CS305 – Programming Languages

Fall 2025 – 2026

HOMEWORK 3

Implementing a Semantic Analyzer for Calculator Scientifique (CS)

Due date: December ~~8th~~ 15th, 2025 @ 23:55

NOTE

Only SUCourse submission is allowed. No submission by e-mail. Please see the note at the end of this document for the late submission policy.

1 Introduction

In this homework, you will implement a tool that includes a simple semantic analyzer and a translator for the CS language. Detailed information about the context-free grammar of this programming language was given in the second homework document. You can check it for more information.

The tool that you will implement in this homework will first check if a given CS program has any grammatical syntax errors. The tool will also perform some semantic checks if there are no syntax errors. If these checks pass, your tool will print some results. Read the rest of the document for more information.

2 Parser and Scanner

The scanner and the parser, which you can use to implement this homework, are provided to you. The semantic analysis will require you to implement an attribute grammar. You can start from the scanner/parser files provided, or, of course, you can write your own versions of scanner and parser from scratch.

3 Semantic Rules

Your semantic analyzer should start by performing an analysis for the following semantic rules. Your semantic analyzer must print out an error message for each violation of these rules. Note that after printing out an error message, your semantic analyzer must not terminate and keep working to find further violations, if any exist.

1. For function definitions in the definition block, the names of all functions must be unique. In other words, redefinitions or function overloading are not possible. If there are redefined functions, for each redefinition, an error message in the following format must be printed:

   `LINE_REDEFINED_FUNCTION_(REDEFINED_FUNCTION_VARIABLE_NAME)`

   Here, `LINE` must be the line on which the redefined function definition appears. For example, if there is such a definition block:

```
1
2       f(x) = x^2 + 5;
3       g(t,u) = t + u;
4       f(y,z) = y^3z^2 - 2yz;
5       g(k) = 6k^2;
6       f(t) = t;
```

   Then the following error message will be printed:

   `4_REDEFINED_FUNCTION_(f)`

   `5_REDEFINED_FUNCTION_(g)`

   `6_REDEFINED_FUNCTION_(f)`

2. For function definitions in the definition block, all variables used on the right-hand side must be valid function parameters. In other words, undefined parameters shall not be used in the function definitions. If there are undefined parameters, for each undefined parameter, an error message in the following format must be printed:

   `LINE_UNDEFINED_FUNCTION_PARAMETER_(UNDEFINED_FUNCTION_PARAMETER_NAME)`

   Here, `LINE` must be the line on which the undefined function parameter appears. Keep in mind that, if more than one undefined function parameter comes on the same line, then the error messages should be printed in lexicographic order of the undefined parameters. For example, if there is such a definition block:

```
1       f(x) = y^2 + 5;
2       g(t,u) = t +
3                z;
4       k(y,z) = b^3a^2 - 2yz;
5       j(k) = 6k^2;
6       l(t) = x;
```

Then the following error message will be printed:

`1_UNDEFINED_FUNCTION_PARAMETER_(y)`

`3_UNDEFINED_FUNCTION_PARAMETER_(z)`

`4_UNDEFINED_FUNCTION_PARAMETER_(a)`

`4_UNDEFINED_FUNCTION_PARAMETER_(b)`

`6_UNDEFINED_FUNCTION_PARAMETER_(x)`

3. For function expressions in the calculation block, the number of parameters must match the number of parameters in the definition of the function. That is, the same arity must be provided. If there is any contradiction to this rule, an error message in the following format must be printed:

   `LINE_ARITY_CONTRADICTION_(FUNCTION_NAME)`

   Here, `LINE` must be the line on which function name with the arity contradiction appears. For example, if there is such a program:

```
1  f(x) = x^2 + 5;
2  g(t,u) = t + u;
3
4  calculate f(x,y) +
5            g(a) f(a,
6                   b,
7                   c)
8            g
9            (p,r,s);
```

   Then the following error message will be printed:

   `4_ARITY_CONTRADICTION_(f)`

   `5_ARITY_CONTRADICTION_(f)`

   `8_ARITY_CONTRADICTION_(g)`

   Note: Here, the first appearance of function `g` on line 5 does not cause an arity contradiction; it is an implicit multiplication instead.

