Princeton University

Computer Science 217: Introduction to Programming Systems

Data Types in C

Goals of C

Designers wanted C to:	But also:
Support system programming	Support application programming
Be low-level	Be portable
Be easy for people to handle	Be easy for computers to handle

- Conflicting goals on multiple dimensions!
- Result: different design decisions than Java

Primitive Data Types

- **integer** data types
- **floating-point** data types
- **pointer** data types
- **no character** data type (use small integer types instead)
- **no** character **string** data type (use arrays of small ints instead)
- **no logical or boolean** data types (use integers instead)

For "under the hood" details,
stay tuned for
"number systems" lecture
next week

Integer Data Types

Integer types of various sizes: **`signed char`**, **`short`**, **`int`**, **`long`**

- **`char`** is 1 byte
  - Number of bits per byte is unspecified!
    (but in the 21st century, pretty safe to assume it's 8)
- Sizes of other integer types not fully specified but *constrained:*
  - **`int`** was intended to be "natural word size"
  - 2 ≤ sizeof(**`short`**) ≤ sizeof(**`int`**) ≤ sizeof(**`long`**)

On ArmLab:

- Natural word size: 8 bytes ("64-bit machine")
- **`char`**: 1 byte
- **`short`**: 2 bytes
- **`int`**: 4 bytes (compatibility with widespread 32-bit code)
- **`long`**: 8 bytes

What decisions did the designers of Java make?

Integer Literals

- Decimal: 123
- Octal: 0173 = 123
- Hexadecimal: 0x7B = 123
- Use "L" suffix to indicate **`long`** literal
- No 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

unsigned types: **`unsigned char`**, **`unsigned short`**, **`unsigned int`**, and **`unsigned long`**

- Holds only non-negative integers
- Conversion rules for mixed-type expressions
  (Generally, mixing signed and unsigned converts to unsigned)
- See King book Section 7.4 for details

Unsigned Integer Literals

Default is signed

- Use "U" suffix to indicate unsigned literal

Examples

- **`unsigned int`**:
  - **`123U`**, **`0173U`**, **`0x7BU`**
  - **`123, 0173, 0x7B`** will work just fine in practice; technically there is an implicit cast from signed to unsigned, but in these cases it shouldn't make a difference.
- **`unsigned long`**:
  - **`123UL`**, **`0173UL`**, **`0x7BUL`**
- **`unsigned short`**:
  - **`(unsigned short)123`**, **`(unsigned short)0173`**, **`(unsigned short)0x7B`**

"Character" Data Type

The C `char` type

- **`char`** is designed to hold an ASCII character
  - And should be used when you're dealing with characters: character-manipulation functions we've seen (such as **`toupper`**) take and return **`char`**
- **`char`** might be signed (-128..127) or unsigned (0..255)
  - But since 0 ≤ ASCII ≤ 127 it doesn't really matter
- If you want a 1-byte type for *calculation,* you might (should?) specify **`signed char`** or **`unsigned char`**

Character Literals

Single quote syntax: **`'a'`**

Use backslash (the **escape character**) to express special characters

- Examples (with numeric equivalents in ASCII):

```
'a'    the a character (97, 01100001B, 61H)
'\141' the a character, octal form
'\x61' the a character, hexadecimal form
'b'    the b character (98, 01100010B, 62H)
'A'    the A character (65, 01000001B, 41H)
'B'    the B character (66, 01000010B, 42H)
'\0'   the null character (0, 00000000B, 0H)
'0'    the zero character (48, 00110000B, 30H)
'1'    the one character (49, 00110001B, 31H)
'\n'   the newline character (10, 00001010B, AH)
'\t'   the horizontal tab character (9, 00001001B, 9H)
'\\'   the backslash character (92, 01011100B, 5CH)
'\''   the single quote character (96, 01100000B, 60H)
```

Strings and String Literals

Issue: How should C represent strings and string literals?

Rationale:

- Natural to represent a string as a sequence of contiguous chars
- How to know where char sequence ends?
  - Store length together with char sequence?
  - Store special "sentinel" char after char sequence?

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

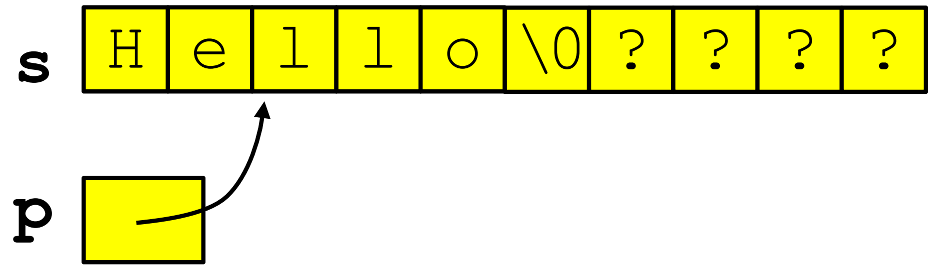
Examples

- **`'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?

