CIS241

System-Level Programming and Utilities

C Types (and Printing)

Erik Fredericks, frederer@gvsu.edu

Fall 2025

> **Based on material provided by Erin Carrier, Austin Ferguson, and Katherine Bowers**

Data types

Non-floating point

- `int` - at least 16 bits
- `long` - at least 32 bits
- `long long` - at least 64 bits

Floating point

- `float` - typically 32 bits
- `double` - typically 64 bits

Character

- `char`

Boolean?

`true` / `false`?

https://stackoverflow.com/questions/1921539/using-boolean-values-in-c

Pre C-99 - no boolean type!

- Use a `char` or `int` with an 'accepted' value for what is true and false
  - Note that for `ints` - 0 is `false` and **non-zero** is `true`!

C version?

https://stackoverflow.com/questions/36662063/how-can-i-know-the-version-of-c

```
gcc -dM -E - < /dev/null | grep __STDC_VERSION__ | awk '{ print $2 " --> " $3 }'
```

- My output: `__STDC_VERSION__ --> 201710L`
  - C17 (might see `c89`, `c99`, `c11`)

Booleans (`c17` or `c99`)

```c
#include <stdio.h>
#include <stdbool.h> // need the boolean header!

int main(void)
{
  bool var;
  var = true;

  printf("Boolean value: %d\n", var);
  printf("Boolean value as str: %s\n", var ? "true" : "false");

  return 0;
}
```

Printing output

`printf(controlstring [, data])`

- `controlstring` indicates surrounding text to print and how to format variable pritning
- `data` is optional - print variable values
  - i.e., what variable to print!
  - `controlstring` contains format specifiers for each `data` printed

Format specifiers

integer: `%d` (you may see `%i`)

- Can add additional formatting information
- Add number before `d` - specify minimum width `%3d`
- Specify 0-fill: `%03d`
- Specify left justify: `%-3d`

Format specifiers

float/double: `%f`, `%e`, `%g`

- `%f`: fixed point notation
- `%e`: exponential notation
- `%g`: chooses between normal and exponential (drops trailing)

Number before decimal - total width to use

Number after decimal - places after decimal point

Use `0` for 0-fill, use `-` for left justify

e.g.

We can still further customize:

```
printf("PI = %08.3f", 3.14);
```

Here:

- `0` is for zero pad
- `8` is for 8 total characters (including the decimal point)
- `3` is for places after decimal point

Format specifiers

Character: `%c`

String: `%s`

- Same with int/float you can add:
  - Number to specify width
  - `-` to specify left-justify

Signedness

These are different:

- `int x = 5;`
- `unsigned int y = 5;`

Why would we ever use unsigned ints?

- Range!
- Unsigned ints can hold numbers twice as large.
- Technically, ASCII chars are unsigned chars

Type conversions (implicit)

C will automatically convert types in certain scenarios:

- **Operating on mismatched types**
  - `3.0 / 2`
- **Assigning values**
  - `double x = 5;`
  - `int y = 4.0;`
- **Calling functions**
  - `float_exp(2, 3);`

Type conversions (explicit)

We can also typecast!

- `int x = (int) 5.0;`
- `(unsigned int) -1;`

Format: `(type) expression;`