COL106
Data Structures and Algorithms

Subodh Sharma and Rahul Garg

Number Representation

Floating Point Numbers

Bit-Wise Operations in C/C++

unsigned int x, y, z;

x = 0xF0;

y = 0x8D;

z = x & y;

z = x | y;

z = (x >> 2)

z = (y << 1)

z = (x >> 2) | (y << 1)

```c
int main() {
    int x, y, z;
    x = 0xF0;
    y = 0x8D;
    z = x & y;

    printf("Value of x is: %d %X %o\n", x, x, x);
    printf("Value of y is: %d %x %o\n", y, y, y);
    printf("Value of z is: %d %X %o\n", z, z, z);

    z = (x >> 2);
    printf("Value of z is: %d %X %o\n", z, z, z);

    z = (y << 1);
    printf("Value of z is: %d %X %o\n", z, z, z);

    z = (x >> 2) | (y << 1);
    printf("Value of z is: %d %X %o\n", z, z, z);

}
```

```c
int  main() {
    int x, y, z;
    x = 0xF0;
    y = 0x8D;
    z = x & y;

    printf("Value of x is: %d %X %o\n", x, x, x);
    printf("Value of y is: %d %x %o\n", y, y, y);
    printf("Value of z is: %d %X %o\n", z, z, z);

    z = (x >> 2);
    printf("Value of z is: %d %X %o\n", z, z, z);

    z = (y << 1);
    printf("Value of z is: %d %X %o\n", z, z, z);

    z = (x >> 2) | (y << 1);
    printf("Value of z is: %d %X %o\n", z, z, z);

}
```

```
rahulgarg@cerebrum:~/home/COL106/2023M/Lectures$ g++ number_rep.cc; ./a.out
Value of x is: 240 F0 360
Value of y is: 141 8d 215
Value of z is: 128 80 200
Value of z is: 60 3C 74
Value of z is: 282 11A 432
Value of z is: 318 13E 476
rahulgarg@cerebrum:~/home/COL106/2023M/Lectures$
```

Interesting Examples

Example 1

Array of chars

```
if (“A” == “A”)
    cout << “\”A\”” and \”A\” are equal”;
else
    cout << “\”A\”” and \”A\” are not equal”;
```

Example 2

```
if (‘A’ == ‘A’)
    cout << “\’A\’” and \’A\’ are equal”;
else
    cout << “\’A\’” and \’A\’ are not equal”;
```

Shift Operations

- Left shift: **`x << y`**
  - Shift bit-vector `x` left by `y` positio
    - Throw away extra bits on left
    - Fill with 0s on right
  - Multiply by 2**y
- Right shift: **`x >> y`**
  - Shift bit-vector `x` right by `y` positions
    - Throw away extra bits on right
  - Logical shift (for unsigned)
    - Fill with 0s on left
  - Arithmetic shift (for signed)
    - Replicate most significant bit on righ
    - Maintain sign of x
  - Divide by 2**y
  - Correct truncation (towards 0) requires some care with signed numbers

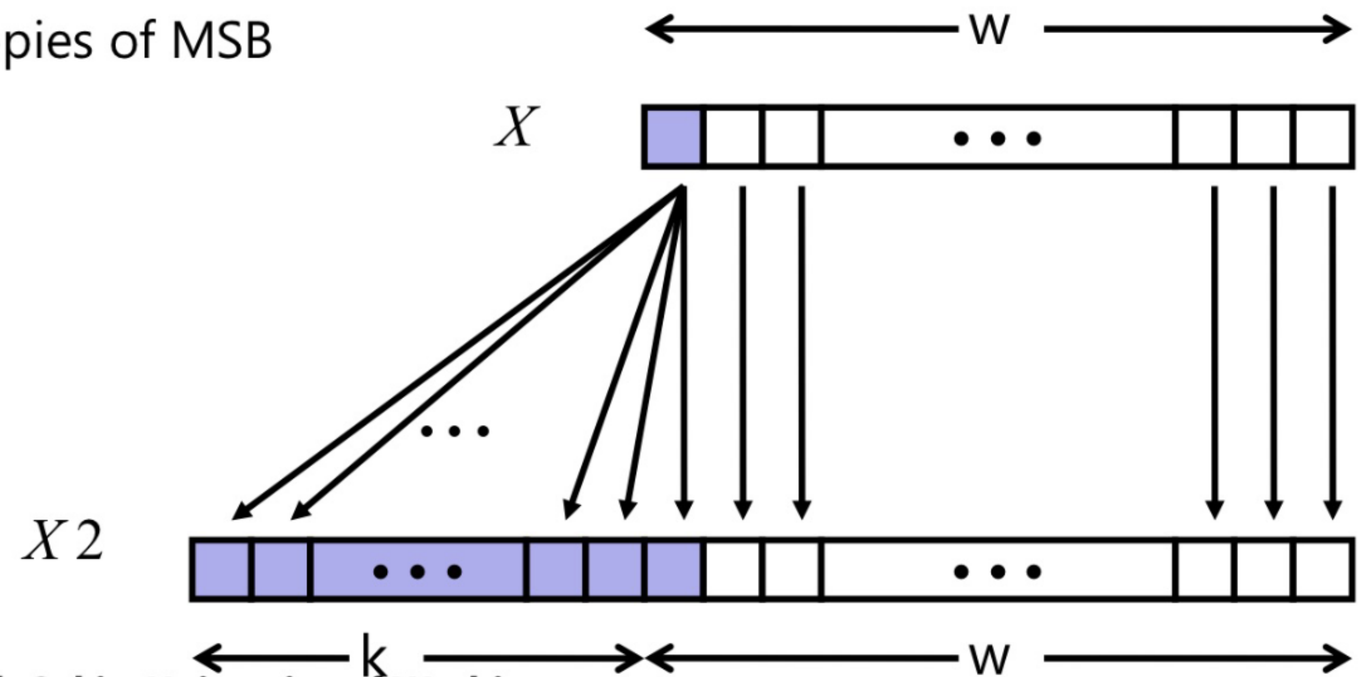
Argument x	01100010
<< 3	00010000
Logical >> 2	00011000
Arithmetic >> 2	00011000

Argument x	10100010
<< 3	00010000
Logical >> 2	00101000
Arithmetic >> 2	11101000

Slide adapted from: Mark Oskin-University of Washington

Sign Extension

- Task:
  - Given w-bit signed integer x
  - Convert it to w+k-bit integer with same value
- Rule:
  - Make k copies of sign bit:
  - $X\,2 = \underbrace{x_{w-1}, \ldots, x_{w-1}}_{\text{k copies of MSB}}, x_{w-1}, x_{w-2}, \ldots, x_0$



Slide adapted from: Mark Oskin-University of Washington

Sign Extension Example

```
short int x =  12345;
int       ix = (int) x;
short int y = -12345;
int       iy = (int) y;
```

	Decimal	Hex	Binary
x	12345	30 39	00110000 01101101
ix	12345	00 00 30 39	00000000 00000000 00110000 01101101
y	-12345	CF C7	11001111 11000111
iy	-12345	FF FF CF C7	11111111 11111111 11001111 11000111

- Converting from smaller to larger integer data type
- C automatically performs sign extension
- You might have to if converting a bizarre data type to a native one (e.g. PMC counters are sometimes 48 bits)

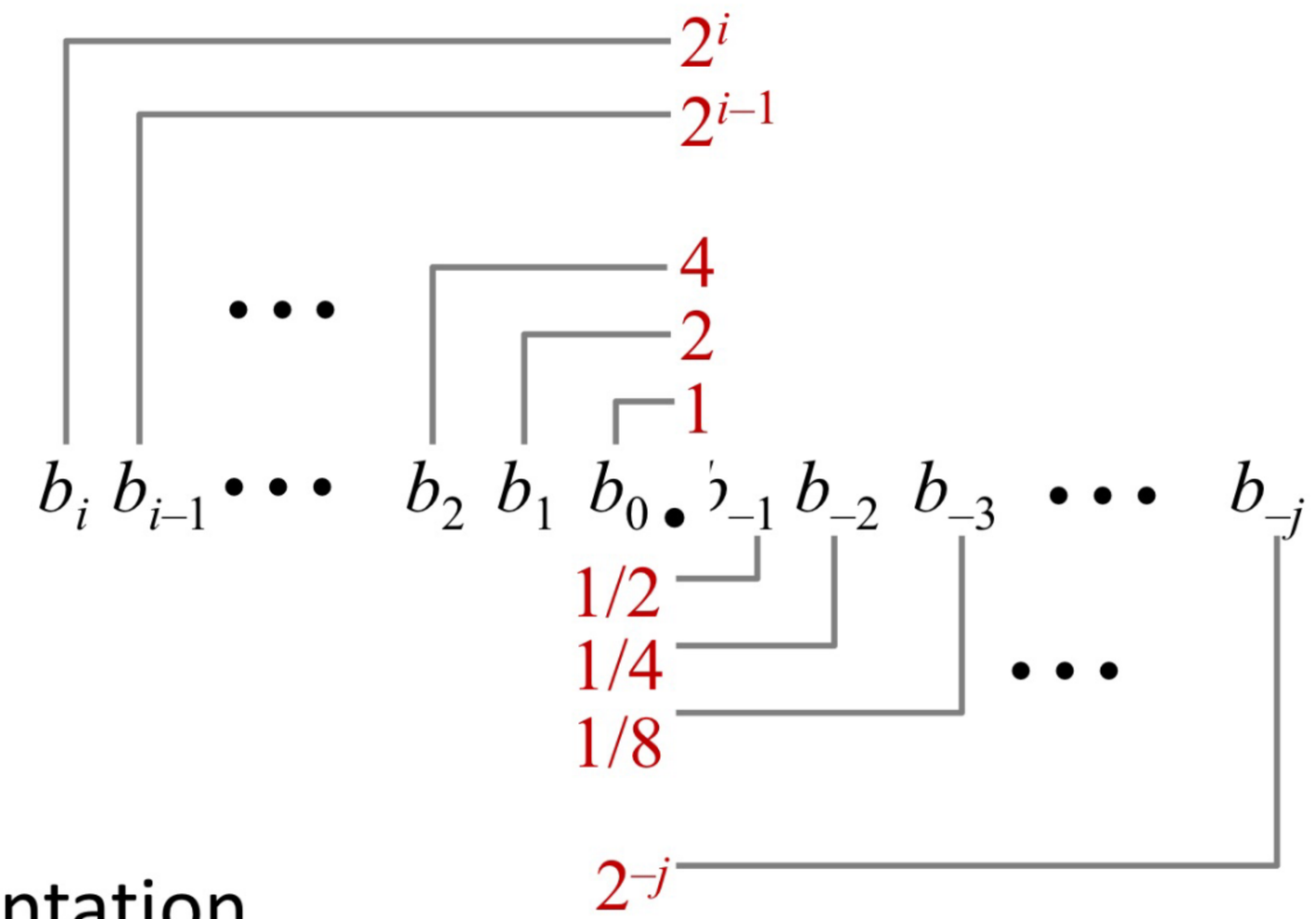
Representing Real Numbers

Fractional binary numbers

- What is 23.982?

