**به نام خدا**

نام استاد: دکتر سعید پارسا

نام درس: کامپایلر

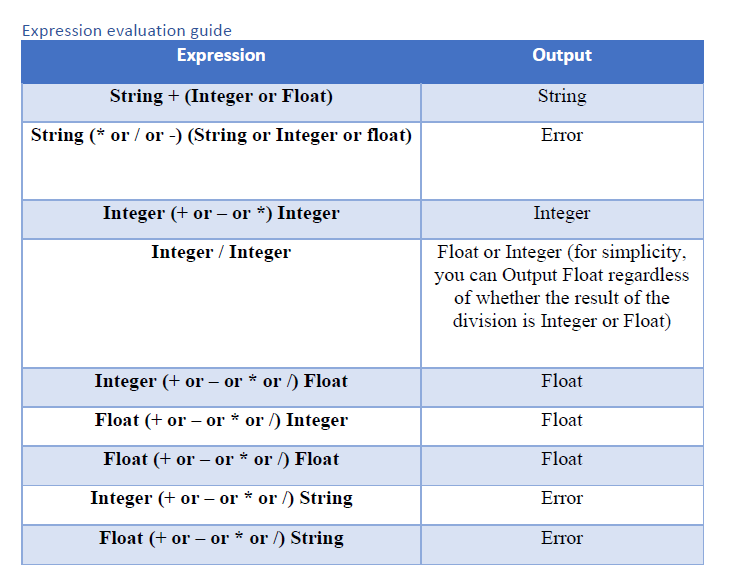
نام و نام خانوادگی: نیوشا یقینی

شماره دانشجویی: 98522346

گزارش تمرین: HW2

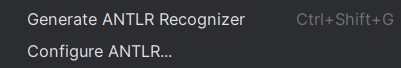
تاریخ: 19/01/1403

در این تمرین Listener به نحوی تعریف شده است که شرایط زیر را برقرار کند:



مراحل:

1. ابتدا با استفاده از 2 گزینه زیر فولدر gen را ساختیم.



1. سپس با توجه به تمرین سری قبل فایل پایتون Listener را ایجاد کردیم و توابع exit فایل TypeCheckerListener از فولدر gen را داخل آن کپی کردیم.
2. برای درک بهتر فرآیند صدا زده شدن توابع داخل هرکدام از آن ها print گذاشتیم و مثال های مختلف را اجرا کردیم.
3. و در ادامه به کمک خواندن بهتر grammar فهمیدیم که توابع exitFactFloat و exitFactInteger و exitFactString داخلی ترین توابع هستند و برای پیاده سازی بهتر است از این ها شروع شود، و بعد از آن ها توابع exitTerm1 و exitTerm2 که "\*" و "/" هستند داخلی ترین هستند و الویت برای پیاده سازی قرار میگیرند، و در اخر هم توابع exitExpr1 و exitExpr2 که "+" و "-" هستند را پیاده سازی میکنیم. (توجه شود که Term3 همان Factor و Expr3 همان Term در نظر گرفته شده است پس نیاز بخصوصی برای پیاده سازی آن ها وجود ندارد.)
4. در اینجا باید ابتدا به راه حلی میرسیدیم که بتوانیم به کمک آن بتوان ترتیب ورودی ها را هم در نظر گرفت، به همین دلیل، در قسمت تعریف property های کلاس امان، یک stack اضافه کردیم که هربار که به برگ ها میرسیم آن را به stack اضافه کنیم و در مرحله ی بعد برش داریم، تا بتوانیم با توجه به ترتیب ورودی ها در مورد جنس آن ها در ادامه تصمیم بگیریم. در همینجا یک property دیگر به نام error نیز تعریف می کنیم که در صورتی که در جایی از معادله به type error خوردیم این مقدار را برابر yes قرار میدهیم تا حواسمان باشد دیگر نیازی نیست ادامه دهیم.
5. در مرحله بعدی توابع انتخاب شده را پیاده سازی کردیم.



