Concepts of Programming Languages

Lecture 3 - Imperative Programming

Patrick Donnelly

Montana State University

Spring 2014

Administrivia

Assignments:

Homework #1 : due 01.24

Reading:

Chapters 1 and 2

I really hate this darn machine;
I wish they would sell it;
It won't do what I want it to,
but only what I tell it.

Anonymous Programmer's Lament

Imperative Programming

Oldest and most well-developed paradigm

Mirrors computer architecture

Languages:

- Fortran, Pascal
- C, Clite
- Ada 83
- Perl

Imperative Programming

Programs written in imperative programming languages consist of:

- A program state
- Instructions that change the program state

Program instructions are “imperative” in the grammatical sense of imperative verbs that express a command

John von Neuman (1908 - 1957)

Hungarian-American mathematician

Came to Princeton in 1930's

Became interested in computers while participating in the development of the hydrogen bomb.

First person to document the basic concepts of stored program computers

The von Neumann-Eckert Model

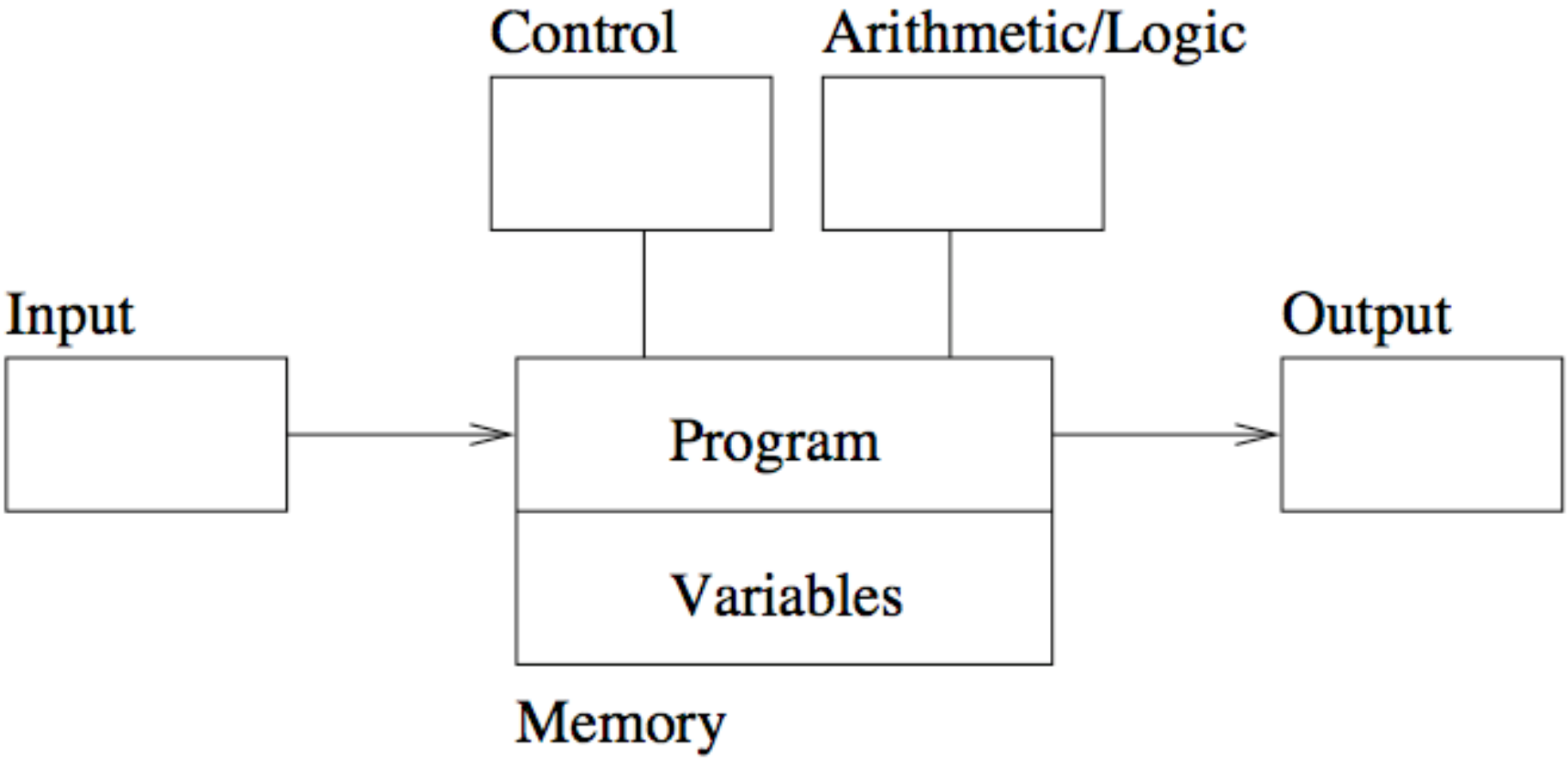


Figure 1.1: The von Neumann-Eckert Computer Model

What Makes a Language Imperative?

In a von Neumann machine, memory holds:

- Instructions
- Data
- Assignment statement
- Others:
  - Conditional branching
  - Unconditional branch (goto)

There is a duality of instructions and data $\rightarrow$ programs can be self modifying

Von Neumann outlined this structure in a document known as the “First Draft of a Report on the EDVAC” June, 1945

The von Neumann-Eckert Model

Earlier computers had fixed programs: they were hardwired to do one thing.

Sometimes external programs were implemented with paper tape or by setting switches.

Eckert and Mauchly considered stored program computers as early as 1944

During WW II they designed & built the ENIAC (although for simplicity the stored program concept was not included at first)

The von Neumann-Eckert Model

Later (with von Neumann), they worked on the EDVAC

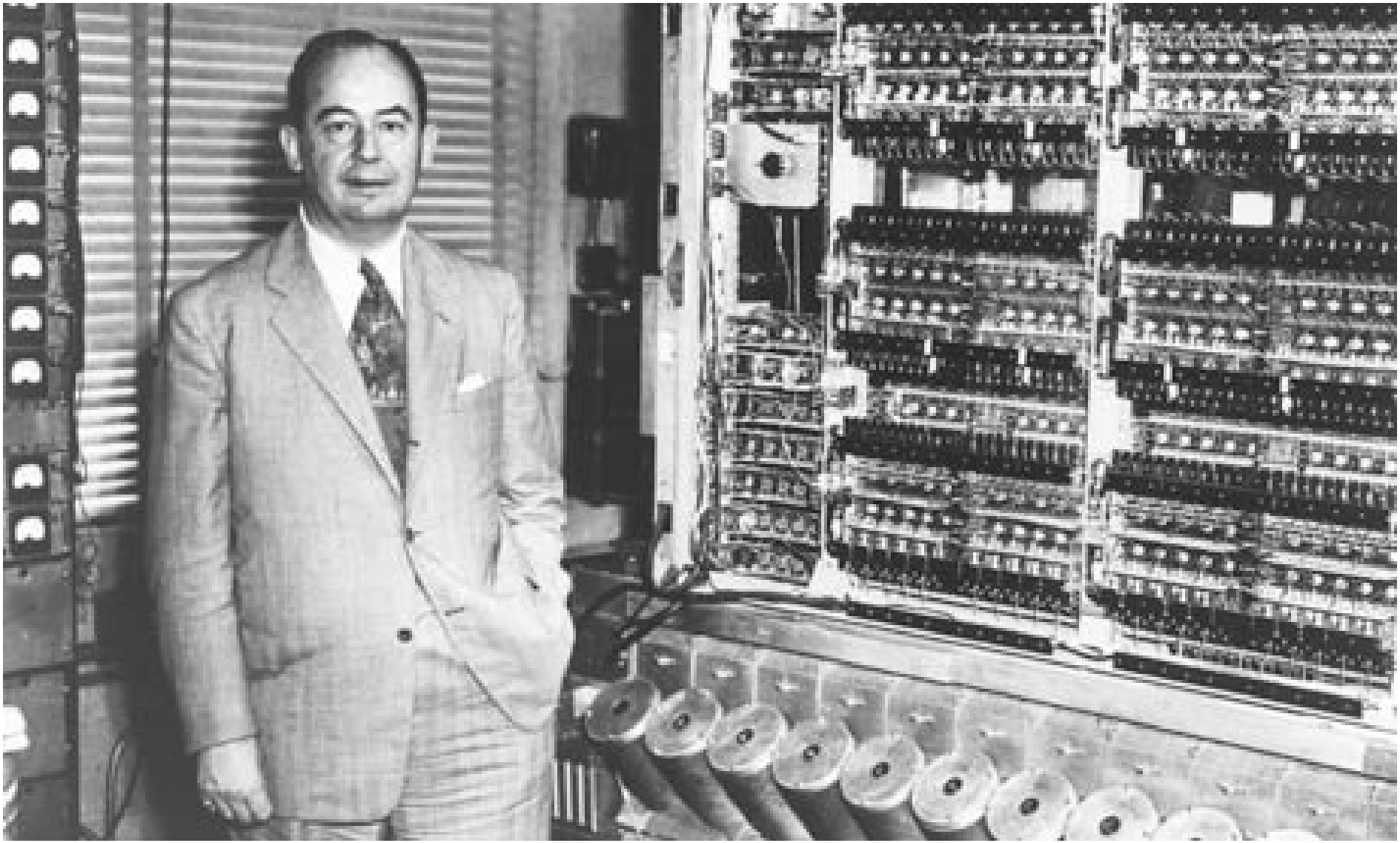
First stored program electronic computer: the Manchester ESSM (Baby)

