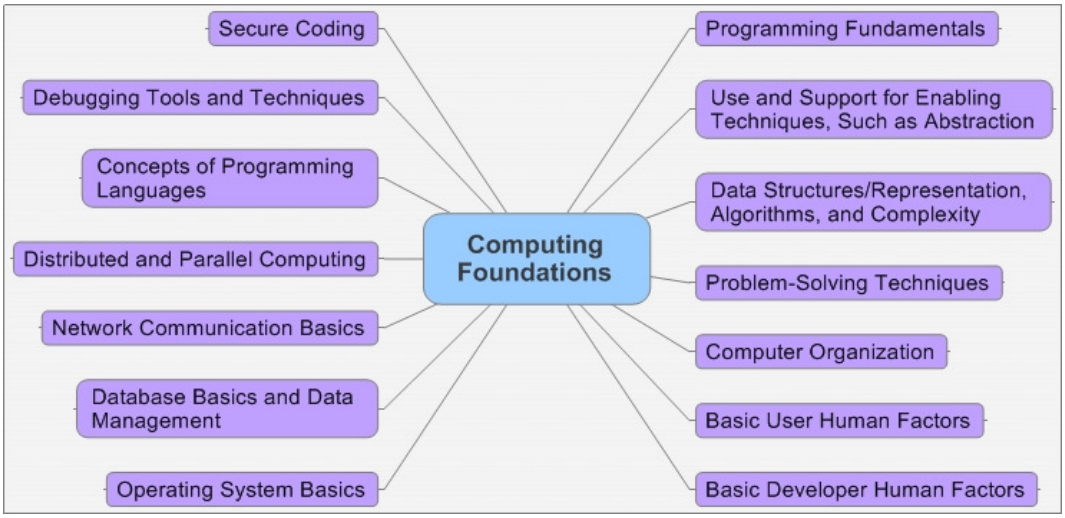
Module – 4 Software Engineering Foundations

Chapter - XIII

Computing Foundations

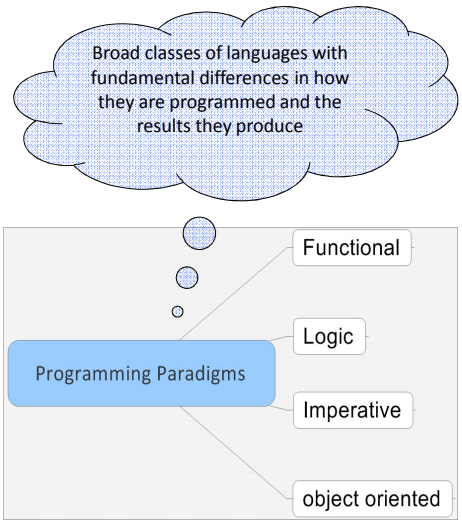
Overview of Topics



Content Area 1

Programming Fundamentals

Programming Paradigms



- **Functional.** Functional languages view programs as an entity—called a function—that accepts inputs and produces outputs. E.g. Scheme, LISP
- **Logic.** Rule-based languages. E.g. Prolog
- **Imperative.** Imperative languages make a set of commands that produce a result by manipulating data. E.g. C, Fortran, Pascal
- **Object-oriented.** OO languages allow user to express a program in terms of objects and messages between those objects. E.g. Java, Smalltalk
- **Scripting.** Scripting languages are more related by their function than by a common language design. E.g. Perl, Ruby, PHP

Ref 4-2

Pseudocode

- For low-level program design
  - Write logic in pseudo code first
    - then convert into source code in a programming language
    - can also become code comments
- Figure shown: Euclidean algorithm
  - for Greatest Common Divisor of two numbers x and y

```
{Precondition: x, y ∈ Z+}
Procedure gcd(x, y)
a ← x
b ← y
while b ≠ 0
    |  r ← a mod b
    |  a ← b
    |  b ← r
{Postcondition: a = gcd(x, y)}
```

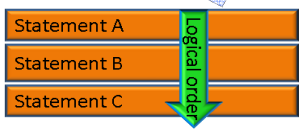
Pseudocode | Programming language constructs + natural language | e.g., **name ← expression**

Ref 4-3

Structured Programming

- "Single entry, single exit" control blocks
  - Recommended by Dijkstra for "structured" programming
  - No need to use *goto*'s
- Böhm and Jacopini's structure theorem
  - *Any* control flow can be implemented using *sequence*, *selection* and *iteration.*

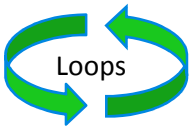
Sequence: Executing statements in logical order



Selection: conditional execution of statements

```
if (alarm active)
    then if (authorized user)
            then allow access
            else sound alarm
```

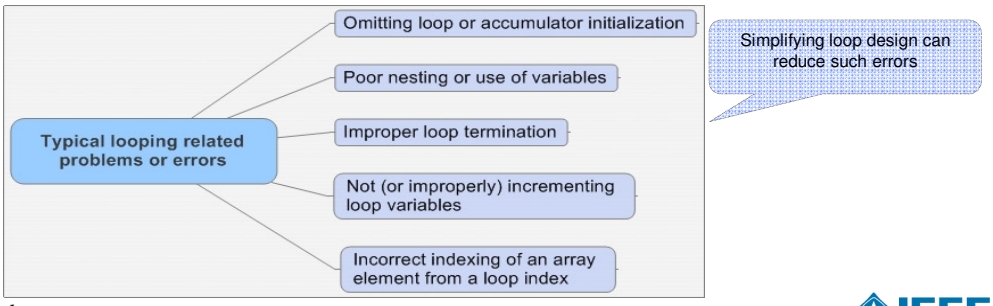
Iteration: executing a series of statements as many times as needed



Ref 4-4

Iteration

- Iteration is also called looping
  - Kinds of loops:
    - *Counted loops* execute only a finite number of times
    - *Continuously evaluated loops* only end when a condition is met
    - *Iterator loops* cycle once for every element in a container class
    - *Endless loops* exist in embedded systems such as pacemakers



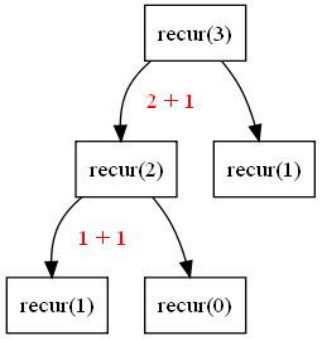
Ref 4-5

Recursion

- Routine calls itself
  - Usually with different set of parameters
  - Must have a "base case" and must logically progress towards that "base case"
  - Alternative to iteration

```
int recur(int n) {
    if(n > 1)
        return recur(n - 1) + recur(n - 2);
    else
        return 1;
}
```

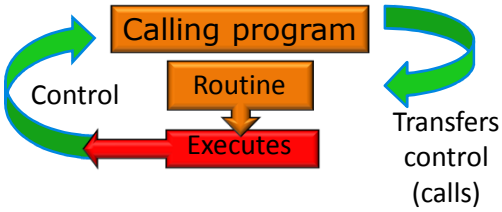
Trace the recursive calls for recur(3);



Ref 4-5

Routines

- Terms vary: Routine, subprogram, subroutine, procedure, algorithm, method
  - Reduce complexity
  - Should have a name that indicates a single, well defined purpose
- Parameters
  - Formal
  - Actual
- Calling parameters
  - Passed by value
  - Passed by reference



Discussion Question

Consider a recursive function "recurr" that calculates and returns a value based on input n, a positive integer.

```
function recurr(n)
    // purpose: ?
    // input: a positive integer n
    // output: a number x equal to the value of recurr(n)
        x := 1
        if n > 10 , then    x := 2*recurr(n-10)
        if n < 10, then     x := x*3
        return x
```

What is the value of recurr(55)?

