Programming in C

CS 3410: Computer System Organization and Programming

[K. Bala, A. Bracy, E. Sirer, Z. Susag, and H. Weatherspoon]

What's your background in C?

A. Never learned C until CS 3410

B. Learned C in another class at Cornell

C. Learned C in a non-university setting

D. Know some C++

E. Know a lot of C++

Why C?

"C is a horrible, horrible programming language."
– David Gries (Cornell)

<expletives deleted> - Michael Clarkson (Cornell)

"C had a good run." – Andrew Myers (Cornell)

"C is for people who don't like to wear seatbelts.“
-my 1st CS professor

"C is quirky, flawed, and an enormous success."
- Dennis Ritchie (creator of C)

"A C program is like a fast dance on a newly waxed dance floor by people carrying razors.“
- Waldi Ravens (who is this?)

After All These Years, the World is Still Powered by C Programming

History:

- UNIX started in 1969; C was created for UNIX and UNIX was rewritten in C in 1972

What is written in C?

- Operating Systems: Windows, Linux, Mac OS X kernel
- Mobile OSes: iOS, Android and Windows Phone kernels
- Databases: Oracle Database, MySQL, MS SQL Server, and PostgreSQL)
- Embedded Systems: alarm clocks, microwaves, coffee maker, every sensor and control device in your car

Why is C Still Used?

- Portability and Efficiency
- Memory manipulation
- Deterministic Usage of Resources
- Code Size

Reasons to Learn C

- Understand the Machine (Think in C)
- Work on Many Interesting C Projects
- Cornell: 3410, 4410, 4411, 4414, *and more*

Why is C Still Used?

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

Energy Efficiency across Programming Language
R Pereira, M Couto, F Ribeiro, R Rua, J Cunha, J P Fernandes, J Saraiva. ACM SIGPLAN International Conference on Software Language Engineering (SLA). Pages 256–267. October 2017. https://doi.org/10.1145/3136014.3136031

C is 100% WYSIWYG*

- There is no magic.
- Nothing is hidden.
- Nothing is protected.

No constructors, no classes, no "new", no inheritance, no base objects.

There are some libraries.

*What You See Is What You Get

Goals for today

- Minimal C basics
  - Minimal C program highlighting C basics
- How to print!
  - Print numbers
  - Print negative numbers
  - Overflow
- More C basics
  - Prototypes, Headers, Libraries
  - Compiling and Linking

Getting started: `minimal.c`

```c
int main() {
    return 0;
}
```

Getting started: `minimal.c`

```c
int main() {
    return 0;
}
```

Compile

```
$ gcc minimal.c –o minimal
```

Execute

```
$ minimal
```

Getting started: `minimal.c`

```
int main() {
    return 0;
}
```

Compile [on 3410 infrastructure]

```
$ rv gcc -Wall -Wextra -Wpedantic -Wshadow -Wformat=2 -std=c17 -o minimal minimal.c
```

Execute [on 3410 infrastructure]

```
$ rv qemu minimal
```

Printing: hello.c

```c
#include <stdio.h>

int main() {

    printf("Hello, 3410!\n");



    return 0;
}
```

Printing: hello.c

```c
#include <stdio.h>

int main() {
    int n = 3410;
    printf("Hello, %d!\n", n);



    return 0;
}
```

Printing: `hello.c`

```c
#include <stdio.h>

int main() {
    int n = 3410;
    printf("Decimal: %d\n", n);
    printf("Binary: %b\n", n);
    printf("Hexadecimal: %x\n", n);

    return 0;
}
```

Printing: `print_int.c`

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

int main() {
    int8_t n = 7;
    printf("n = %hhd\n", n);

    return 0;
}
```

Printing: `print_int.c`

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

int main() {
    int8_t n = 0b00000111;
    printf("n = %hhd\n", n);

    return 0;
}
```

Printing: `print_int_neg.c`

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

int main() {
    int8_t n = 0b10000111;
    printf("n = %hhd\n", n);

    return 0;
}
```

Printing: `print_int_neg.c`

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

int main() {
 int8_t n = 7;
 printf("n (decimal) = %hhd\n", n);
 printf("n (binary) = %hhd\n", n);

 int8_t flipped = ~n + 1;
 printf("n (decimal) = %hhd\n", flipped);
 printf("n (binary) = %hhd\n", flipped);

 return 0;
}
```

Printing: `print_int_neg.c`

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

int8_t flip(int8_t num) {
    return ~num + 1;
}

int main() {
 for (int8_t i = -128; i < 127; ++i) {
   printf("i = %hhd\n", i);

   int8_t negated = -i;
   int8_t flipped = flip(i);
   if(negated != flipped) {
     printf("mismatch\n")}
   }
 }
 return 0;
}
```

overflow.c

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

int main() {
 int8_t num = 0;

 for (int8_t i = 0; i < 500; ++i) {
   num += 1;
   printf("num = %hhd\n", i);
 }

 return 0;
}
```

overflow.c

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

int main() {
 int8_t num = 0;

