**Compiler Design**

**Final project.**

**You have option to use one of programming languages: C++, C#, Python, or Java**

**Part I.(30 points)**

Create the following text file: “finalp1.txt”

1. program f2022;

2. \*\* This program computes and prints the value

3. of a given expression \*\*

4. var

5. \*\* declare variables \*\*

6. a1 , b2a , bc, ba2 : integer ;

7. begin

8. \*\* initialize variables \*\*

9. a1 = 3 ;

10. b2a = 4 ;

11. bc = 5 ;

12. print(bc ); \*\* display bc \*\*

13.

14. \*\* compute the value of the expression \*\*

15. ba2 = a1 \* ( b2a + 2 \* bc) ;

16. print ( “value=”, ba2 ) ; \*\* print the value of ba12 \*\*

17. end.

Apply the following to this file and copy the new version in file “finalp2.txt”

1. Any line/s or part of a line that begins with \*\* and ends with \*\* is considered as a comment line (i.e. lines #2,3,5, 8, 11,13,15), remove them
2. Any blank lines must be removed (i.e. line #13)
3. Extra spaces in each line must be removed, Leave one space before and one after each token (example: line 9: a1 = 3 ; )

The “finalp2.txt” should look like this

1. program f2022 ;

2. var

3. a1 , b2a , bc , ba2 : integer ;

4. begin

5. a1 = 3 ;

6. b2a = 4 ;

7. bc = 5 ;

8. print (wc ) ;

8. ba2 = a1 \* ( b2a + 2 \* bc) ;

9. print ( “value=” , ba2 ) ;

10. end.

**Part II (50 points)**

Use the following grammar ( the part in RED color is the grammar of algebraic expression. You used this grammar for project 10 and 11 where E is <expr>, T is <term>, and F is <factor> )

|  |  |
| --- | --- |
| <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> | 🡪 print (“value”, < identifier > ); |
| <assign> | 🡪 < identifier > = <expr>; |
| <expr> | 🡪 <expr> + < expr> | <expr> - < expr> | < term> |
| <term> | 🡪 <term> \* <factor> | <term> / <factor> | <factor> |
| <factor> | 🡪 < identifier > | <number> | ( <expr> ) |
| <number> | 🡪 <sign><digit>{<digit> } note: this grammar is in EBNF |
| <sign> | 🡪 + | - | λ |
| <digit> | 🡪 0|1|2|…|9 |
| <letter> | 🡪 a | b | c | d | f |

In which **program,** **var, begin, end. , integer** , and **print** are reserved words

|  |
| --- |
| Do this part only if you want to receive “100%” for the project when your program displays the error message , otherwise your maximum score is “<85%”  to determine whether the program in part I is accepted or not. Your program should detect and produce the following error messages  **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)  unknown identifier if variable is not defined  **print** is expected ( if print spelled wrong)  **;** ; is missing  **,**  , is missing  **.**  . is missing  **(** ( is missing  **)** ) is missing |

Output : Either one of the above messages or Accept if there are no error

**Part III ( 20 )**

If there are no ERROR, translate the program to a high-level language: C++, C#, Python, or Java. Run the 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 a1 , b2a , bc, ba2 ;

a1 = 3 ;

b2a = 4 ;

wc = 5 ;

cou<<bc ;

ba2 = a1 \* (b2a + 2 \* bc ) ;

cout<< “value=” <<ba2<<endl;

return 0;

}

The first grammar is the general grammar for the whole program, other grammars are to identify the statement within the program

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

<id> <digit><digit><digit> <digit>

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **program** | f2021 | ; | **var** | a1 , b2a , c, ba12 | **integer;** | **begin** | a1 = 3 ;  b2a = 4 ;  wc = 5 ;  write( c );  ba12 = a1 \* (b2a + 2 \* c ) ;  write(“value=”, ba12); | **end**. |

**What to turn in?** A three whole binder with the following components:

1. **The original grammar**

|  |  |
| --- | --- |
| Original Grammar | |
| <prog> | 🡪 **program** <identifier>; **var** <dec-list> **begin** <stat-list> **end.** |
| <identifier> | 🡪 <letter>{<letter>|<digit>} |
| <dec-list> | 🡪 <dec> : <type> ; |
| <dec> | 🡪 <identifier> , <dec>| < identifier > |
| <type> | 🡪 **integer** |
| <stat-list> | 🡪 <stat> | <stat> <stat-list> |
| <stat> | 🡪 <write> | <assign> |
| <write> | 🡪 **print**(“value=”, < identifier > ); |
| <assign> | 🡪 < identifier > = <expr>; |
| <expr> | 🡪 <expr> + <expr> | <expr> - < expr> | < term> |
| <term> | 🡪 <term> \* <factor> | <term> / <factor>| <factor> |
| <factor> | 🡪 < identifier > | <number> | ( <expr> ) |
| <number> | 🡪 <sign><digit>{<digit> } note: this grammar is in EBNF |
| <sign> | 🡪 + | - | λ |
| <digit> | 🡪 0|1|2|…|9 |
| <letter> | 🡪 a | b | c | d | f |

1. **The original grammar in BNF form** (remove all {, } , and | )
2. **Preparation for the required table:** If you are using **Predictive Parsing Table** (method 1), remove all left recursions. For **LR Parsing Table** (method 2), remove all lambdas. **Show the final form of the grammar you will use in BNF**
3. Show the members of **FIRST** and **FOLLOW** of all non-terminals (assign uppercase letters for non-terminals as you wish for simplicity)

|  |  |  |  |
| --- | --- | --- | --- |
| State | New Name | FIRST | FOLLOW |
| <prog>  <identifier> | P  I | { program }  { a b c d f } | { $ }  ……  { a b c d f  0 1 2 3 4 5 6 7 8 9 } |

1. Show the **Parsing Table**
2. Complete **copy of your program** including all user defined libraries
3. A flash drive containing above documents