

CSI333 Lecture 1

- ▶ A CLASS is a datatype.
A CLASS is a master pattern (like a rubber stamp) for creating one or more OBJECTS, together with code and interface definitions for OPERATIONS (called METHODS) on those objects.
- ▶ An OBJECT or CLASS INSTANCE is some ACTUAL DATA plus a way to run its methods on that data.

Object oriented programming and software design: Most or all pieces of data are CLOSELY ASSOCIATED WITH operations on that data.

But, within computer hardware, that data and those operations are quite different.

Abstraction—One recurring theme from PP Chap. 1

I propose this definition: The act or the result of judiciously ignoring details by inventing and using a concept and a name instead of the well-defined body of details the abstraction replaces.

Examples:

1. Each name (and signature) of a method is an abstraction for the code you wrote in the method body.
2. A class is often an abstraction for some entities in the real world, like dates, shapes, personnel records, etc.
3. An abstract data type (like a STACK) is an abstraction for multiple choices of ways to implement it (like by a linked list versus an array).
4. Read the discussion and examples in PP Chap. 1.

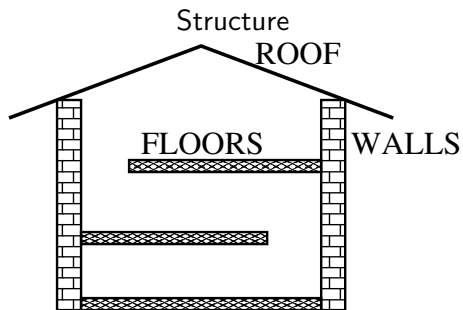
Programming

- ▶ Programming is basically translating an ALGORITHM into a practical PROGRAM to make a computer carry out the algorithm.
- ▶ Algorithm usually means a fairly short composition of directions, where each step is simple to carry out, that is presented to solve a usually simple and well-defined problem.
- ▶ “Algorithm” also indicates that the directions are guaranteed (by whom?) to terminate no matter what is the input. (Otherwise, the directions should be called a “computational procedure”.)
- ▶ Advanced programming and software development goes beyond this because what big programs and systems do is too complicated and often faulty and ill-defined to be expressed by a single problem to be solved by an algorithm.

Course Coverage

1. Machine/Assembly Language: Instruction Set Architecture is instructions & datatypes of computer **hardware**. C related to machine features.
2. C/C++ hardware/software/system interface topics: type implementation, modularization with multiple source file programs, functions, pointers, strings, arguments, preprocessor, etc.
3. Larger, more mature projects and software designs.

House Architecture One

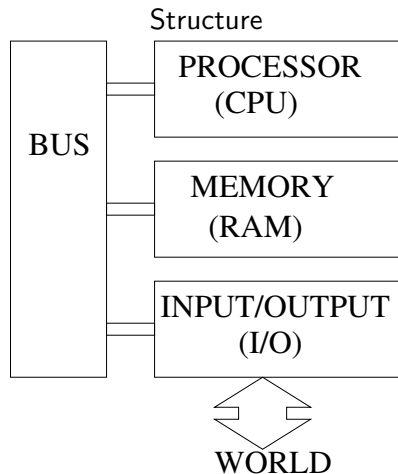


Function

Provide sheltered living spaces for people.

Let's see what we mean by architecture. Here's a very much oversimplified idea of architecture of buildings. Architects talk about form and function.

