Procedural and Structured Programming

Michael McThrow

San Jose State University
Computer Science Department
CS 152 – Programming Paradigms

August 24, 2020

Table of Contents

Table of Contents

Procedural Paradigm

The procedural paradigm of programming emphasizes step-by-step instructions that are generally[1] executed sequentially.

[1] I say "generally" because there's out-of-order instruction execution on modern CPUs.

Instruction Set

- Each processor has its own instruction set that defines the instructions that the processor understands.
- Examples of modern instruction set architectures include (but are not limited to):
  - x86-64 (aka AMD64, the most commonly used ISA in PCs)
  - ARM (used in many devices, including the Apple iPhone, the Raspberry Pi, and upcoming Macs)
  - RISC-V (an open-source ISA that is growing in popularity)
  - POWER (used by Macs from 1994-2006 and is currently used in high-end servers)
  - SPARC (also used in some high-end servers)

The Anatomy of an Instruction

- An instruction is represented as a sequence of binary digits that contain information that the CPU uses to execute the instruction.
- A program is a sequence of instructions.

Problems with Coding in Machine Code

- Tedious
- Error-prone
- Hard to read

Assembly Language

- Instead of coding in binary digits, code in symbolic representations of the instructions.
- Symbolic labels aid the programmer in grouping instructions and for performing branches.
- Use an assembler to convert assembly language code to machine code.

Advantages of Assembly Language

- Easier to write and read programs than machine code.
- Less error-prone.

Fundamental Limitation of Assembly Language

Assembly language is merely a very thin wrapper over machine code. Programming in either machine code or assembly language requires thinking about the problem in terms of the machine.

Goal: What if we can think more in terms of the problem that we want to solve and less in terms of the architectural details of the CPU?

The FORTRAN Programming Language

- Work started on the language in early 1954 at IBM's then-headquarters in New York City; released in April 1957 [Backus 1981].
- The name derives from "The IBM Mathematical FORmula TRANslating System."
- The original version of FORTRAN (retroactively called FORTRAN I) was designed to run on the IBM 704, a computer that can compute approximately 10,000 operations per second [*Programmer's Primer for FORTRAN Automatic Coding System for the IBM 704*, 1957, p. 2].

Quote about How FORTRAN Was Designed

"As far as we were aware, we simply made up the language as we went along."

—John Backus, "The History of Fortran I, II, and III," 1981

Design Goals of FORTRAN

"[O]ne of our goals was to design a language which would make it possible for engineers and scientists to write programs themselves for the 704. We also wanted to eliminate a lot of the bookkeeping and detailed, repetitive planning which hand coding involved. Very early in our work we had in mind the notions of assignment statements, subscribed variables, and the DO statement (which I believe was proposed by Herrick). We felt that these provided a good basis for achieving our goals for the language, and whatever else was needed emerged as we tried to build a way of programming on these basic ideas."

—John Backus, "The History of Fortran I, II, and III," 1981

FORTRAN I Example

Page 15 of *Programmer's Primer for FORTRAN Automatic Coding System for the IBM 704* has the following example:

Quadratic Formula Example

Given values a, b, c, and d punched on a card, and a set of values for the variable x punched one per card, evaluate the function defined by

$$f(x) = \begin{cases} ax^2 + bx + c & \text{if } x < d \\ 0 & \text{if } x = 0 \\ -ax^2 + bx - c & \text{if } x > d \end{cases}$$

for each value of x, and print x and $f(x)$.

FORTRAN I Example

```
10  READ 1, A, B, C, D
11  READ 1, X
12  IF (X-D) 13, 15, 17
13  FOFX = A*X**2. + B*X + C
14  GO TO 18
15  FOFX = 0.
16  GO TO 18
17  FOFX = -A*X**2. + B*X - C
18  PRINT 1, X, FOFX
19  GO TO 11
```

FORTRAN I Example

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)
10		READ 1, A, B, C, D
11		READ 1, X
12		IF (X-D) 13, 15, 17
13		FOFX = A*X**2. + B*X +C
14		GO TO 18
15		FOFX = 0.
16		GO TO 18
17		FOFX = -A*X**2. + B*X - C
18		PRINT 1, X, FOFX
19		GO TO 11

FORTRAN I IF Statement

```
IF E LBL1, LBL2, LBL3
```

Evaluates the expression E.