4. For function expressions in the calculation block, the name of the function must be provided. If there are such function expressions with no function name provided, an error message in the following format must be printed:

   `LINE_MISSING_FUNCTION_NAME`

   Here, `LINE` must be the line on which the `calculate` keyword having a function expression with a missing function name appears. If there are more than such errors for the same `calculate` statement, this error must be printed for each such expression. Keep in mind that, we can understand that an expression is supposed to be a function expression when we see a comma separated expressions between parenthesis. However, if what we have between the parenthesis is not such a comma separated list, but a single expression (e.g. `( x+y )` , in this case this is not necessarily understood as a function expression without a function name. In such a case, no error should be printed. For example, if there is such a program:

```
1   f(x) = x^2 + 5;
2
3   calculate
4           (
5             x,y
6           )
7           (
8             a,b,c
9           )(d)(e + f)
10         ;
11  calculate (a) f(c);
```

Then the following error message will be printed:

```
3_MISSING_FUNCTION_NAME
3_MISSING_FUNCTION_NAME
```

Note that, no error for the `calculate` statement at line 11 is produced, since `(a)` on this line is simply considered as a simple expression referring to the variable `a`.

5. For any function expression in calculation statements, the function used in the expression must be defined in the definition block. If a function expression does not obey this rule, an error message in the following format must be printed:

   `LINE_UNDEFINED_FUNCTION_(UNDEFINED_FUNCTION_VARIABLE_NAME)`

   Here, `LINE` must be the line on which the undefined function appears. For example, if there is such a program:

```
1       f(x) = 2x^3 + 3x - 8;
2       g(a,b) = a^2 + b^2;
3
4       calculate
5                 f(a) +
6                 h(
7                 a,c)
8                 g(x,y) - j
9                 (z,v) -
10                i(t,u)
11                i(k);
```

   Then the following error message will be printed:

   `6_UNDEFINED_FUNCTION_(h)`

   `8_UNDEFINED_FUNCTION_(j)`

   `10_UNDEFINED_FUNCTION_(i)`

6. For any derivation expression in the calculation block, the first parameter must be a defined function name. If the first element is not a defined function name (or which means the function given in the derivation expression is not defined), an error message in the following format must be printed:

LINE_UNDEFINED_FUNCTION_FOR_DERIVATION_(UNDEFINED_FUNCTION_NAME)

Here, LINE must be the line on which the missing function name error appears. For example, if there is such a program:

```
1  f(x) = 4x;
2
3  calculate D(f,x,1);
4  calculate D(
5            g,b,3);
```

Then the following error message will be printed:

5_UNDEFINED_FUNCTION_FOR_DERIVATION_(g)

7. For any derivation expression in the calculation block, the second parameter must be a variable of the function as given in the definition block. If the second parameter is not a variable (which means the variable given in the derivation expression is not used in the left–hand–side of the function in the definition block), an error message in the following format must be printed:

   LINE_UNDEFINED_VARIABLE_FOR_DERIVATION_(UNDEFINED_VARIABLE_NAME)

   Here, LINE must be the line on which the incorrect variable appears. For example, if there is such a program:

```
1  f(x) = 4x;
2  g(a) = a^2;
3  calculate D(f,
4              y,1);
5  calculate D(g,b,3);
```

   Then the following error message will be printed:

   4_UNDEFINED_VARIABLE_FOR_DERIVATION_(y)
   5_UNDEFINED_VARIABLE_FOR_DERIVATION_(b)

8. For any integration expression in the calculation block, the first parameter must be a defined function name. If the first element is not a defined function name (or which means the function given in the integration expression is not defined), an error message in the following format must be printed:

   LINE_UNDEFINED_FUNCTION_FOR_INTEGRATION_(UNDEFINED_FUNCTION_NAME)

   Here, LINE must be the line on which the missing function name error appears. For example, if there is such a program:

```
1  f(x) = 4x;
2
3  calculate I(f,x,1);
4  calculate I(g,b,3);
```

   Then the following error message will be printed:

   4_UNDEFINED_FUNCTION_FOR_INTEGRATION_(g)

9. For any integration expression in the calculation block, the second parameter must be a variable of the function as given in the definition block. If the second parameter is not a variable (which means the variable given in the integration expression is not used in the left–hand–side of the function in the definition block), an error message in the following format must be printed:

