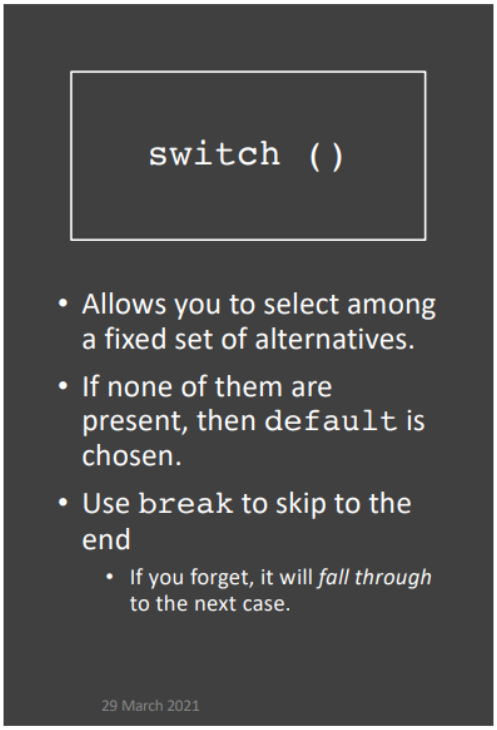
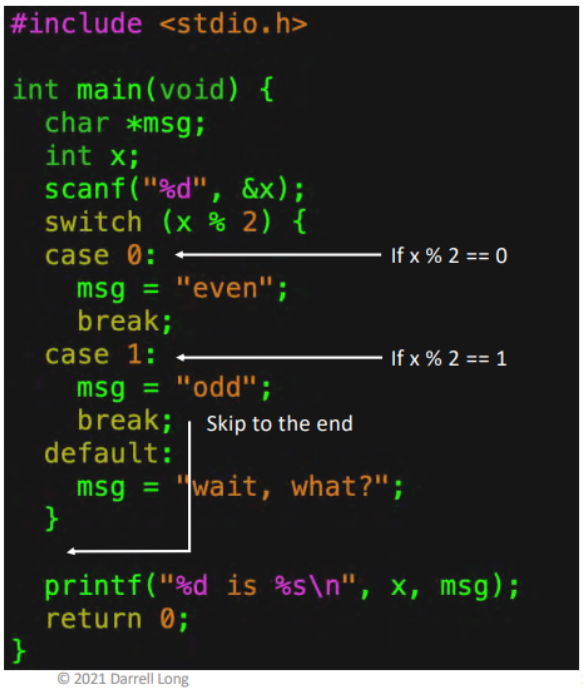
Control Structure (lecture 6):

- Zero is false, non zero is true
- Logical expressions have type int
- Even though {} are not required for a single statement
- Short circuit evaluation allows expression to be determined asap
- switch()





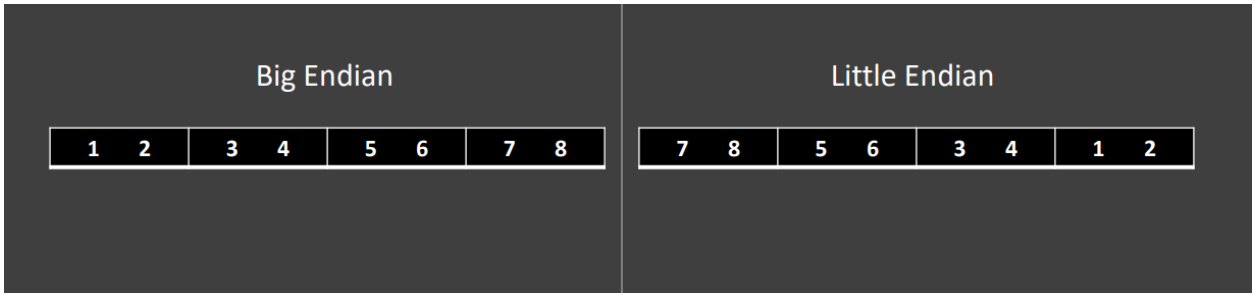
Numbers (bits operation L7):

- Two's complement arithmetic: flip the bits in m, 0 → 1 and 1 → 0, and then add 1 to the result

Let's try $5 = 0101_2$ using 4 bit integers:

- $\sim 0101_2 + 1 = 1010_2 + 1 = 1011_2$
- $0101_2 + 1011_2 = 10000_2$ but we only have four bits, so we drop the high-order (carry) bit and get 0000_2.

