CS3203: Software Engineering Project

Basic SPA Requirements

LN – BASIC_SPA

This lecture covers

(i) What is Static Program Analyzer(SPA) involving source language SIMPLE, PQL and how it works.

(ii) Source Language SIMPLE

(iii) Program Query Language (PQL)

BasicSPA:

(iv) **Basic** Program Design abstractions for SIMPLE

(v) Rules for writing **Basic** Program Queries in PQL

SPA - Static Program Analyzer

- SPA - a tool to address a problem from the domain called **Program Analysis**
- Program analysis : What do programmers need to know to understand programs
  - I need to find code that implements salary computation rules!
  - Where is variable ‘x’ modified? Where is it used?
  - I need to find all statements with sub-expression x*y+z
  - Which statements affect value of ‘x’ at statement #120?
  - Which statements can be affected if I modify statement #20?
- Top-40 Static Program analysis Tools[Nov 13 2020]

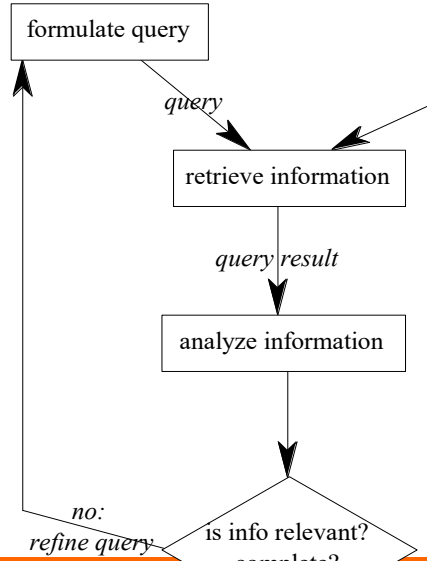
SPA - Static Program Analyzer

looking for missing information to implement request for change ...

sources of program information

application domain	program requirements	globalprogram design	detailed program design	code

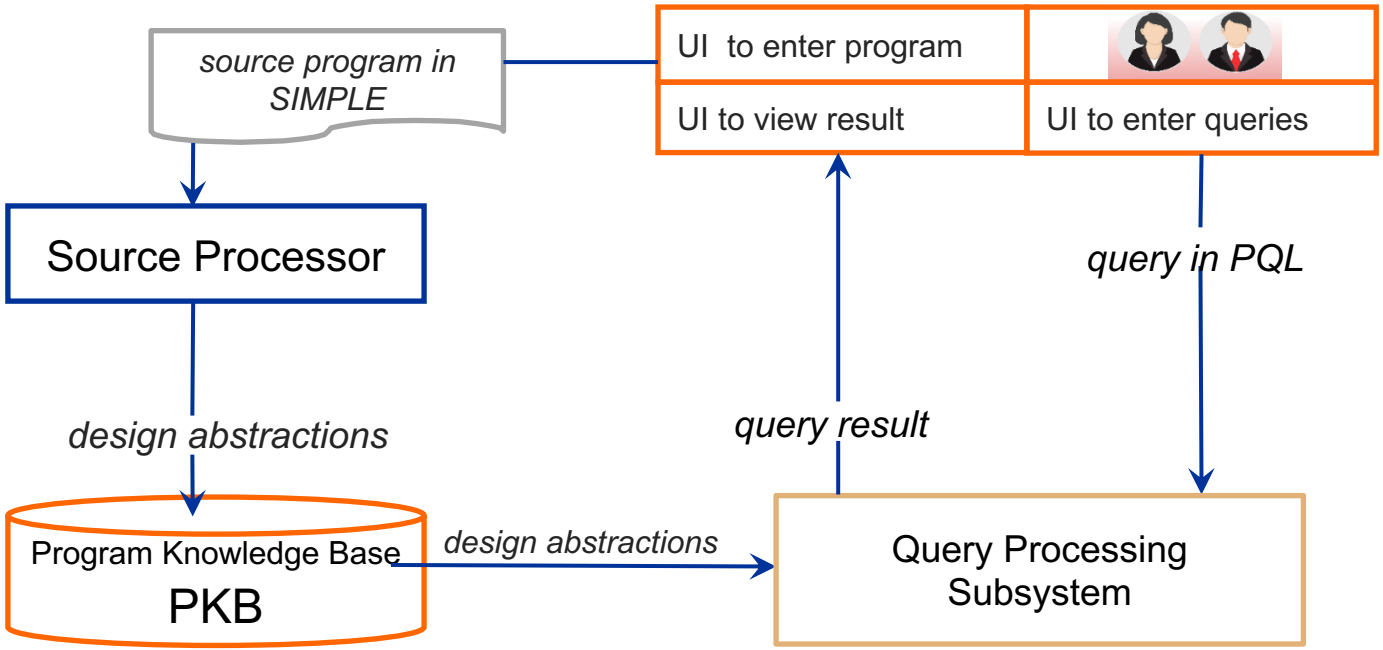
program design model



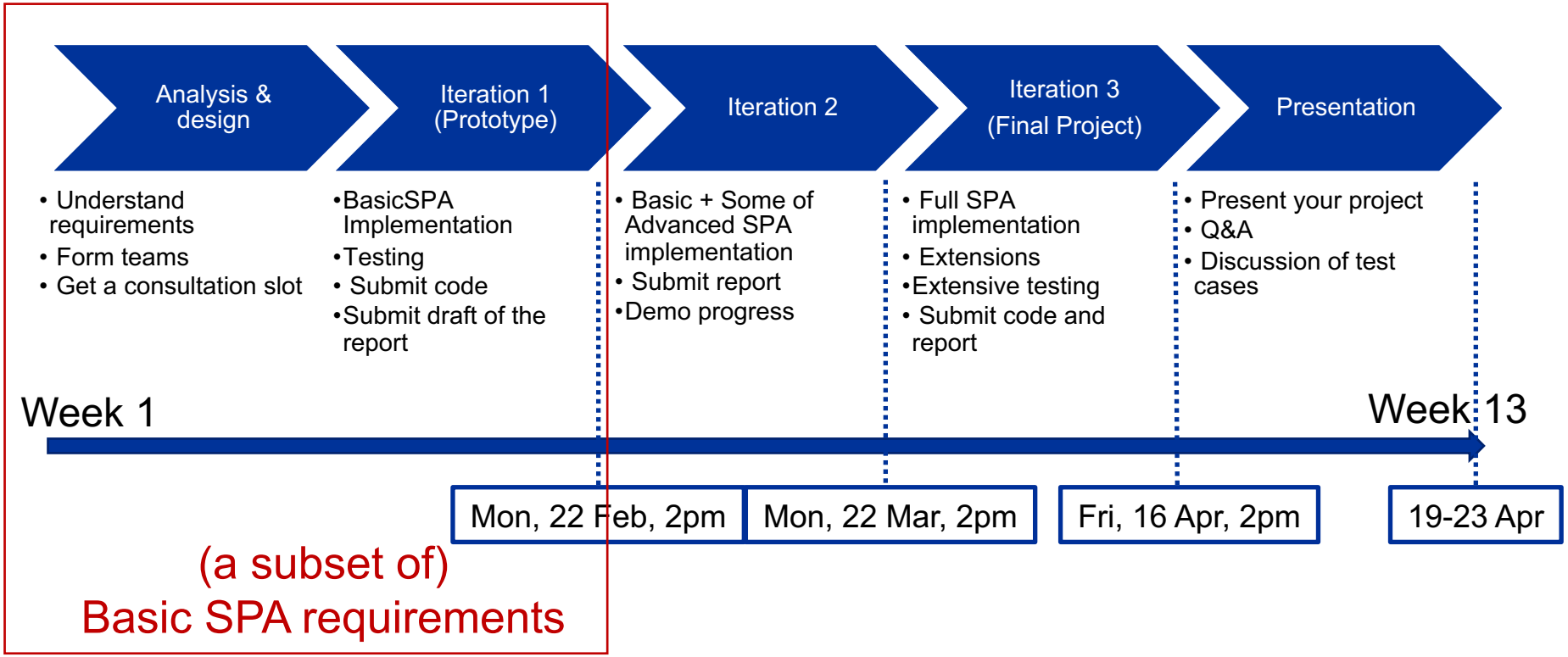
Many program queries can be answered by a **Static Program Analyzer (SPA).**

