Things to know about C

Yep, I meant “C”

Another language?!?

- Yes, you will likely acquire 5+ languages during your time at OSU
- In the world of computer science, you are expected to learn and adapt
- Use the right tool (“language”) for the job
- The core portions of Windows, Linux and OSX are all written in C

A Brief History of C

- Developed by Dennis Ritchie (1941-2011) between 1969 and 1973 at Bell Labs
- C is a successor to B; however, B's inability to take advantage of the PDP-11's advanced features (to which computer Ritchie and Ken Thompson were busily porting UNIX) caused Ritchie to develop C
- UNIX was then re-written in C in 1972, which had been in development at the same time

C vs C++

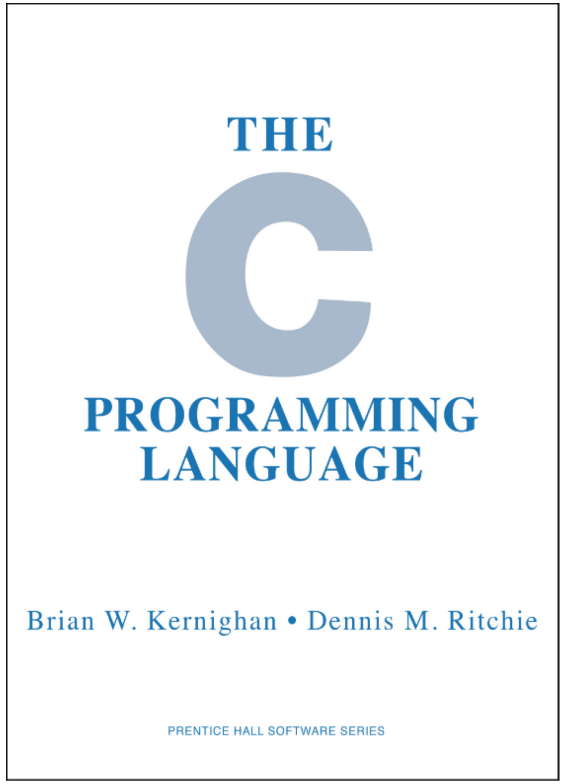
- No objects (everything must be written as functions)
- C cannot pass by reference
- All libraries that will be included end in **.h**
- Strings are always “C-style”
- File I/O works off a file pointer
- Compile with **gcc** (not **g++**)
- http://cs-fundamentals.com/tech-interview/c/difference-between-c-and-cpp.php

C (and C++) are High-Level Languages

- As opposed to a low level language, like assembly
- The original version of C (C89) has 32 reserved keywords, and 50+ operators and syntax characters
- C syntax widely influences programming language syntax development today

HELLO WORLD

```
#include <stdio.h>

int main()
{
   printf("Hello world\n");
   return 0;
}
```



The first C book, written by Ritchie and Brian Kernighan, contains the first usage of a Hello World program put in book form

HELLO WORLD 2.0

```
#include <stdio.h>

int main()
{
    char* myString = "Hello World";
    float version = 2.0f;
    printf("%s\n", myString);
    printf("Version %.2f!\n", version);

    return 0;
}
```

```
$ hw2
Hello World
Version 2.00!
```

printf Formatting Specifiers

```
printf("Version %.2f!\n", version);
```

Specifier	Interpretation
%i or %d	signed decimal integer
%u	unsigned decimal integer
%c	char
%f	floating point
%s	null-terminated string
%x	unsigned int (as hexadecimal)

C vs C++: Dynamic Memory

- Use `malloc()`
- Takes the size of memory in bytes
- Returns a memory address as a `void*`

```
int n = 5;
int* array = (int *) malloc(n * sizeof(int));
// some code...
free(array);
```

Comparing a String

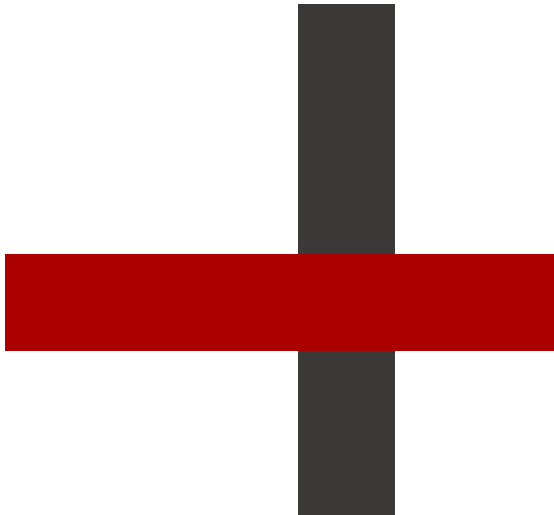
```
#include <stdio.h>
#include <string.h>

void main()
{
        char* my_string = “interesting";
        char* class_number = “CS162";
        int length;

        length = strlen(class_number);
        printf("Length of entered string is = %d\n", length);

        if (strcmp(my_string, class_number) == 0)
                printf("Entered strings are equal.\n");
        else
                printf("Entered strings are not equal.\n");
}
```

```
$ a.out
Length of entered string is = 5
Entered strings are not equal.
```

