**Computer Science 455**

**Final project.** Your demonstration day is …………………………**Fall 2020**

**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 f2020;

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 end 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( line 8: a1 = 3 ; )

The “finalp2.txt” should look like this

1. program f2020 ;

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>, 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 an “A” in the project, otherwise your maximum score is “B”  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

**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;

}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| <prog>🡪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** | f2020 | ; | **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>} |
| <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 |

1. The grammar in BNF form (remove all {, } , and | )

|  |  |
| --- | --- |
| BNF | |
| <prog> | 🡪 **program** <identifier> **;** **var** <dec-list>  **begin** <stat-list> **end.** |
| <identifier> | 🡪 <identifier> <letter> |
| <identifier> | 🡪 <identifier> <digit> |
| <identifier> | 🡪 <letter> |
| <dec-list> | 🡪 <dec> **:** <type> **;** |
| <dec> | 🡪 <identifier> **,** <dec> |
| <dec> | 🡪 <identifier> |
| <type> | 🡪 **integer** |
| <stat-list> | 🡪 <stat> <stat-list> |
| <stat-list> | 🡪 <stat> |
| <stat> | 🡪 <write> |
| <stat> | 🡪 <assign> |
| <write> | 🡪 **write** **(** <str> <identifier> **)** **;** |
| <str> | 🡪**”value=”** **,** |
| <str> | 🡪 λ |
| <assign> | 🡪 <identifier> **=** <expr> **;** |
| <expr> | 🡪 <expr> **+** <term> |
| <expr> | 🡪 <expr> **-** <term> |
| <expr> | 🡪 <term> |
| <term> | 🡪 <term> **\*** <factor> |
| <term> | 🡪 <term> **/** <factor> |
| <term> | 🡪 <factor> |
| <factor> | 🡪 <identifier> |
| <factor> | 🡪 <number> |
| <factor> | 🡪 **(** <expr> **)** |
| <number> | 🡪 <number> <digit> |
| <number> | 🡪 <sign> <digit> |
| <sign> | 🡪 **+** |
| <sign> | 🡪 **-** |
| <sign> | 🡪 **λ** |
| <digit> | 🡪 **0** |
| <digit> | 🡪 **1** |
| <digit> | 🡪 **2** |
| <digit> | 🡪 **3** |
| <digit> | 🡪 **4** |
| <digit> | 🡪 **5** |
| <digit> | 🡪 **6** |
| <digit> | 🡪 **7** |
| <digit> | 🡪 **8** |
| <digit> | 🡪 **9** |
| <letter> | 🡪 **a** |
| <letter> | 🡪 **b** |
| <letter> | 🡪 **c** |
| <letter> | 🡪 **d** |
| <letter> | 🡪 **w** |
| <letter> | 🡪 **f** |

1. If you are using table 1, remove all left recursions. For table 2, remove all lambdas. Show the final form of the grammar in BNF

|  |  |  |
| --- | --- | --- |
|  | **Final BNF** | |
| 1 | <prog> | 🡪 **program** <identifier> **; var** <dec-list> **begin** <stat-list> **end.** |
| 2 | <identifier> | 🡪 <identifier> <letter> |
| 3 | <identifier> | 🡪 <identifier> <digit> |
| 4 | <identifier> | 🡪 <letter> |
| 5 | <dec-list> | 🡪 <dec> **:** <type> **;** |
| 6 | <dec> | 🡪 <identifier> **,** <dec> |
| 7 | <dec> | 🡪 <identifier> |
| 8 | <type> | 🡪 **integer** |
| 9 | <stat-list> | 🡪 <stat> <stat-list> |
| 10 | <stat-list> | 🡪 <stat> |
| 11 | <stat> | 🡪 <write> |
| 12 | <stat> | 🡪 <assign> |
| 13 | <write> | 🡪 **write (** <str> <identifier> **) ;** |
| 14 | <write> | 🡪 **write (** <identifier> **) ;** |
| 15 | <str> | 🡪**”value=” ,** |
| 16 | <assign> | 🡪 <identifier> **=** <expr> **;** |
| 17 | <expr> | 🡪 <expr> **+** <term> |
| 18 | <expr> | 🡪 <expr> **-** <term> |
| 19 | <expr> | 🡪 <term> |
| 20 | <term> | 🡪 <term> **\*** <factor> |
| 21 | <term> | 🡪 <term> **/** <factor> |
| 22 | <term> | 🡪 <factor> |
| 23 | <factor> | 🡪 <identifier> |
| 24 | <factor> | 🡪 <number> |
| 25 | <factor> | 🡪 **(** <expr> **)** |
| 26 | <number> | 🡪 <number> <digit> |
| 27 | <number> | 🡪 <sign> <digit> |
| 28 | <number> | 🡪 <digit> |
| 29 | <sign> | 🡪 **+** |
| 30 | <sign> | 🡪 **-** |
| 31 | <digit> | 🡪 **0** |
| 32 | <digit> | 🡪 **1** |
| 33 | <digit> | 🡪 **2** |
| 34 | <digit> | 🡪 **3** |
| 35 | <digit> | 🡪 **4** |
| 36 | <digit> | 🡪 **5** |
| 37 | <digit> | 🡪 **6** |
| 38 | <digit> | 🡪 **7** |
| 39 | <digit> | 🡪 **8** |
| 40 | <digit> | 🡪 **9** |
| 41 | <letter> | 🡪 **a** |
| 42 | <letter> | 🡪 **b** |
| 43 | <letter> | 🡪 **c** |
| 44 | <letter> | 🡪 **d** |
| 45 | <letter> | 🡪 **w** |
| 46 | <letter> | 🡪 **f** |

1. Find the members of first and follow

|  |  |  |
| --- | --- | --- |
| Non-terminals | FIRST | FOLLOW |
| <prog> | { **program** } | { **$** } |
| <identifier> | { **a b c d w f** } | { **a b c d w f 0 1 2 3 4 5 6 7 8 9 + - \* / = ) , : ;** } |
| <dec-list> | { **a b c d w f** } | { **begin** } |
| <dec> | { **a b c d w f** } | { **:** } |
| <type> | { **integer** } | { **;** } |
| <stat-list> | { **a b c d w f** **write** } | { **end.** } |
| <stat> | { **a b c d w f write** } | { **a b c d w f write end.** } |
| <write> | { **write** } | { **a b c d w f write end.** } |
| <str> | { **“value=”** } | { **a b c d w f** } |
| <assign> | { **a b c d w f** } | { **a b c d w f write end.** } |
| <expr> | { **a b c d w f 0 1 2 3 4 5 6 7 8 9 + - (** } | { **+ - ) ;** } |
| <term> | { **a b c d w f 0 1 2 3 4 5 6 7 8 9 + - (** } | { **+ - \* / ) ;** } |
| <factor> | { **a b c d w f 0 1 2 3 4 5 6 7 8 9 + - (** } | { **+ - \* / ) ;** } |
| <number> | { **0 1 2 3 4 5 6 7 8 9 + -** } | { **0 1 2 3 4 5 6 7 8 9 + - \* / ) ;** } |
| <sign> | { **+ -** } | { **0 1 2 3 4 5 6 7 8 9** } |
| <digit> | { **0 1 2 3 4 5 6 7 8 9** } | { **a b c d w f 0 1 2 3 4 5 6 7 8 9 + - \* / = ) , : ;** } |
| <letter> | { **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

**The parsing table is too large to fit. See attached Microsoft Excel Spreadsheet.**

1. Complete copy of the program including all user defined libraries

**See Attached Source Files. main.py is the main driver.**