- Similarly in base 2, what is 1011.101?

Fractional Binary Numbers



- Representation
  - Bits to right of "binary point" represent fractional powers of 2
  - Represents rational number:

$$\sum_{k=-j}^{i} b_k \times 2^k$$

Slide adapted from: Mark Oskin-University of Washington

Fractional Binary Numbers: Examples

• Value	Representation
5 and 3/4	101.11_2
2 and 7/8	10.111_2
63/64	0.111111_2

- Observations
  - Divide by 2 by shifting right
  - Multiply by 2 by shifting left

Slide adapted from: Mark Oskin-University of Washington

Representable Numbers

- Limitation
  - Can only exactly represent numbers of the form $x/2^k$
  - Other rational numbers have repeating bit representations

- Value — Representation

1/3	$0.0101010101[01]\ldots_2$
1/5	$0.00110011001[0011]\ldots_2$
1/10	$0.000110011001[0011]\ldots_2$

Slide adapted from: Mark Oskin-University of Washington

Fixed Point Representation

- Represent the integer and fractional part of a number using fixed number of digits (bits)
- Example 1: 32 bits for the integral part and 32 bits for the fractional part
- Example 2: 20 bits for the integral part and 44 bits for the fractional part
- Precision: Smallest possible difference between two numbers
- Range: Difference between smallest and largest possible number
- Precision:
- Range:

Fixed Point Pros and Cons

- Pros
  - Simple
  - The same hardware that does integer arithmetic can do fixed point arithmetic
  - In fact, the programmer can use int with an implicit fixed point
- Cons
  - There is no good way to pick where the fixed point should be

    Sometimes you need range, sometimes you need precision. The more you have of one, the less of the other

Slide adapted from: Mark Oskin-University of Washington

IEEE Floating Point

- Floating point representation: Analogous to scientific notation
  - Not 12000000 but 1.2×10^{7}
  - Not 0.0000012 but 1.2×10^{-6}
- IEEE Standard 754
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs
- Driven by numerical concerns
  - Nice standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard

Slide adapted from: Mark Oskin-University of Washington

Floating Point Representation

- Numerical Form:
  $(-1)^s M\ 2^E$
- Sign bit s determines whether number is negative or positive
- Significand (mantissa) M normally a fractional value in range [1.0,2.0).
- Exponent E weights value by power of two

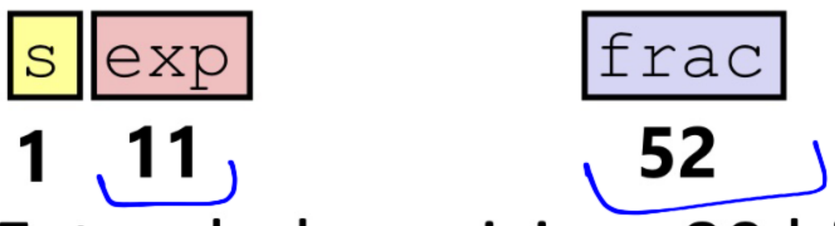
- Encoding
- MSB **s** is sign bit s
- **frac** field encodes M (but is not equal to M)
- **exp** field encodes E (but is not equal to E)

Slide adapted from: Mark Oskin-University of Washington

Types of floating points

- Single precision: 32 bits (float)

- Double precision: 64 bits (double)



- Extended precision: 80 bits (Intel only)

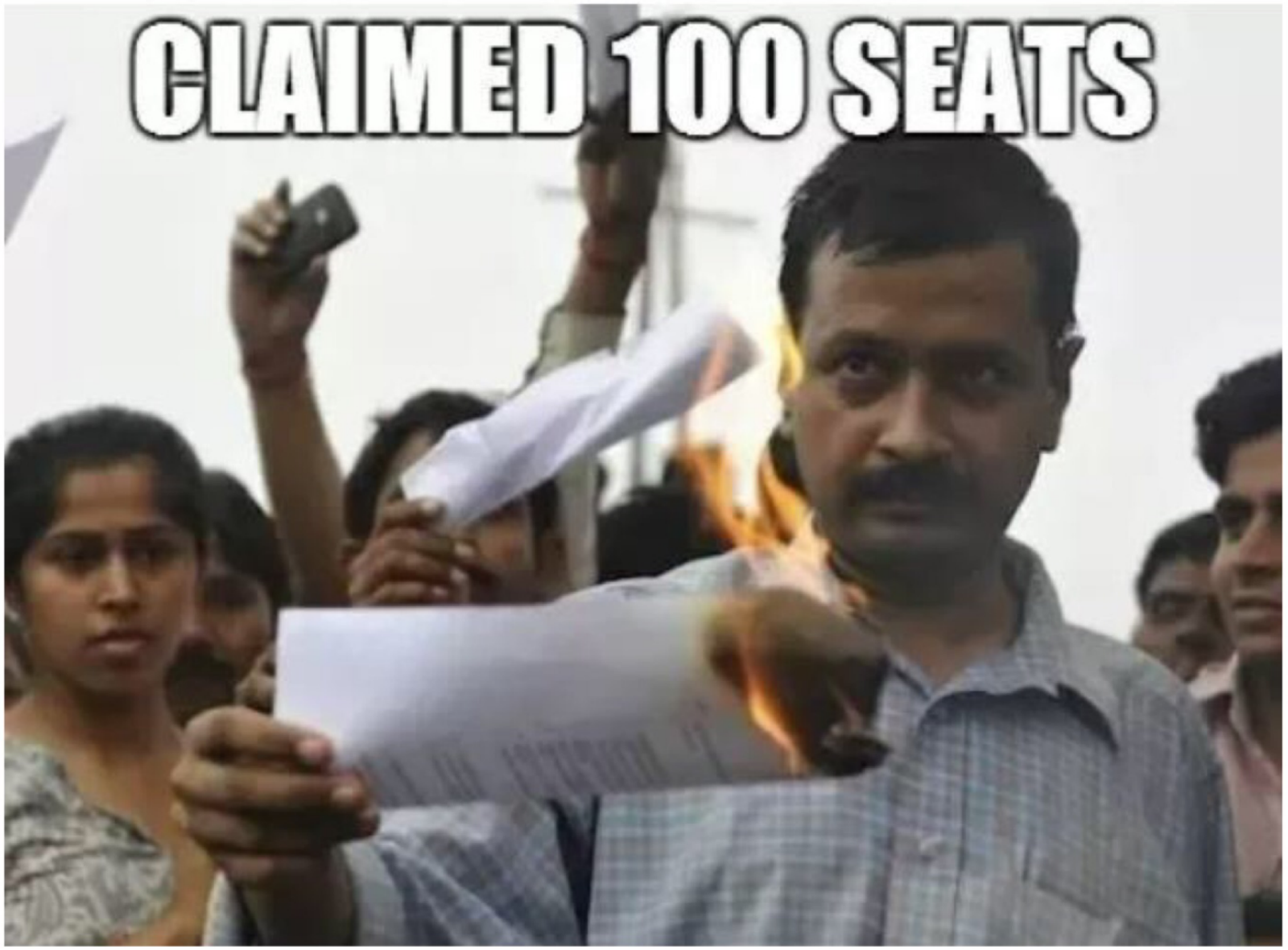
Slide adapted from: Mark Oskin-University of Washington

