CH1

Prog Domains • Scientific, Buisness, AI, System Programing, Web

language Eval Criteria

- Readability: Simplicity, orthogonality, Data types, syntax
- Writability: Simplicity + ortho, Abstraction, Expressivity
- Reliability: Type checking, Exe handling, Aliasing, Read 4write
- Cost: Training, Writing, Exe, Maintaining

language Categories

- Imperative variables, Assign., iteration
- Functional apply fns to param
- Logic Rule Based
- Markup Extended

Trade-Offs

Reliability vs Cost of Exe Java all ref be checked ⬆ exe costs

Readability vs writability write compact ↓ Reabality

Writabitiy (Flex) vs Reliabilty C++ ptrs powerful but unreliable

Implimentation Compilation, Pure Interpretation (No translation), Hybrid

CH3 BNF REVIEW

CH5

Variable Naming
reserved word can not be used as a user-defined name

- case sensitive? → Disadvantage: readability (names that look alike are diff
- Special words, reserved words, Keywords? • aid readability
  Keyword only special in certain contexts
- Length if too short, can not be connotive
- special characters

Binding complile time (var to type) load time (static to mem) Runtime (nonstatic to mem)

Static + Dynamic Binding ← first occurs during exe / can change
↑
First occurs before runtime & remains unchanged through prog exe

Type Binding static, type may be specified by explic/ Implic declaration.

Dynamic type Binding flex but High cost

Static vs Dynamic Scope - Static first declare @ compile, Dynamic declare @ runtim

Reference Environment - collection of all names that are visible in the statement
+ convience
- poor readabitity

Static - all local variables + all visible var in all enclosing scopes

Scope + lifetime are different

Dynamic - the local variables + all visible var in all active subprograms

Named Constants variate bound to a value only when it is bound to storage
~~immutable, can not change after assignment~~ static or dynamic
+ readability
+ maintainabi

CH6.

Data Types, Primitive: Int, float, Bool

Char String type: sequence of char "hello"

Enum type: User-defined, Fixed set of Names

Array type: Ordered collection, all same type (in some languages)

Associative Array: Key-value pairs, fast data lookups

Record type: Groups related fields und one composite data struct. (heterogeneous, individual elem ID by name)

Tuple type: similar to Record elements are not named

List-types: Lisp ( , , ) elements

Union types: var are allowed to store diff types value @ different times during exe.

Ptr Ref types: range of val, consist of mem addr & special val nil address storage (heap)

Optional type: useful when there is a need for var to indicate no value

Type checking: ensure operands of operator are of compatible types

Strong typing: type checking + helps compiler

Type Equivalence: Name-type vs structure type
- equal if same declaration or if use same type name
- equil type if same structure. (more flexible)

struct x, y; same
Set x, y;

[0, 9] [1, 10] same

CH7

Short-cut Eval result is determined w/o evaluating all operators

Type Conversions Narrowing type conversion (float to Int) cannot include all value.

Widening type conversion (Int to float) include @ least approx.

Overload of Operations use of an operator for more than one purpose. (+ for Int & float) - loss of readability + err detection

Mixed Mode has operands of diff types (coercion) decrease type error detection if ... else ↑ Single Way vs MultiWay ( Allow selection of one of any # of statements or statement group (switch)

CH8

Selection statements provides means of choosing between two or more paths of exe

Interative statements Iterator for (i ; stop ; traverse) foreach items in List

transferes execution control to a specifies place in the program (Readability)

Uncondinonal Branching

Guarded Commands new proj that supported verification during dev, order of eval is not important (Selection Gard vs loop)

if <Bool xpress> <statement>

do <Bool> <statement>

CH9 design Issures (Plenty)

Local Ref Enviro

Stack dynamic ← typical

Pros Recursion, storage is shared

cons Allocate/deallocate, indirect Addr, Sub-proj cannot be history ~~[illegible]~~ sensitive

Local Var can be static ↑ opposite

Restore exe status of the caller

transfer control Back to caller

Param Passing

- In-mode caller → callee
- Out-mode caller ← callee
- Inout-mode

↓ Pass by Name

Flexable + late Binding, textual substitution

→ Pass by Value • value of actual param used

→ Pass by Result • extra storage, No val transmitted to subprog

→ Pass by Val-Result Formal param have local storage

→ Pass by Refference

pass an access path, efficient, slower, side effects unwanted aliases.

↑ simple to implement

Subprogram Names as Parameters

- Ref environment: Shallow Binding, Deep binding, Ad hoc
- Indirectly: when there are several & correct one on exe not known untill exe.

Closure – the ref enviro is where it was defined

Coroutines ↳ anonymous function retured by FN call (FN created by FN)

↳ subprog w multiple enteries & controlls them themselves

CH10

- call & return: Subprog linkage

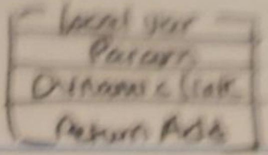
Implementing "simple" subprog:

call semantics

- save exe status of caller
- pass the params
- pass the return address to the called
- transfer controll to the called

return semantics if a FN: move the functional value to a place caller can get it

if pass by val or out mode: param are used, move curr values of param to actual va



Implementing stack - Dynamic local Variables
Nested Subprog All var that can be locally add reside in some activation record instance in the stack
Blocks — user specified local scopes
Implement Dynamic Prog
- Deep Access: non-loc references are found by searching the activation record instances on dynamic chain
- shallow Access: put loc in a central place

CH13 Semaphore - data Struct consisting of a counter & a queue
- wait/release - competition + sync

Monitors - for shared data (encapsulate) restrict access
- Better then Semaphores (can be used to Implement other)

Message Passing generenal model for concurrency (wait for eachother)

CH14 Exception Handing
Event Handling