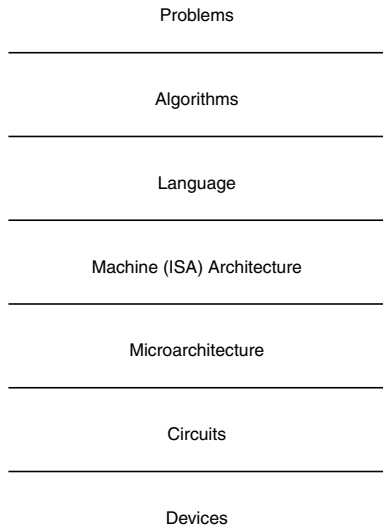
Computer Architecture One



Function

Communicate, store and process **digitally represented** data under **digitally stored** program control

Computer System Architecture—Layered View



Progression from Problem to Program: Expressions

An *expression* expresses a calculation on numbers or other kinds of values (like logic or *Boolean* values). People learn:

$$39.99 + 119.95 * (100\% - 50\%) - 5.00 + 950.00 * (100\% - 10\%)$$

Do the innermost *subexpressions* first:

$$39.99 + 119.95 * 50\% - 5.00 + 950.00 * 90\%$$

$$39.99 + 119.95 * (100\% - 50\%) - 5.00 + 950.00 * (100\% - 10\%)$$

$$39.99 + 119.95 * 50\% - 5.00 + 950.00 * 90\%$$

Now the 2 multiplications are innermost, because, when add and multiply compete, multiply wins:

$$39.99 + 59.975 - 5.00 + 855.00$$

(we say “* has greater *precedence* than +”)

The left addition can be done first:

$$99.965 - 5.00 + 855.00$$

The above means subtract 5.00 from 99.965 then add 855.00.

$$94.965 + 855.00$$

Finish:

$$954.965$$

What is the Computer Science Problem?

- ▶ Computers ain't that smart.
- ▶ Is it a good idea to attempt an algorithm the emulates *exactly* what a person does?
- ▶ What algorithm should be programmed into an electronic *calculator*?
- ▶ Can your algorithm *scale up* to handle large problem instances efficiently?
- ▶ Can ideas about expressions and algorithms to process them be applied to other situations? **YES!** Most programming languages, and the famous SGML/HTML/XML family of languages have the *Nested Expression* kind of syntax.

The Calculator Problem

Let's start with SIMPLE CASES.

3

The Calculator Problem

Let's start with SIMPLE CASES.

3

3

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

3

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

3 That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one:

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one: 3

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one: $3 +$

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one: $3 +$

The Value box stores the previously inputted values while new values and operators are read.

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one: $3 + 4$

The Value box stores the previously inputted values while new values and operators are read. The Operator box stores previously inputted operators while new input is processed.

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one: $3 + 4$

4
3

+

The Value box stores the previously inputted values while new values and operators are read. The Operator box stores previously inputted operators while new input is processed.

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one: $3 + 4 =$

4
3

+

The Value box stores the previously inputted values while new values and operators are read. The Operator box stores previously inputted operators while new input is processed.

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one: $3 + 4 =$

3

4

The Value box stores the previously inputted values while new values and operators are read. The Operator box stores previously inputted operators while new input is processed.

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one: $3 + 4 =$

3 + 4

The Value box stores the previously inputted values while new values and operators are read. The Operator box stores previously inputted operators while new input is processed. Operators and values are REMOVED after they are used.

The Calculator Problem

Let's start with SIMPLE CASES.

$$3 =$$

That's the answer. Easy. Let's try $3 + 4 =$
Key it in, one by one: $3 + 4 =$

3 + 4

The Value box stores the previously inputted values while new values and operators are read. The Operator box stores previously inputted operators while new input is processed. Operators and values are REMOVED after they are used.

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

39.99

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

39.99 +

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

$$39.99 + 119.95$$

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

$$39.99 + 119.95 *$$

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

$$39.99 + 119.95 * ($$

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

$$39.99 + 119.95 * (100\%$$

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

$$39.99 + 119.95 * (100\% -$$

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

$$39.99 + 119.95 * (100\% - 50\%)$$

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

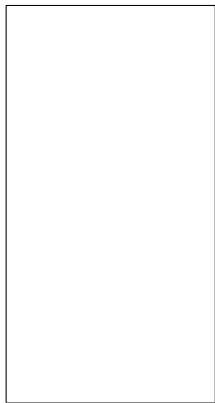
$$39.99 + 119.95 * (100\% - 50\%)$$

The Calculator Problem

Some data is entered *before* the rest of the data. The calculator cannot tell what the user will enter next, but it must remember data already entered.

