AssessedExerciseTemplate-level8-Copy1

November 24, 2020

1 Short Assessed Exercise

2 Level 8

2.1 Ifthy Fairoze

2.2 19 November 2020

2.3 Version 1

2.4 Summary of the Question

Write a program that recursively parses expressions, input as strings, from the following recursively defined language and calculates and prints out the answer to the calculations.

```
<EXP> = + <DIGIT> <EXP> | - <DIGIT> <EXP> | &<EXP> | <DIGIT>
<DIGIT> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Here EXP stands for expressions, + means do addition, - means subtract and & means calculate the sum up to the given number. Legal expressions in this language involve putting the operator before its arguments (this is called Polish notation). Instead of writing 1+2, in this language you write +12. Notice only single digits are allowed and spaces are not allowed. Further legal expressions can be seen below. An example run of the program:

```
Please input the expression 3
The answer is 3
```

Another example run:

```
Please input the expression &3
The answer is 6
```

Another example run:

```
Please input the expression &+23
The answer is 15
```

Another example run:

```
Please input the expression &+1-82
The answer is 28
```

2.5 The literate program development

Building on the what I have learned in from the ECS401U Jupyter Notebook, to answer the question, I created seven methods of inputCharacterStream, CheckNextCharacter, readPastNextCharacter, nextCharacter, evalDIGITS, evalEXP and inputExpression. For each method, I will explain about the method, what is does and how I tested it and provide the code of the method itself. Next, I show how to run the solution to the question. Each code fragment can be run in Jupyter notebooks, here. Finally, I show the full program that can only be run outside of notebooks, along with instructions on how to do this and a summary of the structure of the full program.

2.5.1 inputCharacterStream

What it does This is a primitive method of ADT CharacterStream which asks the user to input an expression to parse. The user has to then give a expression (in Polish notation) and then stores that response in a record with the position of the front of that sequence of text response and returns the record as a result.

Implementation (how it works) This method first needs to create a new abstract data type called CharacterStream as shown in seperate class with fields inputToParse (string type) and position (integer type). The method is a CharacterStream return type. The parameter is a string type which is the message given to the user to ask them to input an expression (given as an argument when calling this method). First it sets up a local scanner to the method, linked to the keyboard. This makes avaliable the method scanner.nextLine(). Then a local record is created called textRecord of the type CharacterStream. Then the message is printed out to the user. Then scanner.nextLine allows the user to enter a response (taking a whole string up until the user hits enter) and this is read and put in record textRecord in the inputToParse field. Then position field of that record is set to 0 as it is the front of the character sequence of the expression given. Then it returns that record as a result of the method.

[2]:
```
/* This is a new abstract type which turns input text to a stream. This is put␣
↪into a record for the input
text to be parsed into a string type field and also position field which
tracks the position of the stream/sequence of text. */
class CharacterStream
{
    String inputToParse;
    int position;
}

/* ADT CharacterStream primitive method: which inputs text and is stored in␣
↪record where the
sequence of text is in a text field and the position field is set to 0
as it is the front of the sequence of text character. Also prints the message␣
↪to user to input expression before
user gives inputs response. */
public static CharacterStream inputCharacterStream(String message)
{
    Scanner scanner = new Scanner(System.in);
    CharacterStream textRecord = new CharacterStream();
    System.out.println(message);
    textRecord.inputToParse = scanner.nextLine();
    textRecord.position = 0;
    return textRecord;
} // END inputCharacterStream
```

Testing

[3]:
```
CharacterStream record = inputCharacterStream("Please input an expression:");
//Stores input and position in record.
```

```
Please input an expression:
+23
```

2.5.2 CheckNextCharacter

What it does This ADT CharacterStream primitive method checks the next character in the sequence of text from the inputted expression (from inputToParse field in record) but does not move position, so position field stays the same. It returns that character checked as a result.

Implementation (how it works) This primitive method first gives a parameter of the record text which will have the text that needs to be parsed in the inputToParse field and the position of the text in the position field (This record will be given as an argument when calling this method). First variable character is declared as a char type. Then the variable expression is declared and intialised by getting a copy of the string of the expression to parse from the inputToParse field of the record text. Then an if statement is used where the test asks if the position value from the field position of record text is greater than or equal to the length of the value in the expression variable. If the test is true then character variable gets a single character value "@". Otherwise if the test is false then character variable gets the char value of expression at position of the value in position field of record text. This is done using built in method charAt(). Then the value in the character variable is returned as a result of the method.

[4]:
```
/* ADT CharacterStream primitive method:
Checks the next character in the sequence of text, but does not move position.␣
 ↪*/
public static char CheckNextCharacter(CharacterStream text)
{
    char character;
    String expression = text.inputToParse;
    if (text.position >= expression.length())
    {
        character = '@';
    }
    else
    {
        character = expression.charAt(text.position);
    }
    return character;
} // END CheckNextCharacter
```

Testing

[5]:
```
char characterText = CheckNextCharacter(record);
System.out.println(characterText);
//This should print first character of expression inputted in first test stored␣
↪in the record called record.
```

+

2.5.3 readPastNextCharacter

What it does This primitive method of ADT CharacterStream moves the front of the sequence of text (text meaning expression inputted) forward so position moves up one.

Implementation (how it works) This method first gets the record text which is given as a parameter which has the text to parse from the inputToParse field and position from the position field. This record is passed through as an argument from calling this method. Then it increments the value in the position field from the text record. This method has no return type so it returns back to continue code from where this was called.

[6]:
```
// ADT CharacterStream primitive method: This moves the front of sequence of␣
↪text forward.
public static void readPastNextCharacter(CharacterStream text)
{
    text.position = text.position + 1;
    return;
} // END readPastNextCharacter
```

Testing

[7]:
```
readPastNextCharacter(record);
int order = record.position;
System.out.println(order);
// This should increment the position of the text from the position field from␣
↪the record. From 0 to 1.
```

