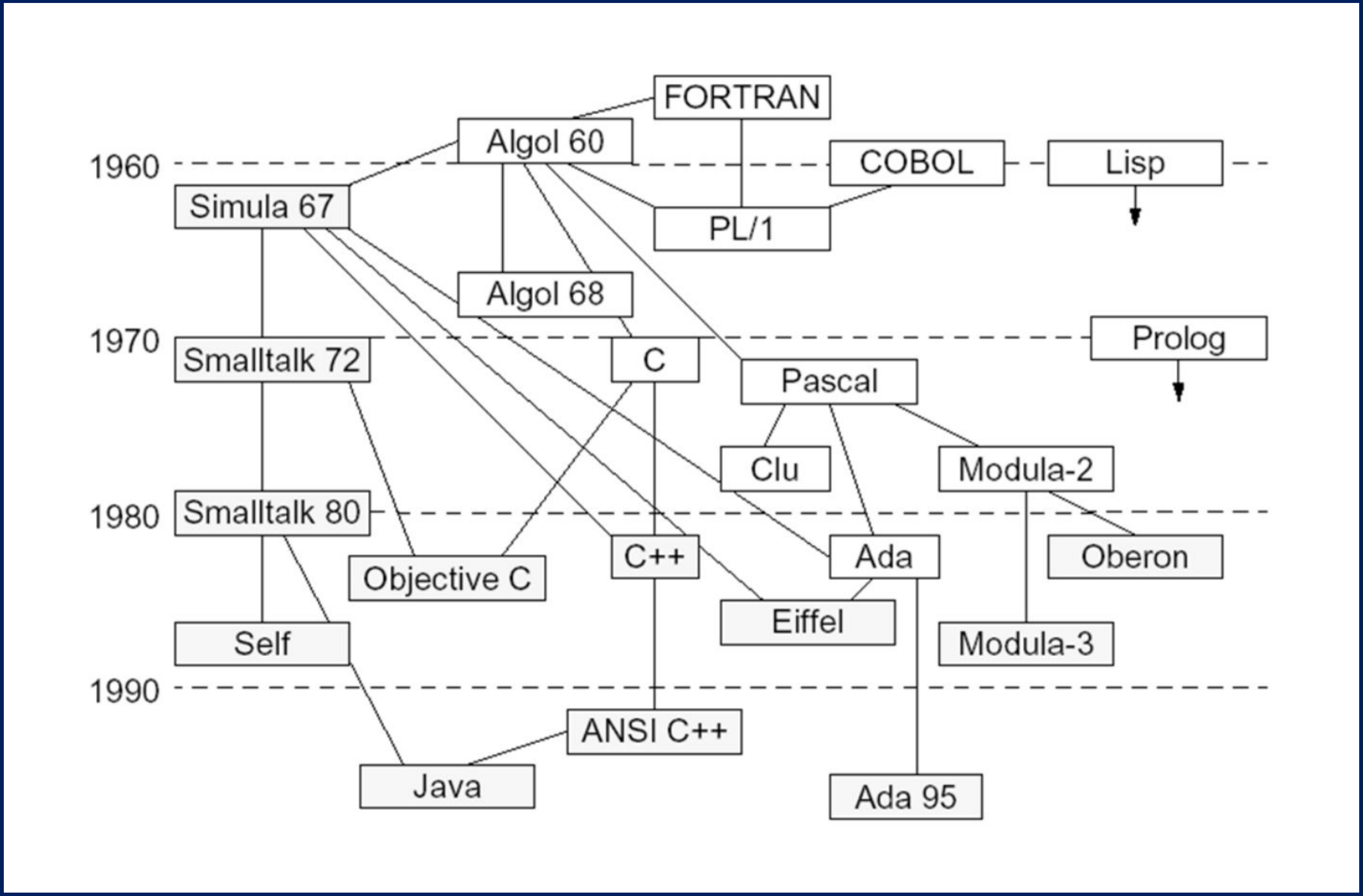
IPT209
Integrative Programming and Technologies

