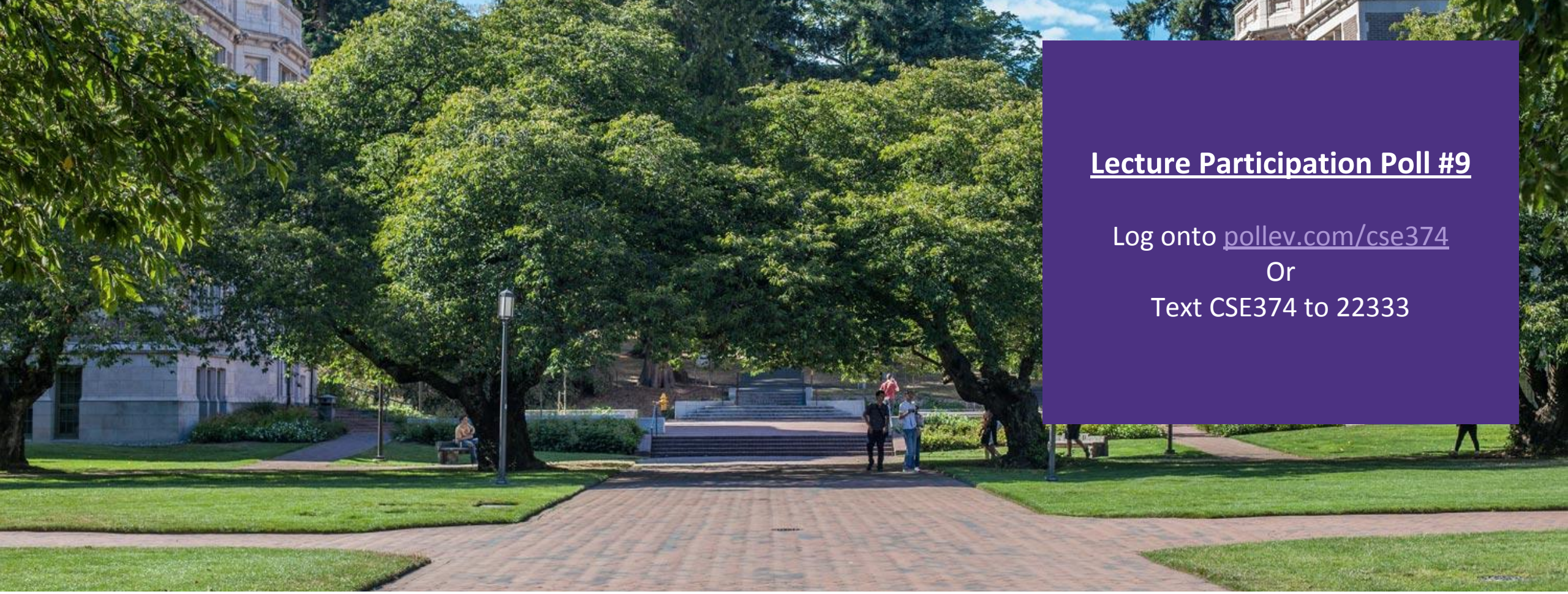


Lecture Participation Poll #9

Log onto pollev.com/cse374
Or
Text CSE374 to 22333

Lecture 9: C Pointers

CSE 374: Intermediate Programming Concepts and Tools

Administrivia

Assignments

Where do computers store data?

- CPU – Central Processing Unit – computer circuitry that followed computer instructions in assembly
- RAM – Random Access Memory – a computer's short-term memory where data is stored during program operation
  - When a program ends the memory in use "goes away"
- Hard disc storage – a computer's long-term memory, this is where data is stored when you need to preserve it across re-starts.
  - Data is stored indefinitely
  - Can be modified by different processes

How do computers store data?

- Large sequences of numbers
  - Numbers are representations for electrical switches "transistors" that make up the brains of the CPU
- All data is binary – 1s and 0s
  - A single digit is called a "bit"
  - Bits come in groups of 8 called "bytes"
  - All instructions can be translated into sequences of binary