Array Stuff

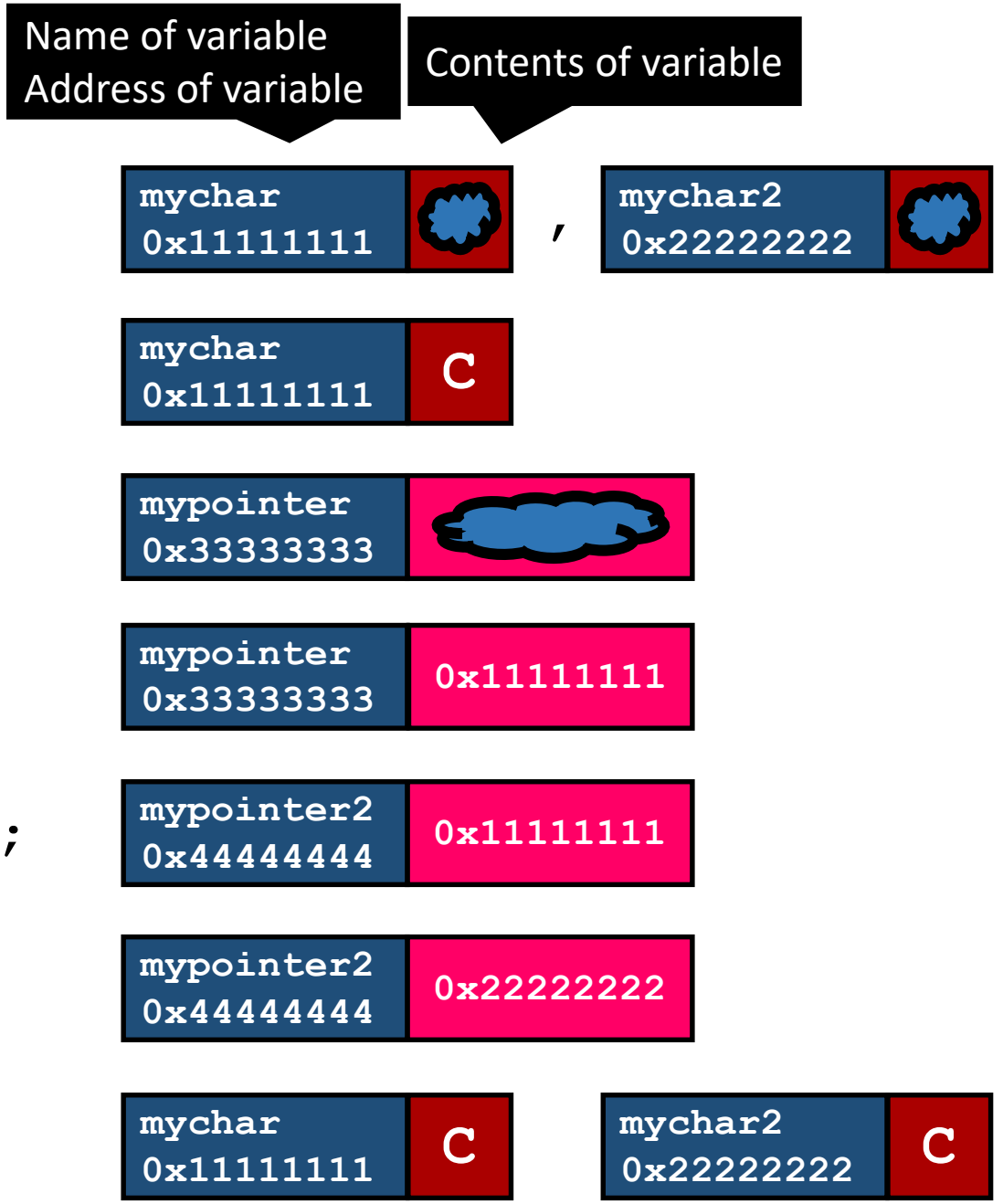
```
#include <stdio.h>

void main() {
  int array[100], maximum, size, c, location = 1;

  printf("Enter the number of elements in array\n");
  scanf("%d", &size);

  printf("Enter %d integers\n", size);

  for (c = 0; c < size; c++)
    scanf("%d", &array[c]);

  maximum = array[0];

  for (c = 1; c < size; c++) {
    if (array[c] > maximum) {
       maximum  = array[c];
       location = c + 1;
    }
  }

  printf("Max element at location %d, value is %d.\n", location, maximum);
}
```

```
$ gcc -o arraystuff arraystuff.c
$ arraystuff
Enter the number of elements in array
5
Enter 5 integers
1 9 3 7 4
Max element at location 2, value is 9.
```