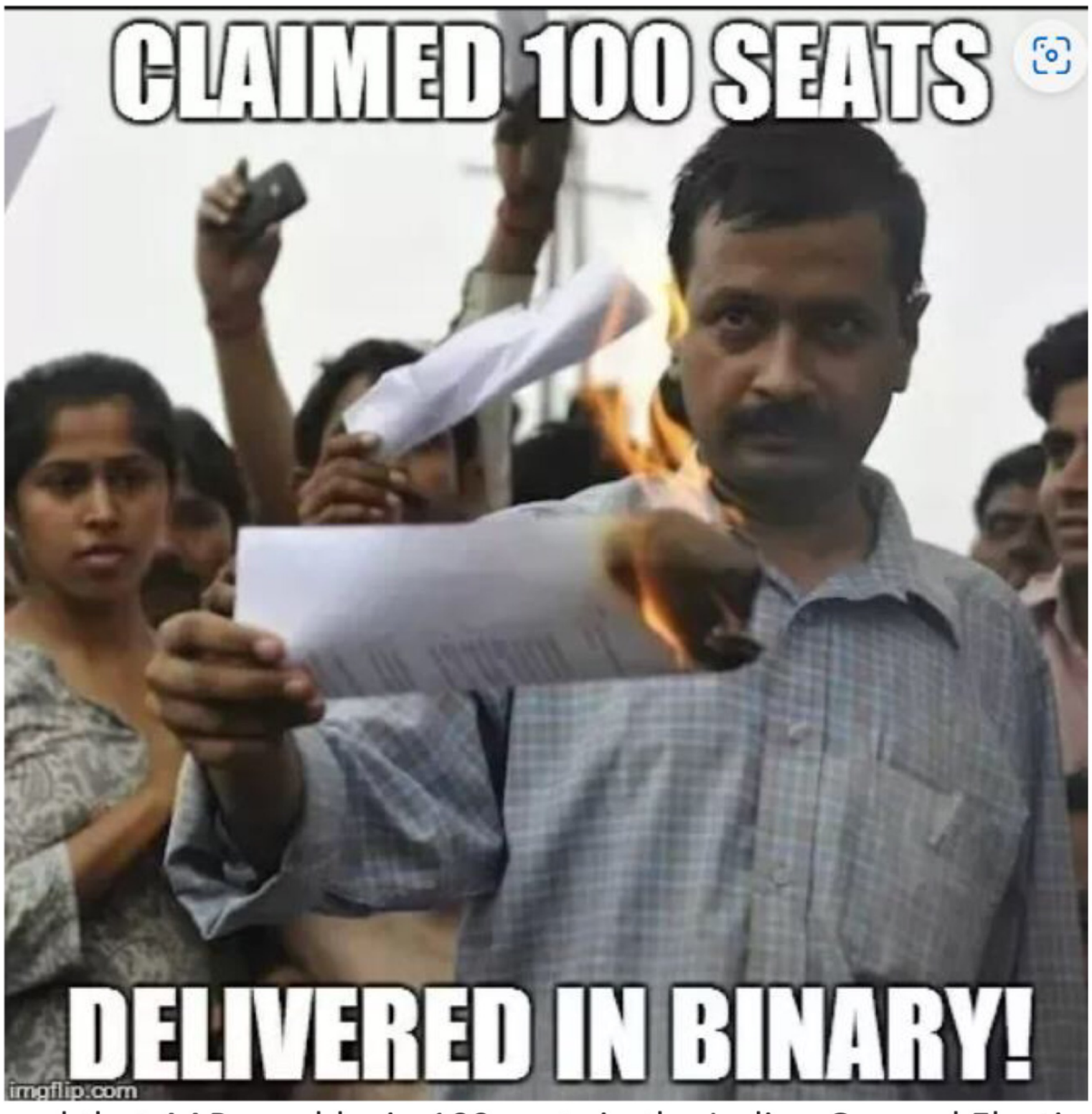
Other Data Representations

- Sizes of objects (in bytes)

Java Data Type	C Data Type	Typical 32-bit	x86-64
boolean	bool	1	1
byte	char	1	1
char		2	2
short	short int	2	2
int	int	4	4
float	float	4	4
	long int	4	8
double	double	8	8
long	long long	8	8
	long double	8	16
(reference)	pointer *	4	8

Slide adapted from: Mark Oskin-University of Washington


CLAIMED 100 SEATS



Kejriwal claimed that AAP would win 100 seats in the Indian General Elections 2014. What he didn't tell was that the number was in binary ;) *In binary, 100 is equal to 4, which is the number of seats AAP got in the 2014 General Elections*

Trees

What are Trees?

What are Trees?

- Non-linear data structure
- To represent hierarchical organization of data
- Examples

To store hierarchy of people

Image source: Shweta, Department of CSE, IIT Delhi

Directory Structure

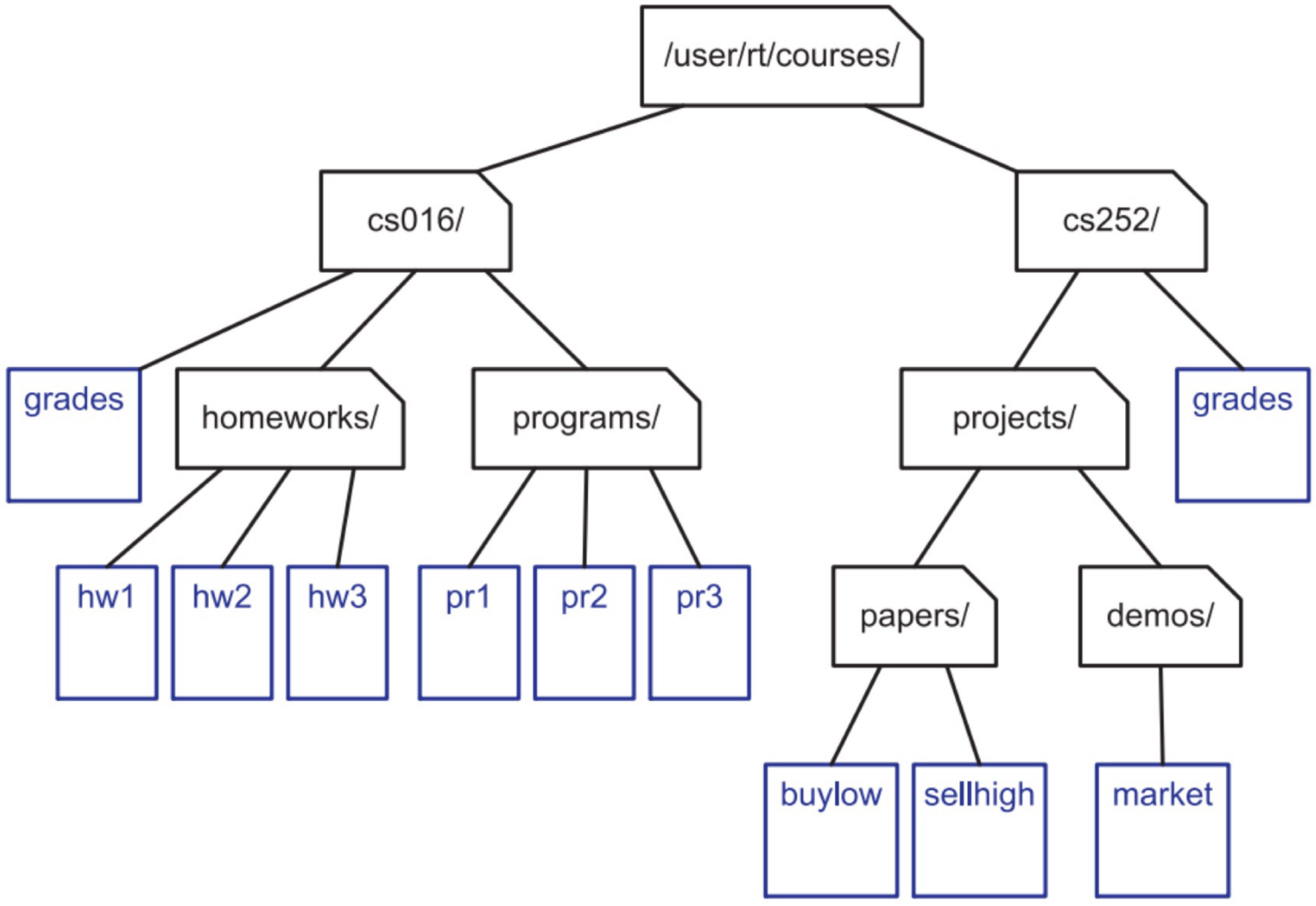


Image source: Data Structures and Algorithms in C++, Goodrich et al.

