CS 449 – Lecture 4

Dr. Jack Lange

String intialization

- Static initialized strings
  - `char * str = "some text";`
  - `char name[] = { 'J', 'a', 'c', 'k', '\0'};`
- Initialize strings at runtime

```
char name[256];
strcpy(name, "Jack");
```

-
- Examples
  - string-constant.c
  - string-var-1.c
  - real-string.c

2 Dimensional Arrays

- `int x[][];`
- Memory is a 1 dimensional array
  - … so, 2D arrays are still 1D arrays
  - Can still access them as 1D arrays if you want
- 2D arrays exist for convenience of access

Passing 2D arrays to functions

- Remember: Arrays ([]) == pointers (*)
- So do 2D arrays ([][]) == double pointers (**)?
  - **NO**
- **2D arrays are pointers also!!!**
  - There is no metadata associated with a 2D array
  - So the name of a 2D array just points to its start location in memory
  - **The function you call must specify the # of columns**
- Example: 2d-array.c

Command line arguments

- Command line arguments are passed to main() via argv
  - `int main(int argc, char * argv[]);`
  - `int main(int argc, char ** argv);`
- `argv` is an array of strings, and strings are arrays of characters
  - So is `argv` a 2D array? **NO**
- `argv` is an array of character pointers
  - *The string data is not included in the array*
  - **Important:** The left of of an array declaration is its type
- If you can understand the layout of `argv`, then you understand pointers
  - **Example**: cmdargs.c

Using arrays

- Biggest problem with arrays: **Buffer Overflows**
- Easy way to ensure you never have a buffer overflow:
  - Ensure you never write more data than you have space for
- Avoiding buffer overflows is not difficult
  - It just requires that you not be lazy…
- **DO NOT USE INPUT FUNCTIONS THAT DO NOT ALLOW LIMITING THE SIZE OF DATA!!!**
  - Use strn*() and **not** str*()
  - Use fgets() + sscanf() and **not** scanf()