Compiler Design CS3071 - Lab 2

David Kelly

October 17, 2014

Contents

1 Introduction

This report will describe the design of a Lexical Analyser which can process a 32-bit integer, octal or hexadecimal constant defined by the following regular expression:

$$((+|-)?[0-9]+)|[0-7]+[bB]|[0-9a-fA-F]+[hH] \tag{1}$$

2 Transition Table

The transition table to check for a valid integer, octal or hexadecimal constant is shown below. It starts in state 1 and it has a total of 8 states.

State 1

This is the first state where the state machine begins. If an octal digit is input, the state machine will transition to state 2. If the input is an 8 or a 9, it will transition to state 3. If a hexadecimal digit is input, transition to state 4. If a sign is entered in this state, the machine will transition to state 5, expecting only decimal digits.

State 2

This state is expecting octal digits, if an octal digit is input the machine will stay in this state (State 2). If a decimal or hexadecimal digit is input, there is no way that the string can be an octal number and so the machine will transition to states 3 and 4 respectively. If the letter "b" is input, the machine transitions to state 7 which will check if the string is indeed a valid octal number or just a hexadecimal number. If the letter "h" is input, the machine will transition to state 8, where it will check if the string is a valid hexadecimal constant. If an end marker is received, then the string must be an integer as only octal digits have been received with no "b" to indicate that the string is an octal constant.

State 3

If a decimal digit is received, the machine will stay in this state, expecting more decimal digits. If a hexadecimal value is received, the machine will transition to state 4. If the letter 'h' is received, then the machine will transition to state 8, expecting the end of a hexadecimal number. If the end marker is received, the string is an integer.

State 4

If a hexadecimal digit is received, the machine will stay in state 4 expecting more hexadecimal digits. If a 'h' is received, the machine will transition to state 8 to check for the end of a hexadecimal string.

State 5

State 5 is expecting a decimal digit to transition to state 6. The purpose of this state is to check that a decimal digit follows a sign at the beginning of a string.

State 6

When decimal digits are received the machine will stay in this state. Since the machine has already received a sign at the beginning of the string in this state, we know that the string must represent an integer to be valid. If an end marker is received, then the string is an integer.

State 7

State 7 is needed to check whether a 'b' in a string will indicate the end of an octal value or if it is part of a hexadecimal number. If the end marker is received, the string is an octal value.

State 8

In this state, the end marker indicates a hexadecimal string as the previous input was hexadecimal digits followed by a 'h'.

State	OCT	DEC	b — B	HEX	SIGN	h — H	-I
1	2	3		4	5		
2	2	3	7	4		8	DEC
3		3		4		8	DEC
4		4		4		8	
5		6					
6		6					DEC
7		4		4		8	OCT
8							

Table 1: Transition Table

Note: Empty cells indicate error state.

Symbol	Description
OCT	Digits from 0 - 7 (A subset of DEC)
DEC	Digits from 0 - 9
b — B	The letter 'b' or 'B' (A subset of HEX)
HEX	Digits from 0 - 9 — a-f — A - F
SIGN	'+' — '-'
h — H	The letter 'h' or 'H'
-I	End Marker

Table 2: Transition Table Symbols

3 Overflow

In the implementation of the Lexical Analyser, overflow is checked by analysing the string input (separate from the state machine). This is done after a Lexical Token has been generated by the program. The program checks the length of the value as part of checking for overflow. For this reason, a value with leading zeros is treated as invalid input.

3.1 Integer Overflow

To check for integer overflow, the program performs the following steps:

- If the length of the input is *greater than* the length of the maximum integer (positive or negative), then there is overflow.
- If the length of the input is *less than* the length of the maximum integer (positive or negative), then there is overflow.
- If the length of the input is *equal to* the length of the maximum integer (positive or negative), then further checking takes place.
- The most significant digit of the input is compared to the most significant digit of the maximum integer. If the input is greater than the maximum integer digit, then overflow has occured. If the input digit is equal to or less than the maximum integer digit then overflow has not occured so far and the program continues to check the rest of the digits.

3.2 Octal Overflow

To check for octal overflow, the program performs the following steps:

- If the length of the input is *greater than* the length of the maximum octal number, then there is overflow.
- If the length of the input is *less than* the length of the maximum octal number, then there is overflow.
- If the length of the input is *equal to* the length of the maximum octal number, then further checking takes place.
- The most significant digit of the input is compared to the most significant digit of the maximum octal number value. If the input digit is greater than the maximum octal digit, then overflow has occured, otherwise there is no overflow.

3.3 Hexadecimal Overflow

To check for hexadecimal overflow, the program performs the following steps:

- If the length of the input is *greater than* the length of the maximum hexadecimal number, then there is overflow.
- If the length of the input is *less than* or *equal to* the length of the maximum hexadecimal number, then there is overflow.

4 Test Inputs

A list of test inputs that will visit all non-error states in the transition table.

0
(HEXADECIMAL, 0) Integer Representation: 0
Program Successful: True

1234
Expected Result: (INTEGER, 1234) Integer Representation: 1234
Program Successful: True

1238
Expected Result: (INTEGER, 1238) Integer Representation: 1238
Program Successful: True

+1238
Expected Result: (INTEGER, +1238) Integer Representation: 1238
Program Successful: True

-1238
Expected Result: (INTEGER, -1238) Integer Representation: 1238
Program Successful: True

1341b
Expected Result: (OCTAL, 1341b) Integer Representation: 737
Program Successful: True

1324bH
Expected Result: ((HEXADECIMAL, 1324bH) Integer Representation: 78411
Program Successful: True

1324bbH
(HEXADECIMAL, 1324bbH) Integer Representation: 1254587
Program Successful: True

5 Implementation

The program comprises of 3 java classes. Each of which is described briefly below.

5.1 LexicalAnalyserApp.java

This class is the main class containing the main method. It takes user input as a string and passes it to the Lexical Analyser class where it will be converted into a Lexical Token.

5.2 LexicalAnalyser.java

This class contains a static method analyseString(). This method contains a state machine that parses through the input string. The state machine is coded from the state transition table shown previously in this report. If the state machine does not encounter any errors, a Lexical Token will be generated. If the state machine encounters an error or a value has overflow or leading zeros, then an exception is thrown and the user is notified.

5.3 LexicalToken.java

This class represents a Lexical Token. It contains a field for the class and a field for the value. It also contains methods that check for leading zero's and overflow. It also converts the string value into an integer value. If an overflow occurs or leading zeros are detected, an exception is thrown and the program exits with a notification for the user.