"Sisyphus" by Frank Williams (b. 1947), USA
2003. Mixed media on canvas. 143x195cm.
In Greek myth, Sisyphus was condemned to roll a boulder up hill over and over again.

Statements

- Control flow
- Iteration and recursion
- Iterators, generators in Icon
- Recursion
- Nondeterminacy

Exception handling does have an impact on the flow of control; we cover this topic elsewhere.

Definition

A *statement* is a construct of a programming language that

1. performs some basic action, or
2. governs the flow of control.

Definition

A *statement* is a construct of a programming language that

1. performs some basic action, or
2. governs the flow of control.

The basic actions of a program are

1. the assignment to store (discussed earlier), and
2. input/output (about which we have not in particular to say).

Control flow

Definition

A computer program directs the computer to execute a stream of instructions from among many possible sequences of instructions. This directed, continuous stream of actions is the *control flow*.

Normally, each statement–left to right, top to bottom, like reading a book in English–in the program is executed one after the other. This is *sequential* flow of control.

Unstructured

The basic computer instruction that departs from the sequential flow of control is the goto. The conditional goto instruction is the essential way in which computer hardware comes to a decision. As such it is indispensable to computation.

```
        ...
        if (A == 0) goto 10
        ...
10      ...
```


EDSGER W. DIJKSTRA
Fundamental contributions to programming as a high, intellectual challenge.
acm
A.M.
TURING
AWARD
1972

Edsger Wybe Dijkstra (1930–2002)

His 1969 letter to the editor "Go to statement considered harmful," in the CACM ignited the `goto` controversy

His 1975 book *A Discipline of Programming*, was the manifesto of the structured revolution which demonstrated it was possible to express computation with human-centric constructs and not tedious machine instructions.

Edsger Wybe Dijkstra (1930–2002)

(I have always heard his name pronounced "dike-stra," and I have been told this is close to the Dutch pronunciation.)

Dijkstra was born in Rotterdam, Netherlands in 1930. ACM Turing Award in 1972, the 1982 IEEE Computer Pioneer Award. He held the Schlumberger Centennial Chair in Computing Sciences at the University of Texas at Austin, 1984-1999, and retired as Professor Emeritus in 1999. Died 6 August 2002 at his home in Nuenen, the Netherlands.

Come From

```
10  J = 1
11  COME FROM 20
12  WRITE (6,40) J
    STOP
13  COME FROM 10
20  J = J+2
40  FORMAT (I4)
```

INTERCAL

This was picked up in INTERCAL

```
(1) PLEASE
    .
    .
    .
(2) DO COME FROM (1)
```

which has included other dubious features including output in Roman Numerals and the “ignore” variables.

Eliminating GOTO

> *"At the IFIP Congress in 1971 I had the pleasure of meeting Dr. Eichi Goto of Japan, who cheerfully complained that he was always being eliminated."*

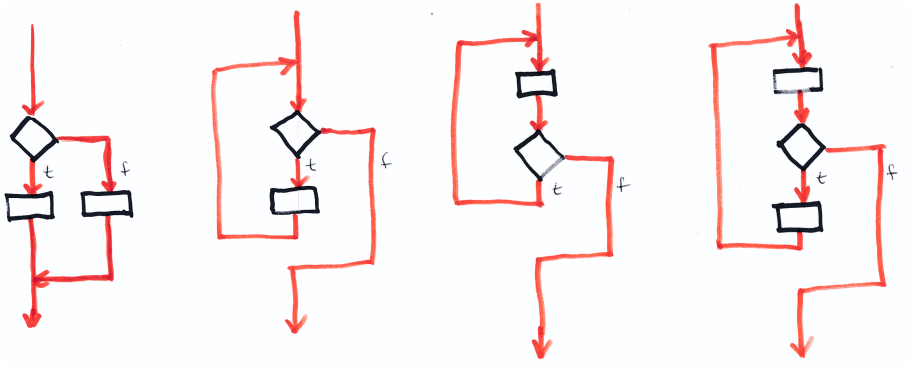
Donald Ervin Knuth, "Structured programming with go to statements," *Computing Surveys*, 1974.

