Integral Data Types in C

Professor Jennifer Rexford

http://www.cs.princeton.edu/~jrex

Goals for this Lecture

- Binary number system
  - Why binary?
  - Converting between decimal and binary
  - … and octal and hexadecimal number systems
- Finite representations of binary integers
  - Unsigned and signed integers
  - Integer addition and subtraction
- Bitwise operators
  - AND, OR, NOT, and XOR
  - Shift-left and shift-right
- The C integral data types
  - char, short, int, long
  - signed and unsigned variants

Why Bits (Binary Digits)?

- Computers are built using digital circuits
  - Inputs and outputs can have only two values
  - True (high voltage) or false (low voltage)
  - Represented as 1 and 0
- Can represent many kinds of information
  - Boolean (true or false)
  - Numbers (23, 79, …)
  - Characters (‘a’, ‘z’, …)
  - Pixels, sounds
  - Internet addresses
- Can manipulate in many ways
  - Read and write
  - Logical operations
  - Arithmetic

3

Base 10 and Base 2

- Decimal (base 10)
  - Each digit represents a power of 10
  - $4173 = 4 \times 10^3 + 1 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$
- Binary (base 2)
  - Each bit represents a power of 2
  - $10110 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22$

Decimal to binary conversion:
Divide repeatedly by 2 and keep remainders

```
12/2 = 6   R = 0
6/2  = 3   R = 0
3/2  = 1   R = 1
1/2  = 0   R = 1
Result = 1100
```

4

Writing Bits is Tedious for People

- Octal (base 8)
  - Digits 0, 1, …, 7
- Hexadecimal (base 16)
  - Digits 0, 1, …, 9, A, B, C, D, E, F

0000 = 0	**1000 = 8**
0001 = 1	**1001 = 9**
0010 = 2	**1010 = A**
0011 = 3	**1011 = B**
0100 = 4	**1100 = C**
0101 = 5	**1101 = D**
0110 = 6	**1110 = E**
0111 = 7	**1111 = F**

Thus the 16-bit binary number

1011 0010 1010 1001

converted to hex is

B2A9

5

Representing Colors: RGB

- Three primary colors
  - Red
  - Green
  - Blue
- Strength
  - 8-bit number for each color (e.g., two hex digits)
  - So, 24 bits to specify a color
- In HTML, on the course Web page
  - Red: <i>Symbol Table Assignment Due</i>
  - Blue: <i>Spring Break</i>
- Same thing in digital cameras
  - Each pixel is a mixture of red, green, and blue

6

Finite Representation of Integers

- Fixed number of bits in memory
  - Usually 8, 16, or 32 bits
  - (1, 2, or 4 bytes)
- Unsigned integer
  - No sign bit
  - Always 0 or a positive number
  - All arithmetic is modulo 2^n
- Examples of unsigned integers
  - 00000001 ➔ 1
  - 00001111 ➔ 15
  - 00010000 ➔ 16
  - 00100001 ➔ 33
  - 11111111 ➔ 255

Adding Two Integers

- From right to left, we add each pair of digits
- We write the sum, and add the carry to the next column

Base 10

	1	9	8
+	2	6	4
Sum	4	6	2
Carry	0	1	1

Base 2

	0	1	1
+	0	0	1
Sum	1	0	0
Carry	0	1	1

Binary Sums and Carries

a	b	Sum
0	0	0
0	1	1
1	0	1
1	1	0

XOR
("exclusive OR")

a	b	Carry
0	0	0
0	1	0
1	0	0
1	1	1

AND

```
  0100 0101  <----  69
+ 0110 0111  <----  103
-----------
  1010 1100  <----  172
```

9

Modulo Arithmetic

- Consider only numbers in a range
  - E.g., five-digit car odometer: 0, 1, …, 99999
  - E.g., eight-bit numbers 0, 1, …, 255
- Roll-over when you run out of space
  - E.g., car odometer goes from 99999 to 0, 1, …
  - E.g., eight-bit number goes from 255 to 0, 1, …
- Adding 2^n doesn't change the answer
  - For eight-bit number, n=8 and 2^n=256
  - E.g., (37 + 256) mod 256 is simply 37
- This can help us do subtraction…
  - Suppose you want to compute a – b
  - Note that this equals a + (256 -1 - b) + 1