   `LINE_UNDEFINED_VARIABLE_FOR_INTEGRATION_(UNDEFINED_VARIABLE_NAME)`

   Here, `LINE` must be the line on which the incorrect variable appears. For example, if there is such a program:

```
1  f(x) = 4x;
2  g(a) = a^2;
3  calculate I
4            (f,y,1);
5  calculate I(g,
6            b
7            ,3);
```

   Then the following error message will be printed:

   `4_UNDEFINED_VARIABLE_FOR_INTEGRATION_(y)`
   `6_UNDEFINED_VARIABLE_FOR_INTEGRATION_(b)`

10. If the program comprises several different semantic errors, then the errors will be printed lexicographically. For instance, if there is such a program:

```
1  f(x) = 4ax + y;
2  g(t,u) = t + u;
3  f(y,z) = y^3z^2 - 2yz;
4  calculate (x,y) f(a,b) g(t,u,z);
5  calculate h(x,y);
6  calculate I(m,t,1);
7  calculate I(k,t,1) D(k,y,3);
8  calculate D(g,b,3);
```

    Then the following error message will be printed:

    `1_UNDEFINED_FUNCTION_PARAMETER_(a)`
    `1_UNDEFINED_FUNCTION_PARAMETER_(y)`
    `3_REDEFINED_FUNCTION_(f)`
    `4_ARITY_CONTRADICTION_(f)`
    `4_ARITY_CONTRADICTION_(g)`
    `4_MISSING_FUNCTION_NAME`
    `5_UNDEFINED_FUNCTION_(h)`
    `6_UNDEFINED_FUNCTION_FOR_INTEGRATION_(m)`
    `7_UNDEFINED_FUNCTION_FOR_DERIVATION_(k)`
    `7_UNDEFINED_FUNCTION_FOR_INTEGRATION_(k)`
    `8_UNDEFINED_VARIABLE_FOR_DERIVATION_(b)`

4 Calculator Scientifique Report

Suppose the input given in CS language is correct both grammatically (with respect to the grammar) and semantically (with respect to the semantic rules given above). In that case, your translator will process the input to print all calculation expressions according to the following rules:

1. The CS program should report the simplest version of the calculation results for all calculation statements, in a canonical form, as explained below.

2. The calculation result is calculated by the following rules.

   - **Computation of function expressions:** A function expression will be replaced by the expression provided for the corresponding function in the definition block. For example, if we have,
     `f(x,y) = 3x^2 + 4xy` in the definition block, then a function expression like `f(a,5)` would be computed as `3a^2 + 4*a*5` whose result is `3a^2 + 20a` after calculation, and a function expression like `f(a,a)` would be computed as `3a^2 + 4*a*a` whose result is `3a^2 + 4a^2 = 7a^2` after calculation.
     Note: If we have an expression like `f(5)` in the calculation part, it can be interpreted in two ways. It can be a function `f` applied on the parameter `5`, or it can mean `f * (5)`. The resolution will be made in the following way. If there is a function with name `f` defined in the definition block, `f(5)` will always be considered as a function expression (keep in mind that, being in this condition check implies that there is no arity contradiction in the calculation statement). However, if there is no function `f` defined in the definitions block, `f(5)` will be considered as `f * 5`.
   - **Computation of derivation expressions:** A derivation expression will be replaced by the result of the derivation of the function name given as the first parameter, with respect to the parameter provided in the derivation expression. The degree of the derivation is provided as the third parameter in the derivation expression. For example, if we have,
     `f(x,y) = 4x^3y + 4xy - 5` then a derivation expression like `D(f,x,2)`, which can be read as *Second derivation of* `f` *in terms of* `x`, would be computed as `24xy`.
     (This is because the 1^{st} derivative of `f` in terms of `x` is `12x^2y + 4y`. And the 2^{nd} derivative of `f`, which is the derivative of the 1^{st} derivative of `f`, in terms of `x` is `24xy`.)
   - **Computation of integration expressions:** An integration expression will be replaced by the result of the integration. The first parameter is the function name, the second parameter is the variable that the integration expression will be calculated in terms of. And the degree of integration is provided as the third parameter in the integration expression. For example, if we have,
     `f(x) = 4x^2` then an integration expression like `I(f,x,2)`, which can be read as *Second integration of* `f` *in terms of* `x`, would be computed as `1/3x^4`.

