Preliminaries

Instructor: C. Pu (Ph.D., Assistant Professor)

puc@marshall.edu

Reasons for Studying Concepts of Programming Languages

- The following is what we believe to be a complete list of potential benefits of studying concepts of programming languages:
  - *Increased capacity to express ideas*
    - It is widely believed that the depth at which people can think is influenced by the expressive power of the language in which they communicate their thoughts.
    - Programmers, in the process of developing software, are similarly constrained.
      - The language in which they develop software places limits on the kinds of control statement, data structures, and abstractions they can use.
      - Awareness of a wider variety of programming language features can reduce such limitations.

Reasons for Studying Concepts of Programming Languages

- The following is what we believe to be a complete list of potential benefits of studying concepts of programming languages:
  - *Improved background for choosing appropriate languages*
    - Many professional programmers have had little formal education in computer science.
      - The result is that many programmers, when given a choice of languages for a new project, use the language with which they are most familiar, even if it is poorly suited for the project at hand.
    - If these programmers were familiar with a wider range of languages and language constructs, they would be better able to choose the language with the features that best address the problem.

Reasons for Studying Concepts of Programming Languages

- The following is what we believe to be a complete list of potential benefits of studying concepts of programming languages:
  - *Increased ability to learn new languages*
    - The process of learning a new programming language can be lengthy and difficult, especially for someone who is comfortable with only one or two languages and has never examined programming languages concepts in general.
    - Once a thorough understanding the fundamental concepts of languages is acquired, it becomes far easier to see how these concepts are incorporated into the design of the languages being learned.

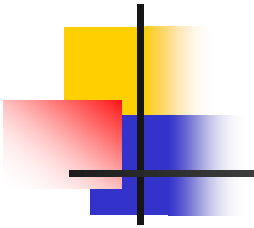
Reasons for Studying Concepts of Programming Languages

- The following is what we believe to be a complete list of potential benefits of studying concepts of programming languages:
  - *Better understanding of the significance of implementation*
    - In learning the concepts of programming languages, it is both interesting and necessary to touch on the implementation issues that affect those concepts.
    - In some cases, an understanding of implementation issues leads to an understanding of why languages are designed the way they are.
      - In turn, this knowledge leads to the ability to use a language more intelligently, as it was designed to be used.
      - We can become a better programmers by understanding the choices among programming languages constructs and the consequences of those choices.

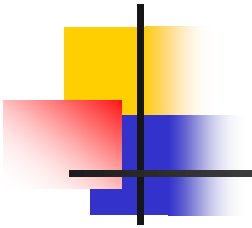
Reasons for Studying Concepts of Programming Languages

- The following is what we believe to be a complete list of potential benefits of studying concepts of programming languages:
  - *Better use of languages that are already known*
    - Many contemporary programming languages are large and complex.
    - It is uncommon for a programmer to be familiar with and use all of the features of a languages he or she uses.
    - By studying the concepts of programming languages, programmers can learn about previously unknown and unused parts of the languages they already use and begin to use those features.

Reasons for Studying Concepts of Programming Languages



- The following is what we believe to be a complete list of potential benefits of studying concepts of programming languages:
  - *Overall advancement of computing*
    - There is a global view of computing that can justify the study of programming language concepts.



Programming Domains

- Computers have been applied to a myriad of different areas, from controlling nuclear power plants to providing video games in mobile phones.

- Because of this great diversity in computer use, programming languages with very different goals have been developed.

Programming Domains

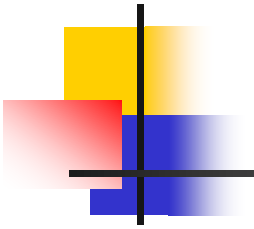
- *Scientific applications*
  - The first digital computers, which appeared in the late 1940s and early 1950s, were invented and used for scientific applications.
  - Typically, the scientific applications of that time used relatively simple data structures, but required large numbers of floating-point arithmetic computations.
  - The most common data structures were arrays and matrices.
  - The most common control structures were counting loops and selections.
  - The early high-level programming languages invented for scientific applications were designed to provide for those needs.
    - The first language for scientific applications was Fortran.
    - ALGOL 60 and most of its descendants were also intended to be used in this area.

Programming Domains

- *Business applications*
  - The use of computers for business applications began in the 1950s.
  - Special computers were developed for this purpose, along with special languages.
  - The first successful high-level language for business was COBOL, the initial version of which appeared in 1960.
    - It is still the most commonly used language for these applications.
  - Business languages are characterized by facilities for producing reports, precise ways of describing and storing decimal numbers and character data, and the ability to specify decimal arithmetic operations.

Programming Domains

- *Artificial intelligence*
  - Artificial intelligence is a broad area of computer applications characterized by the user of symbolic rather than numeric computations.
    - Symbolic computation means that symbols, consisting of names rather than numbers, are manipulated.
  - The first widely used programming language developed for AI applications was the functional language LISP, which appeared in 1959.
    - Most AI applications developed prior to 1990 were written in LISP.
  - During the early 1970s, an alternative approach to some of these applications appeared, logic programming using the Prolog language.