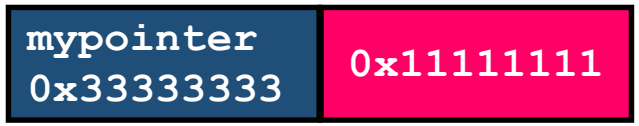


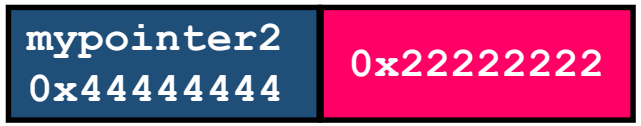
OH CRAP POINTERS

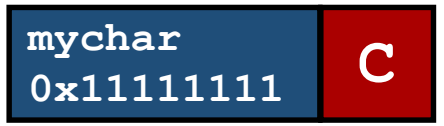
```
char mychar, mychar2;

mychar = 'C';

char* mypointer;

mypointer = &mychar;

char* mypointer2 = mypointer;

mypointer2 = &mychar2;

*mypointer2 = *mypointer;
```

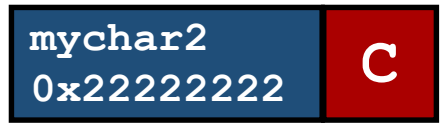


OH CRAP POINTERS - Illegal Commands

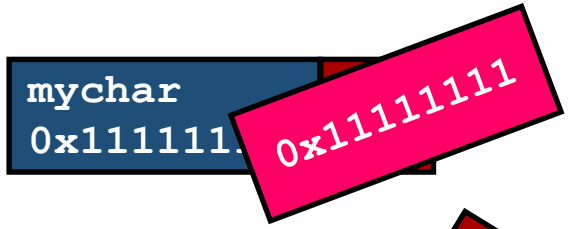






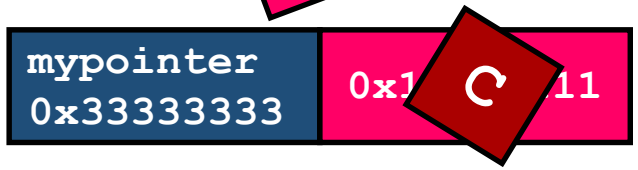



```
mychar = mypointer;
```



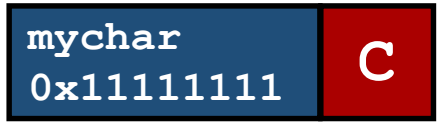
`mychar` can only hold a char, not a pointer to a char!

```
mypointer = mychar;
```



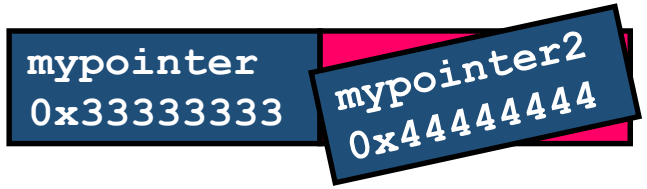
`mypointer` can only hold a pointer to a char, not a char!

```
… *mychar …
```



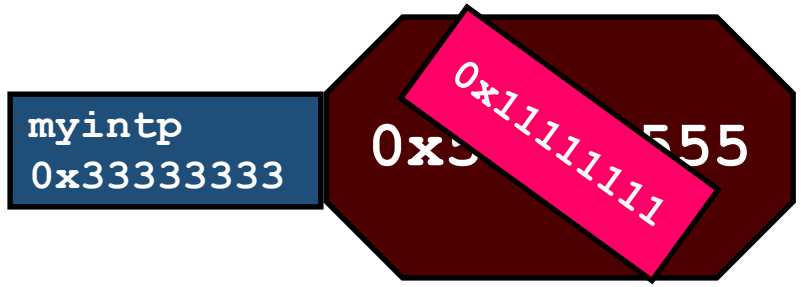
Can't dereference a char, it doesn't hold a pointer to anything!

```
mypointer = &mypointer2;
```



A pointer to a char can't hold the address of a pointer to a char!

```
int* myintp = mypointer;
```



A pointer to an int can't hold a pointer to a char!

OH CRAP POINTERS - Illegal Commands

```
mychar = mypointer;

mypointer = mychar;

mypointer = &mypointe

int* myintp = mypointer;
```