CHAPTER 6: Control Flow Part 1

CMPSC 460 – Principles of Programming Languages

Introduction

- What is *Control Flow*?
  - Order of execution of the operations or statements
  - Transmission of data

Arithmetic Expressions

$$root = \frac{-B + \sqrt{B^2 - 4 \times A \times C}}{2 \times A}$$

```
ROOT = (-B+SQRT(B**2-4*A*C)) / (2*A)
```

Arithmetic Expressions

- Expressions consist of:
  - Simple object
  - Operators and operands
  - function calls

Expressions: Operators

Types of operators:

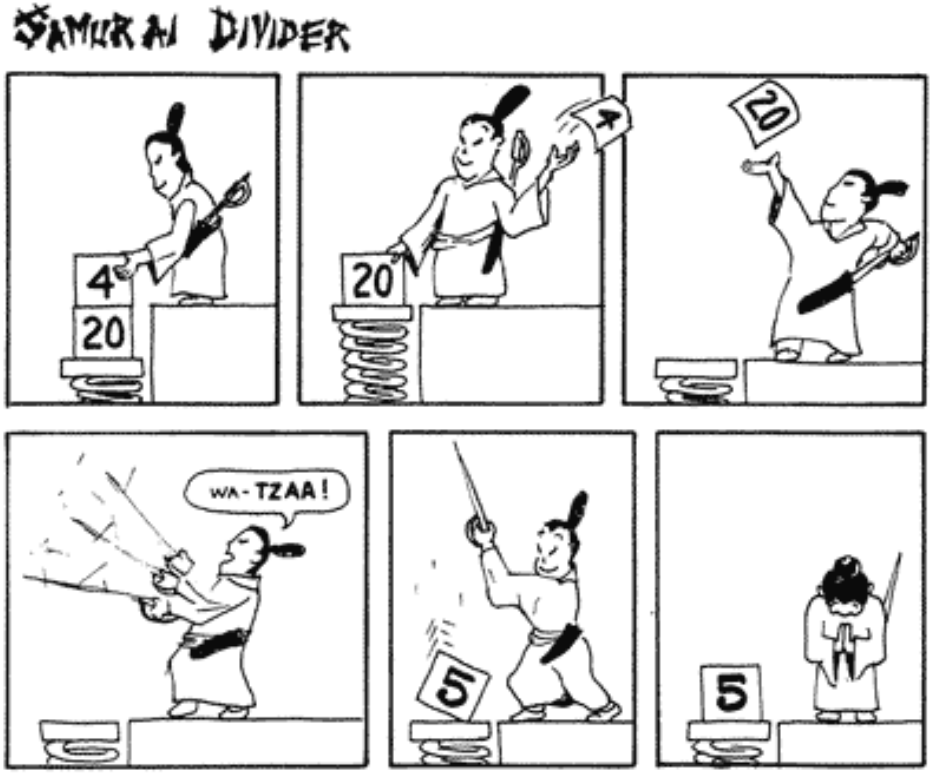
- A ***unary*** operator has one operand
- A ***binary*** operator has two operands
- A ***ternary*** operator has three operands

Design notation:

- Prefix
- Infix
- Postfix

Expressions: Operators

1. 20 4 / .
2. 17 20 132 3 9 + + + + .



From: http://www.forth.com/starting-forth/sf2/sf2.html

Arithmetic Expressions: Design Issues

- Design issues for arithmetic expressions
  - Operator overloading?
  - Operator precedence rules?
  - Operator associativity rules?
  - Order of operand evaluation?
  - Operand evaluation side effects?