A. 165
B. 96
C. 1
D. 48

Answer to Discussion Question

- Answer: B

x = 2*recurr(45)
= 2*2*recur(35)
= 2*2*2*recur(25)
= 2*2*2*2*recur(15)
= 2*2*2*2*2*recur(5)
= 2*2*2*2*2*3
= 32*3
= 96

- **Reference:** Applications Programming in C++ by Johsonbaugh and Kalin.
- **Question Source: CSDA sample questions -** http://www.computer.org/portal/web/csda/sampletest

Error Handling

- Types of errors
  - **Internal errors**: errors in the software that can cause inconsistent operations or failure to execute properly
  - **Exceptions**: "events that cause suspension of normal program execution" [*ISO/IEC Std. 24765*]
    - **Passive exceptions**: Program-generated exceptions such as run-time exceptions
    - **Active exceptions**: Software engineer-coded exceptions to handle validations (of inputs) and expected data errors

Plan for error handling consistently throughout the software system by designing in error handling at the architecture level

Ensure errors are reported in user-friendly terminology that helps users make a constructive response to the error

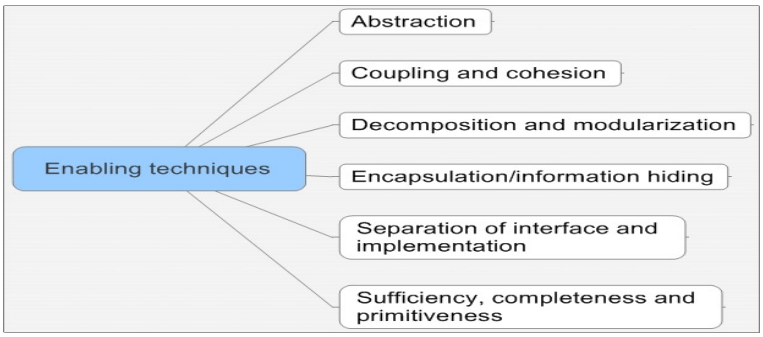
Ref 4-8

Content Area 2

Use and support of enabling techniques, such as abstraction

Enabling Techniques

- Software design principles
  - Key notions considered fundamental to many different software design approaches and concepts. They are also called enabling techniques. [ref: swebok]

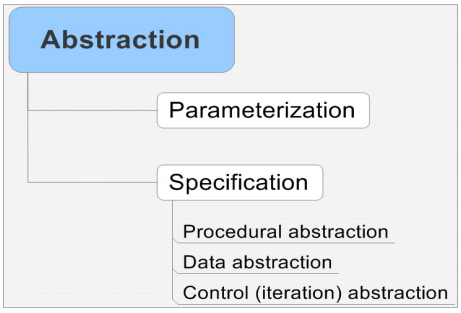


Ref 4-8

Abstraction

- A view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information.
  - Fundamental technique used to cope with complexity.

- Two key abstraction mechanisms
  - *Abstraction by parameterization*: Abstract data/types to use procedures in more situations
  - *Abstraction by specification*: Simply stated, "separate what from the how". Three types:
    - Procedural abstraction
    - Data abstraction
    - Control (iteration) abstraction

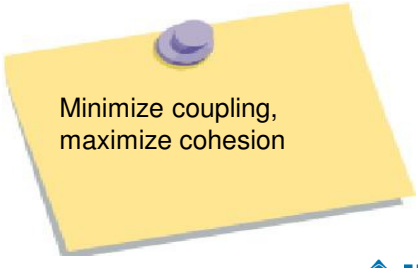


Coupling and Cohesion

- Coupling
  - Manner and degree of interdependence *between* software modules

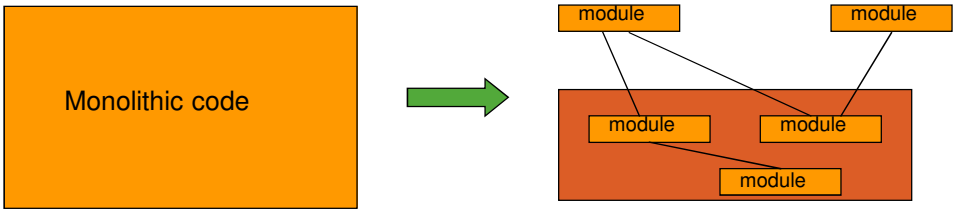
- Cohesion
  - The manner and degree to which the tasks performed by a single software module are related to one another (i.e., *within* a software module)

- We need to balance these two opposing forces while creating modules



Decomposition and Modularization

- Decomposing and modularizing large software into a number of smaller independent ones
  - usually with the goal of placing different functionalities or responsibilities in different components
- Decomposition results in modularization



Ref 4-12

… Enabling Techniques

- Encapsulation/information hiding
  - grouping & packaging elements and internal details of an abstraction and making those details inaccessible
- Separating interface and implementation
  - defining a component by specifying a *public interface* known to the clients,
  - and separate it from the details of how the component is realized
- Sufficiency, completeness & primitiveness
  - A software component should capture all the important characteristics of an abstraction, and nothing more

Parnas: "a programmer is most effective if shielded from, rather than exposed to, the details of systems parts other than his own"

Einstein: "…as simple as possible, but no simpler"

Ref 4-12

Discussion Question

Which of the following is true for software design (while creating modules)?

A. Minimize coupling, minimize cohesion
B. Minimize coupling, maximize cohesion
C. Maximize coupling, minimize cohesion
D. Maximize coupling, maximize cohesion

Answer: B. Minimize coupling, maximize cohesion

Our design goal is to minimize coupling between modules and maximize cohesion of each module

Generics/templates & Polymorphism

- Parametric polymorphism
  - Generics (in Ada/Java/C#) or templates (in C++) are compile-time adaptations of an abstraction
  - We can declare a new data type by filling in information (parameters) for a template
  - This might be constants or type information
- An example: Stack
  - Assume we have a Stack template with DataType parameter
  - We can instantiate Stack by substituting DataType with types
    - Stack of real, Stack of int etc.

Runtime Polymorphism

- OO languages support runtime polymorphism
  - particular method to call/invoke is determined at run time based on the runtime-type of the object on which the method is called
- Runtime polymorphism is useful
  - Allows instances of objects to have different properties at different types
  - allows some specialization in features declared in objects that are otherwise programmed as generically as possible

Ref 4-13

Discussion Question

"Report" class inherits from "Document" class. Document class defines a print() method. Assume polymorphic behavior. Which of the following scenarios would not result in the client calling Document.print() on a Report object called TermReport?

A. Report.print() does not refer to Document.print()
B. Report class does not define print() method.
C. Report.print() calls Document.print() and does nothing else
D. Client casts the TermReport object to Document class before calling print.

Answer to Discussion Question

- Answer: A

  By the usual understanding of polymorphism in OO inheritance, for answers b, c and d, Document.print() would be called; and for answer a, Document.print() would not be called

- **Reference:** Deitel and Deitel, C++: How to Program, 5th Ed. , Prentice Hall, 2005.
- **Question Source: CSDA sample questions -** http://www.computer.org/portal/web/csda/sampletest

Content Area 3

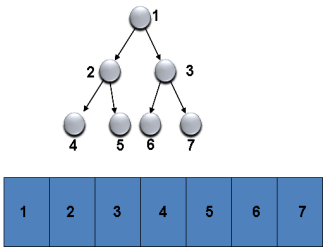
Algorithms, Data Structures/Representation, Complexity

Abstract Data Type (ADT)

- Definition
  - "a data type for which only the properties of the data and the operations to be performed on the data are specified, without concern for how the data will be represented or how the operations will be implemented" [ISO/IEC Std 24765]
- Characteristics
  - A predetermined storage system for the data
  - A set of possible operations
- Examples:
  - Lists, Stacks, Queues, Trees, Graphs etc.
  - ADTs can be built to represent files, menus etc.

Data Structures

- Data structure: "structuring data in memory"
- Same memory representation for different data structures
  - E.g. linear array and full binary tree (see fig)
  - The root node is in index 1.
  - For node with index i, 1 < i < n (for some n):
    - Parent(i) = i/2
    - LeftChild(i) = 2 * i
    - RightChild(i) = 2 * i + 1



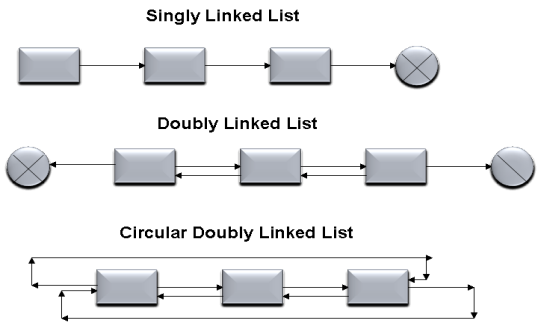
Static Arrays vs. Dynamic Arrays

- Static arrays have fixed size
  - Size known at compile-time and will not change
- Dynamic arrays grow and shrink as needed
  - The size changes at runtime as the data is added and removed

Data Type	Advantages	Disadvantages
Static Arrays (Bounded)	True random access, not just sequential. Uses little memory and data cache. Excellent locality of reference for faster iteration and indexed searches (heapsort).	Usually created by making an array twice as large as needed to reserve capacity (wastes space). If resizing is needed, must copy and replace old array (expensive).
Dynamic Arrays (Unbounded)	All of the advantages of static arrays. Automatically resize. Faster indexing than lists. Low fixed overhead (size and capacity) so good when cache size is vital.	Usually doubles size to prevent frequent resizing (wastes space). Requires time to move all list elements when inserted at an arbitrary location.

