MONTANA
STATE UNIVERSITY

Introduction To C

•••

Finally, Some Code

Today's Lecture

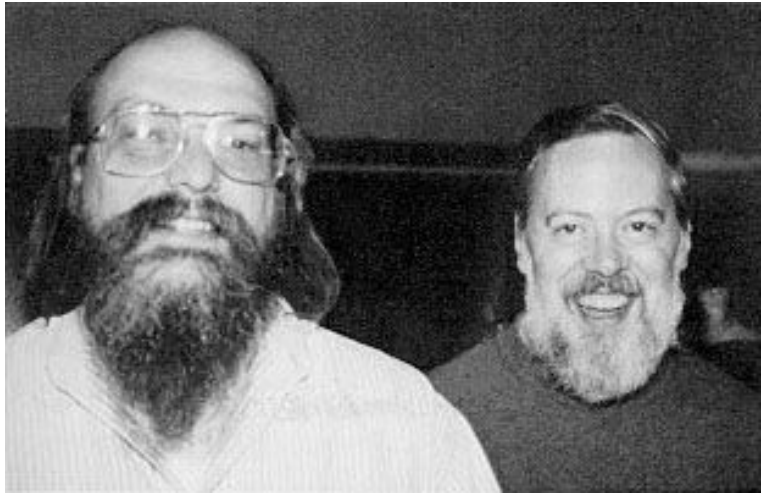
- A bit of background on the C language and its influence
- Overview of basic function declarations
- An overview of primitive data types in C
- C Operators
- Control flow in C
  - For loops & while loops
  - If statements
- Reading for today is K&R Chapters 1-3
  - Strongly recommended!

Online Tutorials

- The Learn C tutorial is a great resource: https://www.learn-c.org
- Relevant Sections For Today:
  - Hello World - https://www.learn-c.org/en/Hello%2C_World%21
  - Variables & Types - https://www.learn-c.org/en/Variables_and_Types
  - Conditions - https://www.learn-c.org/en/Conditions
  - For loops - https://www.learn-c.org/en/For_loops
  - While loops - https://www.learn-c.org/en/While_loops

The C Programming Language

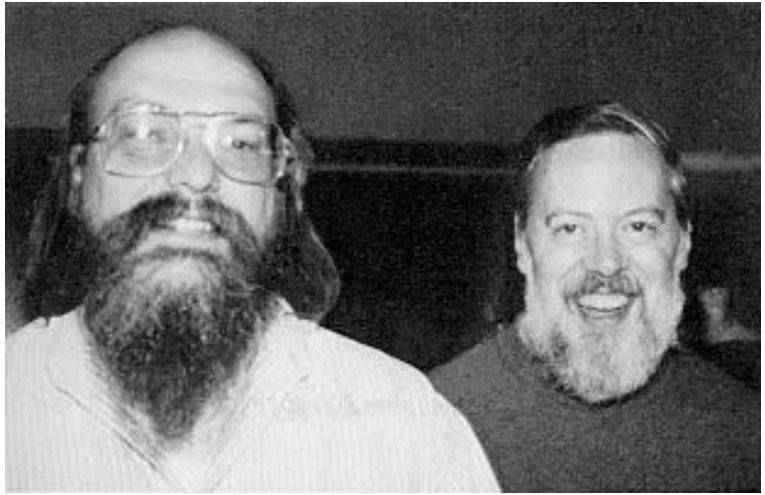
- Invented by Dennis Ritchie in 1972
- Successor to the B programming language
  - Yes, there is an unrelated D programming language
- Designed to help build the original Unix OS
- Pictured: Ken Thompson, Dennis Ritchie



