Object Oriented Programming Course

Monsoon 2021

Programming Paradigms

A high level overview

Dictionary

Search for a word

par·a·digm

/ˈperəˌdīm/

See definitions in:

All | Philosophy | Language

noun

noun: **paradigm**; plural noun: **paradigms**

1. a typical example or pattern of something; a model.
   "there is a new paradigm for public art in this country"

   Similar: model | pattern | example | standard | prototype | archetype | ideal

Programming paradigm

A programming paradigm is a style, or “way,” of programming.

A “way” to think about the how we write and organize programs

Programming using sequence of Imperative Statements

What Is the Imperative Mood? (with Examples)

The imperative mood is a verb form that gives a command. For example:

- Empty the bin, John.
  (This is a verb in the imperative mood.)

- John **empties** the bin.
  (This verb is not in the imperative mood. It is in the indicative mood.)

Commands can include orders, requests, advice, instructions, and warnings.

Imperative Programming

Writing programs in the form of imperative statements (aka commands)

“Do this” and “Do that”

Imperative Statement in English

“ Go to a bakery ”

“ Buy a cookie ”

“ Eat the cookie ”

“ Drink some water ”

Declarative Statements in English

“ There is a bakery in 2nd Street “

Imperative Programming

You may think of this as the “default” paradigm

This is how the machine code works

“Move val to register”

“Load val from register”

“Add values”

“Jump to location”

EXAMPLE OF ASSEMBLY LANGUAGE

High level code

```
D = A * B + A * C;
```

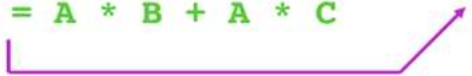
Assume R1 contains the base address of 100

Address		
100	5	A
104	12	B
108	8	C
112		D

Assembly code

```
LD   R2, 0(R1)   ; load A
LD   R3, 4(R1)   ; load B
LD   R4, 8(R1)   ; load C
MUL  R5, R2, R3  ; A * B
MUL  R6, R2, R4  ; A * C
ADD  R5, R5, R6  ; A * B + A * C
SD   12(R1), R5  ; D = A * B + A * C
```

Address		
100	5	A
104	12	B
108	8	C
112	100	D



Problems

Verbose

More error prone

Unnecessary/Accidental Redundancy

Difficult to organize

- Difficult to form a mental model
- Difficult to debug
- Hence, difficult to work in large teams

