ISA 563: Fundamentals of Systems Programming

Variables, Primitive Types, Operations, Expressions, and Control Flow

Jan. 28, 2014

Outline

- Expressions

- Data representation
  - Variable name
  - Variable type

- Primitive types

- Operations on variables

Readings

- TCPL Section 1.2:

    – Variables and Arithmetic Expressions

- TCPL Chapter 2

    – Types, Operators, and Expressions

Expressions

- A C program is a sequence of statements

- A C program is a collection of functions and the data that those functions operate on

- The building block of statements are expressions: combination of language keywords, function calls, operators, and operands that evaluate to a value

Review: What is a Program?

- A program is a sequence of instructions that operate on data

- A C program is a collection of variables functions that process the data held in those variables

- Computers process long strings of 0's and 1's
    - Need a way to refer to portions of those strings as higher-level data objects

Variables

What exactly is a Variable?

- A variable is the concept of a piece of structured data that can be accessed (read or modified) via well-known, standard rules

- A variable is NOT JUST the data it contains!

- A variable also has:

  - A **name** or identifier that provides a way to refer to it

  - A **type** that defines its size (how much memory it uses)

  - A **location** or memory address specifying where the data is stored

Example: Simple Integer Values

- Suppose we want to write a program for processing students' grades

- Need a variable to hold the total scores:

  - `total_score:` 3456

- Pattern:

  - `Variable name:` value

Example: Variable Declaration

- Declaring a variable is a standard action to let the rest of the program know about a piece of data that will be used

- The following **program statement** declares (that is, tells the computer to set aside a memory location for) an integer variable called 'total_score':

```
int total_score;
```

type   variable name

Declaring Variables

- Variables are usually declared at the beginning of the program or function they are used in

- Variable names can be any combination of letters, numbers, or underscores, but must start with a letter or underscore:

  - Valid names: `i, total_score, round2, _test`
  - Invalid names: `$id, 2nd, total score`

- Variable names should be descriptive; avoid names like 'ab', 'x', 'tmp', etc., unless for a good reason

- Make sure you don't try to name a variable after a reserved work (if, for, while, case, switch ...)

Subtle Points about Variable Names

- When you program, you see the variable name

- When the computer executes your program, it actually sees the variable memory address

- In both cases, the data is used behind the scenes

Types

Variable Types

- A type is a hint to the computer on how to handle the data contained in or referred to by the variable

    - Usually this involves size of the storage allocated

- There are 4 basic primitive types in C:

    - `int` (regular integers)

    - `char` (1 character)

    - `float` (single precision floating point number)

    - `double` (double precision floating point number)

Type Modifiers

- Types can be augmented by additional information

- Some simple "type qualifiers" are listed below:

  - **short** (applied to int)

  - **long** (applied to int and double)

  - **signed**

  - **unsigned** (only non-negative values)

  - **const** (specifies that the value cannot be changed)

- We usually drop the 'int' when specifying short or long

showsize demo

Output (sizes are in # of bytes)

```
mabdulla% ./size
    Data Type  Size Bytes            Min Value                  Max Value
         char      1                     -128                        127
unsigned char      1                        0                        255
        short      2                   -32768                      32767
          int      4              -2147483648                 2147483647
         long      4              -2147483648                 2147483647
    long long      8     -9223372036854775808        9223372036854775807
        float      4              1.17549e-38                3.40282e+38
       double      8             2.22507e-308               1.79769e+308
  long double     12             3.3621e-4932               1.18973e+4932
mabdulla%
```

Consttest demo

Language Operators

Operators Overview

- You are familiar with many operators from basic math and logic:
  - Addition (+), subtraction (-), multiplication (*), division (/)
  - AND (&&), OR (||), NOT (!)
- Operators are basically common functions that take their input and produce some output
- Common enough to have their own symbols in a programming language (see above)

Operators (Cont'd)

- C has many operators
  - Some you are familiar with (see previous page)
  - Some not: mod, bitwise AND, OR, XOR, relational

- Operators are:
  - Unary (take one argument, e.g.: !-)
  - Binary (take two arguments, e.g., +-*/<>==)
  - Ternary (take three arguments)

- Classifications:
  - Arithmetic, logic, relational, assignment

Operator Context

- Operators are represented by symbols. Sometimes, the symbols may mean something completely different based on context. For example:

```
int x = -1;     // the '-' operator is negation
int x = 4 - 3; // the '-' operator is subtraction
```

Arithmetic Operators

- Addition is represented by '+':

  - e.g., sum = x + y;

- Subtraction is represented by '-':

  - e.g., diff = x – y;

- Multiplication is represented by '*':

  - e.g., scale = x * y;

- Division is represented by '/':

  - e.g., quotient = x / y;

- Modulus is represented by '%':

  - e.g., remainder = x % y;

Relational Operators

- Assignment operator is '=': e.g., int sum = x;

- Equality operator is '==', e.g., is_equal = (x==y);

- Less than: '<'

- Greater than: '>'

- Less than or equal to: '<='

- Greater than or equal to: '>='

Logical Operator

- AND: (x && y)
- OR: (x || y)
- NOT: (!x)

Bitwise Operators

- Like logical operators, but operate on the individual bits of a variable, not the whole logical value.

```
Int x = 1;
int y = 2;
int r = x || y;
printf("r is: %d", r);
  Output is:  r is: 1
```

```
int x = 1;
int y = 2;
int r = x | y;
printf("r is: %d", r);
  Output is: r is 3
```

Bitwise Operators (Cont'd)

- Bitwise OR: |
- Bitwise AND: &
- Bitwise XOR: ^
- One's complement: ~
- Left shift: <<
- Right shift: >>

Order of Operations

- PEMDAS (power, exponent, mul, div, add, sub)

- For everything else, use parenthesis to say what you mean

- There are other rules. Learn them at your leisure while using the above two. See table 2.1 in TCPL (page 53)

Type Conversions (TCPL, 2.7)

- Key question is of the form: when I {add, sub, mul, div, mod...} and {int, float, long, ...} {with, from, by, ...} a {float, double, long, int...} what happens?

- Intermediate results are converted according to a set of rules. Basic rule is that the results are automatically "graduate" to the type of the larger operant.

Casting

- "Casting" is the process of forcing a type conversion

- Below, the integer value in "sum" is changed into a double type before being used, as is the result of the average score calculation:

```
int n = 100;
int sum = getsum();
double d = (double) sum;
double average = (double) sum / n;
```

Things We haven't Covered in this Section

- Increment and decrement operators

- Assignment operators

- The ternary condition operator

- Short circuit boolean evaluation

- The nuances of type conversion

- Collections of data types and variables (arrays, next lecture)