The Design of C:
A Rational Reconstruction:
Part 1
“C is quirky, flawed, and an enormous success. While accidents of history surely helped, it evidently satisfied a need for a system implementation language efficient enough to displace assembly language, yet sufficiently abstract and fluent to describe algorithms and interactions in a wide variety of environments.”

-- Dennis Ritchie

“When someone says, ‘I want a programming language in which I need only say what I want done,’ give him a lollipop.”

-- Alan Perlis
Goals of this Lecture

Help you learn about:

• The decisions that were made by the designers* of C
• **Why** they made those decisions
  … and thereby…
• The fundamentals of C

Why?

• Learning the design rationale of the C language provides a richer understanding of C itself
• A power programmer knows both the programming language and its design rationale

* Dennis Ritchie & members of standardization committees
Goals of C

<table>
<thead>
<tr>
<th>Designers wanted C to:</th>
<th>But also:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support system programming</td>
<td>Support application programming</td>
</tr>
<tr>
<td>Be low-level</td>
<td>Be portable</td>
</tr>
<tr>
<td>Be easy for people to handle</td>
<td>Be easy for computers to handle</td>
</tr>
</tbody>
</table>

Conflicting goals on multiple dimensions!
Agenda

Data Types
Operators
Statements
I/O Facilities
Issue: What primitive data types should C provide?

Thought process
- C will be used primarily for system programming, and so should handle:
  - Integers
  - Characters
  - Character strings
  - Logical (alias Boolean) data
- C might be used for application programming, and so should handle:
  - Floating-point numbers
  - C should be small/simple
Primitive Data Types

Decisions

• Provide integer data types
• Provide floating-point data types
• Do not (really) provide a character data type
• Do not provide a character string data type
• Do not provide a logical data type
Issue: What integer data types should C provide?

Thought process

- For flexibility, should provide integer data types of various sizes
- For portability at application level, should specify size of each data type
- For portability at system level, should define integer data types in terms of natural word size of computer
- Primary use will be system programming
Integers Data Types

Decisions

- Provide four integer data types: char, short, int, and long
- Type char is 1 byte
  - But number of bits per byte is unspecified!
- Do not specify sizes of others; instead:
  - int is natural word size
  - 2 <= (bytes in short) <= (bytes in int) <= (bytes in long)

On nobel using gcc217

- Natural word size: 4 bytes
- char: 1 byte
- short: 2 bytes
- int: 4 bytes
- long: 4 bytes

What decisions did the designers of Java make?
Issue: How should C represent integer literals?

Thought process
- People naturally use decimal
- System programmers often use binary, octal, hexadecimal
Integer Literals

Decisions
• Use decimal notation as default
• Use "0" prefix to indicate octal notation
• Use "0x" prefix to indicate hexadecimal notation
• Do not allow binary notation; too verbose, error prone
• Use "L" suffix to indicate long literal
• Do not use a suffix to indicate short literal; instead must use cast

Examples
• int: 123, 0173, 0x7B
• long: 123L, 0173L, 0x7BL
• short: (short)123, (short)0173, (short)0x7B
Unsigned Integer Data Types

Issue: Should C have both signed and unsigned integer data types?

Thought process

- Signed types are essential
  - Must represent positive and negative integers
- Unsigned types are useful
  - Unsigned data can be twice as large as signed data
  - Unsigned data are good for bit-level operations (common in system programming)
- Implementing both signed and unsigned data types is complex
  - Must define behavior when an expression involves both
Unsigned Integer Data Types

Decisions

• Provide unsigned integer types: unsigned char, unsigned short, unsigned int, and unsigned long
• Define conversion rules for mixed-type expressions
  • Generally, mixing signed and unsigned converts signed to unsigned
  • See King book Section 7.4 for details

What decisions did the designers of Java make?
Issue: How should C represent unsigned integer literals?

Thought process
• “L” suffix distinguishes long from int
• Also could use a suffix to distinguish signed from unsigned
Unsigned Integer Literals

Decisions
• Default is signed
• Use "U" suffix to indicate unsigned literal

Examples
• unsigned int:
  • 123U, 0173U, 0x7BU
• unsigned long:
  • 123UL, 0173UL, 0x7BUL
• unsigned short:
  • (unsigned short)123, (unsigned short)0173, (unsigned short)0x7B
# Signed and Unsigned Integer Literals

The rules:

<table>
<thead>
<tr>
<th>Literal</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd…d</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>unsigned long</td>
</tr>
<tr>
<td>0dd…d</td>
<td>int</td>
</tr>
<tr>
<td>0xdd…d</td>
<td>unsigned int</td>
</tr>
<tr>
<td></td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>unsigned long</td>
</tr>
<tr>
<td>0dd…dU</td>
<td>unsigned int</td>
</tr>
<tr>
<td>0xdd…dU</td>
<td>unsigned long</td>
</tr>
<tr>
<td>0d</td>
<td>unsigned int</td>
</tr>
<tr>
<td>dd…dL</td>
<td>long</td>
</tr>
<tr>
<td>0d</td>
<td>unsigned long</td>
</tr>
<tr>
<td>dd…dUL</td>
<td>unsigned long</td>
</tr>
<tr>
<td>0dd…dUL</td>
<td>unsigned long</td>
</tr>
<tr>
<td>0xdd…dUL</td>
<td>unsigned long</td>
</tr>
</tbody>
</table>

The type is the first one that can represent the literal without overflow.
Character Data Types

Issue: What character data types should C have?