Programming Domains

- *Systems programming*
  - The operating system and the programming support tools of a computer system are collectively known as its system software.
  - Systems software is used almost continuously and so it must be efficient.
  - Furthermore, it must have low-level features that allow the software interfaces to external devices to be written.
  - The UNIX operating system is written almost entirely in C which has made it relatively easy to port, or move, to different machines.
  - Some of the characteristics of C make it a good choice for systems programming.
    - It is low level, execution efficient, and does not burden the user with many safety restrictions.

Programming Domains

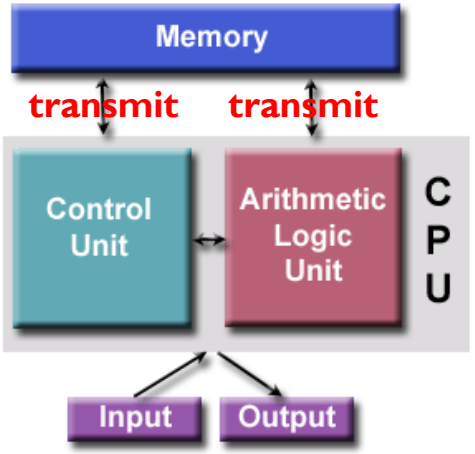
- *Web software*
  - The World Wide Web is supported by an electric collection of languages, ranging from markup languages, such as HTML, which is not a programming language, to general-purpose programming languages, such as Java.
  - Because of the pervasive need for dynamic Web content, some computation capability is often included in the technology of content presentation.
    - This functionality can be provided by embedding programming code in an HTML document.
    - Such code is often in the form of a scripting language, such as Javascript or PHP.

Influences on Language Design

- Factors influence the basic design of programming languages
  - Computer architecture
  - Programming design methodologies

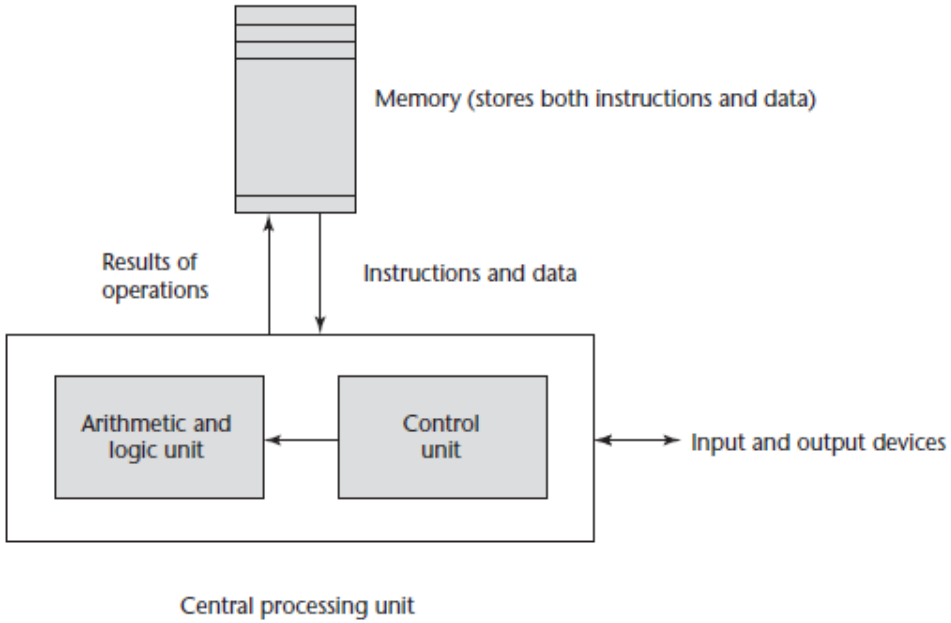
- The basic architecture of computers has had a profound effect on language design.

- Von Neumann architecture
  - Central Processing Unit (CPU)
    - Arithmetic logic unit
    - Control unit
  - Memory (stores data and instructions)
  - Input and output mechanism
  - External storage



Computer Architecture: Von Neumann Architecture

- The overall structure of a Von Neumann computer



1. Instructions and data must be transmitted from memory to CPU.
2. Results of operations in the CPU must be moved back to memory.

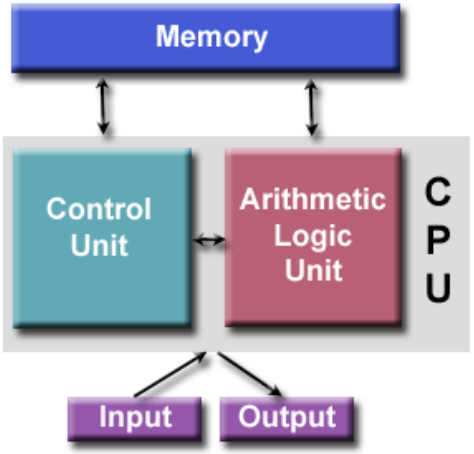
Computer Architecture: Imperative Languages

- Most of the popular languages that have been designed around the Von Neumann architecture are called ***imperative languages***.