داخل توابع ابتدا یک print جهت تشخیص بهتر خروجی ها گذاشتیم، سپس مقدار result را با توجه به جنس تابعی که داخلش هستیم ست کرده و آن را print کرده ایم، و در انتها مقدار result را در stack امان، push می‌کنیم. (print ها برای درک بهتر گذاشته شده‌اند.)

1. در مرحله بعد نوبت توابع exitTerm1 و exitTerm2 است. پیاده سازی آن ها را با توجه به علامتشان و قواعد موجود در ابتدای گزارش انجام داده‌ایم.

# Exit a parse tree produced by TypeCheckerParser#term1.  
def exitTerm1(self, ctx):  
 print("Term1")  
  
 if(self.error=="no"):  
 second\_operator = self.stack.pop()  
 first\_operator = self.stack.pop()  
  
 if(first\_operator=="String"):  
 self.error = "yes"  
 self.result = "Type error: String cannot be multiplied"  
 elif(first\_operator=="Integer"):  
 if(second\_operator=="Integer"):  
 self.result = "Integer"  
 self.stack.append(self.result)  
 elif(second\_operator=="Float"):  
 self.result = "Float"  
 self.stack.append(self.result)  
 else:  
 self.error = "yes"  
 self.result = "Type error: String cannot be multiplied"  
 elif(first\_operator=="Float"):  
 if (second\_operator == "Integer" or second\_operator == "Float"):  
 self.result = "Float"  
 self.stack.append(self.result)  
 else:  
 self.error = "yes"  
 self.result = "Type error: String cannot be multiplied"  
  
 print("result = ", self.result)  
 print("stack = ", self.stack)  
 pass

# Exit a parse tree produced by TypeCheckerParser#term2.  
def exitTerm2(self, ctx):  
 print("Term2")  
  
 if(self.error=="no"):  
 second\_operator = self.stack.pop()  
 first\_operator = self.stack.pop()  
  
 if(first\_operator=="String"):  
 self.error = "yes"  
 self.result = "Type error: String cannot be divided"  
 elif(first\_operator=="Integer"):  
 if(second\_operator=="Integer" or second\_operator=="Float"):  
 self.result = "Float"  
 self.stack.append(self.result)  
 else:  
 self.error = "yes"  
 self.result = "Type error: String cannot be divided"  
 elif(first\_operator=="Float"):  
 if (second\_operator == "Integer" or second\_operator == "Float"):  
 self.result = "Float"  
 self.stack.append(self.result)  
 else:  
 self.error = "yes"  
 self.result = "Type error: String cannot be divided"  
  