- Floating point Numbers: subset of real numbers, subset of rational, subset of integers. Mistake to think of them as reals, or rationals, they are an approximation
- Single precision double precision?
- Big endian, little endian: little endian means it is the low address byte, big endian means high address byte

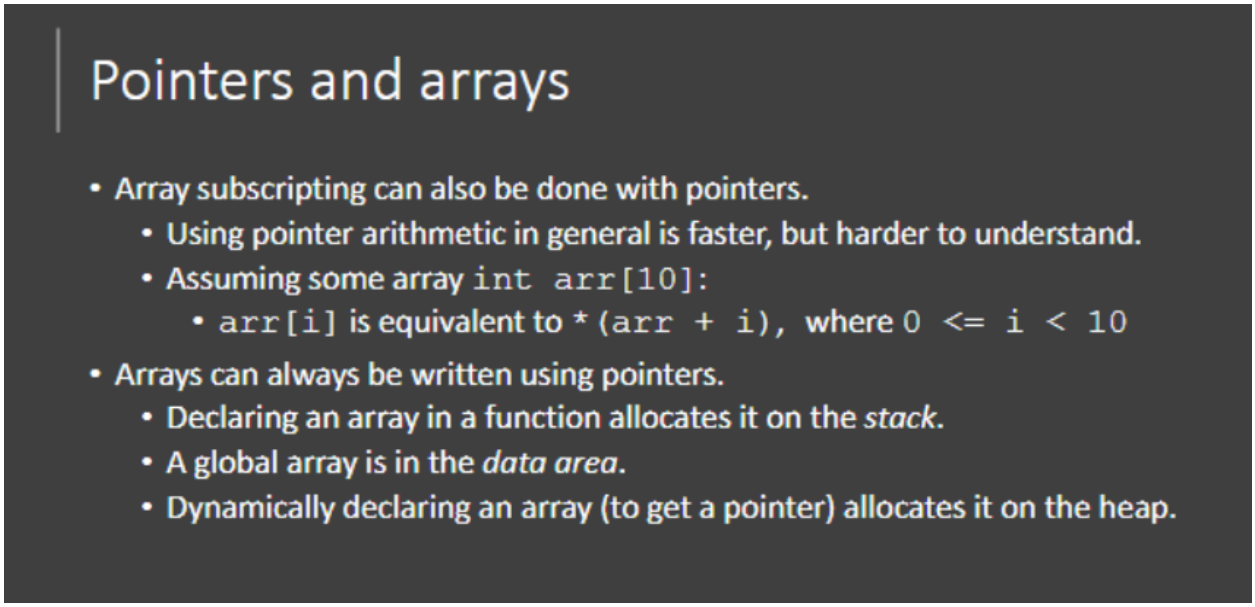


- Random numbers: true random numbers cannot be created using computers. It has advantage of repeatability but has the disadvantage of predictability
- Type promotion: promotes lower type to a higher type when expression has mixed type

Pointer and Dynamic Memory (lecture 8):

- A variable that holds a memory address
- The variable points to the location of an object in memory
- not all pointers contain an address
- Pointers that don't contain an address are set to NULL pointer
- **Memory address**: stored in registers that can be accessed by a specific number (address). Usually, bytes are grouped into words
- Pointers are said to point at the address they are assigned
- Can assign a pointer the address of a variable using the &
- Multiple pointers can point to the same address
- **Dereferencing pointer**: The object pointer points to can be accessed through dereference. (*)