- ***Imperative languages***
  - Focus on how the computer should do computation
  - Uses statements to change a program's state

- Because of the von Neumann architecture, the central features of imperative languages are
  - ***Variable***: variable models the memory cell
  - ***Assignment***: assignment statements are based on the piping operations
  - ***Loop***: the iterative form of repetition

Computer Architecture: Fetch-Execute Cycle



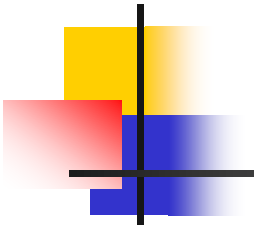
- ***Fetch-execute cycle*** algorithm:

```
initialize the program counter
repeat forever
    fetch the instruction pointed to by the program counter
    increment the program counter to point at the next instruction
    decode the instruction
    execute the instruction
end repeat
```

Summary:

- Programs reside in memory but are executed in the CPU
- Each instruction to be executed must be moved from memory to the processor
- The address of the next instruction to be executed is maintained in a register called the *program counter*

Computer Architecture: Fetch-Execute Cycle



- The "decode the instruction" step in the algorithm means the instruction is examined to determine what action it specifies.
- Program execution terminates when a stop instruction is encountered, although on an actual computer a stop instruction is rarely executed.
- Rather, control transfers from the operating system to a user program for its execution and then back to the operating system when the user program execution is complete.

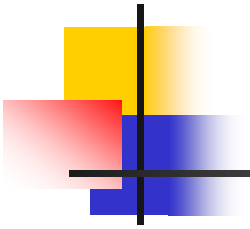
Programming Design Methodologies

- The late 1960s and early 1970s brought an intense analysis of both software development process and programming language design.
  - The shift in the major cost of computing from hardware to software.
    - As hardware cost decreased and programmer costs increased.
  - Progressively larger and more complex problems were being solved by computers.
- The new software development methodologies that emerged as a result of the research of the 1970s were called top-down design and stepwise refinement.
- The primary programming language deficiencies that were discovered were incompleteness of type checking and inadequacy of control statements.

Programming Design Methodologies

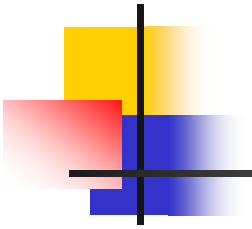
- In the late 1970s, a shift from procedure-oriented to data-oriented program design methodologies began.
  - Data-oriented methods emphasize data design, focusing on the use of abstract data types to solve problems.

- The latest step in the evolution of data-oriented software development, which began in the early 1980s, is object-oriented design.



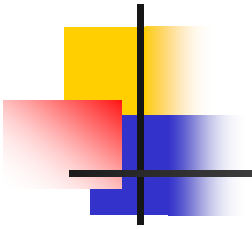
Programming Design Methodologies

- Object-oriented methodology begins with data abstraction, which encapsulates processing with data objects and controls access to data, and adds inheritance and dynamic method binding.
- Inheritance is a powerful concept that greatly enhances the potential reuse of existing software, thereby providing the possibility of significant increases in software development productivity.
  - This is an important factor in the increase in popularity of object-oriented languages.
- Dynamic (run-time) method binding allows more flexible use of inheritance.



Language Categories

- Programming languages are often categorized into four bins:
  - Imperative language
  - Functional language
  - Logic language
  - Object oriented language

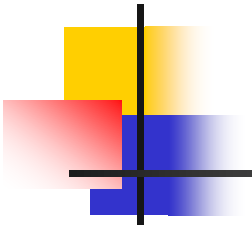


Language Categories

- Programming languages are often categorized into four bins:
  - Imperative language
    - In computer science, imperative programming is a programming paradigm that uses statements that change a program's state.
    - In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform.
  - Functional language
  - Logic language
  - Object oriented language

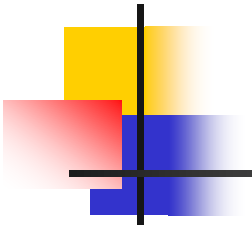
Language Categories

- Programming languages are often categorized into four bins:
  - Imperative language
  - Functional language
    - In computer science, functional programming is a programming paradigm where programs are constructed by applying and composing functions.
  - Logic language
  - Object oriented language



Language Categories

- Programming languages are often categorized into four bins:
  - Imperative language
  - Functional language
  - Logic language
    - Logic programming is a programming paradigm which is largely based on formal logic.
    - Any program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain.
  - Object oriented language



Language Categories

- Programming languages are often categorized into four bins:
  - Imperative language
  - Functional language
  - Logic language
  - Object oriented language
    - Object-oriented programming is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields, and code, in the form of procedures.
    - A feature of objects is that an object's own procedures can access and often modify the data fields of itself.