Decision Trees

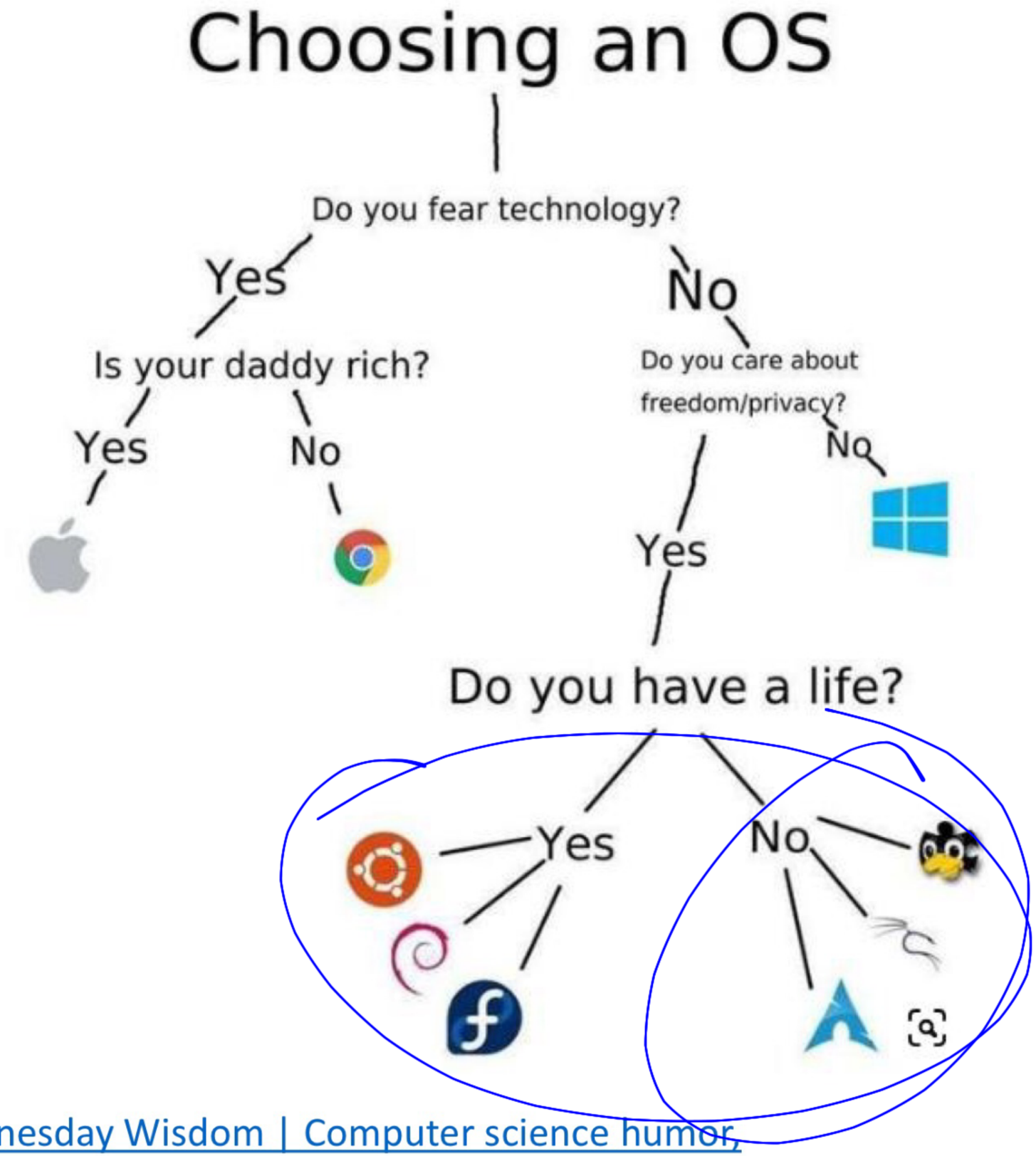


Image source: [nixCraft — Some Wednesday Wisdom | Computer science humor, Programming humor, Computer humor (pinterest.com)]

Hierarchical Organization in a Company

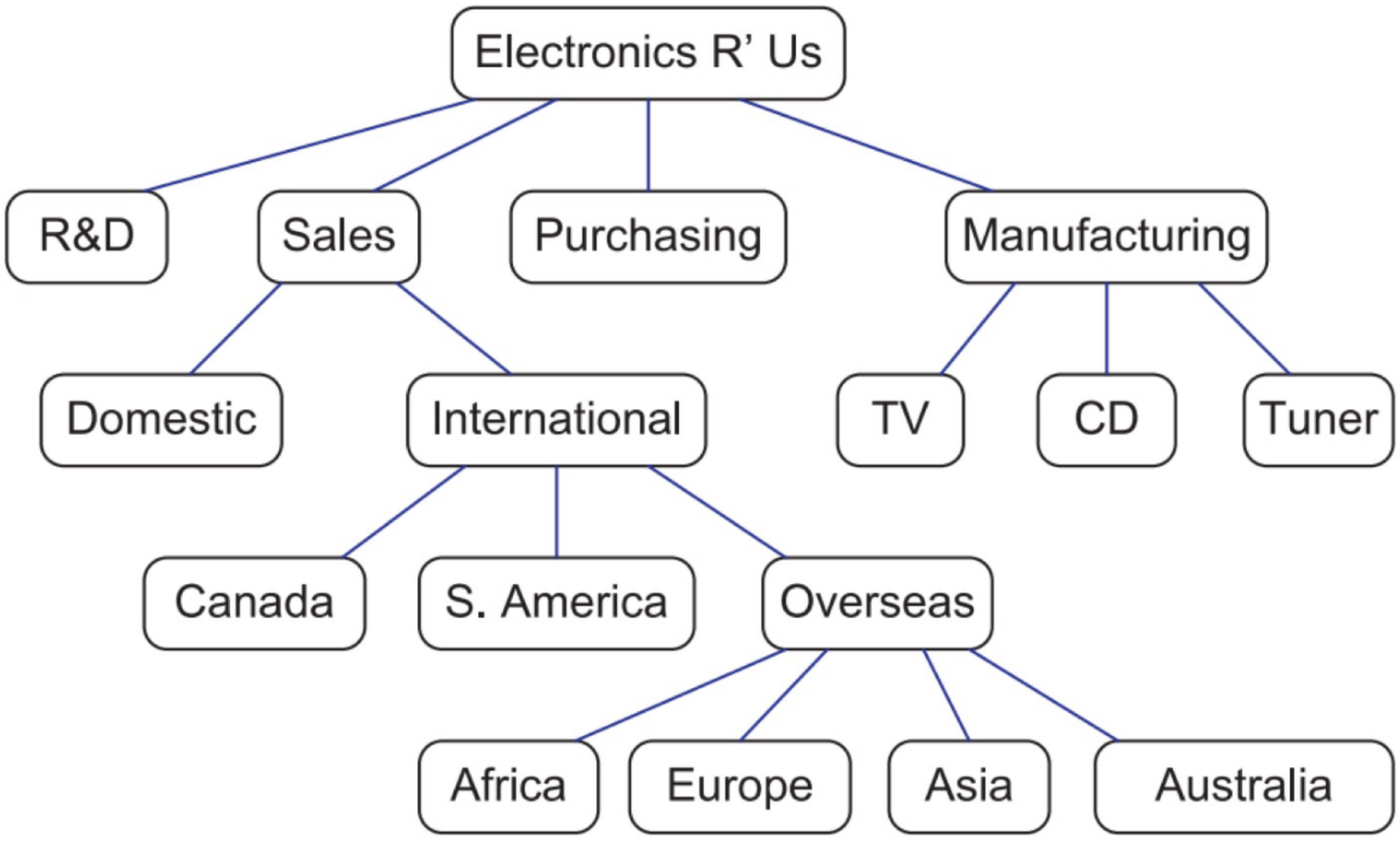
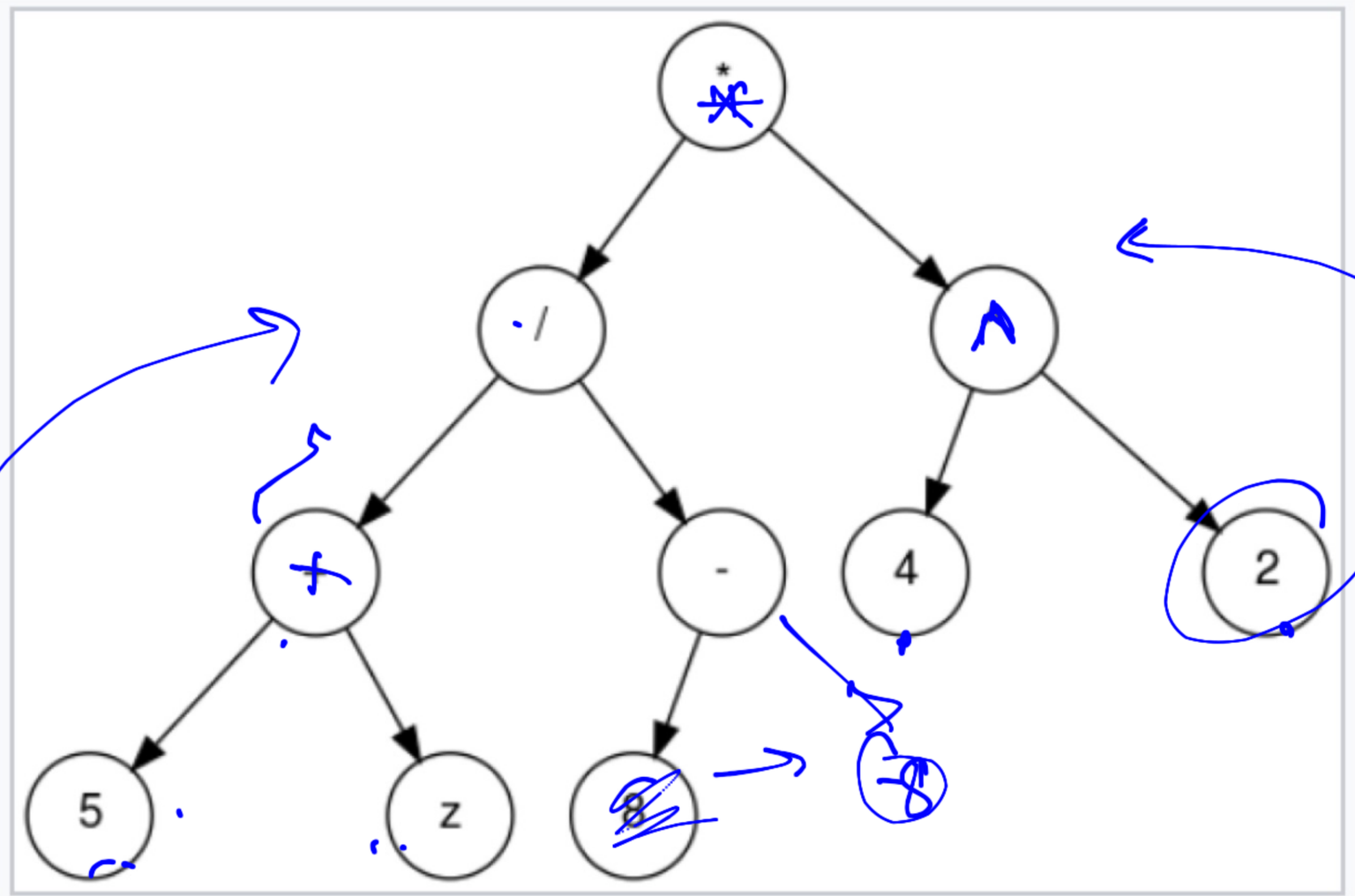


Image source: Data Structures and Algorithms in C++, Goodrich et al.

