Vahe Karamian
Homework # 2 - CS 440 summer 2001
Due Date: Tuesday, July 10

1) What is the input alphabet of each of the following languages?

a) C++ uses ANSII as an input alphabet
b) JAVA uses UNICODE as an input alphabet
c) Pascal uses ANSII as an input alphabet

2) What are the conventions regarding the use of identifiers and numeric constants (integer literals and floating-point literals) in each of the languages listed above?

In C++ an identifier is a series of characters consisting of letters, digits, and underscores (_) that does not begin with a digit. C++ is case sensitive, therefore uppercase and lowercase letters are different, so a1 and A1 are different identifiers.

C++ Integer Literals
Are interpreted as of type {int,unsigned int, long int, long unsigned int} Can be specified in hexadecimal (0xFF, -0X1e), octal (+017, 07777), or decimal (24,-10), with or without a + or - sign. The (x or X)/(0)/(no) prefix selects the number base (16/8/10). If no - is present, the integer literal can be made unsigned with a (u or U) suffix: 0x10u, 24u. The integer literal can be forced to be twice the normal bit-length ("double-precision", and double the range of representable numbers) with the suffix (l or L). The textbook rightly recommends use of 'L' however to avoid confusion of 'l' with '1'. Examples: -0x44e0L, 045UL. Despite what the textbook says, (unsigned) integer literals do not default to type (unsigned) int, but rather are assigned to one of the basic integer types {int, unsigned int, long int, unsigned long int} on a "smallest-shoe to fit" basis. The type short int is not available for integer literals (though this is not really a problem; see discussion in a later lecture on casting)

C++ Floating-Point Literals
Are of type {float, double, long double) of the form: -3.0E4 (double) , 0.0241 (double) , 3e4 (double) , 4.0f-4 (float) , -23L-43 (long double) only base-10 in contrast to integer literals, default is simple - with only a decimal point (no exponent), they are type double; otherwise the exponent symbol specifies float (single-precision), double, and long double (l or L).

In JAVA an identifier is a series of characters consisting of letters, digits, underscores (_) and dollar signs ($) that does not begin with a digit and does not contain any spaces. Some valid identifiers are Welcome1, $value, _value, etc…

JAVA Integer Literals
An integer literal is of type long if it is suffixed with an ASCII letter L or l (ell); otherwise it is of type int (§4.2.1). The suffix L is preferred, because the letter l (ell) is often hard to distinguish from the digit 1 (one). A decimal numeral is either the single ASCII character 0, representing the integer zero, or consists of an ASCII digit from 1 to 9, optionally followed by one or more ASCII digits from 0 to 9, representing a positive integer:

DecimalNumeral:
0
NonZeroDigit Digitsopt

Digits:
Digit
Digits Digit

Digit:
0
NonZeroDigit

NonZeroDigit: one of
1 2 3 4 5 6 7 8 9

A hexadecimal numeral consists of the leading ASCII characters 0x or 0X followed by one or more ASCII hexadecimal digits and can represent a positive, zero, or negative integer. Hexadecimal digits with values 10 through 15 are represented by the ASCII letters a through f or A through F, respectively; each letter used as a hexadecimal digit may be uppercase or lowercase.

HexNumeral:
0 x HexDigits
0 X HexDigits

HexDigits:
HexDigit
HexDigit HexDigits

The following production from §3.3 is repeated here for clarity:

HexDigit: one of
0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

An octal numeral consists of an ASCII digit 0 followed by one or more of the ASCII digits 0 through 7 and can represent a positive, zero, or negative integer.

OctalNumeral:
0 OctalDigits

OctalDigits:
OctalDigit
OctalDigit OctalDigits

OctalDigit: one of
0 1 2 3 4 5 6 7

Note that octal numerals always consist of two or more digits; 0 is always considered to be a decimal numeral-not that it matters much in practice, for the numerals 0, 00, and 0x0 all represent exactly the same integer value. The largest decimal literal of type int is 2147483648 (231). All decimal literals from 0 to 2147483647 may appear anywhere an int literal may appear, but the literal 2147483648 may appear only as the operand of the unary negation operator -.

The largest positive hexadecimal and octal literals of type int are 0x7fffffff and 017777777777, respectively, which equal 2147483647 (231-1). The most negative hexadecimal and octal literals of type int are 0x80000000 and 020000000000, respectively, each of which represents the decimal value -2147483648 (-231). The hexadecimal and octal literals 0xffffffff and 037777777777, respectively, represent the decimal value -1.

A compile-time error occurs if a decimal literal of type int is larger than 2147483648 (231), or if the literal 2147483648 appears anywhere other than as the operand of the unary - operator, or if a hexadecimal or octal int literal does not fit in 32 bits.

Examples of int literals:

0   2   0372   0xDadaCafe   1996   0x00FF00FF

The largest decimal literal of type long is 9223372036854775808L (263). All decimal literals from 0L to 9223372036854775807L may appear anywhere a long literal may appear, but the literal 9223372036854775808L may appear only as the operand of the unary negation operator -. The largest positive hexadecimal and octal literals of type long are 0x7fffffffffffffffL and 0777777777777777777777L, respectively, which equal 9223372036854775807L (263-1). The literals 0x8000000000000000L and 01000000000000000000000L are the most negative long hexadecimal and octal literals, respectively. Each has the decimal value -9223372036854775808L (-263). The hexadecimal and octal literals 0xffffffffffffffffL and 01777777777777777777777L, respectively, represent the decimal value -1L.

A compile-time error occurs if a decimal literal of type long is larger than 9223372036854775808L (263), or if the literal 9223372036854775808L appears anywhere other than as the operand of the unary - operator, or if a hexadecimal or octal long literal does not fit in 64 bits.

