CS3423 - Compilers II

FILEO

A language for file operations

Wednesday 11th October, 2023
Team Number: 10

Name	Roll Number
K Vivek Kumar	CS21BTECH11026
Bhende Adarsh Suresh	CS21BTECH11008
Jarupula Sai Kumar	CS21BTECH11023
Pathlavath Shankar	CS21BTECH11064

Contents

1 Introduction

This document introduces our project, which focuses on developing a Domain Specific Language (DSL) and overview of its associated compiler as part of the final course project. The DSL is designed to address specific needs within a particular domain or problem space. The compiler is the core component responsible for translating DSL code into executable instructions.

2 Motivation

In our search for a new Domain-Specific Language (DSL), we aimed to find an innovative solution to a recurring problem.

2.1 Problem Identification

Throughout five semesters of our academic journey, we noticed the challenge of submitting assignments in a specific folder structure that's easy for computer evaluation. To address this, we want to create a language that efficiently generates the required folder structures. This will benefit all students, ensuring uniformity in structure with a single shared code.

2.2 Expanding the Idea

Recognizing the need for such a DSL, we decided to add more file operations for user convenience. We also plan to handle file creation differences in Linux and Windows, providing a single program for both systems for easy use and standardization.

2.3 Goals

Our DSL program aims to offer the following capabilities:

1. Efficiently create folder structures.
2. Perform various file operations, such as finding, renaming, replacing, emptying, and deleting.
3. Provide a straightforward and easy-to-understand language syntax.

3 Data types

Data types can be broadly categorized into two main groups: primitive data types and non-primitive data types. We are using only primitive data type.

Table 1: Data type Table

Types	Type of Datatype
string	primitive
number	primitive
FILE	primitive
FOLDER	primitive

3.1 Data Type Selection

In the design and development of our domain-specific language (FILEO Language) for file operations, we have carefully selected specific data types to represent different aspects of our language's domain. The choice of these data types is driven by the functional requirements. Below, we provide justification for the selection of each data type:

3.1.1 String

String datatypes are used for file names in our FILEO Language

3.1.2 Number

The "number" data type is for return type of various operations.

3.1.3 FILE

File Handling: The "FILE" data type is selected to represent files references within our FILEO Language. This is crucial for the core functionality of reading, writing, and manipulating files in the FILEO Language designed for file operations.

3.1.4 FOLDER

The "FOLDER" data type is chosen to facilitate directory and folder management.

4 Operations

Table 2: Operations

Special keywords	Action
find	find the word
replace	replace all the word1 with word2
rename	renames the file
empty	empty the file content
write	the content of string in the file
copy	copy the content of file
delete	delete the file

4.1 `.find("word to find"):`

This command is used to search for a specified word within the file's content.

4.2 `.replace("word1","word2"):`

This command replaces all instances of "word1" with "word2" throughout the file's content.

4.3 `.rename("newname"):`

The .rename method is used to change the name of the file.

4.4 `.empty():`

This method is employed to remove all content from the file, effectively emptying it.

4.5 `.write(string):`

This method to write new content into the file.

4.6 `.copy content(file):`

This method facilitates copying the content of another file into the current file.

4.7 `.delete()`:

This method is employed to permanently remove the file.

5 Statements

Within the DSL, numerous statements play essential roles in its operations. In the following subsections, we introduce each of these statements and their specific functions.

5.1 Comments

Comments play a vital role in any programming language, offering a valuable way to provide program visitors with an overview of the program's purpose and functionality. In our DSL, we've incorporated the concept of comments, allowing for clear program documentation enclosed within double dollar signs ($$). A sample program snippet is presented below:

5.1.1 Single-Line Comment

```
...

$$ This is a single-line comment. $$

...
```

5.1.2 Multi-Line Comment

```
...

$$
This is a multi-line comment.
It can span across multiple lines.
$$

...
```

5.2 Folder Structure

We have a seperate section in the code which just involves creating the folder structure. This is the first phase of working which involves in creating the folder structure easily and efficiently.

1. FOLDER is given with a colon (:) at the end.
2. FILE is given without a colon (:) at the end.

The hierarchy of folder and files are distinguished with tab spaces (`\t`). One sample program is given below:

```
CS21BTECH11026:
    TP1:
        T1:
            1.txt
            2.txt
            3.txt
            C1:
                overview.pdf
        P1:
            source_program.c
```

```
O1:
    output_1.txt
    output_2.txt
    output_3.txt
```

5.3 Declaration Statements

Declaration statements involves introducing variables of various datatypes. This can include one of the following:

1. Declaring number constants

   ```
   $$ This declares a variable which can hold integer constant. $$
   number <identifier>;
   number i;
   ```

2. Declaring string constants

   ```
   $$ This declares a variable which can hold strings and character constant. $$
   string <identifier>;
   string name;
   ```

3. Declaring Files

   ```
   $$ This declares a variable which can hold files. $$
   FILE <identifier>;
   FILE f1;
   ```

4. Declaring Folders

   ```
   $$ This declares a variable which can hold folder. $$
   FOLDER <identifier>;
   FOLDER f2;
   ```

5.4 Main Function

In the second part of the language program, we are implementing the operations on various files, hence there should be a start statement for this. We have declared a function known as `svas`, which shall be translated as the `main` function as in C-language.