SPA - Static Program Analyzer

- Source language SIMPLE
- Program Query Language PQL



SPA Iterations



Basic SPA requirements

- Source language SIMPLE
- Types of information stored in PKB
- Program Query Language (PQL) & Common program queries

Source Language SIMPLE

Why SIMPLE?

- Simplified programming language to allow for easy static analysis
  - SIMPLE contains the basic constructs of a programming language for writing meaningful programs
  - Allows students to complete the project in one semester
- What if ….we want to implement SPA for (subset) of C?
  - Extended syntax
  - Variable types
  - Arrays and pointers
  - Scoping

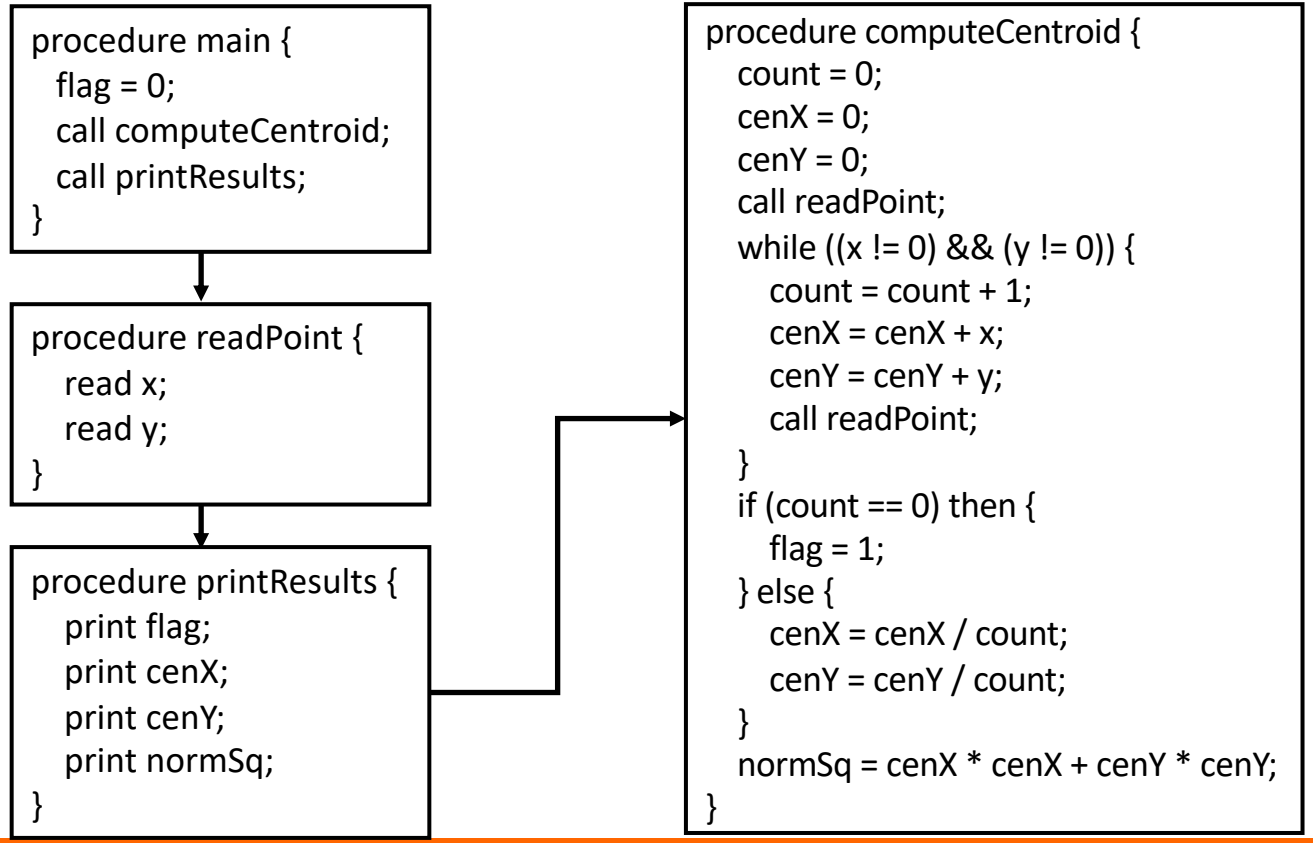
Sample Programs in SIMPLE

```
procedure computeAverage {
   read num1;
   read num2;
   read num3;

   sum = num1 + num2 + num3;
   ave = sum / 3;

   print ave;
}
```

```
procedure printAscending {
   read num1;
   read num2;
   noSwap = 0;

if (num1 > num2) then {
   temp = num1;
   num1 = num2;
   num2 = temp;
} else {
   noSwap = 1;
}

   print num1;
   print num2;
   print noSwap;
}
```

```
procedure sumDigits {
   read number;
   sum = 0;

   while (number > 0) {
      digit = number % 10;
      sum = sum + digit;
      number = number / 10;
   }

   print sum;
}
```

Sample Programs in SIMPLE



SIMPLE language rules

- A program consists of one or more procedures
- Program execution starts by calling the first procedure
- Procedures: no parameters, no nesting, no recursion
- Variables: unique names, global scope, integer type, no declarations
- Conditions: Boolean expressions for if and while statements.
- No arrays, no pointers

SIMPLE language rules

- Program statements:
  - Procedure call, e.g., call p;
  - Assignment, e.g., x = 2; x = a + 2 * b;
  - While statement, e.g., while (i > 0) { … }
  - If statement, e.g.., if (i > 0) then { … } else { … }
  - Read input, e.g., read x;
    - » Note that this is NOT to read the value **from** X. Instead, it is about reading a value (possibly from keyboard) **into** x.
  - Print output, e.g., print x;