Arithmetic Expression Tree



Binary algebraic expression tree equivalent to ((5 + z) / -8) * (4 ^ 2)

Family Tree

Family Tree

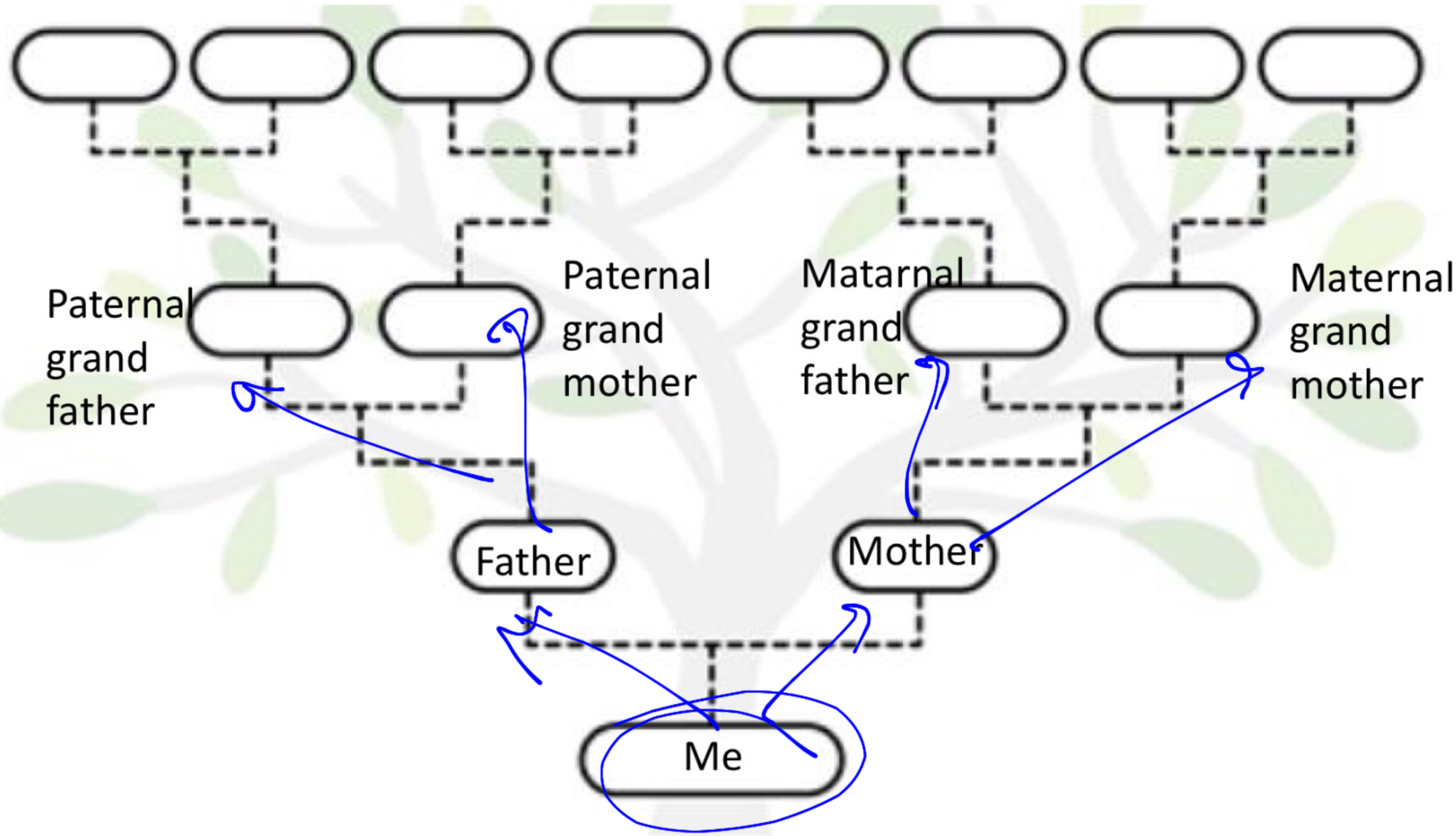
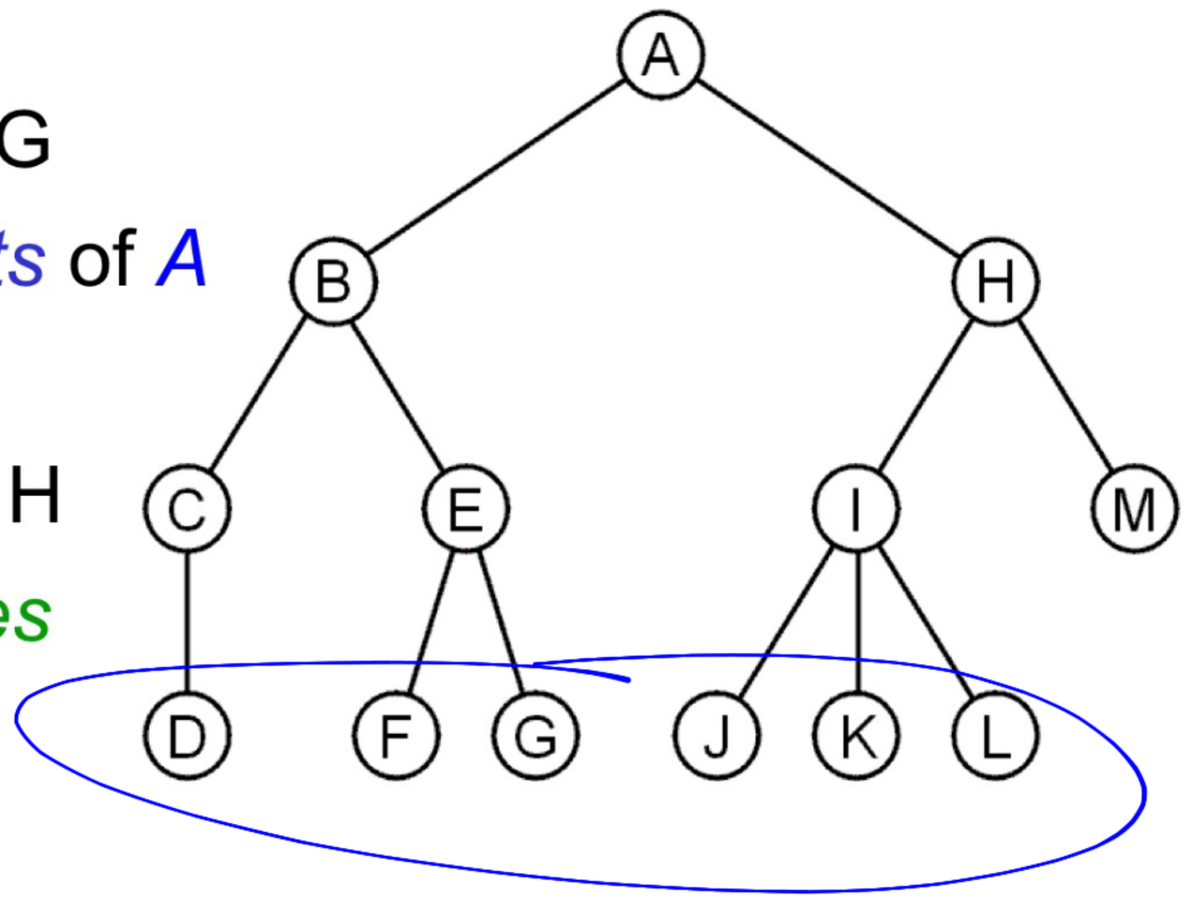


Image Source: [Blank Family Tree Template – Free Family Tree Templates]