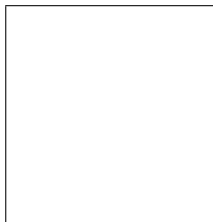
$$39.99 + 119.95 * (100\% - 50\%) - 5.00 + 950.00 * (100\% - 10\%)$$

39.99



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

39.99



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

39.99 +



39.99



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

39.99 +

39.99

+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

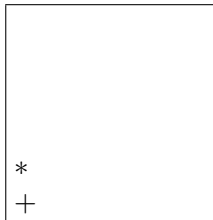
$$39.99 + 119.95$$

39.99

+

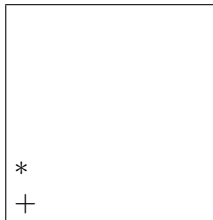
- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 *$$



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * ($$



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\%$$

100%
119.95
39.99

(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% -$$

100%
119.95
39.99

(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% -$$

100%
119.95
39.99

-
(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% - 50\%)$$

100%
119.95
39.99

-
(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% - 50\%)$$

50%
100%
119.95
39.99

-
(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% - 50\% *$$

50%
100%
119.95
39.99

-
(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% - 50\% *$$

50%
100%
119.95
39.99

*
-
(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% - 50\% * 0.2)$$

50%
100%
119.95
39.99

*
-
(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% - 50\% * 0.2)$$

0.2
50%
100%
119.95
39.99

*
-
(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% - 50\% * 0.2)$$

0.2
50%
100%
119.95
39.99

*
-
(
*
+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box;

$$39.99 + 119.95 * (100\% - 50\% * 0.2)$$

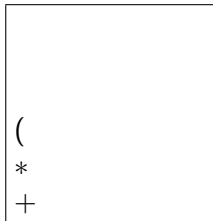
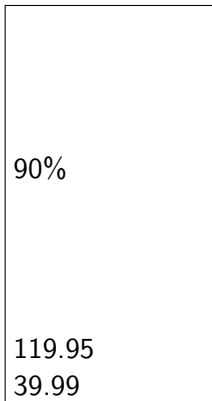
10%

100%
119.95
39.99

-
(
*
+

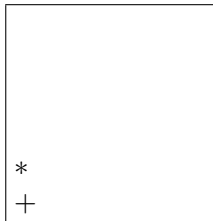
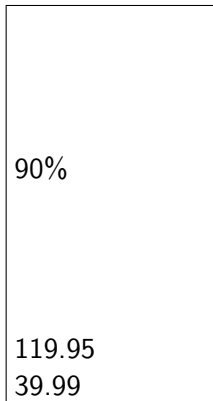
- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result)

$$39.99 + 119.95 * (100\% - 50\% * 0.2)$$



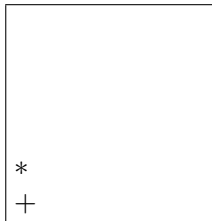
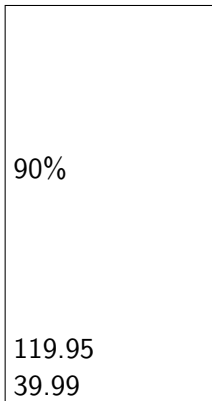
- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched "("

$$39.99 + 119.95 * (100\% - 50\% * 0.2)$$



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched "("

$$39.99 + 119.95 * (100\% - 50\% * 0.2) -$$



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched "("

$$39.99 + 119.95 * (100\% - 50\% * 0.2) -$$

107.95

39.99

+

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched “(”

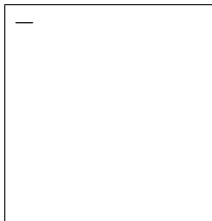
$$39.99 + 119.95 * (100\% - 50\% * 0.2) -$$

