# **Chapter 7: Expressions and Assignment**

## Define terms associated with expressions

### Arithmetic – operators (including parentheses), operands, function calls

#### Operands can be unary, binary or ternary

### Boolean – True or False

#### Sometimes 0 and 1

##### Reduces readability and increases error risk

#### Generally have lower precedence than arithmetic operators

### String

### Programmer-Defined

### Side effects – when a function changes two-way parameter or global variable

#### Problems:

##### Function referenced in an expression alters another operand of the expression.

##### The value of the altered operand changes during evaluation of the expression

## Explain design issues for expressions, including:

### Precedence – operator precedence rules define the order which the operators of an expression are evaluated

#### APL has uniform (no) operator precedence

### Associativity – Defines the order of evaluation for adjacent operators having the same precedence level

#### Binary operators associate L to R

#### Unary operators associate R to L

#### Parentheses override associativity

#### APL: all operators have equal precedence and associate L to R

### Operand evaluation

#### Defines the order of evaluation for individual operands of an expression

##### Variable or Non-static constant: fetch the value from memory (maybe in cache or register)

##### Literal or static constant: usually stored in the machine language instruction, no fetch required

##### Function reference: Execute the function independently

###### The case of most interest due to side effects

##### Prefix increment/ decrement changes operand values

##### Java and C++ evaluate operands in a strict L to R order

### Lazy/eager evaluation

#### Lazy Evaluation (Short Circuit)

##### Suppresses evaluation of terms once the net value of the expression is evident

##### Used for guard conditions

##### C,C++ use lazy evaluation for Boolean operators

##### Ada, Java have both lazy and eager

###### Java: &&, || = lazy, &,| = eager

##### Disadvantage: lazy evaluation introduces potential problem of side effect expressions

## Interpret expressions under specific rules

## Explain options for mixed mode expressions and type conversion

### Type Conversions

#### Dynamic translation of a value from one data type to another

#### Necessary for mixed-mode expressions (expressions containing different types)

#### Narrowing Conversion: one that convers a value to a type with a lesser capacity (int to byte)

#### Widening Conversion: Converts a value to a type with a greater capacity (int to double)

### Type Coercions

#### An implicit type conversion

#### Decreases the compilers ability to detect type errors

#### Languages:

##### Perl: coercions of all type

##### C,C++: narrowing and widening for numerical coercions

##### Java: widening numerical coercion only (can still lose data)

##### Ada: no coercions

### Type Casts

#### An explicit dynamic conversion

#### Java goes before variable while Ada goes after

## Explain relative strengths, weaknesses, and costs of design alternatives

### **Solution 1**: Disallowing Side Effects

#### No 2 way parameters in functions

#### No global references in functions

##### **Advantage**:Simplifies the language by eliminating side effects

##### **Disadvantage**: Programmers want the flexibility of 2-way parameters and global references

### **Solution 2**: Strictly define operand order evaluation

#### **Disadvantage**: Limits compiler optimization (less efficient)

### Overloading Operators

#### Use of an operator for more than one purpose

#### C,C++,F95, Ada allow user-defined overloaded operators

#### **Disadvantage**:

##### Increased error risk, less error detection

##### Decreased Readability

##### User can define nonsense operators

#### **Advantage**: a flexible and powerful feature

## Evaluate different forms of assignment (strengths / weaknesses)

### Operator Variation – a:= b + 1

#### Algol, Pascal, Ada

### Multiple targets – ($a,$b) = (10,3)

#### Perl

### Conditional targets – (a<b) ? a:b=0

#### C, Perl

### Compound operators – sum += next

#### C,Java, Perl

### Unary assignment operators – a++

#### C, Java, Perl

### C,C++, Java: Assignment produces a result, that can be used as operands

### Disadvantage: reduces readability, increases error risk

# **Chapter 8: Control Structures**

## Define terms associated with control structures

### Control Flow: the sequence of instruction execution in a program

#### Program specifications of control flow and data structure

### Control Structure: A statement for managing control flow and the statements that it controls

### Selection Statement: provides the means for choosing between 2 or more paths of control

#### 2-way selectors

#### Multiple-way selectors

### Control Flow Mechanisms

#### Among program units

##### Procedure Calls

##### Exception handing

##### Concurrency (threads)

##### Recursion

#### Within program units

##### Selection

##### Iteration

##### Branching

## List and describe the standard types of statement-level control structures

### Iteration

#### The repeated execution of a statement or group of statements

#### Forms of iteration

##### Counter-controlled (for loop)

##### Logical (while loop)

##### Data structure traversal (data iterator)

#### Design Issues

##### How is iteration controlled

##### Where is the control mechanism for the loop

##### Are branches into/out of the loop allowed

### Selection

#### 2-way (if-then-else)

#### Multi-way(switch/case)

#### Ternary ( ?: ) s

### Branching

#### Branching statements transfer control directly to another statement in the program (GOTO statements)

#### Results in an unstructured program, which is difficult to read and verify

#### Newer languages do not support this

##### Pascal: unsigned int with colon

##### FORTRAN: unsigned integers (100)

##### Algol 60, C: identifiers with colons (start:)

## List and describe the standard forms of selection statements, and explain design issues

### 2-Way

#### Design Issues

##### What is the formal and type of control expression (the conditional)

##### How are then and else clauses specified

###### Else matching problem

Algol Solution

Disallow direct nesting

Ada,F90

Explicit closure (endif)

