Algol Part 3

CS4100
March12, 2012

Structured Programming

- Compound statements drastically reduce the number of GOTOs required
  - In Fortran, GOTO was the workhorse for control
  - Example: *if-then-else*
- GOTO-less programs were easier to read
  - This led people to experiment with abolishing GOTO
  - Dijkstra: "Go To Statement Considered Harmful"
    - Difficulty in reading programs came from conceptual gap between static and dynamic structure of program
    - i.e.: static layout on paper, versus runtime operation
    - Result: languages still have GOTOs, but we don't use them

Principles of Programming

- The Structure Principle
  - The static structure of the program should correspond in a simple way to the dynamic structure of corresponding computations.

Procedures are Recursive

- Recursive definitions are frequent in math and science
  - Define thing in terms of itself
  - Example:
    - Factorial: n! =
      n * (n – 1)! if n > 0
      1 if n = 0
- Algol permits recursive procedures

```
integer procedure fac(n);
  value n; integer n;
  fac := if n = 0 then 1 else n*fac(n-1);
```

  - 'n = 0' is called the stopping condition

Implementing Recursion

- What happens to local variable *n* on recursive call?
  - fac(3) is called, then fac(2), then fac(1), then fac(0)
  - Would location holding 3 be overwritten?
    - Yes, if same activation record was used
  - Solution:
    - Create new activation record for each invocation of *fac*()

Parameter Passing

- Modes in Algol
  - Pass by value
  - Pass by name
- Two modes attempt to distinguish between *input only* and *input/output* parameters

Pass by Value

```
integer procedure fac(n);
  value n; integer n;
```

- First part of pass by value-result (in Fortran)
  - Actual copied into variable corresponding to formal
  - Secure; local variable will not overwrite actual parameter
  - Does not allow output parameters (input only)
  - Inefficient for arrays (or other non-primitive data structures, in general)
    - Copy must be made of entire array in activation record
    - Copying takes time

7

Pass by Name

- Based on substitution
  - Consider

```
integer procedure Inc(n);
  integer n;
  n := n + 1;
```

  - And the call `Inc(i)`
- We need output parameter that will effect `i`, not just local `n`
  - Acts like `i` is substituted for `n`

```
i := i + 1
```

8

Copy Rule

- Procedure can be replaced by its body with actuals substituted for formals
- Revised Report 4.7.3
- Body of `Inc(n)`
  - `i := i + 1`
  - `A[k] := A[k] + 1`
- Not how it is implemented

9

Pass by Name is Powerful

- Evaluate the following using pass by value, reference, and name

```
procedure S(el,k);
  integer el, k;
  begin
        k := 2;
        el := 0;
  end
A[1] := A[2] := 1;
i := 1;
S(A[i], i)
```

- Value A[1] = 1, A[2] = 1, i = 1
- Reference A[1] = 0, A[2] = 1, i = 2
- Name A[1] = 1, A[2] = 0, i = 2

10

"Thunks"

- Implementing pass by name
  - Passing the text?
    - Would need to compile at runtime
      - not possible
  - Copying compiled code?
    - Would increase size of code…
  - Solution: "Thunks"
    - Pass address to compiled code
    - Address of memory location is returned to callee to use as variable

11

Pass by Name is Dangerous!

```
procedure Swap(x, y);
  integer x, y;
  begin integer t;
        t := x;
        x := y;
        y := t;
  end
```

- What is the effect of
  - `Swap(A[i], i)?`
  - `Swap(i, A[i])?`

12

- Swap(r,s), where r=1,s=2

```
procedure Swap(x, y);
  integer x, y;
  begin integer t;
        t := r;          t=1
        r := s;          r=2
        s := t;          s=1
  end
```

13

- Swap(A[i], i) where A[i]=27, i=1

```
procedure Swap(x, y);
  integer x, y;
  begin integer t;
        t := A[i];   t=27
        A[i] := i;   A[1]=1
        i := t;      i=27
  end
```

14

- Swap(i, A[i]), where i=1, A[i]=27

```
procedure Swap(x, y);
  integer x, y;
  begin integer t;
        t := i;      t=1
        i := A[i];   i=27
        A[i] := t;   A[27]=1
  end
```

15

Pass-by-name

- It can be shown that there is no way to define swap in Algol-60 that works for all parameters
- Design mistake when a simple (common) procedure has such surprising properties

16

Parameter Passing Modes

- Pass by value
  - Bind to value at time of call
  - Preserves actual (no output parameters)
  - Inefficient for arrays
- Pass by reference
  - Bind to address at time of call
  - Changes actual (can be used for output)
  - Efficient for all data types
- Pass by name
  - Bind to address of thunk at time of call
  - Changes actual (can be used for output)
  - Efficient, but expensive

17

Out-of-Block GOTOs

```
A: begin array x[1:100];
        ...
   B:  begin array y[1:100];
          ...
       goto exit;
        ...
       end;
exit:
   end
```

- What happens to activation records?
  - Program continues in different block

18

Even worse…

```
begin
  procedure P(n);
  value n; integer n;
         if n = 0 then goto out
         else P(n-1);
   P(25);
out:
end
```

- Indefinite number of activation records…

19

Feature Interaction

- Example:
  - GOTOs are simple
  - Recursion is simple
  - Combination is very messy
- In theory, each feature must be tested with every other one to avoid unintended consequences
- 100 features:
  - Every pair: 100x100 = 10,000 combinations
  - Every three: 100^3 = 1,000,000
  - …

20

The *for*-loop is Very General

for *var* := *exp* **step** *exp2* **until** *exp3* **do** *stat*
for *var* := *exp* **while** *exp2* **do** *stat*