10

One's and Two's Complement

- One's complement: flip every bit
  - E.g., b is 01000101 (i.e., 69 in decimal)
  - One's complement is 10111010
  - That's simply 255-69
- Subtracting from 11111111 is easy (no carry needed!)

```
  1111 1111
- 0100 0101  <---- b
-----------
  1011 1010  <---- one's complement
```

- Two's complement
  - Add 1 to the one's complement
  - E.g., (255 – 69) + 1 → 1011 1011

11

Putting it All Together

- Computing "a – b"
  - Same as "a + 256 – b"
  - Same as "a + (255 – b) + 1"
  - Same as "a + onesComplement(b) + 1"
  - Same as "a + twosComplement(b)"
- Example: 172 – 69
  - The original number 69: 0100 0101
  - One's complement of 69: 1011 1010
  - Two's complement of 69: 1011 1011
  - Add to the number 172: 1010 1100
  - The sum comes to: 0110 0111
  - Equals: 103 in decimal

```
   1010 1100
+  1011 1011
------------
1  0110 0111
```

12

Signed Integers

- Sign-magnitude representation
  - Use one bit to store the sign
    - Zero for positive number
    - One for negative number
  - Examples
    - E.g., 0010 1100 ➔ 44
    - E.g., 1010 1100 ➔ -44
  - Hard to do arithmetic this way, so it is rarely used
- Complement representation
  - One's complement
    - Flip every bit
    - E.g., 1101 0011 ➔ -44
  - Two's complement
    - Flip every bit, then add 1
    - E.g., 1101 0100 ➔ -44

13

Overflow: Running Out of Room

- Adding two large integers together
  - Sum might be too large to store in the number of bits available
  - What happens?
- Unsigned integers
  - All arithmetic is "modulo" arithmetic
  - Sum would just wrap around
- Signed integers
  - Can get nonsense values
  - Example with 16-bit integers
    - Sum: 10000+20000+30000
    - Result: -5536

14

Bitwise Operators: AND and OR

- Bitwise AND (&)

&	0	1
0	0	0
1	0	1

  - Mod on the cheap!
    - E.g., 53 % 16
    - … is same as 53 & 15;

53	0	0	1	1	0	1	0	1
& 15	0	0	0	0	1	1	1	1
5	0	0	0	0	0	1	0	1

- Bitwise OR (|)

|	0	1
0	0	1
1	1	1

15

Bitwise Operators: Not and XOR

- One's complement (~)
  - Turns 0 to 1, and 1 to 0
  - E.g., set last three bits to 0
    - x = x & ~7;
- XOR (^)
  - 0 if both bits are the same
  - 1 if the two bits are different

^	0	1
0	0	1
1	1	0

16

Bitwise Operators: Shift Left/Right

- Shift left (<<): Multiply by powers of 2
  - Shift some # of bits to the left, filling the blanks with 0

53	0	0	1	1	0	1	0	1
53<<2	1	1	0	1	0	0	0	0

- Shift right (>>): Divide by powers of 2
  - Shift some # of bits to the right
    - For unsigned integer, fill in blanks with 0
    - What about signed negative integers? Varies across machines…
      - Can vary from one machine to another!

53	0	0	1	1	0	1	0	1
53>>2	0	0	0	0	1	1	0	1

Example: Counting the 1's

- How many 1 bits in a number?
  - E.g., how many 1 bits in the binary representation of 53?

0	0	1	1	0	1	0	1

  - Four 1 bits
- How to count them?
  - Look at one bit at a time
  - Check if that bit is a 1
  - Increment counter
- How to look at one bit at a time?
  - Look at the last bit: n & 1
  - Check if it is a 1: (n & 1) == 1, or simply (n & 1)

