

Announcements

- Program #3
 - Is on the web
- Reading
 - Notes (Today)
 - Chapter 16 (Tuesday)

Building a Test Suite

- API:

- Int createEmployee(char *lastName, char *firstName);
 - Return unique id for that employee
- Int lookupEmployee(char *lastName, char *firstName, in *id);
 - Return 0 on success, -1 on failure (not found or > 1 match)
 - Fills out id on success
 - firstName can be null
- Int deleteEmployee(int id);
- Int setSalary(int id, float salary);

Possible Test Cases

- `Int createEmployee(char *lastName, char *firstName);`
 - Pass valid data
 - Pass same last name, different first name
 - Pass same last name, first name same as previous
 - Pass null for lastName and/or for firstName
 - Pass an lastName/firstName that has 10MB before null termination.
 - Create 10,000 employees

Test Cases Continued

- `Int lookupEmployee(char *lastName, char *firstName, in *id);`
 - Call before inserting anyone
 - Lookup someone that was created
 - Lookup someone with the same lastname, but different firstName
 - Lookup null last/first names
 - Pass id as null
- `Int deleteEmployee(int id);`
 - Pass valid id
 - Pass invalid id
 - Pass id of someone who has just been deleted
- `Int setSalary(int id, float salary);`
 - Invalid id
 - Negative salary

Representing Characters

- Need
 - Represent common characters
 - Have standards so computers can interoperate
- Common Formats
 - ASCII
 - 7 bits for characters (stored in 8 bits normally)
 - most commonly used
 - UNICODE
 - family of encodings 8, 16, and 32 bits/character
 - allow greater variety of characters
 - Able to represent virtually character in use today
 - and some no longer in use

ASCII

- Represents normal characters on US keyboards
 - A-Z (101-132)
 - a-z (141-172)
 - 0-9 (60-71)
 - punctuation: !@#\$%^&*()_+ -=[]{}| \ ; : " ' < > ? , . /
 - space
 - <control-A> - <control-Z>
 - Also have other names <control-M> is CR (\r)

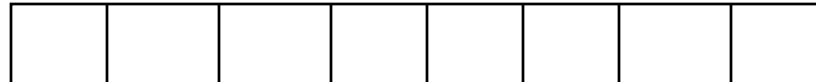
UNICODE

- Unicode Representations

- UTF-32 32 bit representation of all characters
 - all characters are the same size
 - wastes lots of space (2x UTF-16 for most things, 4x ascii for many things)
- UTF-16 16 bit representation of characters
 - some characters are stored in two-character forms
 - popular since most things can be represented in 16 bits
- UTF-8 8 bit representation of characters
 - provides backwards compatibility with ascii
 - low 7 bits are exactly ASCII
 - high bit on indicates part of UNICODE extensions
 - popular for web and other applications

Representing Integers

- All data stored in binary
 - all numbers are 0 or 1
- Unsigned numbers are stored using base 2



128 64 32 16 8 4 2 1

- possible range 0 to 2^n where n is the number of bits
- Signed numbers are stored using two's complement
 - left most bit indicates if a number is positive or negative
 - 0 is positive
 - 1 is a negative number

Adding Binary Numbers

- Add starting from right
- Carry if the number is too large to represent
 - if it is greater than 1

1001

+ 10

1001

+ 11

1011

+ 11

1101

+111

2's complement representation

- To compute a negative value:
 - flip all the bits of the positive value, add 1
- Allows addition of signed and unsigned numbers
- Valid range of numbers
 - -2^{n-1} to $2^{n-1}-1$ for n bits
 - Example: 16 bit number -32,768 to 32,767

How do we represent real numbers?

- Each number has two parts
 - mantissa (represents a number between -1 and 1)
 - represented as a binary number i.e. $0.1 = 1/2$
 - exponent (designates the position of the decimal point)
 - uses normal two's complement form
 - number = $m r^e$ where r is the radix
 - $6132.789 = +0.6132789 \times 10^4$
- Normalization
 - convert to number between -1 and 1
 - if the most significant digit of mantissa is non-zero.

Floating Point Continued

- Computers normally use a radix of 2
- Examples of floating point numbers
 - $1001.11 = .1001110 \times 2^4$
 - $10.5 = 1010.1$ or 10101×2^4
 - $17.451 = 111.011100 = 1110111 \times 2^3$
- IEEE Floating point standard
 - 32 bit floating point (float)
 - 1 sign bit, 8 bits exponent, 23 bits mantissa
 - 64 bit floating point (double)
 - 1 sign bit, 11 bits exponent, 52 bits mantissa
 - most common for real applications
 - 128 bit floating point (quad)
 - 1 sign bit, 15 bits exponent, 112 bits of mantissa