 for (int8_t i = 0; i < 500; ++i) {
   num += 1;
   printf("num = %hhu\n", i);
 }

 return 0;
}
```

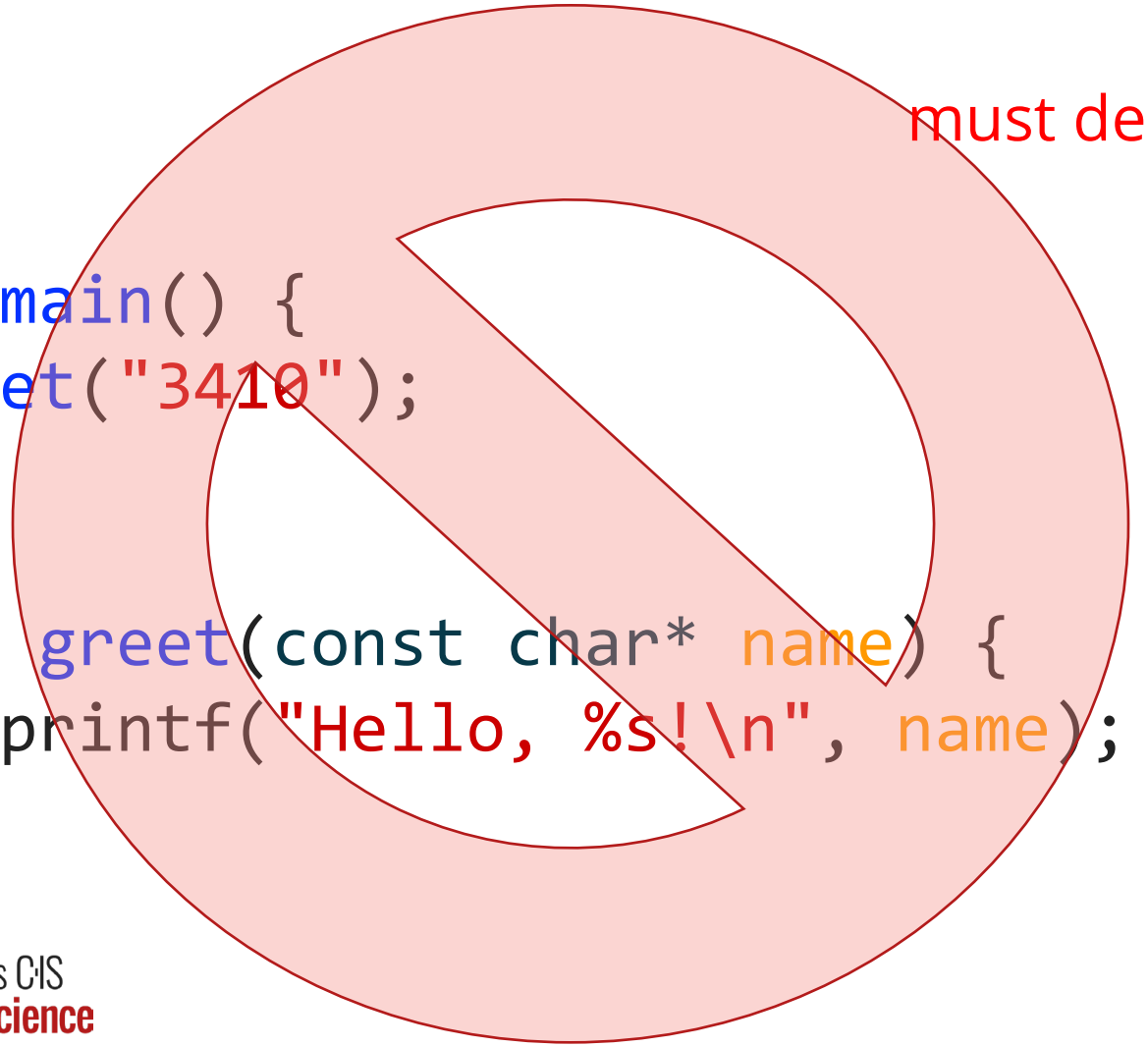
Prototypes, Headers, Libraries

```c
#include <stdio.h>



void greet(const char* name) {
    printf("Hello, %s!\n", name);
}

int main() {
 greet("3410");
}
```

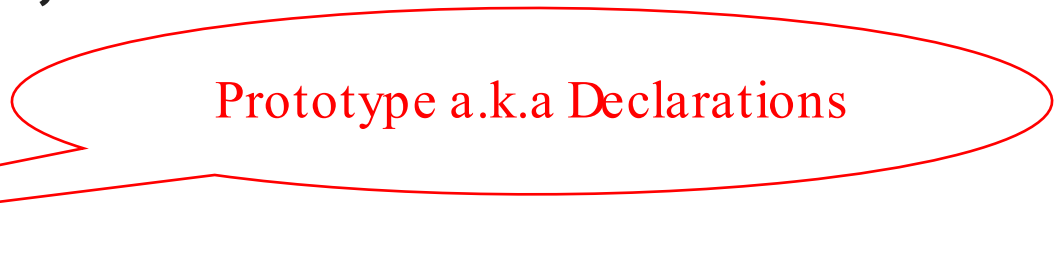
Prototypes, Headers, Libraries

```c
#include <stdio.h>

int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

must declare before use



Prototypes, Headers, Libraries




```c
#include <stdio.h>

void greet(const char* name);

int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Header file: greet.h

```
void greet(const char* name);
```

Prototypes, Headers, Libraries

```c
#include <stdio.h>
#include "greet.h"


int main() {
 greet("3410");
}

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Separating files: greet.c

```c
#include <stdio.h>
#include "greet.h"
```

```c
void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Separating files: main.c

```c
#include <stdio.h>
#include "greet.h"


int main() {
 greet("3410");
}
```

Compile and link

Compile **.c** files together

Compile [on 3410 infrastructure]

```
$ rv gcc main.c greet.c -o main
```

Execute [on 3410 infrastructure]

```
$ rv qemu main
```

Compile and link

Or,

Compile **.c** files ***separately***, then ***link*** them together

Compile [on 3410 infrastructure]

```
$ rv gcc –c main.c -o main.o
$ rv gcc –c greet.c -o greet.o
$ rv gcc main.o greet.o -o main
```

Execute [on 3410 infrastructure]

```
$ rv qemu main
```