Linked Lists

- Node connected to adjacent node(s) through pointer(s)
  - Sequential access (no direct access to nodes)
  - Common types:
    - Singly linked list
    - Doubly linked list
    - Circular linked list
- Fast insertion and deletion
  - But access is slow



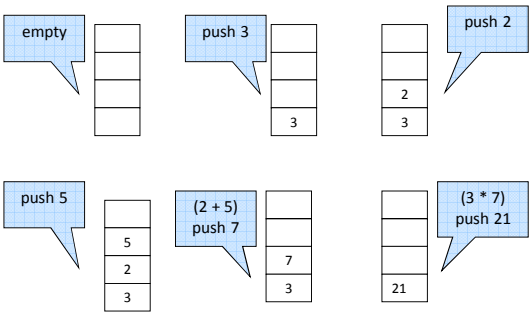
Linked Lists…

Data Type	Advantages	Disadvantages
Singly-Linked Lists	Simplest list type. Can add or remove items from any location in constant time. (Arrays require linear time.) Do not require a large contiguous block of memory; can be stored in many smaller blocks.	Only one direction, sequential search. Inserting and deleting requires knowing list node and prior list node addresses. Linked list of integers is most often needed for algorithms, especially on graphs.
Doubly-Linked Lists	Bidirectional search makes manipulation easier; some algorithms require it. Can insert or delete and only need current list node address.	Only sequential search. Uses more space per list node. Searching, inserting, and deleting require more processing.
Circularly-Linked Lists	Good for structures that are circular by nature, such as buffers, and for lists that require viewing all other items in the list given any item from the list. Can quickly get to the first or last node.	Iteration is complex and can be error prone if not carefully designed.

Ref 4-19

Stack and Queue

- Stack
  - Can add/remove elements only from top (LIFO)
  - E.g., 'Stack frames' when methods execute, pile of dishes
- Queue
  - Can add in tail and remove from head (FIFO)
  - E.g., A keyboard buffer in hardware, billing line in super market
- E.g., Expression evaluation using stack
  - (3 * (2 + 5)) – in infix
  - 3 2 5 + * – in postfix



Ref 4-19

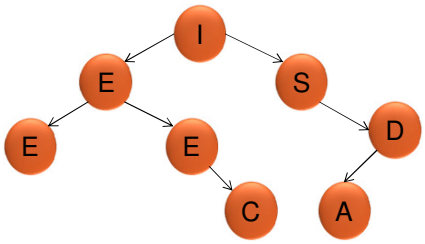
Priority Queue

- A priority queue is a queue that ranks its items
- It doesn't obey FIFO order
  - It adds and deletes items in the queue by a priority key set at the time of insert, instead of by their arrival time
  - In other words, highest "priority" items are retrieved first
- Useful data structure; for example:
  - For prioritizing by smallest or largest element, such as for cost versus profit
  - Used in algorithms, such as heap sort, greedy algorithm, or Dijkstra's shortest-path algorithm

Ref 4-20

Trees

- Hierarchical data structures
- Binary tree (see figure)
  - Each node contains two links (none, one or both might be NULL)
- Tree traversal
  - Depth-first traversal
    - Pre-order sequence (root, left, right)
    - In-order sequence (left, root, right)
    - Post-order sequence (left, right, root)
  - Breadth-first traversal
    - Level order sequence (visit every node in a level before going to a lower level)



- Pre-order: IEEECSDA
- In-order: EEECISAD
- Post-order: ECEEADSI
- Level-order: IESEEDCA

Discussion Question

Assume that you're implementing iterative (i.e. non-recursive) versions of traversing a binary tree. Which one of the following traversal implementations will need a queue data structure (FIFO)?

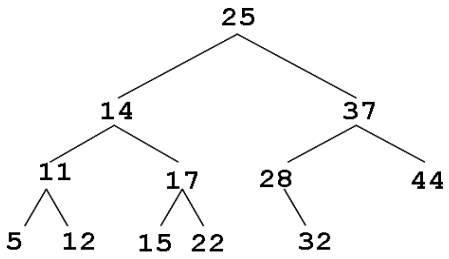
A. Pre-order traversal
B. In-order traversal
C. Post-order traversal
D. Level-order traversal

Answer: D. Level-order traversal

A, B and C are depth-first traversals => we need stack for implementation
D is breadth-first traversal => we need queue for implementation

Binary Search Tree

- Binary search tree (see fig)
  - Ordered binary tree
  - Left sub-tree of a node contains keys less than the node's key
  - Right sub-tree of a node contains keys greater than or equal to node's key
  - Both right and left sub-trees are binary-search trees
- Search is typically fast:
  - It requires O(log N) in the average case (for a balanced tree)
  - It takes O(N) in worst case (an unbalanced – degenerate – tree looks like a linked list)



Big-oh Notation

Big-oh: Let f and g be real-valued functions defined on nonnegative real numbers. F(n) is O(g(n)) if there exists positive numbers C and k such that $|f(n)| \leq C|g(n)|$ for all $n \geq k$.

Determine if $9n^2 + 5n + 3$ is $O(n^2)$.

Answer:

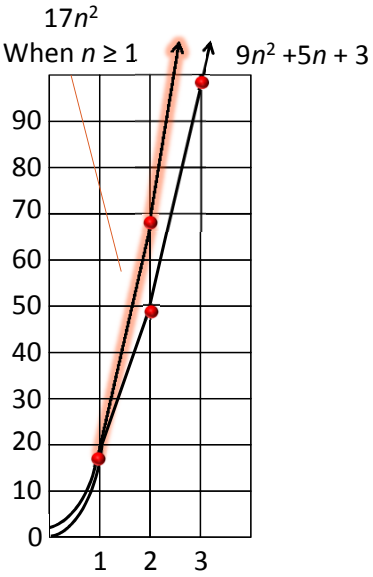
Start with $k = 1$.

When $n \geq 1$, $n < n^2$ $1 < n^2$:

$0 \leq 9n^2 + 5n + 3 \leq 9n^2 + 5n^2 + 3n^2$

$= 17n^2$

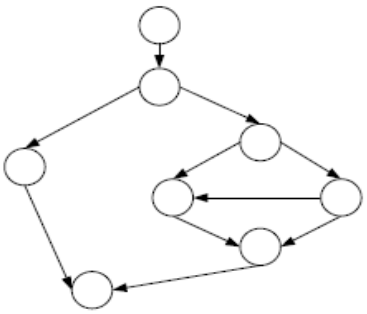
$C(g(n))$ can use $C = 17$ and $k = 1$.



Ref 4-28

Complexity of Graphs

- Cyclomatic number[mccabe]
  - *V(G)* of a graph *G* with *n* vertices, *e* edges, and *p* connected components is: v(G) = e - n + p
  - When we compute this number on the control-flow graph of a function, we get the cyclomatic complexity of that function
  - In this example, we have eight nodes (n = 8) and 10 edges (e = 10); and in a control-flow graph, we have one edge connecting another (p = 2).



[mccabe] Thomas J. McCabe. A complexity measure, ICSE, 1976.

$V(G)$ = Number of Edges – Number of Nodes + 2

$= 10 - 8 + 2 = 4$

Ref 4-28

Discussion Question

Which of the following data structures is used in algorithms such as heapsort, greedy algorithm and Dijkstra's shortest-path algorithm?

A. Deque
B. Stack
C. Queue
D. Priority queue

Answer: D. Priority queue

Ref 4-20

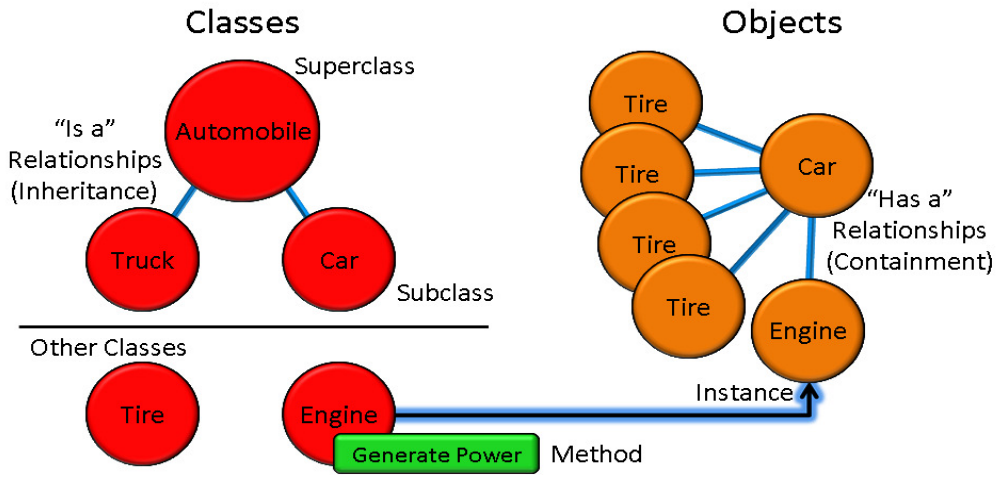
Object Oriented Programming

- *Class*: "An abstraction of the knowledge and behavior of a set of similar things" [*IEEE Std. 1320.2-1998*]
  - Classes help in data abstraction
- *Object*: An instantiation of a class
- *Inheritance ('is-a' relationship)*:
  - When certain broad characteristics are defined in a super-class, its child classes gain or inherit those properties
- *Containment* (*'has-a' relationship*):
  - an object has instances of other objects as subcomponents