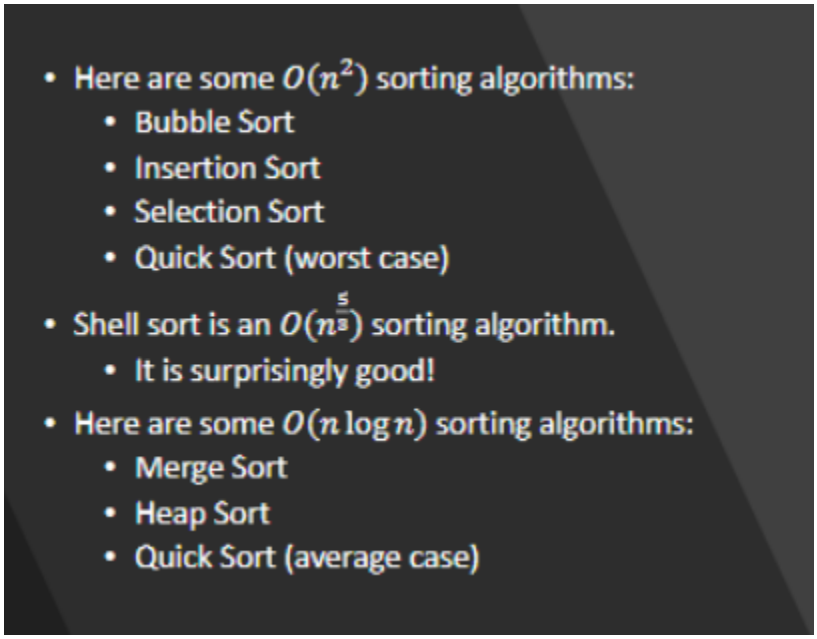
- **Passing by reference:** allows returning multiple values. Allows passing large amount of data
- **Pointers and Arrays:**



- Strings is a pointer to an array of chars. Strings can be indexed, passed by ref
- Pointers to pointers: can be used to pass arrays of arrays, such as list of strings
- Function Pointers • Points to executable code in memory instead of a data value. • Dereferencing a function pointer yields the referenced function.

Sorting:

- How many times can we double 1 before we exceed n? Log n
- Merge sort nlogn



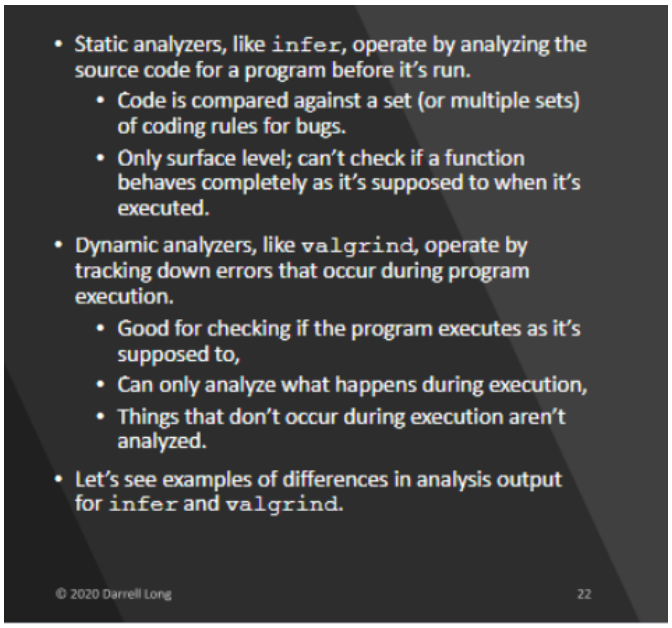
-

- **Max Heap:** a single node is a heap. It is a heap if the parent is a heap and the trees rooted at both children are heaps. A parent's value (key) is greater than that of either child. There is no order among the children

Stacks and Queues:

- **Arrays** allow random access (each element can be accessed directly in constant time). **Linked lists** allow sequential access (each element can only be accessed in a particular order)
- Stacks = lifo
- Capacity depends on allocation. Often implemented with arrays
- Can also be created using linked lists. Each element of the stack points to the next
- Queue is an ADT, FIFO

Dynamic Memory Allocation (lecture 11):

