea

- Define four tests
  - `all_solved`: all levels have a solution
  - `none_solved`: none of the levels have a solution
  - `end_values`: have exhausted all possible values for current level
  - `is_valid`: the current partial solution doesn’t violate any constraints

- Solve the problem incrementally
  - Start by assigning the first possible value to the first level
  - On each level, try all the possible values, in order
  - If the solution is valid (`is_valid`), advance to the next level; otherwise, try the next value on the current level
  - If you cannot find any suitable value for the current level (`end_values`) return to the previous level (backtrack) and try the next value there
  - The search ends when all levels have a solution (`all_solved`) — complete solution
  - The search also ends when you have backtracked until no levels have a solution (`none_solved`). This means that the problem cannot be solved.

Example: The Eight Queens Puzzle

- Place 8 queens on a chess board so that no queen threatens another queen
  - 4,426,165,368 possible positions, 92 solutions

- Levels: rows on the chess board (cannot have more than one queen on a row)
- Partial solution: k queens placed on the first k rows of the board so that they don’t threaten each other (k<8)
- `all_solved`: have placed 8 queens
- `none_solved`: have not placed any queen
- `end_values`: have exhausted all possible columns for current row
- `is_valid`: no two queens on the same column or diagonal
Backtracking: General Design

Initialize position array
While ( ! all_solved )
    If ( end_values )
        Return to previous level (backtrack)
    If ( ! none_solved )
        Retrieve stored position and move to next one
    Else
        Exit loop (no more solutions to search)
    Else
        If (is_valid)
            Store position on current level
            Advance to next level
        Else
            Move to next position on current level

Review of Lecture

• What did we learn?
  – Nested loops
  – Multidimensional arrays
  – Incremental maintenance of aggregates

• Next week
  – Sorting

• Assignments for this week
  – Try to understand how the shellsort implementation from K&R Chapter 3.5 works; read Chapter 5.11 for how to use the library function qsort()
  – Weekly challenge: selection_sort.c
  – Homework: lab12.pdf (on http://ter.ps/enee140), due on Friday at 11:59 pm