**COMP205**
Comparative Programming Languages

Grant Malcolm (grant@csc.liv.ac.uk)

- Introduction to programming languages
- The imperative paradigm
- The functional paradigm
- Other paradigms and concluding remarks

**BOOKS**


**TUTORIALS**

<table>
<thead>
<tr>
<th>Day</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>15:00-17:00</td>
</tr>
<tr>
<td>Tuesday</td>
<td>14:00-16:00</td>
</tr>
<tr>
<td>Thursday</td>
<td>11:00-13:00</td>
</tr>
<tr>
<td>Friday</td>
<td>10:00-12:00</td>
</tr>
</tbody>
</table>

- Four COMP2XX tutorial slots have been timetabled

**INTRODUCTION TO PROGRAMMING PARADIGMS**

http://www.csc.liv.ac.uk/~grant/Teaching/COMP205/

1. Paradigms and the classification of languages
2. Program structure and programming languages as communications media
3. Complexity and program processing

**CLASSIFICATION OF PROGRAMMING LANGUAGES**

To facilitate discussion on any subject it is convenient to group together similar facets of the subject according to some grouping notion. Computer programming languages are no exception.

1. Machine, Assembler and High Level Languages
2. Chronological order of development
3. Generations
4. Levels of abstraction (from machine level)
5. Declarative v Non-declarative
6. Paradigms

**OPERATION OF A COMPUTER PROGRAM**

- A computer program resides in primary memory where it is represented as a set of machine instructions which in turn are represented as sequences of binary digits.
- At any point in time the computer is said to be in a particular state.
- A central feature of the state is the instruction pointer which points to the next machine instruction to be executed.
- The execution sequence of a group of machine instructions is known as the flow of control.

**MACHINE CODE**

- Thus, a program running on a computer is simply a sequence of bits.
- A program in this format is said to be in machine code.
- We can write programs in machine code:

  | 23fc | 0000 | 0001 | 0000 | 0040 |
  | 0cb9 | 0000 | 000a | 0000 | 0040 |
  | 6e0c | 06b9 | 0000 | 0001 | 0000 | 0040 |
  | 60e8 |
**ASSEMBLY LANGUAGE**

- Assembly language (or assembler code) was our first attempt at producing a mechanism for writing programs that was more palatable to ourselves.
- Of course a program written in machine code, in order to “run”, must first be translated (assembled) into machine code.

```
move  #0x1, n
compare:
  cmpl  #0xa, n
cgt   end_of_loop
acddi #0x1, n
bra   compare
end_of_loop:
```

**HIGH LEVEL LANGUAGE**

- From the foregoing we can see that assembler language is not much of an improvement on machine code!
- A more problem-oriented (rather than machine-oriented) mechanism for creating computer programs would also be desirable.
- Hence the advent of higher level languages commencing with the introduction of “Autocodes”, and going on to Algol, Fortran, Pascal, Basic, Ada, C, etc.

**Classification of programming languages:**

1. Machine, Assembler and High Level Languages
2. **Chronological order of development**
3. Generations
4. Levels of abstraction (from machine level)
5. Declarative v Non-declarative
6. Paradigms

**CHRONOLOGICAL CLASSIFICATION OF PROGRAMMING LANGUAGES**

1940s Prelingual phase: Machine code
1950s Exploiting machine power: Assembler code, Autocodes, first version of Fortran
1960s Increasing expressive power: Cobol, Lisp, Algol 60, Basic, PL/1 --- but most “proper” programming still done in assembly language.

**1970s Fighting the “software crisis”:**

2. Increasing program correctness - Structured Programming, modular programming and information hiding.

Examples include Pascal, Algol 68 and C.

**1980s reducing complexity – object orientation, functional programming.**

**1990s exploiting parallel and distributed hardware (going faster!),**

- e.g. various parallel extensions to existing languages and dedicated parallel languages such as occam.