Overloaded Operators


```
[]CHAR repetitions = "ab" + "cd";

[]CHAR array = repetitions * 3;

print(array)
```

Expressions: Operators

- How do we define operators?
  - Some languages define them as functions

Overloaded Operators


```
class Bag:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def __add__(self, b):
        b1 = Bag()
        b1.data = self.data + b.data
        return b1

bag1 = Bag()
bag1.add(5)
bag1.add(4)
bag1 = bag1 + bag1
print(bag1.data)
```

Operator Precedence Rules

- Typical precedence levels
  - parentheses
  - unary operators
  - ** (if the language supports it)
  - *, /
  - +, -

Fortran	Pascal	C	Ada
		++, -- (post-inc., dec.)	
**	not	++, -- (pre-inc., dec.), +, - (unary), &, * (address, contents of), !, ~ (logical, bit-wise not)	abs (absolute value), not, **
*, /	*, /, div, mod, and	* (binary), /, % (modulo division)	*, /, mod, rem
+, - (unary and binary)	+, - (unary and binary), or	+, - (binary)	+, - (unary)
		<<, >> (left and right bit shift)	+, - (binary), & (concatenation)
.eq., .ne., .lt., .le., .gt., .ge. (comparisons)	<, <=, >, >=, =, <>, IN	<, <=, >, >= (inequality tests)	=, /= , <, <=, >, >=
.not.		==, != (equality tests)	
		& (bit-wise and)	
		^ (bit-wise exclusive or)	
		| (bit-wise inclusive or)	
.and.		&& (logical and)	and, or, xor (logical operators)
.or.		|| (logical or)	
.eqv., .neqv. (logical comparisons)		?: (if...then...else)	
		=, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |= (assignment)	
		, (sequencing)	

Operator Precedence Rules

- Example

If A < B and C < D then …

Operator Associativity Rules

Example:

1. `9 - 4 - 2`
2. `a + b * c**d**e/f`
3. `a = b = a + b`
4. `++++++x`
5. `++--++++x`
6. `++--x++`
7. `x = (x=4, y=7);`

APL Example


```
A ← 5
B ← 4
C ← 6
A × B + C
A - B - C
```

Expressions vs. Statements

- What is the difference between a statement and an expression?

Statements

- Statements provide the means to make changes to the state of a program
- Side effect: a construct influences subsequent computation
- Statements:
  - Assignment
  - Input
  - Output

Expression-oriented vs. statement-oriented languages

- expression-oriented:
  - functional languages (Lisp, Scheme, ML)
  - Algol-68
- statement-oriented:
  - most imperative languages
- How about C, C++ and Java?
  - halfway in-between

Expression-oriented Languages


```
1. print( IF  b < c THEN d ELSE e  FI);
2. IF  b < c THEN d ELSE e  FI := a;
3. 2 + 3
4. REAL x := 2.5;
   x +:= 3.0 *:= 4.0;
```

Assignment as an Expression


```
#include <iostream>
using namespace std;

int main() {
     int x = 7, y = 3;
     (x < y ? x : y) = 2;
     cout << x << endl;
     cout << y << endl;
     return 0;
}
```

Assignment Statements

- The general syntax

  `<target_var> <assign_operator> <expression>`

- The assignment operator

  = FORTRAN, BASIC, the C-based languages

  := ALGOLs, Pascal, Ada

No Boolean Type in C


```
int z = 7, y=10, x =20;
if(x > y > z)
    cout << "True"<<endl;
else
    cout << "False"<<endl;
```

Multiway Assignment

Perl, Python, ML, CLU and Ruby support list assignments:

- Allows the following:

```
a, b = c, d
($first, $second, $third) = (20, 30, 40);
```

Multiway Assignment


```
#include <iostream>
using namespace std;

int main()
{
   int x=4, y=5;
   x , y = 44, 55;

   cout<< x <<endl;
   cout<< y <<endl;
   return 0;
}
```