130.945

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched “(”

$$39.99 + 119.95 * (100\% - 50\% * 0.2) -$$

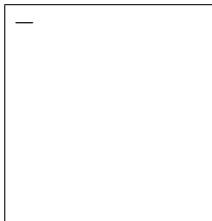
130.945



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched “(”

$$39.99 + 119.95 * (100\% - 50\% * 0.2) - 5.00$$

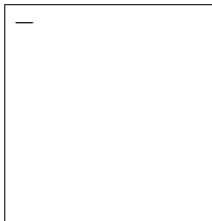
130.945



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched “(”

$$39.99 + 119.95 * (100\% - 50\% * 0.2) - 5.00$$

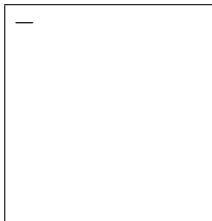
5.0
130.945



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched “(”

$$39.99 + 119.95 * (100\% - 50\% * 0.2) - 5.00\text{end}$$

5.0
130.945



- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched "("

$$39.99 + 119.95 * (100\% - 50\% * 0.2) - 5.00\text{end}$$

131.945

- ▶ Use a Value Box and an Operator Box to keep input (and results) until they are needed.
- ▶ Each step: Choose either (1) PUSH current input into relevant box; OR (2) REDUCE (replace values and operator with their result) OR (3) remove a matched “(”

Project 1

Your job is to understand the algorithm based on these ideas and then implement it (for Boolean, not numeric expressions) by a C++ or Java program. HOMEWORK: practice it on paper.

Observations:

1. The value or operator accessed or removed from the corresponding “box” is ALWAYS the LAST ONE put in the box but not yet removed. Those boxes can be *stacks*, i.e., pushdown or Last-In-First-Out (LIFO) stores.
2. The input is processed left-to-right.
3. The decision to PUSH, or REDUCE, or REMOVE a “(” is based only on the current input token and the top of the Operator stack.
4. When two operators affect the PUSH or REDUCE decision, the decision is based on their relative *precedence* (and associativity).

It is really a CS1-CS2 Review:

1. Implement and use two stacks.
(Stacks recur over and over in this course...how function linkage works is a topic.)
2. Organize the program so the BoolExpression class holds the method to do the evaluation. (Computer hardware is built of gates that do logical, i.e., *Boolean* operations on bits.)

Programming Project 2

- ▶ Start using C on Unix.
- ▶ Input 32-bit words from the user in several different ways.
- ▶ See those bits in two common ways: Hexadecimal and the bits as 0s and 1s.
- ▶ Output (i.e., interpret) the 32-bit words in several different ways.

Loop until commanded to exit:

1. Print a menu and receive an input choice, or command to use currently stored data.
2. (exit, or) Run an input operation according to that choice.
3. Print the input bits.
4. Print a menu and receive an output choice.
5. Run an output operation according to that choice.

Instructions and help material are distributed ... Labs this week will cover UNIX and the needed C elements.

Bit - Binary Digit

Basic Unit of Information stored and manipulated in our computers.

Computer ¹ hardware stores & manipulates & transmits all data **digitally**: which means with on/off, 0/1, **true/false** (usually) electrical² signals.

A bit (binary digit) is a single unit of memory or transmission that can have only 2 possible values.

¹and much other popular electronic product

²also optical and magnetic

A representation is a coding scheme to give meaning to bit strings (sequences).

Different kinds of data are each represented by different meanings we give to sequences of bits.

Some hardware (printers, keyboards e.g.) embodies particular representations: 01000010 makes standard printers print **B**

The base or radix 2 (binary) number system is used by present computers to represent non-negative integers.