Cartoon by Alan Guzman
Florida Tech, 2019

Structured Programming

Compound statements (statements containing other statements) act to divert the control flow and serve a purpose like parentheses in expressions.

“Flaw” Charts



Structured Programming

if, **while**, **do–until**

Single entry point and single exit point. Note also the case/switch statement.

```
loop
   exit when [c]
   ...
   ...
end loop;
```

```
loop
   ...
   ...
   exit when [c]
end loops;
```

```
loop
   ...
   exit when [c]
   ...
end loop;
```

Structured Programming

Even the most convoluted control rarely calls for a goto.

```
loop            for (;;) {
   ...              ...
   exit;            break;
   ...              ...
end loop;       }
```

Midtest Loops

Modula (the precursor to Modula-2) introduced a mid-test loop with one or more exit points as a supplement to the pretest (while) and post-test (repeat-until) loops.

5.13 Loop statements

Loop statements specify the repeated execution of st
sequences. The repetition can be terminated depending
values of possibly several Boolean expressions, calle
conditions.

```
loopstatement = "loop" statementsequence
    {"when" expression ["do" statementsequence] "exit"
     statementsequence } "end".
```

Wirth, “Modula”, 1976 page 18

Midtest Loops

```
LOOP
     ⟨statement list⟩0
WHEN ⟨condition⟩1 EXIT;
     ⟨statement list⟩1
WHEN ⟨condition⟩2 EXIT;
     ⟨statement list⟩2
⋮
END;
```

In Modula-2 this gave way to the less structured, but more convenient `EXIT` statement. A loop may have several exit statements.

The exit statements may be nested in other control constructs (WHILE, CASE, IF, etc.), but they break out of only the most deeply nested LOOP statement.

```
LOOP
    ⟨statement list⟩0
    IF ⟨condition⟩1 THEN EXIT END;
    ⟨statement list⟩1
END;
```

Ada has an exit-when statement that breaks out of any nested loop.

```
l: loop
    ⟨statement list⟩0
    exit l when ⟨condition⟩1;
    ⟨statement list⟩1
end loop l;
```

Java is much the same but has no special loop construct; one must use the `while` or the `for` loop.

```
l: for (;;) {
    ⟨statement list⟩0
    if ⟨condition⟩1 break l;
    ⟨statement list⟩1
}
```

Without a label `break` (and `continue`) in Java refer to the most closely enclosing `for`, `while`, `do`, or `switch` statement.

Non-local control:

- goto
- multi-level break [just mentioned]
- exception handling [later]
- multi-level return; call with continuation [omitted]

Structured Programming

goto/main.adb
goto/alt.adb
goto/fun.adb

Bohm-Jacopini said all you needed was one while loop. This is approach to structuring a programming is more likely to obscure than improve it.
Requiring all loops to have a single exit may write local variables and also code publication. Studies who programming complex interaction in Pascal is more error prone.
Kosaraju in 1973 proved that it's possible to avoid adding additional variables in structured programming, as long as arbitrary-depth, multi-level breaks from loops are allowed. But duplicating some come may be necessary.

Iteration

Modula-2

```
FOR i := first TO last BY step DO
   ...
END
```

Most languages, including Algol 68, Pascal, Ada, Fortran 77 and 90/95, and Modula-3 prohibit changes to the loop index within the body of the loop. In Ada, step is 1 or -1.
Tip: Use a for loop when you know the number times a block of code needs to be repeated.

Iterators

Sebesta, "Section 8.4.4 Iteration Based on Data Structures," pages 336–337. A Perl example:

```
@names = {"Bob", "Carol", "Ted", "Alice"}
foreach $name (@names) {
   print $name;
}
```

A Python example of iterating over a list:

```
names = ["Bob", "Carol", "Ted", "Alice"]
for name in names:
    print name
```

Iterators

Clu examples, Scott, Figure 6.5, page 288 and Figure 6.6, page 289.