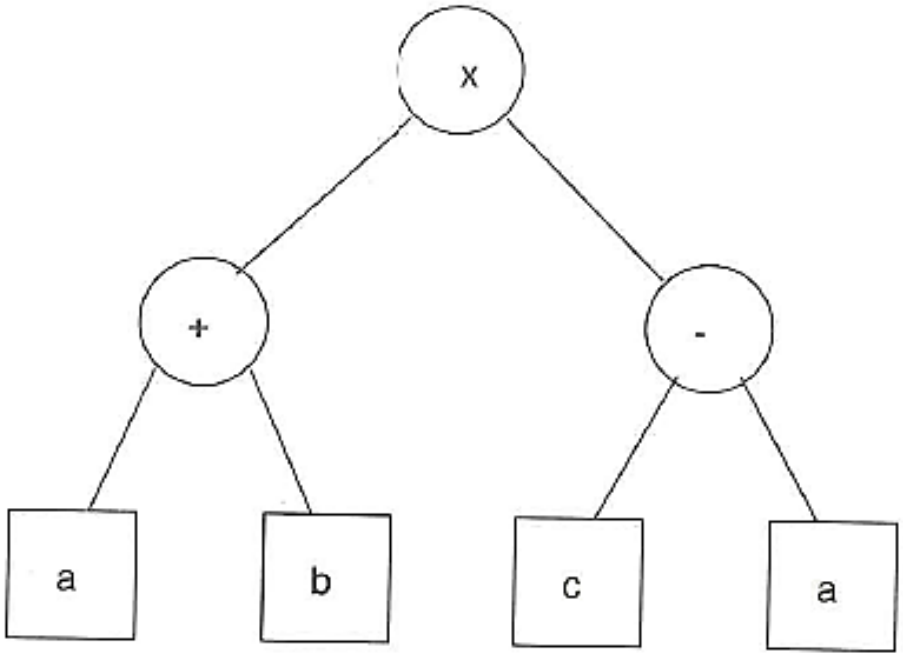
Multiway Assignment – Go Example


```
package main

import "fmt"

func swap(x, y string) (string, string){
    return y, x
}

func main(){
    x:= "hello"
    y:= "world"
    a, b := swap(x, y)
    fmt.Println(x, y)
    fmt.Println(a, b)
}
```

Definite Assignment


```
int i;
int j = 3;

if (j > 0) {
    i = 2;
}
if (j > 0) {
    System.out.println(i);
}
```

