Tokenizer

Wael Ayadi and Parth Mehrotra

Our Tokenizer was created in a way that would minimize "bad tokens". Our program is structured largely as a finite state machine that takes each character as an input. The states and several of the testcases that highlight features are listed bellow.

Character Read Flow:

First character is alphabetic (a-z)

- *Word case*

First character is a non-zero numeric character (1-9)

- Assume *Decimal case*.

First character is zero

Maybe decimal, octal, float or hexadecimal. * *Zero case*

First character is a whitespace character

Space (' '), tab ('\t'), vertical tab ('\v'), form feed ('\f'), newline ('\n') and carriage return ('\r'). * Ignore this character. Skip to the next character.

First character is a period

Maybe character period or start of a float. * *Period Case*

First character is a minus sign.

Maybe negative decimal, negative float, or Special character '-' * *Minus case*

First character is a special character (excluding '.' and '-')

- *Special Character case*

If anything else

- *Bad Token case*

CASE LIST

Word Case

EX: "ABC123" is one word. * Upon reaching anything that's not alphanumeric, *END*.

Decimal Case

EX: "12354" is a decimal. * if reaches anything that's not numeric, *END* * if reaches a period: * If the character that follows the period is non-numeric, then it's a *END* + *Special Character 'period'*
EX: "456." is *Decimal 456* + *Special Character '.'* * Otherwise, it is *Float case*.

Period Case

NOTE: Period case is not a Token. It is either part of *Float case* or *Special Character case*
EX: "." is period case. "-." is also period case. * Check following character: * if it is a numeric, *Float case* * else, *Special character '.'* or *Special character '-'* + *Special character '.'*

Minus Case

NOTE: Minus case is not a Token. It is either a *Decimal*, *Float*, or *Special Character*
EX: "-" is minus case. * Check the following character: * if it is numeric, *Decimal case*. * if it is a period, *Period case*. * if it is a '>', then *Special Character "Structure Pointer"*

Float Case

EX: "3.14" is one float. * Continue until reaching any non-numeric character or the letter "e". * if it is not the letter e, *END* * else, check the following character * if it is non-numeric or not '-', *END* + continue with character e. * if it is '-' or '+', check the next character * if it is not numeric, *END* + continue with character e. * else, continue until hitting something non-numeric * if it is numeric, continue until hitting something non-numeric.

Zero Case

NOTE: Zero Case is not a Token.
EX: "0" is zero case. * Look at the character after 0. * if it's anything that's non-numeric, not the letter x, the number `0`, or not a period. Then it is the *Decimal* `0`
EX: "0ABC" is *Decimal 0* + *Word ABC* * if it's a number less than 8, *Octal case*. * if it's the letter x, *X case* * if it's a period, check the following character: * if it is numeric, *Float case* * else, *Decimal '0'* + *Special Character '.'*

X Case

NOTE: X Case is not a Token.

EX: "0x" is X case. * If then the following character is 0-9 || a-f, then *Hex case*. * If it's alphabetic and NOT a-f then it's *Decimal '0'* + *Word case*. * If special character, *Decimal '0'* + *Word case 'x'* + *Special Character case*.

Octal Case

EX: "0456" is Octal case. * If anything non-numeric or greater than 7, *END*.

NOTE: There is no octal quantity "0" with this implementation. "00" = *Decimal '0'* + *Decimal '0'*

Hex Case

EX: "0x45AF" is Hex case. * If anything non-alphanumeric or letters greater than f, *END*.

Comment Case

EX: "//1234" is Comment Case. "*/1234" is also Comment case. * if comment started with '//', skip all following characters until a '\n' is reached. * if comment started with '/*', skip all following characters until a '*/' is reached.

Quote Case

EX: "ABCD"1n89YS&hd1u2b8dg8"123" has a Quote case in it * if started with a single quote, all following characters until a single quote is reached. Call the token a string. * if started with a double quote, all following characters until a double quote is reached. Call the token a string.

