ORGANIZATION OF PROGRAMMING LANGUAGES

Class Overview

- Registration webpage:
  - http://teaching.yfolajimi.com/register.html
- Class Web page
  - http://teaching.yfolajimi.com/opl.html
- Lecture materials
  - Textbooks:
    - Concepts of Programming Languages by Robert Sebesta
    - Programming languages, principles and paradigm by Allen Tucker and Robert Noonan
  - Resources on the class webpage (slides, pdfs, videos etc)
  - Papers, publications and other online resources

Course outline

- Introduction and brief history of programming languages
- Imperative programming,
- Generative Grammars, Lexical and syntactic analysis,
- variables, bindings and scope
- data types and type checking
- functional programming scheme,
- expression of assignments
- program statements
- program units
- logic programming.

REASONS FOR STUDYING CONCEPTS OF PROGRAMMING LANGUAGES

1. Increased ability to express ideas/algorithms
   - In Natural language, the depth at which people think is influenced by the expressive power of the language they use. In programming language, the complexity of the algorithms that people Implement is influenced by the set of constructs available in the programming Language
2. Improved background for choosing appropriate Languages

- Many programmers use the language with which they are most familiar, even though poorly suited for their new project. It is ideal to use the most appropriate language.

3. Increased ability to learn new languages

- For instance, knowing the concept s of object oriented programming OOP makes learning Java significantly easier and also, knowing the grammar of one's native language makes it easier to learn another language.

4. Better Understanding of Significance of implementation
5. Better use of languages that are already known
6. The overall advancement of computing

APPLICATION DOMAINS

Scientific Applications

2. Data processing Applications

3. Text processing Applications

4. Artificial intelligence Applications

5. Systems Programming Applications

6. Web software

SCIENTIFIC APPLICATIONS

- those which predominantly manipulate numbers and arrays of numbers, using mathematical and statistical principles as a basis for the algorithms.
- These algorithms encompass such problem as statistical significance test, linear programming, regression analysis and numerical approximations for the solution of differential and integral equations.
- Examples:
  - FORTRAN, Pascal, Math lab.

DATA PROCESSING APPLICATIONS

- Those programming problems whose predominant interest is in the creation, maintenance, extraction and summarization of data in records and files.
- Example:
- COBOL is a programming language that can be used for data processing applications.

TEXT PROCESSING APPLICATIONS

- those whose principal activity involves the manipulation of natural language text, rather than numbers as their data. SNOBOL and C language have strong text processing capabilities

ARTIFICIAL INTELLIGENCE APPLICATIONS

- those programs which are designed principally to emulate intelligent behavior.
- They include game playing algorithms such as chess, natural language understanding programs, computer vision, robotics and expert systems.
- Examples
  - LISP has been the predominant AI programming language,
  - PROLOG using the principle of ‘’Logic programming’’
  - Lately AI applications are written in Java, c++ and python.

SYSTEMS PROGRAMMING APPLICATIONS

- involve developing those programs that interface the computer system (the hardware) with the programmer and the operator.
- These programs include compilers, assembles, interpreters, input-output routines, program management facilities and schedules for utilizing and serving the various resources that comprise the system.
- Examples:
  - Ada and Modula – 2 are examples of programming languages
  - C.

WEB SOFTWARE

- collection of languages which include:
  - Markup (e.g. XHTML)
  - Scripting for dynamic content under which we have the
    - Client side, using scripts embedded in the XHTML documents e.g. Javascript, PHP
    - Server side, using the common Gateway interface e.g. JSP, ASP, PHP
  - General- purpose, executed on the web server through cGI e.g. Java, C++.

CRITERIA FOR LANGUAGE EVALUATION AND COMPARISION

1. Expressivity

- the ability of a language to clearly reflect the meaning intended by the algorithm designer (the programmer).
- “expressive” language permits an utterance to be compactly stated, and encourages the use of statement forms associated with structured programming (usually “while “loops and “if – then – else” statements).

2. Well – Definedness

- the language’s syntax and semantics are free of ambiguity, are internally consistent and complete. Thus the implementer of a well defined language should have, within its definition a complete specification of all the language’s expressive forms and their meanings. The programmer, by the same virtue should be able to predict exactly the behavior of each expression before it is actually executed.

