CS 220: Survey of Programming Languages

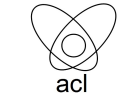
Lecture Slides

Imperative Programming Languages

Jan Michael C. Yap

Algorithms and Complexity Laboratory
Department of Computer Science
University of the Philippines, Diliman
jcyap@dcs.upd.edu.ph

Session 4



Imperative Programming Languages

Imperative Programming Languages

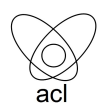
FORTRAN

Ada

Pascal

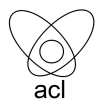
Imperative Programming Languages

Characteristics of an imperative PL



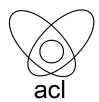
Characteristics of an imperative PL

- Statements are executed in a step-wise, sequential, manner



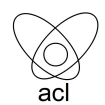
Characteristics of an imperative PL

- Statements are executed in a step-wise, sequential, manner
  - Order of execution is crucial



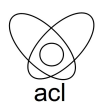
Characteristics of an imperative PL

- Statements are executed in a step-wise, sequential, manner
  - Order of execution is crucial
- Destructive assignment

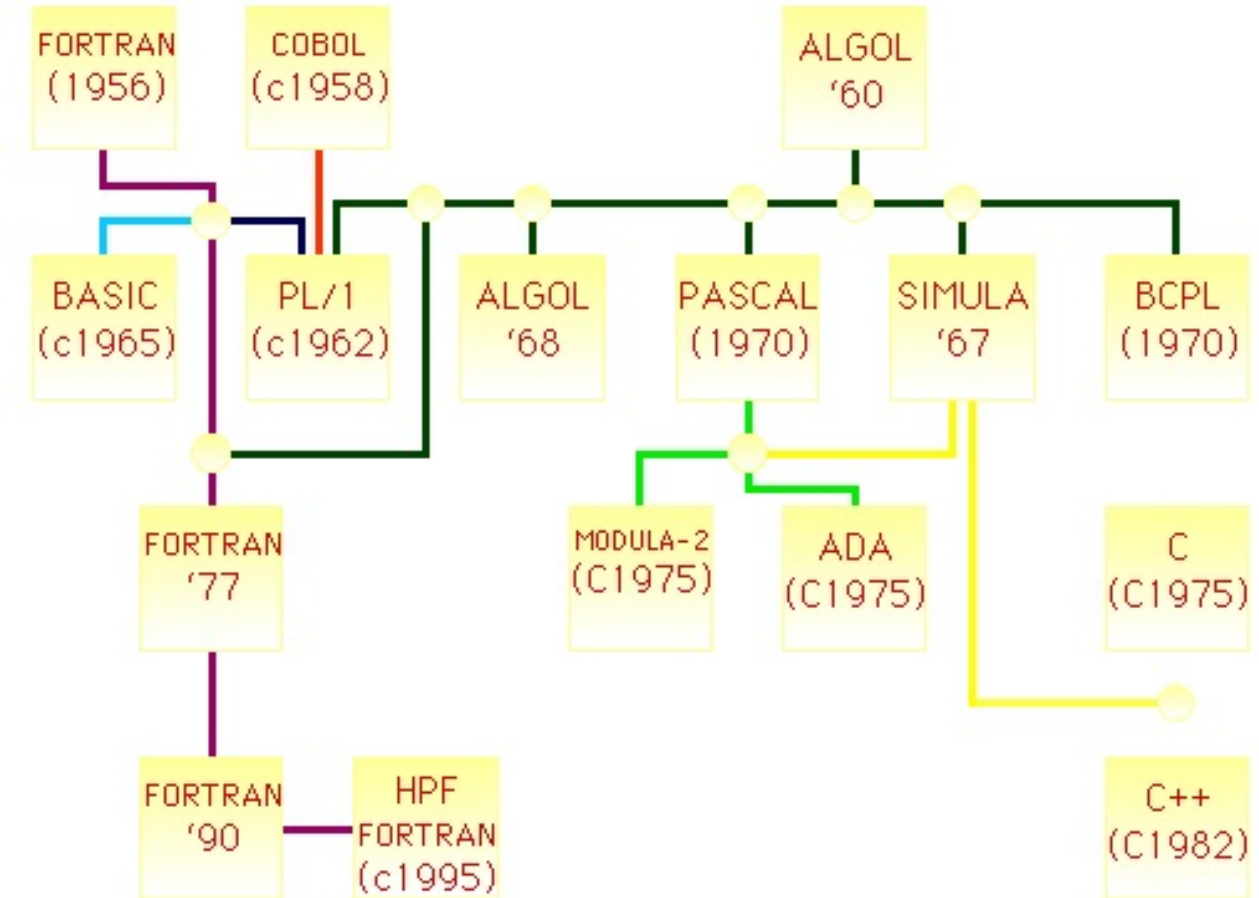


Characteristics of an imperative PL

- Statements are executed in a step-wise, sequential, manner
  - Order of execution is crucial
- Destructive assignment
- Control is the responsibility of the programmer

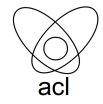


"Lineage" of imperative PLs

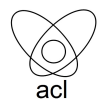


http://cgi.csc.liv.ac.uk/~grant/Teaching/COMP205/intro.html

Imperative Programming Languages



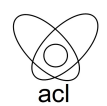
Background



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/fortran/fortran.html

Background

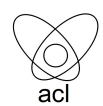
- FORTRAN (stands for FORmula TRANslation) was developed by a team of programmers at IBM led by John Backus, and was first published in 1957



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/fortran/fortran.html

Background

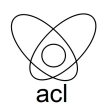
- FORTRAN (stands for FORmula TRANslation) was developed by a team of programmers at IBM led by John Backus, and was first published in 1957
- FORTRAN was the first high-level language, using the first compiler ever developed



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/fortran/fortran.html

Background

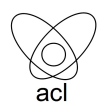
- FORTRAN (stands for FORmula TRANslation) was developed by a team of programmers at IBM led by John Backus, and was first published in 1957