Counting the Number of '1' Bits

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   unsigned n, count;

   printf("Number: ");
   if (scanf("%u", &n) != 1) {
      fprintf(stderr, "Error: Expect number.\n");
      exit(EXIT_FAILURE);
   }

   for (count=0; n; n >>= 1)
      count += (n & 1);

   printf("Number of 1 bits: %u\n", count);
   return 0;
}
```

Data Types

- Programming languages combine:
  - Bits into bytes
  - Bytes into larger entities
- Combinations of bytes have types; why?
  - Facilitates abstraction
  - Enables compiler to do type checking
- C has 11 primitive data types
  - 8 integral data types (described in this lecture)
    - Four different sizes (char, short, int, and long)
    - Signed vs. unsigned
  - 3 floating-point data types (described in next lecture)

C Integral Data Types

- Why char vs. short vs. int vs. long?
  - Small sizes conserve memory
  - Large sizes provide more range
- Why signed vs. unsigned?
  - Signed types allow negatives
  - Unsigned types allow larger positive numbers
  - (Dubious value: Java omits unsigned types)
- When to use unsigned?
  - When you really need that extra bit
  - When you'll do lots of bit shifting
  - When you'll never do (a < 0) test

21

C Integral Data Types (continued)

- Integral types:

Type	Bytes	Typically Used to Store
signed char	1	The numeric code of a character
unsigned char	1	The numeric code of a character
(signed) short	2*	A small integer
unsigned short	2*	A small non-negative integer
(signed) int	4*	An integer
unsigned int	4*	A non-negative integer
(signed) long	4*	An integer
unsigned long	4*	A non-negative integer

* On hats; C90 standard does not specify size

22

The int Data Type

- Description: A positive or negative integer
  - Same as signed int
- Size: System dependent
  - 16 <= bits in short <= bits in int <= bits in long
  - Usually 16 bits (alias 2 bytes) or 32 bits (alias 4 bytes)
  - The "natural word size" of the computer

23

The int Data Type (cont.)

- Example constants (assuming 4 bytes)

Two's complement

High-order bit indicates sign

Constant	Binary Representation	Note
`123`	`00000000 00000000 00000000 01111011`	`decimal form`
`-123`	`11111111 11111111 11111111 10000101`	`negative form`
`2147483647`	`01111111 11111111 11111111 11111111`	`largest`
`-2147483648`	`10000000 00000000 00000000 00000000`	`smallest`
`0173`	`00000000 00000000 00000000 01111011`	`octal form`
`0x7B`	`00000000 00000000 00000000 01111011`	`hexadecimal form`

Leading zero means octal

Leading zero-x means hexadecimal

24

The unsigned int Data Type

- Description: A positive integer
- Size: System dependent
  - Same as int
- Example constants (assuming 4 bytes)

Constant	Binary Representation	Note
123U	00000000 00000000 00000000 01111011	decimal form
4294967295U	11111111 11111111 11111111 11111111	largest
0U	00000000 00000000 00000000 00000000	smallest
0173U	00000000 00000000 00000000 01111011	octal form
0x7BU	00000000 00000000 00000000 01111011	hexadecimal form

Note "U" suffix

Same range as int, but shifted on number line

25

The long Data Type

- Description: A positive or negative integer
  - Same as signed long
- Size: System dependent
  - 16 <= bits in short <= bits in int <= bits in long
  - Usually 32 bits, alias 4 bytes
- Example constants (assuming 4 bytes)

Constant	Binary Representation	Note
123L	00000000 00000000 00000000 01111011	decimal form
-123L	11111111 11111111 11111111 10000101	negative form
2147483647L	01111111 11111111 11111111 11111111	largest
-2147483648L	10000000 00000000 00000000 00000000	smallest
0173L	00000000 00000000 00000000 01111011	octal form
0x7BL	00000000 00000000 00000000 01111011	hexadecimal form

Note "L" suffix

26

The unsigned long Data Type

- Description: A positive integer
- Size: System dependent
  - Same as long
- Example constants (assuming 4 bytes)

Constant	Binary Representation	Note
`123UL`	`00000000 00000000 00000000 01111011`	`decimal form`
`4294967295UL`	`11111111 11111111 11111111 11111111`	`largest`
`0UL`	`00000000 00000000 00000000 00000000`	`smallest`
`0173UL`	`00000000 00000000 00000000 01111011`	`octal form`
`0x7BUL`	`00000000 00000000 00000000 01111011`	`hexadecimal form`

Note “UL” suffix

27

The short Data Type

- Description: A positive or negative integer
  - Same as signed short
- Size: System dependent
  - 16 <= bits in short <= bits in int <= bits in long
  - Usually 16 bits, alias 2 bytes
- Example constants (assuming 2 bytes)

Constant	Binary Representation	Note
`(short)123`	`00000000 01111011`	`decimal form`
`(short)-123`	`11111111 10000101`	`negative form`
`(short)32767`	`01111111 11111111`	`largest`
`(short)-32768`	`10000000 00000000`	`smallest`
`(short)0173`	`00000000 01111011`	`octal form`
`(short)0x7B`	`00000000 01111011`	`hexadecimal form`

No way to express constant of type short, so must use cast

28

The unsigned short Data Type

- Description: A positive integer
- Size: System dependent
  - Same as short
- Example constants (assuming 4 bytes)

Constant	Binary Representation	Note
`(unsigned short)123U`	`00000000 01111011`	`decimal form`
`(unsigned short)65535U`	`11111111 11111111`	`largest`
`(unsigned short)0U`	`00000000 00000000`	`smallest`
`(unsigned short)0173U`	`00000000 01111011`	`octal form`
`(unsigned short)0x7BU`	`00000000 01111011`	`hexadecimal form`

No way to express constant of type unsigned short, so must use cast

29

The signed char Data Type

- Description: A (small) positive or negative integer
- Size: 1 byte
- Example constants

Constant	Binary Representation	Note
`(signed char)123`	`01111011`	`decimal form`
`(signed char)-123`	`10000101`	`negative form`
`(signed char)127`	`01111111`	`largest`
`(signed char)-128`	`10000000`	`smallest`
`(signed char)0173`	`01111011`	`octal form`
`(signed char)0x7B`	`01111011`	`hexadecimal form`

No way to express constant of type signed char, so must use cast

30

The unsigned char Data Type

- Description: A (small) positive integer
- Size: 1 byte
- Example constants

Constant	Binary Representation	Note
`(unsigned char)123`	`01111011`	`decimal form`
`(unsigned char)255`	`11111111`	`largest`
`(unsigned char)0`	`00000000`	`smallest`
`(unsigned char)0173`	`01111011`	`octal form`
`(unsigned char)0x7B`	`01111011`	`hexadecimal form`

No way to express constant of type unsigned char, so must use cast

The char Data Type

- On some systems, char means signed char
- On other systems, char means unsigned char
- Obstacle to portability