Ref 4-29

Inheritance vs. Containment

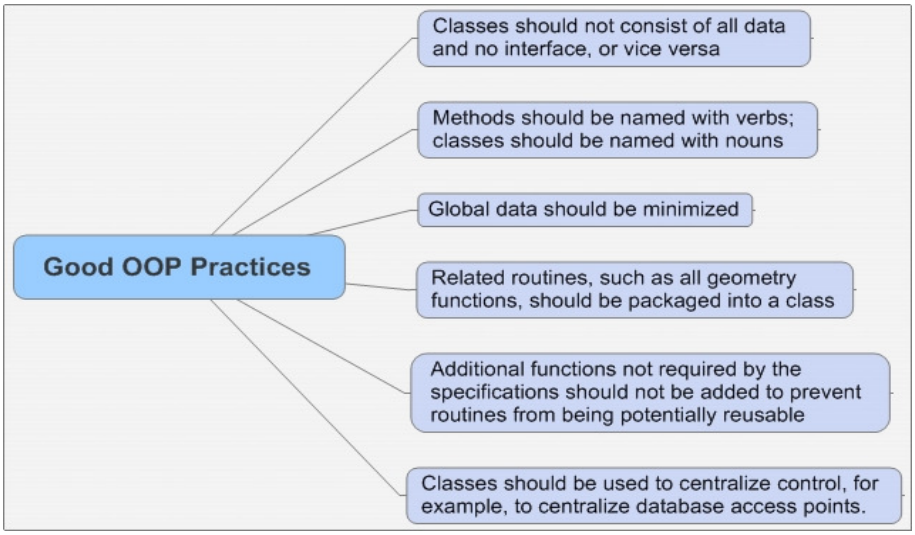
Content Area 3: Algorithms, Data Structures



Ref 4-22

Practical considerations in OOP

Content Area 3: Algorithms, Data Structures



Ref 4-22

Content Area 4

Problem-Solving Techniques

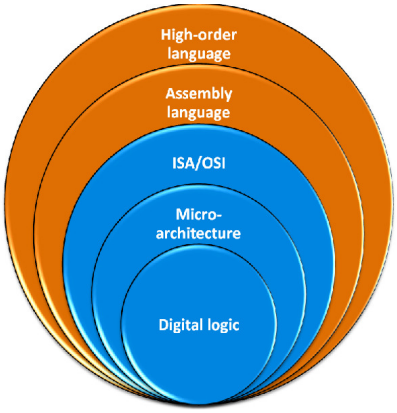
(Not covered in the reading material and in this presentation)

Content Area 5

Computer Organization

Computers: Levels of Abstraction

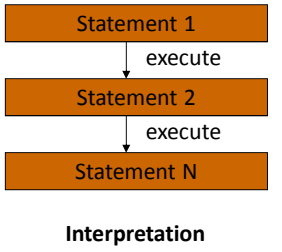
- We can view computers at different levels of abstractions
- Lower levels of abstraction
  - *Digital logic*. Uses hardware to control physical on/off gates.
  - *Micro-architecture*. Implements the Instruction Set Architecture (ISA).
  - *ISA/OSI*. It's a hybrid layer.
    - Instruction Set Architecture has hardware specific instructions.
    - The Operating System Interface (OSI) layer handles commands and is a level above the hardware.



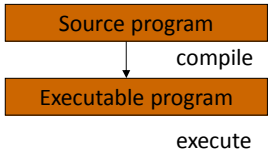
Ref 4-29

Interpretation vs. Compilation

- Basic ways in which human-readable code can be converted into a form that machine can understand and execute
  - *Interpretation*: To translate and execute each statement or construct of a computer program before translating and executing the next
  - *Compilation*: To translate a computer program expressed in a high-order language into its machine language equivalent.



Interpretation

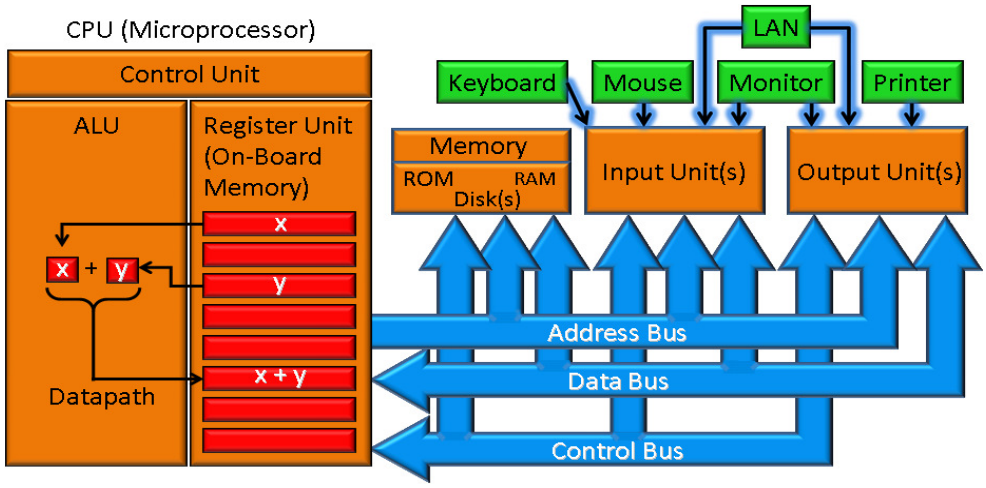


execute

Compilation

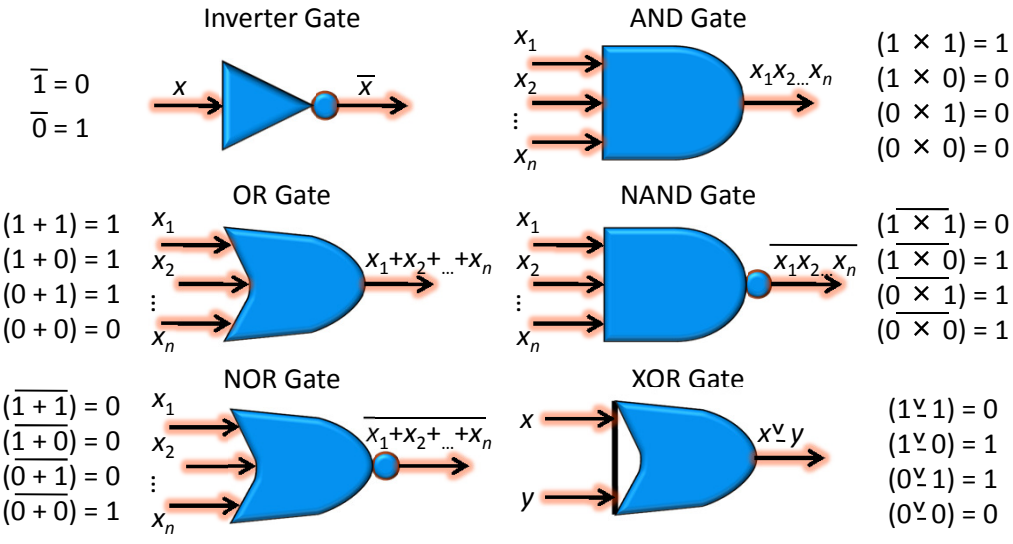
Ref 4-30

Basic Computer Hardware



Ref 4-31

Logic Gates

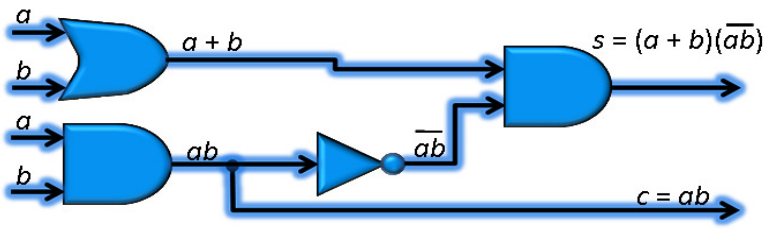


Ref 4-34

Digital Logic: Half Adder

- Two types of adder circuits: half-adder & full-adder
  - Half-adder adds two bits and produces a sum and a carry

Half Adder



Input		Output	
a	b	s	c
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

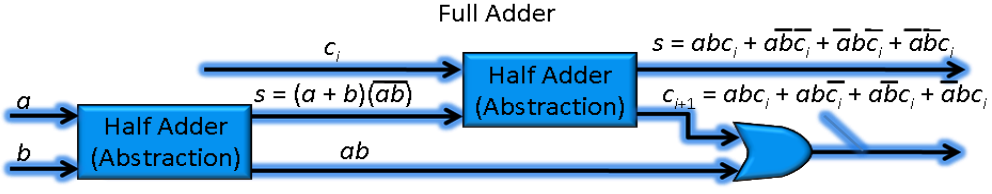
Ref 4-35

Digital Logic: Full Adder

- Like a half-adder, a full-adder adds two bits and also a carry bit c_i; it produces a sum and a new carry bit c_{i+1}

