Component Design:

Main

main (int argc, const char \*argv[])

Create a scanner

Create a token

Have scanner start reading the file

Check for <identifiers>

Return 0

Print

Print (char source\_name[], char date[])

Set the filename to source\_name

get current time

set the print date to the current time

set the page number to zero

printLine (char line[])

increment the line count

if line\_count> page height Print header

print the string argument

PrintPageHeader ()

Print header (page number, source file name , current date)

PrintToken (Token \*token)

Increment the line count

Switch (token-> getCode()) {

NUMBER is an integer ->print integer

Number is a real -> print a real

String is a string -> print the string

Default -> print token

}

Scanner

Scanner (FILE \*source\_file, char source\_name[], char date[], Print printer)

Src\_file= source\_file

Copy (src\_name, source name)

Copy (todays\_date, date)

Initialize char table to identify what type of char we are looking at

Initialize Line numer=0

Source line [0] = ‘\0’

getSourceLine(char source\_buffer)

create source buffer

create fale Boolean

get a line from the filestream

if line received then true

return Boolean

getToken()

initialize a character code variable

skip past all the blanks

examine ch for LETTER, DIGIT, QUOTE, EOF, or SPECIAL

call appropriate function depending on ch

return new\_token

getChar(char souce\_buffer[])

set a temp char to EOF

if at the end of line ->return null character

else return the char at the index

skipBanks (char source\_buffer[])

skip past the blanks

return pointer to the first non blank character

skipComments (char source\_buffer[])

skip past the comments

return pointer to the first non blank character

watch for the EOF character

getWord (char \*str, char \*token\_ptr. Token \*tok)

Extract the word

Downshift the word, to make it lower case

Check if the word is a reserved word

If is not a reserved word its an identifier

Set token to identifier

getNumber (char \*str, char \*token\_ptr, Token \*tok)

extract number and convert it to a literal number

check if real or float

temp string number

set the token type to NUMBER

getString (char \*str, char \*token\_ptr, Token \*tok)

Initialize a temporary string

Whie char ch is not a ‘\’’

Read more characters

Append characters to temp string

Set the setType to STRING\_LIT

Set the setCode to STRING

getSpecial (char \*str, char \*token\_ptr, Token \*tok)

initialize the temp string

check for character operators (:,<,>,.,|)

read next character

if (= or .) -> append both to temp string

else -> append first character to the temp string

downshiftWord (char word[])

make all characters in the incoming word lower case

isReservedWord (char \*str, Token \*tok)

Scan the token table for reserved words

If it is a reserved word -> set the token code member ->return True

Else -> return False

getLineNumer ()

return line\_number

Token

Token ()

Initialize variables for binary search tree (lines, left, righ)

setCode (TokenCode newCode)

set newCode

getCode ()

Return token code member

setType (LiteralType newType)

set Type

getType ()

return Token code member

setLiteral (int newInteger)

setLiteral to integer

getIntLiteral ()

return Token code member

setLiteral (int newReal)

setLiteral to real

getRealLiteral ()

return Token code member

setLiteral (string newString)

setLiteral to String

malloc space for string

copy stringLiteral to newString

getStringLiteral ()

return Token code member

setTokenString (string s)

setTokenString = s

getTokenString (string s)

return Token code member

getLeft ()

get left “leaf”

getRight ()

get right “leaf”

//implement binary tree

addLineNumber (int lineNumber)

add line number to node

addTokenNodeToBinarySearchTree(Token\* &headToken, Token\* newToken, int lineNumber)

Add token node to the binary search tree

getLinesString ()

return lone to ToString Expand

getBinarySearchTreeLinesStringsInOrder (Token\*head)

arrange print of head

get head left or right

LineNumberNode

LineNumberNode ()

Set val to 0

Set next to NULL

LineNumberNode (int val)

Set val to val

Set next to NULL

addNode (LineNumberNode \*&headNode, LineNumberNode\* newNode)

if headNode= Null

headNode is NewNode

return

else

ptr next= newNode

return