Operand Evaluation

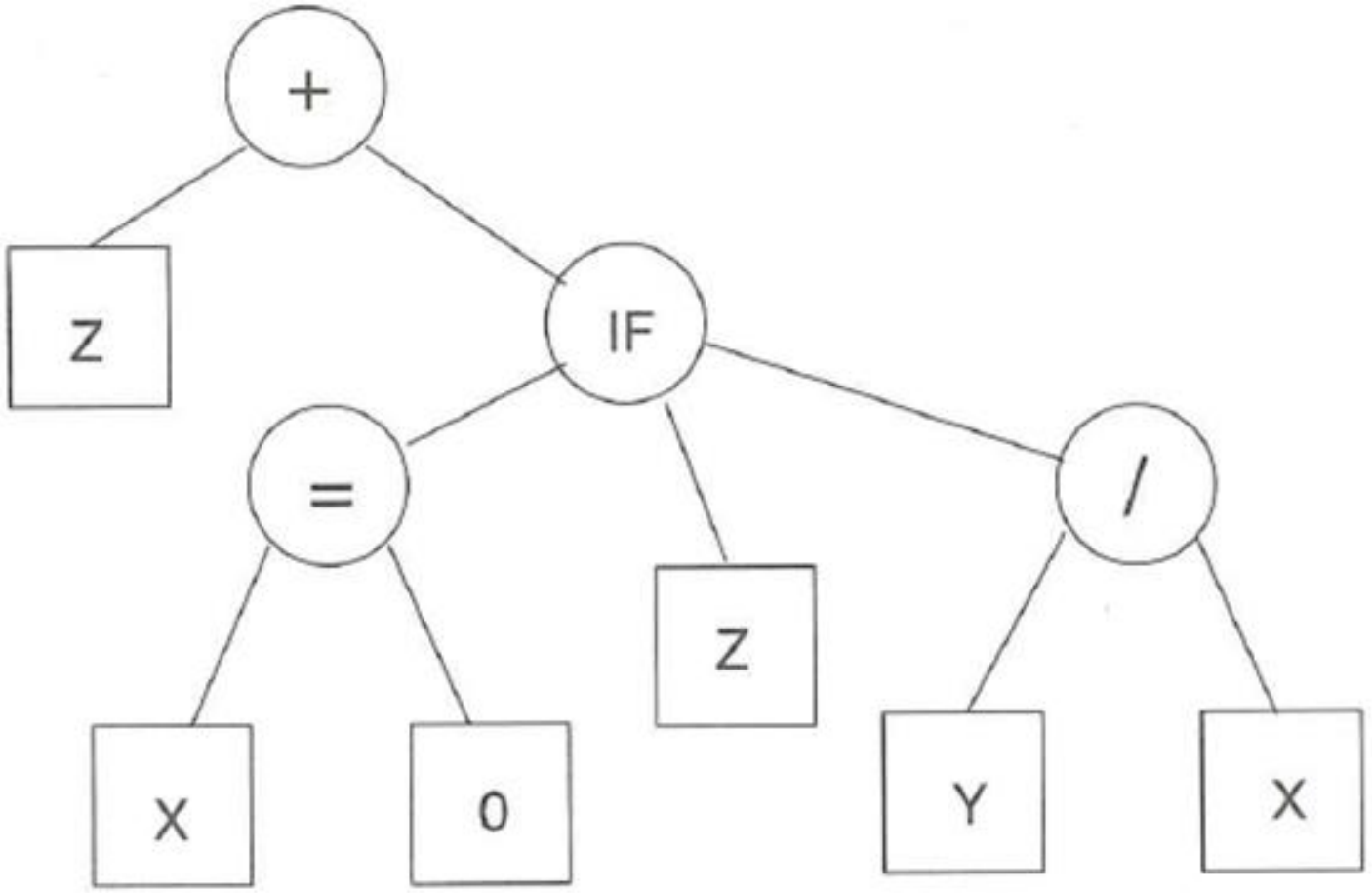


From: Terrence W. Pratt and Marvin V. Zelkowitz, Programming Languages: Design and Implementations.

Operand Evaluation

- Are we going to evaluate all operands or not?
  - Eager
  - Short circuit
  - Lazy
- Is the order of evaluation important?

Operand Evaluation



From: Terrence W. Pratt and Marvin V. Zelkowitz, Programming Languages: Design and Implementations.

Short Circuit Evaluation

- Examples:

```
1. if (a == 0 || b/a > c) ...
2. if (p && p->foo) ...
3. while (i <= UB && v[i] > c) ...
```

Short Circuit Evaluation

- Do all programming languages support short-circuit evaluation of Boolean operations?

- Pascal: No
- C, C++, and Java:
  - `&&` and `||`: short-circuit evaluation but also provide
  - `&` and `|`: are not short circuit
- Ada:
  - `If (A=0) or else (B/A > C) then ...`
  - `If (A/=0) and then (B/A > C) then ...`

Lazy Evaluation


```
(define x 5)
(define y 7)
(define f (lambda (a b c)
            c))
(f 4 5 (delay (+ x y)))
```

Side Effects


```
a = a * fun(x) + a;
```

Side Effects


```
#include <iostream>
using namespace std;
int a;
int fun (int y)
{
   a*=y;
   return 3;
}
int main ()
{
   a =1;
   int x=4;

   a = a * fun(x) + a;
   cout << a << endl;

   return 0;
}
```

Side Effects


```
int subtract (int a, int b)
{
    return a - b;
}
int main ()
{
    int x = 0;
    int y = subtract (++x , x++);
    cout <<"y = "<< y;
    return 0;
}
```

Side Effects


```
class effects
{
public static void main(String[] args)
{
        int x = 3;
        print (x, x++);

        System.out.println(x + ++x);
}
static void print(int i, int j)
{

        System.out.println(i);
        System.out.println(j);

}
}
```

Side Effects

- Are side effects allowed in Programming Languages?
  - Side effects should not be allowed
  - It's OK to have side effects
    - Do we restrict the evaluation order

References

- Michael L. Scott, Programming Language Pragmatics, Morgan Kaufmann, 3rd edition, 2009.
- Robert W. Sebesta, Concepts of Programming Languages, Addison Wesley, 10th edition, 2012.
- Terrence W. Pratt and Marvin V. Zelkowitz, Programming Languages: Design and Implementations, Prentice Hall, 4th edition, 2001.