- DMA is allocating memory for variables on the heap during program run-time
- Different from compile time allocation. CTA requires the exact size and type of storage at compile time, DMA is calculated and allocates the exact memory it needs during run time
- Why is DMA good? Stack space is limited so we want variables to last beyond the lifetime of its current scope
- **The heap:** is a large region of unmanaged memory. Variables using heap memory can be accessed globally with access to the pointer
- free(): Pointers that have been freed should be set to NULL to mitigate user-after-free vulnerabilities
- Valgrind: detects use of uninitialized memory, reading/writing memory after freed, reading/writing off the end of allocated blocks of memory, reading/writing inappropriate ares on the stack, memory leaks
- Static vs dynamic analyzers:
- 

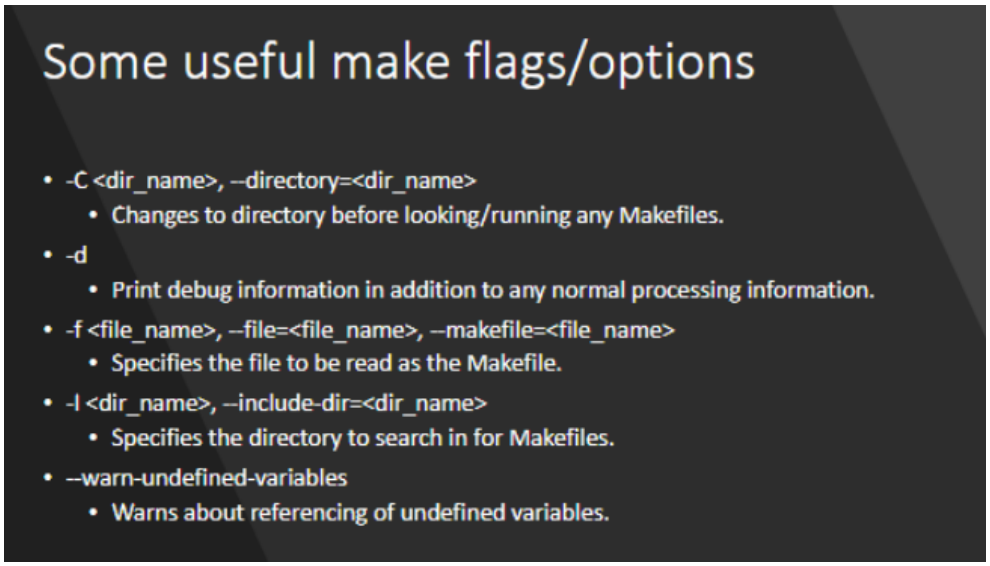
- **Infer:** static analysis tool, checks for null pointer exceptions, resource leakage, race conditions, and missing lock guards. What are reported as memory leaks are freed later on in the code in other functions

Recursion (L12):

