Brief History of Simulation

1950’s: Coding in *procedural* languages
   *EX*: FORTRAN, Basic, …

1960’s: *General*-purpose simulation languages
   *EX*: GPSS, SIMSCRIPT, SLAM, DYNAMO, …

1970-80’s: *Special*-purpose simulation languages
   *EX*: ManSim, Crystal Ball, COMNET, SIMAN, …

1990’s: Menu-driven, graphical *simulators*
   *EX*: Arena, Extend, ProModel, Witness, …
SIMAN

- SImulation Modeling and ANalysis
- Developed by Dennis Pegden ~ 1982
- 1st simulation language for the PC
- General purpose with manufacturing capabilities
- Process orientation: Entities flow through resources
- Input & Output Analyzers, Interactive Debugger
- Still serves as Arena’s core language
Arena-SIMAN Relationship

Arena is really a graphical interface for SIMAN

After GO, Arena translates “filename.doe” into:

- “filename.mod” → Model structure: Blocks
- “filename.exp” → Model data: Elements

Compile & linker step does error & consistency checks:

- “filename.p” → Executable program

Model runs & eventually finishes:

- “filename” → Output reports (Access database)
- “filename.out” → Simpler summary report (old but good)
Possible Errors

Some **Linker** Errors:

EX: Undefined identifiers: “John” vs. “Jon”

EX: Mismatched labels: “Label5” vs. LabelFive”

To avoid: Use pull-down lists as much as possible

Some **Run-time** Errors:

EX: Exceeded 150 entity limitation

EX: Index value = 0: using 0 as an index into an array

To locate: Use Find & Edit buttons
Model Frame

- Specifies system’s processes
- Arena’s flowchart modules ➔ ~70 SIMAN Blocks
- Entities move through & \textit{activate} the blocks
- Blocks have \textit{operands} (or \textit{arguments}):
  - Operands determine block’s exact function
  - Attribute names may be used in most operands
  - Some operands have default values
- To see this frame: \textit{Run} > \textit{SIMAN} > \textit{View}
Experiment Frame

• Constructed behind the scenes by Arena
• Arena’s data modules ➔ SIMAN Elements
• Elements list the names of Resources, Queues, Variables, Schedules, Sets, …
• Philosophy: Separate model’s experimental conditions (parameters) from its structure
• To see this frame: Run > SIMAN > View
Additional Arena Details

• Arena is case insensitive: NAME = name
• Almost all of SIMAN’s syntax is done for you by Arena, except …
• Expressions & Conditions require care
  \texttt{EX: DISC(0.4, 1, 0.7, 2, 1.0, 3)}
  \texttt{EX: IF, (Demand > 50) .AND. (Type .EQ. 2) …}
• Use “Expression Builder” whenever possible
Variables

• Variables are “global”
  Variable values are seen by all entities
• User-defined variables:
  **EX**: ServiceTarget, TotalCost, ...
  Arrays are allowed, e.g., Rate(period)
• Pre-defined (or system) variables:
  **EX**: TNOW = current simulation time
  **EX**: NQ(QueueID) = # of entities in queue
Attributes

• Attribute values are “local”
  Attribute values are entity-specific

• User-defined attributes:
  EX: Assign: $APT = \text{EXPO}(27)$;
  EX: Assign: $\text{TimeOfArrival} = \text{TNOW}$;

• Pre-defined (system) attributes:
  EX: $\text{Picture} = \text{Green Ball}$
  EX: $\text{Station} = \text{DrillPress Station}$
Expressions

- Involve standard arithmetic operators, functions, variables & entity attributes
- Usual order of operations: ( ), **, *, /, +, –
- Evaluated from Left to Right
  \[ \text{EX: } \text{TNOW} - (\text{TimeIn} + \text{TravelTime}) \]
- Use ( ) to override:
  \[ \text{EX: } \text{TNOW} - (\text{TimeIn} + \text{TravelTime}) \]
Logical Operators

- Can use symbol or 2-letter equivalent:
  - `<` (LT)  `<=` (LE)  `==` (EQ)
  - `>` (GT)  `>=` (GE)  `<>` (NE)
- `==` is a logical operator; `=` is used for assignments
- Logical expressions evaluate to 1 if true, 0 if false
  - **EX:** Assign: Demand = 10*(TNOW > 60)
- `{ .OR., .AND. }` can be combined with others
  - **EX:** IF, (Type == 2) .OR. (TNOW < 480) …
- There is no `.NOT.` logical operator
Math Functions

SIMAN has 20 standard math functions, e.g.,

ABS(a) \rightarrow Absolute Value
AINT(a) \rightarrow Truncate to get integer part
AMOD(a_1, a_2) \rightarrow Real remainder
ANINT(a) \rightarrow Round to nearest integer
MN(a_1, a_2, \ldots) \rightarrow Minimum value
MX(a_1, a_2, \ldots) \rightarrow Maximum value
SQRT(a) \rightarrow Square root