Performing endPre on a string of characters

Consider an arithmetic expression consisting of binary operators (+, -, *, /) and operands. Assume that each operand is representable as a single alphabetic character, i.e., a, b, c, etc. (For simplicity, unary operators are not allowed.) The objective is to determine whether or not a given string of operators and operands constitutes a valid prefix expression. To that end, assume that there exists a class of prefix expressions having a single private data member strExp that is a string of characters.

The author presents a function, called endPre, which is central in this discussion. It takes a single nonnegative integer first as the argument and returns the index last of the character in the given expression such that the substring of strExp starting at first and ending at last constitutes a valid prefix (sub)expression, where 0 ≤ first ≤ last ≤ strExp.length() – 1. In case no such prefix (sub)expression exists, the function returns -1.

Example: Consider the string +/–abc*de that is of length nine. The algorithm builds a table that presents the value of endPre(first) for 0 ≤ first ≤ 8.

<table>
<thead>
<tr>
<th>first:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>strExp(first):</td>
<td>+</td>
<td>/</td>
<td>–</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>*</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>endPre(first):</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

That endPre(2) = 4 means there is a substring that starts at index 2, ends at index 4 and constitutes a valid prefix sub-expression. Observe that if strExp(first) is an operand (that is an alphabetic character in the present discussion), then endPre(first) = first, otherwise endPre(first) ≥ first + 2 in a valid prefix expression.

A pseudocode of the algorithm follows.
int endPre(int first)
// This function works relative to a string strExp whose
// individual characters are indexed from 0 thru strExp.length() - 1.
// Returns the index of the last character that starts at index first.
// Returns -1 if no such prefix sub-expression exists.
{
    last = strExp.length() - 1;
    if(first < 0 or first > last)
        return -1; // no prefix sub-expression exists
    ch = character at position first of strExp;
    if(ch is an alphabetic character)
        return first; // in this case, endPre(first) = first.
    // In what follows, ch is an operator.
    // Recursively find the end of the prefix sub-expression
    // starting at first+1.
    firstEnd = endPre(first+1); // recursive call
    if(firstEnd > -1)
        return endPre(firstEnd+1); // next recursive call
    else
        return -1;
} // end of endPre

Let us perform this algorithm on the example string strExp = +/-abc*de.

endPre(0):
first = 0;
last = strExp.length() - 1 = 9 - 1 = 8.
ch = ‘+’;
firstEnd = endPre(1) // recursive call
endPre(0) = endPre(firstEnd + 1) // recursive call

Note that this function call entails two recursive calls, the first of which is endPre(1).

endPre(1):
first = 1;
ch = ‘/’; // ch is an operator.
firstEnd = endPre(2) // recursive call
endPre(1) = endPre(firstEnd + 1) // recursive call

endPre(2):
first = 2;
ch = ‘-’; // ch is an operator.
firstEnd = endPre(3) // recursive call
endPre(2) = endPre(firstEnd + 1) // recursive call

endPre(3):
first = 3;
ch = ‘a’; // ch is an operand.
endPre(3) = 3.
At this point, \texttt{firstEnd} of \texttt{endPre(2)} receives the value \texttt{endPre(3) = 3}, and \texttt{endPre(4)} is activated.

\begin{verbatim}
endPre(4):
    first = 4;
    ch = 'b'; // ch is an operand.
    endPre(4) = 4.
\end{verbatim}

At this point, \texttt{endPre(2)} receives the value \texttt{endPre(4) = 4}, and control returns to \texttt{endPre(1)} where \texttt{firstEnd} of \texttt{endPre(1)} receives the value \texttt{4}, and \texttt{endPre(5)} is activated.

\begin{verbatim}
endPre(5):
    first = 5;
    ch = 'c'; // ch is an operand.
    endPre(5) = 5.
\end{verbatim}

At this point, \texttt{endPre(1)} receives the value \texttt{endPre(5) = 5}; in the process, control returns to \texttt{endPre(0)} where \texttt{firstEnd} gets the value \texttt{5} and \texttt{endPre(6)} is activated.

\begin{verbatim}
endPre(6):
    first = 6;
    ch = '*' ; // ch is an operator.
    firstEnd = endPre(7) // recursive call
    endPre(6) = endPre(firstEnd + 1) // recursive call
\end{verbatim}

\begin{verbatim}
endPre(7):
    first = 7;
    ch = 'd'; // ch is an operand.
    endPre(7) = 7.
\end{verbatim}

At this point, \texttt{firstEnd} of \texttt{endPre(6)} receives the value \texttt{endPre(7) = 7}, and \texttt{endPre(8)} is activated.

\begin{verbatim}
endPre(8):
    first = 8;
    ch = 'e'; // ch is an operand.
    endPre(8) = 8.
\end{verbatim}

At this point, \texttt{endPre(6)} receives the value \texttt{endPre(8) = 8}, and control returns to \texttt{endPre(0)} where \texttt{endPre(0)} also receives the value \texttt{8}, leading to the successful termination of the entire procedure.