Input			Output	
a	b	c_i	s	c_{i+1}
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

Full Adder



Ref 4-35

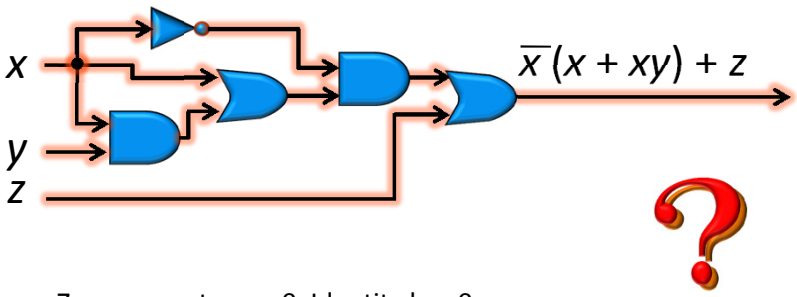
Boolean Algebra Identities

De Morgan's Laws	Law of Double Complement
• $\overline{(xy)} = \overline{x} + \overline{y}$ • $\overline{(x+y)} = \overline{x}\,\overline{y}$	• $\overline{\overline{x}} = x$
Identity Laws	**Domination Laws**
• $x + 0 = x$ • $x1 = x$	• $x + 1 = 1$ • $x0 = 0$
Idempotent Laws	**Commutative Laws**
• $x + x = x$ • $xx = x$	• $x + y = y + x$ • $xy = yx$
Associative Laws	**Distributive Laws**
• $(x + y) + z = x + (y + z)$ • $(xy)z = x(yz)$	• $x + yz = (x + y)(x + z)$ • $x(y + z) = xy + xz$
Absorption Laws	**Unit and Zero Property**
• $x + xy = x$ • $x(x + y) = x$	• **Unit property:** $x + \overline{x} = 1$ • **Zero property:** $x\overline{x} = 0$

Discussion Question

Which of the following shows a minimization of the following circuit? (Hint: use Boolean algebra identities on previous slide)

A. z
B. xyz
C. xz
D. xy + z

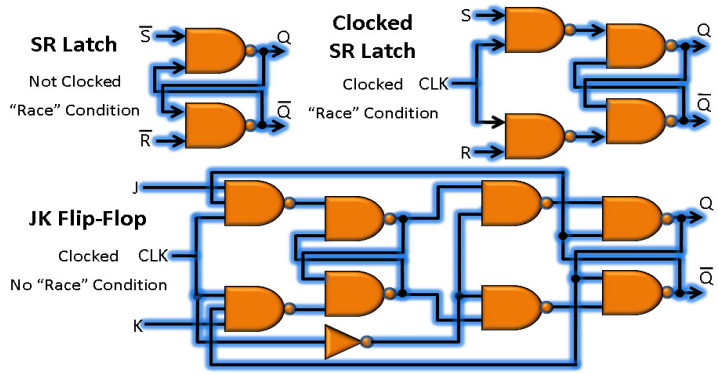


Answer: A. z

Absorption law $x + xy = x$. Zero property $xx = 0$. Identity law $0 + z = z$.

Memory Systems

- Memory circuits produce a constant output based on prior inputs
  - SR-Latches and flip-flops are examples of 1-bit memory



Ref 4-38

Numeric Data Representation

- 2's complement representation
  - Most popular method for representing integers
  - Left-most bit is sign-bit (0 means positive; 1 means negative)
  - All zeros represents integer value zero
  - All ones represents integer value -1
  - From zero, incrementing by 1 results in successive positive integers
    - Till the value 0 followed by all 1's is reached
  - From -1, we get successive negative integers by counting backwards
    - Till the value 1 followed by all 0's is reached

Bit pattern for 3-bit integers in 2's complement representation

Binary	Decimal
011	3
010	2
001	1
000	0
111	-1
110	-2
101	-3
100	-4

Ref 4-37

Discussion Question

What is the binary equivalent (i.e., bit pattern) of the decimal value -1 in a 8-bit signed integer (in 2's complement representation)?

A. 1000 0001
B. 1111 1111
C. 0111 1111
D. 1111 1110

Answer: B. 1111 1111

Ref 4-37

Discussion Question

What is the decimal equivalent of the binary value 10101.0101?

A. 42.3125
B. 31.05
C. 21.3125
D. 31.20

10101.01010

16 8 4 2 1 ½ ¼ ⅛ 1/16 1/32

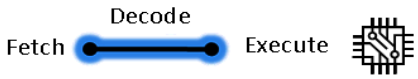
$= (21.3125)_{10}$

Answer: C. 21.3125

Ref 4-37

Machine Instructions

- Machine language
  - "a language that can be recognized by the processing unit of a computer" [*ISO/IEC Std. 24765]*
- Types of instruction sets
  - Reduced Instruction Set Computing (RISC)
    - optimizes the instruction set to make the system fast and efficient
  - Complex Instruction Set Computing (CISC)
    - is easier to program
- Machine cycle
  - Also called "fetch-decode-execute" cycle



Communication Errors

- Errors can occur when transferring or storing data
- Error detection
  - Parity bit
    - extra bit added to each string to detect single bit error in data
  - Cyclic Redundancy Check (CRC)
    - CRC & checksums use 'check-byte' to detect errors
- Error correcting codes (codes that repair detected errors)
  - E.g. Hamming code; analyzes a bit pattern used for encoding symbols
  - Will repair any illegal bit pattern by replacing it with the legal pattern

Content Area 6

Basic User Human Factors

Visual Design – Colors

- Rules of thumb for visual design of a user interface
  - Limit the number of colors used
    - one guideline: don't use more than 5 colors
  - Use color change to show change in system status
  - Use color coding to support the tasks users are trying to perform
  - Use color in thoughtful and consistent way
  - Be careful about color pairing
    - People cannot focus on red and blue simultaneously
    - Many people are red-green color blind.

Ref 4-43

Visual Design – Fonts

- Rules of thumb for visual design of a user interface
  - Do not use more than four different font sizes per screen.
  - Use serif or `sans serif` fonts appropriately as the visual task demands.
  - DO NOT USE ALL UPPERCASE LETTERS
    - Use a mixture of uppercase and lowercase letters.
  - Use underlining, **bold**, , or other markers sparingly.
  - On text screens, *don't* use i inverse video fonts on a single screen.

Visual Design – Graphics

- Rules of thumb for visual design of a user interface
  - Graphics, especially animation, can be helpful to the user.
  - Do not use a large graphic as a background; it makes text harder to read and visual cues harder to see.
  - Use standard graphical objects (buttons, check boxes) when possible.
    - Familiar is better than fancier; for example, "click here" might not be recognized.

Discussion Question

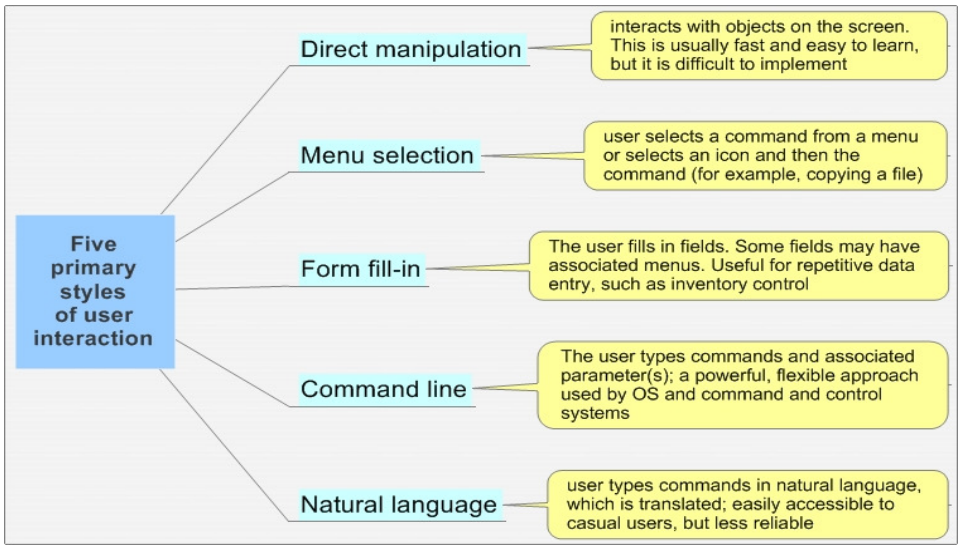
Which of the following is *false* of basic user human factors to consider when constructing software?