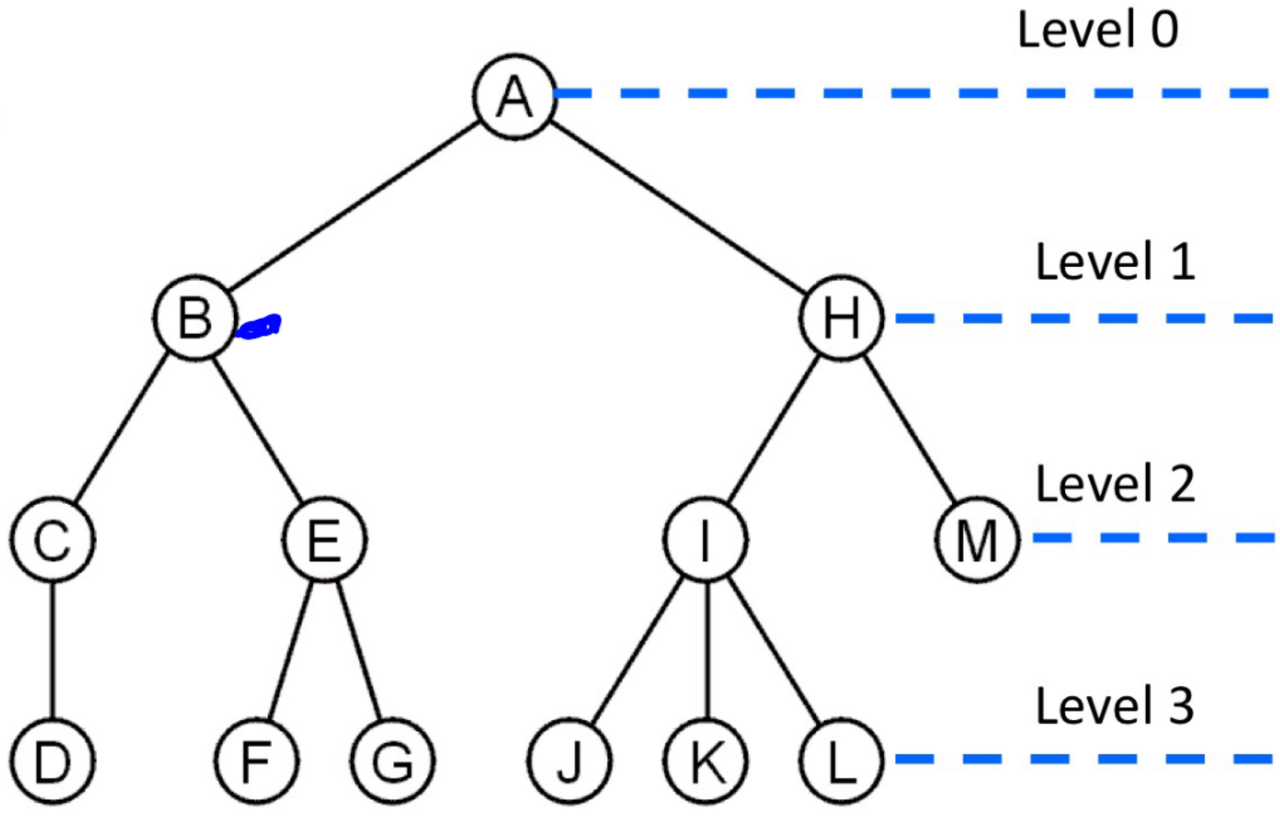
Tree: Notations

- Collection of nodes with parent-child relationship
- *A* is the *root* node
- *B* is *parent* of C & E
- *B* is *ancestor* of C, E, D, F, G
- D, C and G are *descendants* of *A*
- *C* is the *sibling* of E
- *I* and *M* are the *children* of H
- *D, F, G, J, K, L, M* are *leaves*



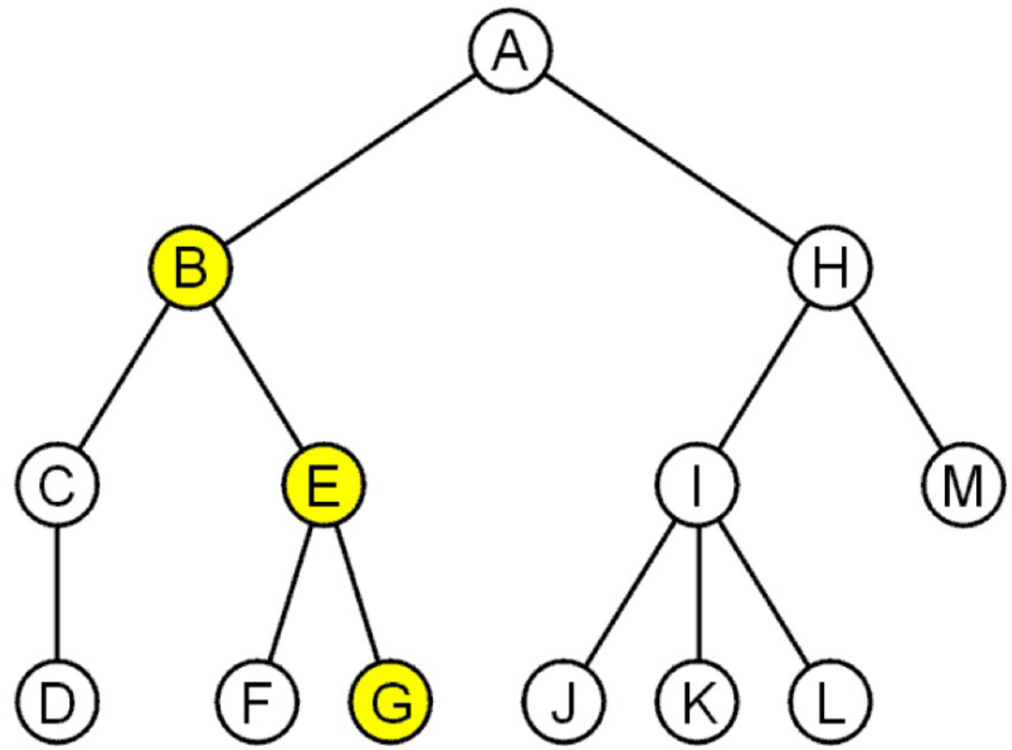
Tree: Notations

- *A, B, C, E, H, I,* are *internal nodes*
- The *degree* of node I is *3*
- The *depth (level)* of *E* is *2*
- The *height* of the tree is *3*



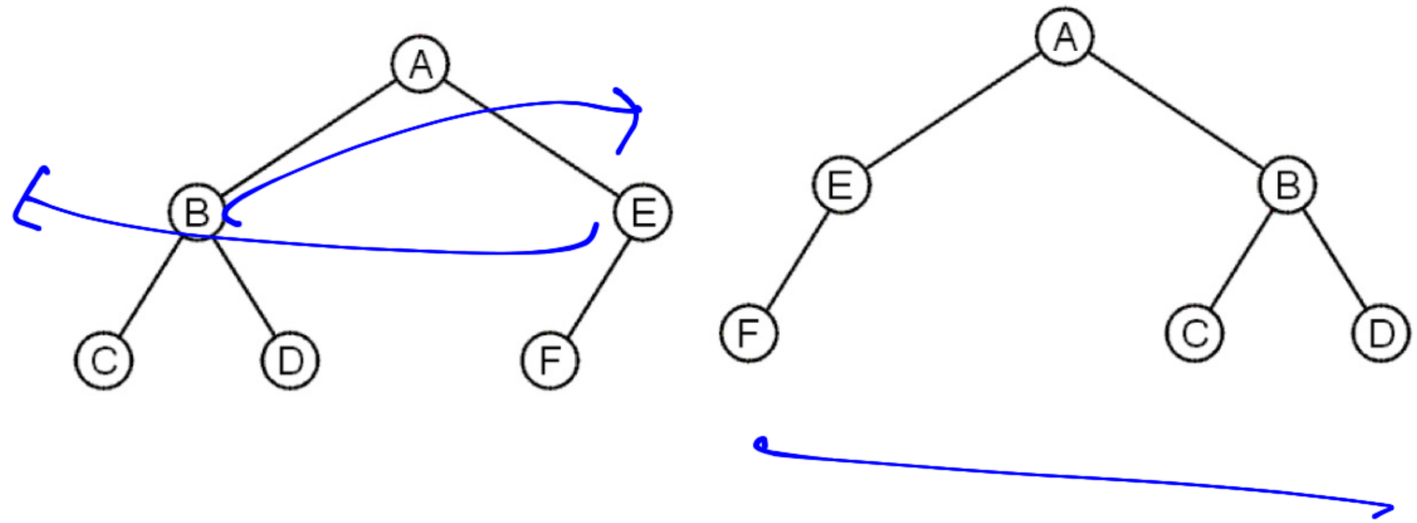
Tree: Notations

- A path is a sequence of nodes $(a_0, a_1, ..., a_n)$ where a_{k+1} is a child of a_k
- The length of this path is n
- *E.g.*, the path (B, E, G) has length 2
- For each node in a tree, there exists a unique path from the root node to that node



- Depth of a node is the length of the path form root
- The height of a tree is defined as the maximum depth of any node within the tree
- The height of a tree with one node (root) is zero

