Readings K&R 1-5

Lab Learnings (20)

1. Basic Environment
2. Control Structures & Scalars

* Q&A about what people learned from Lab.

What you should have learned
+ Declarations
+ Control structures (K&R ch 3)

1. statements;
2. blocks & if - sequencing
3. conditional
   if (expression) statement
   else statement

   language concept of boolean
   - use ?

4. dispatch
   switch (expression) {
     case const-exp: statements
     default: statements
   }

5. while (expression) statement
6. for (expr; expr; expr; Next) statement

7. do statement
   while (expression);

8. functions

Outline

1. Lab Learnings
2. Concepts beneath this
   - scale values may
   - limits to finite rep
   - where data lives

- scalar vs complex
- C data types

  char - 1 byte, holds one "character" in local character set (ASCII)

  int - "integer", typically reflects the natural size of integers on host machine

  float - single precision FP, very specific meaning

  double

  type qualifier (like size or a wrapper class)

  length - guide representation

  short int x; typically 16, no bigger than int

  int y; typically 16 or 32, but sometimes 64

  long int z; at least 32

  specific sizes are defined by machine dependent include files

  "<system/types>" # include <sys/types.h>

  int8_t, uint16_t

  signed

  int x;

  signed int y; some
c

  unsigned int z; different meaning, especially comparison

  constant

  const unsigned short int y;

  What about boolean?

  - All types have special \( \phi \equiv \text{False}, \bar{\phi} \equiv \text{True} \)
- Literals

  - Program text representation of values

  + chars 'a'
    - Special chars have escape sequence
    \?, \", \n \<
    - Display control chars have escapes too
      \a - bell
      \b - backspace
      \f - formfeed
      \n - newline
      \r - carriage return
      \t - tab
      \v - vertical tab (???)

  - Explicit 8 bit values
    \000 \001
    null char \0

  - Symbolic constants
    + program names for constants
      enum boolean \& false, true \&
      \0 \& success

  + Numbers
    17 - decimal 17
    017 - octal 17 \& dec 15
    0x17 - hex 17 \& dec 23 \& 027

Example: EOF defined in stdio.h
Concepts beneath the Lab

- Meaning vs representation

**Characters**

- 'A' - prints as A
  - ! = 'b'
- ' ' - space, separates words
- '0' - the character zero

**Numbers**

- Whole Numbers
  - Zero element
  - Successor
  - Addition

- $n$-bit unsigned int

**Representation**

- 65 0101 0x41
- 32 0010 0x20
- 10 0010 0x0A
- 26 0101 0x10

**How many Chaus?**

- 61, 621, 0001, ... 00010610001...
- All mean the same

- $a - a = 0$
- $a + 0 = a$
- $a \times 0 = 0$
- $a \times 1 = a$

- $a + b = b + a$
- $a \neq b \Rightarrow a - b \neq 0$

- Comm.
- Assoc.
- Distrib.

- Finite set
- How many?
- Biggest

Map representation to "meaning"

- $b_{n-1} \cdots b_0 \rightarrow b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \cdots + b_0$
What happens when you go "off the end"?

4 bits

2-4

12 + 8
12
+ 8
---
0

C. overflow & underflow are silent

JAVA - No unsigned
- modular arithmetic
Signed Integer

- n bits $\Rightarrow 2^n$ values
  - which ones?

\[
-2^{n-1} = -2^{n-1} + 2^{n-1}
\]

1000

\[
\text{Subtraction}
\]

1111 = 0001

\[
\text{Addition}
\]

2

1

-2

What is the mapping?

\[b_{n-1}b_{n-2} \cdots b_0 \Rightarrow -2^{n-1} \cdot b_0 + 2^{n-2} \cdot b_1 + \cdots + 2 \cdot b_{n-2} + b_{n-1}\]

\[
\begin{array}{cccc}
\text{is} & 1000 & \rightarrow & 0011 \\
\text{un/\text{un}} & 9 & 3 & \text{True} \\
\text{int/\text{int}} & -7 & 3 & \text{False} \\
\text{int/\text{un}} & -7 & 3 & \text{False} \\
\text{un/\text{int}} & 9 & 3 & \text{True}
\end{array}
\]
Moving values of different sizes

\[
\begin{align*}
\frac{1}{16} & \\
\frac{1}{32} & \\
\end{align*}
\]

Can always treat a scalar like a larger one.

What goes in is:

\[
\text{unsigned} - \text{zero extend} \quad \{ \text{preserves meaning} \}
\]

\[
0 \cdot b^m + 2^m \cdot b^m + \cdots 2^m \cdot b^n = 2^n \cdot b^{m-1} + \cdots 2^m \cdot b^n.
\]

\[
\text{and all the zeros you like}
\]

Moving to smaller - you have to be explicit

\[\Rightarrow \text{ truncation} \]
Where data lives

C language concepts

```c
extern int c;  // declared here, but defined elsewhere
int b;
static int d;  // visible outside this file

Function foo() {
  int a;
}
```

Local "automatic", visible within function

Machine concepts

Memory

- The values of variables are held in memory
- Brought into registers
- Operated on
- Stored back

Putting the two together

Stack
- locals
- heap
- "static"
- code

What about "bigger" data types?

Witnessed & Understood by me, ________________________ Date __________
Invented by ________________________ Date __________
Recorded by ________________________