```
1
```

2.5.4 nextCharacter

What it does This primitive method of ADT CharacterStream gets a character from the front of the sequence of text (text of expression inputted) and moves the front of the sequence of text forward and returns the character front of the sequence of text as a result.

Implementation (how it works) This method first gives the parameter which is the record text that contains the text to parse and the position of that sequence of text from the fields inputToParse and position. This record is passed through from an argument when calling this method. Then it gets the next character in that text from calling the method CheckNextCharacter (with the record text as the argument) and stores it in variable character of type char. Then the position of the text is incremented by calling the method readPastNextCharacter (with the record text as the argument). Then as a result of the method the value in variable character is returned as a result of the method.

[8]:
```
/* ADT CharacterStream primitive method: This gets a character from the front␣
 ↪of the sequence of text and
moves the front of sequence of text forward. */
public static char nextCharacter(CharacterStream text)
{
    char character = CheckNextCharacter(text);
    readPastNextCharacter(text);
    return character;
} // END nextCharacter
```

Testing

[9]:
```
char NextCharacterText = nextCharacter(record);
System.out.println(NextCharacterText); //Print next character after position 0.
```

```
2
```

2.5.5 evalDIGITS

What it does This method goes over the full expression inputted and evaluates if it is a digit and gives the integer of that digit as a result of the method.

Implementation (how it works) First parameter is given which gives access to the record text which contains the text to parse and position of that text from the fields. This record data needed is passed through by calling this method and getting putting the record as the argument. Then the answer variable is declared and initialised where it is an integer type and holds the value -2 intially. Then it declares variable next_character as a char type and it holds the next character in the expression inputted by calling the method nextCharacter (with record text as the argument). Then an if-then-else staircase is used. Where the tests are: if the value in next_character is equal to a number from 0 to 9. If one of the tests are true then answer will get that number and hold it as an integer type. Otherwise if all the tests are false then it will print that the character is not an expression and end the whole program. Then the value in variable answer is returned as a result of the method.

[10]:
```
/* This is the digits rule of the recursively defined language. Where only␣
 ↪single digits are allowed (from 0-9)
This evaluate the character in the expression inputted if it is a digit and␣
 ↪gives the integer of that digit.
Rule : <DIGITS> = 0|1|2|3|4|5|6|7|8|9 */
```

```
public static int evalDIGITS(CharacterStream text)
{
    int answer = -2;
    char next_character = nextCharacter(text);
    if (next_character == '0') answer = 0;
    else if (next_character == '1') answer = 1;
    else if (next_character == '2') answer = 2;
    else if (next_character == '3') answer = 3;
    else if (next_character == '4') answer = 4;
    else if (next_character == '5') answer = 5;
    else if (next_character == '6') answer = 6;
    else if (next_character == '7') answer = 7;
    else if (next_character == '8') answer = 8;
    else if (next_character == '9') answer = 9;
    else
    {
        System.out.println("This is not an expression.");
        System.exit(0);
    }
    return answer;
} // END evalDIGITS
```

Testing

[11]:
```
int answer = evalDIGITS(record);
System.out.println(answer); // Should print next character digit 3 (next␣
↪position after digit 2).
```