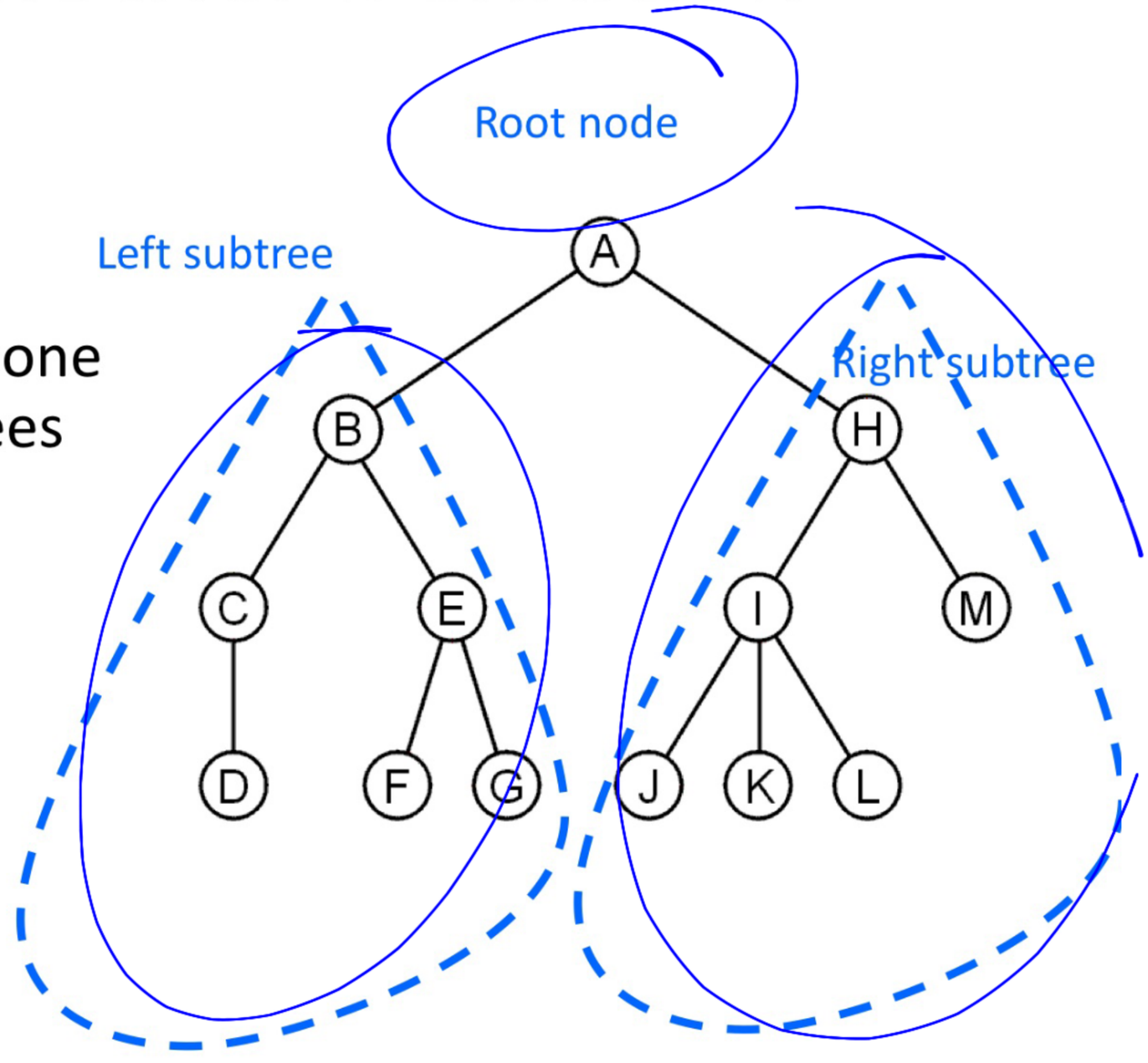
Binary Tree



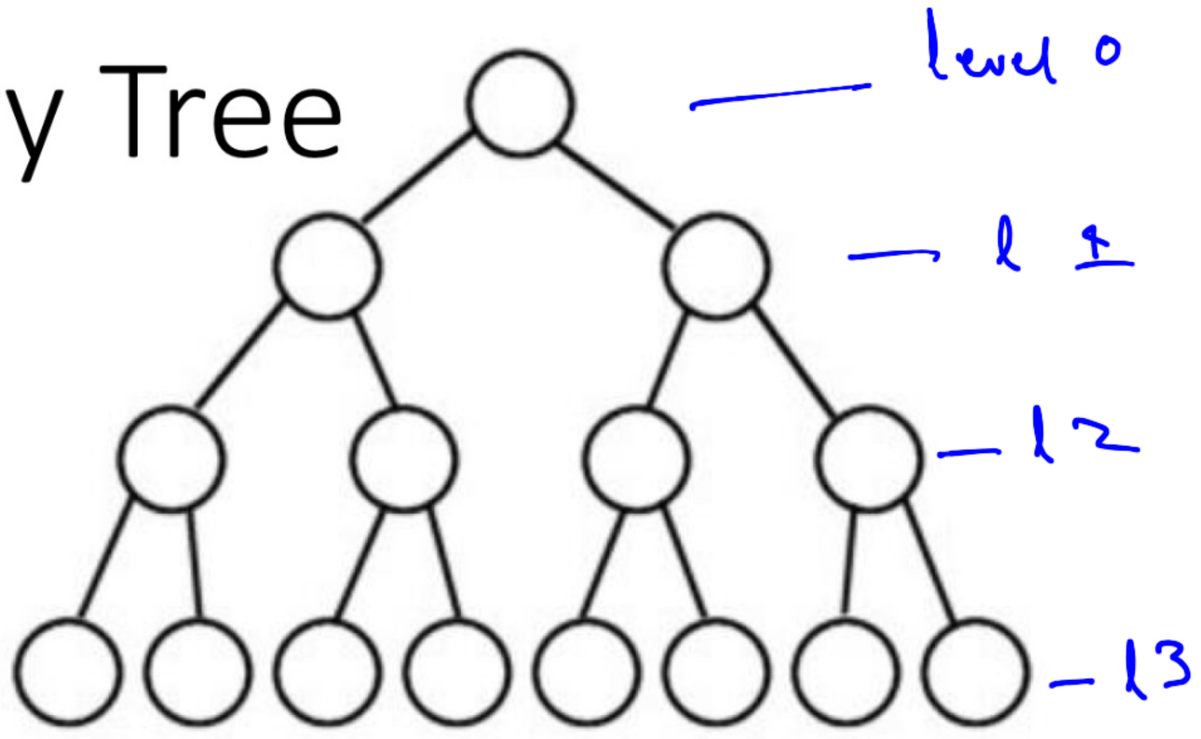
- An **ordered tree** is one in which the children of each node are ordered
- Binary tree is an ordered tree where each node may have at most 2 children
- A “left child” and a “right child”
- The above two trees are equivalent if we consider them to be unordered trees

Binary Tree: Another Definition

- A binary tree is
  - Either a leaf node
  - Or a root node with one or two binary subtrees



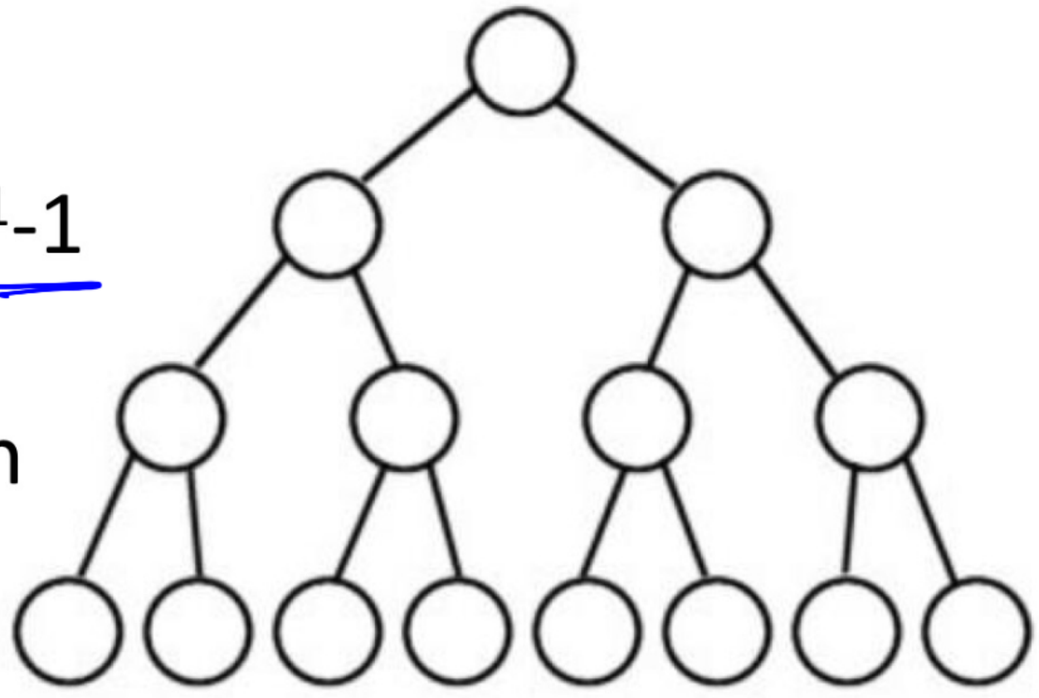
A Complete Binary Tree



- Level i has 2^i nodes
- In a tree of height h
  - Leaves are at depth h
  - No. of leaves is 2^h
  - No. of internal nodes = $1+2+2^2+\ldots+2^h-1 = 2^h-1$
  - No of internal nodes = no of leaves -1
  - Total no. of nodes is $2^{h+1}-1 = n$
- In a tree of n nodes
- No of leaves is $(n+1)/2$
- Height = log2(no of leaves)

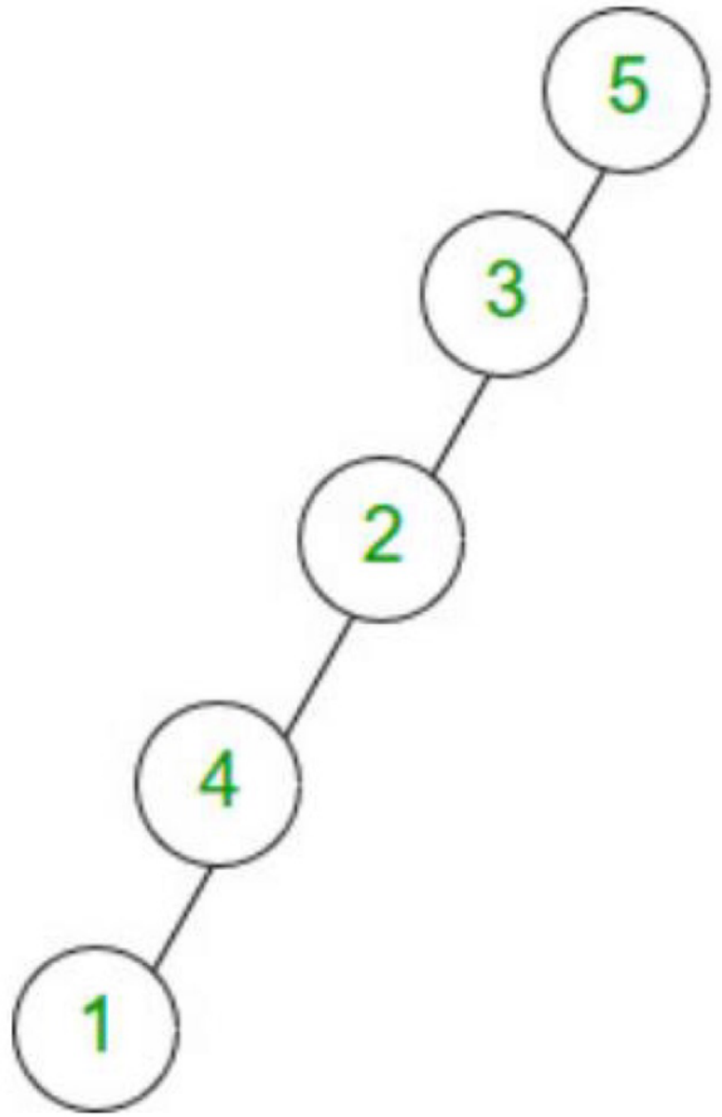
Minimum Height of a Binary Tree

- A binary tree of height h has
- At most 2^i nodes at level i
- At most $1+2+2^2+\ldots+2^h = 2^{h+1}-1$ nodes
- If the tree has n nodes then
  - $n \leq 2^{h+1}-1$
- Hence $h \geq \log_2[(n+1)/2]$