Special Character Case

List of *Special* Special characters: * '/' Forward Slash
This depends on the following character: * if '=', then '/=' DivideEquals * if '/', then *Comment case* * if Asterisk, then *Comment case* * else, Forward slash

- '\' Back Slash This depends on the following character:
- is whitespace if the following character is a 't', 'v', 'f', 'n', 'r'
- else, Back Slash
- ''' Single Quotation Mark
- '"' Double Quotation Mark
- if either of these, *Quote case*

List of Special characters: * '(' Left Parenthesis * ')' Right Parenthesis * '[' Left Bracket * ']' Right Bracket * '.' Period * '->' Structure Pointer * '*' *Asterisk* * '=' TimesEquals * '&' Ampersand * '&&' Logical AND * '&=' BinaryANDEquals * '!' Negate or Exclamation Point * '~' One's Complement or Tilda * '+' Plus * '++' Increment * '+=" PlusEquals * '-' Dash * '--' Decrement * '-=' MinusEquals * '%' Percent * '%=' ModuloEquals * '>' Greater than * '>>' Shift right * '>>=' RightShiftEquals * '>=' Greater or equal * '<' Less than * '<<' Shift left * '<<=' LeftShiftEquals * '<=' Less or equal * '=' Equal

* '==' Equals * '!=' Not equals * '^' Bitwise Exclusive OR * '^=' BinaryExclusiveOREquals * '|' Bitwise OR * '||' Logical OR * '|=' BinaryOREquals * '?' Question Mark * ',' Comma * '`' Grave Accent * '@' At * '#' Hashtag or Pound * '$' Dollar Sign * '_' Underscore * '{' Left Curly Bracket * '}' Right Curly Bracket * '\' Back Slash * ':' Colon * ';' Semicolon

Bad Token Case

- If the ASCII character is unrecognized, *Bad Token*

List of C Keywords

auto break case char const continue default do double else enum extern float for goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while

Test cases

Simple word case

```
-bash-4.1$ ./a.out "hello how are you"
Word "hello"
Word "how"
Word "are"
Word "you"
```

Different delimeters

```
-bash-4.1$ ./a.out "hello#how?are%you"
Word "hello"
Hashtag "#"
Word "how"
Question Mark "?"
Word "are"
Percent "%"
Word "you"
```

Number then word

```
-bash-4.1$ ./a.out "123hello"
Decimal Number "123"
Word "hello"
```

Hex then word

```
-bash-4.1$ ./a.out "0x1342hello"
Hexadecimal Number "0x1342"
Word "hello"
```

Octal then word

```
-bash-4.1$ ./a.out "0234asdflksjfd"
Octal Number "0234"
Word "asdflksjfd"
```

Octal then decimal

```
-bash-4.1$ ./a.out "012899"
Octal Number "012"
Decimal Number "899"
```

Negative number and words

```
-bash-4.1$ ./a.out "-3hello  hi"
Decimal Number "-3"
Word "hello"
Word "hi"
```

Negative floats and numbers

```
-bash-4.1$ ./a.out "-3.14hello  hi"
float "-3.14"
Word "hello"
Word "hi"
```

Multiple decimal points

```
-bash-4.1$ ./a.out "-3.14.15.16"
float "-3.14"
float ".15"
float ".16"
```

Words then float

```
-bash-4.1$ ./a.out "zero point five 0.5"
Word "zero"
Word "point"
```

```
Word "five"
float "0.5"
```

C Keywords

```
-bash-4.1$ ./a.out "some C keywords char float"
Word "some"
Word "C"
Word "keywords"
C Keyword "char"
C Keyword "float"
```

Ignoring block comments

```
-bash-4.1$ ./a.out "oh theres no comment here /*shhh Im invisible*/. None at all"
Word "oh"
Word "theres"
Word "no"
Word "comment"
Word "here"
Period "."
Word "None"
Word "at"
Word "all"
```

Ignoring line comments

```
-bash-4.1$ ./a.out "some C keywords char//this should not show up\n float"
Word "some"
Word "C"
Word "keywords"
C Keyword "char"
C Keyword "float"
```

Identifying strings with apostrophes

```
-bash-4.1$ ./a.out "this is a string with apostrophe: 'with apostrophes'"
Word "this"
Word "is"
Word "a"
Word "string"
Word "with"
Word "apostrophe"
Colon ":"
String "'with apostrophes'"
```

Identifying strings with quotations

```
-bash-4.1$ ./a.out "this is a string with quotations: \"with quotations\""
Word "this"
Word "is"
Word "a"
Word "string"
Word "with"
Word "quotations"
Colon ":"
String ""with quotations""
```