```
3
```

2.5.6 evalEXP

What it does This method goes over the full expression inputted (in Polish notation) and evaluates if it is addition, subtraction or to calculate sum up to a given number or digit itself and it would calculate the answer following the recursively defined language and return the answer as a result.

Implementation (how it works) First this method gives access to the record text which contains the text to parse and the position of that text from the fields where it is held. This record data is passed through via argument when calling this method. Then answer variable is declared as integer. The variable next_character of type char holds the next character of the text to parse (from record text) which is the first character. The position is unchanged. This is done by calling the method CheckNextCharacter. Then it uses if-then-else statements to test the value in variable next_character. So the first test asks if the value in next_character is equal to the addition symbol and if this is true then it reads past the addition symbol by calling method readPastNextCharacter and then gets the next character which is a digit and gets the integer of it and stores it in variable dig. This is done by calling method evalDIGITS. Then it gets the next character which is an expression and gets the integer value of that expression and stores it into variable exp. This is a recursive problem done by doing recursion by calling the evalEXP method (in the same method). Then the two values in variables dig and exp are added together and the result is stored in variable answer. The second test asks if the value in variable next_character is equal to the subtraction symbol and if this is true then it reads past the subtraction symbol by calling method readPast-NextCharacter. Then it get the next character which is a digit and gets the integer of it and stores it in variable dig. This is done by calling method evalDIGITS. Then it gets the next character which is an expression and gets the integer value of that expression and stores it into variable exp. This is a recursive problem which is solved by doing recursion by calling the evalEXP method (in the same method). The two values in variables dig and exp are subtracted and the result is stored in variable answer. The third test asks if the value in the variable next_character is equal to the & symbol (which means calculate the sum up to the given number). If this is true then it reads past the & symbol in the inputted expression by calling method readPastNextCharacter. Then it get the next character which is an expression and gets the integer value of it and stores it in variable exp. This is a recursive problem solved by doing recursion which involves calling method evalEXP (in the same method). Then is gets the answer by doing the calulation up to that given number (value in variable exp) which involves a formula to do it. It then stores the answer from the calculation in variable answer. Otherwise, if all the test are false in this if-then-else staircase it then gets the next character which is a digit (as the next character cannot be a symbol) and gets the integer value of it and stores it into answer variable. At the end it returns the value in answer as a result of this method.

[12]:
```
/* This is the expression rule of the recursively defined language.
Rule: <EXP> = + <DIGIT> <EXP> | - <DIGIT> <EXP> | &<EXP> | <DIGIT>
Where + is addition and - is subtraction and & is to calculate sum up to a␣
 ↪given number.
Since input is given polish notation so operator comes first and then operands␣
 ↪and calculates answer. */
public static int evalEXP(CharacterStream text)
{
    int answer;
    char next_character = CheckNextCharacter(text);
    if (next_character == '+')
    {
        readPastNextCharacter(text);
        int dig = evalDIGITS(text);
        int exp = evalEXP(text);
        answer = dig + exp;
    }
    else if (next_character == '-')
    {
        readPastNextCharacter(text);
        int dig = evalDIGITS(text);
        int exp = evalEXP(text);
        answer = dig - exp;
    }
    else if (next_character == '&')
    {
        readPastNextCharacter(text);
        int exp = evalEXP(text);
        answer = (exp * (exp + 1)) / 2;
    }
    else
    {
        answer = evalDIGITS(text);
    }
    return answer;
} // END evalEXP
```

Testing

[13]:
```
CharacterStream record2 = inputCharacterStream("Please enter an expression:");
int answer = evalEXP(record2);
System.out.println(answer);
//Expression given like +23 (of Polish Notation) should give answer of 5 (since␣
 ↪2+3=5).
```

```
Please enter an expression:
+23
5
```

2.5.7 inputExpression

What it does This method does all the work by calling the necessary methods to asks the user to input an expression and gets the user's inputted expression and get the answer and print it to the user and end the program after that.

Implementation (how it works) This method is the method that is called in the main method. First it gets the input of the expression from the user by calling the method inputCharacterStream which gives an argument that is a string message asking the user to input the expression. The response from the user which is the inputted expression is stored in variable input of abstract type CharacterStream. It then gets the answer of that expression inputted by calling method evalEXP with the input variable which holds the expression inputted as the argument. Then the answer is printed out to the user. Then the program is ended or terminated.

[15]:
```
/* This is the method doing all the work. This method tells the user to input␣
 ↪an expression to parse
and the user responds by inputting an expression and calculates and prints the␣
 ↪answer of that expression. */
public static void inputExpression()
{
    CharacterStream input = inputCharacterStream("Please input the expression");
    int answer = evalEXP(input);
    System.out.println("The answer is " + answer);
    System.exit(0);
    return;
} // END inputExpression
```

Testing

[ ]:
```
inputExpression();
```

```
Please input the expression
3
The answer is 3
```

2.5.8 Running the program

Run the following call to simulate running the complete program.

[ ]:
```
inputExpression();
```

