2D Arrays

2025 Winter APS105: Computer Fundamentals

Jon Eyolfson

Lecture 20

1.0.2

Previously, We Made 1D Arrays

The syntax for declaring arrays and assigning values is:

```
<type> <name>[<array_size>] = {<comma_separated_values>};
```

Where you replace:

- `<type> <name> <array_size>` with the same rules as before
- `<comma_separated_values>` with the values you’d like to assign (in order)
  - The values must match the type of the array

Optionally, we can omit the `array_size`

```
<type> <name>[] = {<comma_separated_values>};
```

We Can Create 2D Arrays

The syntax for declaring 2D arrays is:

```
  <type> <name>[<first_size>][<second_size>];
```

Where you replace:

- `<type>` by the type for each value (or element) of the array
- `<name>` by a name you want to give the array (group of values)
- `<first_size>` by the number of arrays you want in the next dimension
- `<second_size>` by the number of elements in each array

Let’s Create a 2D Array That’s 2 by 3

We can declare:

```
int table[2][3];
```

If we want to think about rows and columns, this array is 2 rows and 3 columns

Just like 1D arrays, dimensions are 0-indexed, to access the element in row 1 column 2, we need to access `table[0][1]`

Same as Before, the Initial Values are Undefined

```c
#include <stdio.h>
#include <stdlib.h>

#define NUM_ROWS 2
#define NUM_COLS 3

int main(void) {
    int table[NUM_ROWS][NUM_COLS];
    for (int i = 0; i < NUM_ROWS; ++i) {
        for (int j = 0; j < NUM_COLS; ++j) {
            printf("table[%d][%d] = %d\n", i, j, table[i][j]);
        }
    }
    return EXIT_SUCCESS;
}
```

We Can Assign Values to Elements in the Declaration

Similar to 1D arrays, use curly brackets with values separated by commas

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

The first set of curly brackets are for the entire 2D array

- The inner set of curly brackets are for a “row”

We Can Assign Values to Elements in the Declaration

Similar to 1D arrays, use curly brackets with values separated by commas

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

The first set of curly brackets are for the entire 2D array
  The inner set of curly brackets are for a “row”

In this example: `table[0][1] = 2;`

We Could Omit the Inner Set of Curly Brackets

We could also initialize our 2D array as follows:

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

This produces:

```
table[0][0]: 1
table[0][1]: 2
table[0][2]: 3
table[1][0]: 4
table[1][1]: 5
table[1][2]: 6
```

We Could Omit the Inner Set of Curly Brackets

We could also initialize our 2D array as follows:

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

This produces:

```
table[0][0]: 1
table[0][1]: 2
table[0][2]: 3
table[1][0]: 4
table[1][1]: 5
table[1][2]: 6
```

This is because 2D arrays are stored in row-major order

Row-Major Order: Elements of a Row are Beside Each Other

We *could* represent a 2D array using a plain 1D array

Assuming we have an `int` array, `table`, we can index elements:

```
  table[rowIndex * NUM_COLS + colIndex]
```

Row-Major Order: Elements of a Row are Beside Each Other

We *could* represent a 2D array using a plain 1D array

Assuming we have an `int` array, `table`, we can index elements:

```
  table[rowIndex * NUM_COLS + colIndex]
```

The following is equivalent:

```
  table[0][0] → table[0]
  table[0][1] → table[1]
  table[0][2] → table[2]
  table[1][0] → table[3]
  table[1][1] → table[4]
  table[1][2] → table[5]
```

Row-Major Order: Elements of a Row are Beside Each Other

We *could* represent a 2D array using a plain 1D array

Assuming we have an `int` array, `table`, we can index elements:

```
  table[rowIndex * NUM_COLS + colIndex]
```

The following is equivalent:

```
  table[0][0] → table[0]
  table[0][1] → table[1]
  table[0][2] → table[2]
  table[1][0] → table[3]
  table[1][1] → table[4]
  table[1][2] → table[5]
```

`&table[i][j]` is also the same as `table + i * NUM_COLS + j`

We Can Only Let C Determine the Number of Rows

We cannot do:

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

C needs to know the number of columns to calculate the number of rows:

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

We Can Only Let C Determine the Number of Rows

We cannot do:

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

C needs to know the number of columns to calculate the number of rows:

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

You can only omit the first dimension for multidimensional arrays

The Rest of a Row Gets Filled with 0s

We can do:

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

If we output we'll see:

```
table[0][0]: 1
table[0][1]: 2
table[0][2]: 0
table[1][0]: 4
table[1][1]: 5
table[1][2]: 0
```

The Rest of a Row Gets Filled with 0s

We can do:

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

If we output we'll see:

```
table[0][0]: 1
table[0][1]: 2
table[0][2]: 0
table[1][0]: 4
table[1][1]: 5
table[1][2]: 0
```

Note: this is the same for plain arrays, if you initialize at least one value, the rest of the **known** size is filled with 0s

We Can Have Dynamically Sized Arrays on the Stack

As long as the C compiler isn’t ancient, we can avoid some uses of `malloc`

```
#include <stdio.h>
#include <stdlib.h>

int inputLength(void) {
    int length = 0;
    do {
        scanf("%d", &length);
    } while (length <= 0);
    return length;
}

int main(void) {
    int arrayLength = inputLength();
    int array[arrayLength];
    return EXIT_SUCCESS;
}
```

All Array Rules Apply to the First Dimension (This is Valid)

```c
#include <stdio.h>
#include <stdlib.h>

#define ARRAY_LENGTH(array) (sizeof((array))/sizeof((array)[0]))
#define NUM_COLS 3

int main(void) {
    int table[][NUM_COLS] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    int numRows = ARRAY_LENGTH(table);
    for (int i = 0; i < numRows; ++i) {
        for (int j = 0; j < NUM_COLS; ++j) {
            printf("table[%d][%d]: %d\n", i, j, table[i][j]);
        }
    }
    return EXIT_SUCCESS;
}
```

You Can Use Dynamic Lengths in Function Arguments

You could write a function prototype as:

```
void foo(int numRows, int numCols, int table[][numCols]);
```

However, the variable used in the multidimensional array must be declared before the array itself

You Can Use Dynamic Lengths in Function Arguments

You could write a function prototype as:

```
void foo(int numRows, int numCols, int table[][numCols]);
```

However, the variable used in the multidimensional array must be declared before the array itself

You cannot write:

```
void foo(int numRows, int table[][numCols], int numCols);
```

You could optionally write:

```
void foo(int numRows, int numCols, int table[numRows][numCols]);
```

For 2D Arrays, All Columns Need to be the Same Length

If we want the columns to be different sizes,
we have to use `malloc` with a different approach

We create an array of pointers,
one pointer for each row,
and `malloc` elements for the row

Example of Using an Array of Pointers

```
int main(void) {
→   printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```

----- main -----

Heap **Stack**

Example of Using an Array of Pointers

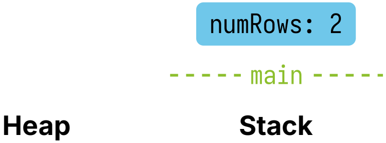
```
int main(void) {
    printf("Number of rows: ");
→   int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```

----- main -----

Heap **Stack**

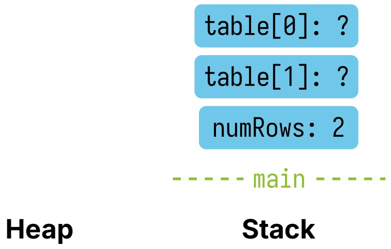
Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
→   int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



table[0]: ?

table[1]: ?

numRows: 2

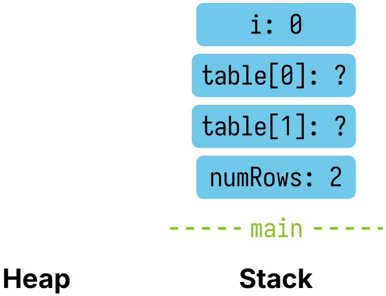
----- main -----

Heap

Stack

Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
→     printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



i: 0

table[0]: ?

table[1]: ?

