

Sequence Control

Dr. Nguyen Hua Phung

HCMC University of Technology, Viet Nam

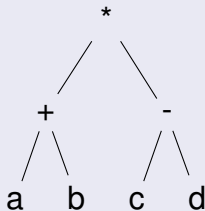
07, 2014

- 1 **Expressions**
- 2 **Statements**
- 3 **Program Units**

- An expression is a syntactic entity whose evaluation either:
 - produces a value
 - fails to terminate \rightarrow undefined
- Examples
 - $4 + 3 * 2$
 - $(a + b) * (c - a)$
 - $(b \neq 0) ? (a/b) : 0$

Expression Evaluation Mechanism

Expressions have functional composition nature



$(a + b) * (c - d)$

Expression Syntax

- Infix
- Prefix
- Postfix

$$(a + b) * (c - d)$$

- Good for **binary operators**
- Used in most **imperative programming** language
- More than two operands?
 $(b \neq 0) ? (a/b) : 0$
- Smalltalk:
myBox displayOn: myScreen at: 100@50

$$3 + 4 * 5 = 23, \text{ not } 35$$

- Evaluation priorities in mathematics
- Programming languages define their own precedence levels based on mathematics
- A bit different precedence rules among languages can be confusing

- If operators have the same level of precedence, then apply associativity rules
- Mostly left-to-right, except exponentiation operator
- An expression contains only one operator
 - Mathematics: associative
 - Computer: optimization but potential problems
 - $10^{20} * 10^{20} * 10^{-20}$

- Alter the precedence and associativity
 $(A + B) * C$
- Using **parentheses**, a language can even omit precedence and associativity rules
 - APL
- Advantage: simple
- Disadvantage: writability and readability

If statement

```
if (count == 0)
    average = 0;
else
    average = sum / count;
```

Conditional Expression

```
average = (count == 0) ? 0 : sum / count;
```

- C-based languages, Perl, JavaScript, Ruby

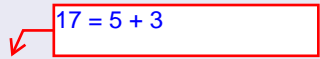
trừ với âm phải dùng 2
kí hiệu khác nhau

- Polish Prefix: * + a b - c d
- Cambridge Polish Prefix: (* (+ a b) (- c d))
- Normal Prefix: *(+(a,b),-(c,d))
 - Derived from mathematical function $f(x,y)$
 - Parentheses and precedence is no required, provided the -arity of operator is known
 - Mostly see in unary operators
 - LISP: (**append** a b c my_list)

- Polish Postfix: $a\ b\ +\ c\ d\ -\ *$
- Cambridge Polish Postfix: $((a\ b\ +)\ (c\ d\ -)\ *)$
- Normal Postfix: $((a,b)+,(c,d)-)^*$
 - Common usage: factorial operator (5!)
 - Used in intermediate code by some compilers
 - PostScript: (Hello World!) **show**

C program

```
int a = 5;
int fun1() {
    a = 17;
    return 3;
}
void main() {
    a = a + fun1();
}
```



What is the value of a? **8 20**

Reason: **Side-effect** on the operand of the expression

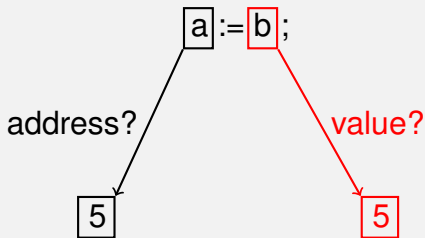
- Eager evaluation
 - First evaluate all operands
 - Then operators
 - How about $a == 0 ? b : b / a$
- Lazy evaluation
 - Pass the un-evaluated operands to the operator
 - Operator decide which operands are required
 - Much more expensive than eager
- Lazy for conditional, eager for the rest

`(a == 0) || (b/a > 2)`

- If the first operand is evaluated as **true**, the second will be short-circuited
- Otherwise, "divide by zero"
- How about `(a > b) || (b++ / 3)` ?
- Some languages provide two sets of boolean operators: short- and non short-circuit
 - Ada: "and", "or" versus "and then", "or else"

- A statement is a syntactic entity whose evaluation:
 - does not return a value, but
 - changes the state of the system
- Example,
a = 5;
print "pippo"
begin ... end

leftExpr AssignOperator rightExpr



- Evaluate left or right first is up to implementers

- C-based languages consider **assignment as an expression**

```
while ( (ch = getchar()) != EOF) { ... }
```

- Introduce compound and unary assignment operators (+, -, ++, --)
 - Increasing code legibility
 - Avoiding unforeseen side effects

- Control statements
 - Selecting among alternative control flow paths
 - Causing the repeated execution of sequences of statements
- Control structure is a control statement and the collection of its controlled statements

```
if control_expression  
  then clause  
  else clause
```

