

Error handling:

1) Errno

2) Perror & Strerror

→ Print an error as a human-readable string

6/28

Character Encoding ASCII:

// GCC for hw assign

• 8 bit = byte

• ASCII is 7 bits value 0 to 127.

• 128 - 255 are not ASCII

← don't try to print these!

• 0 to 31 Non-Printable Control Characters

• PRINTABLE CHARS 32 to 127

• All digits GROUPED TOGETHER

• Encoding digits < Uppercase < Lowercase

* Digit char - '0' = Arithmetic value of digit char

• Control chars = char constants in C

CHEX

0x00 - 00 '0' NUL

0x07 - 07 'a' BELL

0x08 - 08 'b' Backspace

0x09 - 09 't' Tab

0x0A - 0A 'n' NewLine (00001010) BITS FOR NEWLINE

0x0B - 0B 'v' Vertical Tab

0x0C - 0C 'f' Form feed

- 0D 'r' Carriage Return ... Start from most left again

- 1B 'e' Escape

• Some math

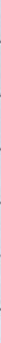

- Binary base 2

- Base 10

→ Power of 2

→ Available digits: 0 ~ 9

→ Available digits 0, 1



DEC	Binary	HEX
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

4 bits = 1 HEX Digit

HEX DEC
 ☆ 0x39 ! = 39
 ↳ How C communicates for HEX number.

- $\textcircled{22}$
- 24
- $\textcircled{16+8+2+1}$
- $\rightarrow \cancel{00011011}$
- $$\begin{array}{r} 3 \\ 8 \overline{) 24} \\ \underline{24} \\ 0 \end{array}$$

VAL = 033; ← octal 33

- 32 bit number

32 bit number

octal	0	6	1	1	4	4	3	1	4	6	4
31	32	33	34	0011	0001	0011	0010	0011	0011	0011	0100
				HEX	3	1	3	2	3	3	4

- Fixed Point Integer Representation

Positive numbers 32 bit HEX

00000000

1 00000001

2 00000002

*Need to represent positive, negative, and zero values.

option 2. Signed magnitude

- Integer has two parts

- Sign bit (0 = positive, 1 = negative)

- Magnitude 31 bits, sign bit

e) $56_{10} = 38_{10} = \overset{\swarrow}{0011\ 1000} \rightarrow 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011\ 1000$

$$-56_{10} = 38_{16} = 10111000$$

① = 00₁₆ = 0000 0000 Binary

- ① = $80_{16} = 1000\ 0000$ Binary

Problem: two Zeros (+0 & -0)

- Complicated hardware

Option #2. One's Complement $(-127 \dots +127)$

- Positive numbers represented as before

- Negative numbers are \oplus numbers with all bits flipped

ex) $56_{10} = 38_{16} = 00111000$
 \uparrow flipped bits

$$-56_{10} = C7_{16} = 11000111$$

0 = 00 u = 0000 0000

$$-0 = FF_{16} = 1111\ 1111$$

Problem: Same problem as option #1

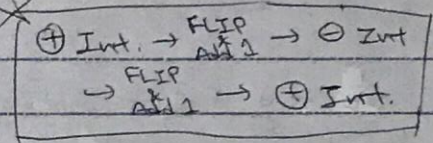
Option #3: Two's Complement (-128... +127)

- \oplus Int. as before

- \ominus Int. take \oplus Int. \rightarrow Flip bits \rightarrow Add 1
 \rightarrow Discard carry

ex) $56_{16} = 38_{16} = 00111000$

$$-56_{10} = C8_{16} = 11001000 \text{ Add } 1$$



$0 = 00000000$
 11111111 FLIP BITS
 10000000
 ← Discard carry

* 1 bit 0, 1 \Rightarrow can represent 2^n distinct values in n bits
 2 bits 00, 01, 10, 11

• $-128_{10} = 10000000$
 01111111 FLIP * -128_{10} does not change!
 $-128_{10} = 10000000 \text{ Add 1}$



LEFT & RIGHT Shifting

Only for fixed point Int. (ex) long, char)

Left Shift

• Operands are integer value, number of bits

ex) Before

00|00|00|01

After

00|00|00|02

Left shift 1 bit

Before

00|00|00|01

After

00|00|00|04

Left shift 2 bits

high order bits

low order bits

ex) Before: 100000000000 000000000000 000000000000 0001

After: * 000000000000 000000000000 000000000000 0010 Left shift 1 bit

High order bits

Low order bits

In C,

int A, B;

A = 1;

B = A << 1;

B <<= 1;

Integer of bits, can be a variable or expression

left shift operator

- can shift char, short, int, long
- Does NOT work for float, double

ex) '3' << 2

'3' = 0x33

00|1100|1100
← Discard

ex) 1 = 0000 0001
 FFFF FF FE ^{FLIP} Look at HEX Table
 -1 = FFFF FFFF ^{Add 1}

★ Right Shift:

- Operands are int. value, number of bits
- can shift char, short, int, long
- Does not work for float, double

Two types
 Logical: Low Order bits are lost, 0 bits added at high order bits
 Arithmetic: Low order bits are lost, high order sign bit is added at high order bits.

Logical right shift: 0 → → Discarded
 Arithmetic " " : ^{sign bit} → Discarded

★ Very Important!

// shifts every thing & replace sign bit with zero.

ex) Logical Right Shift

Before: 01 | 01 | B6 | A2
 After: 00 | 10 | 1B | 6A Logical right shift 4 bits

1011000
 011011000
 2 11011000
 3 111011000
 F

// preserves the sign bit & shift every thing right

ex) Arithmetic Right Shift

Before: B4 | A2 | C3 | E1
 After: FB | 4A | 2C | 3E ARS 4 bits

07 | B4 | C2 | E3 00 | 7B | 4C | 2E ARS 4 bits

4 111110111000
 F B 4

C Programming Structure:

★

FFFFFFFF Smallest negative
 8000 0000 Largest Negative
 7FFFFFFF Largest positive
 1
 0 zero
 FFFFFFFF Smallest negative

⇒ Overflow

Source file

Header file

main.c

preprocessor directives

```
#include "factorial.h" // for factorial()
#include <stdio.h> // for printf()
```

C = compiled language

```
int
main()
{
    int x, y;
    x = 3;
    y = factorial(x);
    printf("The factorial of %d is %d\n", x, y);
    return 0;
}
```

Factorial.h

Multiple Inclusion Protection

```
#ifndef Factorial_h
#define Factorial_h

// Declare all types HERE!
// ex) Enum, struct, etc...

int
Factorial(int x);

#endif
```

function declaration

Factorial.c

functions are global! (unless static)

function definition

```
#include Factorial.h
int
Factorial(int x)
{
    if (x <= 0) return 1;
    else return x * factorial(x-1);
}
```

- * Cannot define function more than once
- * Can declare header files at multiple source files

on iLAB Linux machine

```
gcc main.c Factorial.c -o A.out ← executable program
gcc -o factorial main.c Factorial.c -o factorial
gcc -Wall -o factorial main.c Factorial.c
```

Warn all
↑
show what the comp is trying to do