The C Programming Language

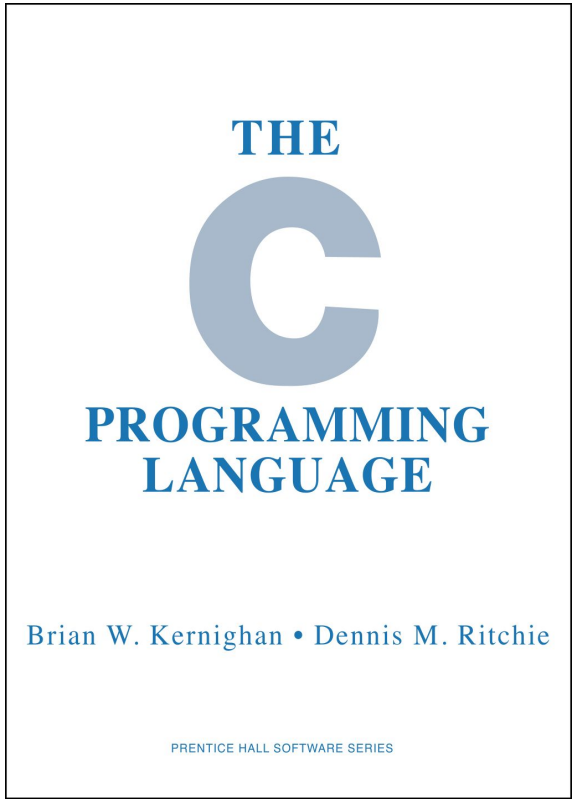
- Dennis Ritchie died in 2001

  “Ritchie was under the radar. His name was not a household name at all, but... if you had a microscope and could look in a computer, you'd see his work everywhere inside.”



The C Programming Language

- A hugely influential language
- Many future languages are based on it
  - Java
  - C++
  - Javascript
  - etc.



The C Programming Language

- The javascript for loop
- Unbelievably, this is basically C
- Is this because C was great?
- Or because Javascript is terrible?

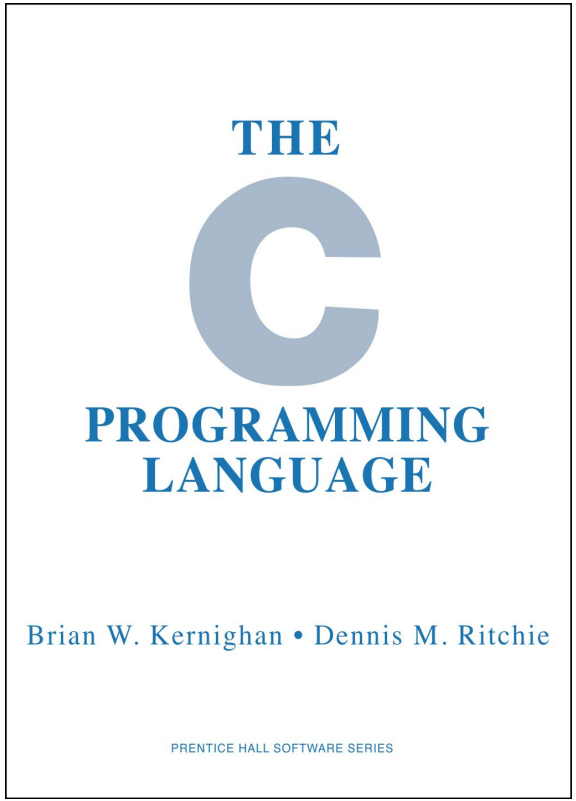
```
var i;
for (i = 0; i < cars.length; i++) {
  text += cars[i] + "<br>";
}
```

The C Programming Language

- The javascript for loop
- Unbelievably, this is basically C
- Is this because C was great?
- Or because Javascript is terrible?