numRows: 2

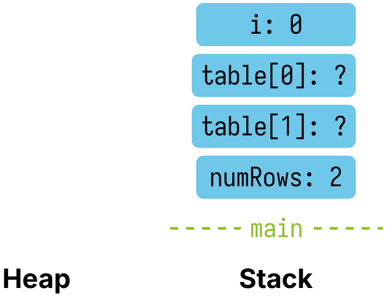
----- main -----

Heap

Stack

Example of Using an Array of Pointers

```c
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



i: 0

table[0]: ?

table[1]: ?

numRows: 2

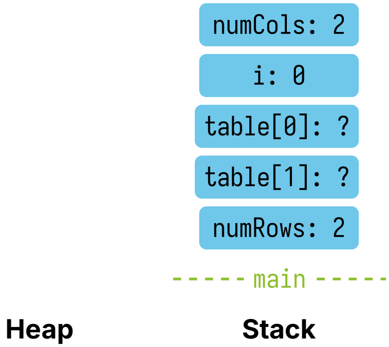
----- main -----

Heap

Stack

Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



numCols: 2

i: 0

table[0]: ?

table[1]: ?

numRows: 2

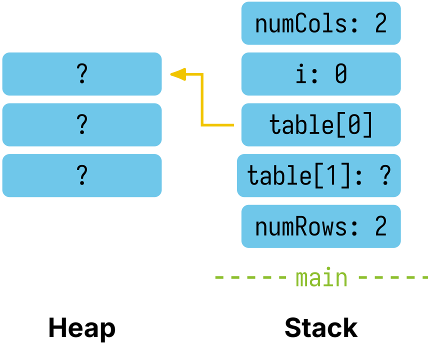
----- main -----

Heap

Stack

Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



numCols: 2

i: 0

table[0]

table[1]: ?

numRows: 2

?

?

?

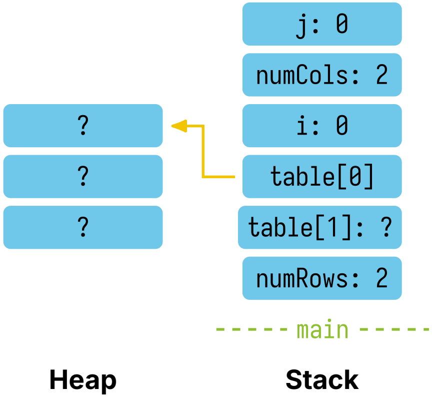
- - - - - main - - - - -

Heap

Stack

Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



j: 0

numCols: 2

?

i: 0

?

table[0]

?

table[1]: ?

numRows: 2

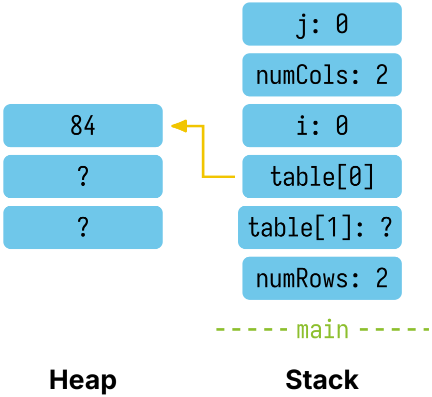
----- main -----

Heap

Stack

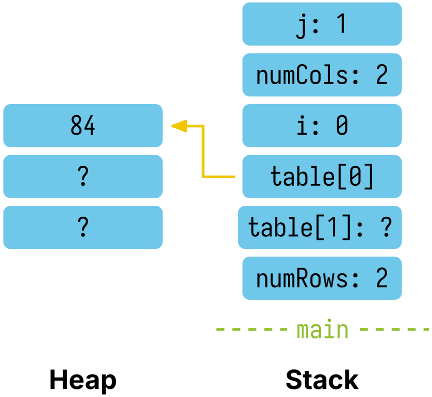
Example of Using an Array of Pointers

```c
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



j: 1

numCols: 2

i: 0

table[0]

table[1]: ?

numRows: 2

----- main -----

84

?