Programming Languages Elements and Paradigms

Programming Language Timeline

- **FlowMatic**
  - 1955 Grace Hopper UNIVAC
- **ForTran**
  - 1956 John Backus IBM
- **AlgOL**
  - 1958 ACM Language Committee
- **LISP**
  - 1958 John McCarthy MIT
- **CoBOL**
  - 1960 Committee on Data Systems Languages
- **BASIC**
  - 1964 John Kemeny & Thomas Kurtz Dartmouth
- **PL/I**
  - 1964 IBM Committee
- **Simula**
  - 1967 Norwegian Computing Center Kristen Nygaard & Ole-Johan Dahl
- **Logo**
  - 1968 Seymour Papert MIT
- **Pascal**
  - 1970 Nicklaus Wirth Switzerland
- **C**
  - 1972 Dennis Ritchie & Kenneth Thompson Bell Labs
- **Smalltalk**
  - 1972 Alan Kay Xerox PARC
- **ADA**
  - 1981 DOD
- **Objective C**
  - 1985 Brad Cox Stepstone Systems
- **C++**
  - 1986 Bjarne Stroustrup Bell Labs
- **Eiffel**
  - 1989 Bertrand Meyer France
- **Visual BASIC**
  - 1990 Microsoft
- **Delphi**
  - 1995 Borland
- **Object CoBOL**
  - 1995 MicroFocus
- **Java**
  - 1995 Sun Microsystems

Programming Language History



Human Computer Interaction

One of the primary goals of HCI is to create technology that is easy and enjoyable to use, even for people with little or no technical expertise.

This involves designing interfaces that are intuitive, clear, and efficient, with minimal cognitive load and a low error rate.

HCI also involves understanding the social and cultural factors that affect technology use, including issues of privacy, security, and trust.

Human Computer Interaction

HCI research and practice have many applications in a variety of fields, including software development, web design, game design, virtual and augmented reality, mobile devices, and assistive technology for people with disabilities.

By improving the usability and effectiveness of technology, HCI can enhance people's lives, increase productivity, and facilitate communication and collaboration.

Human Computer Interaction

HCI research and practice have many applications in a variety of fields, including software development, web design, game design, virtual and augmented reality, mobile devices, and assistive technology for people with disabilities.

By improving the usability and effectiveness of technology, HCI can enhance people's lives, increase productivity, and facilitate communication and collaboration.

Human Computer Interaction

HCI research and practice have many applications in a variety of fields, including software development, web design, game design, virtual and augmented reality, mobile devices, and assistive technology for people with disabilities.

By improving the usability and effectiveness of technology, HCI can enhance people's lives, increase productivity, and facilitate communication and collaboration.

Human Computer Interaction & Programming

In summary, programming and HCI are closely interlinked and rely on each other to create effective and user-friendly interactive systems.

Good programming practices can contribute to the effectiveness and usability of an interface, while HCI principles provide guidance for designing and evaluating systems that meet users' needs and preferences.

Programming Languages

The number of programming languages in the world depends on the rules you establish for deciding whether or not a language counts.

- **TIOBE – 250**
- **Wikipedia - 700**
- **FOLDOC - 1,000**
- **The Language List - 2,500**
- **HOPL - 8,945**
- **J.E. Sammet - ~165 (In 1971)**

"Programming languages are the least usable, but most powerful human-computer interfaces ever invented"

Programming Languages and Abstraction

Programming languages provide an abstraction from a computer's instruction set architecture

- **Low-level programming languages provide little or no abstraction, e.g., machine code and assembly language**
  - Difficult to use
  - Allows to program efficiently and with a low memory footprint
- **High-level programming languages isolate the execution semantics of a computer architecture from the specification of the program**
  - Simplifies program development

Machine code

```
8B542408 83FA0077 06B80000 0000C383
C9010000 008D0419 83FA0376 078BD98B
B84AEBF1 5BC3
```

Assembly language

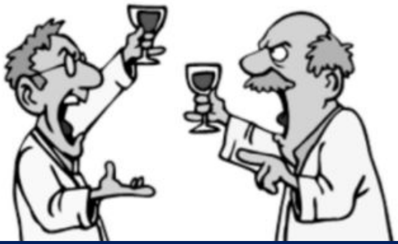
```
mov edx, [esp+8]
cmp edx, 0
ja @f
mov eax, 0
ret
```

High-level language

```
unsigned int fib(unsigned int n) {
    if (n <= 0)
      return 0;
    else if (n <= 2)
      return 1;
    else
      ...
}
```

How do Programming Languages Differ?

- Common Constructs:
  - basic data types (numbers,etc.);
  - variables;
  - expressions;
  - statements;
  - keywords;
  - control constructs;
  - procedures;
  - comments;
  - errors ...



- Uncommon Constructs:
  - type declarations;
  - special types (strings, arrays, matrices,...);
  - sequential execution;
  - concurrency constructs;
  - packages/modules;
  - objects;
  - general functions;
  - generics;
  - modifiable state;...

Elements of Programming Languages

Programming languages have many similarities with natural languages

e.g., they conform to rules for syntax and semantics, there are many dialects, etc.