Thought process

• The most common character codes are (were!) ASCII and EBCDIC
• ASCII is 7-bit
• EBCDIC is 8-bit
Character Data Types

Decision

• Use type `char`!
Character Literals

Issue: How should C represent character literals?

Thought process
- Could represent character literals as `int` literals, with truncation of high-order bytes
- More portable & readable to use single quote syntax ('a', 'b', etc.); but then…
- Need special way to represent the single quote character
- Need special ways to represent unusual characters (e.g. newline, tab, etc.)
Character Literals

Decisions

- Provide single quote syntax
- Use backslash (the escape character) to express special characters

Examples (with numeric equivalents in ASCII):

'|a| the a character (97, 01100001_B, 61_H)
'|\0141' the a character, octal character form
'|\x61' the a character, hexadecimal character form
'|b| the b character (98, 01100010_B, 62_H)
'|A| the A character (65, 01000001_B, 41_H)
'|B| the B character (66, 01000010_B, 42_H)
'|\0| the null character (0, 00000000_B, 0_H)
'|0| the zero character (48, 00110000_B, 30_H)
'|1| the one character (49, 00110001_B, 31_H)
'|n| the newline character (10, 00001010_B, A_H)
'|t| the horizontal tab character (9, 00001001_B, 9_H)
'|\| the backslash character (92, 01011100_B, 5C_H)
'|\'| the single quote character (96, 01100000_B, 60_H)
Issue: How should C represent strings and string literals?

Thought process
• Natural to represent a string as a sequence of contiguous chars
• How to know where char sequence ends?
  • Store length before char sequence?
  • Store special “sentinel” char after char sequence?
• C should be small/simple
Strings and String Literals

Decisions

- Adopt a convention
  - String is a sequence of contiguous chars
  - String is terminated with null char ("\0")
- Use double-quote syntax (e.g. "hello") to represent a string literal
- Provide no other language features for handling strings
  - Delegate string handling to standard library functions

Examples

- 'a' is a char literal
- "abcd" is a string literal
- "a" is a string literal

How many bytes?

What decisions did the designers of Java make?
Logical Data Type

Issue: How should C represent logical data?

Thought process
• Representing a logical value (TRUE or FALSE) requires only one bit
• Smallest entity that can be addressed is one byte
• Type char is one byte, so could be used to represent logical values
• C should be small/simple
Logical Data Type

Decisions
- Don't define a logical data type
- Represent logical data using type `char`
  - Or any integer type
  - Or any primitive type!!!
- Convention: 0 => FALSE, non-0 => TRUE
- Convention used by:
  - Relational operators (<, >, etc.)
  - Logical operators (!, &&, ||)
  - Statements (if, while, etc.)
Aside: Logical Data Type Shortcuts

Note

• Using integer data to represent logical data permits shortcuts

```c
...  
int i;  
...  
if (i)  /* same as (i != 0) */  
    statement1;  
else  
    statement2;  
...```
Aside: Logical Data Type Dangers

Note

- The lack of logical data type hampers compiler's ability to detect some errors with certainty

```java
... int i;
...
i = 0;
...
if (i = 5)
    statement1;
...
```

What happens in Java?

What happens in C?
Issue: What floating-point data types should C have?

Thought process

- System programs use floating-point data infrequently
- But some application domains (e.g. scientific) use floating-point data often
- C should support system programming primarily
- But why not allow C to support application programming?
- For portability at application level, should specify size of each data type
- For portability at system level, should define floating point data types as natural for underlying hardware
Floating-Point Data Types

Decisions

• Provide three floating-point data types: float, double, and long double
• Don’t specify sizes
• bytes in float <= bytes in double <= bytes in long double

On nobel using gcc217

• float: 4 bytes
• double: 8 bytes
• long double: 12 bytes
Floating-Point Literals

Issue: How should C represent floating-point literals?

Thought process
• Convenient to allow both fixed-point and scientific notation
• Decimal is sufficient; no need for octal or hexadecimal
Floating-Point Literals

Decisions
- Allow fixed-point and scientific notation
- Any literal that contains decimal point or "E" is floating-point
- The default floating-point type is double
- Append "F" to indicate float
- Append "L" to indicate long double

Examples
- double: 123.456, 1E-2, -1.23456E4
- float: 123.456F, 1E-2F, -1.23456E4F
- long double: 123.456L, 1E-2L, -1.23456E4L
Data Types Summary: C vs. Java

Java only
- boolean, byte

C only
- unsigned char, unsigned short, unsigned int, unsigned long

Sizes
- Java: Sizes of all types are specified
- C: Sizes of all types except char are system-dependent

Type char
- Java: char consists of 2 bytes
- C: char consists of 1 byte
Continued next lecture