Note that, normally $\int\int 4x^2\,dx = \frac{1}{3}x^4 + c_1x + c_2$, where c_1, c_2 are integration constants. However, for the sake of simplicity, we will consider all such integration constants as 0. That is, we calculate the 1^{st} integration of `f` as $\frac{4}{3}x^3 + 0$.

- If the calculation statement does not consist of any derivation or integration statement, then apply classical polynomial function substitution rules.
- If there are derivation or integration statements, then apply the derivation or integration accordingly. In the statements, the first parameter is the function name, the second is the variable name, and the third is the calculation level. For instance, `I(f,x,2)` means the 2nd integration of the function `f` with respect to `x`; and `D(g,y,3)` means the 3rd derivation of the function `g` with respect to `y`.

After the computations for function expressions, derivation expressions, and integration expressions are performed as explained above, we will be left with an expression consisting of coefficients, variables (raised to a power), and operators. This expression will be simplified using standard arithmetic rules and then printed in the output, as explained below.

3. Each line in the output corresponds to a calculation statement in the input.

4. The output format is as follows. A line produced for a calculation statement consists of three parts:
(i) the entire expression given in the input after the calculation statement (with all white space characters removed),
(ii) an equality symbol,
(iii) the result of the expression calculated.

   For example, if the program looks like the following:

```
1  f(x) = 3x;
2  calculate f(x) + f(3);
3  calculate f(y) + f(z);
```

   The report to be produced will be this:

```
f(x)+f(3)=3x+9
f(y)+f(z)=3y+3z
```

5. The result of the calculation printed to the right of the equality symbol is an expression consisting of a sequence of *terms*, which are separated by addition and subtraction operators. For example, in:

```
x^9+504x^6-2x-6
```

   the terms are `x^9`, `504x^6`, `2x`, and `6`.

6. Each term consists of a *constant coefficient*, followed by a sequence of all lowercase letters in the alphabet (each one representing a variable), where each variable is raised to a power.

   - If the constant coefficient is 1, it is not printed.
   - If the power of a variable is 1, then the power is not printed.
   - If the power of a variable is 0, then the variable and its power are not printed.
   - **Important:** Each term will be printed in the lexicographic order of the variables that it includes.

   In other words, the following is printed: `4a^2c^3e`
   for the term `4a^2b^0c^3d^0e^1f^0g^0h^0...x^0y^0z^0`.
   NOT `4a^2ec^3` or in any different order.

7. There cannot be two different terms that agree in all the powers of the variables. For example, we cannot have `4a^2c^3e+5a^2c^3e` as two distinct terms in the output. Instead, they must be combined into a single term as `9a^2c^3e`.

8. In the output, the terms are printed in **decreasing** order with respect to the following rule. The constant coefficient of the term is not important. Term t1 is greater than term t2, if and only if

   - the power of a variable in t1 is strictly greater than the power of the same variable in t2, and
   - for all the previous variables, we have the same power in both t1 and t2.

   For all of the following cases, the first term (t1) is greater than the second (t2):
   `a^3` > `4a^2`
   `4a^2c^3` > `8ac^6`
   `4a^2c^3d` > `8a^2c^3`
   `a^2xyz` > `ab^2c^3`

9. If a function is defined in the first block, however does not appear in the calculation block, this function name will not be reported.

10. If the definition block contains no function definitions, yet the calculation block contains statements, then your program must report the result of the expressions in the calculation statements. For example, if the program looks like the following:

```
1   calculate (xy) * g(b) + f(2x + y) g(5b);
2   calculate (xy) * g(b) + f(2x) - y g(5b x);
```

The report to be produced will be this:

```
(xy)*g(b)+f(2x+y)g(5b)=10bfgx+5bfgy+bgxy
(xy)*g(b)+f(2x)-yg(5bx)=-4bgxy+2fx
```

Note 1: Note that even if we do not have fractions in the input, there may be fractional constant coefficients in the resultant expression due to integrations. The fractions will be printed in their simplest form, i.e., we will print `3/2` instead of `6/4`.

Note 2: Note that the grammar allows to have complex expressions to be used as parameters in the function expressions in the calculation block. For instance, `calculate f(x+2, a-3b)`, or `calculate g(h(x-y), h(z))`.

However, **only for testing the output generation capability of your programs**, we will not test your program with such inputs. We will only use variables or constants as parameters in function expressions. In other words, our function expressions will be of the form `calculate f(x,b)`, or `calculate g(1,x)`. Therefore, you do not need to handle the complex cases to produce the output.