- FORTRAN was the first high-level language, using the first compiler ever developed
  - Credited for giving rise to compiler theory



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/fortran/fortran.html

Evaluation

- Data types[1]



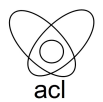
[1] http://cs.ubishops.ca/ljensen/fortran/pointer.htm

Evaluation

- Data types[1]
  - Integers, real single and double precision floating point numbers, complex numbers, characters, logical as primitive types

[1] http://cs.ubishops.ca/ljensen/fortran/pointer.htm



Evaluation

- Data types[1]
  - Integers, real single and double precision floating point numbers, complex numbers, characters, logical as primitive types
  - Support for multidimensional arrays, which can be unbounded

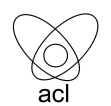
[1] http://cs.ubishops.ca/ljensen/fortran/pointer.htm

Evaluation

- Data types[1]
  - Integers, real single and double precision floating point numbers, complex numbers, characters, logical as primitive types
  - Support for multidimensional arrays, which can be unbounded
  - Pointers are derived types

[1] http://cs.ubishops.ca/ljensen/fortran/pointer.htm



Evaluation

- Data types[1]
  - Integers, real single and double precision floating point numbers, complex numbers, characters, logical as primitive types
  - Support for multidimensional arrays, which can be unbounded
  - Pointers are derived types
  - Allows for derived structures using the `TYPE - END TYPE` declaration

[1] http://cs.ubishops.ca/ljensen/fortran/pointer.htm

Evaluation

- Syntax design (Fortran 90[2])

[2] http://slebok.github.io/zoo/fortran/f90/derricks/extracted/index.html
[3] http://p.web.umkc.edu/pgd5ab/www/fortran_abstraction_and_encapsul.htm

Evaluation

- Syntax design (Fortran 90[2])
  - Number of production rules: 331
  - Number of top alternatives: 751
  - Number of symbols: 3,718
  - Vocabulary
    - Nonterminal symbols: 345
    - Terminal symbols: 173

[2] http://slebok.github.io/zoo/fortran/f90/derricks/extracted/index.html
[3] http://p.web.umkc.edu/pgd5ab/www/fortran_abstraction_and_encapsul.htm

Evaluation

- Syntax design (Fortran 90[2])
  - Number of production rules: 331
  - Number of top alternatives: 751
  - Number of symbols: 3,718
  - Vocabulary
    - Nonterminal symbols: 345
    - Terminal symbols: 173
- Abstraction[3]

