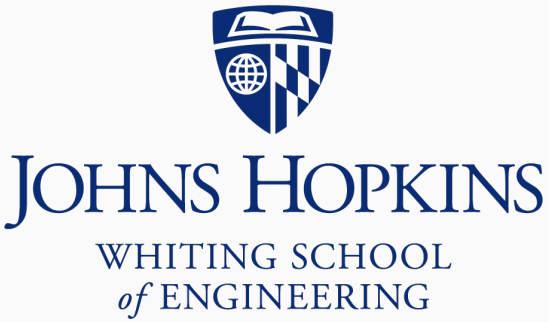
Beyond 1D arrays

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

Beyond 1D arrays

We've already seen `char* argv[]`: an array of strings

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    printf("argc = %d\n", argc);
    for(int i = 0; i < argc; i++) {
        printf("argv[%d] = %s\n", i, argv[i]);
    }
    return 0;
}
```

```
$ gcc args_eg_1.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out rosebud
argc = 2
argv[0] = ./a.out
argv[1] = rosebud
```

Arrays of strings

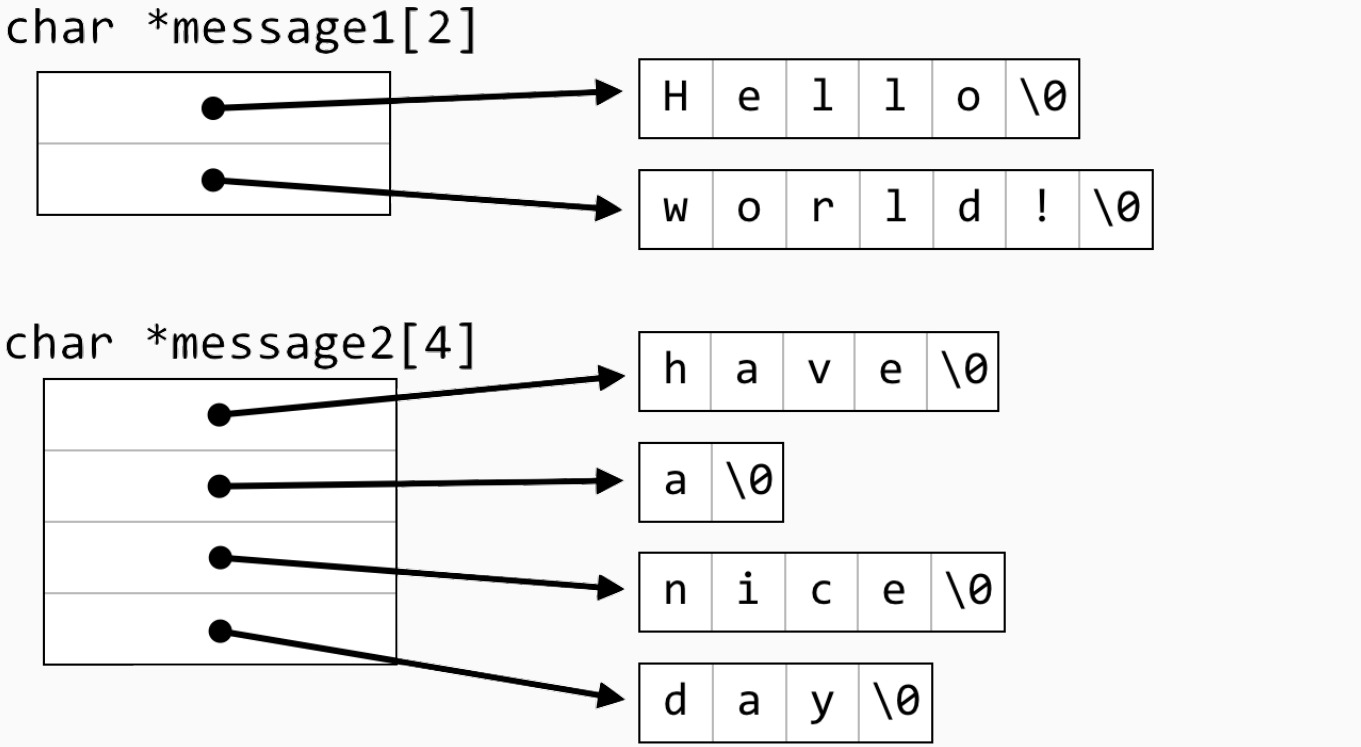
```c
#include <stdio.h>

void print_list(char **message, const int num) {
    for(int i = 0; i < num; i++) {
        printf("%s ", message[i]);
    }
    putchar('\n');
}

int main() {
    char *message1[] = { "Hello", "world!" };
    char *message2[] = { "Have", "a", "nice", "day" };
    print_list(message1, 2);
    print_list(message2, 4);
    return 0;
}
```

Arrays of strings

```
$ gcc array_of_strs.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
Hello world!
Have a nice day
```

Arrays of strings

```
char *message1[] = { "Hello", "world!" };
char *message2[] = { "Have", "a", "nice", "day" };
```



Two-dimensional arrays

Say we want to represent a tic tac toe board

O	-	X
-	O	-
X	-	X

An array seems appropriate, but we want a *two-dimensional* array

For each element, we'll use a char: (`'X'`, `'O'` or `'-'`)

Two-dimensional arrays

Option 1:

```
char board[9];
```

board is a *one-dimensional* array with 9 elements, but we use it to represent a 3x3 board

To access element at row `i`, column `j`, use `board[i * 3 + j]`

0	1	2	O	-	X
3	4	5	-	O	-
6	7	8	X	-	X

This is *row-major* order; it is common in practice

Two-dimensional arrays

```c
#include <stdio.h>

void print_board(char board[]) {
    for(int i = 0; i < 3; i++) {      // rows
        for(int j = 0; j < 3; j++) {  // columns
            printf("%c ", board[i * 3 + j]);
        }
        putchar('\n');
    }
}

int main() {
    char board[9] = {'O', 'O', 'X',
                     '-', 'O', '-',
                     'X', '-', 'X'};
    print_board(board);
    return 0;
}
```

Two-dimensional arrays

```
$ gcc ttt_ex.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
O O X
- O -
X - X
```

Two-dimensional arrays

Option 2:

```
char board[3][3];
```

board is a *two-dimensional* array with 3 rows and 3 columns

C "understands" the rows and columns; to access element at row `i`, column `j`, use `board[i][j]`

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]
[2][0]	[2][1]	[2][2]

O	-	X
-	O	-
X	-	X

Two-dimensional arrays

```c
#include <stdio.h>

void print_board(char board[][3]) {
    for(int i = 0; i < 3; i++) {      // rows
        for(int j = 0; j < 3; j++) {  // columns
            printf("%c ", board[i][j]);
        }
        putchar('\n');
    }
}

int main() {
    char board[3][3] = {{'O', 'O', 'X'},
                        {'-', 'O', '-'},
                        {'X', '-', 'X'}};
    print_board(board);
    return 0;
}
```

Two-dimensional arrays

```
$ gcc ttt_ex_2.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
O O X
- O -
X - X
```

Two-dimensional arrays

Type of the array parameter is important; we *must* specify the parameter type along with

```
void print_board(char board[][3]) {
    // ...
}
```