```
int a[256];
char c;
c = (char)255;
…
… a[c] …
   /* char is unsigned => a[255] => OK */
   /* char is signed => a[-1] => out of bounds */
```

The char Data Type (cont.)

- On your system, is char signed or unsigned?

```
#include <stdio.h>
int main(void) {
   char c = (char)0x80;
   if (c > 0)
      printf("unsigned");
   else
      printf("signed");
   return 0;
}
```

- Output on hats

```
signed
```

33

The char Data Type (cont.)

- Q:
  - Why is type char called “char” (meaning “character”)?
- A:
  - Type char can be used for limited range arithmetic
    - As indicated previously
  - However, type char is used more often to store a **character code**
    - As shown next lecture

34

Summary

- Computer represents everything in binary
  - Integers, floating-point numbers, characters, addresses, …
  - Pixels, sounds, colors, etc.
- Binary arithmetic through logic operations
  - Sum (XOR) and Carry (AND)
  - Two’s complement for subtraction
- Binary operations in C
  - AND, OR, NOT, and XOR
  - Shift left and shift right
  - Useful for efficient and concise code, though sometimes cryptic
- C integral data types
  - char, short, int, long (signed and unsigned)

The Rest of the Week

- Reading
  - Required: *C Programming*: 4, 5, 6, 7, 14, 15, and 20.1
  - Recommended: *Computer Systems*: 2
  - Recommended: *Programming with GNU Software*: 3, 6
- Monday office hours
  - My office hours by appointment, instead of usual 4:30pm
- Wednesday’s lecture
  - C Fundamentals
- Programming assignment
  - A “Decomment” Program
  - Due Sunday at 9pm