Concrete vs Abstract Syntax of a language

- Every programming language has a **concrete syntax**.
  - Defines what the programs look like to the programmer
  - **assign** : var_name ‘=’ expr ‘;’
  - **if** : ‘if’ ‘(’ cond_expr ‘)’ ‘then’ ‘{‘ stmtLst ‘}’ ‘else’ ‘{‘ stmtLst ‘}’

- Every implementation of a programming language uses an **abstract syntax**
  - Defines the way the programs look like to the analyser or compiler
  - **assign** : variable expr
  - **if** : cond_expr stmtLst stmtLst

Concrete Syntax Grammar for SIMPLE

- Lexical tokens:
  - **LETTER** : A-Z | a-z -- capital or small letter
  - **DIGIT** : 0-9
  - **NAME** : LETTER (LETTER | DIGIT)*
  - **INTEGER** : DIGIT+

- Grammar rules:
  - **program** : procedure+
  - **procedure** : ‘procedure’ proc_name ‘{‘ stmtLst ‘}’
  - **stmtLst** : stmt+
  - **stmt** : read | print | call | while | if | assign
  - **read** : ‘read’ var_name;
  - **print** : ‘print’ var_name;
  - **call** : ‘call’ proc_name ‘;’
  - **while** : ‘while’ ‘(’ cond_expr ‘)’ ‘{‘ stmtLst ‘}’
  - **if** : ‘if’ ‘(’ cond_expr ‘)’ ‘then’ ‘{‘ stmtLst ‘}’ ‘else’ ‘{‘ stmtLst ‘}’
  - **assign** : var_name ‘=’ expr ‘;’
  - **cond_expr** : rel_expr | ‘!’ ‘(’ cond_expr ‘)’ | ‘(’ cond_expr ‘)’ ‘&&’ ‘(’ cond_expr ‘)’ | ‘(’ cond_expr ‘)’ ‘||’ ‘(’ cond_expr ‘)’
  - **rel_expr** : rel_factor ‘>’ rel_factor | rel_factor ‘>=’ rel_factor | rel_factor ‘<’ rel_factor | rel_factor ‘<=’ rel_factor | rel_factor ‘==’ rel_factor | rel_factor ‘!=’ rel_factor
  - **rel_factor** : var_name | const_value | expr
  - **expr** : expr ‘+’ term | expr ‘-’ term | term
  - **term** : term ‘*’ factor | term ‘/’ factor | term ‘%’ factor | factor
  - **factor** : var_name | const_value | ‘(’ expr ‘)’
  - **var_name**, **proc_name**: NAME
  - **const_value** : INTEGER

Abstract Syntax Grammar for SIMPLE

- Lexical tokens:
  - **LETTER** : A-Z | a-z
  - **DIGIT** : 0-9
  - **NAME** : LETTER (LETTER | DIGIT)*
  - **INTEGER** : DIGIT+

- Grammar rules:
  - **program** : procedure+
  - **procedure** : stmtLst
  - **stmtLst** : stmt+
  - **stmt** : read | print | call | while | if | assign
  - **read**, **print** : variable
  - **while**: cond_expr stmtLst
  - **if** : cond_expr stmtLst stmtLst
  - **assign** : variable expr
  - **cond_expr** : rel_expr | not | and | or
  - **not**: cond_expr
  - **and**, **or** : cond_expr cond_expr
  - **rel_expr** : gt | gte | lt | lte | eq | neq
  - **gt**, **gte**, **lt**, **lte**, **eq**, **neq** : rel_factor rel_factor
  - **rel_factor** : variable | constant | expr
  - **expr** : plus | minus | times | div | mod | ref
  - **plus**, **minus**, **times**, **div**, **mod** : expr expr
  - **ref** : variable | constant

- Attributes and attribute value types:
  - **procedure.procName**, **call.procName**, **variable.varName**, **read.varName**, **print.varName** : NAME
  - **constant.value** : INTEGER
  - **stmt.stmt#, read.stmt#, print.stmt#, call.stmt#, while.stmt#, if.stmt#, assign.stmt#**: INTEGER

Meta Symbols reference

Meta symbols:

- a* - repetition 0 or more times of a
- a+ - repetition 1 or more times of a
- a|b - either a or b may appear

Lexical tokens:

- LETTER : A-Z | a-z
  - capital or small letter
- DIGIT : 0-9
- NAME: LETTER (LETTER | DIGIT)*
  - procedure, variable and attribute names are strings of letters, digits, starting with a letter
- INTEGER : DIGIT+
  - constants are sequences of digits
  - if more than one digit, the first digit cannot be 0

