The image part with relationship ID rId2 was not found in the file.

Department of Computer Science

Computer Data Representation (1):

Lecture #2

Prof. Soraya Abad-Mota, PhD

The image part with relationship ID rId2 was not found in the file.

About the index cards

- Please put your Full name and the date at the top of the card (this saves me time finding the name to record it in the excel file for participation).
- I return them in order by last name (under green cards)
- Place the new one in the pile (under blue cards) containing the first letter of your LAST NAME
- What follows are from the index cards
  - Additional skills for successful programming
  - Good practices
  - Knowledge

The image part with relationship ID rId13 was not found in the file.

Additional skills

- Take breaks related to time management
- Analytical and mathematical skills, stats and probability
- Algorithm development (how?)
- Some were too general: problem-solving, algorithm knowledge?, quick thinking, research
- Reading (understanding) code
- Skills for team work
- Reading documentation
- Debugging (what about it?)
- Asking for help
- Focus, determination

The image part with relationship ID rId13 was not found in the file.

From the index cards

Good practice (not really skills, but using good practices)

- Naming conventions
- Outlining objectives (requirements?)
- Modularity
- Making your code readable
- Making a habit of applying good practices

Knowledge (not skills)

- Programming languages
- Specific tools

The image part with relationship ID rId13 was not found in the file.

To attain the Learning Objectives (of each lecture and the course)

- In the lectures we cover some operations/concepts, you must complete the details by reading the textbook (we indicate which sections by lecture and week on canvas).
- Practice with the book exercises as you read along.
- Many details will be more clear when you practice, something still unclear?
  1. Submit Muddy points at end of lecture
  2. Attend office hours

The image part with relationship ID rId13 was not found in the file.

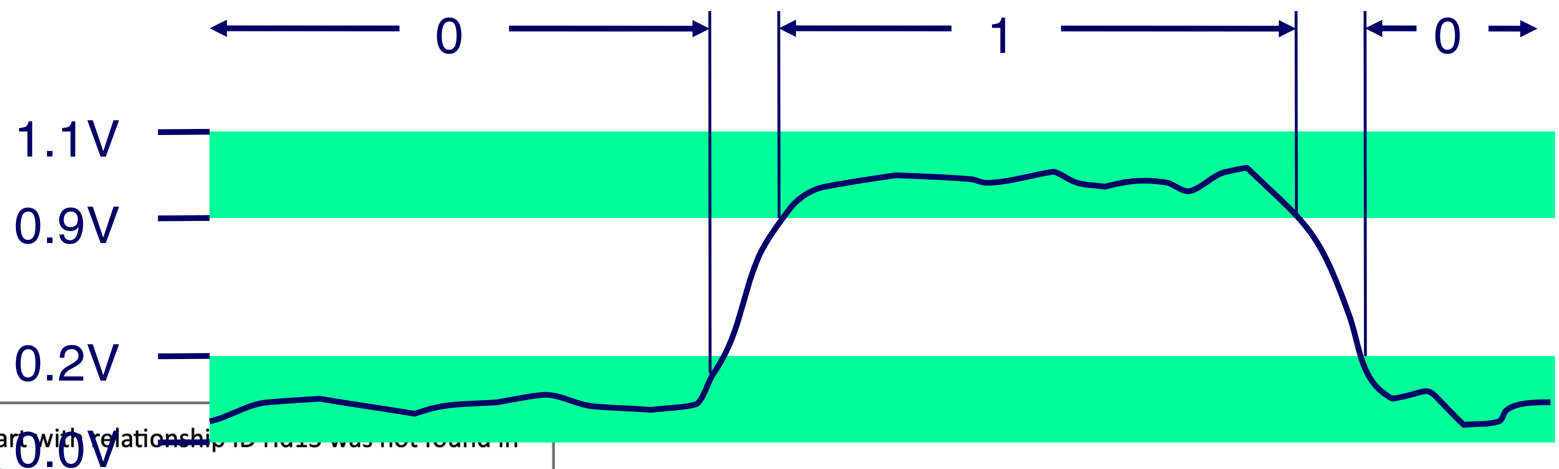
Learning objectives

After this module students should be able to:

1. Define the notions of bits and bytes.
2. Explain why and how computers use bits to represent information.
3. Describe the integral types.
4. Define and perform the following set of operations in C: bit-level operations, logic operations, shift and mask operations.