[2] http://slebok.github.io/zoo/fortran/f90/derricks/extracted/index.html
[3] http://p.web.umkc.edu/pgd5ab/www/fortran_abstraction_and_encapsul.htm

Evaluation

- Syntax design (Fortran 90[2])
  - Number of production rules: 331
  - Number of top alternatives: 751
  - Number of symbols: 3,718
  - Vocabulary
    - Nonterminal symbols: 345
    - Terminal symbols: 173
- Abstraction[3]
  - References to non-local variables were through `COMMON` blocks

[2] http://slebok.github.io/zoo/fortran/f90/derricks/extracted/index.html
[3] http://p.web.umkc.edu/pgd5ab/www/fortran_abstraction_and_encapsul.htm

Evaluation

- Syntax design (Fortran 90[2])
  - Number of production rules: 331
  - Number of top alternatives: 751
  - Number of symbols: 3,718
  - Vocabulary
    - Nonterminal symbols: 345
    - Terminal symbols: 173
- Abstraction[3]
  - References to non-local variables were through `COMMON` blocks
  - `MODULE` to allow blocks of parameters, variables, and subprograms to be both shared and re-used.

[2] http://slebok.github.io/zoo/fortran/f90/derricks/extracted/index.html
[3] http://p.web.umkc.edu/pgd5ab/www/fortran_abstraction_and_encapsul.htm

Evaluation

- Expressivity

[4] http://www.southampton.ac.uk/~fangohr/randomnotes/iop_cpg_newsletter/2007_1

[5] see https://www.nsc.liu.se/~boein/ifip/kyoto/reid.txt

Evaluation

- Expressivity
  - On its own, it isn't as expressive as modern imperative language (e.g. no shortcut operators)

[4] http://www.southampton.ac.uk/~fangohr/randomnotes/iop_cpg_newsletter/2007_1
[5] see https://www.nsc.liu.se/~boein/ifip/kyoto/reid.txt

Evaluation

- Expressivity
  - On its own, it isn't as expressive as modern imperative language (e.g. no shortcut operators)
  - There are some efforts to improve expressiveness, albeit through modules[4]

[4] http://www.southampton.ac.uk/~fangohr/randomnotes/iop_cpg_newsletter/2007_1

[5] see https://www.nsc.liu.se/~boein/ifip/kyoto/reid.txt

Evaluation

- Expressivity
  - On its own, it isn't as expressive as modern imperative language (e.g. no shortcut operators)
  - There are some efforts to improve expressiveness, albeit through modules[4]
- Exception handling: No exception handling in pure Fortran 95, but an attempt was made although never standardized[5]

[4] http://www.southampton.ac.uk/~fangohr/randomnotes/iop_cpg_newsletter/2007_1

[5] see https://www.nsc.liu.se/~boein/ifip/kyoto/reid.txt

Evaluation

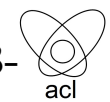
- Restricted aliasing: Uses the `in`, `out`, and `inout` keywords to control aliasing in parameter passing

---

http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

Evaluation

- Restricted aliasing: Uses the `in`, `out`, and `inout` keywords to control aliasing in parameter passing
- Efficiency

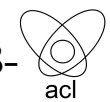


---

http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

Evaluation

- Restricted aliasing: Uses the `in`, `out`, and `inout` keywords to control aliasing in parameter passing
- Efficiency

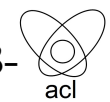


---

http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

Evaluation

- Restricted aliasing: Uses the `in`, `out`, and `inout` keywords to control aliasing in parameter passing
- Efficiency
  - Static type checking



http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

Evaluation

- Restricted aliasing: Uses the `in`, `out`, and `inout` keywords to control aliasing in parameter passing
- Efficiency
  - Static type checking
  - Portable



http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

Evaluation

- Restricted aliasing: Uses the `in`, `out`, and `inout` keywords to control aliasing in parameter passing
- Efficiency
  - Static type checking
  - Portable
  - No recursion

---

http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

Evaluation

- Restricted aliasing: Uses the `in`, `out`, and `inout` keywords to control aliasing in parameter passing
- Efficiency
  - Static type checking
  - Portable
  - No recursion
  - No dynamic memory allocation

---

http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

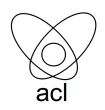
Evaluation

- Restricted aliasing: Uses the `in`, `out`, and `inout` keywords to control aliasing in parameter passing
- Efficiency
  - Static type checking
  - Portable
  - No recursion
  - No dynamic memory allocation
  - No array bounds checking

