# 8/24/17

* Programming Language: Notational system for expressing computation in both human readable and machine readable format.
* Control abstraction
  + If stmts, loops, subroutines, libraries
* Data abstraction;
  + Double y = 3.47821;
  + Int x = 34;
  + Structs/records
  + arrays
* Object oriented programming (O.O.)
  + Combines methods and structures.
  + Encapsulation
  + Private
  + Grouping of data and methods
  + Abstraction
* Soft real time: streaming audio/video,
  + It can “fail a little”
* Hard real time
  + It **cannot** fail
* Readability criteria
  + Orthogonality
  + Support for abstraction
  + Expressivity

# 9/5/17

* Evaluation of languages for figuring out which language to use.
  + Cost
  + Speed of execution
  + Ease to learn
* Computer Architecture influence
  + Well-known computer architect: Von Neumann
  + Paradigms:
    - Imperative:
      * Sequence of steps
      * Algorithm
      * Variables to hold state
      * Loops
  + 50s: Worried about computer efficiency
  + late 1960s People efficiency becomes important.
* Logic:
  + A program is a set of facts and rules

## 9/7/17

* ALGOL is the first true high level language
  + Introduced:
    - Blocks
    - Parameters
    - Recursion
  + Problem is it didn’t have a compiler
  + Was never widely used
    - Because hard to compile
    - And lack of support from IBM
    - everyone used Fortran already
* COBOL
  + Was created to manipulate data much better
* Basic
  + Was created to make it easier for programmers and easier to learn for non-science students.

# 9/14/17

* Lexical: find tokens/terminals
  + Ex: if while
  + Identifiers
  + Literal values
  + Symbols & operators { } > = etc.
  + Regular expressions find these
* Syntax: Tokens in a valid order
  + Specified with a “Grammar”
* CFG
  + Context free grammar
    - Things that look the same no matter where they are used.
  + BNF: Bakus-Naur Form is a way of writing a CFG.
* Program -> begin statements end.

**PGM**

Begin stmts end. By(1)

Begin stmt stmts end by ()

* BNF Grammar:
  1. Set of terminals (token)
     + {begin end. = ; output xyz 0…9}
  2. set of non-terminals
     + {pgm stmts stmt assignment output ident value}
  3. set rules/productions of form
     + nonterm 🡪 any combination of non-terminals & terminals.
  4. Start symbol: non termento.
     + Start derivation or parse tree by default: 1st one = pgm

# 10/17/17

* Subprograms
  1. Procedure
  2. Method
  3. Function
* Most languages pass params with call by value
  1. C++ has call by reference as well
  2. Ada has:
     + Call by value default in
     + Call by value-result in out for primitives integer
     + Call by reference in out for composite types like records/arrays
     + Call by result out
  3. Java Technically only has call by value … however since all objects are references, acts like call by reference
* Call by value:
  1. At call, evaluate actual params. To get values. Actual can be any expression
  2. Create local vars for formal & initialize to values of actual. Created left to right … pushed on stack
  3. During subroutine execution, acts like local variables.
  4. At return, formal are “destroyed” & not passed back … actual do not change right to left … popped off stack.
* Every Subroutine can have formal params & local vars which are created at call activation record gets created for each subroutine call.
  1. Local vars.
  2. Formal parameters: push left to right
  3. [Register Values]
  4. Return Address: What do you do after subroutine is completely finished.
     + Next statement

# 10/26/17

* ADT (Abstract Data Type)
  + Grouping of data & Set of operations on that data.
  + Stack – abstraction
  + Queue – Information Hiding
  + List – Encapsulation
  + Stack –data type: Int objects
  + C-HeaderFile:
    - Type def itemtype
    - Typedef stack
    - Void push(Stack s, itemtype i)

# 10/31/17

* THREADS!!
  + Difference between processes and threads
  + Processes: program in execution
    - Have different address spaces
    - Each process/program doesn’t touch each other process
  + Each process has multiple threads:
  + Threads:
    - Can share address spaces
    - Can have same code or different code per threads
* Semaphore
  + Acquire:
    - Subtract 1 from int value
    - If new value is < 0, block caller thread
  + Release:
    - Add 1 to int value
    - If new value is -, unblock one caller
  + Run-time
    - System ensures only one thread can call these at a time
    - Calls are mutually exclusive