Iterators

Idiom for stepping through all the key-value pairs in a Java Map collection:

```
for (Iterator i=m.entrySet().iterator();i.hasNext();){
    final Map.entry e = (Map.Entry) i.next();
    System.out.printf ("%s: %s%n", e.getKey(), e.getValue());
}
```

With the enhanced `for` loop introduced with in the revised Java language introduced in 2004:

```
for (Map.entry e: m.entrySet()) {
    System.out.printf ("%s: %s%n", e.getKey(), e.getValue());
}
```

Icon Generators

Griswold, et al, "Generators in ICON," *TOPLAS*, 1981.

```
... to ... by ...
upto (chars,str) # all positions at which any char in chars appears
find (substr,str)


every i := first to last by step do { write(i) }
every i:=1 + upto(' ',s) do { write(i) }
every write (1+upto(' ',s))

if 2>6 then { ... }
if (i:=find(``ab'',s))>6 then { ... }
```

Python Generators

```
def gensquares(N):
    for i in range(N):
        yield i ** 2

for i in gensquares(5):
    print i, "; "
```

List Comprehension

List Comprehension Python (also in Haskell)

```
[expression for name in sequence [if conditional] ...]

>>> [x**2 for x in range(10)]

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> noprimes = [j for i in range(2,8) for j in range(2*i, 50, i)]
>>> primes = [x for x in range(2,50) if x not in noprimes]
>>> print primes
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]

composite = [j | i <- [2..8], j <- [2*i, 3*i .. 50]]
primes = [x | x <- [2..50], not (x `elem` composite)]
```

Proposed Ada 202x Map/Reduce Iterators for parallelism

```
--  Reduction expression to sum array elements
Result : Integer :=
   [for Element of Arr => Element]'Reduce("+", 0);

--  Create an unbounded string containing the alphabet
Alphabet : Unbounded_String :=
   [for Letter in 'A'..'Z' => Letter]'
      Reduce ("&", Null_Unbounded_String, "&");

--  Determine how many people in a database are over 30
ThirtySomethings : constant Natural :=
   (for P of Personnel => (if Age(P)>30 then 1 else 0)]'
      Reduce("+", 0);
```

Proposed Ada 202x Map/Reduce Iterators for parallelism

```
parallel
  P := Compute_Prime (Bits=>4096, RNG=>"/dev/random");
and
  P := Compute_Prime (Bits=>4096, RNG=>"/dev/random");
end parallel
parallel
  N := P*Q;
and
  Phi := (P-1) * (Q - 1);
end parallel
parallel
  Public_Key := Make_Public_Key (N, Phi)
and
  Private_Key := Make_Private_Key (N, Phi)
end parallel
```

Recursion

By iteration we mean loops (while, for) as opposed to recursion.

```
int gcd(int a, int b) {
   if (a==b) return a;
   else (a>b) return gcd(a-b,b);
   else return gcd(a,b-a);
```

```
int gcd(int a, int b) {
   start:
   if (a==b) return a;
   else if (a>b) {
      a=a-b; goto start;
   } else {
      b=b-a; goto start;
   }
}
```

Recursion

Theory: Don't need iteration, recursion is enough.
Practice: Can be as efficient as iteration *tail recursion*. (Sebesta, 9th ed., Section 15.4, page 660.) Optimizing transformations like *continuation-passing style* is an advanced topic in functional programming.

> *We really did not understand the implications of recursion, or its value, at the ALGOL 60 meeting in Paris. McCarthy did, but the rest of us didn't.*
>
> Perlis, 1978

Nondeterminacy

Dijkstra, "Guarded commands, nondeterminacy and the formal derivation of programs", 1975.
Compute the maximum:

if $x \geq y \rightarrow m := x$
$[]$ $y \geq x \rightarrow m := y$
fi.

Compute the gcd:

$x := X; y := Y;$
do $x > y \rightarrow x := x - y$
$[]$ $y > x \rightarrow y := y - x$
od.

Used in Hoare's CSP and for concurrency in Occam, Ada and SR.

Parallelism

FORALL a HPF (High-Performance FORTRAN) extension to Fortran 90 found in the Fortran 95 standard.

```
FORALL (i=1:size, j=i:size)
  hilbert(i,j) = 1.0 / real (i+j+1)
END FORALL
```

Sebesta, seventh edition, section 13.9.1, page 574.