Example: 0001 1100 1000 0110

$$\begin{aligned} &0 \cdot 2^{15} + 0 \cdot 2^{14} + 0 \cdot 2^{13} + 1 \cdot 2^{12} \\ &+ 1 \cdot 2^{11} + 1 \cdot 2^{10} + 0 \cdot 2^9 + 0 \cdot 2^8 \\ &+ 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 \\ &+ 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \end{aligned}$$

$$\begin{aligned} &= (0+0+0+4096)+(2048+1024+0+0)+(256+0+0+0)+(0+4+2+0) \\ &= 7302 \end{aligned}$$

Computer Experts all know their powers of 2:

2^1	2	2^{11}	2048 = 2Ki
2^2	4	2^{12}	4098 = 4Ki
2^3	8	2^{13}	8192 = 8Ki
2^4	16 _{ten} = 0x10	2^{14}	16,384 = 16Ki
2^5	32 _{ten} = 0x20	2^{15}	32,768 = 32Ki
2^6	64 _{ten} = 0x40	2^{16}	65,536 = 64Ki
2^7	128 _{ten} = 0x80	2^8	256
2^9	512	2^{10}	1024 = 1kibi = 1Ki
2^{20}	1,048,576 = 1Mi $\approx 10^6$	2^{30}	1,073,741,824 = 1Gi $\approx 10^9$

“Thus 1024 bytes of storage is officially a kibibyte, not a kilobyte. However, computer professionals generally dislike this unit (they say it sounds like a cat food) so the ambiguity in the size of a kilobyte persists. The prefix is a contraction of “kilobinary.” The symbol Ki-, rather than ki-, was chosen for uniformity with the other binary prefixes (Mi-, Gi-, etc.)” (Russ Rowlett, UNC)

Addition in Decimal

$$\begin{array}{r} 36 \\ 87 \\ \hline \end{array}$$

Addition in Decimal

$$\begin{array}{r} 1 \quad \text{carries} \\ 36 \\ 87 \\ \hline 3 \end{array}$$

Addition in Decimal

$$\begin{array}{r} 11 \text{ carries} \\ 36 \\ 87 \\ \hline 23 \end{array}$$

Addition in Decimal

$$\begin{array}{r} 11 \text{ carries} \\ 36 \\ 87 \\ \hline 123 \end{array}$$

Addition in Binary: Single digits

$$\begin{array}{r} 0 \\ 0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ 1 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ 1 \\ \hline 1 \ 0 = 2 \end{array}$$

$$\begin{array}{r} 0 \\ 0 \\ 0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ 0 \\ 1 \\ \hline 1 \end{array} \quad \begin{array}{r} 0 \\ 1 \\ 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ 0 \\ 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 0 \\ 1 \\ 1 \\ \hline 1 \ 0 \end{array} \quad \begin{array}{r} 1 \\ 0 \\ 1 \\ \hline 1 \ 0 \end{array} \quad \begin{array}{r} 1 \\ 1 \\ 0 \\ \hline 1 \ 0 \end{array}$$

$$\begin{array}{r} 1 \\ 1 \\ 1 \\ \hline 1 \ 1 = 3 \end{array}$$

way!

Mathematics requires a binary computer to work this

Express these rules with LOGIC

0	0	1	1	<i>A</i>
0	1	0	1	<i>B</i>
<hr/>				<hr/>
0 0	0 1	0 1	1 0	<i>Carry</i> <i>Sum</i>

Explain logically how *A* and *B* determine *Sum* and *Carry*?

Express these rules with LOGIC

0	0	1	1	<i>A</i>
0	1	0	1	<i>B</i>
<hr/>				<hr/>
0 0	0 1	0 1	1 0	<i>Carry</i> <i>Sum</i>

Explain logically how *A* and *B* determine *Sum* and *Carry*?

Sum = 1 exactly when (*A* = 1 and *B* = 0) or (*A* = 0 and *B* = 1).

Express these rules with LOGIC

0	0	1	1	<i>A</i>
0	1	0	1	<i>B</i>
<hr/>				<hr/>
0 0	0 1	0 1	1 0	<i>Carry Sum</i>

Explain logically how *A* and *B* determine *Sum* and *Carry*?

Sum = 1 exactly when (*A* = 1 and *B* = 0) or (*A* = 0 and *B* = 1).