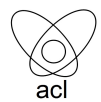
---

http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

Imperative Programming Languages



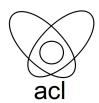
Background



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/ada/ada.html

Background

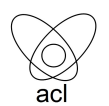
- The Ada language is the result of the US Army, Navy, and Air Force proposed to develop a high-level language for embedded systems



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/ada/ada.html

Background

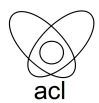
- The Ada language is the result of the US Army, Navy, and Air Force proposed to develop a high-level language for embedded systems
  - Noted as the most extensive and most expensive language design effort ever undertaken



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/ada/ada.html

Background

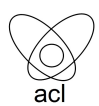
- The Ada language is the result of the US Army, Navy, and Air Force proposed to develop a high-level language for embedded systems
  - Noted as the most extensive and most expensive language design effort ever undertaken
- A complete language design specification for Ada was created in 1977



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/ada/ada.html

Background

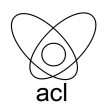
- The Ada language is the result of the US Army, Navy, and Air Force proposed to develop a high-level language for embedded systems
  - Noted as the most extensive and most expensive language design effort ever undertaken
- A complete language design specification for Ada was created in 1977
  - In May of 1979, the Cii Honeywell/Bull language design was chosen as the (preliminary) standard for Ada



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/ada/ada.html

Background

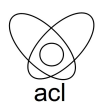
- The Ada language is the result of the US Army, Navy, and Air Force proposed to develop a high-level language for embedded systems
  - Noted as the most extensive and most expensive language design effort ever undertaken
- A complete language design specification for Ada was created in 1977
  - In May of 1979, the Cii Honeywell/Bull language design was chosen as the (preliminary) standard for Ada
  - In November 1979, over 500 language reports were received from 15 different countries



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/ada/ada.html

Background

- The Ada language is the result of the US Army, Navy, and Air Force proposed to develop a high-level language for embedded systems
  - Noted as the most extensive and most expensive language design effort ever undertaken
- A complete language design specification for Ada was created in 1977
  - In May of 1979, the Cii Honeywell/Bull language design was chosen as the (preliminary) standard for Ada
  - In November 1979, over 500 language reports were received from 15 different countries
  - Based on these reports, a revised language design was published in February of 1980



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/ada/ada.html

Background

- The Ada language is the result of the US Army, Navy, and Air Force proposed to develop a high-level language for embedded systems
  - Noted as the most extensive and most expensive language design effort ever undertaken
- A complete language design specification for Ada was created in 1977
  - In May of 1979, the Cii Honeywell/Bull language design was chosen as the (preliminary) standard for Ada
  - In November 1979, over 500 language reports were received from 15 different countries
  - Based on these reports, a revised language design was published in February of 1980
  - The final official version of the language was settled upon a few years after

http://groups.engin.umd.umich.edu/CIS/course.des/cis400/ada/ada.html

Evaluation

- Data types[6]

acl

[6] http://www.adaic.org/learn/materials/intro/part2

Evaluation

- Data types[6]
  - Integers, floating and fixed point real numbers, complex numbers, characters, enumerations as primitive types

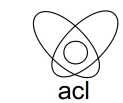
[6] http://www.adaic.org/learn/materials/intro/part2

Evaluation

- Data types[6]
  - Integers, floating and fixed point real numbers, complex numbers, characters, enumerations as primitive types
  - Support for multidimensional arrays

[6] http://www.adaic.org/learn/materials/intro/part2



Evaluation

- Data types[6]
  - Integers, floating and fixed point real numbers, complex numbers, characters, enumerations as primitive types
  - Support for multidimensional arrays
  - Allows for derived structures using the `record` type declaration

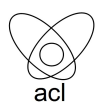
[6] http://www.adaic.org/learn/materials/intro/part2

Evaluation

- Syntax design (Ada 2005[7])
  - Number of production rules: 394
  - Number of top alternatives: 621
  - Number of symbols: 2,973
  - Vocabulary
    - Nonterminal symbols: 405
    - Terminal symbols: 101

[7] http://slebok.github.io/zoo/ada/ada2005/txl/lehyaric-cordy/extracted/index.html