Everything is bits

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways
  - Computers determine what to do (instructions)
  - … and represent and manipulate numbers, sets, strings, etc…
- Why bits? Electronic Implementation
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires



The image part with relationship ID rId13 was not found in the file.

For example, can count in binary

- Base 2 Number Representation
  - Represent 15213_{10} as 11101101101101_2 (verify that you know)
  - Represent 1.20_{10} as $1.0011001100110011[0011]..._2$
  - Represent 1.5213×10^4 as $1.1101101101101_2 \times 2^{13}$
- We learn how are numbers represented in the computer:
  - Begin with integer representations (integral types),
  - then study floating point (FP)

The image part with relationship ID rId13 was not found in the file.

Encoding Byte Values

- Byte = 8 bits
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}
    - Base 16 number representation
    - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
    - Write $FA1D37B_{16}$ in C as
      - 0xFA1D37B or
      - 0xfa1d37b

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

The image part with relationship ID rId13 was not found in the file.

Bits, Bytes, and Integers

- Representing information as bits
- Bit-level manipulations
- Integers
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary
- Representations in memory, pointers, strings

The image part with relationship ID rId13 was not found in the file.

Practice conversion

- Convert 0x39A7F8 to binary:

- Convert 1100100101111011 to hexadecimal

The image part with relationship ID rId13 was not found in the file.

Practice conversion

- Convert 0x39A7F8 to binary:

3 9 A 7 F 8

0011 1001 1010 0111 1111 1000

each hexadecimal (basic) value is converted into binary using 4 bits

- Convert 1100100101111011 to hexadecimal

1100 1001 0111 1011

C 9 7 B

every four bits are translated into one hexadecimal value (“digit”)

The image part with relationship ID rId13 was not found in the file.

Boolean Algebra

- Developed by George Boole in 19th Century
  - Algebraic representation of logic
    - Encode "True" as 1 and "False" as 0

And

- A&B = 1 when both A=1 and B=1

&	0	1
0	0	0
1	0	1

Or

- A|B = 1 when either A=1 or B=1

|	0	1
0	0	1
1	1	1

Not

- ~A = 1 when A=0

~	
0	1
1	0

Exclusive-Or (Xor)

- A^B = 1 when either A=1 or B=1, but not both

^	0	1
0	0	1
1	1	0

General Boolean Algebras

- Operate on Bit Vectors
  - Operations applied bitwise

```
  01101001     01101001     01101001
& 01010101   | 01010101   ^ 01010101   ~ 01010101
  01000001     01111101     00111100     10101010
```

- All of the Properties of Boolean Algebra apply (p. 52)

The image part with relationship ID rId13 was not found in the file.

Boolean Algebra

- Boolean operations |, &, and ~ operating on bit vectors of length w form a *Boolean Algebra for integer w > 0*
- Boolean Algebra has many properties of arithmetic over integers.
  - E.g. op & distributes over |
  - a & (b | c) = (a & b) | (a & c)
  - Just as multiplication distributes over addition
- Boolean | distributes over & but in integer arithmetic, addition does not distribute over multiplication, i.e. a + (b * c) is not = to (a + b) * (a + c)

The image part with relationship ID rId13 was not found in the file.

Boolean Ring

- Operations ^, &, and ~ on bit vectors of length w form a *Boolean Ring*
- Many properties of Boolean rings also hold on integer arithmetic
- In integer arithmetic, every integer x has an additive inverse; the equivalent to addition in Boolean is ^
- Some interesting results arise from this.

The image part with relationship ID rId13 was not found in the file.

Why are we interested in these mathematical forms?

The image part with relationship ID rId13 was not found in the file.

Learning objectives

After this session students should be able to:

1. Define the notions of bits and bytes.
2. Explain why and how computers use bits to represent information.
3. Describe the integral types
4. Define and perform the following set of operations in C: bit-level operations, logic operations, shift and mask operations.

The image part with relationship ID rId13 was not found in the file.

Integral Types (booleans, characters, Integers, etc.)

- integral means "*of or denoted by an integer*"
- Simplest machine data types are ones that represent integral types
- You should already be familiar with the basics of this, including in C
  - Characters are just 8-bit integers
  - Used for booleans (0 = false, non-zero = true, implementation)
  - Basic boolean logic – AND, OR, XOR, NOT, shifts and masks
- These can be used for interesting things in C
- In general, in assignments (Datalab, for example) you may not use high-level logical operations.

The image part with relationship ID rId13 was not found in the file.