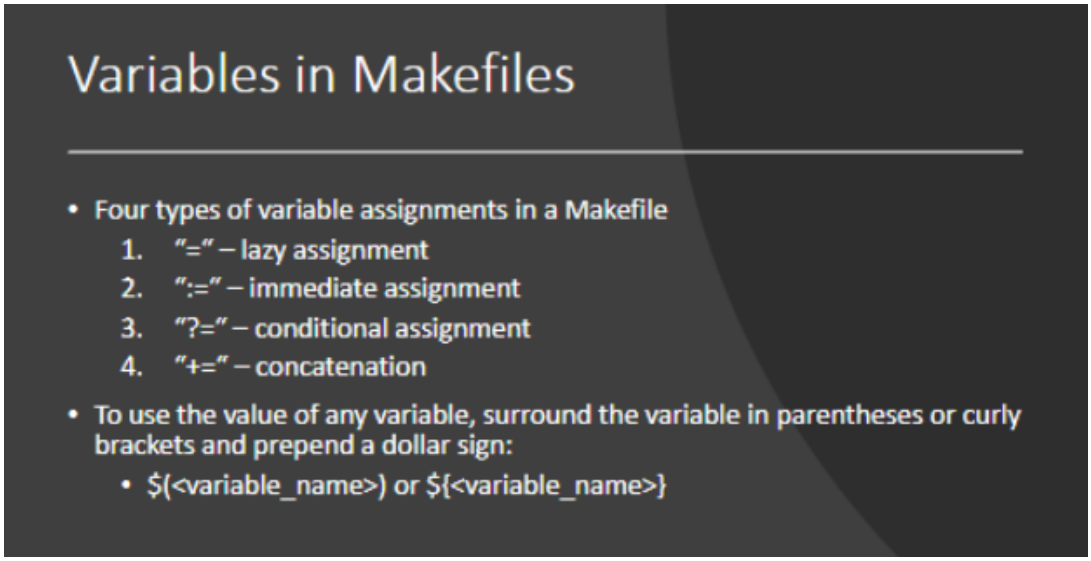
- Fibonacci numbers:
-
```
int fib_1(int k) {
  if (k == 0 || k == 1) {
    return k;
  } else {
    return fib_1(k - 1) + fib_1(k - 2);
  }
}
```
- Is recursive and runs in O(2^n)
-
```
int fib_2(int k) {
  int a = 0, b = 1, s = 0;
  if (k < 2) {
    return k;
  }
  for (int i = 2; i <= k; i += 1) {
    s = a + b;
    a = b;
    b = s;
  }
  return s;
}
```
- Iterative alg and runs in O(n)
- How fast can we search? Binary search. O(log(n)). Binary search requires that the array is sorted
- Recursion is for searching

Make (L14):

- Makefile composed of rules
- 1. A target 2. A set of dependencies 3. A set of commands
- Target = name of rule
- Make <target_name>
- Phony target: a target that when its rule is executed, does not produce a file with the same name
- Flag options



- Variables in Makefiles

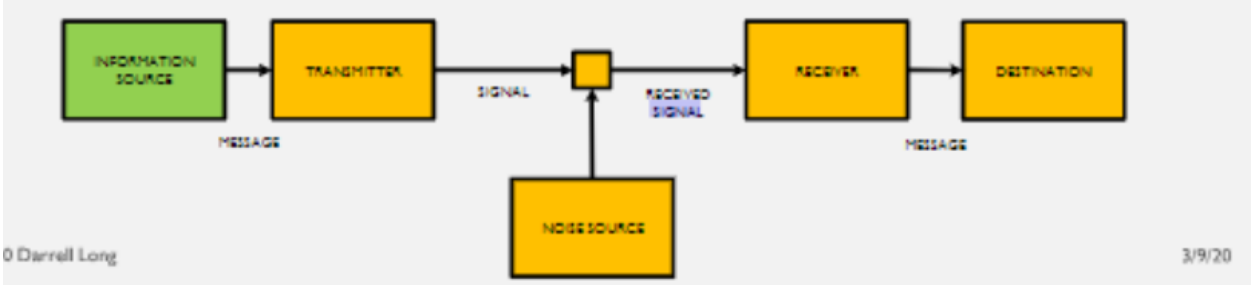


- Lazy assignment: "=" contents of the variable assignment are stored as-is, variables as variables, reference as reference
- Immediate Assignment: ":=" Variables assignment is evaluated, and result assigned to the variable
- Conditional assignment "?="
- Concatenation - "+=" behaves like = if the variable in questions has not been defined
- Command: An action to be executed
- Commands, compilers and flag

- Variables are used to factor makefiles to make them easier to maintain.
- Some conventional Makefile variables used for compiling C programs:
  - CC – the C compiler to use (typically gcc or cc or clang).
  - CFLAGS – a list of compiler flags (each flag is prefixed with a hyphen "-").
    - Some good flags to use when compiling C programs:
      - Wall, Wextra, Wpedantic, Werror, g.
  - OBJ – a list of object files to build and link.
  - SRC – a list of source files (.c files).

