1 - Programming Language Generations

- There are numerous "takes" in this!
- Here, we are using MacLennan's take, from the course text;

2 - Characteristics of First-Generation Programming Languages (page 1)

[source: MacLennan, p. 92]

- the classic example: FORTRAN
- "In general, ... the **structures** of first generation languages are based on the structures of the **computers** in the early 1960's"
  - "...natural, since the only experience people had in programming was in programming these machines"...!

3 - First-Generation (page 2)

- "This machine orientation is especially apparent in first generation **control structures**"
  - non-nested
  - "depend heavily on the GOTO for building any but the simplest control structures"
  - "One exception ... the definite iteration statement [FORTRAN's DO-loop] ... which IS hierarchical in first-generation languages."
- "Recursive procedures are not permitted in most first-generation languages (BASIC is an exception)"
- "there is generally only **one** parameter passing mode (typically, pass by reference)"

4 - First-Generation (page 3)

- "machine orientation ... can also be seen in the types of **data structures** provided ... patterned after the layout of memory on the computers available around 1960."
  - "data structure primitives ... are fixed and floating point numbers of various precisions, characters, and logical values --
  - ...just the kinds of values manipulated by the instructions on these computers"
- "The data structure **constructors** are **arrays** and, in business-oriented languages, **records**, which are the ways storage was commonly organized."
- "As with control structures, first-generation languages provide little facility for **hierarchical** data organization (an exception is COBOL's record structure). That is, data structures cannot be nested."

5 - First-Generation (page 4)

- "characterized by a **relatively weak type system**;
  - that is, it is easy to **subvert** the type system or do representation-dependent programming."
  - "(Machine independence and portability were not major concerns in the first generation.)
- "Hierarchical structure is also absent from first-generation name structures, with disjoint scopes being the rule."
  - "variable names are bound directly and statically to memory locations since there is no dynamic memory management."

6 - First-Generation (page 5)

- "syntactic structures ... are characterized by a card-oriented, **linear** arrangement of statements patterned after assembly language"
  - "...most of these languages had numeric statement labels that are suggestive of machine addresses"
- BUT they "go significantly beyond assembly languages ... in their provision of algebraic notation"
- "Their usual lexical conventions are to ignore blanks and to recognize keywords in context."

7 - First-Generation (page 6)

- "In summary, the salient characteristics of the first generation are:
  - **machine** orientation and
  - **linear** structures."
- ... "**second** generation makes important moves in the directions of
  - **application** orientation and
  - **hierarchical** structure."

8 - Characteristics of Second-Generation Programming Languages (page 1)

[source: MacLennan, pp. 163-164]

- the first, and classic, example: Algol-60
- "second-generation structures are **elaborations** and **generalizations** of the corresponding first-generation structures"
- data structures:
  - still "very close to first-generation structures",
  - with only simple generalizations (different lower bounds, dynamic arrays)
- still linear; closely patterned on machine-addressing modes

9 - Second-Generation (page 2)

- "usually have strong-typing of the built-in types"
- name structures: one of the 2nd generation's biggest contributions is here: **hierarchical nesting**!
  - better control of name space
  - permits efficient dynamic memory allocation
- "The introduction of **block structure** is perhaps the most characteristic attribute of this language generation."

10 - Second-Generation (p. 3)

- structured control structures - "which, by hierarchically structuring the control flow, eliminate the need for confusing networks of goto's"
- "also **elaborated** many of the first generation's control structures" --
- sometimes with good results:
  - recursive procedures,
  - the idea of a choice of parameter-passing modes
- sometimes, not (or at least with more questionable results)!
  - the proliferation of **baroque** and **expensive** constructs

11 - Second-Generation (p. 4)

- syntactic structures:
  - "saw a shift **away** from fixed formats,
  - toward **free**-format languages with **machine-independent** lexical conventions"
- "a number of languages shifted to **keyword** or **reserved word** policies, although the keyword-in-context rule was also used (PL/I)"

12 - Second-Generation (p. 5)

- "In general, the second generation can be seen as the full flowering of the technology of language design and implementation.
- The many new techniques developed in this period encouraged unbridled generalization - with both desirable and undesirable consequences."
- "We will see that the third generation tried to **compensate** for the excesses while **retaining** the accomplishments."