A. Feedback messages should be the same for all types of users
B. Use color change to show change in system status
C. Use underlining, bold, inverse video, or other markers sparingly
D. Use standard graphical objects (buttons, check boxes) when possible

Ans: A. Feedback messages should be the same for all types of users

Ref 4-43

User Interaction

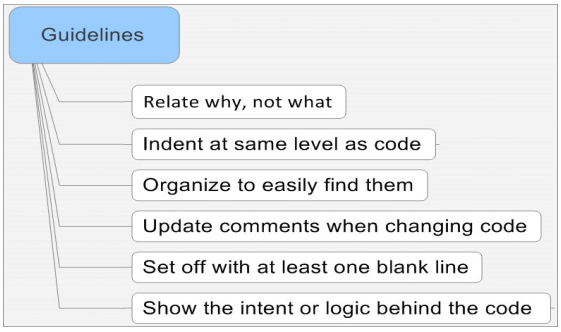


Ref 4-44

Content Area 7

Basic Developer Human Factors

Comments



- Automated comments
  - If properly commented in a specific format, we can extract documentation from comments itself.
  - Example: Creating API documentation using javadoc from Java code

Ref 4-45

Readability

- Style and layout ease review and have an impact on quality
  - does not affect speed or memory usage, but still important
- Style – making systematic & deliberate choices
- Layout – indicate logical organization of code.
  - Consistent formatting choices
  - Indent control structures to indicate flow of control
  - Use blank lines and blank spaces as visual aids

```
if (alarm active)
     then (if (authorized user = true)
            then (allow access)
            else (sound alarm)
    )
```

good

bad

```
if (alarm active) then (if (authorized user = true)
then (allow access)
else (sound alarm))
```


Complexity

- Complex source code is prone to bugs
  - Some code is necessarily complex; some code can be simplified
  - Projects must have standards to keep components within size limits
    - Use techniques to make complex code simple

Using large number (say 25) of logical checks like this, is bad

```
function message(errornumber)
if  errornumber == 1
     return("File not found.");
elseif errornumber  == 2
     return("Invalid entry.");
elseif errornumber == 3
     return("Insufficient data.");
end
```

Using table driven methods can simplify code in this case

```
Function message(errornumber)
Msgarray[100] = "File not found.", "Invalid entry.", "Insufficient data." …
return msgarray(errornumber)
```

Content Area 8

Operating System Basics

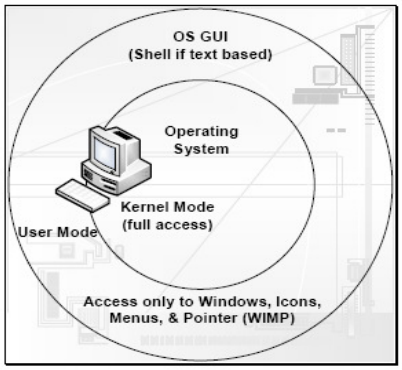
What is an OS?

- A definition (ISO/IEC 24765):
  - "a collection of software, firmware, and hardware elements that controls the execution of computer programs and provides such services as computer resource allocation, job control, input/output control, and file management in a computer system"
- Functions
  - Controls computer's hardware resources such as memory, processor etc.
  - Provides a level of abstract to hide hardware level details from users
  - Also a resource manager – allocates and balances requests for processors, memory, I/O devices etc.

Ref 4-49

Two modes in an OS

- OS GUI (shell):
  - Is called the user interface program
  - Is frequently mistaken for OS itself!
  - Runs in user mode
    - allows only a subset of instruction set
    - excludes all machine & I/O controls
- OS itself:
  - Resides at much lower level than GUI
  - Runs in kernel mode
    - can execute all instructions in the machine's instruction set
    - full access to all computer hardware

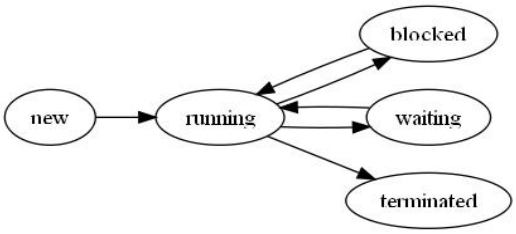


  Ref 4-50

Program Execution

- A process
  - is a program in execution
  - can be in many states
    - Running, waiting, blocked etc.

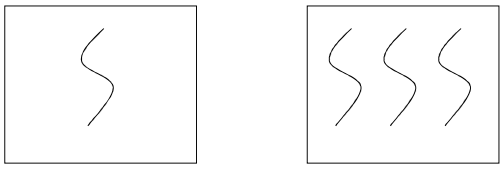
  (see fig)



- Simultaneous execution
  - Two or more processes can be run on a single-CPU computer
  - One process must be suspended for other to resume
  - OS remembers the "context" (program's execution state) to enable this feature

  Ref 4-50

Concurrency



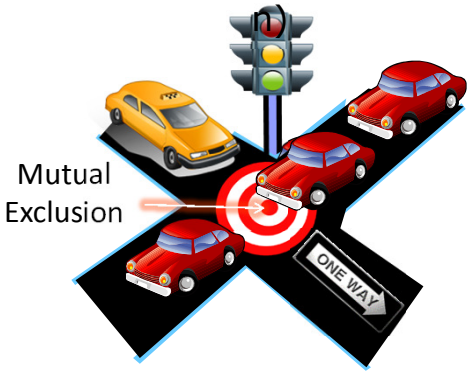
- Concurrency
  - allows for multiple threads to share a common address space and its associated components, such as file descriptors or environment variables.
- Single- vs. multi-CPU systems
  - In single-CPU systems, concurrency is simulated by OS
  - On multi-CPU systems, true concurrency since many tasks can execute at the same time
- Advantages and problems
  - Concurrency increases the computer efficiency and throughput
  - But creates a set of problem areas, like 'data races' and 'dead locks'
    - 'data race': Unsynchronized read/ write to a location by many threads
    - 'dead locks': Covered in next few slides

Ref 4-51

Critical sections and semaphores

- Critical section
  - An area of the program where multiple threads can share the same resource
- Semaphore
  - What if resources can't be used simultaneously, such as a printer?
  - Semaphore can be used to regulate a critical section

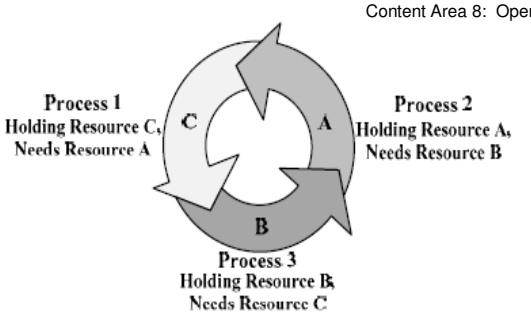
Semaphore (Synchronizatio

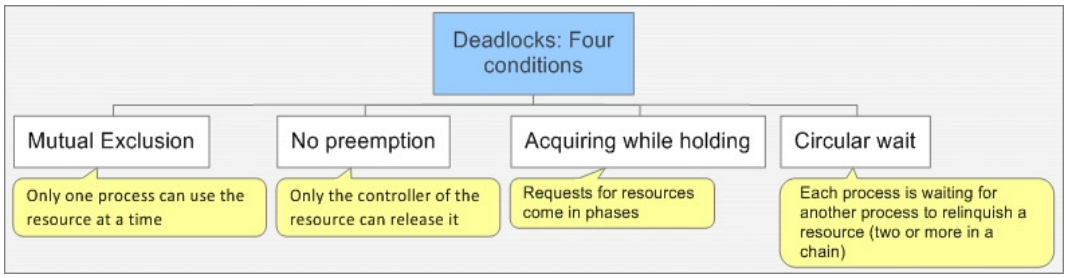


Ref 4-52

Deadlocks

- Tasks are suspended indefinitely because each task is waiting for other to complete





Discussion Question

Which of the following is *not* a necessary condition for deadlocks?

A. Mutual exclusion
B. Acquiring while holding
C. Circular wait
D. Non-repudiation

Ans: D. Non-repudiation

Content Area 9

Database Basics and Data Management

Database Systems

- A database presents a multidimensional view of data
  - Unlike data stored in files (known as 'flat files') that present data from one view point
- What is a DBMS (DataBase Management System)?
  - Software that organizes the data in the database into a conceptual model and performs all actual manipulations of the data

DBMS: Some notes

- Handles database instructions so applications can issue generic commands
- Has full access to the schema so it can perform these tasks
- Contains a number of algorithms to manipulate the data, and the routines of applications essentially use these as subroutines
- Ensures data consistency

Ref 4-56

Discussion Question

Which of the following is MOST LIKELY true of databases and database management systems (DBMS)?