- The wildcard function: can be used in rules as the "*" rm *.o
- The patsubst function: formatted as $, finds white-space words that match and replaces them
- Makefile can include other makefiles

Data Compression (L15):

- 5 parts of communication
- **Information source:** produces a message or a sequence of messages to be communicated to the receiving terminal
- 

- **Transmitter:** operates on the message to produce a signal suitable for transmission over the channel. This is where compression/ encryption occurs
- **Channel:** the medium through which a signal is transmitted. Examples: Radio freq, beams of light, wires, fibre optic
- **Receiver:** performs the inverse operation of that done by the transmitter
- **Destination:** the intended target of the message
- Greatest probability of correctly guessing the symbol = least entropy
- Consider message AABC:

- Consider message (2), which is $AABC$.
  - $p(A) = \frac{1}{2}, p(B) = \frac{1}{4}, p(C) = \frac{1}{4}$
- Using the formula for entropy we see that

$$\begin{aligned} H &= -\sum_{i=1}^{n} p_i \log_2(p_i) \\ &= -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) \\ &= \frac{3}{2} \end{aligned}$$

- Which means the entropy for this message is $\frac{3}{2}$.

-

- Consider message (3), which is $ABCD$.
  - $p(A) = \frac{1}{4}, p(B) = \frac{1}{4}, p(C) = \frac{1}{4}, p(D) = \frac{1}{4}$
- Using the formula for entropy we see that

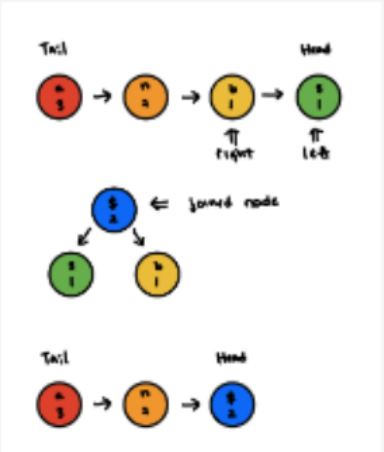
$$\begin{aligned} H &= -\sum_{i=1}^{n} p_i \log_2(p_i) \\ &= -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) \\ &= 2 \end{aligned}$$

- Which means the entropy for this message is 2.

-

- Huffman coding: construct histogram, pq with lower fq means higher priority,
- Building tree

- We will create a Huffman tree using the nodes in the priority queue.
- While the queue has more than one node,
  1. Dequeue a node. This will be the left child node.
  2. Dequeue another node. This will be the right child node.
  3. Create a parent node for the left and right child nodes. The frequency of the parent node is the sum of its children's frequencies.
  4. Enqueue the parent node.
- The last node in the queue is the root of the tree.



