Computer Systems

CS107

Cynthia Lee

Today's Topics

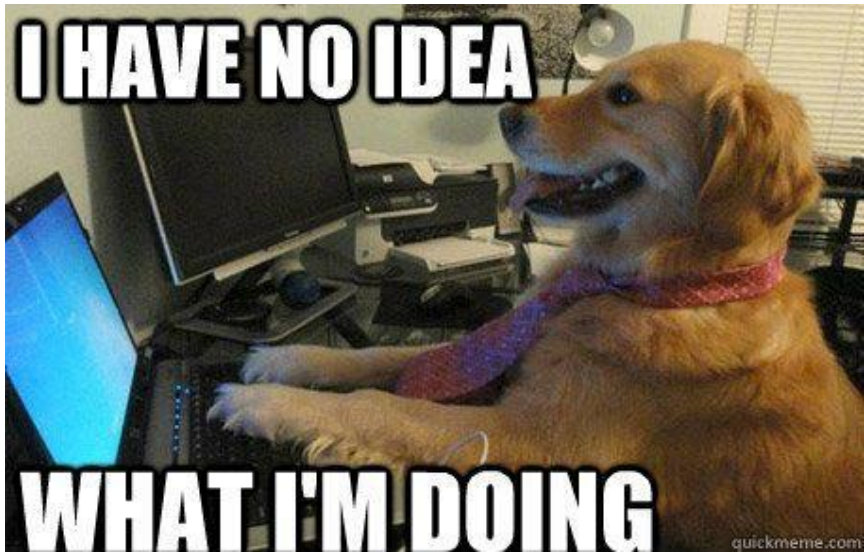
- Your first C program, your first command-line compilation
- Pointers and arrays
  - Review from CS106B/X, but digging deeper
- Strings

NEXT LECTURE:

- More on C pointers and arrays, pointer arithmetic, files

- **For today (optional):**
  - If you have a laptop with you (totally optional), you may want to connect to myth
  - This will allow you to type along with me as we write our first C program
  - `cp -r /afs/ir/class/cs107/samples/lect2/ ~`

C Programming: Getting Started



It's ok to feel like this right now when it comes to Unix. We'll continue to work on that while we introduce C. It's important you spend as much time as you can building muscle memory with the tools.

History and background of C

- Birthdate around 1970
- Created to make writing Unix (the OS itself) and tools for Unix easier
- Part of the C/C++/Java family of languages
  - (with C++ and Java coming later)
- Design principles:
  - Small, simple abstractions of hardware
  - Minimalist aesthetic
  - C is much more concerned with efficiency and minimalism than safety (Java) or convenient high-level services and abstractions (Java, C++)

C

C++

Java

Comparison of C, Java, C++

- **Some things will be very familiar:**
  - Syntax
  - Basic data types
  - Arithmetic, relational, and logical operators
- **You may be sad about what's missing:**
  - No power features of C++ (overloading operators, default arguments, pass by reference, classes/objects, fancy ADTs)
  - Thin standard libraries (no graphics, networking, etc)
  - Weak compiler checks, almost no runtime checks
- **Benefits:**
  - Small language footprint (not much to learn)
- **Philosophical difference:**
  - Procedural (C)
  - Procedural + Objects (C++)
  - Object-Oriented (Java)

Things to watch for in our live coding of hello.c:

- **First pass:**
  - Structure of C program
    - #include
    - main, argc, argv
- Unix commands to compile, run program
  - Makefile, make command
  - Data types: int, char, char* strings, arrays
- Interactive output and input
  - printf
  - format codes (%s, %d, %f)

- **Second pass (new features):**
  - making global constant with #define
  - scanf
- C runtime errors (segmentation fault)
- CTRL-C to kill something in Unix

Basic anatomy of main()

```
int main(int argc, char * argv[])
{
   // stuff
   return 0;
}
```

- Return value always int (just return 0 all the time and otherwise ignore it)
- argc is the size of the argv array
- argv array is a collection of the arguments that are typed on the command line in Unix when you run the program (captured as strings)
  - The 0th argument is the name of the command itself
  - Args 1 and on are the arguments

printf()

```
// like System.out.print() or cout
printf("Hello, world!");

// like System.out.println() or cout << … << endl
printf("Hello, world!\n");
```

Escape sequence	Meaning
\n	Newline
\\	\ (single backslash)
\t	Tab

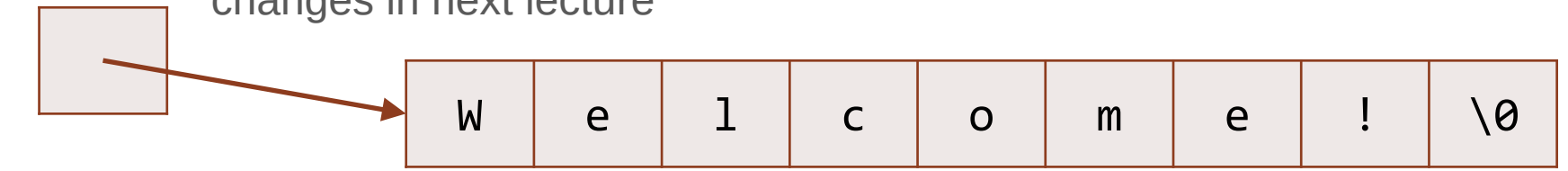
strings

- **Unlike C++, there is no "string" class** (no classes at all in C!)
- Strings are arrays of individual characters (char):