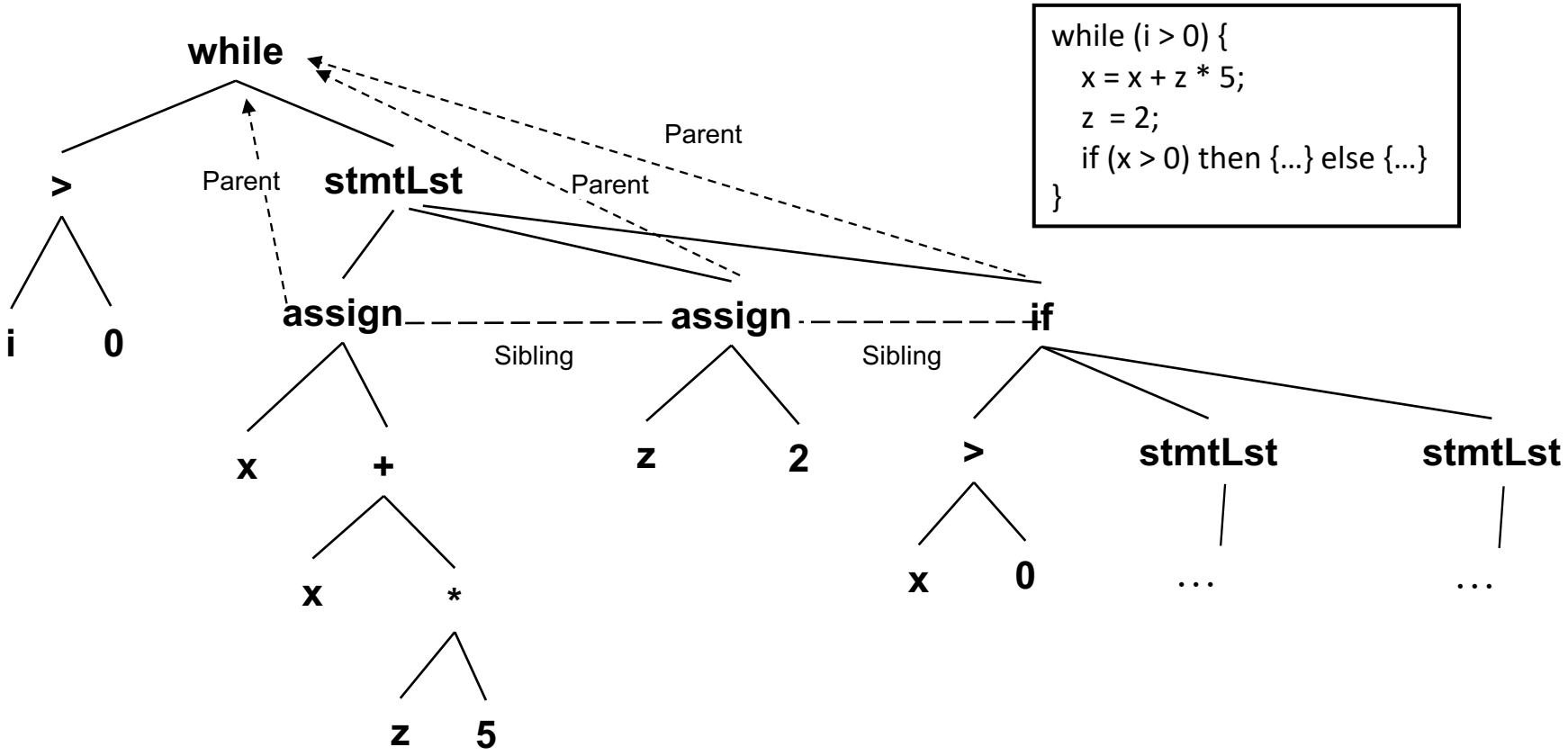
Representing a SIMPLE Source program

For example, an Abstract Syntax tree(AST) could be used as an abstract representation of the SIMPLE source program.

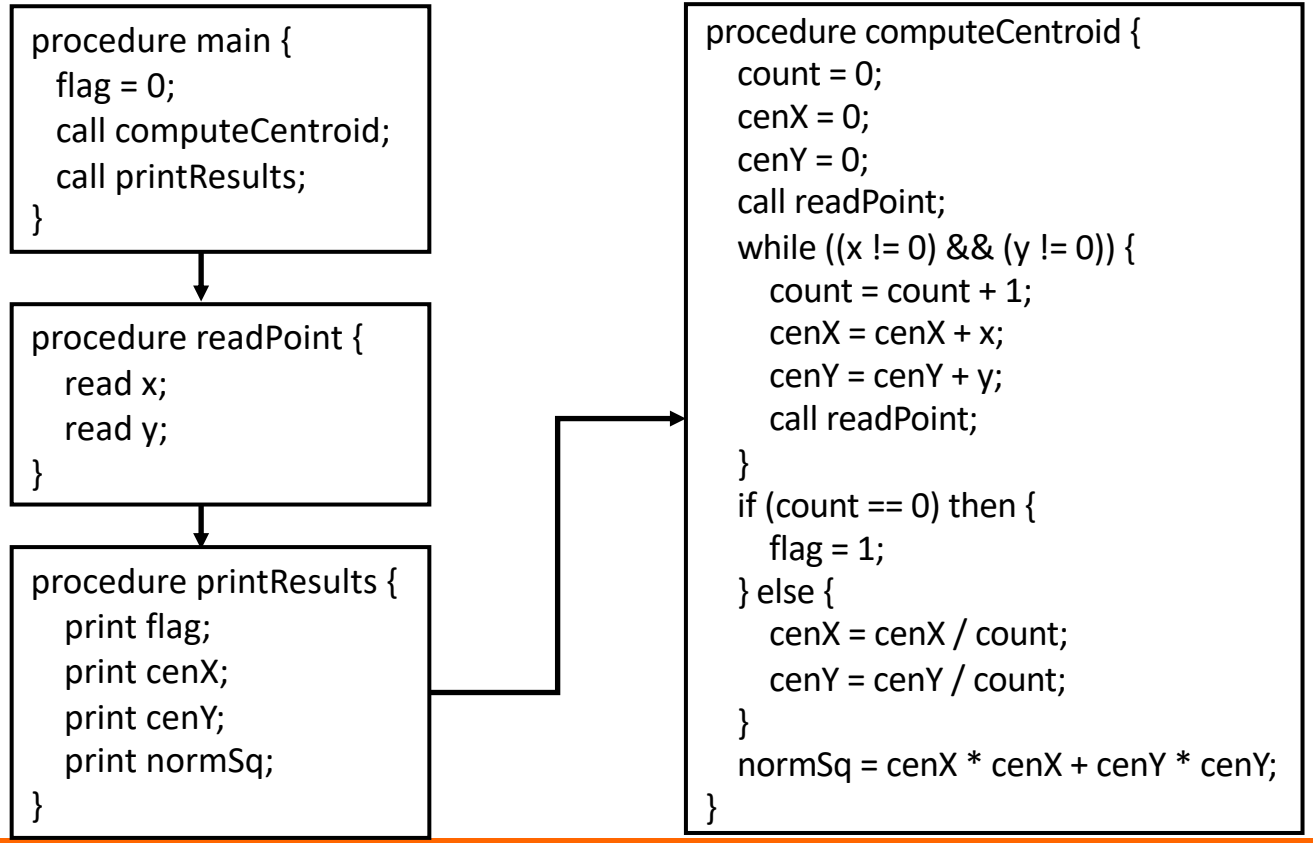
- See examples in next few slides

Could we use some other structure for the representation of the SIMPLE source program?

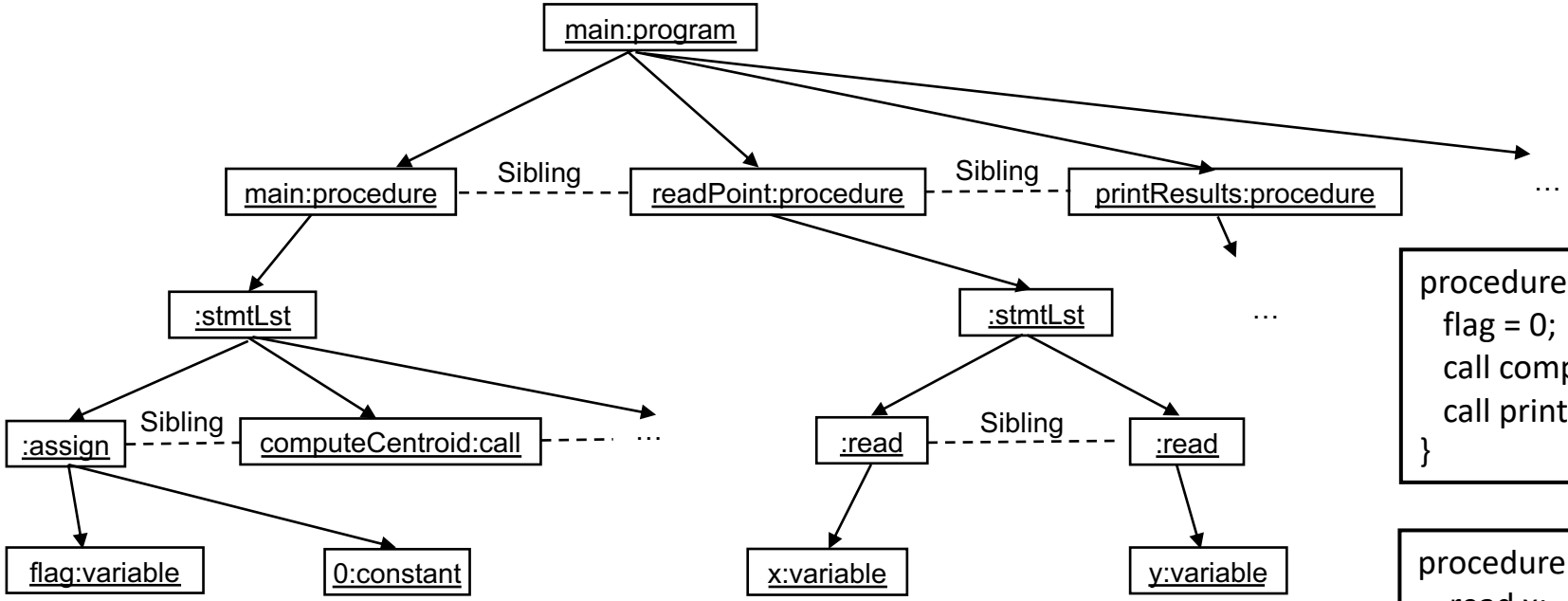
Example: while loop as an **Abstract Syntax Tree (AST)**