$$\texttt{GO TO} \begin{cases} LBL1 & \text{if } E < 0 \\ LBL2 & \text{if } E = 0 \\ LBL3 & \text{if } E > 0 \end{cases}$$

Another FORTRAN I Example

From page 49 of *Programmer's Primer for FORTRAN Automatic Coding System for the IBM 704*:

Matrix Multiplication Example

Given the matrix **A** with dimensions 10×15 and the matrix **B** with dimensions 15×12, compute the elements C_{ij} of the matrix $\mathbf{C} = \mathbf{AB}$. To compute any element C_{ij}, select the i row of **A** and the j column of **B**, and sum the products of their corresponding elements. The general formula for this computation is

$$C_{ij} = \sum_{k=1}^{15} A_{ik} B_{kj}$$

Another FORTRAN I Example

```
      DIMENSION A(10, 15), B(15, 12), C(10, 12)
 3    FORMAT (5E14.5)
      READ 3, A, B
 4    DO 30 I = 1, 10
 5    DO 30 J = 1, 12
 6    C(I,J) = 0.0
10    DO 20 K = 1, 15
20    C(I,J) = C(I,J) + A(I,K)*B(K,J)
30    PRINT 50, I, J, C(I,J)
50    FORMAT (2I5, E16.7)
60    STOP
```

FORTRAN I DO Statement

```
DO LIMIT VAR = BEGIN, END[, STEP]
```

Executes lines up to and including `LIMIT` while `VAR` is between `BEGIN` and `END` (inclusive). `VAR` is incremented by 1 unless `STEP` is specified, in which the amount that `VAR` is incremented is the value of `STEP`.
`DO` loops also have support for `CONTINUE` statements, which works just like it does in Java and later programming languages.

Features of FORTRAN I

- Variables, including reassignment
- Various arithmetic operators and built-in mathematical functions
- Support for integer and floating-point numbers
- Arrays (including native multi-dimensional arrays)
- I/O from punch cards, tape storage, and drum storage (the main memory of the 704)
- `IF` and `DO` statements
- `GO TO`
- Functions (note that recursion wasn't supported); would be further refined in FORTRAN II, released in 1958.

Summary of FORTRAN I

- FORTRAN I revolutionized programming by enabling programmers to express their code in a manner that was less tied to the idiosyncrasies of the machine.
- Led to the development of many other languages that were targeted toward specific types of users (e.g., COBOL was targeted to business users).

Designing ALGOL

- Around the time FORTRAN was released, there was quite a flurry of programming languages being developed for all sorts of machines.
- Facing the prospect of a plethora of incompatible languages, in 1958 delegates from the Association for Computing Machinery (ACM) met with delegates from the *Gesellschaft für angewandte Mathematik und Mechanik* (GAMM), a German society for applied math and mechanics, to design a common programming language.
- John Backus, the leader of the team that developed FORTRAN, was one of the ACM delegates.

Design Goals of ALGOL

- Can be implemented on a variety of machines
- Can meet the programming needs of scientists and engineers
- Be a compelling improvement over existing languages such as FORTRAN.

ALGOL 58

Notable features included:

- Integer, real, and boolean data types, along with arrays of these types
- `switch` and `for` statements
- `if either ... or if ... end` syntax for conditional statements.
- Compound statements
- Functions and procedures

Difference between a Function and a Procedure in ALGOL 58

To define the difference succinctly,

- A function is an identifier and a list of parameters that are mapped to an *expression*.
- A procedure is an identifier and a list of parameters that are mapped to a *statement*.

Shortcomings of ALGOL 58

Certain parts of the language were underspecified or omitted (most notably I/O support), resulting in the language specification being treated more like a collection of suggestions and less like a standard to comply to [Alan J. Perlis, "The American Side of the Development of ALGOL," 1981].

Point of View from Alan J. Perlis

"It should be remembered that the Zurich report was a draft. Algol58 was not intended to be the language around which international conformity would immediately congeal. But it was a skeleton, internationally conceived, from which constructive improvements would lead to the sought after uniform language."

Alan J. Perlis, "The American Side of the Development of ALGOL," 1981

That sought-after uniform language would be ALGOL 60.

ALGOL 60

Notable features included:

- Blocks (replaced ALGOL 58's compound statements) and lexical scoping
- Dropped the distinction between functions and procedures in ALGOL 58
- Recursive procedure calls
- Call by value and call by name semantics
- `if either ... or if ... end` syntax replaced with more familiar `if ... else ...` syntax.

