COSC160: Data Structures

Jeremy Bolton, PhD
Assistant Teaching Professor

GEORGETOWN UNIVERSITY

Outline

I. What is Data?
   I. Data representations on a computer

II. Basic Structures
   I. Goals of Structures
   II. Applications

Review Topics (next lecture)

- Time Complexity
- Recursion and Recurrences
- Memory Allocation and Management
  - Pointers and Chaining
  - Stack vs Heap
  - Garbage Collection
  - Deep vs Shallow Copy
- Coding Practices:
  - Design and Modeling
  - Debugging
  - OOP, Templates, …

What is Data?

- A peak under the hood …
  - A good understanding of how data is represented in a computer will make you a better computer scientist
    - Many algorithms (and data structures) depend on (and take advantage of) the representation and structure of data
      - Data Representation, EG: Fast Multiplication (by two)
      - Data Organization, EG: Heap Sort

Binary Representations

- Data is encoded into computers as binary string of various lengths
  - Each bit (latch in hardware) can store a 0 or 1
  - A byte is an 8 bit sequence
  - Each unique binary string can be used to represent a different integer.
  - However, binary strings can be used to represent various data.

- EG: binary strings as integers (polynomial expansion)

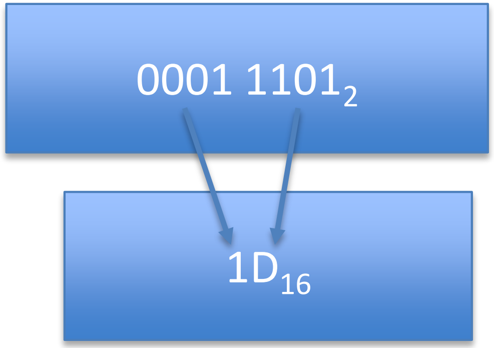
  $0001\ 1101_2$

  $2^7(0)+ 2^6(0)+ 2^5(0)+ 2^4(1) + 2^3(1)+ 2^2(1)+ 2^1(0)+ 2^0(1) = 29$

- Hexi-decimal Representation

  $16^1(1)+ 16^1(13) = 29$

  $1D_{16}$



Data Types

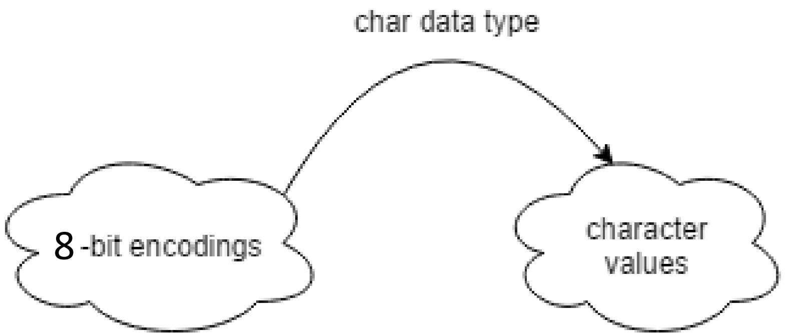
- Binary strings interpreted differently based on data type
  - Data type can be seen as a set of all possible values of some category; or more generally a data type is a mapping from set of all valid binary strings (of some length) to a value.

- Example: chars
  - Assumes 8-bit

EG:

$0100\ 0100 \rightarrow 'D'$

$0000\ 1010 \rightarrow '\backslash n'$



Integer Representation

- unsigned ints
  - Unsigned ints are generally represented using the standard polynomial expansion of binary sequences.

- ints are generally stored in “2’s complement” which allows for the representation of negative integers and allows for an efficient implementation of arithmetic.
  - Example (assuming 4-byte representation)
  - Conversion to two’s complement of negative int
    1. List binary representation of positive value.
    2. Invert binary digits
    3. Add 1

Example: Twos Complement

- Steps
  1. List binary representation of positive value.
  2. Invert binary digits
  3. Add 1

- What is two's complement representation of -67?