A. The routines of applications use DBMS algorithms as subroutines to manipulate data

B. Applications using a DBMS are responsible for setting access privileges to data subsets

C. A database and its DBMS must have tight coupling

D. Only the OS has full access to the schema, not the DBMS

Ans: A. The routines of applications use DBMS algorithms as subroutines to manipulate data

Relational database: An example

- How the DBMS creates a new relation for a customer's shopping cart in a book sale website
  - DBMS operation: Customer_No_22345_Shopping_Cart <- SELECT from BOOK where Book_ISBN = "0-13-600652-9"

Attributes (Columns)

Relation (Table) **Book**

Book_ISBN	Book_Descr	Author	Pub_Date	Amt_In_Stock	List_Price
0-13-600652-9	Operating Systems	Reese, Jim	06/2008	9,002	149.00
0-13-600792-7	Networking	Jones, Ron	07/2008	4,003	125.49
0-13-600802-3	Computer Organization	Clugel, et. al.	07/2009	0	51.25

Tuples (Rows)

Customer_No_22345_Shopping_Cart

Book_ISBN	Book_Descr	Author	List_Price	Qty	Ship_Cost	Order_Total
0-13-600652-9	Operating Systems	Reese, Jim	149.00	2	10.00	308.00
0-14-740438-2	Coding	Fluegel, Alice	155.00	9	45.00	1440.00
0-32-540645-5	Software Engineering	Johnson, Abe	98.00	9	45.00	927.00

Relation (Table) **Customer**

Customer_No	Customer_Name	Customer_Address
22333	Frank Stillworth	234 Rapid Lane, Apple Valley, MN, 55403
22342	John Smith	3234 Leland Avenue, Dallas, TX 33756
22345	Hansen Corp.	P.O. Box 44332, Upper Lake, UT 89703

Object Oriented DBMS

- OODBMS
  - A database that creates classes and objects with links designating their relationships
  - Intends to provide consistency between the language used to create the applications and the databases
- OO applications & OODBMS
  - In applications, objects are destroyed when the application terminates
  - OODMBS provides permanent storage for the object data

Advantages:
OODBMS over RDBMS

- Relational databases have difficulty with mixed structures.
  - E.g., handling first name and last names - OODBMS can treat parts of names as needed
- OODBMS objects have intelligent code.
  - E.g., how to store video or audio content; relational databases don't easily adapt to new attribute types

Ref 4-59

Content Area 10

Network Communication Basics

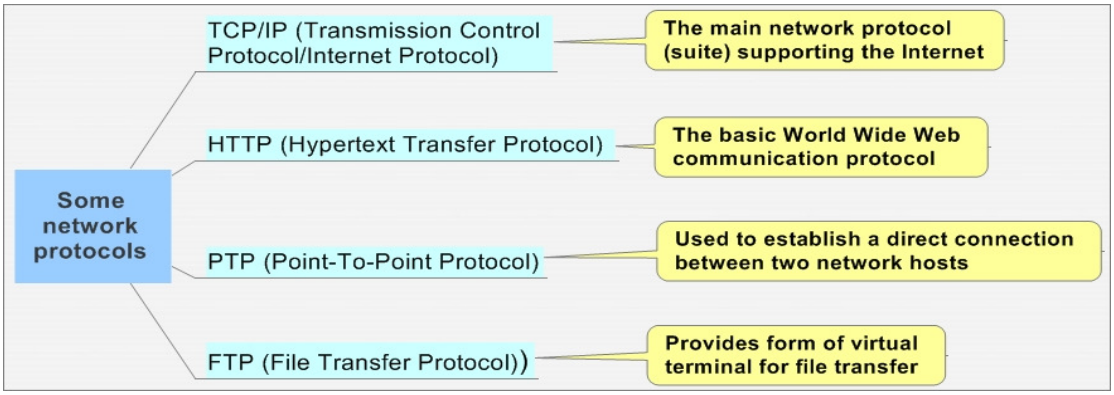
Computer Networks

- Computer network
  - "data processing nodes interconnected for data communication"
  - Useful for information sharing and resource sharing among computers. E.g., using a common printer
  - Don't confuse it with distributed system: It is a collection of computers that act like a single system
- Two common types
  - LAN and WAN
- Internet Service Providers (ISP)
  - Companies providing internet access at different levels (Tier-1, 2, etc.)

Ref 4-61

Network Protocols

- Protocols: Sets of rules that govern communications among devices following the standard



Ref 4-62

TCP/IP and OSI Reference Model

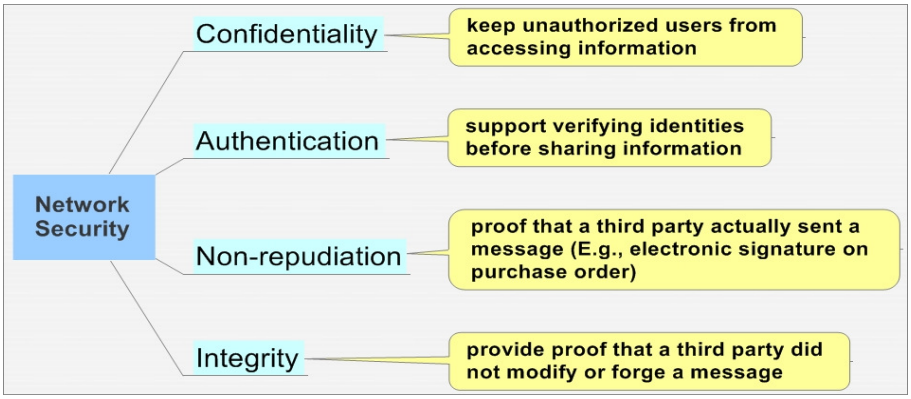
- TCP/IP
  - has a reference model to show layers of abstraction in communications using this protocol
- ISO/OSI
  - a similar reference model having more layers is the OSI (open systems interconnection) reference model
- The figure shows the comparison of layers between the two

TCP/IP	Layer	OSI
Application	7	Application
n/a	6	Presentation
n/a	5	Session
Transport	4	Transport
Internet	3	Network
Host-to-network	2	Data link
	1	Physical

Ref 4-62

Network Security

- Network security must provide confidentiality, authentication, non-repudiation and integrity (see fig).



Ref 4-63

Terminology: Security and Encryption

- Cryptography
  - converting information text into an unintelligible form with the ability to recover information again
- Cryptology
  - the process of inventing encryption algorithms
- Symmetric-key algorithms
  - use a key to encrypt the plain text and the same key to decrypt the cipher-text
    - weak-link is distribution — if the key is compromised, the encryption is compromised

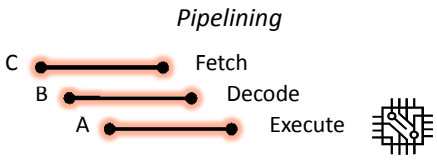
Ref 4-63

Content Area 11

Distributed and Parallel Computing

Pipelining

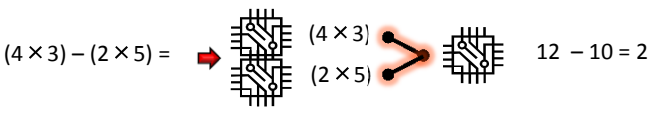
- While one instruction is being executed, one or more subsequent instructions can be fetched, and decoded
  - so they subsequent instructions are ready to execute immediately
  - However, instructions like JUMP can void pipelining (because wrong instructions have got fetched and decoded)
- Pipelining is for single instruction, single data (SISD) stream architecture processors (those with just one processor)
  - For improving throughput



Ref 4-66

Parallel Processing

- Parallel processing advances the concept of pipelining
  - by executing several instructions simultaneously using multiple CPUs
  - multiple-instruction stream, multiple-data stream (MIMD)
- Multiprocessors: Computers with many processing units
  - Processing units are attached to same main memory; processors leave messages in this shared space to coordinate activities



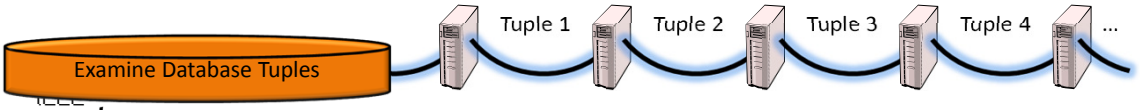
Ref 4-66

Distributed Computing

- Executing process in a single-CPU is sufficient for most (simple) tasks
  - For example, computing first 1000 prime numbers
- For performance, better to run complex tasks in multiple-CPUs
  - For example, a website can run application, web server can be run on a single-CPU machine
  - But it is better to run each of them separately on single-/multi-CPU machine

Distributed Computing …

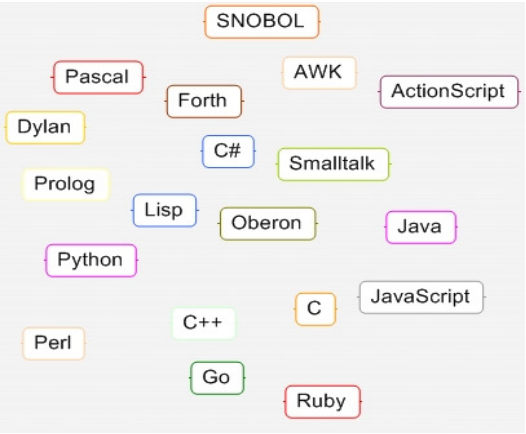
- "The spreading of computation and data across a number of computers connected by a network"
  - single instruction, multiple data (SIMD)
  - links processors in a daisy chain
  - each processor works on portions of a data stream in concert
  - can also take place by chaining together multiple, independent computers
- Distributed computing best used
  - when a repetitive task must be applied to a large data set

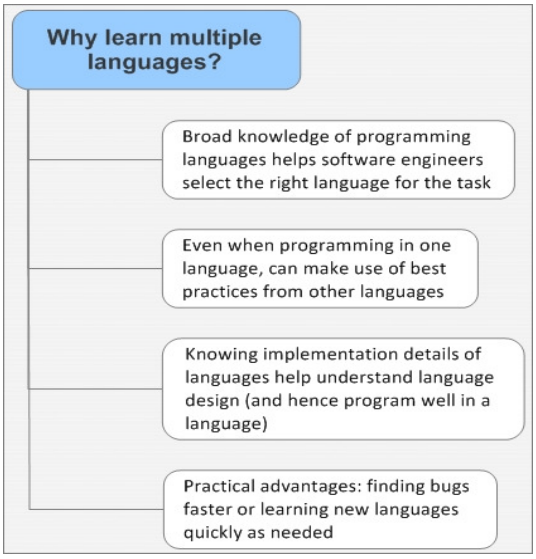


Content Area 12

Concepts of Programming Languages

Why Learn Multiple Languages?





