Goal of Document

• Plan for implementation
• Should be written to communicate to others your intent for implementation
• Gives you opportunity to think about potential problems in implementation
High-level design shows only conceptual classes in system, not details of data structures
Diagram Details

- Show a class even if you use a library class
- Diagram is about design not implementation
- Template stack class can be diagrammed showing argument
Low-Level Class Diagram

- Not what was asked for in assignment
- Show data structure details, for implementation
- Example: stack implementation (if write own)
Calculator Class Form (1)

Name: Calculator

Base class(es): none

Purpose: Implements RPN calculator for polynomials in variables x, y and z. Implements add and subtract operations.

States: none

Constructors:

  Calculator() - creates calculator with empty stack
  Calculator(Calculator) - creates copy of calculator
Calculator Class Form (2)

**Operations:**

*Mutators:*

- `enter(polynomial)` - push polynomial onto stack
- `add()` - add top two polynomials
- `subtract()` - subtract top two polynomials

*Accessors:*

- `display(ostream&)` - display contents of stack to stream

**Fields:**

- Stack of polynomials
Calculator Enter Function

**Prototype:** enter ( polynomial p)

**Purpose:** Pushes polynomial onto stack for future operations.

**Receives:** Polynomial object p.

**Returns:** nothing.

**Remarks:** When done stack should have p as top, and previous contents preserved.
Calculator Add Function

Prototype: add ()

Purpose: Adds top two polynomials on stack and pushes result. If not two polynomials, do nothing.

Receives: nothing.

Returns: boolean - true if addition performed.

Remarks: Stack size should be reduced by one, top element is sum of top two elements.
Calculator Subtract Function

Prototype: subtract ()

Purpose: Performs RPN subtraction using contents of stack, and pushes result on stack. If top element is b, second element a, then computes a-b. If not two polynomials, don’t do operation.

Receives: nothing.

Returns: boolean - true if subtraction performed.

Remarks: Stack size should be reduced by one, top element is difference of top two elements.
Calculator Display Function

Prototype: display ( ostream& os)
Purpose: Displays contents of stack starting with top element.
Receives: Output stream reference.
Returns: nothing.
Remarks: When function completes, stack should be unchanged.
Name: Stack

Base class(es): none.

Purpose: Implements stack of polynomials.

States: empty and not empty.

Constructors:

- Stack ( ) - creates empty stack
- Stack(Stack) - creates copy of stack
Operations:

Mutators:
push (polynomial) - push polynomial onto stack
pop () - remove and return top of stack

Accessors:
size () - return number of elements on stack
empty() - returns true if stack is empty, false otherwise
display(ostream&) - display contents of stack to stream

Fields:
List of polynomials
Stack Push Function

Prototype: `push ( polynomial p )`

Purpose: Makes polynomial p the top element of the stack.

Receives: Polynomial p.

Returns: nothing.

Remarks: When done stack should have p as top element, and otherwise stack should be the same as before.
Stack Pop Function

Prototype: pop ()

Purpose: If stack is not empty, removes and returns the top element of stack.

Receives: nothing.

Returns: top element of stack.

Remarks: When done stack should have same elements as before except for top.
Stack Display Function

**Prototype:** display (ostream& os)

**Purpose:** Display the contents of the stack from top element to bottom. If stack is empty, print “Empty Stack”.

**Receives:** Output stream reference.

**Returns:** nothing.

**Remarks:** When function completes, stack should be unchanged.
Polynomial Class Form (1)

Name: Polynomial
Base class(es): none.
Purpose: Polynomials in variables x, y and z.
States: none.
 Constructors:
  Polynomial ( ) - creates zero polynomial
  Polynomial(Polynomial) - creates copy of polynomial
Polynomial Class Form (2)

**Operations:**

**Mutators:**
- add (polynomial, polynomial) - add two polynomials
- subtract(polynomial, polynomial) - subtract two polynomials

**Accessors:**
- isZero( ) - returns true if polynomial is zero
- display(ostream&) - prints polynomial to stream

**Fields:**
- List of Terms
Polynomial Add Function

**Prototype:** `add (polynomial, polynomial)`

**Purpose:** Combines assignment and addition of polynomials. Assigns result of adding parameters to implicit polynomial object.

**Receives:** Two polynomials.

**Returns:** nothing.

**Remarks:** When done, implicit object should hold sum of two parameters, parameter objects should not be affected.
Polynomial Subtract Function

Prototype: subtract (polynomial, polynomial)

Purpose: Combines assignment and subtraction of polynomials. Assigns result of subtracting second parameters from first to implicit polynomial object.

Receives: Two polynomials.

Returns: nothing.

Remarks: When done, implicit object should hold difference of two parameters, parameter objects should not be affected.
Term Class Form (1)

**Name:** Term

**Base class(es):** none

**Purpose:** Polynomial terms in variables x, y and z.

**States:** none.

**Constructors:**
- Term ( ) - creates zero term (coefficient zero)
- Term(Term) - creates copy of term
- Term(coeff, monomial) - create nonzero term
Term Class Form (2)

**Operations:**

*Mutators:*
- `combine (term, term)` - add coefficients if monomials equal

*Accessors:*
- `getCoeff()` - return coefficient
- `getMonomial()` - return monomial
- `isZero()` - returns true if term is zero (coefficient is zero)
- `pretty()` - returns object for printing

**Fields:**
- Coefficient
- Monomial
Term Combine Function

Prototype: combine (term, term)

Purpose: Combines terms with equal monomials by adding coefficients. If monomials are not equal, does nothing.

Receives: Two terms.

Returns: nothing.

Remarks: When done, implicit object should hold sum of coefficients of parameters and same monomial as parameters, parameter objects should not be affected.
Monomial Class Form (1)

Name: Monomial

Base class(es): none.

Purpose: Monomial in variables x, y and z.

States: none.

Constructors:

Monomial ( ) - creates unit monomial (exponents all zero)
Monomial(Monomial) - creates copy of monomial
Monomial(exponent, exponent, exponent) - create monomial with given exponents for x, y and z.
Monomial Class Form (2)

Operations:

*Mutators:* None

*Accessors:*
  - compare(term) - compare monomials
  - isUnit() - returns true if all exponents are zero
  - pretty() - returns object for printing

Fields:

Exponents for x, y and z
Monomial Compare Function

**Prototype:** compare (term)

**Purpose:** Lexicographically compare implicit term with parameter. Return whether implicit is less than, equal to, or greater than parameter.

**Receives:** Implicit and parameter terms.

**Returns:** Less, Equal, or Greater

**Remarks:** Return type should probably be enumerated type. Can use to implement comparison operators.
Lexicographic Comparison

Monomial $m$ is less than monomial $n$ if
1. $m.x < n.x$; or, if $m.x = n.x$ then
2. $m.y < n.y$; or, if $m.y = n.y$ then
3. $m.z < n.z$.

Otherwise, if all three exponents are equal then $m$ and $n$ are equal
Planning Ahead

• Parsing
  – separate from Calculator class
  – Easier to change parser

• Printing
  – pretty() functions in term and monomial classes
  – hooks for project 3
  – for project 2 can just make return type typecast to string