© 2020 Darrell Long

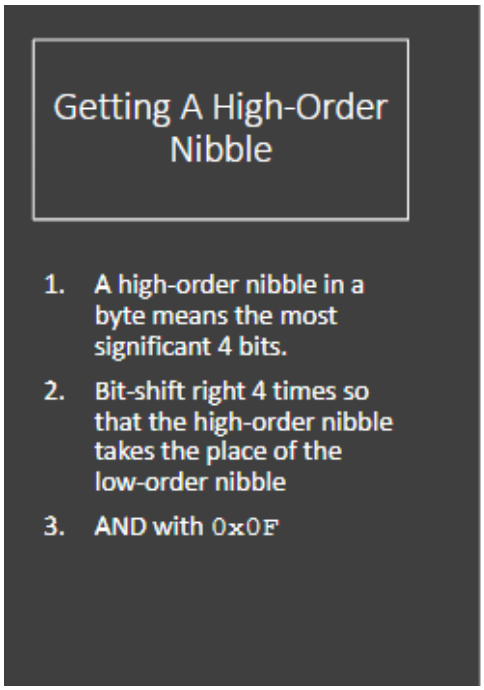
- Building code table

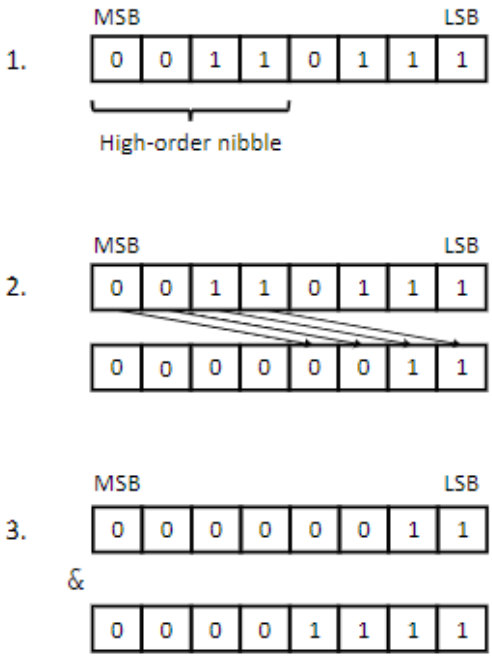
- We now populate a table of codes.
  - Like the histogram there are 256 indices, and each index stores a binary code.
- We will utilize a stack of bits to keep track of the code.
- To build these codes, we perform a *post-order traversal*.
  - If we walk down to the left child, we push a 0 to the bit-stack.
  - If we walk down to the right child, we push a 1 to the bit-stack.
  - If we reach a leaf node, the code for the node's symbol is the code in the bit-stack.
- A code can be, at maximum, 256 bits long!

-

Bit Vectors (L16):

- Arithmetic right shift: sign bits are shifted in on the left
- Getting and setting high order nibble



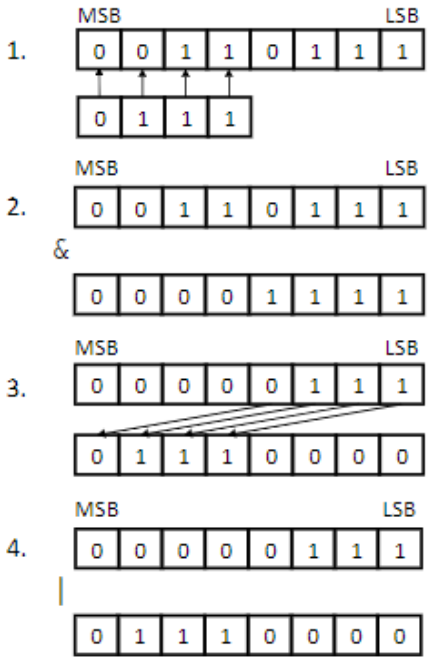


Setting A High-Order Nibble

1. We want to place a nibble into the higher-order bits of a byte
2. AND byte with `0x0F`
3. Bit-shift nibble left 4 times
4. OR byte with bit-shifted nibble

7 February 2020

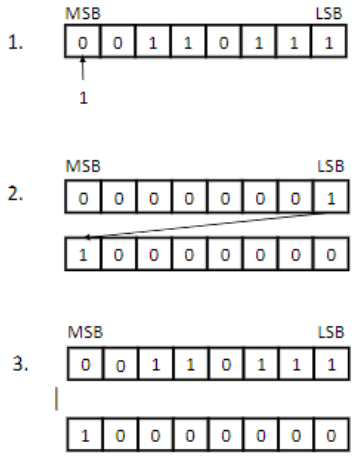
© 2020 Darrell Long



Setting, clearing, and getting a bit

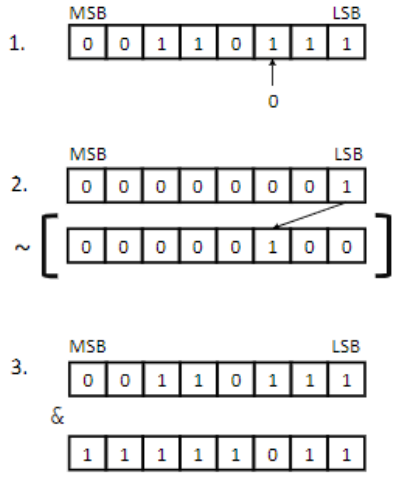
Setting A Bit

1. We want to set the bit at index 7 in a byte.
2. Take another byte with the bit at index 7 set
   - Can do this by shifting `0x1` left 7 times.
3. OR the bytes together to set the bit.