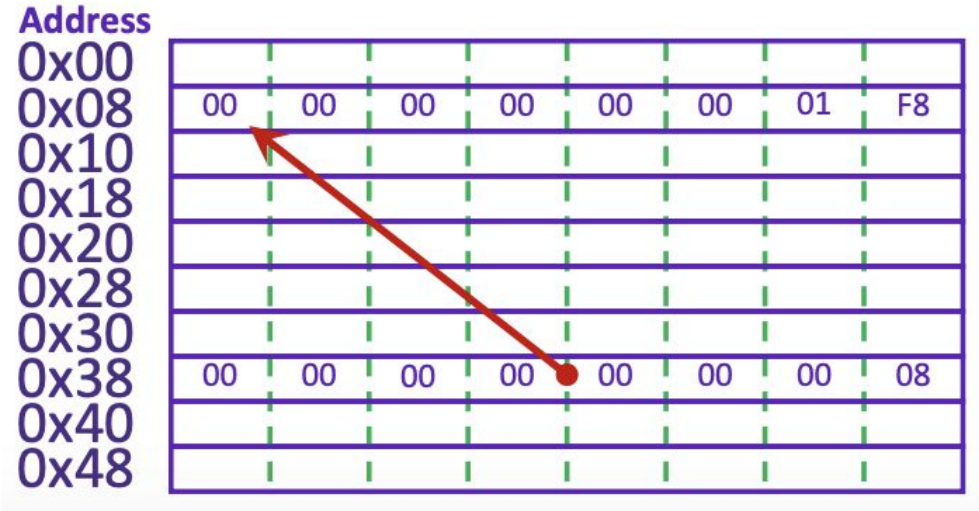
english	h	e	l	l	o
ascii	104	101	108	108	111
binary	01101000	01100101	01101100	01101100	01101111

- Numbers represent other types of data
  - ASCII – each byte represents a letter of the English alphabet
  - Unicode – similar encoding structure to ASCII but covers a wider range of characters including non-English characters, emojis etc…
  - Images – represented by a 2D array of "pixels"
    - Each pixel is represented by 3 numbers: Red, Blue and Green values 0-255

Binary Explained

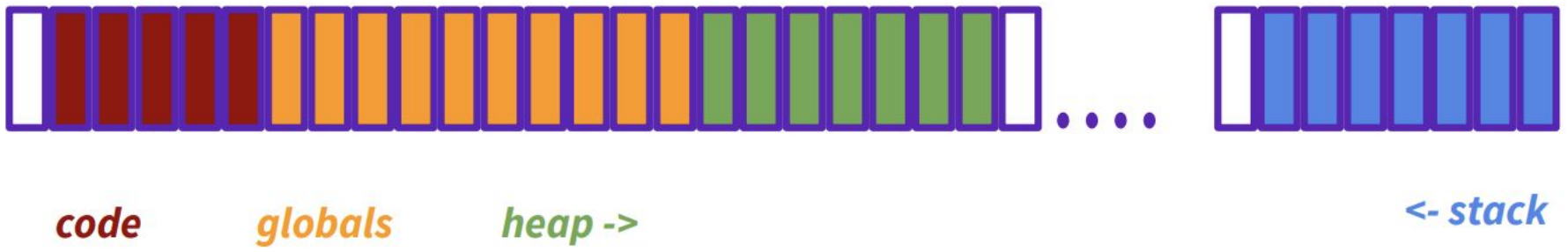
Addresses in Memory

- Computer memory operates just like an array – addresses and the spaces they represent
  - Spaces are measured in ”bytes” of 8 bits
- Each space in memory is referred to by its address
  - Value 504 stored at address 0x08
  - Address of value 504 stored at 0x38
- A pointer is a data object that holds an address
  - Addresses can point to any type of data because they simply point to any space in memory
  - Like a “contact” object that stores someone’s phone number, doesn’t store the actual person
  - Pointers are also stored in memory
  - Pointers can point to other pointers! <follow down the rabbit hole>
  - Pointers can **either** point to a single variable or an array



Program Memory allocation

- As a program executes it interacts with the computer's working memory
  - Code - Sets aside space for the code compiled instructions
  - Globals - Then sets aside space for global variables, static constants, string literals, things that get declared at program initialization
  - Heap – As program executes this space of memory is used for local variables that get allocated and deallocated (new or 'malloc' variables)
  - Stack – holds and serves the current instructions in order that they are received (First In First Out)
  - Both the heap and stack grow dynamically throughout the run of a program
    - If they meet in the middle that means the program has run out of memory



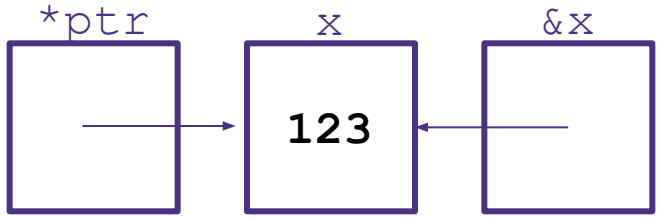
Pointer and Address Syntax in C

`int *ptr;` also works! Programmer preference

```
int* ptr; // a variable of type "pointer to int" without assignment

int x = 123; //an int variable called "x" that stores "123"

ptr = &x; // store the address of "x" in "ptr"
```



- `*` Means "pointer to type"
  - `*` placed after type indicates a pointer data type
    - Similar in java if you add [] after type you declare an array of that type
    - `int*` means "pointer to int"
- `&` means "address variable"
  - Placing an & before a variable name will give you the address in memory of that variable

https://www.youtube.com/watch?v=5VnDaHBi8dM < Binky!