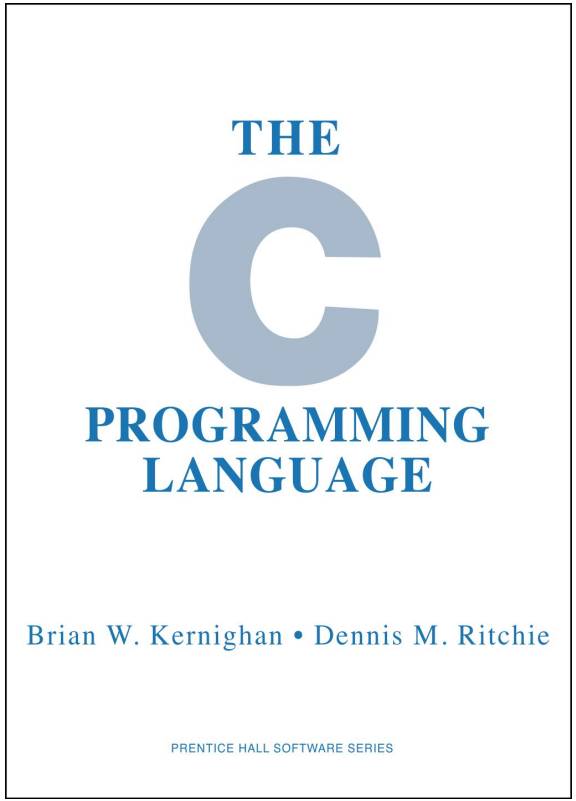
The C Programming Language

- C is still widely used in industry
- Linux kernel development is done in C
- Many embedded systems
- Many core server systems
  - DBMS
  - Web Servers
  - DNS



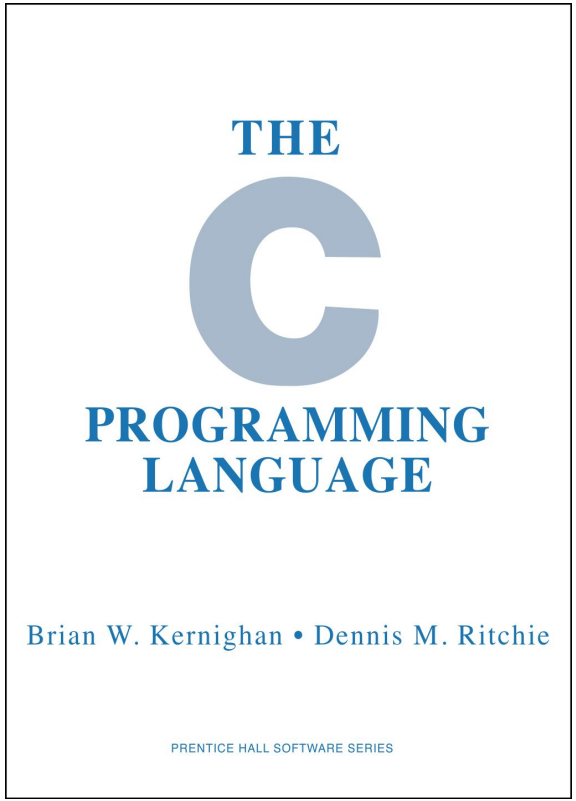
The C Programming Language

- Why?
- A few reasons
  - C is fast
  - C is portable
  - C is easy *enough*
  - C was there when the internet was being built
  - Historical momentum is huge



The C Programming Language

- Is C Hard?
- No, it’s a very simple language
  - Way simpler than Python!
- Yes, it’s a complex environment
  - Libraries are crazy and old
  - Memory management is difficult (at first)
  - Close to the machine, but we don’t know the machine now



Hello C

```c
#include <stdio.h>

int main() {
  printf("Hello, World!");
  return 0;
}
```

Includes

- include is a macro
- Macros in C are *pre-processed*
- They are inserted *as text replacements* before the C parser runs
- This technique is used extensively in C
- Angle brackets vs. strings
  - #include <foo.h> // look for foo.h on system path
  - #include “foo.h” // look for foo.h in the current directory, then the system path

Hello C

```c
#include <stdio.h>

int main() {
  printf("Hello, World!");
  return 0;
}
```

Data Types In C

- Common C Data Type
  - int - usually means a 32 bit integer
  - long - *also* usually means a 32 bit integer
    - (ノ°Д°)ノ︵ ┻━┻
  - long long - a 64 bit integer
  - unsigned long long - an unsigned 64 bit integer
  - char - 8 bit integer, represents a character (usually)
  - float - 32 bit floating point decimal number
  - double - 64 bit floating point decimal number

