Data Structures in C

Prof. Georg Feil

Variables

Summer 2018

Acknowledgement

- These lecture slides are based on slides and other material by Professor Magdin Stoica
- Additional sources are cited separately

Reading Assignment (required)

- C for Programmers (supplementary textbook)
  - Section 2.4 (Arithmetic in C)
  - Sections 3.9 and 3.10 (Assignment Operators)

Variables

- A variable has 4 main properties:
  - Data type
  - Name
  - Value
  - Scope
- It can also have additional type qualifiers like const (similar to *final*)

Properties of Variables: Data Type

- The data type of a variable determines the type of information it can remember
  - For now we will use only fundamental data types, also known as built-in types or primitive types
  - These types are defined by the programming language itself (numbers, characters etc.)
- We will learn 13 fundamental data types in this course
  - Many (but not all) have the same names as Java data types like int, double, float, char
  - Their storage size in memory (# bits) may be different than Java
  - Most numeric data types in C have signed and unsigned versions
  - There are a few other data types in the C99 standard we'll skip

Properties of Variables: Name

- The name of the variable is how programs refer to it
- You can choose almost any name you want for a variable
  - Some names are not allowed by the compiler
  - Some names are undesirable according to our own programming rules or coding standard, but the compiler doesn’t complain about them
- You may use the same kinds of names for variables in C as in Java

Properties of Variables: Name

- Variable names can start with a letter or underscore _
  - Avoid variable names that start with an underscore (used by the system)
- They can contain letters, numbers and underscores
  - Technically they can also contain $ but that is reserved for the system so don’t use it!
- *Cannot* use keywords (like long or static) for any name
- Variable names should be descriptive to let the reader know what kind of information they will hold
  - ⚠ int i;   // what does this variable hold?
  - ✓ int count;   // what does this variable hold?

Properties of Variables: Value

- The variable's value is the actual information that is being remembered
  - The variable's value is stored in memory (RAM)
- The kind of information (value) that can be stored in a variable is determined by its data type
  - whole number
  - floating point number
  - character
- It's a good idea to give a variable an initial value when you declare it

Properties of Variables: Scope

- The scope of a variable determines which parts of the program can access (“see”) and modify the variable
- The scope depends on where the variable is declared and whether the static qualifier keyword is used in the variable declaration
    - If you declare a variable anywhere in a function, it’s a local variable
    - If you declare a variable outside of functions (usually at the top of the source file) it’s a global variable
    - If you add the static keyword to a global variable then it’s a “module” variable, only visible in that source (.c) file. **Note this is different from Java’s use of static

Declaring a variable in a function

```
int main(int argc, char** argv)
{
    int radius = 5;
    …
}
```

The name of the variable

Fundamental type for whole numbers

Scope: This is a **local** variable because it’s declared inside a function

Declaring a variable outside functions

```
long int g_ans = 42;

int main(int argc, char** argv)
{
   printf("The answer is %ld\n", g_ans);
   …
}
```

Scope: This is a **global** variable because it's declared outside functions

Note: Qualifiers like 'static' go before the data type

Fundamental Data Types: Integer

- Integer types are used to store whole numbers
  - short int : Numbers are stored in 16 bits (-32,768 to 32,767)
  - int : Numbers are usually stored in 32 bits (-2,147,483,648 to 2,147,483,647

    On embedded platforms an int is often 16 bits
  - long int : Numbers are stored in 32 or 64 bits (as for int or -2^{63} to 2^{63}-1)
  - long long int : Numbers are stored in 64 bits (-2^{63} to 2^{63}-1)

- Each of these types can also be unsigned, e.g. unsigned long int
  - Java has only signed integers, but C has both signed and unsigned
- Unsigned integer data types can hold numbers that are twice as big as the corresponding signed data type, but can't hold negative numbers
  - For example unsigned int may store 0 to 2^{32}-1, or 0 to 4,294,967,295

Fundamental Data Types: Integer

- Some C data type sizes depend on the system (processor/compiler)
  - On 32/64 bit computer systems int is 32 bits, and long int is 32 or 64 bits
  - On “small” processors (8 or 16 bit) an int is usually 16 bits

- If you’re not sure of the size of a data type on a particular system, you can check it using sizeof, e.g.

```
printf("Size of int is %d bytes\n", sizeof(int));
```

Fundamental Data Types: Floating Pt

- Floating-point types are used to store real numbers
  - float : numbers are stored in 32 bits
    (about 6 or 7 decimal digits of precision)
  - double : numbers are stored in 64 bits
    (about 15 or 16 decimal digits of precision)
- These work like the Java types of the same name
- Format in memory follows IEEE-754 standard
  - Sign, exponent, mantissa
- We should use double by default
  - There is usually no benefit to using float and it can cause problems like loss of precision or extra data type conversions

