Lexical units.

Lexical units are elements of the source text that have been recognized as such and tokenized by the compiler.

A lexical analyzer is a pattern matcher for character strings, it’s a “front-end” for the parser.

Data types.

A data type defines a collection of data objects and a set of predefined operations on those objects.

Primitive data types: Those not defined in terms of other data types.

Integer: Java’s signed integer sizes: byte, short, int, long.

Floating Point: Model real numbers, but only as approximations. Languages for scientific use support at least two floating-point types (e.g., float and double; sometimes more)

Complex: Each value consists of two floats, the real part and the imaginary part

Decimal: Store a fixed number of decimal digits, in coded form (BCD), Advantage: accuracy, Disadvantages: limited range, wastes memory

Boolean: Range of values: two elements, one for “true” and one for “false”. Could be implemented as bits, but often as bytes. Advantage: readability.

Character: Stored as numeric codings. Most commonly used coding: ASCII

Character String Types: Values are sequences of characters. (Is or is not a primitive data type in different languages)

(

- **String** (or **str** or **text**). Used for a **combination of any characters that appear on a keyboard**, such as letters, numbers and symbols.
- **Character** (or **char**). Used for **single letters**.
- **Integer** (or **int**). Used for **whole numbers**.
- **Float** (or **Real**). Used for **numbers that contain decimal points**, or for **fractions**.

- **Boolean** (or **bool**). Used where data is restricted to **True/False** or **yes/no** options.

)

Named constant.

A named constant is a variable that is bound to a value only when it is bound to storage, used to parameterize programs. The advantages are readability and modifiability. The binding of values to named constants can be either static (called manifest constants) or dynamic.

Variable.

A variable is an abstraction of a memory cell.

Variables can be characterized as a sextuple of attributes:

Name: not all variables have them

Address: the memory address with which it is associated

Type: determines the range of values of variables and the set of operations that are defined for values of that type; in the case of floating point, type also determines the precision. The l-value of a variable is its address. The r-value of a variable is its value.

Value: the contents of the location with which the variable is associated

Lifetime: the time during which it is bound to a particular memory cell

Scope: the range of statements over which it is visible.

Expressions

An expression in a programming language is a combination of one or more explicit values, constants, variables, operators and functions that the programming language interprets and computes to produce another value.

Statements.

In computer programming, a statement is a syntactic unit of an imperative programming language that expresses some action to be carried out. A program written in such a language is formed by a sequence of one or more statements. A statement may have internal components (e.g., expressions).

Program units.

A program unit is a sequence of one or more lines, organized as statements, comments, and INCLUDE directives. Specifically, a program unit can be:

