Project 3 Design

BJK’s HW 4 Sample Solution
Fall 2000
Token Class Diagram

Note: Lexer class is Token factory
Alternative Class Diagram

Token
- getType()
- getLexeme()

Lexer
- getToken()

NumberTok
- getNumber()
Token Class Form (1)

Name: Token
Base class(es): none
Purpose: tokens for parser.
States: none
Constructors:
  Token(string, toktype) – creates token
  Token(Token) – creates copy of token
Token Class Form (2)

Operations:

Mutators: none

Accessors:

getType() – return type of token
getLexeme() – return string of token

Fields:

type
lexeme – string for token
Token Types

• Use enumerated type for token type

• Values:
  – Number
  – Plus, Minus, Power
  – Enter, Add, Subtract, Display, Exit
  – Identifier

• Allows token to be used in switch on token type

• Note: if want to recognize end of file, include EOF
Name: NumberTok
Base class(es): Token.
Purpose: Number token.
States: none.
 Constructors:
   NumberTok (string) - creates number token
   NumberTok(NumberTok) - creates copy of token
NumberTok Class Form (2)

Operations:

Mutators: none.

Accessors:

getNumber() - return number value of token

Fields:

number
IdentTok Class Form (1)

Name: IdentTok
Base class(es): Token.
Purpose: Token for identifiers.
States: none.
Constructors:
   IdentTok (string) – creates identifier token
   IdentTok(IdentTok) – creates copy of token
IdentTok Class Form (2)

Operations:

*Mutators*: none.

*Accessors*: none.

Fields:

none.
KeywordTok Class Form (1)

Name: KeywordTok
Base class(es): Token.
Purpose: Token for keywords.
States: none.
Constructors:
  KeywordTok (string) – creates keyword token
  KeywordTok(KeywordTok) – creates copy of token
KeywordTok Class Form (2)

Operations:

*Mutators*: none.

*Accessors*: none.

Fields:

none.
SymbolTok Class Form (1)

**Name:** SymbolTok

**Base class(es):** Token.

**Purpose:** Token for symbols.

**States:** none.

**Constructors:**
- SymbolTok (string) – creates symbol token
- SymbolTok(SymbolTok) – creates copy of token
SymbolTok Class Form (2)

**Operations:**

*Mutators*: none.

*Accessors*: none.

**Fields:**

none.
SymbolTok Constructor

Prototype: SymbolTok (string)

Purpose: Constructs token object for symbol string. Uses string to determine type.

Receives: Lexeme string.

Returns: nothing.

Remarks: Must set type of object according to string.
Design Issue

• Example of problems should realize in design:
  – How will derived class constructors set type of token in body?
  – Has to do with C++ implementation

• Requires a change to responsibilities
  – Give token class protected mutator
  – Move typing responsibility elsewhere (later)
Lexer Class Form (1)

Name: Lexer
Base class(es): none
Purpose: Constructs stream of tokens from text file.
States: none.
 Constructors:
  Lexer (filename) – creates lexer for file
  Lexer(Lexer) - creates copy of lexer
Lexer Class Form (2)

Operations:

Mutators:

getToken() – extract token from file

Accessors: none.

Fields:

File stream

Current line
Lexer getToken Function

Prototype: getToken ( )
Purpose: Extracts next token from file.
Receives: nothing.
Returns: nothing.
Remarks: Needs to read line by line so can ignore comment lines.
Alternative Class Diagram

Token

- getType()
- getLexeme()

Lexer

- getToken()

NumberTok

- getNumber()
Design Alternative

- Using only one derived class
  - Requires Lexer or Token class to figure out type
  - Probably don’t want Token to know about types
  - Lexer needs to identify types anyway

- Using different derived classes
  - Derived class manages any needed conversion
  - Lexer just has to identify type, not convert
Pretty Printer Class Diagram

Pretty Object

- length()
- print(…)

* *

Pretty String

Pretty Break

Pretty Block
Pretty Object is bridge object – avoid dealing with pointers
Pretty Object Class Form (1)

**Name:** Pretty Object

**Base class(es):** none

**Purpose:** (Abstract) Base pretty printer object

**States:** none.

**Constructors:** none.
Pretty Object Class Form (2)

Operations:

*Mutators*: none.