Examples of long literals:

0l   0777L   0x100000000L   2147483648L   0xC0B0L

JAVA Floating-Point Literals

A floating-point literal has the following parts: a whole-number part, a decimal point (represented by an ASCII period character), a fractional part, an exponent, and a type suffix. The exponent, if present, is indicated by the ASCII letter e or E followed by an optionally signed integer.

At least one digit, in either the whole number or the fraction part, and either a decimal point, an exponent, or a float type suffix are required. All other parts are optional.

A floating-point literal is of type float if it is suffixed with an ASCII letter F or f; otherwise its type is double and it can optionally be suffixed with an ASCII letter D or d.

```
FloatingPointLiteral:
          Digits . Digitsopt ExponentPartopt FloatTypeSuffixopt
          . Digits ExponentPartopt FloatTypeSuffixopt
          Digits ExponentPart FloatTypeSuffixopt
          Digits ExponentPartopt FloatTypeSuffix

ExponentPart:
          ExponentIndicator SignedInteger

ExponentIndicator: one of
          e E

SignedInteger:
          Signopt Digits

Sign: one of
          + -

FloatTypeSuffix: one of
          f F d D
```

The elements of the types float and double are those values that can be represented using the IEEE 754 32-bit single-precision and 64-bit double-precision binary floating-point formats, respectively.
The details of proper input conversion from a Unicode string representation of a floating-point number to the internal IEEE 754 binary floating-point representation are described for the methods valueOf of class Float and class Double of the package java.lang.

The largest positive finite float literal is 3.40282347e+38f. The smallest positive finite nonzero literal of type float is 1.40239846e-45f. The largest positive finite double literal is 1.79769313486231570e+308. The smallest positive finite nonzero literal of type double is 4.94065645841246544e-324.

A compile-time error occurs if a nonzero floating-point literal is too large, so that on rounded conversion to its internal representation it becomes an IEEE 754 infinity. A program can represent infinities without producing a compile-time error by using constant expressions such as 1f/0f or -1d/0d or by using the predefined constants POSITIVE_INFINITY and NEGATIVE_INFINITY of the classes Float and Double.

A compile-time error occurs if a nonzero floating-point literal is too small, so that, on rounded conversion to its internal representation, it becomes a zero. A compile-time error does not occur if a nonzero floating-point literal has a small value that, on rounded conversion to its internal representation, becomes a nonzero denormalized number.

Predefined constants representing Not-a-Number values are defined in the classes Float and Double as Float.NaN and Double.NaN.

Examples of float literals:

```
1e1f      2.f       .3f       0f        3.14f     6.022137e+23f
```

Examples of double literals:

```
1e1       2.        .3        0.0       3.14                1e-9d               1e137
```

There is no provision for expressing floating-point literals in other than decimal radix. However, method intBitsToFloat of class Float and method longBitsToDouble of class Double provide a way to express floating-point values in terms of hexadecimal or octal integer literals.
For example, the value of:

```
Double.longBitsToDouble(0x400921FB54442D18L)
```

is equal to the value of Math.PI.

In Pascal an identifier must begin with a letter from the English alphabet. Can be followed by alphanumeric characters (alphabetic characters and numerals), or the underscore (_). May not contain special characters, such as: ~ ! @ # $ % ^ & * ( ) _ + ` - = { } [ ] : " ; ' < > ? , . / | \

Pascal Integer Literals

Whole numbers are numbers with no fractional part.
Leading zeroes are not significant in whole numbers

The syntax for whole numbers is

```
   whole-number = decimal-whole-number |
                  hexadecimal-whole-number |
                  binary-whole-number
```

Irie Pascal supports whole numbers with values between
-2147483647 and +4294967295

Decimal whole-numbers:
A decimal whole numnber uses base 10.

The syntax for decimal whole numbers is as follows:

```
   decimal-whole-number = digit-sequence

   digit-sequence = digit { digit }

   digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

Here are some examples of valid decimal whole number

```
   100 0 7453 000005
```

Pascal Floating-Point Literals

Real numbers are numbers with fractional parts. Leading zeroes are not significant in reals.

The syntax for real numbers is as follows

```
   real-number =
      digit-sequence '.' fractional-part [ exponent scale-factor ] |
      digit-sequence exponent scale-factor
```

digit-sequence = digit { digit }

fractional-part = digit-sequence

exponent = 'e' | 'E'

scale-factor = [sign] digit-sequence

Here are some examples of real numbers:
1.23456e2 which is equal to 123.456
1.23456e02 which is also equal to 123.456
009863434455e-07 which is equal to 986.3434455
7e-1 which is equal to 0.7

Irie Pascal supports real numbers with values between about

1e308

and about

-1e308

3) Given the following piece of simple Java program as the input to a scanner, what will be the output of the scanner (please show the detailed token stream)? Also, please define (use regular expressions/definitions to define) each of the tokens used here.

```
Public class Laugh {
          Public Laugh( ) { }
          Public void laugh( ) {
                    System.out.println( “haha” );
          }
}
```

TOKEN	LEXEMES
t_public	public
t_class	Class
t_id	L(L|D)*
t_void	Void
t_period	.
t_leftparenthesis	(
t_rightparenthesis	)
t_leftbraces	{
t_rightbraces	}
t_semicolon	;
t_quotation	“
t_ws	
t_System	System
t_out	.out
t_println	.println

```
t_public t_class <t_id, 0> t_leftbraces
t_public <t_id, 1> t_leftparethesis t_rightparenthesis t_leftbraces t_rightbraces
t_public t_void <t_id, 2> leftparethesis t_rightparenthesis t_leftbraces
t_System t_period t_out t_period T_println t_leftparenthesis t_quotation <String,
haha> t_quotation t_rightparenthesis t_semicolon
t_rightbraces
t_rightbraces
```