[8] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm

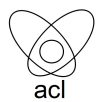


Evaluation

- Syntax design (Ada 2005[7])
  - Number of production rules: 394
  - Number of top alternatives: 621
  - Number of symbols: 2,973
  - Vocabulary
    - Nonterminal symbols: 405
    - Terminal symbols: 101
- Abstraction[8]

[7] http://slebok.github.io/zoo/ada/ada2005/txl/lehyaric-cordy/extracted/index.html

[8] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm

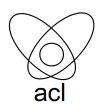


Evaluation

- Syntax design (Ada 2005[7])
  - Number of production rules: 394
  - Number of top alternatives: 621
  - Number of symbols: 2,973
  - Vocabulary
    - Nonterminal symbols: 405
    - Terminal symbols: 101
- Abstraction[8]
  - Access grants to variables and procedures can be declared via `public` and `private`

[7] http://slebok.github.io/zoo/ada/ada2005/txl/lehyaric-cordy/extracted/index.html

[8] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm

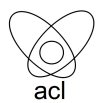


Evaluation

- Syntax design (Ada 2005[7])
  - Number of production rules: 394
  - Number of top alternatives: 621
  - Number of symbols: 2,973
  - Vocabulary
    - Nonterminal symbols: 405
    - Terminal symbols: 101
- Abstraction[8]
  - Access grants to variables and procedures can be declared via `public` and `private`
  - `package` declaration to allow blocks of parameters, variables, and subprograms to be both shared and re-used.

[7] http://slebok.github.io/zoo/ada/ada2005/txl/lehyaric-cordy/extracted/index.html

[8] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm

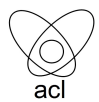


Evaluation

- Syntax design (Ada 2005[7])
  - Number of production rules: 394
  - Number of top alternatives: 621
  - Number of symbols: 2,973
  - Vocabulary
    - Nonterminal symbols: 405
    - Terminal symbols: 101
- Abstraction[8]
  - Access grants to variables and procedures can be declared via `public` and `private`
  - `package` declaration to allow blocks of parameters, variables, and subprograms to be both shared and re-used.
- Expressivity: At par with (most) imperative PLs

[7] http://slebok.github.io/zoo/ada/ada2005/txl/lehyaric-cordy/extracted/index.html

[8] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm

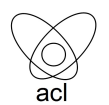


Evaluation

- Exception handling: Has a `exception-when` statement to perform in-program exception handling

[9] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm
[10] http://www.adaic.org/community/compilers-conformity





Evaluation

- Exception handling: Has a `exception-when` statement to perform in-program exception handling
- Restricted aliasing: Uses the `IN` and `OUT` keywords to control aliasing in parameter passing

[9] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm
[10] http://www.adaic.org/community/compilers-conformity

Evaluation

- Exception handling: Has a `exception-when` statement to perform in-program exception handling
- Restricted aliasing: Uses the `IN` and `OUT` keywords to control aliasing in parameter passing
- Efficiency:

[9] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm
[10] http://www.adaic.org/community/compilers-conformity

Evaluation

- Exception handling: Has a `exception-when` statement to perform in-program exception handling
- Restricted aliasing: Uses the `IN` and `OUT` keywords to control aliasing in parameter passing
- Efficiency:
  - Known to be a language with strong (static) type-checking (The '95 version, at least) [9]

[9] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm
[10] http://www.adaic.org/community/compilers-conformity

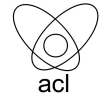


Evaluation

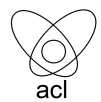
- Exception handling: Has a `exception-when` statement to perform in-program exception handling
- Restricted aliasing: Uses the `IN` and `OUT` keywords to control aliasing in parameter passing
- Efficiency:
  - Known to be a language with strong (static) type-checking (The '95 version, at least) [9]
  - Compilers and dialects must conform to the standards set by the Ada Conformity Assessment Authority (ACAA) to "to be dependable, reusable, portable, maintainable, and legible", and in the case of Ada 95 and Ada 2005, "highly reliable"[10]

[9] http://www.infres.enst.fr/~pautet/Ada95/chap06.htm
[10] http://www.adaic.org/community/compilers-conformity

Imperative Programming Languages



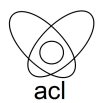
Background



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/pascal/pascal.html

Background

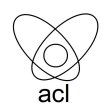
- Pascal programming language was originally developed by Niklaus Wirth, the original published definition of which appeared in 1971