Victoria University of Manchester

Executed its first program June 21, 1948

A number of other stored program machines were under development around this time.

Stored-Program Computer 1945



History of Imperative Languages

first were assembly languages

1954-55	Fortan	developed for IBM 704
1958	ALGOL	
1960	COBOL	developed by government committee
1964	BASIC	
1970	Pascal	developed by Niklaus Wirth
1972	C	developed by Dennis Ritchie
1978-83	Ada	developed by DoD

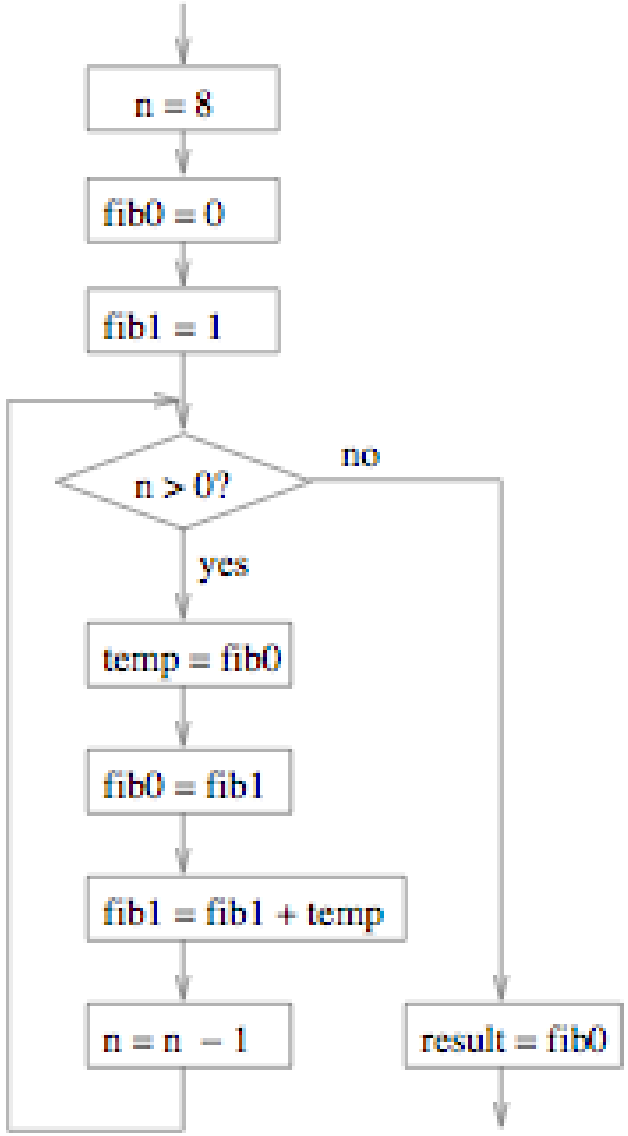
Imperative Programming Language

Definition

An ***imperative programming language*** is one which is Turing complete and also supports certain common historical features:

- Control structures
- Input/Output
- Error and exception handling
- Procedural abstraction
- Expressions and assignment
- Library support for data strctures

Flowchart for Fibonacci



Imperative Programming Language

Imperative programming languages specify a sequence of operations for the computer to execute.

