**Syntax:** form of expr, stmts, and units. **Semantics:** meaning of exprs. Syntax and Smtcs. Should be closely related. **Lexemes** are the smallest syntactic unit. Lexemes include numbers, operators, and spec. words. A program is in essence a string of lexemes. **tokens** are groups of lexemes, operators may be a token, while “=” is a lexeme from that group. **Recognizers** can distinguish code from noncode they are used to recognize code given to them, **generators** can generate valid code, they can be useful for determining if your code evaluating it. **Grammars** are formal-generation mechanisms for the syntax of a program. **Context-Free Grammars** are used to describe entire languages with few exceptions to that. John Backus and Peter Naur developed **BNF** or the **Backus-Naur Form**, first prototyped for ALGOL 58, it was concurrently developed by Chomsky and Backus, then expanded on for ALGOL 60 by Naur. Meta-languages are languages that describe another language. BNF is a good example. You define an abstraction (<assign>) for example with a left-hand side and right-hand side. The LHS is the abstraction to be defined, the RHS consists of tokens, lexemes, and/or references to other abstractions. This LHS and RHS combo is a **rule** or **production**. The abstractions are called **nonterminal symbols**, the tokens, etc used are **terminals.** Grammars will be essentially sets of rules. Attribute grammars describe certain rules such as type compatibility, adding to brevity. **Static Semantics** describe rules that are not easily or possibly described by BNF. **Attributes** are like vars. Allow value assignment, additionally computation functions exist for attributes allowing specifications of attribute computation. Predicate functions state the semantic rules of a language. **Dynamic Semantics** are the meanings of exprs, stmts, and units. **Operational Semantics** describe meaning through effects and changes in machine state.

**Interpretation vs. Compilation and Hybrid Implementation**. Compilation translates HL code to machine code. Interpretation involves exactly that, highly inefficient. Hybrid systems turn HL code into a mid-level. Then it interprets the mid-level code. Although, JIT converts then compiles when needed. All methods of translating code use **lexical analyzers** and **parsers** (**syntactic analyzer)**. Lexical analyzers care for tokens of most kinds, while syntax analyzers process exprs and stmts. The two are separated to improve simplicity of both (lexical and parser are simpler when sep., efficiency of the lexical analyzer (parser is not optimizable), and portability (syntax is often indepen., lexical is not) . **Lexical** extacts lexemes to produce tokens, functions which get lexeme, produce token and get assoc. code. **Top-Down** take general form and use rules to form next sentential. **Bottom-Up** takes a sentential form and reverses it to the previous form, starting from the leaves of the parse tree, it must find the **handle** or proper RHS for a sent. form. Parsing of unambiguous grammars, are complicated and inefficient since they are general, more specific -> faster.

Subprograms have a single entry, caller is suspended during called execution, control returns to caller. Definition -> interface and actions. Call -> request to subprog. to run. **Active** when called but no done executing. **Header** specifies code unit is a subprog., provides name, list of parameters. Param. Profile contains, num, order, and type of formal params. The **protocol** is the param prof. and return type if it is a func. Func. declarations are required for static typing of params, they are prototypes. Formals are in the header and are bound through other vars., actuals are those passed in the func call. Positional require positioning the be the same, keyword require same name to be used. Procedures do not return values, functions do. Procedures are old and are mostly replaced by funcs with void return val. Procedures would use shared vars or parameters to return any changed data. Functions should not produce side effects, if they are mathematically faithful. In/Out/Inout modes, receive data, send data, both from caller to called. **Polymorphism** the ability of functions to have different definitions or headers. Subtype – allows a func to access derived classes of a base class offered. Parametric – allows a func to use different types (generics). Remember Overloaded Operators.

Activation Records – Local Vars, Params, Dyn. Link, Return Address, from top to bot. Dynamic link points to AR base. In static scopes, provides traceback info, in dynamic provides access to non-locals. All ARs form on a run-time stack. EP points at the base of our stack, the first AR. Will always point to the base of the AR of the currently running unit. Offsets are used to determine the top of the AR set.