- Expressions can be any arithmetic expression, including
  - **for** i := i/2 **while** i>1 **do** *stat*
  - Lists
    - **for** days := 31, 28, 31,30, 31, 30 **do** *stat*
  - Conditional expressions (vs. conditional statements!)
    - **for** days := 31,
      if mod(year, 4) = 0 **then** 29 **else** 28,
      28, 31, 30, 31, 30 **do** *stat*
  - Combinations of above
    - **for** i := 3, 7,
      i/2 **while** i>1,
      11 **step** 1 **until** 16
      do *stat*

21

- <for statement> ::= <for clause> <statement> |
  <label>: <for statement>
- <for clause> ::= **for** <variable> := <for list> **do**
- <for list> ::= <for list element> | <for list> , <for list element>
- <for list element> ::= <arithmetic expression> |
  <arithmetic expression> **step** <arithmetic expression>
  until <arithmetic expression> |
  <arithmetic expression> **while** <Boolean expression>

- **for** q:=1 **step** s **until** n **do** A[q]:=B[q]
- **for** k:=1,V1x2 **while** V1<N **do**
  for j:=I+G,L,1 **step** 1 **until** N, C+D
  do A[k,j]:=B[k,j]

22

Baroque Features

- Fascination-oriented features of little use
  - They did it because they could
  - Getting away from assembly languages as far as possible
- Baroque takes on pejorative meaning

23

Baroque

- 1 : of, relating to, or having the characteristics of a style of artistic expression prevalent especially in the 17th century that is marked generally by use of complex forms, bold ornamentation, and the juxtaposition of contrasting elements often conveying a sense of drama, movement, and tension
- 2 : characterized by grotesqueness, extravagance, complexity, or flamboyance
- 3 : irregularly shaped —used of gems <a baroque pearl>
- baroque. (2009). In Merriam-Webster Online Dictionary. Retrieved April 28, 2009, from http://www.merriam-webster.com/dictionary/baroque

24

Handling Cases: `switch`

```
begin
   switch wageStatus = fulltime, parttime, hourly;
   ...
   goto wageStatus[i];
fulltime:       ...handle fulltime case...
                goto done;
parttime:       ...handle parttime case...
                goto done;
hourly:         ...handle hourly case...
                goto done;
done:   ...
end;
```

- Elaboration on computed GOTO of Fortran (and IBM 650)
- Confusing, since switch, goto, and labels can be anywhere in the program
- Label list can contain conditionals (`if i>0 then M else N`)

25

Machine Independence

- Get away from formats tied to particular computers, punch cards -> free format
- How should a program be formatted?
  - Left justify, one statement per line
  - Like English sentence
  - Structured (hierarchical)
    - Obeys structure principle
- Most languages followed Algol in free format

26

Machine Independence

- Representation issues
  - Different hardware
    - Input devices
    - Character sets
  - Different conventions
    - Math vs cs
    - American vs European
    - Comma (European) vs point (American) almost defeats Algol

27

Machine Independence

- Theorem: The more trivial the point the more vehemently people will fight over it
- Which symbols
  - Only those available in all sets
    - Too limiting
  - Independent of particular sets
    - chosen

28

Compromise

- Three representations
  - Reference language used in language specifications
    - E.g. "up arrow"
  - Publication language used in publications
    - E.g. sub- and super-scripts
  - Hardware language to be used by implementers
    - Use appropriate character set
    - I/O for the computer system

29

Lexical Conventions

- Reserved words
  - Cannot be used as identifiers
  - Most languages
- Key words
  - Words used by language are marked
    - E.g. Different font or bold
    - Hard to type
  - Algol
- Keywords in context

30

Keywords in context

- FORTRAN
- Words used by language are only keywords in context where expected
  - Hard to catch errors
- Legal in PL/I

```
IF IF THEN
    THEN = 0;
ELSE
    ELSE = 0;
```

31

Some Design Considerations

- From David Billington, *The Tower and the Bridge* 1993
- Techological Activities
  - Values
    - Efficiency
    - Economy
    - Elegance
  - Dimensions
    - Scientific
    - Social
    - Symbolic

32

Efficiency

- Materials used
- Scientific Issue
- Memory
- Time
  - Programmer
  - Compiler
  - Run

33

Economy

- Cost-benefit
- Social Issue
- Benefit to programming community
- Cost: trade-offs
  - Computer vs programmer time
  - Increasing cost of residual bugs
  - Program maintenance vs development

34

Economy

- Social Influences
  - Manufacturer support
  - Prestigious universities teach
  - Approved by influential organizations
  - Standardized
  - Used by "real" programmers
- Monetary values are unstable as is social climate

35

Elegance

- Under-engineered
  - Risk of unanticipated interactions
- Over-engineered
  - Inefficient or uneconomical
- Can't always rely solely on mathematical analysis
  - Always incomplete
    - Simplifications
    - assumptions

36

Elegance

- General Principle: Designs that look good are good
- Function follows form
  - But needs to be deep (not superficial)
- Should be a joy to use
  - Comfortable and safe

37

Elegance

- Aesthetics comes from experience
- Design obsessively
  - Criticize
  - Revise
  - Discard

38

In Summary, Algol

- Never had widespread use
  - No I/O
  - Competing directly with FORTRAN
- Major milestones
  - Block-structured
  - Nested
  - Recursive
  - Free-form
  - BNF - mathematical theory of formal languages

39

Algol by reputation

- General
- Regular
- Elegant
- Orthogonal

40

Second Generation

- Elaborations and generalizations of first generation
  - Strong typing of built-in types
  - Name structures hierarchically nested
  - Structured control structures
    - Recursion
    - Parameter passing
  - Syntactic structures
    - Machine independent
    - Moving away from fixed formats

41