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/pascal/pascal.html

Background

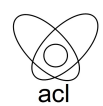
- Pascal programming language was originally developed by Niklaus Wirth, the original published definition of which appeared in 1971
- Pascal aimed to provide features that were lacking in other languages of the time



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/pascal/pascal.html

Background

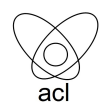
- Pascal programming language was originally developed by Niklaus Wirth, the original published definition of which appeared in 1971
- Pascal aimed to provide features that were lacking in other languages of the time
  - Efficiency in implementation and execution



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/pascal/pascal.html

Background

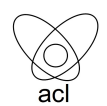
- Pascal programming language was originally developed by Niklaus Wirth, the original published definition of which appeared in 1971
- Pascal aimed to provide features that were lacking in other languages of the time
  - Efficiency in implementation and execution
  - Allows for development of well-structured and well-organized programs



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/pascal/pascal.html

Background

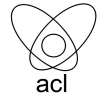
- Pascal programming language was originally developed by Niklaus Wirth, the original published definition of which appeared in 1971
- Pascal aimed to provide features that were lacking in other languages of the time
  - Efficiency in implementation and execution
  - Allows for development of well-structured and well-organized programs
  - Served as a vehicle for teaching computer programming



http://groups.engin.umd.umich.edu/CIS/course.des/cis400/pascal/pascal.html

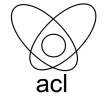
Evaluation

- Data types



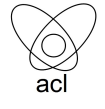
Evaluation

- Data types
  - Integers, real single and double precision floating point numbers, complex numbers, characters, logical as primitive types



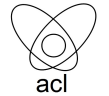
Evaluation

- Data types
  - Integers, real single and double precision floating point numbers, complex numbers, characters, logical as primitive types
  - Support for multidimensional arrays, and in later versions, a string type as a "special" kind of array



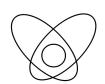
Evaluation

- Data types
  - Integers, real single and double precision floating point numbers, complex numbers, characters, logical as primitive types
  - Support for multidimensional arrays, and in later versions, a string type as a "special" kind of array
  - Allows for derived structures using the `Record` type declaration



Evaluation

- Syntax design (Delphi 2006[11])



[11] http://slebok.github.io/zoo/pascal/delphi/delphi2006/cangas/extracted/index.html

Evaluation

- Syntax design (Delphi 2006[11])
  - Number of production rules: 186
  - Number of top alternatives: 387
  - Number of symbols: 1,444
  - Vocabulary
    - Nonterminal symbols: 197
    - Terminal symbols: 184

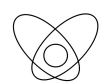
[11] http://slebok.github.io/zoo/pascal/delphi/delphi2006/cangas/extracted/index.html

Evaluation

- Syntax design (Delphi 2006[11])
  - Number of production rules: 186
  - Number of top alternatives: 387
  - Number of symbols: 1,444
  - Vocabulary
    - Nonterminal symbols: 197
    - Terminal symbols: 184
  - Uniformity issues
    - Constants cannot be declared with values given by expressions, even though expressions are accepted in all other contexts when a value is needed
    - `if` and `while` need `begin-end` to take in multiple statements, but repeat loops do not

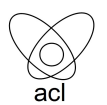
[11] http://slebok.github.io/zoo/pascal/delphi/delphi2006/cangas/extracted/index.html





Evaluation

- Abstraction



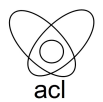
[12] http://www.freepascal.org/docs-html/ref/refse102.html

Evaluation

- Abstraction
  - In Delphi and Object Pascal, access grants to variables and procedures can be declared via `public`, `private`, and `protected`

[12] http://www.freepascal.org/docs-html/ref/refse102.html

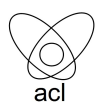


Evaluation

- Abstraction
  - In Delphi and Object Pascal, access grants to variables and procedures can be declared via `public`, `private`, and `protected`
  - `program` declaration to allow blocks of parameters, variables, and subprograms to be both shared and re-used.