Learning objectives

After this session students should be able to:

1. Define the notions of bits and bytes.
2. Explain why and how computers use bits to represent information.
3. Describe the integral types
4. Define and perform the following set of operations in C: bit-level operations, logic operations, shift and mask operations.

The image part with relationship ID rId13 was not found in the file.

Example: Representing & Manipulating (finite) sets

- Representation
  - bit vector of length w, used to represent subsets of {0, …, w–1}
  - $a_j = 1$ if $j \in A$
    - 01101001 { 0, 3, 5, 6 }
    - *76543210*
    - 01010101 { 0, 2, 4, 6 }
    - *76543210*
- set operations performed as logical operations over bits
  - & Intersection 01000001 { 0, 6 }
  - | Union 01111101 { 0, 2, 3, 4, 5, 6 }
  - ^ exclusive or 00111100 { 2, 3, 4, 5 }
  - ~ Complement(of 2nd) 10101010 { 1, 3, 5, 7 }

The image part with relationship ID rId13 was not found in the file.

Bit-Level Operations in C

(applicable to integral types)

- Operations &, |, ~, ^ available in C
  - Apply to any "integral" data type
    - `long, int, short, char, unsigned`
  - View arguments as bit vectors
  - Arguments applied bit-wise
- Examples (Char data type = 8 bits)
  - ~ 0x41 → 0xBE
    - ~ 01000001_2 -> 10111110_2
  - ~ 0x00 -> 0xFF
    - ~00000000_2 -> 11111111_2
  - 0x69 & 0x55 -> 0x41
    - 01101001_2 & 01010101_2 -> 01000001_2
  - 0x69 | 0x55 -> 0x7D
    - 01101001_2 | 01010101_2 -> 01111101_2

Steps:

1. convert to binary
2. operate bit-by-bit
3. convert to hex

The image part with relationship ID rId13 was not found in the file.

Contrast: Logic Operations in C

- Contrast to Logical Operators (Sec. 2.1.8)
  - `&&, ||, !`
    - View 0 as “False”
    - Anything nonzero as “True”
    - Always return 0 or 1
    - Early termination (see last e.g. below)
- Examples (char data type)
  - `!0x41 --> 0x00 (false)` `NOT(True) -> False`
  - `!0x00 --> 0x01 (true)`
  - `!!0x41 --> 0x01 (true)`

  - `0x69 && 0x55 --> 0x01`
  - `0x69 || 0x55 --> 0x01`
  - `p && *p` (avoids null pointer access)

Contrast: Logic Operations in C

- Contrast to Logical Operators
  - `&&`, `||`, ...
    - View 0 as ...

Watch out for && vs. & (and || vs. |)...
One of the more common oopsies in C programming

- Ex...
  - `!`...
  - `!`...
  - `!!`...
  - `0x69 && 0x55 → 0x01`
  - `0x69 || 0x55 → 0x01`
  - `p && *p` (avoids null pointer access)

The image part with relationship ID rId3 was not found in the file.

Shift Operations (Sec. 2.1.9)

- Left Shift: `x << y`
  - Shift bit-vector **x** left **y** positions
    - Throw away extra (y) bits on left
    - Fill with 0's on right
- Right Shift: `x >> y`
  - Shift bit-vector **x** right **y** positions
    - Throw away extra bits on right
  - Logical shift
    - Fill with 0's on left
  - Arithmetic shift
    - Replicate most significant bit on left
- Undefined Behavior
  - Shift amount < 0 or ≥ word size (read aside on p. 59)

Argument **x**	01100010
<< 3	00010*000*
Log. >> 2	*00*011000
Arith. >> 2	*00*011000

Argument **x**	10100010
<< 3	00010*000*
Log. >> 2	*00*101000
Arith. >> 2	*11*101000

The image part with relationship ID rId13 was not found in the file.

Practice Problem 2.16 p.58

- For x in hex
  - convert to binary first, then perform the shift
  - `x << 3 (provide result in binary and hex)`
  - `x >> 2 (logical)`
  - `x >> 2 (arithmetic)`
- `x = 0xC3, 0x75, 0x87, 0x66`

The image part with relationship ID rId13 was not found in the file.

Muddy points (from past courses)

1. Ox is the prefix that is attached to a number in C, when you want to indicate the number is a hexadecimal value.
2. A vector of bits is simply a sequence of bits of some length n.
3. Remember that 1 byte = 8 bits

The image part with relationship ID rId13 was not found in the file.