It was also the first programming language to be defined using Backus-Naur Form, a notation for defining context-free grammars.

ALGOL 60 Examples

Examples are from "Revised Report on the Algorithmic Language ALGOL 60" [Naur et al. 1960]:

```
comment Example of a procedure definition
procedure Spur (a) Order: (n); value n;
array a; integer n; real s;
begin integer k;
s:=0;
for k:=1 step 1 until n do s:=s+a[k,k]
end

comment Example of a procedure call
Spur (A) Order: (7) Result to: (V)
```

ALGOL 60 Examples

Examples are from "Revised Report on the Algorithmic Language ALGOL 60" [Naur et al. 1960]:

```
procedure Absmax (a) Size: (n, m) Result: (y)
Subscripts: (i, k);
array a; integer n, m, i, k; real y;
begin integer p, q;
y := 0;
for p:=1 step 1 until n do for q:=1 step 1 until m do
if abs(a[p,q]])>y then begin y:=abs(a[p,q]);
    i:=p; k:=q end end Absmax

Absmax (A, N, M, Yy, I, K)
```

Shortcomings of ALGOL 60

- No standardized I/O facilities, which hampered portability.
- The use of symbols that did not exist on many keyboards.
- There were some ambiguities in the language; see Donald Knuth's 1967 paper "The Remaining Trouble Spots in ALGOL 60."

Influence of ALGOL 60

ALGOL 60 was used for decades by academics to describe algorithms, and it inspired the design of other programming languages, including ALGOL 68 (a rather large programming language that wasn't as influential as ALGOL 60 but is still an influence), BCPL (the grandfather of C), Pascal (which was commonly used in introductory programming courses in many universities from the 1970s until the mid-1990s), and Simula (the first object-oriented programming language, which influenced C++).

Table of Contents

"Go-to Statement Considered Harmful"

- This is "the shot heard 'round the world" in the history of computer programming, one of the most influential papers written in the field.
- So, why exactly, are `GO TO` statements bad?

"Go-to Statement Considered Harmful"

When your program has no `GO TO` statements, it is easy to determine the current runtime state of the program by keeping track of the current line of code (program location) and the state of declared variables. Dijkstra refers to this as "the progress of the process."

"Go-to Statement Considered Harmful"

However, once your program introduces `GO TO` statements, it becomes more difficult, sometimes impossible, to determine the progress of the process simply by keeping track of the program's current location and the state of the program's declared variables.

Structured Programming

Definition (Structured Programming)

An approach to procedural programming where programs are written in such a way where it is possible to determine a program's progress simply by keeping track of the program's current location and the state of the program's declared variables.

Tenets of Structured Programming

- Only one entry and exit per procedure or loop.
  - No `BREAK` statements.
  - One `RETURN` statement per procedure.
- No `GO TO` statements.

Does this code adhere to the structured programming paradigm?

```
public int linearSearch(int[] items, int key) {
  for (int i = 0; i < items.length; i++) {
      if (items[i] == key)
          return i;
  }
  return -1;
}
```

Does this code adhere to the structured programming paradigm?

```
public int linearSearch(int[] items, int key) {
  for (int i = 0; i < items.length; i++) {
      if (items[i] == key)
          return i;
  }
  return -1;
}
```

No. It is not structured because there are two `return` statements in the method.

Here is a version that adheres to the structured programming paradigm:

```
public int linearSearch(int[] items, int key) {
  int index = -1;
  for (int i = 0; i < items.length && index == -1;
       i++) {
    if (items[i] == key)
      index = i;
  }
  return index;
}
```

Why Structured Programming?

Structured programming makes it easier to reason about programs in a mathematical fashion, thus making them easier to formally prove their correctness.

> *"Program testing can be used to show the presence of bugs, but never to show their absence!"*
>
> *—Edsger W. Dijkstra, Structured Programming, 1972, p. 6*

Structured Programming in Practice

- Adopted heavily since the 1970s, to the point that some procedural programming languages such as Java and Python lack `GO TO` statements.
- Exceptions are made for `break` and for exceptions (e.g., `throw` in C++ and Java).
- Some style guides allow for the use of `GO TO` (in languages that have it) for breaking out of nested loops.
- `GO TO` statements are used quite frequently in the Linux kernel, which is written in C.

Table of Contents

The topics that will be covered on Wednesday include:

- Parsing
- Context-free Grammars
- Abstract Syntax Trees