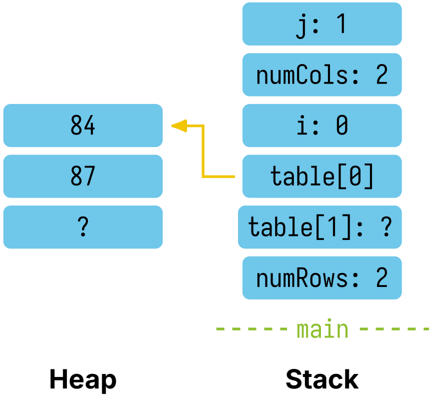
?

Heap

Stack

Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



84

87

?

j: 1

numCols: 2

i: 0

table[0]

table[1]: ?

numRows: 2

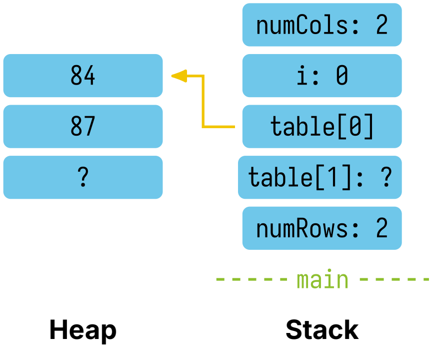
- - - - - main - - - - -

Heap

Stack

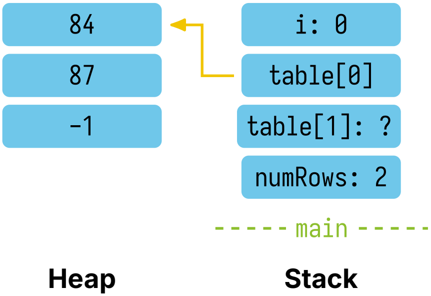
Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



84

87

-1

i: 0

table[0]

table[1]: ?

numRows: 2

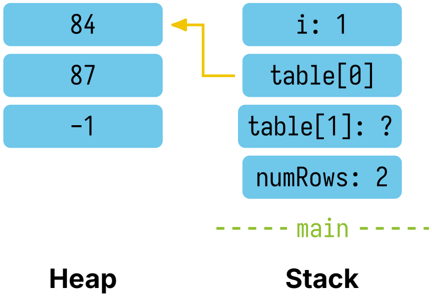
----- main -----

Heap

Stack

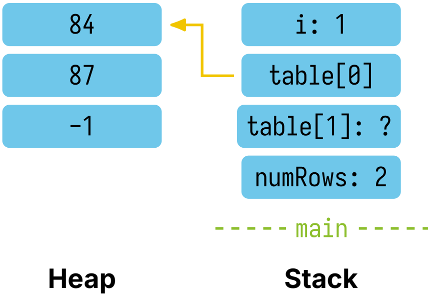
Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
→     printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



84

87

-1

i: 1

table[0]

table[1]: ?

numRows: 2

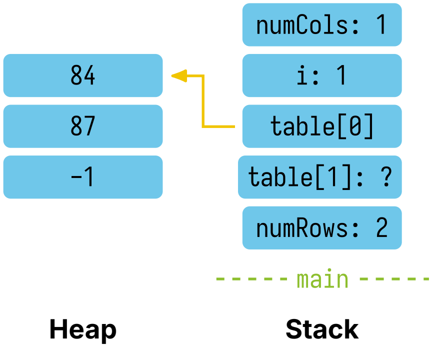
main

Heap

Stack

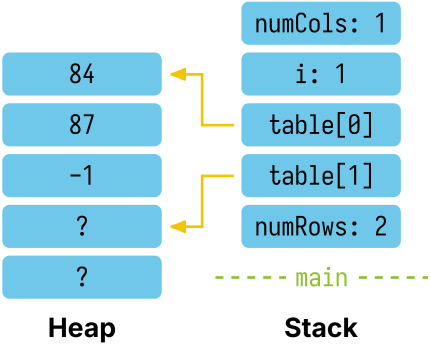
Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



84
87
-1
?
?