Data Types In C

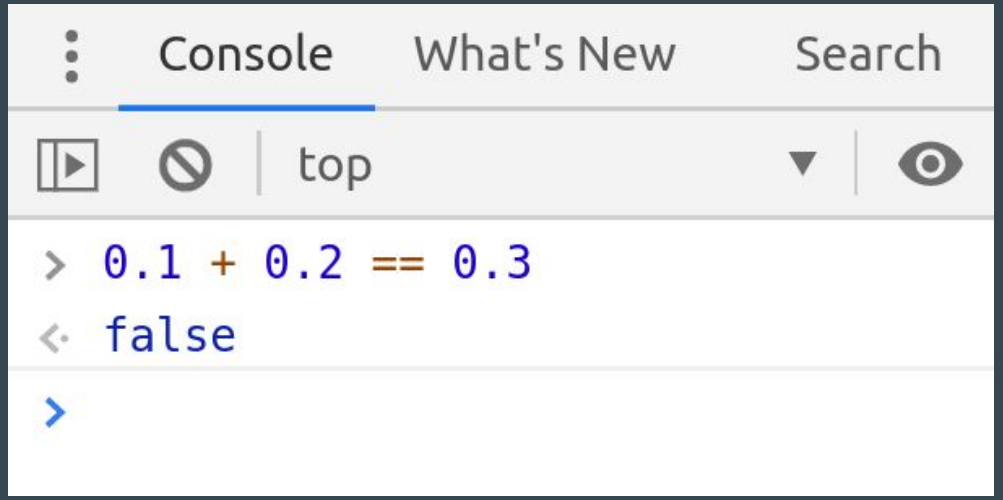
- C is relatively permissive in allowing *casting* between different data types
- What do you think this code prints?

```c
#include <stdio.h>
void cast_demo() {
    int i;
    double x = 1.9;
    i = x;
    printf( format: "i is %d", i);
}
```

A Javascript Aside

- Javascript is an easy programming language, right?
- There is only one numeric type, Number
  - https://javascript.info/types
- No worrying about all these different types of numbers flying around...

A Javascript Aside



Hello C

```c
#include <stdio.h>

int main() {
  printf("Hello, World!");
  return 0;
}
```

Function definition

- Syntax should be familiar
- Return type, followed by function name, followed by argument list
- Arguments are comma separated and declared in the following format:

  type_name arg_name

- Curly braces around function body
- This syntax influenced many other language (e.g Java)

Hello C

```c
#include <stdio.h>

int main() {
  printf("Hello, World!");
  return 0;
}
```

The printf function

- This demonstrates calling a library function
- Syntax should again be familiar
- We are passing in a string literal
- What does the signature of printf look like?

```
extern int printf (const char *__restrict __format, ...);
```

- uhhhhh.

The printf function

- This method signature shows how C can be hard
- Ignore the extern stuff, let's look at the first argument definition
- This function takes a *pointer* to a char
  - const - "Compiler, I will not assign a new value to this variable name"
  - __restrict - "Compiler, I will only use this pointer to access this data"

```
extern int printf (const char *__restrict __format, ...);
```

- Ugly. You won't need to deal with this stuff much.

Hello C

```c
#include <stdio.h>

int main() {
  printf("Hello, World!");
  return 0;
}
```

The return statement

- The main function returns the value 0
- This signals to the *C runtime* that the program terminated properly
- Yes, there is a C runtime!
  - It does some basic command line parsing for you
  - It provides a memory manager
    - NOT a garbage collector though!
  - It deals with the OS calls necessary to start and end a program
  - There’s actually quite a bit of code there
- And that’s it for Hello World in C

Control Flow

•••

for, while, do/while

Control Flow - For Loop

```c
#include <stdio.h>

int main() {
  int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  int factorial = 1;

  int i;

  for(i=0;i<10;i++){
    factorial *= array[i];
  }

  printf("10! is %d.\n", factorial);
}
```