3. Data types and structures

- the ability of a language to support a variety of data values (integers, real, strings, pointers etc.) and non elementary collect ions of these.

CRITERIA FOR LANGUAGE EVALUATION AND COMPARISION

4. Modularity

- Modularity has two aspects: the language's support for sub-programming and the language's extensibility in the sense of allowing programmer – defined operators and data types. By sub programming, we mean the ability to define independent procedures and functions (subprograms), and communicate via parameters or global variables with the invoking program.

5. Input-Output facilities

- In evaluating a language's "Input-Output facilities" we are looking at its support for sequential, indexed, and random access files, as well as its support for database and information retrieval functions

6. Portability

- A language which has "portability" is one which is implemented on a variety of computers. That is, its design is relatively"machine – independent". Languages which are well-defined tend to be more portable than others.

CRITERIA FOR LANGUAGE EVALUATION AND COMPARISION

7. Efficiency

- An “efficient” language is one which permits fast compilation and execution on the machines where it is implemented. Traditionally, FORTRAN and COBOL have been relatively efficient languages in their respective application areas.

8. Pedagogy

- Some languages have better “pedagogy” than others. That is, they are intrinsically easier to teach and to learn, they have better textbooks; they are implemented in a better program development environment, they are widely known and used by the best programmers in an application area.

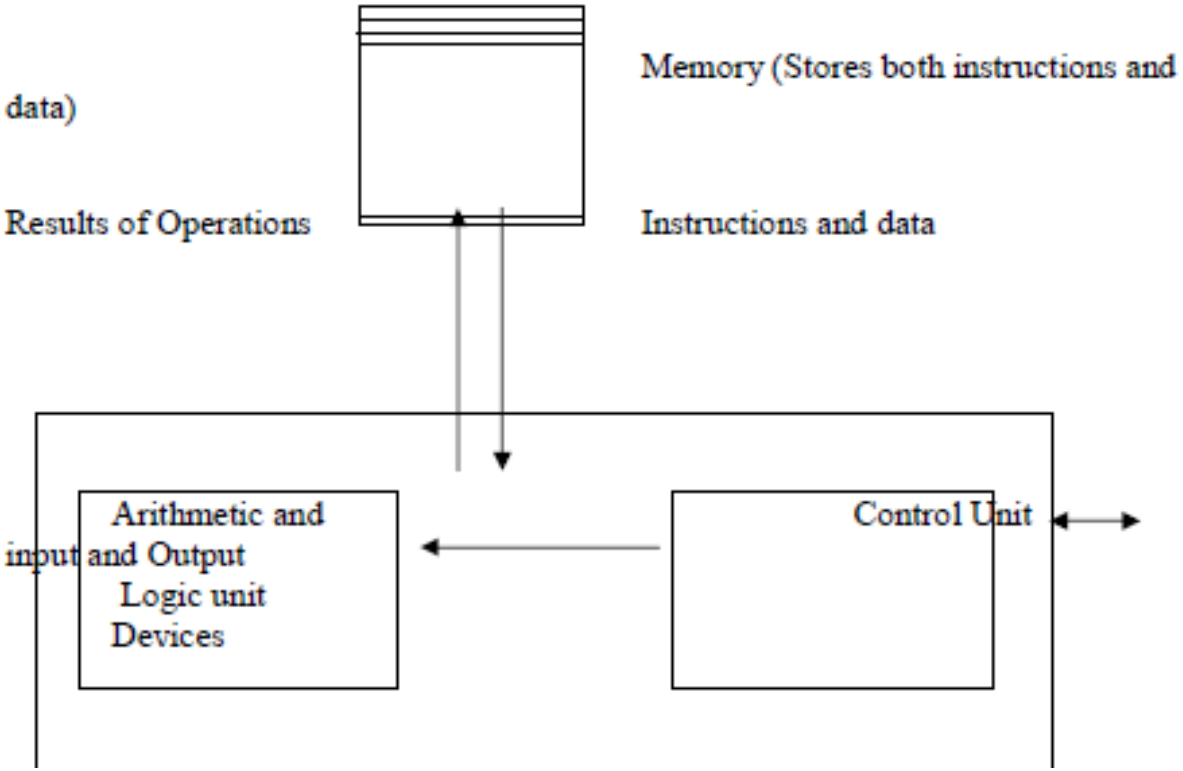
9. Generality

- This means that a language is useful in a wide range of programming applications. For instance, APL has been used in mathematical applications involving matrix algebra and in business applications as well.