13 - Characteristics of Third-Generation Programming Languages (page 1)

[source: MacLennan, pp. 208-209]

- the classic example: Pascal
- show an emphasis on **simplicity** and **efficiency**
  - generally a reaction against the **excesses** of the second generation
- syntactic structures: essentially those of the second generation

14 - Third-Generation (page 2)

- data structures: shift of emphasis from the **machine** to the **application**
  - provision of **user-defined data types** -- now users can create the data types needed by their applications;
  - also exemplified by application-oriented type constructors, like sets, subranges, and enumeration types
- also characterized by the ability to nest data structures to any depth! (to organize data hierarchically)
- name structures: generally some simplification of Algol-60 block structure
  - BUT, also "typically have new **binding** and **scope-defining** constructs, often associated with data type constructors, such as records and enumeration types"

15 - Third-Generation (page 3)

- control structures: **simplified**, **efficient** versions of those found in the second generation;
  - especially apparent in Pascal's for-loop
  - Pascal also provided two separate constructs for indefinite iteration: while-loop, repeat-loop
  - rejection of name parameters and similar delayed-evaluation mechanisms
- also often include application-oriented control structures, such as Pascal's case-statement

16 - Third-Generation (page 4)

- in summary: combines **practical** engineering principles with the **technical** achievements of the 2nd generation --
  - the result, especially in the case of Pascal, is a **simple**, **efficient**, **secure** tool for many applications

17 - Characteristics of Fourth-Generation Programming Languages (page 1)

[source: MacLennan, pp. 305-306]

- the classic example: Ada
- rather more non-standard than his first 3; but then, as he notes that he means fourth generation **programming** languages;
  - "fourth-generation language" is sometimes used "to refer to application generator programs, which might or might not be programming languages in the technical sense discussed in the first two pages of the Introduction."
- Sometimes I've seen 4GL's characterized as languages where you indicate **what** you want, and not **how** to get it -- that's another different "direction";

18 - Fourth-Generation (page 2)

- some characteristics are just a **consolidation** and **correction** of certain third generation characteristics;
- others are important new facilities;
- most important contribution: in **name structures**
  - MacLennan's fourth generation is essentially synonymous with **data abstraction language**!
- primary characteristic: provision of an encapsulation facility supporting:
  - the separation of specification and definition
  - information hiding
  - name access by mutual consent

19 - Fourth-Generation (page 3)

- "Most of these languages allow encapsulated modules to be **generic** (or **polymorphic**)", which can lead to operator identification issues;"
- control structures: "It is characteristic of this generation to provide for **concurrent programming**"
  - "Most ... use some form of message-passing as a means of synchronization and communication among concurrent tasks"
  - "Protected data structures ... are also typical."
  - "On the other hand, the **basic** framework of these languages is still **sequential**."
- "typically also have a dynamically-scoped **exception mechanism** for handling both system- and user-defined errors"

20 - Fourth-Generation (page 4)

- **data structure constructors**: similar to those of the third generation, except some problems (**array parameters!**) have been corrected;
- "**primitive data structures** tend to be more **complicated** than the third generation, because of the desire to control accuracy and precision in numeric types"
- "**syntactic structures** ... are largely those of the second and third [generations]... in the Algol/Pascal tradition.
  - The major exception is a preference for **fully-bracketed structures**."
  - ^ ...which, for example, are another solution to the dangling-else problem!

21 - Another language family... (page 1)

[source: MacLennan, p. 207-208]

- "In the early 1960's teams at Cambridge and London Universities developed a semantically sophisticated but very complex language called CPL"
  - "explained as "Cambridge Plus London" or "Combined Programming Language"
  - "The latter acronym hints at its complexity,
  - ...for CPL exhibits the full **baroque** flowering of the second generation" [Algol-60's generation, remember!]

22 - Another family... (page 2)

- "Like Algol-60, CPL posed implementation challenges for its designers,
- ...so they wanted to implement it in the "best" programming language: CPL."
  - not as weird as it sounds -- "Pascal was implemented in Pascal;
  - it was expected of any **decent** general purpose language that it would be the best vehicle for implementing its own compiler"...!
- BUT, in the case of CPL,
  - "To simplify this process Martin Richards designed (1967) a subset of CPL, called **BCPL** for "Basic CPL", which included only those features **ESSENTIAL** for **systems** implementation."

23 - Another family... (page 3)

- an "alternative" meaning to BCPL:
  - "Badly Constructed Programming Language"
  - (the joking-acronym heard by one CS student at Rice in the late 1970's/early 1980's (OK, my husband) from another CS student there...)
- Ultimately [ironically?], the CPL project died away, but BCPL became a moderately popular systems implementation language in the early 1970's"
  - (contemporary with early Pascal/early Prolog?)

