PROJECT

In this project, you will write a lex yacc program that will convert a python code to c++ code. We do not want you to handle all the possible python codes. The input python codes can contain only assignment and if/else statements. Your project is to convert the given assignment and if/else statements into c++ language.

The project contains syntax and semantic analysis. First try to code the syntax analysis part, test your code, then try to write the semantic analysis part.

1. Syntax Analysis Part

In the syntax analysis, you should write lex and yacc code that will recognize and accept the given python code.

1.1. Assignment statement

- The code will contain several assignment states. An assignment state has a form of: VAR = Operand Operator Operand Operator …
- VAR is a variable. Operand can be a variable name, integer, float or string value. Numbers can be positive or negative. Operators can be “+”, “-”, “/” and “*”
- There can be infinite number of Operand and Operator
- Example
  X = a + 5 + cc * 12 - 14 / X

1.2. If/else statement

- The input code can contain if, if/else or if/elif/else statements. Also it can contain nested if statements (elif and else included)
- The body of if/else statements can contain several assignment or if/else statements.
- The body of the if/else states should start 1 tab inside. For each nested if, there must be an extra tab in the beginning. If it is not clear look at the example.
- The comparison part in the if/elif can have only comparison. Thus, the input code will not contain “**and**” or “**or**” in the comparison.
- Comparison can be “==”, “!=”, “<”,”<=”, “>” or “>=”.
- You can see an example in figure 1.

```
If a > b :
   a = 1
elif a < b:
   If a <= 0:
      a = 0
   a = a * a
else:
   a = b + 1
```

Figure 1: Example if/else statement

2. Semantic Analysis Part

In the semantic analysis part, you should check some rules to see if the given python code is correct or not. Also you should convert the given python code to c++ in this part.

2.1. Finding tab inconsistency

- A statement cannot start one or more tabs inside, if it is not in an if-statement.
- If there is a if-statement, elif or else statement, then there must be at least one statement that will start one tab inside just after the statement
- The number of tabs in the beginning of line cannot be bigger than the number of unclosed if/elif/else statements
- An if-statement is accepted as closed when a statement with same or lower indentation level comes except else or elif come statements. As it is seen in the example below, the “if” in line 2 is closed in line 5. Because line 5 is the first statement that has the same indentation level with the second if. Similarly, the first “if” statement is closed in line 7 since the line 7 is the first statement that has the same indentation level with the first if.

line	
1	If a == 5: first if is opened
2	If b == 3: second if is opened
3	c = 5 # there is 2 unclosed if
4	b = 2 # there is 2 unclosed if
5	c = c * 2 # at this point second if closed.
6	b = a * 2 # there is 1 unclosed if
7	a = c + b # at this point first if closed
8	a = a * 2 # there is no unclosed if

line	both of the if is closed in line 4 at the same time. Because indentation level of line 4 is equal or smaller than both if

1
2
3
4
	If a == 5:
	  If b == 3:
	  c = 5
	b = 2 #

- Print an error message “there is a tab inconsistency in line x” where x is the corresponding line
- There are examples of the errors you should catch in your code.

```
A = 3
  A = 5
```

```
if a == 3:
c=3
```

```
if a == 3:
  c=3
else:
d=2
```

```
if a == 3:
  if b > 5:
  b = a
```

2.2. If/else inconsistency

- For each elif and else statement there must be an unclosed if before the statement.
- An if-statement is accepted as closed when a statement with same indentation level comes except else or elif come statements.
- Print an error message “if/else consistency in line x” where x is the corresponding line.
- To be able to catch the if/else inconsistency, you need to find when an if is closed or not
- Examples with errors

```
else:
   d=2
```

```
if a == 3:
   c=3
d=2 # in this line if is closed so the next else does not have an if
else:
   c=2
```

```
If a == 3:
   c=3
else:
   c=2
elif:
   c=1
```

2.3. Type inconsistency

- You have already implemented type inconsistency (mismatch) in prelab 3. You will do the same thing in your project.
- In prelab3, you have assumed that a variable will not have more than 1 type. But as you know, python uses dynamic type binding and the type of the variable can change through the program flow.
- In the first example below, you can see the type of A is string initially. Then it is changed to integer in the next line. So the addition operation is valid in the example 1. On contrary, in example 2, the type of A is integer initially. And it is changed to string in the next line. So your program should give an error message for example 2.

Example 1 (valid)	Example 2 (invalid)
A = “deneme” A = 10 B = A + 20	A = 10 A = “deneme” B = A + 20

- Also In addition to the prelab 3, you should check the consistency in the comparison part. You cannot compare a string with an integer or a float. If the input code contains such a comparison, your program should detect it.

- Print an error message “type inconsistency in line x” where x is the corresponding line.