*Accessors*:

- `length()` – return length of object
- `print(os, space, blockindent, after, margin)`

Fields:

none.
Pretty String Class Form (1)

Name: Pretty String
Base class(es): Pretty Object
Purpose: pretty printer object for a string.
States: none.
Constructors:
  PrettyString(string) – string object
  PrettyString(PrettyString) – copy pretty string
Pretty String Class Form (2)

Operations:

*Mutators*: none.

*Accessors*:

print(os, space, blockindent, after, margin)
length()

Fields:

string.
Pretty String Print Function

Prototype: print(os, space, blockindent, after, margin)

Purpose: Print string in pretty printed form to os.

Receives: output stream, integers: space left on line, indent for new line, distance to end of block, and margin of text.

Returns: nothing.

Remarks: Should decrease space by length of string after printing.
Pretty Break Class Form (1)

Name: Pretty Break
Base class(es): Pretty Object
Purpose: pretty printer object for a break. Prints as either a space of specified length or a new line.
States: none.
Constructors:
    PrettyBreak(length) – break object of length
    PrettyBreak(PrettyBreak) – copy break
Pretty Break Class Form (2)

Operations:

Mutators: none.

Accessors:

print(os, space, blockindent, after, margin)
length()

Fields:

length.
Pretty Break Print Function

Prototype: print(os, space, blockindent, after, margin)

Purpose: Print break to os as either length spaces, or as newline if not enough room.

Receives: output stream, integers: space left on line, indent for new line, distance to end of block, and margin of text.

Returns: nothing.

Remarks: If enough space for block left on line, then should decrease space by length of string after printing number of spaces. Otherwise, print newline, indentation spaces, and set space to be margin.
Pretty Break Print Pseudo-Code

Object: Pretty Break
Service: Print
Input: os, space, blockindent, after, margin

if break length + after <= space then
    print length number of spaces;
decrease space by length
else //not enough space
    print newline; reset space to margin;
    print margin - blockspace spaces; decrease space
Pretty Block Class Form (1)

Name: Pretty Block
Base class(es): Pretty Object
Purpose: pretty printer object for a block of text that should ideally be printed on a line. Will also print as indented on new line if doesn’t fit on one line.
States: none.
Constructors:
   PrettyBlock(list, indent) – block object
   PrettyBlock(PrettyBlock) – copy block
Pretty Block Class Form (2)

Operations:

Mutators: none.

Accessors:

print(os, space, blockindent, after, margin)

length()

Fields:

length, indent, list of Pretty Objects
Pretty Block Print Function

Prototype: print(os, space, blockindent, after, margin)

Purpose: Print each object in block to os.

Receives: output stream, integers: space left on line, indent for new line, distance to end of block, and margin of text.

Returns: nothing.

Remarks: none.
Pretty Block Print Pseudo-Code

Object: Pretty Block
Service: Print
Input: os, space, blockindent, after, margin

blockindent = space - indent

for each object obj in list of block
    breakdist = break distance for rest of block
    obj.print(os, space, blockindent, breakdist, margin)
Design Notes

- Pseudo-Code is form of low-level design
- Used to communicate specific algorithm for function if necessary
- (Not required by HW assignment)
Tokens and Rest of Program

• Token and Lexer class
  – Parser has Lexer object for input file
  – Calls getToken to get next token and decide what to do using Token::getType()
  – Use either lexeme or number to extract information to build polynomial
Pretty Printer and Rest of Program

• When display polynomial construct a block object for polynomial
• Scheme: block for term, string for operation, breaks in between
• So, $3x^2 y^3 + z$ would become one block containing: a block for $3x^2 y^3$, a length 1 break, a string object for +, a length 1 break, and a block for $z$
Classes and Rest of Program

• These classes have no direct relationship to other classes and are not shown in same diagram

• Would change details of pretty() function for Term and Monomial classes