Arrays of characters




```
char s[10] = {'H','e','l','l','o',0};
```

(or, equivalently)

```
char s[10] = "Hello";
```

```
char *p = s+2;
```

p is a **pointer**: it contains the *address* of another variable

```
printf("Je%s!", p);
```

prints **Jello!**

Unicode

Back in 1970s, English was the only language in the world[citation needed], so we all used this alphabet:

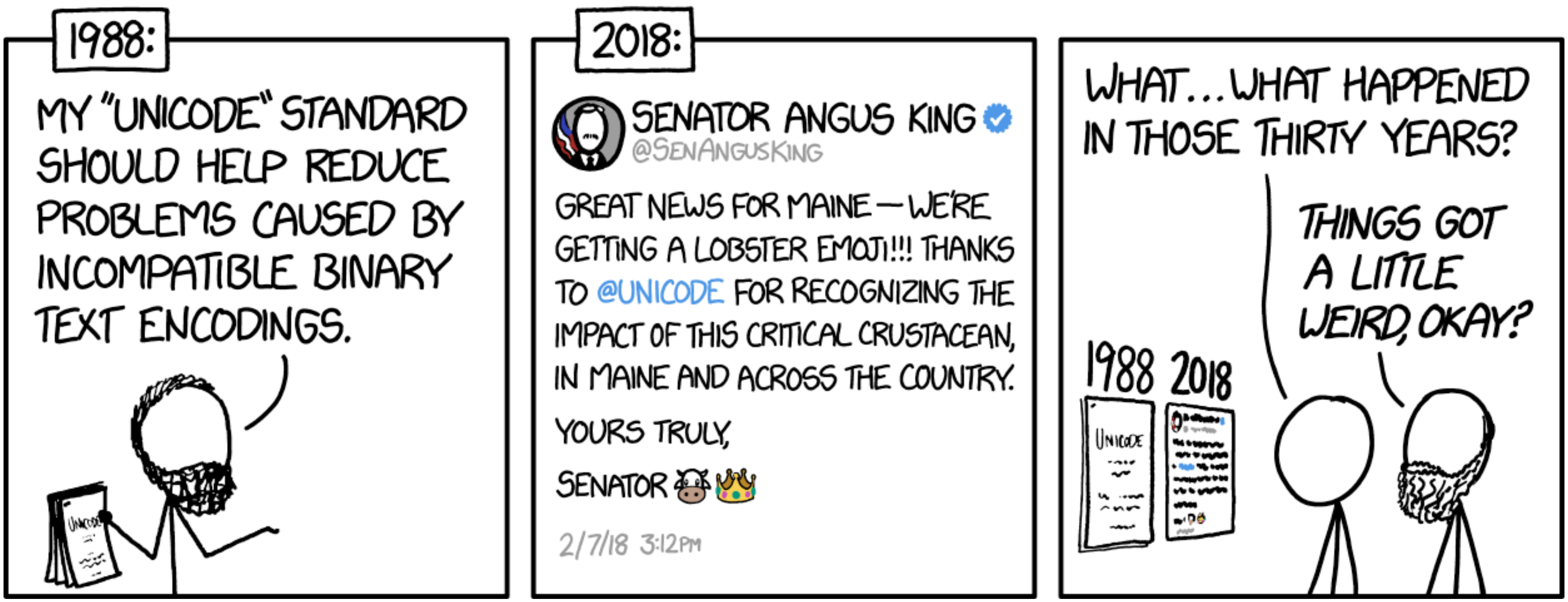
20	21 !	22 "	23 #	24 $	25 %	26 &	27 '	28 (	29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	7B {	7C |	7D }	7E ~	

ASCII: American Standard Code for Information Interchange

In the 21st century, it turns out that there are other people and languages out there, so we need:

Modern Unicode

When Java was designed, Unicode fit into 16 bits, so **`char`** in Java was 16 bits long. Then this happened:



https://xkcd.com/1953/

Unicode and UTF-8

Lots of characters in today's Unicode

- 100,000+ defined, capacity for > 1 million

Can't modify size of `char` in C

Solution: variable-length encoding (UTF-8)

- Standard ASCII characters use 1 byte
- Most Latin-based alphabets use 2 bytes
- Chinese, Japanese, Korean characters use 3 bytes
- Historic scripts, mathematical symbols, and emoji use 4 bytes
- This won't be on the exam!

Logical Data Types

No separate logical or Boolean data type

Represent logical data using type **`char`** or **`int`**

- Or any integer type
- Or any primitive type!

Conventions:

- Statements (**`if`**, **`while`**, etc.) use 0 ⇒ FALSE, ≠0 ⇒ TRUE
- Relational operators (**`<`**, **`>`**, etc.) and logical operators (**`!`**, **`&&`**, **`||`**) produce the result 0 or 1

Logical Data Type Shortcuts

Using integers to represent logical data permits shortcuts

```
…
int i;
…
if (i)  /* same as (i != 0) */
   statement1;
else
   statement2;
…
```

It also permits some *really* bad code…

```
i = (1 != 2) + (3 > 4);
```

iClicker Question

Q: What is `i` set to in the following code?

```
i = (1 != 2) + (3 > 4);
```

A. 0

B. 1

C. 2

D. 3

E. 4

Logical Data Type Dangers

The lack of a logical data type hampers compiler's ability to detect some errors

```
…
int i;
…
i = 0;
…
if (i = 5)
   statement1;
…
```

What happens in Java?

What happens in C?

Floating-Point Data Types

C specifies:

- Three floating-point data types:
  `float`, **`double`**, and **`long double`**
- Sizes unspecified, but constrained:
  sizeof(**`float`**) ≤ sizeof(**`double`**) ≤ sizeof(**`long double`**)

On ArmLab (and on pretty much any 21st-century computer using the IEEE standard)

- **`float`**: 4 bytes
- **`double`**: 8 bytes

On ArmLab (but varying a lot across architecures)

- **`long double`**: 16 bytes

Floating-Point Literals

How to write a floating-point number?

- Either fixed-point or "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`**, **`long double`**

Sizes

- **Java**: Sizes of all types are specified, and *portable*
- **C**: Sizes of all types except **`char`** are system-dependent

Type char

- **Java**: **`char`** is 2 bytes (to hold all 1995-era Unicode values)
- **C**: **`char`** is 1 byte