Data types and printf

Eric McCreath

Learning C

- It is difficult to present material on C programming such that ensuing content purely builds on previous content. Therefore, we simply *jump into* the C programming language and then focus on different aspects

- It is presumed that students of this course have some programming background. Hence, this material will be progressed through very quickly.

- These notes are not intended be a complete reference on the C programming language, just an outline on certain core topics

- "Practical C Programming" by Oualline is a good reference. However, there are a lot of other good books and online resources covering this material.

Not many basic data types

- There aren't many basic data types in C.
- The basic data types primarily used in C include:
  - **integer** - stores whole numbers (numbers without a decimal point). These include: *int, char, short, long,* ..
  - **floating-point number** - stores a large range of fractional numbers (from the very small to the very large). These include: *float, double*
  - **void** - this can be thought of as an *empty* data type or as *any* data type. If the type of a variable is unknown, it can be referred to as type *void*
- Other higher level languages often have strings and booleans as part of the language. C does not!

Integer

- Integers include numbers like 5, 1000, -30, 0, ...
  - They are represented using 2s-complement and have a fixed range.
  - The range will depend on the architecture (this creates significant problems when moving code between different architectures).
  - On most desktop systems we use, integers are 32-bit.
- To declare an integer use the **int** keyword. For e.g.

```
int value;
```

Integer

- C allows a range of different integers. These include:
  - **char** - a single byte
  - **unsigned char** - representing a number that ranges from 0 to 255, rather than -128 to 127.
  - **short** - normally 16-bit
  - **unsigned short**
  - **int** - normally 32-bit
  - **unsigned int**
  - **long**
  - **unsigned long**
  - **long long**
  - **unsigned long long**

printf

- To use printf, remember to include its header file:

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

- printf requires a format string followed by an optional number of expressions as arguments:

```
printf(<fomat>, <expression1>, <expression2>, ...);
```

- Remember:

```
printf("Hello World!\n");
```

- Use %d for integers. For e.g.

```
int value = 70;
printf("v: %d twov: %d justfive: %d\n", value, 2*value, 5);
/* prints "v: 70 twov: 140 justfive: 5" */
```

- Use "man 3 printf" to get more info on the formatting used by printf

Floating-point

- **float** - 32-bit single precision float
- **double** - 64-bit double precision float
- Literals can be expressed using a decimal point:

```
float value1 = 3.14;
```

- Literals can also be expressed using an exponent form:

```
float value2 = 1.5E3;
float value3 = -1.6E-6;
float value4 = 100e4;
```

- %f can be used within printf. For e.g.

```
printf("val: %f", value1);
```

- %e can be used to display using the exponent style
- With floats, the precision is limited by how the number is represented. Although errors may be small, they often accumulate!

booleans and flags

- An integer type is used for representing boolean values. 0 is false and anything non-zero is true

- Two common traps:

```
if (x = 100) { ....
```

and

```
#define TRUE 1
#define FALSE 0
:
if (x == TRUE) { ...
```

- Particular bits within an integer can even be used to represent different flags using *bit-wise* operations

```
#define FLAG1 0x01
#define FLAG2 0x02
#define FLAG3 0x04
:
flag = flag | FLAG2 // set FLAG2
flag = flag & ~FLAG2 // clearing FLAG2
if (flag && FLAG2)  // checking a flag
```

Fixed-point numbers and pointers

- Generally fixed-point number representation is not supported in C
  - Although, gcc has some support for fixed-point numbers

- Pointers are also a basic data type in C
  - They point to locations in memory

More about pointers in Week 5.

Exercises

- Read the man page for printf (man 3 printf)
- Write a program that outputs both integer and floating point numbers
  - Explore how printf can be used to justify printed numbers
  - Print a right-justified (aligned along the rightmost digit) column of numbers
  - Print floating-point numbers with a fixed number of decimal places displayed