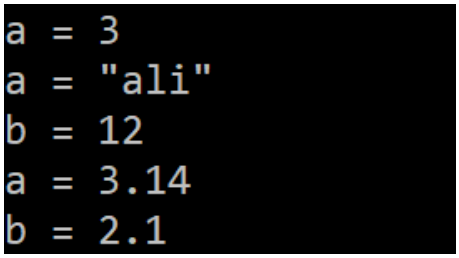
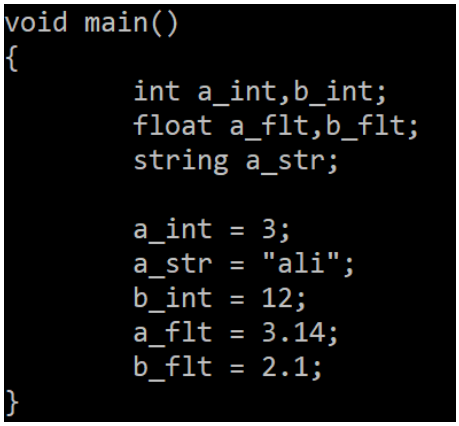
- There is a case you should not consider in the project. Let’s assume that we have code like below. In this case we do not know whether the program goes into if state or else state. Thus, we cannot decide the type of the variable A with running the program. Thus, you can assume that the type of a variable will not change in the if/elif/else statements. So in your project do not try to solve cases like this.

```
…
...
If b < 10:
    A = 10
else:
    A = “ali”
b = A + 10
```

3. Desired Output format

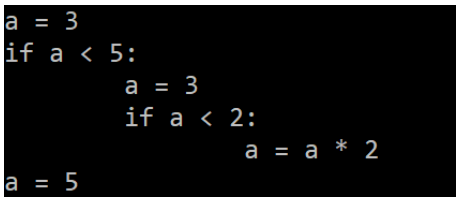
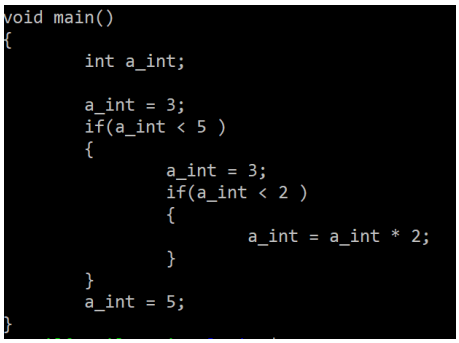
3.1. Type Declaration

- In assignment 3, you have already put type declaration of variables in the head of the program. In the project you will upgrade it.
- In python, a variable can have multiple types in the flow of the program since it uses dynamic type binding. Thus, a variable can be integer in the beginning, but it can be string in the end.
- In c++, the type of the variable cannot change in the flow of the program. So somehow we have to solve this problem while converting a python code into c++.
- In project, a variable can be int, float or string. So you should keep the record of the types that a variable uses. Then, convert a variable into a typed-variable in c++ output.
- For example, if a variable x is used as int, float and string in the input code, output should contain x_int, x_float and x_str. But if it is used only as int and string, then the output should only contain x_int and x_str.
- Each variable should be converted into a typed variable in the output and all typed-variables should be declared in the beginning of the main.
- Order of the declaration should be in: int, float and string as it is seen in the example below.

input	output
	

3.2. Indentation and { }

- Your code should be surrounded by the main function.
- Each line should start with an extra tab.
- Each if, else if and else body should start with “{“ with the correct indentation level.
- Each if, else if and else body should end with “}“ with the correct indentation level.

input	output
	

4. Testing Your Code and Grading Policy

You are given a set of valid input files and corresponding output files. Also, you are given a set of invalid input files that you should detect the inconsistency. Your project will be evaluated according to two points; syntax analysis and semantic analysis:

- First, your program should have a grammar that will recognise the input.
- Second, the correctness of your output for the given input file.

Also, a part of your project grade will be given from a few large input files that contain all the cases. Those input files will not be provided to you. Thus, you should write a generic program not just for the given examples.

4.1 Valid input files

- You should be able to create outputs that are exactly the same with the provided one.
- You do not need to write onto a file. Use redirection if you wish.
    - ./project input1.txt > outputStudent.txt
- The input files will not be large and each of them will contain only one of the cases in the project.
- If your output file is **exactly the same** as the provided one, you will get a full grade.
    - So do not try to do everything at the same time.
    - Finish one part, test your code and if you are sure it is okay save this version. Then, pass to the next part.
- We will compare the output files with diff command in terminal:
    - diff -s output1.txt outputStudent.txt
    - If two files are same, you will see on the screen:
        - Files output1.txt and outputStudent.txt are identical
    - If not, you will see which lines are different.
    - So, if we cannot see the “... are identical” message you will not get a grade from that input file

4.2 Invalid input files

- If a file contains invalid code, then you should print only an error message.
- The error message should be clear about the type of the error.
- Do not just print “error” on the screen.

4.3 File format

- In the project you will be dealing with a lot of whitespace. Some whitespace characters can vary on different operating systems like newline.
- Use dos2unix command to be sure that all the files are in the same format.
    - You can install it: sudo apt install dos2unix
    - And use it: dos2unix input1.txt

4.5 Report

Write a report that briefly explains in which input files you are successful in terms of syntax and semantic analysis.

5 SUBMISSION

The files you should submit

- Your lex file
- Your yacc file
- Your Makefile

Your makefile should work correctly. And the first line of the Makefile should contain:

```
all: lex yacc
        g++ lex.yy.c y.tab.c -ll -o project
```

It is essential to name your executable as “project” as it is seen above.

GOOD LUCK