**Computer Science 323**

**Final project.** Your demonstration day is …………………………**Fall 2021**

**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 f2021;

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

3. Of an expression \*\*

4. var

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

6. a1 , b2a , wc, ba12 : integer ;

7. begin

8. a1 = 3 ;

9. b2a = 4 ;

10. wc = 5 ;

11. write(wc ); \*\* display wc \*\*

12.

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

14. ba12 = a1 \* ( b2a + 2 \* wc) ;

15. write ( “value=”, ba12 ) ; \*\* print the value of ba12 \*\*

16. end.

Apply the following rules 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, 11,13,15), remove them
2. Any blank lines must be removed (i.e. line #12)
3. Extra spaces in each line must be removed, Leave one space before and one after each token (example: line 8: a1 = 3 ; )

The “finalp2.txt” should look like this

1. program f2021 ;

2. var

3. a1 , b2a , wc , ba12 : integer ;

4. begin

5. a1 = 3 ;

6. b2a = 4 ;

7. wc = 5 ;

8. write (wc ) ;

8. ba12 = a1 \* ( b2a + 2 \* wc) ;

9. write ( “value=” , ba12 ) ;

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>} |
| <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> } |
| <sign> | 🡪 + | - | λ |
| <digit> | 🡪 0|1|2|…|9 |
| <letter> | 🡪 a|b|c|d|w|f |

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

|  |
| --- |
| Do this part only if you want to receive “100%” for the project( 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  **;** is missing  **,**  is missing  **.**  is missing  **(** Left parentheses is missing  **)** Right parentheses is missing  If the **write** spells wrong, issue an error message |

Output : Either one of the above messages or No error/ Some errors without error messages

**Part III ( 20 )**

If there are no ERRORS, 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 , wc, ba12 ;

a1 = 3 ;

b2a = 4 ;

wc = 5 ;

cou<<wc ;

ba12 = a1 \* (b2a + 2 \* wc ) ;

cout<< “value=” <<ba12<<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?

1. The original grammar

|  |  |
| --- | --- |
| Original 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: this grammar is in EBNF |
| <sign> | 🡪 + | - | λ |
| <digit> | 🡪 0|1|2|…|9 |
| <letter> | 🡪 a|b|c|d|w|f |

1. This grammar in BNF form (remove all {, } , and | )
2. If you are using table 1(predictive parsing table), remove all left recursions. For table 2, remove all lambdas. Show the final form of the grammar you will use in BNF
3. Find the members of first and follow of all terminals ( assign a single letter as you wish for simplicity)

|  |  |  |  |
| --- | --- | --- | --- |
| Non-terminals | | FIRST | FOLLOW |
| P  ……  L | <program>  ………  <letter> | { program }  { a b c d w f } | { $ }  ……  {a b c d w f  0 1 2 3 4 5 6 7 8 9 ; , ) = : \* / + - } |

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