- Proved to be fundamental and essential parts of all programming languages

```
if (sum == 0)
    if (count == 0)
        result = 0;
    else
        result = 1;
```

- Solution: including block in every cases
- Not all languages have this problem
 - Fortran 95, Ada, Ruby: use a special word to end the statement
 - Python: indentation matters

- Allows the selection of one of any number of statements or statement groups
- Perl, Python: don't have this
- Issues:
 - Type of selector expression?
 - How are selectable segments specified?
 - Execute only one segment or multiple segments?
 - How are case values specified?
 - What if values fall out of selectable segments?

```
switch (index) {
```

```
case 1:
```

```
case 3:
```

```
    odd += 1;  
    sumodd += index;  
    break;
```

```
case 2:
```

```
case 4:
```

```
    even += 1;  
    sumeven += index;  
    break;
```

```
default: printf("Error in switch").
```

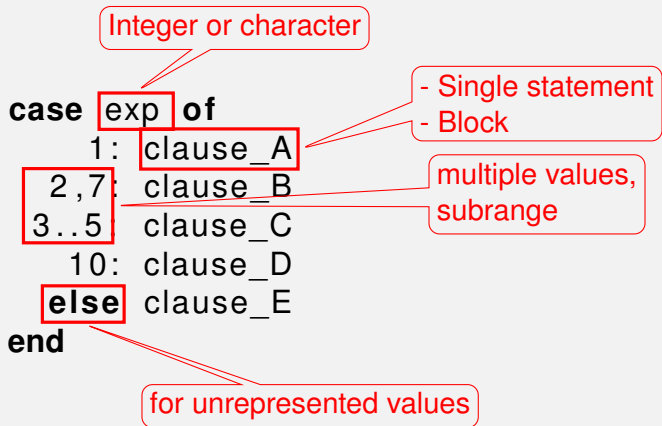
```
}
```

Type must be **int**
Exact value

- Stmt sequences
- Block

Multiple segments exited by **break**

for unrepresented values



- Cause a statement or collection of statements to be executed zero, one or more times
- Essential for the power of the computer
 - Programs would be huge and inflexible
 - Large amounts of time to write
 - Mammoth amounts of memory to store
- Design questions:
 - How is iteration controlled?
 - Logic, counting
 - Where should the control appear in the loop?
 - Pretest and posttest

- Counter-controlled loops must have:
 - Loop variable
 - Initial and terminal values
 - Stepsize

General form

for $i := \text{first}$ **to** last **by** step
do
 loop body
end

constant

Known number of iterations
before executing

Semantic

```
[define end_save]
  end_save := last
  i = first
  loop:
    if i > end_save goto out
    [loop body]
    i := i + step
    goto loop
  out:
[undefine end_save]
```

General form

```
for (expr_1; expr_2; expr_3)  
loop body
```



Can be infinite loop

Semantic

```
expr_1  
loop:  
if expr_2 = 0 goto out  
[loop body]  
expr_3  
goto loop  
out: ...
```

- Repeat based on Boolean expression rather than a counter
- Are more general than counter-controlled
- Design issues:
 - Should the control be **pretest or posttest**?
 - Should the logically controlled loop be a special form of a counting loop or a separate statement?

Forms

```
while (ctrl_expr)  
    loop body
```

Semantics

```
loop:  
if ctrl_expr is false  
    goto out  
[loop body]  
goto loop  
out:...
```

```
do  
    loop body  
while (ctrl_expr);
```

```
loop:  
[loop body]  
if ctrl_expr goto loop
```

- Programmer can choose a location for loop control rather than top or bottom
- Simple design: infinite loops but include user-located loop exits
- Languages have exit statements: **break** and **continue**
- A need for restricted goto statement

```
while (sum < 1000) {  
    getnext(value);  
    if (value < 0) break;  
    sum += value;  
}
```

- What if we replace **break** by **continue**?

- Controlled by the number of elements in a data structure
- Iterator:
 - Called at the beginning of each iteration
 - Returns an element each time it is called in some specific order
- Pre-defined or user-defined iterator


```
String[] strList = {"Bob", "Carol", "Ted"};  
...  
foreach (String name in strList)  
    Console.WriteLine("Name:{0}", name);
```

- Unconditional branch, or **goto**, is the most powerful statement for controlling the flow of execution of a program's statements
- Dangerous: difficult to read, as the result, highly unreliable and costly to maintain
- Structured programming: **say no to goto**
- Java, Python, Ruby: no goto
- It still exists in form of loop exit, but they are severely restricted gotos.

to be continued

- Expressions
 - Operator precedence and associativity
 - Side effects
- Statements
 - Assignment
 - Selection Statement
 - Loop structures