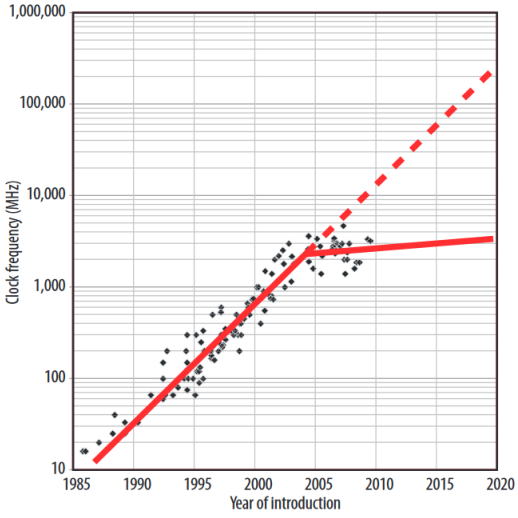
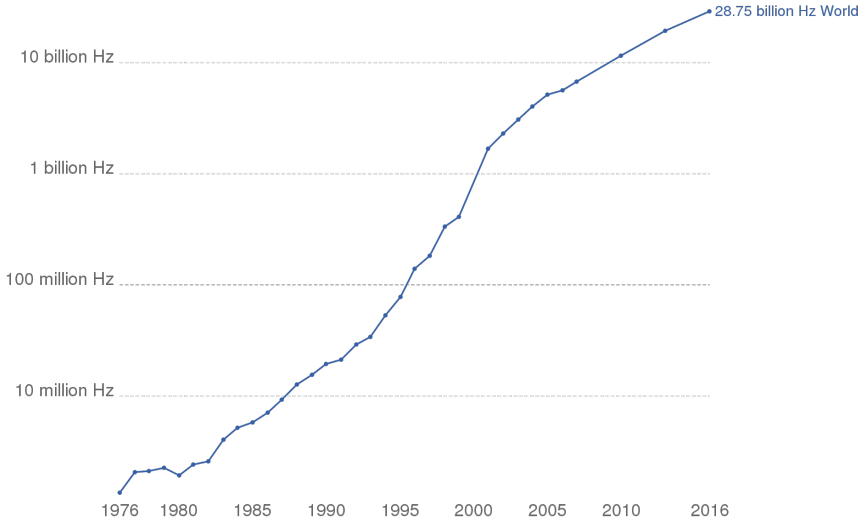


Figure 2. Historical growth in single-processor performance and a forecast of processor performance to 2020, based on the ITRS roadmap. A dashed line represents expectations if single-processor performance had continued its historical trend.

Microprocessor clock speed

Microprocessor clock speed measures the number of pulses per second generated by an oscillator that sets the tempo for the processor. It is measured in hertz (pulses per second).



Source: Ray Kurzweil (2005, updated to 2016). The Singularity Is Near: When Humans Transcend Biology.

Significance of Nondeterminacy

Usually programming languages prescribe exactly the order of execution for all statements even though the algorithm may not depend on the order.

```
X := 3;  Y := 7;  // one order
Y := 7;  X := 3;  // another
```

Computer hardware today is highly parallel and it requires great effort on the part of compiler to find some of the parallelism hidden in the unnecessarily, over-constrained program sequence.

Significance of Nondeterminacy

Propose Ada 202x parallel block statement

```
parallel do
   X := 3;
and
   Y := 7;
end do;
```

Significance of Nondeterminacy

Proposed Ada 202x parallel loop

```
parallel for I in 1 .. 1_000 loop
   A(I) := B(I) + C(I);
end loop;

parallel for Elem of Arr loop
   Elem := Elem * 2;
end loop;
```

Pure functional and logic programming languages have no sequentially (with applies no opportunity to observe side effects), and so have limitless opportunity for parallelism.

Miscellaneous

Exception handling, see Sebesta, seventh edition, Chapter 11.
Call-backs, Java GUI event handling, see Sebesta, seventh edition, Chapter 11.
Where is the main? Applets (run by the browser), serverlets (run by the server), GUI programs (reactive to user gestures).

Review

Scott 6.9, Review Questions