Ref 4-67

Selecting a Programming Language

Criteria	Programming Concept
Readability, constructability, & dependability	▪ Simplicity, reduced by feature multiplicity/operator overloading ▪ Orthogonality ▪ Syntax ▪ Control statements ▪ Data types
Constructability & dependability	▪ Abstraction ▪ Expressivity
Dependability	▪ Readability and constructability have influence on dependability ▪ Exception handling ▪ Type checking ▪ Aliasing ▪ Optimization

Ref 4-68

Types of Languages

Low-Level Programming Languages	Used for…
Assembly language/Assembler—One instruction per statement.	Processor-specific applications where code size or execution speed is vital.
Mid-Level Programming Languages	**Used for…**
C—Systems control flow and low-level constructs.	Updating C software; many operators; uses pointers and addresses; weakly typed.
High-Level Programming Languages	**Used for…**
Fortran—1st high-level language, introduced high-level loops and variables.	Engineering and scientific; many options on arrays and user-defined data types.
High-Level Object-Oriented Programming Languages	**Used for…**
Java—Syntax like C and C++; virtual machine runs byte code from Java source code.	Web applications and systems that must run on any platform.
Scripting Languages	**Used for…**

Ref 4-69

Content Area 13

Debugging Tools and Techniques

(Not covered in the reading material and in this presentation)

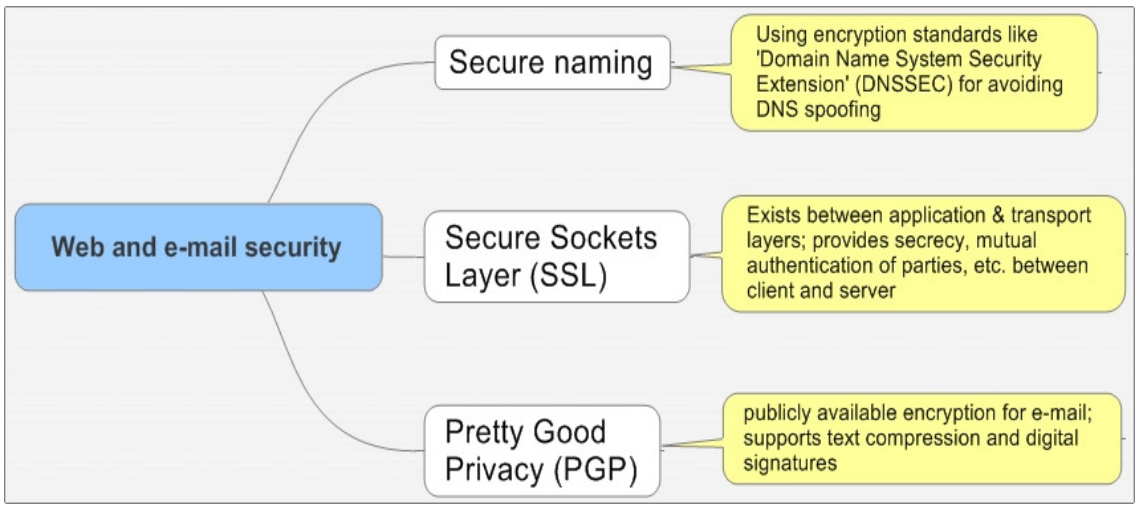
Content Area 14

Secure Coding

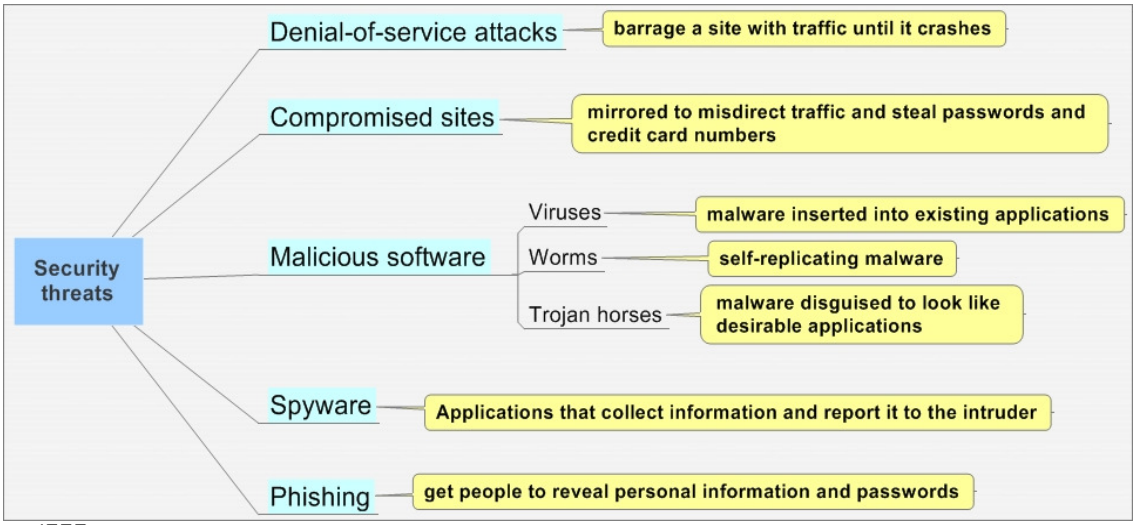
Security Engineering & Secure Coding

- Security Engineering
  - Security cannot be 'added to' systems
  - must be 'designed in' before implementation
- Secure coding
  - Security engineering implementation during s/w construction
- Web security - a vital area for secure coding
  - Secure naming
  - Authenticated & secure connections
  - Special handling of executable programs etc.

Web and E-mail Security



Security Threats



Discussion Question

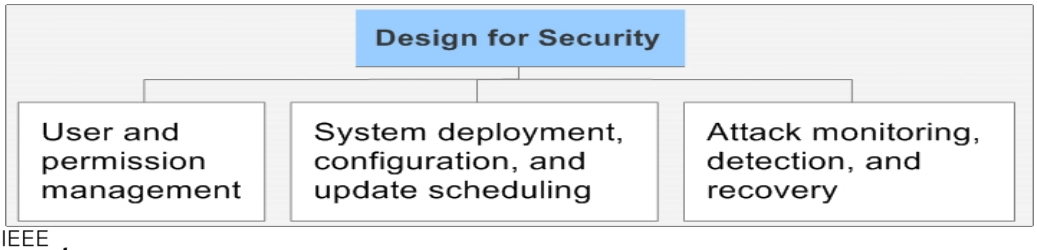
Which of the following describes "malware disguised to look like a desirable application"?

A. Virus
B. Worm
C. Trojan horse
D. Spyware

Ans: C. Trojan horse

Design for Security

- Applications must be designed for security
  - Not just detect/withstand attacks, must also recover from attacks
- Security design should start at architectural design stage
  - Poor architecture can make it impossible to provide confidentiality/system integrity



Discussion Question

An active intruder to a corporate site attempts to use DNS spoofing, creating a poisoned cache. This intruder is trying to create a:

A. Denial-of-service attack.
B. Virus that will infect the rest of the network.
C. Compromised site that misdirects traffic.
D. Security flaw at the architecture level.

Ans: C. Compromised site that misdirects traffic.

Suggested Reading

- *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, 2004 ed., Los Alamitos, California: IEEE Computer Society Press, 2004.
- Sommerville, Ian. *Software Engineering*, 8th ed. New York: Addison-Wesley, 2007.
- Tanenbaum, A. *Computer Networks*, 4th ed. New Delhi: Prentice Hall of India, 2006.
- McConnell, Steve. *Code Complete*, Redmond, Washington: Microsoft Press, 2004.