 print("result = ", self.result)  
 print("stack = ", self.stack)  
 pass

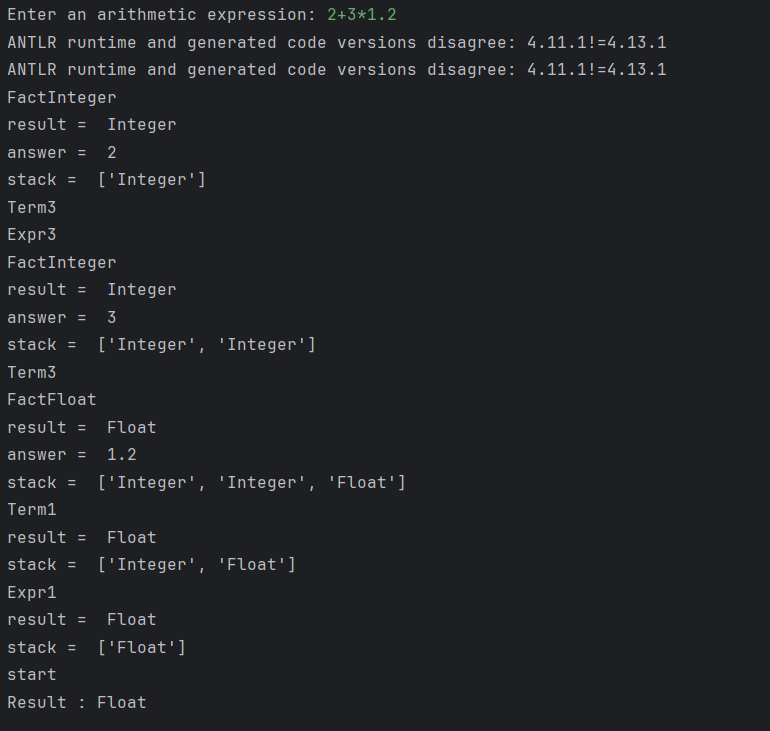
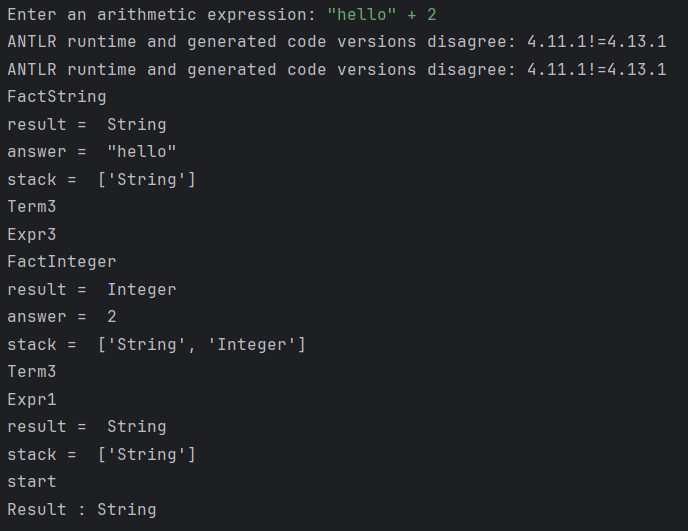
1. در ادامه نیز توابع exitExpr1 و exitExpr2 را با توجه به علامتشان و قواعد موجود در ابتدای گزارش پیاده سازی کردیم.

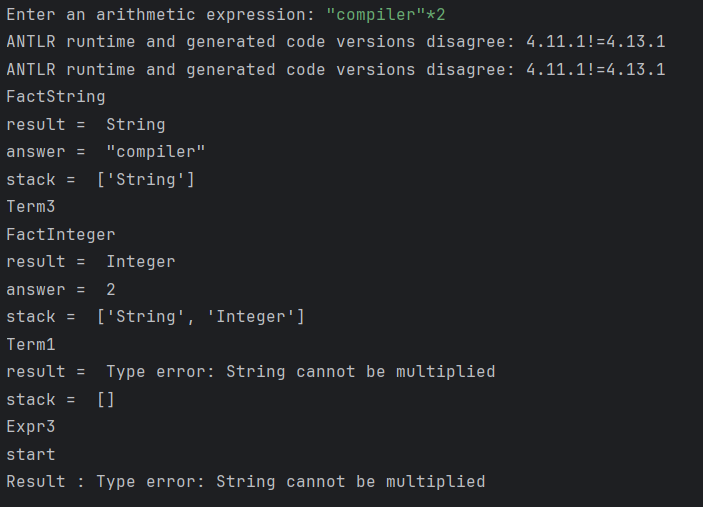
برای پیاده سازی ابتدا چک می‌کنیم که error برابر yes نباشد، چراکه اگر باشد، یعنی یک جای معادله تا به الان غلط بوده است و نیازی به ادامه دادن نیست.

