Operators

CS 217

Arithmetic Operators

• Binary arithmetic operators: + * /
• Modulus (remainder) operator: %
  \( x \% y \)
  is the remainder when \( x \) is divided by \( y \)
  well defined only when \( x > 0 \) and \( y > 0 \)
• Unary operators: - +
• Precedence: unary higher than binary
  \(-2*a+b\)
  is parsed as \(((\neg 2)*a)+b\)
• Associativity: left to right
  \(a+b+c\)
  is parsed as \(((a+b)+c)\)

Portability

• Print a number in decimal
  
  ```c
  void putd(int n) {
    if (n < 0) {
      putchar('-');
      n = -n;
    }
    if (n >= 10)
      putd(n/10);
      putchar(n%10 + '0');
  }
  ```
• Can this program print
  \(\text{INT_MIN} = -2147483648\)
Machine Arithmetic

- Computer values are of fixed-length (32-bits)
- For example, with 6-bits (0..5, right to left)
  largest number: \(111111_2 = 63_{10} = 2^6 - 1\)
  smallest number: \(000000_2 = 0\)
- What is 50 + 20?
  \[
  \begin{array}{c}
  110010 \\
  + 010100 \\
  \hline
  1000110
  \end{array}
  \]
- Spilling over the lefthand side is overflow

Machine Arithmetic (cont)

- Sign-magnitude notation
  bit \(n-1\) is the sign; 0 for +, 1 for -
  bits \(n-2\) through 0 hold an unsigned number
  largest number: \(01111...111_2 = 2^{n-1} - 1\)
  smallest number: \(11111...111_2 = -(2^{n-1} - 1)\)
  addition and subtraction are complicated when signs differ, so sign-magnitude is rarely used

Machine Arithmetic (cont)

- One’s-complement notation
  bit \(n-1\) is the sign; bits \(n-2...0\) hold an unsigned number
  bits \(n-2...0\) hold the complement of negative numbers
  \(-k = (2^n - 1) - k = 1111...111 - k\)
  largest number: \(01111...111_2 = 2^{n-1} - 1\)
  smallest number: \(10000...000_2 = -(2^{n-1} - 1)\)
  addition and subtraction are easy, but there are two representations for 0
Machine Arithmetic (cont)

- Two’s-complement notation
  bit $n$-1 is the sign; bits $n-2..0$ hold an unsigned number
  bits $n-2..0$ hold the complement of negative numbers +1
  $-k = (2^n - k) = (2^n - 1) - k + 1$
  largest number: $01111...111 = 2^n - 1$
  smallest number: $10000...000 = -2^{n-1}$
  6-bit examples: “complement and increment” to negate
  +6 000110 111001 111010 -6
  -6 111010 00101 000110 +6
  +0 000000 111111 000000 +0
  +31 011111 100000 100001 -31
  -32 100000 011111 100000 -32

Machine Arithmetic (cont)

- To add 2’s-complement number, ignore signs
  and add the unsigned bit strings
  +20 010100 -20 101100
  +13 001001 -13 101100
  +3 001000 -3 101100
  +7 000111 -7 110000
  +27 011111 -27 100101

- Signed overflow occurs if the carry into the sign bit
  differs from the carry out of the sign bit
  +20 010100 -20 101100
  +31 010001 -31 101111
  -27 100101 +27 011011

Return to putd Example

- Convert negative numbers
  static void putneg(int n) {
    putneg(n/10);
    putchar("0123456789[\-(n%10)]");
  }
  void putd(int n) {
    if (n < 0) {
      putchar('-');
      putneg(n);
    } else
      putneg(-n);
  }
Portability (cont)

• n/10 and n%10 are implementation dependent when n < 0

```c
int a, b, q, r;
q = a/b; r = a%b;
```

ANSI Standard guarantees only that

- \(q \times b + r = a\)
- \(|r| < |b|\)
- \(r \geq 0\) when \(a \geq 0 \land b > 0\)
- \(r\) might be negative if \(a\) is negative

5 / (-3) = -1.666...

if 5/(-3) == -2
- 5%(-3) = 5 - (-2)(-3) = -1
if 5/(-3) == -1
- 5%(-3) = 5 - (-1)(-3) = 2