```
Please input the expression
+23
The answer is 5
```

2.6 The complete program

This version will only compile here.

[1]:
```
// NAME: Ifthy Fairoze
// DATE: 19/11/2020
// VERSION: 1
/* BRIEF OVERVIEW OF PURPOSE: This program is a recursive parsing calculator␣
 ↪which parses expressions inputted
via Polish notation following the recursively defined language and calculates␣
 ↪and prints outs the answer. */

import java.util.Scanner; // Needed to make Scanner available

/* This is a new abstract type which turns input text to a stream. This is put␣
 ↪into a record for the input
text to be parsed into a string type field and also position field which
tracks the position of the stream/sequence of text. */
class CharacterStream
{
    String inputToParse;
    int position;
}

class Parsing_Calculator
{
    public static void main (String [] a)
    {
        inputExpression();
        System.exit(0);
    }

    /* ADT CharacterStream primitive method: which inputs text and is stored in␣
↪record where the
    sequence of text is in a text field and the position field is set to 0
    as it is the front of the sequence of text character.
    Also prints the message to user to input expression before user gives␣
↪inputs response. */
    public static CharacterStream inputCharacterStream(String message)
    {
        Scanner scanner = new Scanner(System.in);
        CharacterStream textRecord = new CharacterStream();
        System.out.println(message);
        textRecord.inputToParse = scanner.nextLine();
        textRecord.position = 0;
        return textRecord;
    } // END inputCharacterStream

    /* ADT CharacterStream primitive method:
    Checks the next character in the sequence of text, but does not move␣
↪position. */
    public static char CheckNextCharacter(CharacterStream text)
    {
        char character;
        String expression = text.inputToParse;
        if (text.position >= expression.length())
        {
            character = '@';
        }
        else
        {
            character = expression.charAt(text.position);
        }
        return character;
    } // END CheckNextCharacter

    // ADT CharacterStream primitive method: This moves the front of sequence␣
↪of text forward.
    public static void readPastNextCharacter(CharacterStream text)
    {
        text.position = text.position + 1;
        return;
    } // END readPastNextCharacter
    /* ADT CharacterStream primitive method: This gets a character from the␣
↪front of the sequence of text and
    moves the front of sequence of text forward. */
    public static char nextCharacter(CharacterStream text)
    {
        char character = CheckNextCharacter(text);
        readPastNextCharacter(text);
        return character;
    } // END nextCharacter

    /* This is the digits rule of the recursively defined language. Where only␣
↪single digits are allowed (from 0-9)
    This evaluate the character in the expression inputted if it is a digit and␣
↪gives the integer of that digit.
    Rule : <DIGITS> = 0|1|2|3|4|5|6|7|8|9 */
    public static int evalDIGITS(CharacterStream text)
    {
        int answer = -2;
        char next_character = nextCharacter(text);
        if (next_character == '0') answer = 0;
        else if (next_character == '1') answer = 1;
        else if (next_character == '2') answer = 2;
        else if (next_character == '3') answer = 3;
        else if (next_character == '4') answer = 4;
        else if (next_character == '5') answer = 5;
        else if (next_character == '6') answer = 6;
        else if (next_character == '7') answer = 7;
        else if (next_character == '8') answer = 8;
        else if (next_character == '9') answer = 9;
        else
        {
            System.out.println("This is not an expression.");
            System.exit(0);
        }
        return answer;
    } // END evalDIGITS

    /* This is the expression rule of the recursively defined language.
    Rule: <EXP> = + <DIGIT> <EXP> | - <DIGIT> <EXP> | &<EXP> | <DIGIT>
    Where + is addition and - is subtraction and & is to calculate sum up to a␣
↪given number.
    Since input is given polish notation so operator comes first and then␣
↪operands and calculates answer. */
    public static int evalEXP(CharacterStream text)
    {
        int answer;
        char next_character = CheckNextCharacter(text);
        if (next_character == '+')
        {
            readPastNextCharacter(text);
            int dig = evalDIGITS(text);
            int exp = evalEXP(text);
            answer = dig + exp;
        }
        else if (next_character == '-')
        {
            readPastNextCharacter(text);
            int dig = evalDIGITS(text);
            int exp = evalEXP(text);
            answer = dig - exp;
        }
        else if (next_character == '&')
        {
            readPastNextCharacter(text);
            int exp = evalEXP(text);
            answer = (exp * (exp + 1)) / 2;
        }
        else
        {
            answer = evalDIGITS(text);
        }
        return answer;
    } // END evalEXP

    /* This is the method doing all the work. This method tells the user to␣
↪input an expression to parse
    and the user responds by inputting an expression and calculates and prints␣
↪the answer of that expression. */
    public static void inputExpression()
    {
        CharacterStream input = inputCharacterStream("Please input the␣
↪expression");
        int answer = evalEXP(input);
        System.out.println("The answer is " + answer);
        System.exit(0);
        return;
    } // END inputExpression
}
```

END OF LITERATE DOCUMENT