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100\ 0011, 0x0043$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011\ 1100, 0xFFBC$$
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011\ 1101, 0xFFBD$$

Memory requirements of common data types

- Each data type may have different encoding schemes, as noted previously.
- Different data types may also have notably different lengths, or memory requirements.

Type Name	Bytes	Other Names	Range of Values
int	4	signed	–2,147,483,648 to 2,147,483,647
unsigned int	4	unsigned	0 to 4,294,967,295
short	2	short int, signed short int	–32,768 to 32,767
unsigned short	2	unsigned short int	0 to 65,535
long	4	long int, signed long int	–2,147,483,648 to 2,147,483,647
unsigned long	4	unsigned long int	0 to 4,294,967,295
long long	8	none (but equivalent to __int64)	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
short	2	short int, signed short int	–32,768 to 32,767
bool	1	none	false or true
char	1	none	–128 to 127 by default 0 to 255 when compiled by using

Low-Level Operations on Data

- Binary Operations
  - Bit-wise Logical OR: |
  - Bit-wise Logical AND: &
  - Bit-wise Logical NOT: ~
  - Bit-wise Logical XOR: ^
  - Bit-wise shift left: <<
  - Bit-wise shift right: >>

```
      0110 0001 1101 0100
|     1011 1000 1110 0100
      1111  1001 1111 0100

      0110 0001 1101 0100
&     1011 1000 1110 0100
      0010 0000 1100 0100

    0011 1000 1110 0100 << 2
      1110  0011 1001 0000
```

Confirm the following:

5 | 3 = 7

4 & 3 = 0

3 << 1 = 6

There are also many low-level (binary level), operators available.

GEORGETOWN UNIVERSITY

Why is knowing data representation important?

- You’re a computer scientist!

- Many structures and algorithms are optimized based on the low level representations of data
  - Lower level CPU operations are fast!

- If you can encode data using minimal data encodings, you can reduce memory requirements and *make algorithms and structures more efficient!*

Example: Memory Requirements

- Assume you are tasked with documenting (on a computer) major milestones in recent history (since 1 AD) by year. How would you store the ‘year’ information?

Example: ("Faster") Multiplication by 2

- In this example, the bitwise alternatives may execute near 2 or 3 times faster.
- The speed of operators is determined by the CPU and underlying architecture.
- ALSO NOTE: compilers are "clever" and may attempt to optimize your code. Thus compiler may make this edit (without explicit notice to you – the programmer) when converting C++ code to machine code. As a result, it may be the case that there is no difference in execution time.

- Equivalent code snippets:

```
x = x * 2;
x = x << 1;

x = x * 8;
x = x << 3;
```

Summary of Data

- Data is encoded as binary sequences in computers
- Data types are mappings from a set of binary sequences to a set of values
- Knowing more about data representations and operators available will allow for more efficient structures and algorithms for data.

Data Structures

- Why Structures for Data?
  - Simple: it is practical and efficient
    - Keeps Data Organized
      - Filing cabinet analogy
    - Can provide for efficient storage, retrieval, and manipulation (or analysis) of data
      - How do we measure “efficient”?
      - Time complexity analysis – more to come!
    - Aligns with OOP paradigm: promotes reusability

Applications of Basic Data Structures

- Clear Applications
  - Data: storage and retrieval

- Many Other Applications
  - Calculations.
    - E.G. Polynomial evaluation
  - Help to facilitate efficient algorithms
    - Sorting
    - Traversals

Goals of Data Structures

- Computer science is the practice of problem solving.
  - Data structures are tools used to help solve a problem or accomplish a task.
  - Good Solutions (Algorithms):
    - Good solutions are correct.
    - Good solutions are practical.
    - Good solutions are efficient.
      - Efficient in time
      - Efficient in place

- Low level representations
  - Understanding lower level details related to memory management will make you a better programmer (of ***efficient*** structures and algorithms)