

COMP205 IMPERATIVE LANGUAGES

INTRODUCTION

- 1) Definition
- 2) Note on structured and modular programming, and information hiding
- 3) Example imperative languages
- 4) Features of imperative languages

(Q) WHAT IS AN IMPERATIVE LANGUAGE?

- We can define such languages according to the characteristics that they display:
 - 1) By default statements (commands) are executed in a step-wise, sequential, manner.
 - 2) As a result order of execution is crucial.
 - 3) Destructive assignment.
 - 4) Control is the responsibility of the programmer - programmers must explicitly concern themselves with issues such as memory allocation and declaration of variables.

Q) OUT OF THE FIVE PRINCIPAL PROGRAMMING LANGUAGE PARADIGMS THE IMPERATIVE IS THE MOST POPULAR, WHY?

- Imperative programs tend to run much faster than other types of program.
- The imperative paradigm is much more in tune with the computer community's way of thinking.
- Programmers are prepared to sacrifice some of the advanced features of higher level languages in exchange for speed of execution.
- There is a tendency for programmers to get "bogged down" with control issues.

COMPILED OR INTERPRETED?

- Most imperative languages are designed explicitly for compilation.
- Some very high-level imperative languages are interpreted (e.g. ICON).

- The concept of structured programming devised in late 1960s (Dijkstra and others).
- Emphasizes programming using sequences, conditions and repetition (no jumps and "goto" statements).
- Constructs which have a predictable logic, i.e. they are entered at the top and exited at the bottom.
- Structured programming enhances readability and hence maintainability.



Dijkstra ("Go-to statement considered harmful")

- The concept of modular programming was first taken seriously in the early 1970s.
- Modular programming is concerned with subdivision of programs into manageable “chunks”.
- The concept also encompasses ideas about levels of abstraction – modules are usually arranged in a hierarchy of abstraction.
- Modularity also espouses the concept of *information hiding*.

- Information hiding is concerned with the idea that the attention of programmers should not be distracted by details not central to their current programming task, i.e. programmers should not be required to be concerned with unnecessary details.
- In the imperative paradigm this is achieved through the use of *modules* or *packages*.
- Modules (packages) should be designed so that information contained within those modules is “hidden” from other modules (packages).

FORTRAN

- ‘The IBM Mathematical FORmula TRANslating system’.
- Designed circa 1955 (by, amongst others, Backus).
- First successful attempt to improve on “assembly languages”.
- Still widely used for numerical applications.
- Has been revised to take account of ideas on structured programming etc., however is decreasing in popularity

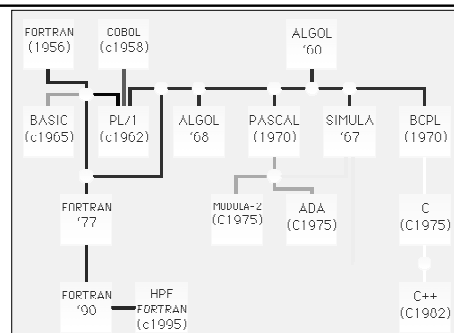
ALGOL 60

- ‘ALGOrithmic Language 1960’.
- Developed in 1950s through a joint European-American committee.
- First block-structured language.
- First language whose syntax was defined using BNF.
- Direct ancestor of most modern imperative languages.

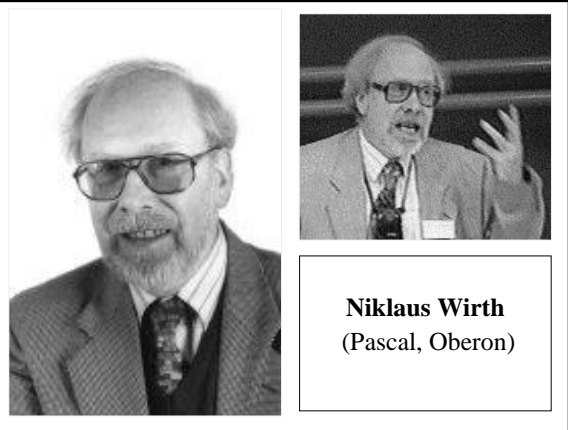
COBOL

- ‘COMmon Business Oriented Language’
- Developed in 1950s through a committee consisting mainly of US computer manufacturers.
- Designed to process large data files and is therefore the most extensively used language for data processing.
- Has been revised several times, but revisions have been unable to take into account programming concepts such as structured programming and modularity.

IMPERATIVE LANGUAGES “FAMILY TREE”

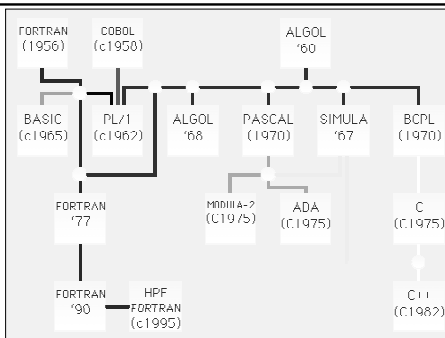


- **BASIC** - Beginner's All-purpose Symbolic Instruction Code. Teaching language made popular by the advent of the microcomputer in the mid 70s.
- **PL/1** - Intended to be an all purpose programming language to support both scientific and data processing applications. Did not live up to its aspirations.
- **Algol'68** - Successor to ALGOL'60. Proved less popular than its predecessor.
- **Pascal** - A popular block structured teaching language devised by Niklaus Wirth.
- **Simula'67** - Simulation language, significance of which is that it introduced the "class" concept.



Niklaus Wirth
(Pascal, Oberon)

IMPERATIVE LANGUAGES "FAMILY TREE"



- **Modula-2** - Pascal extended with modules.
- **Ada** - Pascal based language sponsored by the US Department of Defence.
- **BCPL** - Systems programming languages, significance of which is that it is a precursor to C.
- **C** - Systems programming language used in the development of the UNIX system. Is at a lower level than most imperative languages. Standardised in 1988 to ANSI C.
- **C++** - Object-oriented extension to C.
- **Java**

FEATURES OF IMPERATIVE LANGUAGES

- They are usually "typed". Broadly we can identify two categories of imperative data type:
 - 1) Basic data types.
 - 2) Higher level data types.
- Components comprise:
 - 1) Data declarations.
 - 2) Expressions which yield values.
 - 3) Statements which carry out some operation, e.g. assignment statements, conditional statements, program constructs.

FEATURES OF IMPERATIVE LANGUAGES Continued

- An I/O mechanism
- An error-handling mechanism
- A method of grouping all of the above into a complete program - (program composition).

COMPOSITION OF THE IMPERATIVE PART OF THE COURSE

12 Lectures:

- [1] Introduction (this lecture)
- [2] Data
- [3] Basic Data Types I
- [4] Basic Data Types II
- [5] Higher Level Data Types I (arrays)
- [6] Higher Level Data Types II (more on arrays)
- [7] Higher Level Data Types III (structures/records)
- [8] Higher Level Data Types IV (linked lists, unions)
- [9] Expressions and statements
- [10] Program Constructs I
- [11] Program Constructs II and Error Handling
- [12] Program Composition I (scope rules)
- [13] Program Composition II (parameter passing)
- [14] Program Composition III (modules, packages, ADTs).

SUMMARY

- 1) Definition
- 2) Note on structured and modular programming, and information hiding
- 3) Example imperative languages
- 4) Features of imperative languages