C, Java

Static Semantics: else goes with nearest if

Compound statements can nest else-less ifs

Python

Indentation

##### How should the meaning of nested selectors be specified

#### Fortran II

##### Negative Logic – poor readability

##### Limited to a single statement, to select more need to use goto

##### If statement

#### Algol 60

##### If, then, else

##### Statements could also be compound

###### “Then begin”

#### C

##### Conditional form is integer (no Boolean)

##### Error risk from mixed-mode expressions and side effects

#### Java, Ada

##### Boolean expressions only

##### Strong type safety

#### Cobol

##### Implied subject/operator

### Multiple Way

#### Design Issues

##### What is the form and type of control expression

##### How are the selectable segments specified

##### Is execution flow through the structure restricted to include just a single segments

##### What is done about unrepresented expression values (default case)

##### Is the selection implemented as an expression or control statement

#### Fortran II

##### Arithmetic IF (3 way selector w/ branching)

##### **Disadvantages**:

###### Segment exits require GOTOs

###### Not encapsulated

#### Pascal

##### Case statements

##### Design Choices:

###### Expression is any ordinal type

###### Segments can be single or compound

###### Only one segments can be executed

#### Ada

##### Similar to Pascal

##### List of constants must be exhaustive

###### Often accomplished with others clause

###### Makes Ada’s selection construct more reliable

#### C,C++, Java

##### Design Choices

###### Control expression can be integer only

###### Segments can be statement sequences or compound statements

###### No implicit branch at segment end

Programmer must supply break (often source of error)

Trade off flexibility for realiability

Optional default clause for unrepresented values

#### Ruby

##### Selection forms an expression

#### Lisp

##### Selection forms a function with a return value

#### Else-if

##### Multiple-way selectors can also be implemented using else-ifs

## List and describe the standard forms of iterative statements, and explain design issues

### Counter Controlled Iteration

#### Design Issues:

##### What are the type and scope of the loop variable

##### What is the value of the loop variable at loop termination

##### Should changes to loop variable and parameters be allowed in the loop body

##### Should the loop parameters be evaluated only once, or every iterations

#### s

### Logically Controlled iteration

#### Design Issues:

##### Pretest or Posttest

##### Control: while(true) or until(true)

##### A special case of counting loop statement (or a separate statement)

#### C,C++, Java have while and do while loops

##### Java must be Boolean

#### Perl

##### 2 pretest logical loops, while and until, but no posttest logical loop

### Data structure Iteration

#### Iterates over the elements of some data structure

#### Control Mechanism is a call to a function that returns the next element in some order, if there is one, else exit loop

#### C,C++

##### For statement can be used with pointers

#### Java, Perl

##### For each statements

## Explain conditional branching and evaluate its use

## Interpret control statements in Java, Perl, Racket, and Prolog

## Explain the significance of Dijkstra's claim and the Bohm-Jacopini Theorem

### Djikstra realized goto statements were harmful, increases programming errors

### Bohm-Jacopin Thm: any algorithm can be coded without branches (goto), using sequential execution, if and while

# **Chapters 9 & 10: Subprograms**

## List and explain design Issues for subprograms

### How are parameters passed

### Are local variables static or dynamic

### Can subprograms be passed as arguments

### Can subprogram definitions be nested

### Can subprograms be overloades

### Can subprograms be polymorphic (generic)

## Define terms associated with subprograms

### Procedure/subroutine: user defined operation invoked with a program statement

### Function: a user defined operation on some parameter that returns a value (generally has no side effects

### Formal parameter (parameter): a dummy variable listed in the subprogram header

### Actual param (argument): a value or address used in the subprogram call statement and used by the sub program in computation

### Parameter profile: the number, order, and types of its parameters

### Signature: name and parameter profile

### Protocol: return tyoe and parameter profile

### Header/decleration: first line of definition, including return type, name and profile

### Subprogram def: description of the actions of the subprogram abstraction

### Subprogram call: is an explicit request that the subprogram can be executed

### Generic of polymorphic subprogram: takes parameters of different types

## Explain and apply standard parameter-passing modes (both semantic and implementation models)

### Semantic model:

#### In mode (call)

#### Out mode (return)

#### In-out mode (both)

### Implementation model:

#### Transfer a value:

##### Pass by value

###### Requires copying of parameters, extra memory and extra time to initialize call

##### Pass by result

###### Requires an extra level of indirection- extra time during execution

###### In mode and out mode parameters must be read protected or write protected

##### Pass by value result

###### In mode: actual parameter value is copied to formal parameter upon entry

###### Out mode: formal parameter value is copied to actual parameter upon return

###### In-out mode: both copies are made

###### Advantage: simple semantics

###### Disadvantage:

Allocation and movement of data are required (space and time cost)

Duplicate parameter space (virtual, actual)

Order dependence is a problem if the same actual parameter is used more than once, or if a global is used as an actual parameter

#### Transfer an access path:

##### Pass by reference

###### In mode: formal parameter can be used for reference, but cannot be assigned a new value

###### Out mode: formal parameter value can be assigned but cannot be referenced

###### In-out mode: both reference and assignment are allowed

###### Advantage:

Efficient use of memory, time

Allows controlled side effects

###### Disadvantages

In mode parameters must be write protected and out mode parameters must be read protected

Additional layer of indirect addressing introduced by the reference increases cost of addressing

Aliasing results from using an actual parameter more than once, or using global as parameters

#### Replace variable name:

##### Pass by name

###### Resulting implementation

If actual scalar variable, it is effectively pass by reference

If actual is constant expression it is effectively pass by value

If actual is an array element it is like nothing else

###### Disadvantages

Most expensive method, due to constant reevaluation of the parameter

Difficult semantic: very hard to read and understand

## Explain and apply standard return-value modes

### Return by value: is a local variable

### Return by reference: is a reference

#### Advantage: efficiency

## Explain how parameter passing and return values are implemented via the run-time stack, and interpret subprogram execution scenarios

## List and explain design issues for overloaded subprograms

### Overloaded subprogram: has the same name as another subprogram in the same referencing environment

### C++, Java and Ada have overloaded subprograms built in and users can define there own