Computer Systems

CS107

Cynthia Lee

Today’s Topics

- Basic structure of main function in C
- Strings
- Pointers and arrays
  - Review from CS106B/X, but digging deeper

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 Basics

Main function

Printf()

Scanf()

Stanford University

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");
// like cout << "Hello, " << name << "!" << endl
printf("Hello, %s!\n", name);
```

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

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[]

scanf()

```
int age
double gpa;
printf("Enter your age and GPA like this (age,GPA): ");
scanf("(%d,%g)", &age, &gpa); // ex: (19,3.57)
printf("Your age is: %d and GPA is:%g\n", age, gpa);
```

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[]

C Basics: strings

strings

- **<u>Unlike C++, there is no “string” class</u>** (no classes at all in C!)
- Strings are arrays of individual characters (type `char`):

W	e	l	c	o	m	e	!	\0

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

- Can be declared in two ways:

  1. `char * str;`

  › This way creates a pointer that we can use to point to an array of characters that was created and set to a value elsewhere.

  2. `char str[30];`

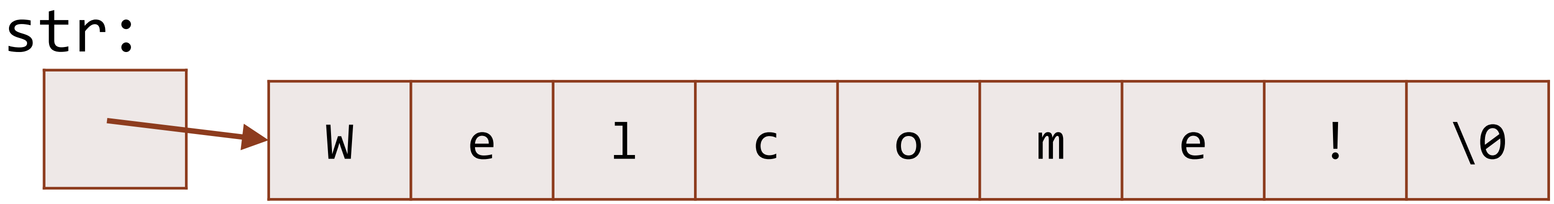
  › This way creates an array of characters that we can use to store a string when we set its values to be characters (ending in `'\0'`)

strings

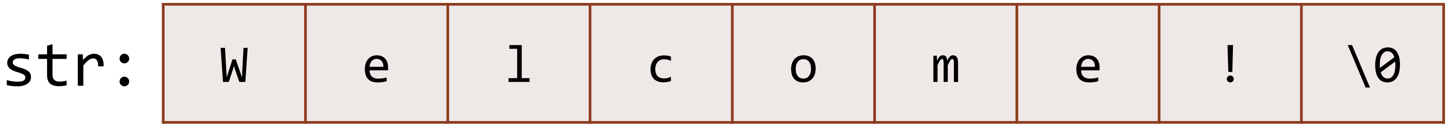
- `char * str;`
- `char str[30];`

- These two ways of declaring are *almost* interchangeable—see below that the memory diagrams look slightly different. We will learn more about the subtleties of these changes in next lecture.

```
char * str;
// additional code that creates an array and writes Welcome in it
```



```
char str[30];
// additional code that writes Welcome in the array
```



#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 >
strncmp(str1, str2, N)// like strcmp, but only checks the
                     // first N chars (at most)
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
strchr(str1, ch)     // returns a ptr to the first occurrence
                     // of ch in str1 (put single quotes
                     //around ch since it’s just one char
strdup(str)          // returns a new (malloc’ed) copy of str
```

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

- `strcpy`
- `strcat`
- `strlen`

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

str1:	?	?	x	y	z	?	?	?	?

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

str1:	W	e	\0	y	z	?	?	?	?

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

str1:	W	e	l	c	o	m	e	!	\0

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 codes (or both) could give an error, and why?

ERROR

uninitialized memory

```
char str1[9];
strlen(str1);              // OPTION A: does this cause an error?
```

str1:									

int x;

```
char str2[9];
strcpy(str2, "We");
strcat(str2, "lcome!!"); // OPTION B: does this cause an error?
```

int y;

ERROR

str2:	W	e	l	c	o	m	e	!	!	\0

Passing an Array to a Function

(CODE DEMO)

Stanford University

Starter code (needs work)

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

double sum(double arr[])
{
    double total = 0.0;
    /* 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 = 0.0;
    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!