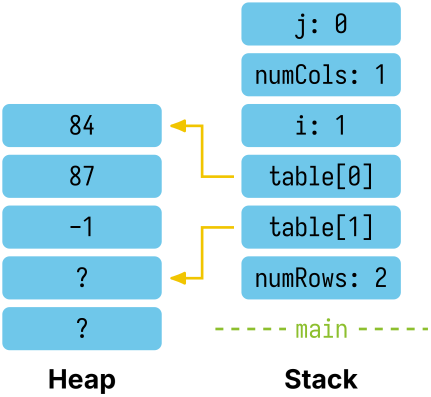
numCols: 1
i: 1
table[0]
table[1]
numRows: 2
----- main -----

Heap

Stack

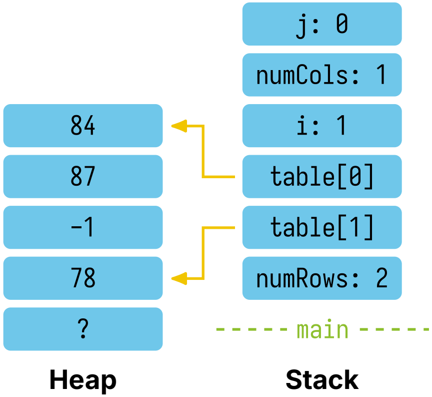
Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



84
87
-1
78
?

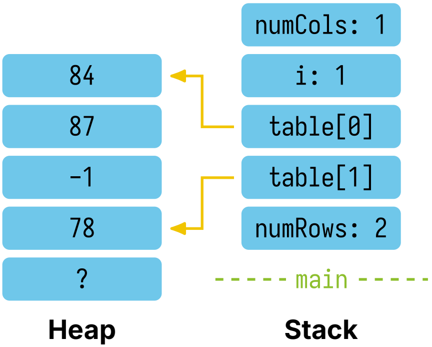
Heap

j: 0
numCols: 1
i: 1
table[0]
table[1]
numRows: 2
----- main -----

Stack

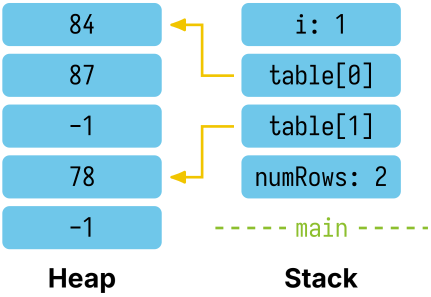
Example of Using an Array of Pointers

```c
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



Example of Using an Array of Pointers

```c
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



84

87

-1

78

-1

i: 1

table[0]

table[1]

numRows: 2

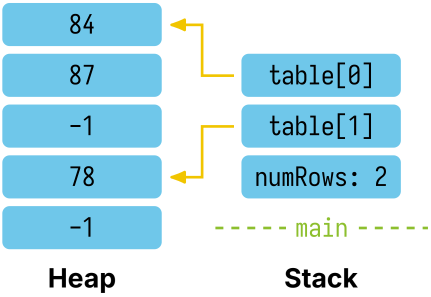
main

Heap

Stack

Example of Using an Array of Pointers

```
int main(void) {
    printf("Number of rows: ");
    int numRows = inputLength();
    int *table[numRows];

    for (int i = 0; i < numRows; ++i) {
      printf("Number of columns: ");
      int numCols = inputLength();
      table[i] = malloc(
        sizeof(int) * (numCols + 1)
      );
      for (int j=0; j<numCols; ++j) {
        table[i][j] = rand() % 100 + 1;
      }
      table[i][numCols] = -1;
    }
    return EXIT_SUCCESS;
}
```



84

87

-1

78

-1

Heap

table[0]

table[1]

numRows: 2

main

Stack

The Previous Example Isn’t a True Multidimensional Array

In true multidimensional arrays:

`table[i][j];` is the same as `table[i * NUM_COLS + j];`

However, what we did in the previous example was:

```
int *row = table[i];
int element = row[j];
```

The Previous Example Isn’t a True Multidimensional Array

In true multidimensional arrays:
`table[i][j];` is the same as `table[i * NUM_COLS + j];`

However, what we did in the previous example was:

```
int *row = table[i];
int element = row[j];
```

The declaration of `table` (if we used `malloc`) could be: `int *table[];`
However, we could also write it as: `int **table;`

Our first double pointer!

Exercise (Time Permitting): Tic-Tac-Toe

Please check the YouTube recording!