Important: Please be aware that inputs like `calculate g(h(x-y), h(z))` can be used to test the semantic analysis capability of your programs. In other words, we do not want you to implement the calculation of complex expressions, yet if there exist any semantic errors in those expressions, we expect you to detect them and report them in your output.

If a complete program looks like the following:

```
1   f(x) = 3x;
2   g()=5 ;
3   h(x) = x + 2
4   ;
5   j(x,y) = (3x^2 - 2) (y^3 + 5);
6   k(x) = x^9 - x^3;
7
8   calculate f(x) + f(3);
9   calculate g() * f(y);
10  calculate h(x) j(a,b);
11  calculate k(x) + D(k,x,3);
12  calculate I(f,x,1);
13  calculate I(f,x,2);
14  calculate z(x);
```

The report to be produced will look like the following:

```
f(x)+f(3)=3x+9
g()*f(y)=15y
h(x)j(a,b)=3a^2b^3x+6a^2b^3+15a^2x+30a^2-2b^3x-4b^3-10x-20
k(x)+D(k,x,3)=x^9+504x^6-x^3-6
I(f,x,1)=3/2x^2
I(f,x,2)=1/2x^3
z(x)=xz
```

5 How to Submit

Submit your flex file and bison file named as `username-hw3.flx` and `username-hw3.y` respectively, where `username` is your SU-Net username. You may use additional files, such as a header file or additional C files. Please also upload such files as well. We will compile your files by using the following commands:

```
flex username-hw3.flx
bison -d username-hw3.y
gcc -o username-hw3 lex.yy.c username-hw3.tab.c -lfl
```

So, make sure that these three commands are enough to produce the executable. If we assume that there is a CS file named `example1.cs`, we will try out your parser by using the following command line:

```
username-hw3 < example1.cs
```

6 Notes

- **Important**: Name your files as you are told and **don't zip them.**
  [-20 points otherwise]
- **Important**: Even though we provide a parser to you, you can create your own parser from scratch, or you can use your parser from the 2^{nd} homework. Make sure that the parser you use **must** work in an **unambiguous** way. In other words, you shall not get any ambiguity warnings once it is compiled.
  [-20 points otherwise]
  You can assume the parser we gave works unambiguously.
- **Important:** Note that, since this homework does not aim to measure your grammar creation ability, we will not test your homework on syntactically erroneous test cases. However, if you use your own parser or scanner, and somehow your grammar gives a syntax error for a correct case, then you will lose points.
- **Important**: Make sure you include the correct "`...tab.h`" file in your scanner, and make sure you can compile your parser using the commands given in Section 5. If we are not able to compile your code with those commands **your grade will be zero for this homework.**
- **Important**: Some test cases are shared on the `cs305.sabanciuniv.edu` server. We are hoping to share a golden later, but no promise currently.
- **Important: Since this homework is evaluated automatically, make sure your output is exactly as it is supposed to be. Some of the points that we can think of are:**
  - **There should be no extra space at the beginning or at the end of a line.**

  - **Make sure that the spelling is as it is given in the homework document.**
  - **We check in a case-sensitive manner (e.g. "ERROR" $\neq$ "error")**

- You may get help from our TA or from your friends. However, **you must implement the homework by yourself**.
- Start working on the homework immediately.
- If you develop your code or create your test files on your own computer (not on `cs305.sabanciuniv.edu`), there can be incompatibilities once you transfer them to the `cs305` server. Since the grading will be done automatically on the `cs305` server, we strongly encourage you to do your development on the `cs305` server, or at least test your code on the `cs305` server before submitting it. If you prefer not to test your implementation on the `cs305` server, this means you accept to take the risks of incompatibilities. Even if you may have spent hours on the homework, you can easily get 0 due to such incompatibilities.

LATE SUBMISSION POLICY

Late submission is allowed subject to the following conditions:

- Your homework grade will be decided by multiplying what you get from the test cases by a "submission time factor (STF)".
- If you submit on time (i.e. before the deadline), your STF is 1. So, you don't lose anything.
- If you submit late, you will lose 0.01 of your STF for every 5 mins of delay.
- We will not accept any homework later than 500 mins after the deadline.
- SUCourse's timestamp will be used for STF computation.
- If you submit multiple times, the last submission time will be used.