Example: SIMPLE program



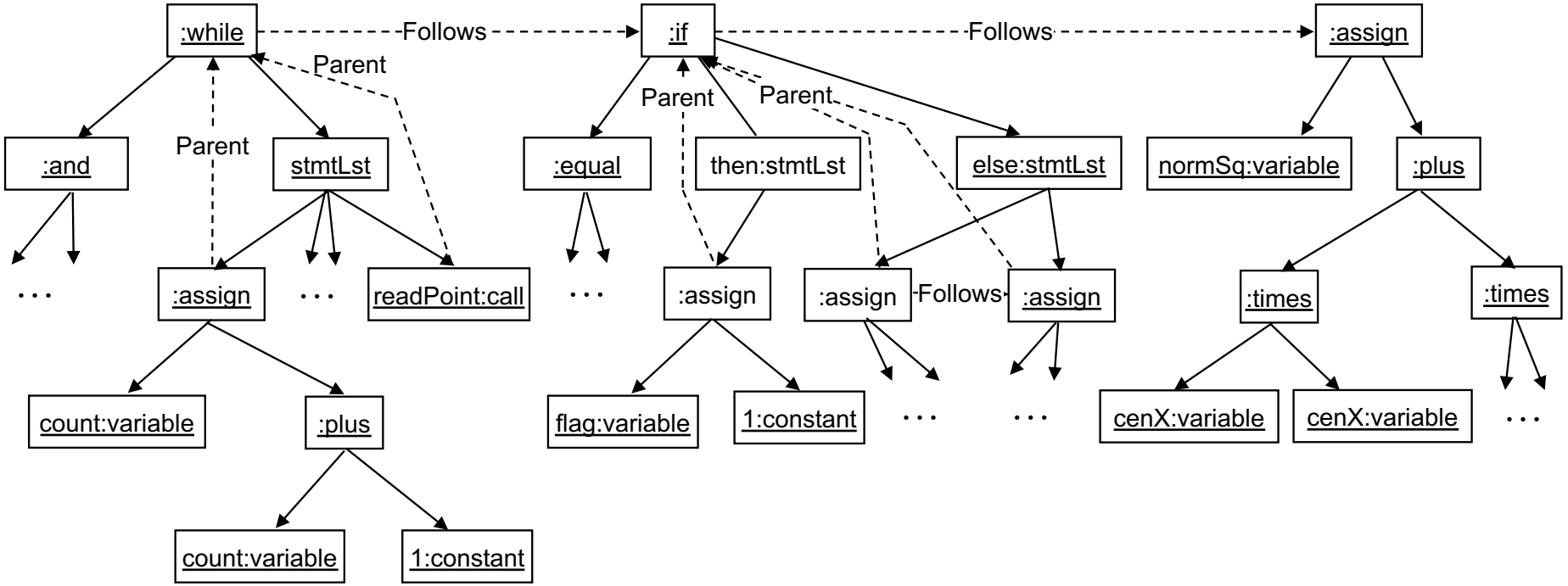
Abstract Syntax Tree (AST) for procedures main and readPoint




```
procedure main {
  flag = 0;
  call computeCentroid;
  call printResults;
}
```

```
procedure readPoint {
   read x;
   read y;
}
```

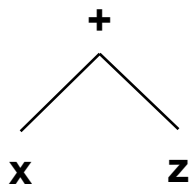
Partial AST for procedure computeCentroid



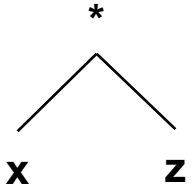
Note: All sibling links have been omitted for clarity.

AST for Expressions

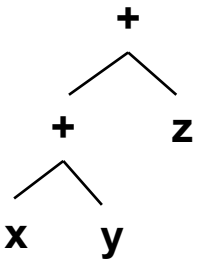
x + z



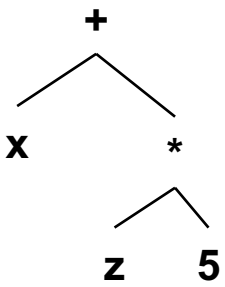
x * z



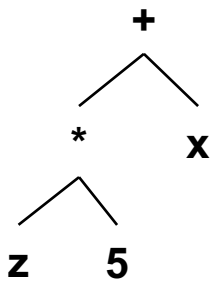
x + y + z



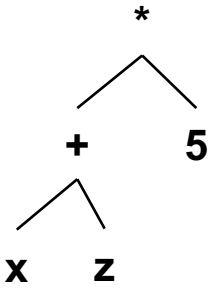
x + z * 5



z * 5 + x



(x + z) * 5



Types of information stored in PKB

Program Design Entities

- procedure
- stmtLst
- stmt
- read (statement)
- print (statement)
- assign (statement)
- call (statement)
- while (statement)
- if (statement)
- variable
- constant

Examples

Queries	What to store
What are the **variables** in the program?	Variable information
Which **statements** are assignment statements?	Statement information
Which **statements** appear after another **statement**?	**Relationships** between statements
Which **variables** are modified in a **statement**?	**Relationships** between statements and variables
Which **variables** are used in a **procedure**?	**Relationships** between procedures and variables

Program Design Abstractions

- Program design abstractions are relationships (e.g., Follows, Modifies) between program design entities (e.g., statements, variables)

- Example: A typical query
  - Does statement 6 modify variable "count"?
- Solution: Store information about Modifies relationship

Example: Modifies

- Example: A typical query
  - Does statement 6 modify variable "count"?
  - The answer depends on whether **Modifies (6, "count")** is true.

```
procedure computeCentroid {
1.      count = 0;
2.      cenX = 0;
3.      cenY = 0;
4.      call readPoint;
5.      while ((x != 0) && (y != 0)) {
6.         count = count + 1;
7.         cenX = cenX + x;
8.         cenY = cenY + y;
9.         call readPoint;
        }
10.     if (count == 0) then {
11.        flag = 1;
        } else {
12.        cenX = cenX / count;
13.        cenY = cenY / count;
        }
14.     normSq = cenX * cenX + cenY * cenY;
}
```

BasicSPA Design Abstractions

Design Abstraction	Relationship between
Follows/ Follows*	Statements
Parent/ Parent*	Statements
Uses	Statement/Procedure and Variable
Modifies	Statement/Procedure and Variable

Follows and Follows*

For any statements s1 and s2:

- **Follows (s1, s2)** holds if they are at the same nesting level, in the same statement list (stmtLst), and s2 appears immediately after s1
- **Follows* (s1, s2)** holds if
  - Follows (s1, s2) or
  - Follows (s1, s) and Follows* (s, s2) for some statement s

Examples of Follows and Follows*

Which relationships hold?