Programming languages have several key elements that define their syntax, semantics, and functionality. These elements include:

Syntax, Semantics, Data types, Control structures, Functions and procedures, Variables, Input and output

Elements of Programming Languages

Programming languages have many similarities with natural languages

e.g., they conform to rules for syntax and semantics, there are many dialects, etc.

Programming languages have several key elements that define their syntax, semantics, and functionality. These elements include:

Syntax, Semantics, Data types, Control structures, Functions and procedures, Variables, Input and output

Elements of PL: Syntax

- **Programming languages have different syntax, which refers to the rules that govern how code is written and formatted.**

- **Some languages use curly braces ({}) to delimit blocks of code, while others use indentation. Some languages are case-sensitive, while others are not.**

- **Programming syntax includes elements such as:**

  Keywords, Operators, Identifiers, Punctuation, Comments

- **In addition to these elements, programming syntax also includes rules for formatting code, such as indentation and line breaks, to make the code easier to read and understand.**

Elements of PL: Syntax

The syntax of a language describes how well-formed expressions should look like

This includes putting together symbols to form valid tokens as well as stringing together tokens to form valid expressions

> **For example, the following (English) sentence is not correct:**
>
> *“Furiously slqxp ideas grn colorless.*
>
> **In contrast, the sentence**
>
> *“Colorless green ideas sleep furiously.”*
>
> **is syntactically correct (but it does not make any sense).**

The syntax of a programming language is usually described by a formalism called grammar

Elements of PL: Semantics

Programming languages also differ in their semantics, which refers to the meaning of the code.

> For example, some languages are strongly typed, meaning that variables must be declared with a specific data type and cannot be changed. Other languages are dynamically typed, meaning that variables can be assigned different data types at runtime.

Programming semantics refers to the meaning of code and how it is executed by a computer. It is concerned with what a program does, rather than how it does it.

Programming semantics includes a range of concepts and principles that describe how programming languages work, including:

> Data types, Variables, Control structures, Functions, Scope, Object-oriented programming concepts

Elements of PL: Semantics

There are several approaches to programming semantics that are used to define the meaning of programming languages. Some of the common approaches include:

Operational semantics: This approach defines the meaning of a program in terms of its execution.

Denotational semantics: This approach defines the meaning of a program in terms of its mathematical properties.

Axiomatic semantics: This approach defines the meaning of a program in terms of logical rules that describe the behavior of the program.

Elements of PL: Typing

- **A programming language needs to organize data in some way**
- **The constructs and mechanisms to do this are called type system**
- **Types help in**
  - designing programs
  - checking correctness
  - determining storage requirements
- **Type System**

  The type system of a language usually includes:
  - a set of predefined data types, e.g., integer, string
  - a mechanism to create new types, e.g., typedef
  - mechanisms for controlling types:
    - equivalence rules: when are two types the same?
    - compatibility rules: when can one type be substituted for another?
    - inference rules: how is a type assigned to a complex expression?
- **rules for checking types, e.g., static vs. dynamic**

Elements of PL: Data Types

A language is typed if it specifies for every operation to which data it can be applied

Languages such as assembly or machine languages can be untyped Assembly language: all data is represented by bitstrings (to which all operations can be applied)

Languages such as markup or scripting languages can have very few types XML with DTDs: elements can contain other elements or parsed character data (#PCDATA)

Elements of PL: Strong and Weak Typing

There is a distinction between weak typing and strong typing

- In strongly typed languages, applying the wrong operation to typed data will raise an error
  - Languages supporting strong typing are also called type-safe

- Weakly typed languages perform implicit type conversion if data do not perfectly match, i.e., one type can be interpreted as another
  - e.g., the string “3.4028E+12” representing a number might be treated as a number
  - May produce unpredictable results

Elements of PL: Type Casting

In some languages it is possible to bypass implicit type conversion done by the compiler

Type casting is an explicit type conversion defined within a program

Example of type casting:

```
double da = 3.3;
double db = 3.3;
double dc = 3.4;
int result1 = (int)da + (int)db + (int)dc; //result == 9
```

Implicit type conversion gives a different result (conversion is after addition)

```
int result2 = da + db + dc; //result == 10
```

Elements of PL: Static vs. Dynamic Type Checking

- We also distinguish between languages depending on when they check typing constraints
- In static typing we check the types and their constraints before executing the program
  - Can be done during the compilation of a program
- When using dynamic typing, we check the typing during program execution
- Although some people feel quite strongly about this, each approach has pros and cons

Static typing:

+ less error-prone

- sometimes too restrictive

Dynamic typing:

+ more flexible

- harder to debug (if things go wrong)