W	e	l	c	o	m	e	!	\0

  - *Always must end with special null terminating character `\0`!*

- Can be declared as:
  - `char * str`
  - `char str[30]`
  - These are "sort of" interchangeable—see below that the memory diagrams look slightly different. We will learn more about what changes in next lecture



#include <string.h>

- Some useful string functions:
  - `strcat(str1, str2)  // concat str2 to the end of str1`
  - `strcmp(str1, str2)  // returns 0 if strings are equal,`
    `// otherwise -1 or 1, for < or >`
  - `strdup(str)  // returns a new (malloc'ed) copy of str`
  - `strcpy(str1, str2)  // copies contents of str2 to str1`
  - `strlen(str)  // finds the length of a str`
  - `strstr(str1, str2)  // returns a ptr to the first occurrence`
    `// of str2 in str1`

String library functions and the importance of: (1) having enough space (2) null character

- Some useful string functions:
  - `strcat(str1, str2)  // concat str2 to the end of str1`

```
char str1[9];
```

str1:

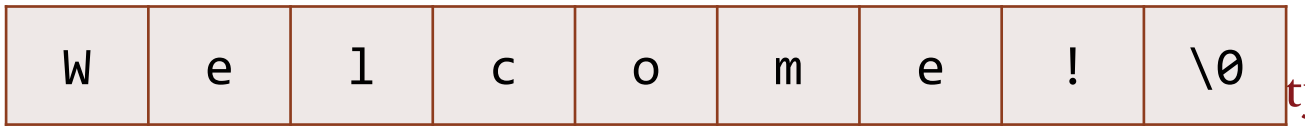
```
strcpy(str1, "We");
```

str1:


```
int len = strlen(str1);   // this is 2, not 3!

strcat(str1, "lcome!");
```

str:



String library functions and the importance of: (1) having enough space (2) null character

- Some useful string functions:
  - `strcat(str1, str2)  // concat str2 to the end of str1`

- **Your turn!** Which of these two (or both) could give a crash/error, and why?

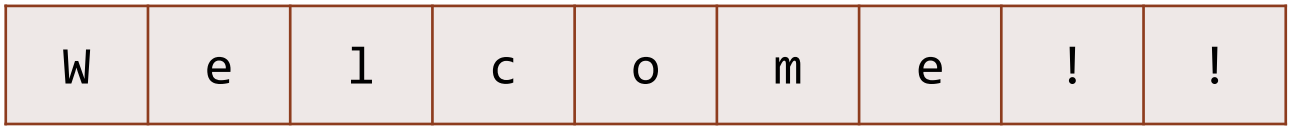
```
char str1[9];
```

str1:

```
strlen(str1);
```

```
strcat(str1, "lcome!!");
```

str:



\0

Things to watch for in our live coding of hello.c:

- **First pass:**
  - Structure of C program
    - #include
    - main, argc, argv
- Unix commands to compile, run program
  - Makefile, make command
  - Data types: int, char, char* strings, arrays
- Interactive output and input
  - printf
  - format codes (%s, %d, %f)

- **Second pass (new features):**
  - making global constant with #define
  - scanf
- C runtime errors (segmentation fault)
- CTRL-C to kill something in Unix

scanf()

```
char name[10];
int age, height;
printf("Enter your first name, age, and height: ");
scanf("%s %d %d", name, &age, &height);
printf("Name: %s\tAge:%d\tHeight:%d\n", name, age, height);
```

Pattern	Use	Type of variable
%d	Integer	int
%c	A single character	char
%f, %lf	Non-integer number	float, double
%s	Whitespace-separated string	char * or char[]

Passing an Array to a Function

(Code demo)

Starter code (needs work)

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

double sum(double arr[])
{
    double total = 0;
    /* loop over array and sum */

    return total;
}

int main(int argc, char *argv[])
{
    double arr[] = {1.1, 2.2, 3.3, 4.4};
    double total = 0.0;

    /* want to call sum to calculate total of array values */
    total = sum(arr);

    printf("Sum = %g\n", total);

    return 0;
}
```

Key points from the code example:

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

double sum(double arr[], int length)
{
    double total = 0.o;
    for (int i=0; i<length; i++)
        total += arr[i];
    return total;
}

int main(int argc, char *argv[])
{
    double arr[] = {1.1, 2.2, 3.3, 4.4};
    double total = 0.0;

    /* want to call sum to calculate total of array values */
    total = sum(arr, 4);

    printf("Sum = %g\n", total);

    return 0;
}
```

- double arr[] and double *arr are equivalent for parameter types
  - Not quite true for local variable declarations
- Any time we pass an array ([] or * notation), we need to also pass along its accompanying array size
  - They're always a pair
  - **Example:** argc to go with argv!