A. Follows (1, 2)

B. Follows (4, 5)

C. Follows (5, 6)

D. Follows (9, 10)

E. Follows (5, 10)

F. Follows (11, 12)

G. Follows* (3,10)

H. Follows* (12, 14)

I. Follows* (1, 14)

```
procedure computeCentroid {
1.      count = 0;
2.      cenX = 0;
3.      cenY = 0;
4.      call readPoint;
5.      while ((x != 0) && (y != 0)) {
6.         count = count + 1;
7.         cenX = cenX + x;
8.         cenY = cenY + y;
9.         call readPoint;
        }
10.     if (count == 0) then {
11.        flag = 1;
        } else {
12.        cenX = cenX / count;
13.        cenY = cenY / count;
        }
14.     normSq = cenX * cenX + cenY * cenY;
}
```

Parent and Parent*

For any statements s1 and s2:

- **Parent (s1, s2)** holds if s2 is directly nested in s1
- **Parent* (s1, s2)** holds if
  - Parent (s1, s2) or
  - Parent (s1, s) and Parent* (s, s2) for some statement s

Examples of Parent and Parent*

Which relationships hold?

A. Parent (2, 3)

B. Parent (4, 7)

C. Parent (5, 7)

D. Parent (9, 5)

E. Parent* (5, 7)

F. Parent* (10, 13)

G. Parent* (10, 14)

```
procedure computeCentroid {
1.      count = 0;
2.      cenX = 0;
3.      cenY = 0;
4.      call readPoint;
5.      while ((x != 0) && (y != 0)) {
6.         count = count + 1;
7.         cenX = cenX + x;
8.         cenY = cenY + y;
9.         call readPoint;
        }
10.     if (count == 0) then {
11.        flag = 1;
        } else {
12.        cenX = cenX / count;
13.        cenY = cenY / count;
        }
14.     normSq = cenX * cenX + cenY * cenY;
}
```

Uses

Design entities	Description
Assignment a Variable v	Uses (a, v) holds if variable **v** appears on the right hand side of **a**
Print statement pn Variable v	Uses (pn, v) holds if variable **v** appears in **pn**.
Container statement s (i.e. "if" or "while") Variable v	Uses (s, v) holds if **v** appears in the condition of **s**, or there is a statement **s1** in the container such that Uses(s1, v) holds
Procedure p Variable v	Uses (p, v) holds if there is a statement **s** in **p** or in a procedure called (directly or indirectly) from **p** such that Uses (s, v) holds.
Procedure call c (i.e. "call p") Variable v	Uses (c, v) is defined in the same way as Uses (p, v).

Examples of Uses

Which relationships hold?

A. Uses (3, "count")

B. Uses (7, "x")

C. Uses (9, "y")

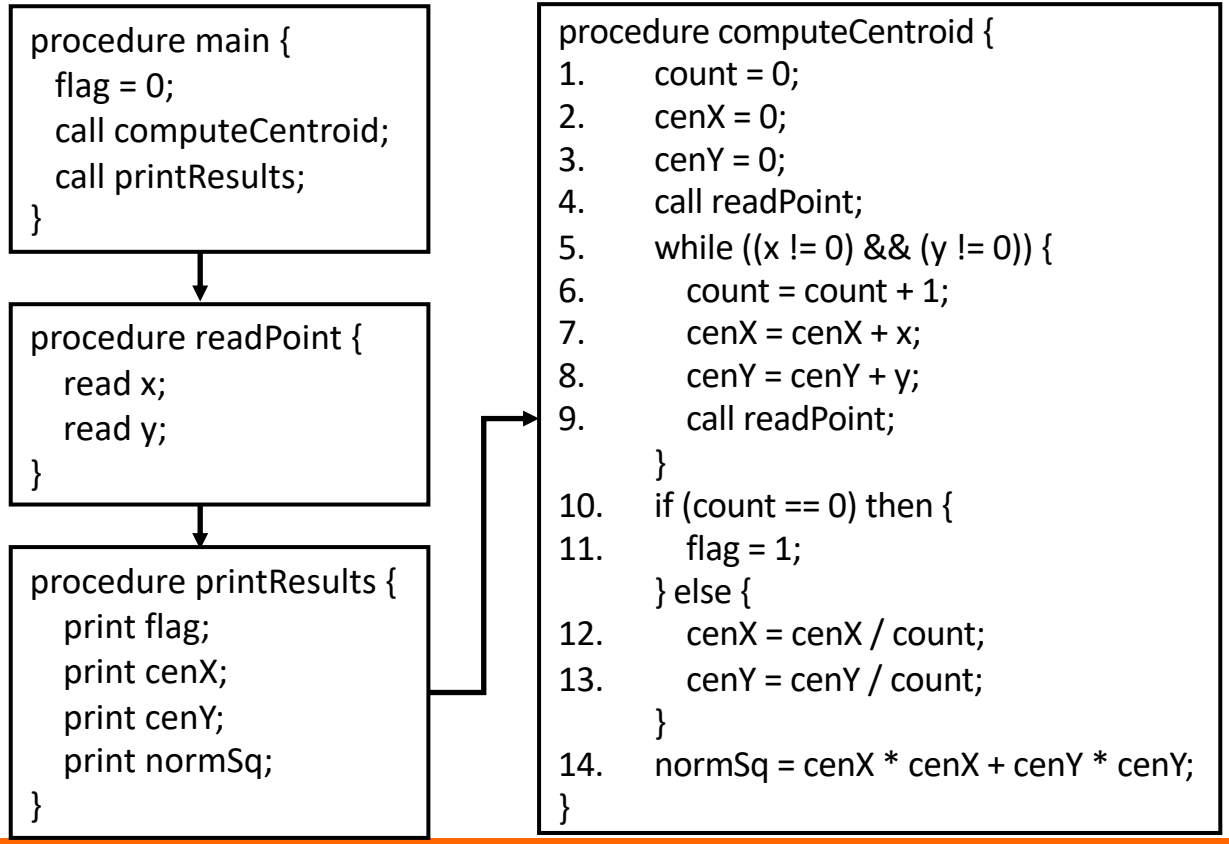
D. Uses (10, "flag")

E. Uses (10, "count")

F. Uses ("main", "flag")

Which procedures use the variable "cenX"?

Which variables are used in statement 5?



