Helwan University

Faculty of computers and information

Computer Science department

Spring 2021-2022

Compilers

Project #1

جامعة حلوان

Helwan University

Faculty of Computers and Information

Project #1 Language

Description:

A program in Project #1 consists of a sequence of function definitions -. Each function consists in turn of variable declarations, type declarations, function declarations, and statements. The types in Project #1 are very restricted look at table 1.in addition to table 1, single dimensional arrays and pointer types are possibly using user defined struct types as in C Language. The array index value can only be simple unary expression such as an identifier, a constant or another simple array access expression. Project# 1 Language is case sensitive.

Scanner:

Lexical Analysis:

Project #1 Scanner must recognize the following keywords and returns Return Token in table 1:

Keywords	Meaning	Return Token
If—Else	conditional statements	Condition
Iow	Integer type	Integer
SIow	Signed Integer type	SInteger
Chlo	Character Type	Character
Chain	Group of characters	String
Iowf	Float type	Float
SIowf	Signed Float type	SFloat
Worthless	Void Type	Void
Loopwhen/Iteratewhen	repeatedly execute code as long as condition is true	Loop

Turnback	**Return a value from a function**	**Return**
Stop	**Break immediately from a loop**	**Break**
Loli	**grouped list of variables placed under one name**	**Struct**
(+, -, *, /,)	**Used to add, subtract, multiply and divide respectively**	**Arithmetic Operation**
(&&, ||, ~)	**Used to and, or and not repectively**	**Logic operators**
(==, <, >, !=, <=, >=)	**Used to describe relations**	**relational operators**
=	**Used to describe Assignment operation**	**Assignment operator**
->	**Used in loli to access loli elements**	**Access Operator**
{,},[,]	**Used to group statements or array index respectively**	**Braces**
[0-9] and any combination	**Used to describe numbers**	**Constant**
“,’	**Used in defining strings and single character reprctively**	**Quotation Mark**
Include	**Used to include one file in another**	**Inclusion**

Table 1: Tokens Description

The Scanner also recognizes identifiers. An identifier is a sequence of letters and digits, starting with a letter. The underscore ‘_’ counts as a letter. For each identifier, Project#1 Scanner returns the token IDENTIFIER. Project#1 language allows many identifiers to be identified by one type separated by comma (,)

Comments in Project #1 :

Project#1 includes two types of comments single line comments are prefixed by $$$ and multiple line comment are written between /$ and $/.Your scanner must ignore all comments and white.

Include file command:

In order to facilitate the inclusion of multiple files, your Project#1 scanner is also responsible for directly handling the include file command. When encountering the include directive placing at the first column of a given line, the scanner must open the file indicated by the file name in the directive and start processing its contents. Once the included file has been processed the scanner must return to processing the original file. An included file may also include another file and so forth. If the file names does not exist in the local directory you should simply ignore the include command and proceed with the tokens in the current file.

Tokens and return values:

You must build a dictionary to save Keywords that are defined in Project #1 language.

Project#1 Language Delimiters (words and lines):

The words are delimited by Space and tab. The line delimiter is semicolon (;)and newline.

Output format:

Scanner:

In case of correct token: Line #: (Number of line) Token Text: ---------- Token Type: ----------

In case of Error tokens: Line #: (Number of line) Error in Token Text: ------ -------

Total NO of errors: ***(NO of errors found)***

Parser:

Firstly you must sate Scanner phase output as above then state Parser Phase output

In case of correct Statement: Line #: (Number of line) Matched Rule Used:--------------

In case of Error: Line #: (Number of line) Not Matched

Total NO of errors: ***(NO of errors found)***

Parser Grammar rules:

1. *program → declaration-list|comment| include_command*
2. *declaration-list → declaration-list declaration | declaration*
3. *declaration → var-declaration | fun-declaration*
4. *var-declaration → type-specifier* ***ID*** *;*
5. *type-specifier →* **Iow | SIow | Chlo | Chain | Iowf | SIowf | Worthless**
6. *fun-declaration → type-specifier* ***ID*** *(params) compound-stmt| comment type-specifier* ***ID***
7. *params → param-list |* **Worthless** *|* ***e***
8. *param-list → param-list* ***,*** *param | param*
9. *param → type-specifier* ***ID***
10. *compound-stmt → {comment local-declarations statement-list* ***}*** **|** *{local-declarations statement-list* ***}***
11. *local-declarations →local-declarations var-declaration |* ***e***
12. *statement-list → statement-list statement |* ***e***
13. *statement → expression-stmt | compound-stmt | selection-stmt | iteration-stmt | jump-stmt*
14. *expression-stmt → expression* ***;*** *|* ***;***
15. *selection-stmt →* ***if (*** *expression* ***)*** *statement*
    | ***if (*** *expression* ***)*** *statement* ***else*** *statement*
16. *iteration-stmt → Loop-statement | Iterate-statement*
17. *Loop-statement →* ***Loopwhen(*** *expression* ***)*** *statement*

18. *Iterate -statement* → **Iteratewhen (** *expression* **;** *expression* **;** *expression* **)** *statement*
19. *jump-stmt* → **Turnback** *expression* **; | *Stop* ;**
20. *expression* → *id-assign* = *expression* | *simple-expression* | *id-assign*
21. *id-assign* → ***ID***
22. *simple-expression* → *additive-expression relop additive-expression* | *additive-expression*
23. *relop* → **<= | < | > | >= | == | *!=* | && | ||**
24. *additive-expression* → *additive-expression addop term* | *term*
25. *addop* → **+ | -**
26. *term* → *term mulop factor* | *factor*
27. *mulop* → ***** | ***/***
28. *factor* → **(** *expression* **)** | *id-assign* | *call* | *num*
29. *call* → ***ID (*** *args* ***)***
30. *args* → *arg-list* | ***e***
31. *arg-list* → *arg-list* **,** *expression* | *expression*
32. *num* → *Signed num* |*Unsigned num*
33. *Unsigned num* → *value*
34. *Signed num* → *pos-num* | *neg-num*
35. *pos-num* → **+** *value*
36. *neg-num* → **-** *value*
37. *value* → ***INT_NUM*** | ***FLOAT_NUM***
38. *comment* →/* STR */ | /// STR
39. *include_command* →include (F_name.txt);
40. *F_name* →STR

Note: (e) means Ebslon

Sample Input and output:

<u>Input:</u>

1-/$This is main function $/

2-Worthless decrease(){

3-int 3num=5;

4-Loopwhen(counter<num){

```
5-reg3=reg3-1; } }
```

<u>Scanner Output:</u>

```
Line : 2 Token Text: Worthless      Token Type: Void
Line : 2 Token Text: decrease       Token Type: Identifier
Line : 2 Token Text: (                Token Type: Braces
Line : 2 Token Text: )                Token Type: Braces
Line : 2 Token Text: {                Token Type: Braces
Line : 3 Token Text: int            Token Type: Identifier
Line : 3  Error in Token Text:3num
Line : 3 Token Text: =                Token Type: Assignment operator
Line : 3 Token Text: 5                Token Type: Constant
-----------------------------Etc.
Total NO of errors: 1
```

<u>Scanner and Parser</u> <u>Output:</u>

Firstly you must sate Scanner phase output as in scanner sample input and output then state parser output based on scanner output

Line : 1 *Matched*   Rule used: *Comment*

Line : 2 *Matched*   Rule used: *fun-declaration*

Line : 3 *Not Matched*

-----------------------------Etc.

Total NO of errors: 1