**2000s Genetic programming languages, DNA computing, bio-computing?**
THE SOFTWARE CRISIS

- The phrase software crisis alludes to a set of problems encountered in the development of computer software during the 1960s when attempting to build larger and larger software systems using existing development techniques.
- As a result:
  - 1. Schedule and cost estimates were often grossly inaccurate.
  - 2. Productivity of programmers could not keep up with demand.
  - 3. Poor quality software was produced.
- To address these problems the discipline of software engineering came into being.

Classification of programming languages:

1. Machine, Assembler and High Level Languages
2. Chronological order of development
3. Generations
4. Levels of abstraction (from machine level)
5. Declarative v Non-declarative
6. Paradigms

LANGUAGE GENERATIONS

<table>
<thead>
<tr>
<th>Generation</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Machine languages</td>
</tr>
<tr>
<td>2nd</td>
<td>Assembly languages</td>
</tr>
<tr>
<td>3rd</td>
<td>Procedural languages</td>
</tr>
<tr>
<td>4th</td>
<td>Application languages (4GLs)</td>
</tr>
<tr>
<td>5th</td>
<td>AI techniques, inference languages</td>
</tr>
<tr>
<td>6th</td>
<td>Neural networks (?), others….</td>
</tr>
</tbody>
</table>

Classification of programming languages:

1. Machine, Assembler and High Level Languages
2. Chronological order of development
3. Generations
4. Levels of abstraction (from machine level)
5. Declarative v Non-declarative
6. Paradigms

LANGUAGE LEVELS OF ABSTRACTION  
(Bal and Grane 94)

<table>
<thead>
<tr>
<th>Level</th>
<th>Instructions</th>
<th>Memory handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low level</td>
<td>Simple machine-like instructions</td>
<td>Direct memory access and allocation</td>
</tr>
<tr>
<td>High level</td>
<td>Expressions and explicit flow of control</td>
<td>Memory access and allocation through operators</td>
</tr>
<tr>
<td>Very high</td>
<td>Fully abstract machine</td>
<td>Fully hidden memory access and automatic allocation</td>
</tr>
<tr>
<td>languages</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Classification of programming languages:

1. Machine, Assembler and High Level Languages
2. Chronological order of development
3. Generations
4. Levels of abstraction (from machine level)
5. Declarative v Non-declarative
6. Paradigms
DECLARATIVE v NON-DECLARATIVE PROGRAMMING

Languages can also be classified by the emphasis they put on “what is to be achieved” against “how it is to be achieved”.

The first are said to be **declarative** (e.g. functional and logic languages).

The second is said to be **non-declarative** or **procedural** (e.g. imperative languages).

---

Classification of programming languages:

1. Machine, Assembler and High Level Languages
2. Chronological order of development
3. Generations
4. Levels of abstraction (from machine level)
5. Declarative v Non-declarative
6. Paradigms

---

PROGRAMMING PARADIGMS?

- In science a paradigm describes a set of techniques that have been found to be effective for a given *problem domain* (i.e. somebody somewhere must believe in it).
- A paradigm can typically be expressed in terms of a single principle (even if this is in fact an over simplification).
- This principle must be supported by a set of techniques.
- In the context of programming languages we say that a paradigm induces a particular way of thinking about the programming task.

---

We can identify four principal programming paradigms:

1. Imperative (e.g. Pascal, Ada, C).
2. Object-oriented (e.g. Java).
3. Functional (e.g. Haskell, SML).
4. Logic (e.g. Prolog).

---

PROGRAMMING MODELS

- The 4 main programming paradigms aim at solving general programming problems, but sometimes there are additional aspects to a problem which require us to “tweak” a paradigm.
- The result is not a new paradigm but a *programming model* founded on a particular paradigm.
- An example is parallel or distributed programming.

---

SUMMARY

- Classification of languages:
  1. Machine, assembler & high level
  2. Chronological order
  3. Generations
  4. Levels of abstraction
  5. Declarative v Non-declarative.
- Paradigms
- Programming models