Clearing A Bit

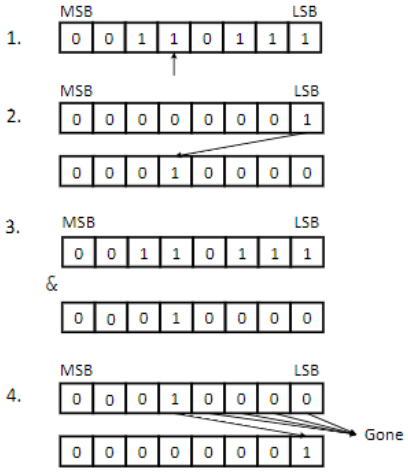
1. We want to clear the bit at index 2 in a byte.
2. Take another byte with all bits set *except* the bit at index 2.
   - Can do this by shifting 0x1 left 2 times and taking the bitwise NOT of the result.
3. AND the bytes together to clear the bit.



Getting A Bit

1. We want to get, or return, the value of the bit at index 4 in a byte.
2. Take another byte with the bit at index 4 set.
   - Can do this by left-shifting 0x1 4 times.
3. AND the bytes together to mask every bit except the bit at index 4.
4. Right-shift the AND-ed result 4 times to get the value.

7 February 2020



Linked lists (L17):

- Singly linked list contains a data field and a pointer to the next node in the list
- Double list each node contains a data field and pointers to the next and previous nodes in the list
- Last node is a null pointer
- No fixed mem allocation
- No need to know the initial size of the list
- Implemented linear data structures like stacks and queues
- Requires extra mem
- **Circular Singly Linked List:** last node points back to the tail
- **Doubly linked lists:** each node has a pointer to both the previous and next nodes. Less memory efficient uses sentinel

- **Sentinel Nodes:** used to mark the ends of a linked list. When performing insertion, nodes will always go between 2 nodes

Trees (L18):

- A tree is a type of directed acrylic graph
- Tree can be null or pointing to two trees
- Node: smallest entity in a tree, containing some value
- Binary tree: each node has up to 2 children
- Root is the starting point of tree. If null, then the tree is empty
- Parent: a node that points to child nodes
- Child: a node connected to a parent node. Can also be the root of a subtree
- Subtree: A tree rooted at some node n. Must contain all descendants of n
- Leaf: A node that contains no children. Means that children are null
- Traversal:

```
void preorder_print(Node *root) {
    if (root) {
        printf("%d\n", root->key);
        preorder_print(root->left);
        preorder_print(root->right);
    }
}

void inorder_print(Node *root) {
    if (root) {
        inorder_print(root->left);
        printf("%d\n", root->key);
        inorder_print(root->right);
    }
}

void postorder_print(Node *root) {
    if (root) {
        postorder_print(root->left);
        postorder_print(root->right);
        printf("%d\n", root->key);
    }
}
```

Files (L19):

- **Long term- info storage:** We want to store large amounts of data. • The definition of large has changed by 6 orders of magnitude over the years. • Information stored must survive the termination of the process using it. • Memory (DRAM) does not survive turning the computer off. • Files are accessed using names, memory is accessed using addresses.
- **File Access:** Sequential access • Read all bytes/records from the beginning • Cannot jump around, may rewind or back up • Convenient when medium was magnetic tape Random access • Bytes/records read in any order • Essential for data base systems • Read can be: • Move file marker (seek), then read or ... • Read and then move file marker

- **File Operations:** Create • Delete • Open • Close • Read • Write • Append • Seek • Get attributes • Set Attributes • Rename
- **Directories:** Naming is better than using numbers, but still limited. • Humans like to group things together for convenience. • We use hierarchy to manage complexity. • File systems allow this to be done with directories • You may call them folders. • Grouping makes it easier to: • Find files in the first place: remember the enclosing directories for the file, • Locate related files, • Determine which files are related.

Debugging (L22):

-