Programming Paradigms

Programming languages

• A Programming language is a notational system for describing tasks/computations in a machine and human readable form.

• Most computer languages are designed to facilitate certain operations and not others: numerical computation, or text manipulation, or I/O.

• More broadly, a computer language typically embodies a particular programming paradigm.

Characteristics of a programming language:
Every language has syntax and semantics:

• Syntax: The syntax of a program is the form of its declarations, expressions, statements and program units.

• Semantic: The semantic of a program is concerned with the meaning of its program.
Which programming language?

- Since a task can be solved in different ways (paradigms), the language used to describe the solution differs in abstractions, structures due to the way in which the problem is solved.

- There is no theory that dictates the best paradigm to solve a particular problem.

- Efforts by Sebesta in his Concepts of Programming Languages book:
  
  - He based his evaluation criteria on three factors and 9 characteristics.
  
  - The three criteria (R,W,R) are:
    - Readability
    - Writability
    - Reliability
• The nine characteristics are:
  ▪ Simplicity/orthogonality (R,W,R):
    • “Orthogonality in a programming language means that a relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language” [Sebesta]
    • Relatively small set of primitive constructions combined in a number (of logically consistent) ways to provide the required control and data structures.
  ▪ The concepts of a programming language do not interfere with each other: different methods of passing parameters.
  ▪ Non-orthogonality: means exceptions to the general language rules, which make it harder to learn. It means that you cannot combine language features in all possible ways.
  ▪ Examples of non-orthogonal languages:
    o Arrays in Perl4 can't contain other arrays.
    o In C, parameters can be passed by value, unless they are arrays, in which case they are passed by reference.
  ▪ Control structures (R,W,R)
  ▪ Data types and structures (R,W,R)
  ▪ Syntax design (R,W,R)
- Support for abstraction ,(W,R)
- Expressivity ,(W,R):
  - Programming languages with poor support for abstraction and weak primitives will have poor writability.
- Type checking ,(R)
- Exception handling ,(R)
- Restricted aliasing ,(R):
  - example: (C)

```c
int salary, *p_salary;
salary = 98000;
p_salary = &salary;
salary and *p_salary are aliases.
```

<table>
<thead>
<tr>
<th>Syntax design</th>
<th>Readability</th>
<th>Writability</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control structures</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Data types</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Simplicity/orthogonality</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Abstraction</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Expressivity</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Type checking</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Exception handling</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Restricted aliasing</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
</tbody>
</table>
• Maintainability
  o **Factoring**: The ability to group related features into a single unit. Use subroutines to group related computations units so they can be re-used in different parts of the application.
  o **Locality**: The ability to implement information hiding so that changes to a grouping (either control or data) are transparent.

• Cost
  o Programmer training
  o Software creation
  o Compilation
  o Execution
  o Compiler cost
  o Poor reliability
  o Maintenance

• Others: portability and generality
Programming paradigms

- The paradigms are not exclusive, but reflect the different emphasis of language designers. Most practical languages embody features of more than one paradigm.

Classification:

<table>
<thead>
<tr>
<th>Imperative/Algorithmic</th>
<th>Declarative</th>
<th>Object-Oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Functional Programming</td>
<td>Logic Programming</td>
</tr>
<tr>
<td>Algol</td>
<td>Lisp</td>
<td>Prolog</td>
</tr>
<tr>
<td>Cobol</td>
<td>Haskell</td>
<td></td>
</tr>
<tr>
<td>PL/1</td>
<td>ML</td>
<td></td>
</tr>
<tr>
<td>Ada</td>
<td>Miranda</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>APL</td>
<td></td>
</tr>
<tr>
<td>Modula-3</td>
<td></td>
<td>Smalltalk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simula</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C++</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Java</td>
</tr>
</tbody>
</table>

Imperative paradigms

- It is based on commands that update variables in storage. The Latin word *imperare* means “to command”.
- The language provides statements, such as *assignment statements*, which explicitly change the *state* of the memory of the computer.
- This model closely matches the actual executions of computer and usually has high execution efficiency.
• Many people also find the imperative paradigm to be a more natural way of expressing themselves.
**Functional programming paradigms**

- In this paradigm we express computations as the evaluation of mathematical functions.

- Functional programming paradigms treat values as single entities. Unlike variables, values are never modified. Instead, values are transformed into new values.

- Computations of functional languages are performed largely through applying functions to values, i.e., (+ 4 5).
Logic programming paradigms

- In this paradigm we express computation in exclusively in terms of *mathematical logic*.

- While the functional paradigm emphasizes the idea of a mathematical function, the logic paradigm focuses on predicate logic, in which the basic concept is a *relation*.

- Logic languages are useful for expressing problems where it is not obvious what the functions should be.

- For example consider the *uncle* relationship: a given person can have many *uncles*, and another person can be *uncle* to many *nieces* and *nephews*.

- Let us consider now how we can define the *brother* relation in terms of simpler relations and properties *father, mother, and male*. Using the Prolog logic language one can say:

```
brother(X,Y)      /* X is the brother of Y */
                 /* if there are two people F and M for which */
father(F,X),     /* F is the father of X */
father(F,Y),     /* and F is the father of Y */
mother(M,X),     /* and M is the mother of X */
mother(M,Y),     /* and M is the mother of Y */
male(X).         /* and X is male */
```
The Object-Oriented Paradigm

- OO programming paradigm is not just a few new features added to a programming language, but it a new way of thinking about the process of decomposing problems and developing programming solutions.

- Alan Kay characterized the fundamental of OOP as follows:
  - Everything is modeled as object
  - Computation is performed by message passing: objects communicate with one another via message passing.
  - Every object is an instance of a class where a class represents a grouping of similar objects.
  - Inheritance: defines the relationships between classes.

- The Object Oriented paradigm focuses on the objects that a program is representing, and on allowing them to exhibit "behavior".

- Unlike imperative paradigm, where data are passive and procedures are active, in the O-O paradigm data is combined with procedures to give objects, which are thereby rendered active.
Concurrent programming

- Improve performance
- Multiprogramming systems attempt to utilize resources that would otherwise be wasted, by running two or more jobs concurrently.
- Multiaccess systems extend this principle, allowing many jobs to be run, each on behalf of a user at an interactive terminal.

- Concurrency can be classified into:
  - Apparent concurrency: single processor (interleaved execution of concurrent tasks)
  - Real concurrency: multiprocessor environment

- Issues:
  - How to synchronize the interactions among concurrently executing processes to maintain the internal data integrity.
  - Another problem is to schedule the racing processes for a limited set of shared resources.