INFLUENCES ON LANGUAGE DESIGN

- **1. Computer Architecture:**
  - Languages are developed around the prevalent computer architecture, known as the Von Neumann architecture (the most prevalent computer architecture).
  - The connection speed between a computer's memory and its processor determines the speed of that computer. Program instructions often can be executed much faster than the speed of the connection; the connection speed thus, results in a bottleneck (Von Neumann bottleneck). It is the primary limiting factor in the speed of computers.

Fig; Von Neumann Architecture



LANGUAGE PARADIGMS (Developments in Programming Methodology)

1. Imperative

- This is designed around the Von Neumann architecture. Computation is performed through statements that change a program's state. Central features are variables, assignment statements and iteration, sequency of commands, explicit state update via assignment. Examples of such languages are Fortran, Algol, Pascal, C/C++, Java, Perl, Javascript, Visual BASIC.NET.

2. Functional

- Here, the main means of making computations is by applying functions to parameters. Examples are LISP, Scheme, ML, Haskell. It may also include OO (Object Oriented) concepts.

3. Logic

- This is Rule-based (rules are specified in no particular order). Computations here are made through a logical inference process. Examples are PROLOG and CLIPS. This may also include OO concepts.

TRADE-OFFS IN LANGUAGE DESIGN

Reliability Vs. Cost of Execution:

- For example, Java demands that all references to array elements be checked for proper indexing, which leads to increased execution costs.

2. Readability vs. Writability:

- **APL provides many powerful operators land a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability.**

3. Writability (Flexibility) vs. reliability:

- The pointers in c++ for instance are powerful and very flexible but are unreliable.

IMPLEMENTATION METHODS

1. **Compilation**
   - Programs are translated into machine Language & System calls
2. **Interpretation**
   - Programs are interpreted by another program (an interpreter)
3. **Hybrid**
   - Programs translated into an intermediate language for easy interpretation
4. **Just –in-time**
   - Hybrid implementation, then compile sub programs code the first time they are called.

COMPILATION

- - Translated high level program (source language) into machine code (machine language)
  - – - Slow translation, fast execution
  - – - Compilation process has several phases
    - Lexical analysis converts characters in the source program into lexical units (e.g. identifiers, operators, keywords).
    - Syntactic analysis: transforms lexical units into parse trees which represent the syntactic structure of the program.
    - Semantics analysis check for errors hard to detect during syntactic analysis; generate intermediate code.
    - Code generation – Machine code is generated

INTERPRETATION

- - Easier implementation of programs (run-time errors can easily and immediately be displayed).
- - Slower execution (10 to 100 times slower than compiled programs)
- - Often requires more memory space and is now rare3 for traditional highlevel languages.
- - Significant comeback with some Web scripting languages like PHP and JavaScript.
- - Interpreters usually implement as a read-eval-print loop:
  - Read expression in the input language (usually translating it in some internal form)
  - Evaluates the internal forms of the expression
  - Print the result of the evaluation
  - Loops and reads the next input expression until exit
- - Interpreters act as a virtual machine for the source language:
  - Fetch execute cycle replaced by the read-eval-print loop
  - Usually has a core component, called the interpreter “run-time” that is a compile program running on the native machine.

HYBRID IMPLEMENTAITON

- - This involves a compromise between compilers and pure interpreters. A high level program is translated to an intermediate language that allows easy interpretation.
- Hybrid implementation is faster than pure interpretation.
- Examples of the implementation occur in Perl and Java.
  - Perl programs are partially compiled to detect errors before interpretation.
  - Initial implementat6ions of Java were hybrid. The intermediate form, byte code, provides portability to any machine that has a byte code interpreter and a run time system (together, these are called Java Virtual Machine).

JUST-IN-TIME IMPLEMENTATION

- This implementation initially translates programs to an intermediate language then compile the intermediate language of the subprograms into machine code when they are called.
  - Machine code version is kept for subsequent calls. Just-in-time systems are widely used for Java programs. Also .NET languages are implemented with a JIT system.