Declarative languages describe the solution space, provide knowledge required to get there, but don't describe the steps to get to the solution.

Functional languages (e.g., Haskell, OCaml) and **logic** languages are declarative (e.g., Prolog).

Imperative Programming Language

Nicholas Wirth described imperative programs as being “algorithms plus data structures”.

Algorithms become programs through the process of procedural abstraction and stepwise refinement.

Libraries of reusable functions support the process (functions = procedures)

Definition

Imperative programming + procedures = ***procedural programming***.

Procedural Abstraction

Definition

Procedural abstraction allows the programmer to be concerned mainly with a function interface, ignoring the details of how it is computed.

Procedural Abstraction

Definition

Procedural abstraction allows the programmer to be concerned mainly with a function interface, ignoring the details of how it is computed.

Abstraction allows us to think about **what** is being done, not **how** it is implemented.

Stepwise Refinement

Definition

The process of ***stepwise refinement*** utilizes procedural abstraction to develop an algorithm starting with a general form and ending with an implementation.

This is also called ***functional decomposition***.

e.g., sort(list, len)

Programmers start with a description of what the program should do, including I/O, and repeatedly break the problem into smaller parts, until the sub-problems can be expressed in terms of the primitive states and data types in the language.

Structured Programming

Definition

Structured programming is a disciplined approach to imperative program design.

Uses procedural abstraction and top-down design to identify program components (also called modules or structures).

Program structures combined in a limited number of ways, so understanding how each structure works means you understand how the program works

Program control flow is based on decisions, sequences, loops, but. . . .

Does not use goto statements

Modules are developed, tested separately and then integrated into the whole program.

Characteristics of Imperative Languages

Statements are commands:

- Command order is critical to correct execution
- Programmers control all aspects: algorithm specification, memory management, variable declarations, etc.

They work by modifying program state

Statements reflect machine language instructions.

Features of Imperative Languages

They are usually “typed” either statically or dynamically.

- Basic data types (e.g.,int, float, boolean, char)
- Compound data types (structs, arrays).

Statement types:

- Declarations, Assignment, Conditionals, Loops . . . .

I/O and error handling mechanisms.

A method of grouping all of the above into a complete program - (program composition).

- Procedural abstraction, step-wise refinement, function mechanisms.

Assignment

Assignment statement is fundamental:

```
target = expression
```

This is a destructive assignment statement (changes program state).

Assignment operators: $=$ or $:=$ or psuedocode $\leftarrow$

Based on machine operations such as `MOV` or `STO`.

Expressions

Expressions represent a value and have a type.

Understanding expressions means understanding operator precedence, operator overloading, casting and type conversion, among other issues.

Simple arithmetic expressions are based on machine language arithmetic operators (`DIV, MUL,` etc)

Logical operators are based on similar ML instructions (`AND, OR,` etc)

Recall assembly uses different instructions for different types (integer vs. floating point).

Expressions and Assignment

Definition

In ***copy semantics***, an expression is evaluated to a value, which is copied to the target; used by imperative languages.

Definition

In ***reference semantics***, an expression is evaluated to an object, whose pointer is copied to the target; used by object-oriented languages.

Libraries

There exist vast libraries of functions for most imperative languages.

International Mathematics and Statistics Library (IMSL) contains thousands of mathematical and statistical functions for many languages.

Partially accounts for the longevity of languages like Fortran, Cobol, and C.

Imperative Programming

Imperative programming is the oldest programming paradigm

It is based on the von Neumann-Eckley model of a computer

It works by changing the program state through assignment statements

Procedural abstraction and structured programming are its design techniques.