**Compiler Design, Final project, Spring 2024**

**Part I.( 20 points)**

Create the following text file: “finalv1.txt”

1. program s2024;

2. //This program computes and prints the value

3. of an expression//

4. var

5. // declare variables //

6. p1 , q2s , r, pp : integer ;

7. begin

8. p1 = 3 ;

9. pp= 23 ;

10. q2s = 4 ;

11. r = 5 ;

12. p1 = 2\*p1+ r\*( q2s + pp);

13. write ( p1 ); // display p1 //

14.

15. //compute the value of this expression //

16. pp = p1 \* ( q2s + 2 \* rc) ;

17. write ( “value=”, pp ) ; //print the value of pp //

18

19. end.

Apply the following rules to this file and copy the new version in the file “final24.txt” to make it easier to read one token at a time.

1. Any line that begins with // and ends at // are considered as a comment line(s) (i.e. lines #2-3,5,13,15,17), remove them all.
2. Remove all blank line(s) (i.e. line #14, 18)
3. Extra spaces in each line must be removed, leave one space before and one after each token to make tokenization easier.

The “finalf24.txt” should look like this

1. program s2024;

2. var

3. p1 , q2s , r , pp : integer ;

4. begin

5. p1 = 3 ;

6. pp= 23 ;

7. q2s = 4 ;

8. r = 5 ;

9. p1 = 2 \* p1 + r \* ( q2s + pp ) ;

10. write( p1 ) ; // display p1

11. pp = p1 \* ( q2s + 2 \* rc ) ;

17. write( “value=”, pp ) ; //print the value of pp

18. end.

**Part II (50 points)**

Use the following grammar:

|  |  |
| --- | --- |
| <prog> | 🡪 **program** <identifier>; **var** <dec-list> **begin** <stat-list> **end.** |
| <identifier> | 🡪 <letter>{<letter>|<digit>} note. *This grammar is in EBNF* |
| <dec-list> | 🡪 <dec> : <type> ; |
| <dec> | 🡪 <identifier> , <dec> | < identifier > |
| <type> | 🡪 **integer** |
| <stat-list> | 🡪 <stat> | <stat> <stat-list> |
| <stat> | 🡪 <write> | <assign> |
| <write> | 🡪 **write**( <str> < identifier > ); |
| <str> | 🡪”value=”, | λ |
| <assign> | 🡪 < identifier > = <expr>; |
| <expr> | 🡪 <expr> + <term> | <expr> - <term> | <term> |
| <term> | 🡪 <term> \* <factor> | <term> / <factor>| <factor> |
| <factor> | 🡪 < identifier > | <number> | ( <expr> ) |
| <number> | 🡪 <sign><digit>{ <digit> } note: *the grammar is in EBNF* |
| <sign> | 🡪 + | - | λ |
| <digit> | 🡪 0|1|2|…|9 |
| <letter> | 🡪 p | q | r | s |

In which **program,** **var, begin, end. , inetger** , and **write** are reserved words

Have your program (compiler) to check the grammar of the program and display “No Error” if the given program satisfied all grammar’s rule, issue “some errors” otherwise.

**Have**  your program detect the following errors and issue a corresponding error message:

**program** is expected (if program is missing or spelled wrong)

**var**  is expected (if var is missing or spelled wrong)

**begin** is expected (if begin is missing or spelled wrong)

**end.** is expected (if end. is missing or spelled wrong)

**integer** is expected (if integer is missing or spelled wrong)

**write** is expected (if spells wrong)

unknown identifier if the variable is not defined in the declaration block

; ; is missing (if the grammar required a ; )

, is missing (if the grammar equired a , )

. is missing (if the grammar required a .)

( Left parentheses is missing

) Right parentheses is missing

**Output** : Either one of the above messages or **No Error**

**Part III ( 20 points )**

If there are no ERRORS, translate the program to a high level language (C++,C#, Python, or Java). Use the high level language compiler to run your program to display the same output. This is the C++ version of the program in part I

#include <iostream>

using namespace std;

int main()

{

int p1 , q2s , r , pp ;

p1 = 3 ;

pp= 23 ;

q2s = 4 ;

r = 5 ;

p1 = 2 \* p1 + r \* ( q2s + pp ) ;

cout<< p1 ;

pp = p1 \* ( q2s + 2 \* rc ) ;

cout<< “value=” << pp ; //print the value of pp

}

The whole program grammar consists of ONE line, and each non-terminal in the grammar will be defined over-and-over until hits a terminal.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| program | <identifier> | ; | var | <dec-list> | begin | <stat-list> | end. |

**<id>{<id><digit>} <dec> : <type> <stat> <stat-list>**

<id> <digit><digit><digit> <digit> <stat> <stat-list>

<identifier>,<dec> <stat> <stat-list>

<identifier>,<dec> <stat> <stat-list>

<identifier> ,<dec> <stat> <stat-list>

<identifier>,<identifier>

<stat>

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| program | S2024 | ; | var | p1 , q2s ,r, pp | integer | begin | p1 = 3 ;  pp = 23 ;;  c = 5 ;  …..  ….  . | end. |

**(10 points)** Turn in the following in a folder and a flash card which includes the program as well. Everything must be typed (no hand writing is accepted).

1. Cover page:

Computer Science xxx

Spring 2024

Final Project

Group Members

…………

Method Used:

Language Used:

1. The original program on page “1”
2. The original grammar on page “2”
3. The grammar in BNF form (remove all {, } , and | )

|  |  |
| --- | --- |
| The original grammar in BNF after removing all EBNF grammars. | |
| <prog> | 🡪**program** <identifier>, **var** <dec-list> begin <stat-list> **end.** |
| <identifier> | 🡪<letter> <post-identifier> |
| <post-identifier> | 🡪<letter> <post-identifier> |
| <pos-identifier> | 🡪<digit><post-identifier> |
| <post-identifier> | 🡪 |
| .  . |  |
| <letter> | p |
| <letter> | q |
| <letter> | r |
| <letter> | s |

1. If you are using predictive parsing table, remove all left recursive rules. If you are using LR parsing table, remove all lambdas. Show the final form of the grammar in BNF

|  |  |
| --- | --- |
| Final BNF grammar for predictive /LR parsing method | |
| .  . |  |
| E | 🡪E + T |
| E | 🡪E – T |
| E | 🡪T |
| . | . |
| L | 🡪q |
| L | 🡪r |
| L | 🡪s |

1. List all terminals and non-terminals

|  |  |
| --- | --- |
| **List of non-terminals** | **List of terminals** |
| |  |  | | --- | --- | | Old name | New name | | <prog> | P | | <identifiers> | I | | …… |  | | <expr> | E | | <term> | T | | ….. |  | | <digit> | D | | <letter> | L | | + - \* …….p q r s |

1. Find the members of first and follow

|  |  |  |  |
| --- | --- | --- | --- |
| Non-terminals | | FIRST | FOLLOW |
| P  I  …………… | <prog>  <identifier>  ……… | program  p q r s  …….. | $  p q r s + - / \* 0 1 2 …….9 ; , )  …….. |

1. Show the parsing table
2. Complete copy of the program including all user’s defined libraries

Copy of the sample run for:

1. The given program in part I
2. The translated program to a high-level language and its sample run.

Please, turn in all in a folder and insert your flash drive in your folder as well.