Portability (cont)

• Check for sign of n%10; handle both cases

```c
static void putneg(int n) {
    int q = n/10, r = n%10;
    if (r > 0) {
        r -= 10;
        q++;
    }
    if (n <= -10)
        putneg(q);
    putchar(\'0123456789\'[-r]);
}
```
An Easier Way
#include <limits.h>
#include <stdio.h>
static void putu(unsigned n) {
    if (n > 10)
        putchar(0123456789[n%10]);
}
void putd(int n) {
    if (n == INT_MIN) {
        putchar('-');
        putu((unsigned)INT_MAX+1);
    } else if (n < 0) {
        putchar('-');
        putu(-n);
    } else
        putu(n);
}

Increment/Decrement
* Prefix ops increment before returning value
    n = 5;
    x = ++n;
    x is 6, n is 6
* Postfix ops increment after returning value
    n = 5;
    x = n++;
    x is 5, n is 6
* Operands of ++ and -- must be variables

Relational & Logical Ops
* Logical values are int s: 0 is false !0 is true
* Relational ops: > >= < <=
* Equality ops: == !=
* Unary logical negation: !
* Logical connectives: && ||
* Evaluation is left-to-right as far as needed
  && stops when outcome known to be 0
  || stops when outcome known to be 10
**Bit Operations**

- Bitwise logical operations apply to all integers
  - \& \textbf{bitwise AND} \quad 1 \& 1 = 1 \quad 0 \& 1 = 0
  - | \textbf{bitwise inclusive OR} \quad 1 | 0 = 1 \quad 0 | 0 = 0
  - ^ \textbf{bitwise exclusive OR} \quad 1 ^ 1 = 0 \quad 1 ^ 0 = 1
  - ~ \textbf{bitwise complement} \quad ~1 = 0 \quad ~0 = 1

- The | operation is used to “turn on” bits
  ```
  #define BIT0 0x1
  #define BIT1 0x2
  #define BITS (BIT0 | BIT1)
  flags = flags | BITS;
  ```

- The & op is used to “mask off” bits
  ```
  test = flags & BITS;
  ```

**Bit Operations (cont)**

- Assuming 16-bit quantities
  ```
  BIT0 = 0000000000000001
  BIT1 = 0000000000000010
  BITS = 0000000000000011
  flags = 0100011100000001
  flags | BITS = 0100011100000011
  flags & BITS = 0000000000000001
  ```

**Shifting**

- Shift operators: \texttt{<<} \texttt{>>}
  - \texttt{<<} shifts \texttt{x} \texttt{y} bit positions
  - \texttt{>>} shifts \texttt{x} \texttt{y} right bit positions

- When shifting right:
  - if \texttt{x} is signed, may be \texttt{logical} or \texttt{arithmetic}
  - if \texttt{x} is unsigned, shift is always logical
  - arithmetic shift fills with sign bit
  - logical shift fills with \texttt{0}

- When shifting left: always fill with \texttt{0}
Shifting (cont)

• Assuming 16-bit quantities
  
  \[
  \text{bits} = \begin{array}{c}
  11000111000001 \\
  \text{bits} \ll 2 = 00011100000100 \\
  \text{bits} \gg 2 = 111100011100000 \\
  \text{bits} \gg 2 = 001100011100000
  \end{array}
  \]

  (arithmetic)

  (logical)

• Which do you get?
  implementor’s choice (i.e., not portable)

Assignment

• Assignment is an operator, not a statement
  
  \[
  \text{c} = \text{getchar}(); \\
  \text{if} (\text{c} == \text{EOF}) . . .
  \]

  can be written as

  \[
  \text{if} ((\text{c} = \text{getchar}()) == \text{EOF}) . . .
  \]

• Watch out for typos
  
  \[
  \text{if} (\text{c} = \text{EOF}) . . .
  \]

• Combine assignment with other operators
  
  \[
  \text{i} = \text{i} + 2; \quad \text{is the same as} \quad \text{i} += 2; \\
  \text{f} = \text{f} \mid \text{BITS} \quad \text{is the same as} \quad \text{f} |= \text{BITS};
  \]