24 - Another family... (page 4)

- Why mention CPL and BCPL?
- ...because one of the places where BCPL was popular was at Bell Labs "...when the earliest versions of Unix (for an 8K PDP-7!) was being developed"
  - "Since the Unix project needed a systems implementation language, in 1969-1970 Ken Thompson designed a language called "B"
- In a 1993 SIGPLAN article, Dennis Ritchie describes B as so:
  - "it is BCPL squeezed into 8K bytes of memory and filtered through Thompson's brain"...!
- "At first there was little concern for portability, and the language was very close to the machine";

25 - Another family... (page 5)

- "In particular, like BCPL and many other systems implementation languages, B was **typeless**;
  - that is, it had a **single data type** corresponding to a word of PDP-7 memory"!!
  - "This is, of course, the **extreme** of weak typing, and is more typical of assembly language than even of first-generation languages, which typically have several data types and some notion of type checking."
- "Many other language design decisions were dictated by the **limited memory** available to compile B on the PDP-7."

26 - Another family... (page 6)

- So - a PDP-11 arrived in 1970 [at Bell Labs], and "With [its] arrival ... the Unix team became aware of problems with B's addressing scheme,
  - which was **incompatible** with memory addressing on the PDP-11."
- "Therefore, in 1971 Dennis M. Ritchie began extending B to include rudimentary data types (for purposes of memory allocation and addressing, NOT type checking),
  - inspired in many respects by Algol-68." [!]
- "Eventually he called this language **C**, as the successor of B;
  - its evolution was mostly complete by 1973,
  - when it was used to rewrite the kernel of the Unix operating system."

27 - Another family... (page 7)

- Like most of the languages we've discussed, C had some early modifications;
  - "Some additional third-generation features, such as **union** and **enumeration types**, were added in the late 1970's,
  - but attempts to port Unix to other computers accented the portability and security issues of weak typing."
- "Therefore a more restricted type system was designed, but it was not enforced [!] by most compilers;
  - instead programmers had to rely on a separate type-checker (called **lint**)."

28 - Another family... (page 8)

- In a way, the evolution of C recapitulates the history of programming languages,
  - with its shift from **efficiency** to **portability** and **security**."
- "Ritchie acknowledges that [C] contains many infelicities,
  - some of which result from attempting to maintain upward compatibility with B and BCPL;
  - others are "historical accidents or mistakes". "

29 - Another family... (page 9)

- "The first published description (The C Programming Language by Kernighan and Ritchie) was published in 1978,
  - but it was not a language definition per se, since it was vague about many issues, and it was not consistent with the "reference compiler" (pcc, the portable C compiler)."
- "Many of these problems were solved by the development of an ANSI standard C, which began in 1983; it was approved in 1989."
- "During the late 1970's and early 1980's, use of Unix spread outside of AT&T, mostly within the university and industrial research communities, and C spread with it."

30 - Another family... (page 10)

- "By the late 1980's it had become a popular language for programming personal computers (for which, again, **efficiency** was often critical)."
- "It is ALSO used as an **output language** for compilers of OTHER languages (much as a structured assembler might be used),
  - [does that make it something like a BYTECODE for those languages?! 8-)]
- ...and [of course] has been the basis for other languages, such as C++"

31 - Another family... (page 11)

- Why didn't we cover C as one of our languages thus far?
  - MacLennan also makes an interesting argument that C **mixes** characteristics of **three** language generations;
- "As a consequence of its **history**, C combines characteristics of **several** generations."
- Third generation:
  - "it [does have] some **third-generation** features, such as **hierarchical data structures**."
  - (structs, which CAN be nested; not unlike Pascal's records?)

32 - Another family... (page 12)

- Second generation:
  - "Also, it borrows many ideas from **second** generation languages, such as Algol-68, CPL, and PL/I."
  - e.g., "its low-level model of arrays and pointers complicates or precludes optimization on some computers [!]"
  - "(in effect, C's storage model is lower level than the machine's)"!

33 - Another family... (page 13)

- First generation:
  - "In some ways it even returns to the first generation"; e.g.,"it **does not permit** nested procedures or environments."
  - "Thus is has poor support for **modular programming** (a key issue addressed by fourth-generation languages")
  - "It also resurrected some first generation syntactic conventions, such as using = for assignment and requiring declarations to start with a keyword."

34 - Another family... (page 14)

- But - "Perhaps we should not be surprised at the reappearance of **first generation** characteristics in C;
- at least some were a **direct consequence** of its orientation to a machine, the 8K PDP-7, of comparable power to the machines for which the first generation languages were designed." ...!
  - "Thus we may say that C was motivated by concerns **similar** to those that motivated first-generation language designers"...!

35 - Another family... (page 15)

- "Ritchie remarks that "C is quirky, flawed, and an enormous success." "
- "Aside from riding on the back of Unix popularity,
- he suggests that the success of C can be attributed to its
  - simplicity, [!],
  - efficiency,
  - portability, [!]
  - closeness to the machine, [aren't those contradictory?]
  - and its evolution in an environment in which it was used to write practical programs."