Expressions & Statements

expression – pure mathematical form
statement – imperative form
– side effects

Structured Control

two is enough: while B do S
if B then S [else S]
no other statements needed (proven)
— but inconvenience
– single entry single exit blocks

Expressions

operators: unary prefix ++x
unary postfix x++
binary x+y
ternary x?y:z

precedence (a*b)+c
associativity (a+b)+c left, a+(b+c) right

functions: names with arguments
low level: operators are m/c instns
fns are called

syntactically: same
Ocaml: a+b ≡ ((+) a b)
Scheme: all same
Haskell: f a b ≡ (a `f` b)

applicative order eval

```
let sq x = x*x
let dbl x = x+x
```

```
sq(dbl(2)) = sq(2+2)
           = sq(4)
           = 4*4
           = 16
```

normal order

```
sq(dbl(2)) = dbl(2)*dbl(2)
           = (2+2)*(2+2)
           = 4*4
           = 16
```

Evaluation order

```
int x=1
f(){
    x+=1
    ret x
}
g(a,b){
    ret a+b
}
main(){
    y=g(x,f(x))
}
```

```
rax = x //1
rdx = f(x) //2        L → R
y = g(rax,rdx) //3
```

```
push f(x) //2         R → L
push x    //2
y = call g //4
```

So: 3 or 4

Java strict L → R

C C++ no.

a[i++] = b[i++] ~~effect~~ ??

~~short circuit (lazy) evaluation~~

~~a && b = a ? b : false~~

~~a || b =~~

Short Circuit (lazy) eval

```
y = a && b      t1 = a; if (t1) y = b else y = false
y = a || b      t1 = a; if (t1) y = true else y = b
  a ? b : c   (ternary)
```

Conditionals & Guards

```
if B then S else S
```

guards

```
if B1 → S1
 | B2 → S2
 | B3 → S3
  :
fi
```

} like Scheme (cond)
Dijkstra's guarded commands

dangling else

```
if B1 then if B2 then S1 else S2
```

conventionally else matches closest if before it not already with an else

```
switch (n) {
  case: break
  case: break
    :
  default
}
```

} selects n alternatives in O(1) time
equivalent if else if else if ...
takes O(n) time

Loops

while B do S

guards

```
do B1 → S1
 | B2 → S2
 | B3 → S3
   ⋮
od
```

Dijkstra's guarded cmds. executed until all guards are false

```
do S while B
for(i;?;step) S
for(i in A) S
```

} convenient "syntactic sugar" for while

`for(a;b;c) S ⟹ a;while(b){S;c}`

`for(i:s) f(i) ⟹ for(i=s.begin();i!=s.end();++i) f(*i)`

Goto controversy

EWD Goto Statement considered Harmful
CACM 1968
www.acm.org/classics/oct95

Donald Knuth: Structured programming with the Goto.
Computing Surveys, Dec 1974

---

convenience stmts: continue, break, return

special kinds of goto

Exceptions

controlled non-local gotos

```
try {
  :
}catch(E){
}catch(E){
  :
}finally{
}
```

1. exception thrown
2. goto forward nearest exn handler that matches
3. if handler contains a throw;
   rethrow same exn
4. always execute finally
5. if no exn handler,
   return and rethrow in caller
6. if no handler before main
   abort()

In C: setjmp()
longjmp()