

Introduction to Programming II

Multidimensional Arrays

Chris Brooks

Department of Computer Science
University of San Francisco

Department of Computer Science — University of San Francisco — p. 1/77

23-2: Multidimensional Arrays

- Many times, you want to have an array with more than one dimension.
 - A 2D game board.
 - An array of strings.
 - A bitmap representing a graphic object.
- In C, this is represented as an array of arrays.

Department of Computer Science — University of San Francisco — p. 2/77

23-3: Multidimensional Arrays

- We can statically allocate a multidimensional array like this:

```
int intArray[10][10];
int intArray[2][10] = { { 1,2,3,4,5,6,7,8,9,10},
                       {11,12,13,14,15,16,17,18,19,20} };
```
- We can then access particular cells like this:

```
printf("%d", intArray[0][3]);
intArray[1][2] = 6;
```

Department of Computer Science — University of San Francisco — p. 3/77

23-4: Exercise 1

- Write a C program that:
 - Declares a 10x10 2D array.
 - Builds a multiplication table in this array. (i.e. cell [i][j] should contain i * j)
 - prints out this table.

Department of Computer Science — University of San Francisco — p. 4/77

23-5: Arrays of pointers

- What if we don't know ahead of time how big our array should be?
- Then we need to use malloc to allocate memory on the fly.
- In this case, we treat our 2D array as an array of pointers (or, an array of arrays)

```
int **intArray;
```
- intArray is a pointer to an array of pointers.

Department of Computer Science — University of San Francisco — p. 5/77

23-6: Arrays of pointers

- We start out using malloc as usual (almost):

```
int **intArray = (int **)malloc(10 * sizeof(int *));
```
- We used malloc to create an array of 10 int pointers
- But, none of those pointers point to anything yet.
- We have to go through and use malloc to allocate space for each of those arrays as well.

```
for (i = 0; i < 10; i++)
    intArray[i] = (int *)malloc(10 * sizeof(int));
```

Department of Computer Science — University of San Francisco — p. 6/77

23-7: Exercise 2

- Now modify your multiplication-table program to:
 - Prompt the user for a number of rows and a number of columns.
 - Allocate the table of numbers with malloc.
 - (it should also do all the stuff it did before.)

23-8: Project 4: board struct

- Let's think a bit about what the board struct for project 4 should look like.
- It needs:
 - number of rows
 - number of columns
 - a representation of the board. (a two-dimensional array)

23-9: Project 4: board struct

- It should probably look something like:

```
typedef struct {
    int nrows;
    int ncols;
    int **boardArray;
} board;
```

23-10: CreateRandomBoard

- We're now ready to write createRandomBoard.
- It should take three arguments: a pointer to a board, a number of rows, and a number of columns.
- It should allocate memory for the boardArray and then fill in each cell with a random value (0 or 1).
- Write this function in Board.c, and then write a small main in gameOfLife.c that creates a Board and calls this function.

23-11: printBoard

- The next step is to write printBoard.
- This should take a board as input and print it out to the screen.
- I'd suggest printing '1' for cells containing 1, and '.' in cells containing 0.