Character and Boolean Types

- Character type is used to remember a single character (letter, digit, symbol) or data byte. Can be signed or unsigned.
  - char - Characters are stored in one byte (8 bits)
    - Can be used as an 8-bit integer
    - Range: signed char -128 to 127, unsigned char 0 to 255
- Boolean type is used to remember a true/false or yes/no decision
  - bool - May be stored in as little as 1 bit (implementation specific)
    - The boolean values true and false correspond to 1 and 0
- Note that bool was officially added in C99
- You need to use #include <stdbool.h> when working with bool
- Wikipedia has a good page on C data types
  - http://en.wikipedia.org/wiki/C_data_types

Example: Declaring Variables

```
int sum;
bool isCorrect;
char grade;
short int age;
short age2;
double Radius;
float average;
unsigned int register_sp;
unsigned char byte;
long int big;
long big35;
long long int reallyBig;
signed int justLikeInt;
```

```
int @cool;
float #evenCooler;
double hu^h;
bool ^doubleHuh?;
long long does-not-work;
char 1Number;
```

Red examples are incorrect. Why?

```
int _sum;
unsigned $imTheCompiler;
double canyoureadthisname;
char some_like_this_not_me;
```

Orange examples will compile but are not used by convention

Exercise 1: Practice With Variables

- Using Dev-C++, write a program with a main function that declares all the “blue” variables in the previous slide

1. Ensure the program compiles
2. Change some of the data types, use all the types we have learned
3. Experiment with different names and see what the compiler has to say about them
4. Give some of the variables an initial value
5. Change some of the variables to global variables
6. Print out the values of your global variables with printf() *(see my slides “Formatted Input & Output” for help on how to use printf)*

Exercise 2: C Fundamental Data Types

- I said we would learn 13 fundamental data types
- List them…

Assigning Variable Values

- Values are stored in variables using an assignment operator
  - C assignment operators are very similar to Java
- A statement that initializes a variable or changes its value is called an assignment statement

```
<variable name> = <value>;
```

- The variable has to be declared first (or at the same time)
- The value has to match the variable's data type
- C has shorthand assignment operators like Java: `+=`, `*=`, `++`, `--` etc.
- Assignment statements are also expressions!
  - For example : `total = subTotal = 100.0;`

Exercise 3

- Download the program inc.c from SLATE (week 1)
- Open it in Dev-C++ and examine the program
  Don’t run it yet!
- Write down what you think the program will print
- Run it and see if you were right…

Literals

Literals

- Literals are constant values that appear directly in a program
  - Literals have a data type just like variables: int, double etc.
  - A literal cannot change – it is the value itself
  - Examples: `3, 5.6, true, "John Wayne", 'a'`
- Whole number literals:
  - By default have the type int: `3, 6, 12345, -456`
  - If suffixed with the letter L or l they are treated as having the type long int: `3L, 6l, 1234567890L, -456L`
  - The suffix U indicates an unsigned int (use UL for unsigned long)
- Floating point number literals contain a decimal point:
  - By default have the type double: `4.5, -3.4, 0.46, 12345.456`
  - If suffixed with the letter F or f they are treated as having the type float: `4.5F, -3.4f, 0.46F, 12345.678f`

Literals (cont.)

- Text literals
  - Single-character literals are enclosed in single quotes: 'a', 'c', 'x'
  - Multi-character literals are called strings and are enclosed in double quotes: "abc", "john", "This is a string!", "z", ""

- Boolean (bool) literals
  - `true` – a condition that is true (such as 5 < 10)
  - `false` – a condition that is false (such as 5 > 10)
  - Remember that true is the number `1`, and false is 0 in C
  - For good programming style you should use `true` & `false`
    - Remember to `#include <stdbool.h>`

Examples

```
int value;
value = 5;
value = 10;
value = "John";
bool isCorrect;
isCorrect = true;
isCorrect = 3;
char ch;
ch = 'x';
ch = "Z";
unsigned char smallVal;
smallVal = 78;
smallVal = 123455;
```

Orange examples are incorrect. Why?

Note: The C compiler will not give an error for any of the orange lines!

Expressions

C language operators, expression evaluation, and math are very similar to Java

C Expressions

- An expression is a calculation involving literals (values) or variables combined using operators (or: anything that has a value!)
- Examples
  - 5 + 10 - 42 + 209
  - radius * radius * 3.14
  - percent / 100
- Arithmetic operators
  - All the math operators we know from Java: addition (+), subtraction(-), multiplication(*), division(/) and remainder (%)
- Logical operators
  - All the operators we know from Java: OR (||), AND (&&), NOT (!)
- Order of operations is similar to Java (BODMAS). Use parentheses to clarify and ensure correct order: 10 * (3 + 4)

Expressions Examples

```
int value = 50;
value = 10 * value; // how much is ‘value’?
value = (5 + 5) * value; // what about now?

int quotient = 4 / 3;
double quotient2 = 4 / 3;
quotient = 7 / 25;
int remainder = 7 % 25;

double fraction = 2.0 / 3.0;

bool result = false;
result = !result;   // what is ‘result’?
result = result || false;
result = true && result;
```