Carry = 1 exactly when (*A* = 1 and *B* = 1).

Express these rules with LOGIC

0	0	1	1	<i>A</i>
0	1	0	1	<i>B</i>
<hr/>				<i>Carry</i> <i>Sum</i>
0 0	0 1	0 1	1 0	

Explain logically how *A* and *B* determine *Sum* and *Carry*?

Sum = 1 exactly when (*A* = 1 and *B* = 0) or (*A* = 0 and *B* = 1).

Carry = 1 exactly when (*A* = 1 and *B* = 1).

The electronic *devices* called *logic gates* determine output signals like *Carry* from inputs like *A* and *B*. Typical computation time:

0.1 nanosecond = 10^{-10} second.

3 Basic Gates: AND, OR, and NOT

Truth Table for the AND gate (and AND Boolean Operation):

A	B	$A \text{ AND } B = A \& B$
0	0	0
0	1	0
1	0	0
1	1	1

Other useful gates and Boolean Operations:

X	Y	$X \text{ OR } Y = X Y$
0	0	0
0	1	1
1	0	1
1	1	1

(our OR is *Inclusive*)

W	$\text{NOT } W = \sim W$
0	1
1	0

Some Boolean Expressions

A	B	$\sim B$	$(A \& \sim B)$	$\sim A$	$(\sim A \& B)$	$(A \& \sim B) (\sim A \& B)$
0	0	1	0	1	0	0
0	1	0	0	1	1	1
1	0	1	1	0	0	1
1	1	0	0	0	0	0
Two independent Boolean Variables		Intermediate Values				This is <i>Sum</i> !
						Desired Result

```

#include <stdio.h>
unsigned char N = 0; // 8 bits will be used
for( int i=0; i<1000; i++ )
{
    printf("%d ", N );//Print N as decimal integer.
    N = N + 1;
}

```

overflows or wraps around when N= 255.

```

0 1 ... 253 254 255 0 1 2 3 ... 255 0 ...
252 253 254 255 0 1 2 3 ... 229 230 231

```

Something like this **absolutely must happen** since 8 bits can only distinguish $2^8 = 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 = 256$ different data values.

```
#include <stdio.h>
signed char N = 0; // 8 bits will be used
for( int i=0; i<1000; i++ )
{
    printf("%d ", N );//Print N as decimal integer.
    N = N + 1;
}
```

N declared signed this time

```
0 1 2 3 ... 126 127 -128 -127 -126 -125 ...
-3 -2 -1 0 1 2 ... 126 127 -128 -127 ...
-3 -2 -1 0 1 2 ... 126 127 -128 -127 ...
-3 -2 -1 0 1 2 ... 126 127 -128 -127 ...
... -29 -28 -27 -26 -25
```

The reason for the difference is subtle:

- ▶ Standard C `printf("%d", integer value);`
`%d` format ALWAYS uses SIGNED interpretation.
- ▶ C uses “usual argument conversions” on (8 bit) char types to (16 or 32) bit int since the `printf` function doesn't have declared argument types. `printf` is not type-safe!
- ▶ Bits of unsigned char variables are interpreted in unsigned binary for conversion to int.
- ▶ Bits of signed char variables are interpreted in signed binary for conversion to int.

Method calling

When a (non-static) method is CALLED, it is always called FOR (or ON or WITH) a particular OBJECT.
(That object has type given by the CLASS of the method.)

```
#include "Pile.h"
main()
{
    Pile myPile;
    Pile yourPile;

    myPile.printMe();
    yourPile.makeFullDeck();
    yourPile.printMe();
}
```

```
class Pile
{ ...
  string items[3000];
  int size;
  ...
  void printMe();
  ...
};

void myPile::printMe()
{
  for( int i = 0; i < size; i++ )
  {
    cout << items[i] << endl;
  }
}
```

When the debugger “steps into” a method call, you can view the value of the `this` pointer.

You can view the data in the *object for which the method was called* by DEREFERENCING the `this` pointer; double-click to do this in DDD.