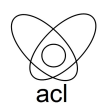
[12] http://www.freepascal.org/docs-html/ref/refse102.html



Evaluation

- Abstraction
  - In Delphi and Object Pascal, access grants to variables and procedures can be declared via `public`, `private`, and `protected`
  - `program` declaration to allow blocks of parameters, variables, and subprograms to be both shared and re-used.
- Expressivity: At par with (most) imperative PLs

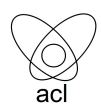
[12] http://www.freepascal.org/docs-html/ref/refse102.html



Evaluation

- Abstraction
  - In Delphi and Object Pascal, access grants to variables and procedures can be declared via `public`, `private`, and `protected`
  - `program` declaration to allow blocks of parameters, variables, and subprograms to be both shared and re-used.
- Expressivity: At par with (most) imperative PLs
- Exception handling: Has a `try-except` statement to perform in-program exception handling in the Delphi version[12]

[12] http://www.freepascal.org/docs-html/ref/refse102.html

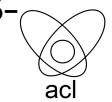


Evaluation

- Restricted aliasing[13]

[13] http://swinbrain.ict.swin.edu.au/wiki/Pass_by_Value_vs._Pass_by_Reference
[14] http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf
[15] http://www.cs.princeton.edu/courses/archive/spr96/cs441/notes/l16.html





Evaluation

- Restricted aliasing[13]
  - Pass-by-value parameter passing is done by default

acl

[13] http://swinbrain.ict.swin.edu.au/wiki/Pass_by_Value_vs._Pass_by_Reference

[14] http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

[15] http://www.cs.princeton.edu/courses/archive/spr96/cs441/notes/l16.html

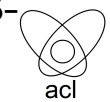
Evaluation

- Restricted aliasing[13]
  - Pass-by-value parameter passing is done by default
  - Pass-by-reference can be done by adding the keyword `var` prior to the name of the formal parameter

[13] http://swinbrain.ict.swin.edu.au/wiki/Pass_by_Value_vs._Pass_by_Reference

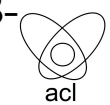
[14] http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

[15] http://www.cs.princeton.edu/courses/archive/spr96/cs441/notes/l16.html



Evaluation

- Restricted aliasing[13]
  - Pass-by-value parameter passing is done by default
  - Pass-by-reference can be done by adding the keyword `var` prior to the name of the formal parameter
- Efficiency[14]



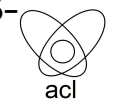
[13] http://swinbrain.ict.swin.edu.au/wiki/Pass_by_Value_vs._Pass_by_Reference

[14] http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

[15] http://www.cs.princeton.edu/courses/archive/spr96/cs441/notes/l16.html

Evaluation

- Restricted aliasing[13]
  - Pass-by-value parameter passing is done by default
  - Pass-by-reference can be done by adding the keyword `var` prior to the name of the formal parameter
- Efficiency[14]
  - Pascal is known as a statically-typed language[15]



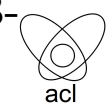
[13] http://swinbrain.ict.swin.edu.au/wiki/Pass_by_Value_vs._Pass_by_Reference

[14] http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

[15] http://www.cs.princeton.edu/courses/archive/spr96/cs441/notes/l16.html

Evaluation

- Restricted aliasing[13]
  - Pass-by-value parameter passing is done by default
  - Pass-by-reference can be done by adding the keyword `var` prior to the name of the formal parameter
- Efficiency[14]
  - Pascal is known as a statically-typed language[15]
  - No forward references



[13] http://swinbrain.ict.swin.edu.au/wiki/Pass_by_Value_vs._Pass_by_Reference

[14] http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

[15] http://www.cs.princeton.edu/courses/archive/spr96/cs441/notes/l16.html

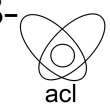
Evaluation

- Restricted aliasing[13]
  - Pass-by-value parameter passing is done by default
  - Pass-by-reference can be done by adding the keyword `var` prior to the name of the formal parameter
- Efficiency[14]
  - Pascal is known as a statically-typed language[15]
  - No forward references
  - One-pass compiler

[13] http://swinbrain.ict.swin.edu.au/wiki/Pass_by_Value_vs._Pass_by_Reference

[14] http://cs.txstate.edu/~ch04/webtest/teaching/courses/5318/lectures/slides2/s3-criteria.pdf

[15] http://www.cs.princeton.edu/courses/archive/spr96/cs441/notes/l16.html



END OF SESSION 4