Modifies

Design entities	Description
Assignment a Variable v	Modifies (a, v) holds if variable **v** appears on the left hand side of **a**
Read statement r Variable v	Modifies (r, v) holds if variable **v** appears in **r**.
Container statement s (i.e. "if" or "while") Variable v	Modifies (s, v) holds if there is a statement **s1** in the container such that Modifies (s1, v) holds.
Procedure p, Variable v	Modifies (p, v) holds if there is a statement **s** in **p** or in a procedure called (directly or indirectly) from **p** such that Modifies (s, v) holds.
Procedure call c (i.e. "call p") Variable v	Modifies (c, v) is defined in the same way as Modifies (p, v).

Examples of Modifies

Which relationships hold?

A. Modifies (1, "x")

B. Modifies (7, "cenX")

C. Modifies (9, "x")

D. Modifies (10, "flag")

E. Modifies (5, "flag")

F. Modifies ("main", "y")

Which procedures modify the variable "cenX"?

Which variables are modified in statement 5?

```
procedure main {
  flag = 0;
  call computeCentroid;
  call printResults;
}
```

```
procedure readPoint {
   read x;
   read y;
}
```

```
procedure printResults {
   print flag;
   print cenX;
   print cenY;
   print normSq;
}
```

```
procedure computeCentroid {
1.    count = 0;
2.    cenX = 0;
3.    cenY = 0;
4.    call readPoint;
5.    while ((x != 0) && (y != 0)) {
6.       count = count + 1;
7.       cenX = cenX + x;
8.       cenY = cenY + y;
9.       call readPoint;
      }
10.   if (count == 0) then {
11.      flag = 1;
      } else {
12.      cenX = cenX / count;
13.      cenY = cenY / count;
      }
14.   normSq = cenX * cenX + cenY * cenY;
}
```

Query Language PQL
&
Common Program Queries

Examples of Program Queries

- What are the variables in the program?
- Which statements are assignment statements?
- Which statements appear after another statement?
- Which variables are modified in a statement?
- Which variables are used in a procedure?

Examples – PQL queries

Which variables have their values modified in statement 6?

variable v;
Select v such that Modifies (6, v)

Find assignments that contain expression “count + 1” on the right hand side

assign a;
Select a pattern a(_ , "count + 1")

BasicSPA Program Query Language (Basic PQL)

- Declaration of synonyms to be used in the query
  - Example: procedure p; variable v; (p: entity procedure, v: entity variable)
- Select clause specifies query result
  - Single return values (e.g., select v)
  - **At most one** such that clause constrains the results in terms of relationships (e.g., such that Modifies (6, v))
  - **At most one** pattern clause constrains results in terms of code patterns (e.g., pattern a (v, _))
- Query results must satisfy all clauses

BasicSPA Program Query Language (Basic PQL)

Meta symbols:
a* - repetition 0 or more times of a
a+ - repetition 1 or more times of a
[a] - repetition 0 or one occurrence of ‘a’
a | b - a or b
brackets (and) are used for grouping

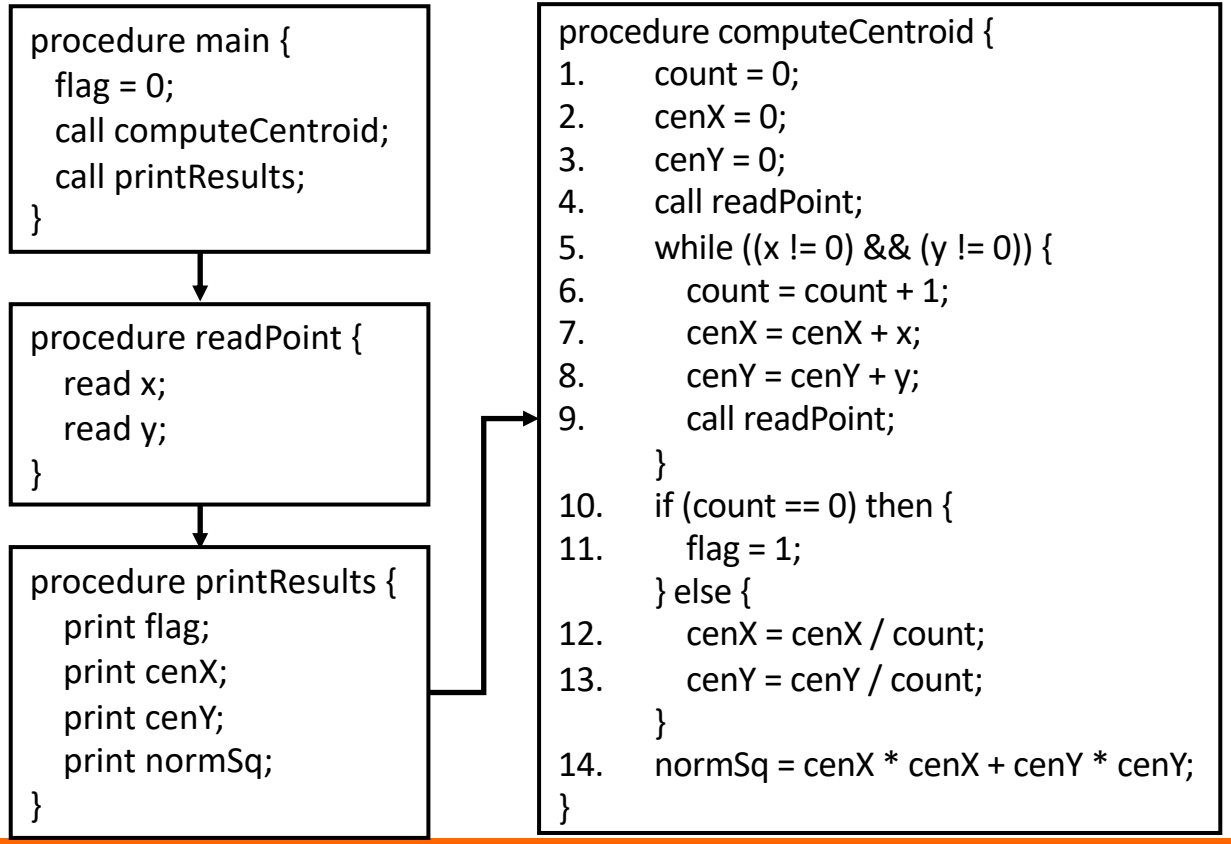
Lexical tokens:
LETTER: A-Z | a-z -- capital or small letter
DIGIT: 0-9
IDENT : LETTER (LETTER | DIGIT)*
NAME : LETTER (LETTER | DIGIT)*
INTEGER : DIGIT+
synonym : IDENT
stmtRef : synonym | ‘_’ | INTEGER
entRef : synonym | ‘_’ | ‘“’ IDENT ‘“’

Grammar rules:
select-cl : declaration* ‘Select’ synonym [suchthat-cl] [pattern-cl]
declaration : design-entity synonym (‘,’ synonym)* ‘;’
design-entity : ‘stmt’ | ‘read’ | ‘print’ | ‘call’ | ‘while’ | ‘if’ | ‘assign’ | ‘variable’ | ‘constant’ | ‘procedure’
suchthat-cl : ‘such that’ relRef