Dereferencing Pointers

```
int x = 123;
int* ptr = &x;
*ptr = 456;
printf("new value of y:%d\n", *ptr);
```

- Placing a * before a pointer **dereferences** the pointer
  - Means "follow this pointer" to the actual data
  - `*ptr = <data>` will update the data stored at the address the pointer is referring to ie 'write to memory'
  - `*ptr` will read the data stored at the address indicated by the pointer
  - Accessing unused addresses causes a 'segmentation fault'
- A **dangling pointer** is one that points to a dead local variable
  - Data that is no longer in use
  - Dereferencing a dangling pointer is "undefined behavior" (UB)
  - UB means ANYTHING could happen
    - Program could crash(best case), silently fail(worst case)
    - GCC can catch this kind of error with a warning, but not always

Strings in C

```
char s1[] = {'c', 's', 'e', '\0'};
char s2[] = "cse";
char* s3 = "cse";
```

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09
a	q	s	h	e	l	l	o	\0	r

All are equivalent ways to define a string in C

There are no "strings" in C, only arrays of characters

- "null terminated array of characters"

`char*` is another way to refer to strings in C

- Technically is a pointer to the first char in the series of chars for the string

Strings cannot be concatenated in C

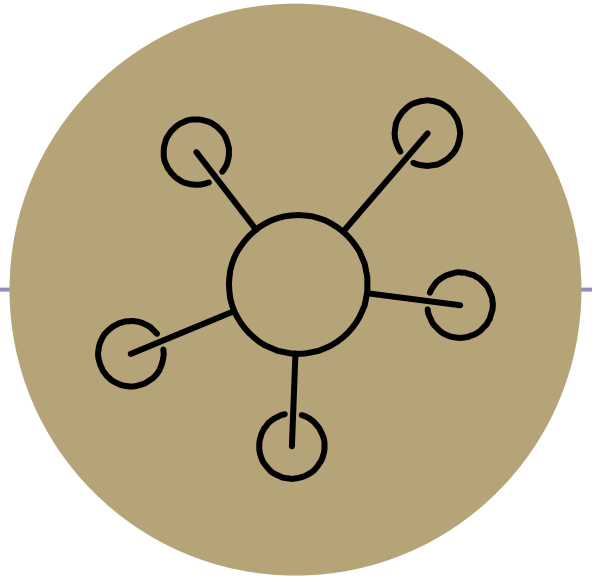
```
printf("hello, " + myName + "\n"); // will not work
```

Printf – print format function

- Produces string literals to stdout based on given string with format tags
  - Format tags are stand ins for where something should be inserted into the string literal
  - %s – string with null termination, %d – int, %f – float
  - Number of format tags should match number of arguments
    - Format tags will be replaced with arguments in given order
- Defined in `stdio.h`
- printf("format string %s", stringVariable);
  - Replaces %s with variable given
  - `printf("hello, %s\n", myName);`

https://en.wikipedia.org/wiki/Printf_format_string

Demo: C pointers



Questions

Binary

https://www.youtube.com/watch?v=LpuPe81bc2w < binary explained

- Base 2 numbering system
- Convention: starts with 0b

0b110 in decimal

0b110 = (1 * 2^2) + (1 * 2^1) + (0 * 2^0) = 4 + 2 + 6

Review: Number Systems

- One byte is…
  - Two hex **nibbles**
  - Eight binary **bits**
  - At most, three decimal **digits** (2-5-5)
- Thus, **one** nibble is four bits!
  - 0x0 = 0b0000 = 0
  - 0xF = 0b1111 = 15
- Helpful exercise: count to 15 in binary!
  - How about 32 in hex? = 0x20 (16's 1's)

Binary
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

Example: Returning a String

```
char* foo();
int main() {
  char* s = foo();
  printf("string: %s\n", s);
}
char* foo() {
  char message[256] = "Hello!";
  return message;
}
```

Fix: Output Parameter (live)

```
#include<string.h>
void foo(char* output, int max_len);
int main() {
  char s[256];
  foo(s, 256);
  printf("String: %s\n", s);
}
void foo(char* output, int max_len) {
  strncpy(output, "Hello!", max_len);
}
```

https://godbolt.org/z/sxddq7