Problems

Verbose

More error prone

Unnecessary/Accidental Redundancy

Difficult to organize

Difficult to form a mental model

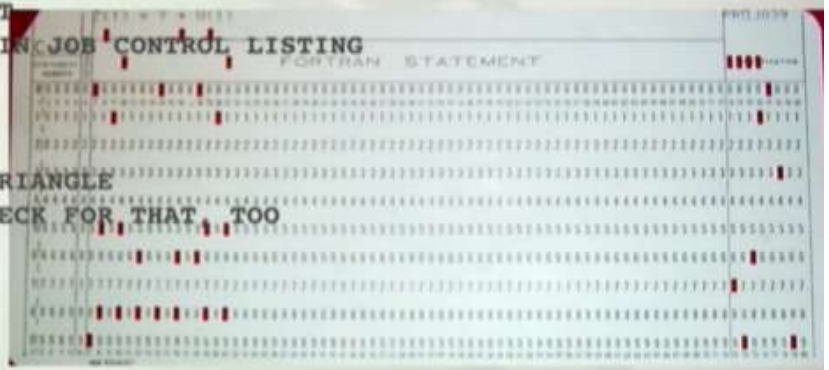
Difficult to debug

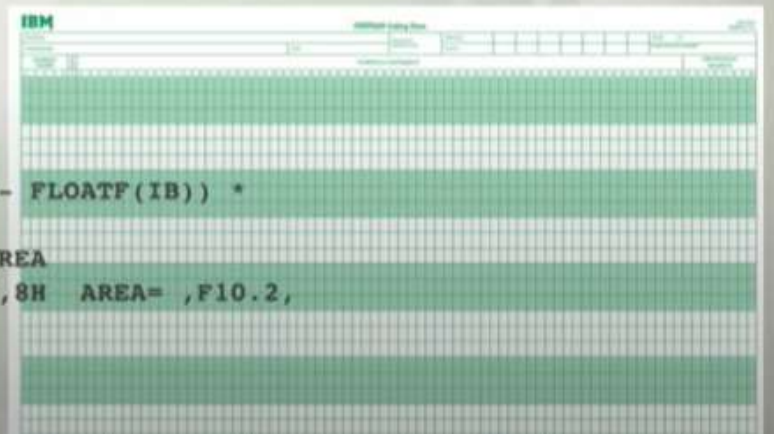
Hence, difficult to work in large teams

1953 FORTRAN

Designed by: John Backus

```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - TAPE READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C IS GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777,777,704
  704 IF (IA+IC-IB) 777,777,705
  705 IF (IB+IC-IA) 777,777,799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```





Problems

Verbose

More error prone

Unnecessary/Accidental Redundancy

Difficult to organize

Difficult to form a mental model

Difficult to debug

Hence, difficult to work in large teams

Problems

Verbose

More error prone

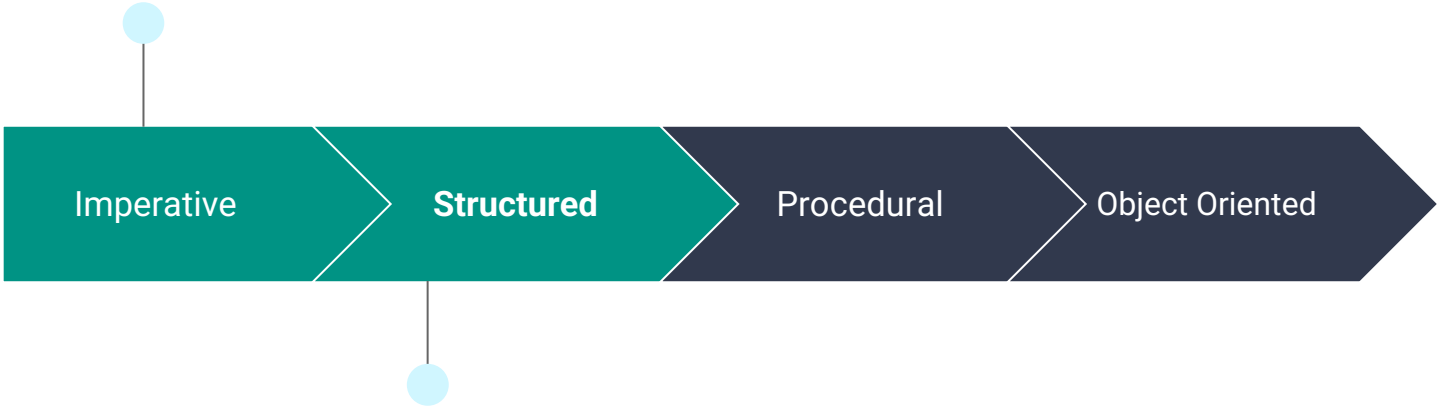
Unnecessary/Accidental Redundancy

Difficult to organize

- Difficult to form a mental model
- Difficult to debug
- Difficult to work in large teams

Dijkstra, E. W. (1968). Letters to the editor: **go to** statement considered harmful. *Communications of the ACM*, *11*(3), 147-148.

Programming using sequence of
Imperative Statements



Let’s do away with harmful goto and introduce control structures IF, FOR, etc.

Structured program theorem