Grammar rules:
relRef : Follows | FollowsT | Parent | ParentT | UsesS | UsesP | ModifiesS | ModifiesP
Follows : ‘Follows’ ‘(’ stmtRef ‘,’ stmtRef ‘)’
FollowsT : ‘Follows*’ ‘(’ stmtRef ‘,’ stmtRef ‘)’
Parent : ‘Parent’ ‘(’ stmtRef ‘,’ stmtRef ‘)’
ParentT : ‘Parent*’ ‘(’ stmtRef ‘,’ stmtRef ‘)’
UsesS : ‘Uses’ ‘(’ stmtRef ‘,’ entRef ‘)’
UsesP : ‘Uses’ ‘(’ entRef ‘,’ entRef ‘)’
ModifiesS : ‘Modifies’ ‘(’ stmtRef ‘,’ entRef ‘)’
ModifiesP : ‘Modifies’ ‘(’ entRef ‘,’ entRef ‘)’
pattern-cl : ‘pattern’ syn-assign ‘(‘ entRef ‘,’ expression-spec ’)’
// syn-assign must be declared as synonym of assignment (design entity ‘assign’).
expression-spec : ‘“‘ expr’“’ | ‘_’ ‘“’ expr ‘“’ ‘_’ | ‘_’
expr: expr ‘+’ term | expr ‘-’ term | term
term: term ‘*’ factor | term ‘/’ factor | term ‘%’ factor | factor
factor: var_name | const_value | ‘(’ expr ‘)’
var_name: NAME
const_value : INTEGER

Examples of program queries in PQL

SIMPLE source for the examples in next few slides.



Examples of program queries in PQL

Q1. What are the variables in the program?

variable v;

Select v

-- answer: variables "flag", "count", "cenX", "cenY", "x", "y", "normSq"

Q2. Which statements follow assignment 6 directly or indirectly?

stmt s;

Select s such that Follows (6, s)*

-- answer: statements #7, #8, and #9

Examples of program queries in PQL

Q3. Which variables have their values modified in statement 6?

variable v;

Select v such that Modifies (6, v)

-- answer: variable "count"

Q4. Which variables are used in assignment 14?

variable v;

Select v such that Uses (14, v)

-- answer: variables "cenX" and "cenY"

Examples of program queries in PQL

Q5. Which procedures modify variable "x"?

procedure p;

Select p such that Modifies (p, "x")

-- answer: procedures "main", "computeCentroid" and "readPoint"

Q6. Find assignments within a loop.

assign a; while w;

Select a such that Parent (w, a)*

-- answer: statements #6, #7 and #8

Examples of program queries in PQL

Q7. Which is the parent of statement #7?

stmt s;

Select s such that Parent (s, 7)

-- *answer*: *statement #5*

Q8. Which are the assignments that use a variable?

assign a; variable v;
Select a such that Uses (a, v)
Select a such that Uses (a, _)

-- answer: statements #6, #7, #8, #12, #13 and #14.

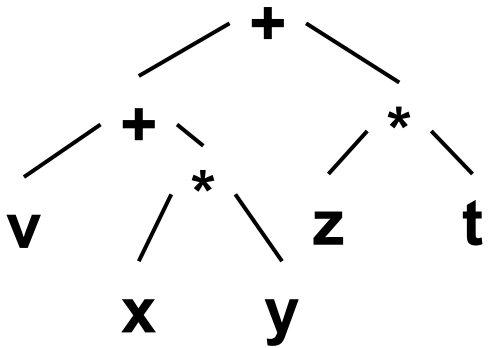
BasicSPA Program Query Language (Basic PQL)

- Declaration of synonyms to be used in the query
  - Example: procedure p; variable v; (p: entity procedure, v: entity variable)
- Select clause specifies query result
  - Single return values (e.g., select v)
  - **At most one** such that clause constrains the results in terms of relationships (e.g., such that Modifies (6, v))
  - **At most one** pattern clause constrains results in terms of code patterns (e.g., pattern a (v, _))
- Query results must satisfy all clauses

Examples of pattern queries

Which of the patterns match with this assignment statement?
*x = v + x * y + z * t*

A. a (_ , "v + x * y + z * t")
B. a (_ , "v")
C. a (_ , _"v"_)
D. a (_ , _"x*y"_)
E. a (_ , _"v+x"_)
F. a (_ , _"v+x*y"_)
G. a (_ , _"y+z*t"_)
H. a (_, _"x * y + z * t"_)
I. a (_ , _"v + x * y + z * t"_)



Examples of program queries in PQL

Q9. Find assignments that contain expression count + 1 on the right hand side

assign a;

Select a pattern a (_ , "count + 1")

-- answer: statement #6

Q10. Find assignments that contain sub-expression cenX * cenX on the right hand side and normSq on the LHS.

assign a;

*Select a pattern a ("normSq" , _"cenX * cenX"_)*

-- answer: statement #14

Examples of program queries in PQL

Q11. Find assignments that use and modify the same variable

assign a; variable v;

Select a such that Uses (a, v) pattern a (v, _)

-- answer: assignments #6, #7, #8, #12, and #13

Examples of program queries in PQL

Q12. Find while loops with assignment to variable "count"

assign a; while w;

Select w such that Parent (w, a) pattern a ("count", _)*

-- answer: statement #5

Order of conditions in program queries

- Changing the order of conditions in a query does NOT change the query result
  - assign a; while w;
  - Select a pattern a ("x", _) such that Uses (a, "x")
  - Select a such that Uses (a, "x") pattern a ("x", _)

  -- answer: None
  - Select a such that Parent* (w, a) pattern a ("count", _)
  - Select a pattern a ("count", _) such that Parent* (w, a)

  -- answer: statement #6
- BUT: Changing the order of conditions may affect query evaluation time.

Format of Results (Important!)

Select	Should return
Statement (stmt / read / print / call / while / if / assign)	Statement number – no “#” prefix Multiple statements should be separated by a space
Variable	Name (no need to use "") Multiple variable names should be separated by a space
Procedure	Name (no need to use "") Multiple procedure names should be separated by a space
Constant	Constant value Multiple constant values should be separated by a space
Empty result (no entities matching the query)	On paper/Tests, write keyword "none" In your SPA implementation, do not populate the list of results with any values (not even keyword "none")
Syntactically Invalid Query	On paper/tests, write “Invalid” and explain why In SPA implementation return empty result

Get familiar with SPA Vocabulary

- SPA
- PKB
- PQL
- AST
- Patterns
- Program Design Entities
- Program Design Abstractions
- Relationships
- Clause
- Synonym

Summary

- This lecture covered
  - (i) What is SPA and how does it work.
  - (ii) Source Language SIMPLE
  - (iii) Basic Program Design abstractions for SIMPLE
  - (iv) Rules for writing Basic Program Queries in PQL
- Further information about Advanced SPA requirements will be covered in a later lecture.