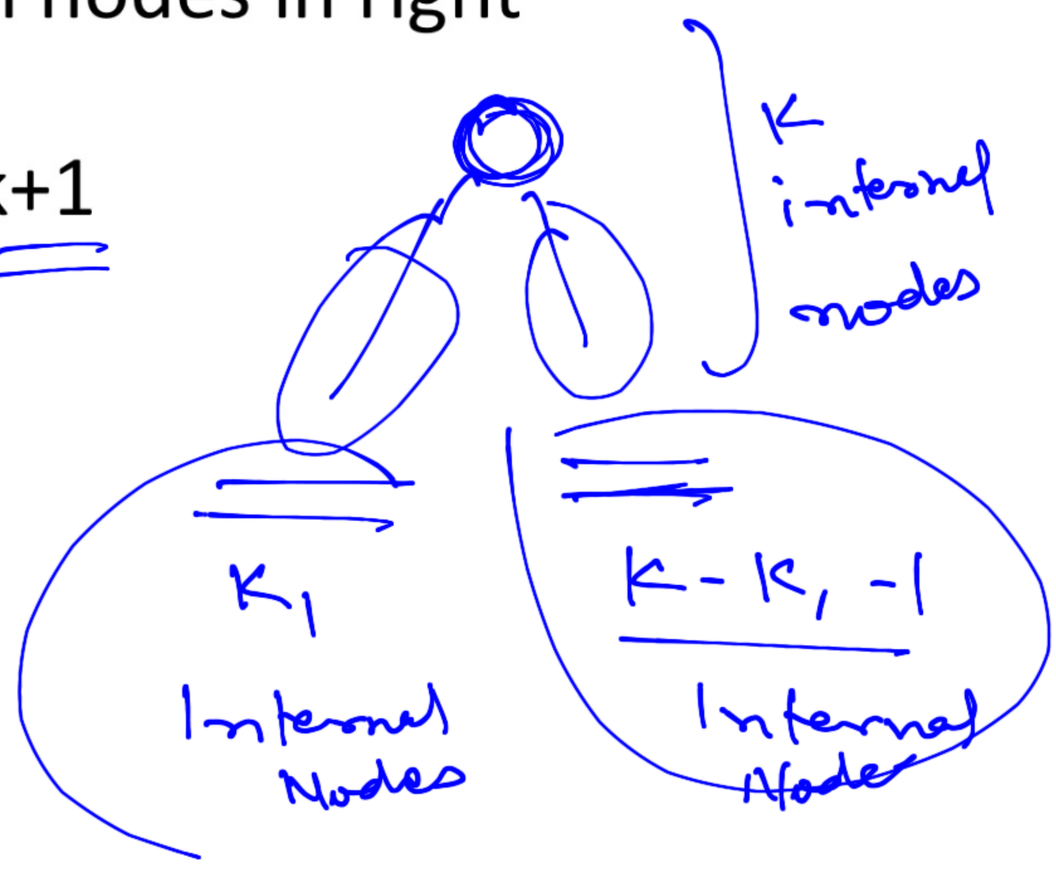
Maximum Height of a Binary Tree

- Maximum height of a binary tree
- A binary tree on n nodes has height at most n-1
- This is obtained when every node (except the leaf) has exactly one child



Number of Leaves in a Binary Tree

- Theorem: No. of leaves ≤ 1 + no of internal nodes.
- Proof: By induction on no of internal nodes
- Tree with 0 node has no leaf and no internal node
- Tree with 1 node has one leaf and no internal node
- Assume statement is true for tree with k-1 internal nodes
- A tree with k internal nodes has k_1 internal nodes in left subtree and ($k-k_1-1$) internal nodes in right subtree
- No of leaves <= (k_1+1)+($k-k_1$) = k+1



Data Structures for General Trees

```
class DLLNode {
private:
     class Tree *node;
public:
     DLLNode *next, *prev;
};
```

```
class Tree {
private:
   int data;
public:
   class DLLNode *header, *trailer;
};
```

Data Structure for Binary Trees

```
class BinaryTreeNode {
private:
    int data;
    class BinaryTreeNode *left, *right;
}
```

Data Structure for Binary Trees

```
class BinaryTreeNode {
private:
    int data;
    class BinaryTreeNode *left, *right, *parent;
}
```

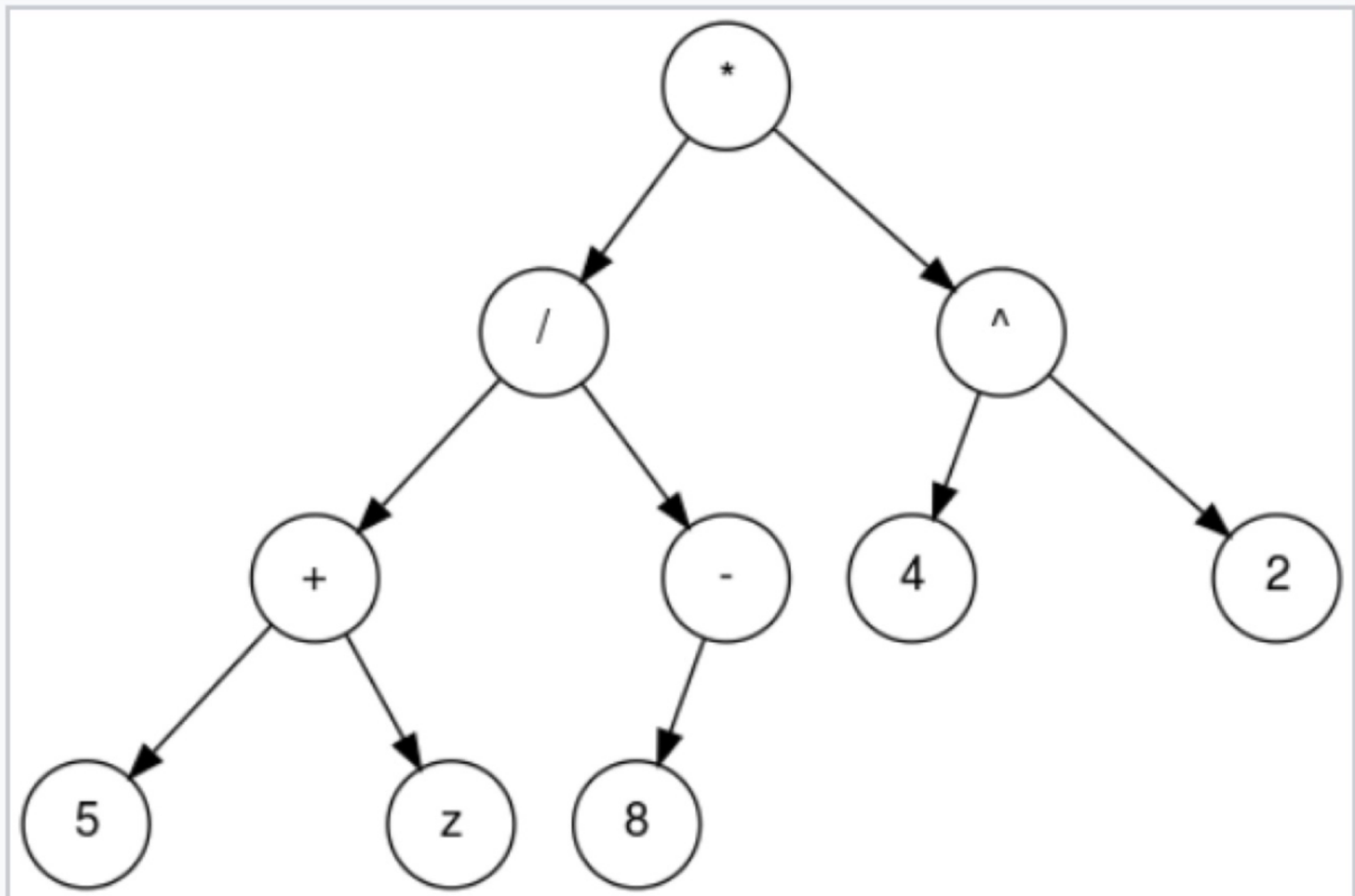
Finding Height of a Binary Tree

```
class BinaryTreeNode {
public:
    int height() {
        if (this == 0) return 0;
        return max(left->height(), right->height());
    }
private:
    int data;
    class BinaryTreeNode *left, *right, *parent;
};
```

Counting Number of Nodes

```
int nodes() {
    if (this == 0) return 0;
    return left->nodes() + right->nodes();
}
```

Arithmetic Expression Tree



Binary algebraic expression tree equivalent to ((5 + z) / -8) * (4 ^ 2)

Evaluating Expression Trees

```
class BinaryExpressionTree {
private:
    double data;
    char op;
    class BinaryExpressionTree *left, *right;
public:
    double evaluate() {
        if (this == 0) return 0;
        else if ((left == 0) && (right == 0)) // Leaf node
            return data;
        else if (op == '+')
            return left->evaluate() + right->evaluate();
        else if (op == '-')
            return left->evaluate() - right->evaluate();
        else if (op == '*')
            return left->evaluate() * right->evaluate();
        else if (op == '/')
            return left->evaluate() / right->evaluate();
    }
};
```

Thank You