Array Syntax

- Should be familiar from Java
- We will discuss Arrays and Pointers in a future lecture
- For now, just accept it

Control Flow - The For Loop

```c
#include <stdio.h>

int main() {
  int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  int factorial = 1;

  int i;

  for(i=0;i<10;i++){
    factorial *= array[i];
  }

  printf("10! is %d.\n", factorial);
}
```

The For Loop

- Again, should be familiar from Java
  - Or Javascript (ノ°Д°）ノ︵ ┻━┻
- Do you notice anything different?

The For Loop

- Again, should be familiar from Java
  - Or Javascript (ノ°Д°）ノ︵ ┻━┻

- Do you notice anything different?
- *Arrays do not encode their length*
- Array indexing is *not* checked
  - You can read or write right off the end of an array
  - C is fast, and decisions like this are why
  - Demo

Control Flow - The For Loop

```c
#include <stdio.h>

int main() {
  int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  int factorial = 1;

  int i;

  for(i=0;i<10;i++){
    factorial *= array[i];
  }

  printf("10! is %d.\n", factorial);
}
```

Arithmetic Operators In C

- C supports many conventional arithmetic operators
  - + - addition
  - - - subtraction
  - * - multiplication
  - / - division
- C supports unary mathematical operators (increment, decrement):
  - x++
  - ++x
  - x--
  - --x

Compound Operators

- Compound operators
  - *= means

    “multiply the left hand side by the right hand side and put the value in the left hand side”

Control Flow - For Loop

```c
#include <stdio.h>

int main() {
  int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
  int factorial = 1;

  int i;

  for(i=0;i<10;i++){
    factorial *= array[i];
  }

  printf("10! is %d.\n", factorial);
}
```

Another printf, with variable arguments

- Printf takes a *format string*
- Format string can have formatting specifiers in it
  - %d means *decimal*
- Var args (the … from the definition) allows you to pass in any number of arguments after the format string
- Matched up numerically with the format specifiers
- The variable arguments are matched up with formatting specifiers by number
- The nth formatting variable will display the nth var arg

Formatting Specifiers

Format Specifier	Description
s	String
c	Single Character
d	Decimal
e	Exponential Floating point
f	Fixed Floating point
g	Uses either e or f depending on which is smaller for the given input
o	Octal
x	Hexadecimal
%	Prints the percentage symbol

Control Flow - While/Do While

```
while (command) {
    command = repl_read_command("battleBit (? for help) > ");
    repl_execute_command(command);
    cb_free(command);
}

do {
    command = repl_read_command("battleBit (? for help) > ");
    repl_execute_command(command);
    cb_free(command);
} while (command);
```

While/Do While

- Loops without a loop variable as in the for loop
- For loop is functionally equivalent
  - Why have both?
- Why would you pick a do/while over a while loop?

Control Flow - If Statements

```c
int foo = 1;
int bar = 2;

if (foo < bar) {
    printf("foo is smaller than bar.");
} else if (foo == bar) {
    printf("foo is equal to bar.");
} else {
    printf("foo is greater than bar.");
}
```

If Statements

- Should be familiar
- Note that C has **no boolean** type by default
- There is a library to supply the _Bool type: <stdbool.h>
- Not widely used
- 0 means false
- Non-0 means true

Relational Operators In C

- Since C doesn’t have a boolean type, if conditions are usually expressed in terms of a relational operator
  - > - greater than
  - >= - greater than or equal
  - < - less than
  - <= - less than or equal
  - == - equal
- Careful! = (assignment) is a valid expression in an if statement

And That's It!

- See? C is easy!
- We have discussed
  - Basic function declarations
  - Basic data types
  - For loops and arrays
  - Do and do/while loops
  - The if statement
  - Basic operators
- This is about 50% of the language
- Next time we will discuss functions in more detail...

MONTANA
STATE UNIVERSITY