For example:

```
$$ This is section 1. $$
Folder:
    File1
    Folder1:
        File2
        Folder2:
    Folder3:
        File3
%%
$$ This is section 2. $$
function svas = {
    number a;
    FILE vivek;
    vivek = open("1.txt");
    a = vivek.find("Vivek");
}
```

5.5 Functions

5.5.1 Function Declaration

```
function <identifier> = <location_of_file>{
    ...
    <various operations>;
    ...
}
```

5.5.2 Function Call

```
fun function_name;
```

5.5.3 Example Code

```
CS21BTECH11026:
    TPY:
        1.txt
        2.txt
    TY:
       1.txt
%%
function myfunction = CS21BTECH11026/TY/{
    FILE myfile = open("1.txt");
    number k = myfile.find("word1");
    number i = myfile.replace("word1", "word2");
}

function svas = {
  fun myfunction;
}
```

6 Lexical Stage

6.1 Reserved Keywords

Table 3: Reserved Keywords in FILEO Language

function
fun
open
find
replace
delete
rename
copy_content
empty
write
svas

6.2 Punctuations

Table 4: Punctuations in FILEO Language

,
;
:
.
=
(
)
{
}
"
/
$

6.3 Identifiers

An identifier is a sequence of characters that can consist of a combination of letters (alphabets) and numbers. The identifier must begin with a letter (alphabet). It is not allowed for the identifier to be composed solely of numbers.

7 Syntax

7.1 Sample Programs

1. **Sample Program 1**

```
Basefolder:
    Folder1:
        1.txt
        2.txt
        3.txt
    Folder2:
        Folder3:
            output.txt
        source_program.c
%%
function check = Basefolder/Folder2/Folder3{
    FILE ouput;
    output = open("output.txt");
    number i = output.find("1\n2");
}
function svas = {
    fun check;
}
```

2. **Sample Program 2**

```
COMPILERS2:
    CS21BTECH11008.txt
```

```
    CS21BTECH11023.txt
    CS21BTECH11026.txt
    CS21BTECH11064.txt
CS21BTECH11026:
    TPY:
        XYZ.txt:
        Y:
            $$ This is a new folder $$
        TY:
            1.txt
            2.pdf
        T1:
            4.doc
            e.s
    TY:
        1.py
        TPY:
            folder.c
        T1:
            1.c
%%
FILE a;
$$ Ymldffhfhdfdf.g,dfg $$
function vivek = CS21BTECH11026/TY/{
    FILE vivek = open("1.txt");
    number k = vivek.find("Vivek");
    number i = vivek.replace("Vivek", "Kumar");
    number i = vivek.replace("Vivek", "Kumar", 2);
    number i = vivek.delete();
    number i = vivek.rename("2.pdf");
    string s = "Vivek\nKumar";
    vivek.copy_content(a);
    vivek.empty();
    vivek.write(s);
}
function svas = {
    fun  vivek;
}
```

7.2 Correct vs Wrong

From our understanding and constraints from our language design, we are mentioning some of the ambiguities which may arise.

Correct Syntax	**Wrong Syntax**
`Folder1:` `    File1`	`Folder1:` `File1`

Correct Syntax	Wrong Syntax

`Folder1:`
`  Folder2:`
`  Folder3:`
`  File2`
	`Folder1:`
	`  Folder2`
	`  Folder3`
	`  File2`

Correct Syntax	Wrong Syntax

`Folder1:`
`  File1`
`  File2`
	`Folder1:`
	`  File1, File2`

8 Advantages

The FILEO Language offers several advantages due to its unique design:

- **User-Friendly Syntax:** FILEO prioritizes a user-friendly experience, making the language easy to learn and work with.
- **Optimized Operations:** Our compiler is designed to optimize each file operation, ensuring efficient execution.

9 Challenges

Despite its advantages, FILEO may encounter certain challenges, including:

- **Efficient Folder Structure Development:** Defining folder structures with tab-spaces and file names can be challenging during development.
- **Complex Single-Line Operations:** The flexibility of performing multiple operations in a single line may pose challenges in terms of readability and usability.
- **Completeness Considerations:** Issues related to file open/close management, such as inappropriate file openings or forgetting to close files, need to be carefully addressed.

10 Conclusion

Providing an overview of the language has been a valuable exercise in understanding the compiler's inner workings and the language's constraints. We are committed to implementing as many features as possible from this document. Please note that the features outlined are subject to refinement and further development.

Correct Syntax	Wrong Syntax

Folder1:
  File1
	Folder1:
	  File1
	  File1

Correct Syntax	Wrong Syntax
... $$ Start of a multi-line comment Its end. $$ ... %% ... $$ Start of another comment End of the comment. $$ ...	... $$ Start of a multi-line comment ... %% ... End of the comment. $$ ...