- The structured program theorem, also called the Böhm–Jacopini theorem is a result in programming language theory. It states that a class of control-flow graphs (historically called flowcharts in this context) can compute any computable function if it combines subprograms in only three specific ways (control structures). These are
  - Executing one subprogram, and then another subprogram (sequence)
  - Executing one of two subprograms according to the value of a boolean expression (selection)
  - Repeatedly executing a subprogram as long as a boolean expression is true (iteration)

Structured Programming

Writing programs using nested-control structures without any **harmful goto** statement

Two types of control structures

- Branching
  - IF/ELSE, SWITCH-CASE
- Loops
  - FOR, WHILE, DO-WHILE

Problems

Verbose

More error prone

Unnecessary/Accidental Redundancy

Difficult to organize

Difficult to form a mental model

Difficult to debug

Hence, difficult to work in large teams

Problems

Verbose

More error prone

Unnecessary/Accidental Redundancy

Difficult to organize

Difficult to form a mental model

Difficult to debug

Hence, difficult to work in large teams

Note: This did not eliminate all errors. But got rid of plenty of errors and confusion which were primarily caused by goto statements.

Problems

Verbose

More error prone

Unnecessary/Accidental Redundancy

Difficult to organize

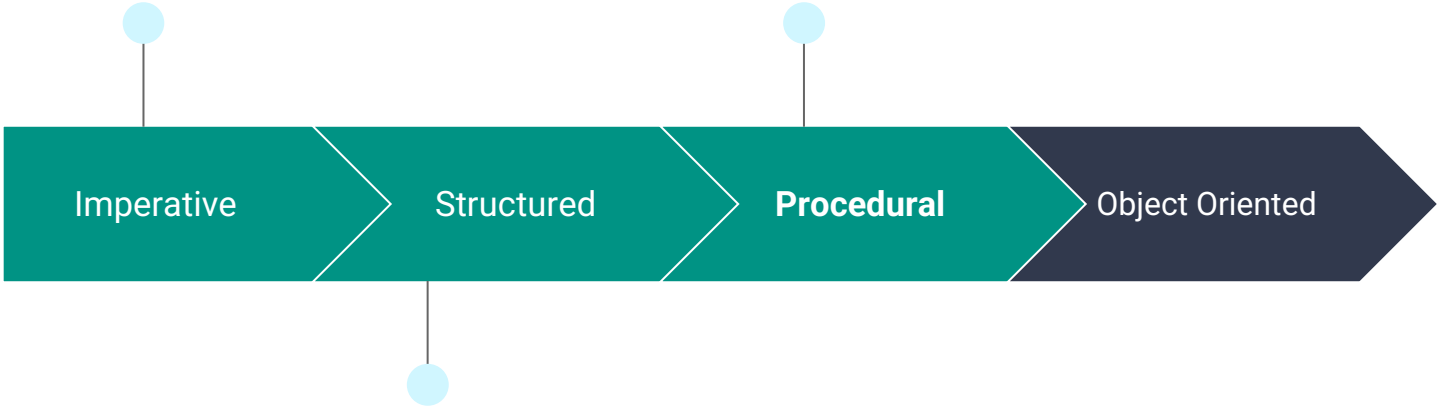
Difficult to form a mental model

Difficult to debug

Hence, difficult to work in large teams

Programming using sequence of Imperative Statements

Organize code into Procedures - to avoid redundancy and to improve organization/understanding

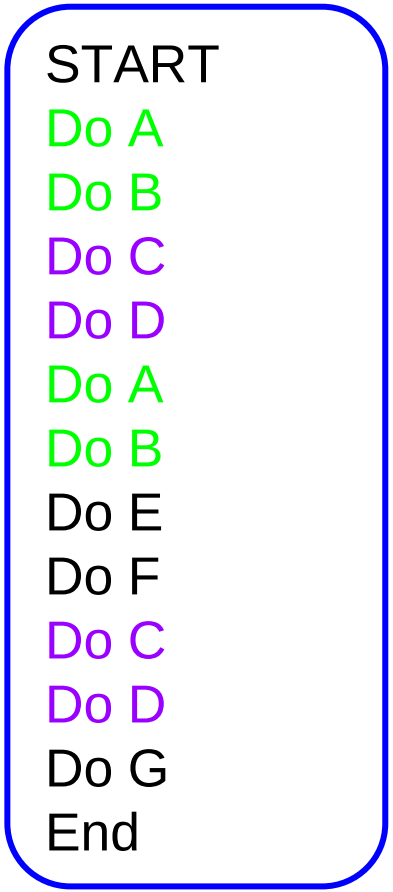


Let’s do away with harmful goto and introduce control structures IF, FOR, etc.

START
Do A
Do B
Do C
Do D
Do A
Do B
Do E
Do F
Do C
Do D
Do G
End

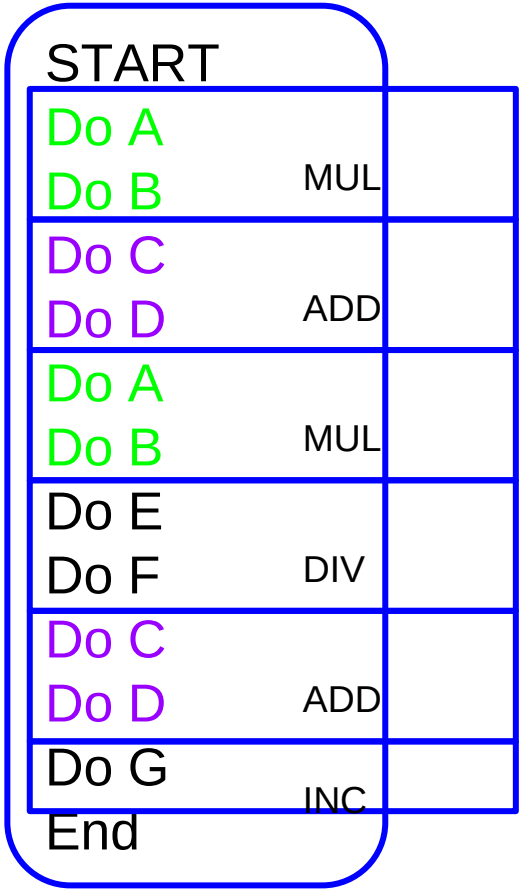
An example of an Imperative Program

- Sequence of Imperative Statements



Redundant Code

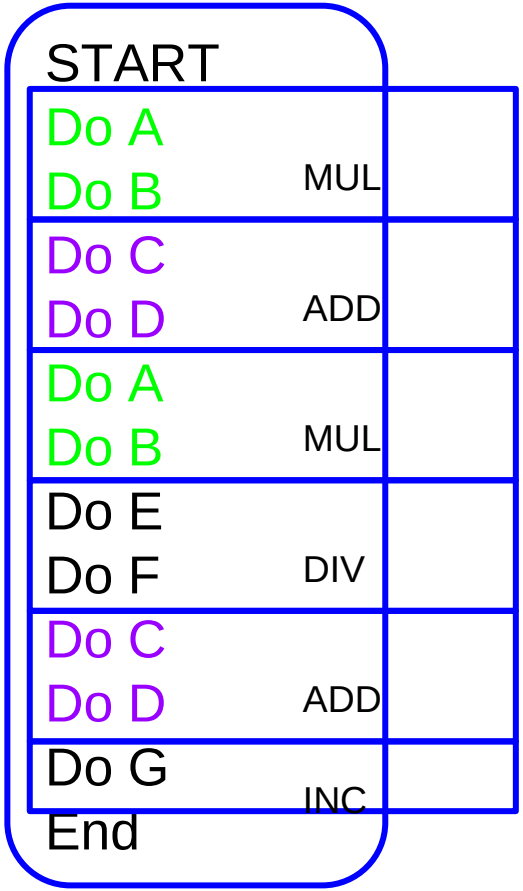
- Pattern of Code that repeats again and again

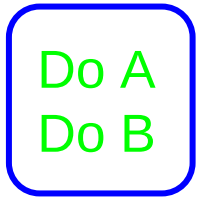


Related Code

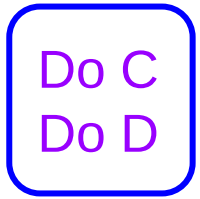
- Sequence of Code which represents a human understandable unit of operation (*a task*)

- A Procedural Approach

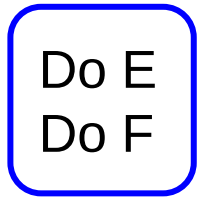




MUL



ADD



DIV

INC

MAIN

C Language

Imperative

- Sequence of Actions

Structured

- Nested Control Structures

Procedural

- Organizes code as procedures

Problems

Verbose

More error prone

Unnecessary/Accidental Redundancy

Difficult to organize

Difficult to form a mental model

Difficult to debug

Hence, difficult to work in large teams