Control

• Control: what gets executed, when, and in what order.

• Abstraction of control:
  – Expression
  – Statement
  – Exception Handling
  – Procedures and functions

Expression vs. Statement

• In pure (mathematical) form:
  – Expression:
    • no side effect
    • return a value
  – Statement:
    • side effect
    • no return value
• Functional languages aim at achieving this pure form
• No clear-cut in most languages

Expression

• Constructed recursively:
  – Basic expression (literal, identifiers)
  – Operators, functions, special symbols
  – Number of operands:
    – unary, binary, ternary operators
  – Operator, function: equivalent concepts
    – (3+4)*5
    – mul( add(3,4), 5)
    – *[+ 3 4] 5
    – **(+(3,4))
  (Ada, prefix notation)
  (LISP, prefix notation)

Postfix notation

• PostScript:
  – /Courier findfont
  – 20 scalefont
  – setfont
  – 72 500 moveto
  – (Hello world!) show

http://en.wikipedia.org/wiki/PostScript

Expression and Side Effects

• Side Effects:
  – changes to memory, input/output
  – Side effects can be undesirable
  – But a program without side effects does nothing!
• Expression:
  – No side effect: Order of evaluating subexpressions doesn’t matter (mathematical forms)
  – Side effect: Order matters
Applicative Order Evaluation (Strict Evaluation)

• Evaluate the operands first, then apply operators (bottom-up evaluation)
  (subexpressions evaluated, no matter whether they are needed)

• But is 3+4 or 5-6 evaluated first?

Order Matters

C:

```c
int x=1;
int f(void) {
    x=x+1;
    return x;
}
main(){
    printf("%d\n", x + f());
    return 0;
}
```

Java:

```java
class example {
    static int x = 1;
    public static int f() {
        x = x+1;
        return x;
    }
    public static void main(String[] args) {
        System.out.println(x+f());
        return 0;
    }
}
```

Many languages don’t specify the order, including C, java.

– C: usually right-to-left
– Java: always left-to-right, but not suggested to rely on that.

Expected Side Effect

• Assignment (expression, not statement)
  \( x = (y = z) \) (right-associative operator)

• \( x++, ++x \)
  ```c
  int x=1;
  int f(void) {
      return x++;
  }
  main(){
      printf("%d\n", x + f());
      return 0;
  }
  ```

Sequence Operator

• \((\text{expr}_1, \text{expr}_2, ..., \text{expr}_n)\)
  – Left to right (this is indeed specified in C)
  – The return value is \(\text{expr}_n\)

```c
x=1;
y=2;
x = (x=x+1, y++, x+y);
printf("%d\n", x);
```

Non-strict evaluation

• Evaluating an expression without necessarily evaluating all the subexpressions.

• short-circuit Boolean expression
  • if-expression, case-expression

Short-Circuit Evaluation

• \( \text{if} (\text{false} \text{ and } x) \ldots \text{if} (\text{true} \text{ or } x) \ldots \)
  – No need to evaluate \(x\), no matter \(x\) is true or false

• What is it good for?
  – if \(i <= \text{lastindex} \text{ and } a[i] >= x)\ldots
  – if \(p != \text{NULL} \text{ and } p->\text{next}==q)\ldots

• Ada: allow both short-circuit and non short-circuit.
  – if \(x ~/= 0\) and \(y ~/= 0\) then \(x ~/= 0\) and \(y ~/= 0\) then ...
  – if \(p->\text{next}==q)\ldots
if-expression

- if (test-exp, then-exp, else-exp)
  - ternary operator
  - test-exp is always evaluated first
  - Either then-exp or else-exp are evaluated, not both

- if e1 then e2 else e3  (ML)
- e1 ? e2 : e3  (C)

- Different from if-statement?

case-expression

- ML:
  - case color of
  - red => “R” |
  - blue => “B” |
  - green => “G” |
  - _ => “AnyColor”;

Normal order evaluation (lazy evaluation)

- When there is no side-effect:
  - Normal order evaluation (Expressions evaluated in mathematical form)
  - Operation evaluated before the operands are evaluated;
  - Operands evaluated only when necessary.

- int double (int x) { return x+x; }
- int square (int x) { return x*x; }

  Applicative order evaluation : square(double(2)) = ...
  Normal order evaluation : square(double(2)) = ...

Examples

- Call by Name (Algol60)
- Macro

  #define swap(a, b) {int t; t = a; a = b; b = t;}

  - What are the problems here?
Statements

- If-statements, case-(switch-)statements, loops