The main program

A module

A block data program unit

An external function subprogram

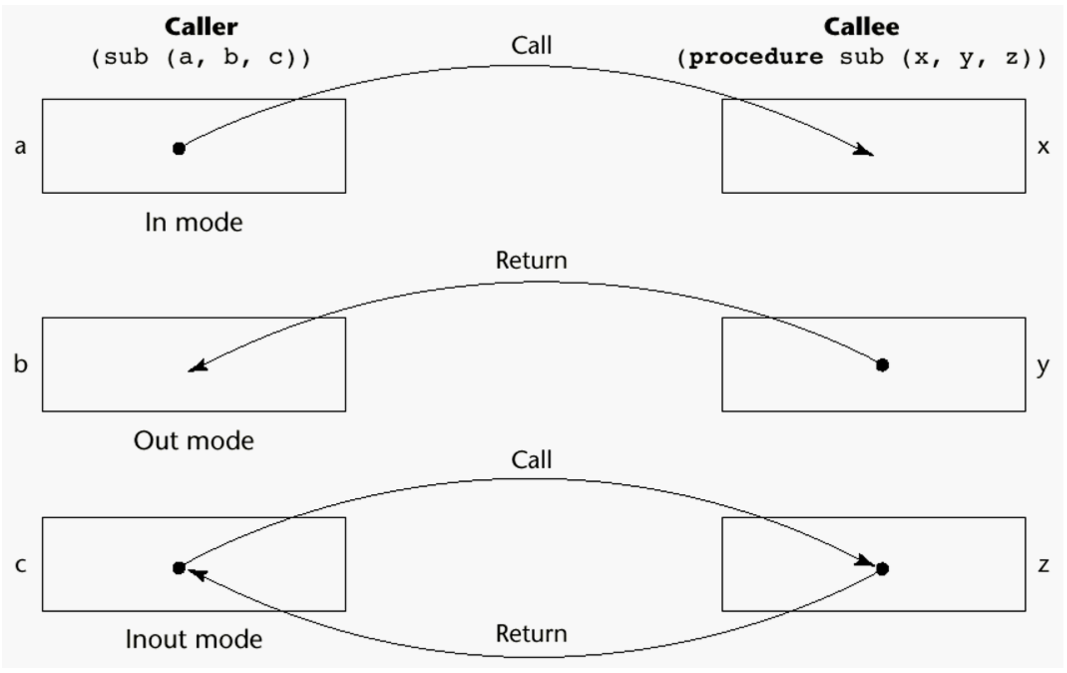
An external subroutine subprogram

Parameter evaluation

Parameter evaluation is the process of mapping formal and actual parameters when a subprogram is called; evaluation also determines the information which governs the communication of the parameters.

Parameter passing methods.

Semantic Models of Parameter Passing: In mode, Out mode, and Inout mode.



Pass-by-Value (In Mode): The value of the actual parameter is used to initialize the corresponding formal parameter.

Pass-by-Result (Out Mode): When a parameter is passed by result, no value is transmitted to the subprogram; the corresponding formal parameter acts as a local variable; its value is transmitted to caller's actual parameter when control is returned to the caller, by physical move.

Pass-by-Value-Result (inout Mode): A combination of pass-by-value and pass-by-result. Sometimes called pass-by-copy. Formal parameters have local storage. Disadvantages: Those of pass-by-result, Those of pass-by-value.

Pass-by-Reference (Inout Mode): Pass an access path. Also called pass-by-sharing. Advantage: Passing process is efficient (no copying and no duplicated storage). Disadvantages: Slower accesses (compared to pass-by-value) to formal parameters, Potentials for unwanted side effects (collisions), Unwanted aliases (access broadened).

Pass-by-Name (Inout Mode): By textual substitution. Formals are bound to an access method at the time of the call, but actual binding to a value or address takes place at the time of a reference or assignment. Allows flexibility in late binding.

Block.

A block is a section of code that has its own local variable whose scope is minimized.

Blocks are user-specified local scopes for variables.

An example in C:

```
{
int temp;
temp = list [upper];
list [upper] = list [lower];
list [lower] = temp
}
```

The lifetime of temp in the above example begins when control enters the block. An advantage of using a local variable like temp is that it cannot interfere with any other variable with the same name.

Scoping.

Scope: The range of statements in which the variable is visible. A variable is visible in a

statement if it can be referenced in that statement.
Static scope: is based on program text and to connect a name reference to a variable, you (or the compiler) must find the declaration.
Dynamic scope: Based on calling sequences of program units, not their textual layout (temporal versus spatial). References to variables are connected to declarations by searching back through the chain of subprogram calls that forced execution to this point.

Abstract data type.

An abstract data type is a user-defined data type that satisfies the following two conditions: The representation of, and operations on, objects of the type are defined in a single syntactic unit. The representation of objects of the type is hidden from the program units that use these objects, so the only operations possible are those provided in the type's definition.
Advantage of the first condition: Program organization, modifiability (everything associated with a data structure is together), and separate compilation.
Advantage the second condition: Reliability--by hiding the data representations, user code cannot directly access objects of the type or depend on the representation, allowing the representation to be changed without affecting user code.

Exception handling.

Many languages allow programs to trap input/output errors (including EOF). An exception is any unusual event, either erroneous or not, detectable by either hardware or software, that may require special processing
. The special processing that may be required after detection of an exception is called exception handling
. The exception handling code unit is called an exception handler
.

Advantages: Error detection code is tedious to write and it clutters the program. Exception handling encourages programmers to consider many different possible errors. Exception propagation allows a high level of reuse of exception handling code.
The catch function:

catch is the name of all handlers--it is an overloaded name, so the formal parameter of each must be unique. The formal parameter need not have a variable. The formal parameter can be used to transfer information to the handler. The formal parameter can be an ellipsis, in which case it handles all exceptions not yet handled.

Throwing Exceptions

Exceptions are all raised explicitly by the statement:

throw [expression];

The brackets are meta-symbols. A throw without an operand can only appear in a handler; when it appears, it simply re-raises the exception, which is then handled elsewhere

. The type of the expression disambiguates the intended handler

.

Unhandled Exceptions:

An unhandled exception is propagated to the caller of the function in which it is raised. This propagation continues to the main function

. If no handler is found, the default handler is called

.