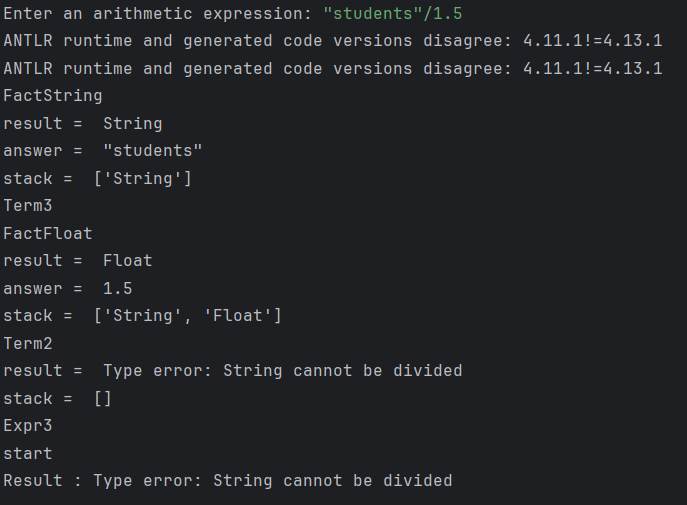
# Exit a parse tree produced by TypeCheckerParser#expr1.  
def exitExpr1(self, ctx):  
 print("Expr1")  
  
 if (self.error == "no"):  
 second\_operator = self.stack.pop()  
 first\_operator = self.stack.pop()  
  
 if (first\_operator == "String"):  
 self.result = "String"  
 self.stack.append(self.result)  
 elif (first\_operator == "Integer"):  
 if (second\_operator == "Integer"):  
 self.result = "Integer"  
 self.stack.append(self.result)  
 elif (second\_operator == "Float"):  
 self.result = "Float"  
 self.stack.append(self.result)  
 else:  
 self.error = "yes"  
 self.result = "Type error: String cannot be concatenated to an Integer"  
 elif (first\_operator == "Float"):  
 if (second\_operator == "Integer" or second\_operator == "Float"):  
 self.result = "Float"  
 self.stack.append(self.result)  
 else:  
 self.error = "yes"  
 self.result = "Type error: String cannot be concatenated to an Float"  
  
 print("result = ", self.result)  
 print("stack = ", self.stack)  
 pass

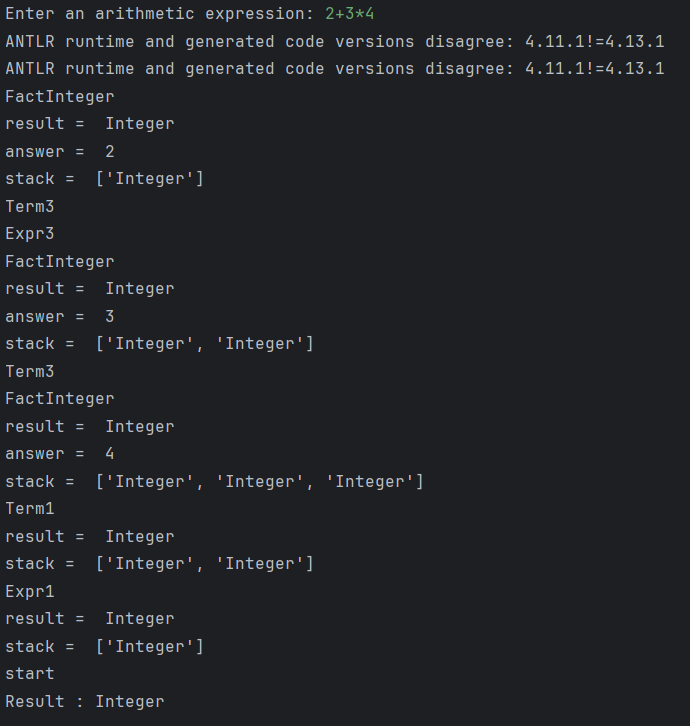
# Exit a parse tree produced by TypeCheckerParser#term2.  
def exitTerm2(self, ctx):  
 print("Term2")  
  
 if(self.error=="no"):  
 second\_operator = self.stack.pop()  
 first\_operator = self.stack.pop()  
  
 if(first\_operator=="String"):  
 self.error = "yes"  
 self.result = "Type error: String cannot be divided"  
 elif(first\_operator=="Integer"):  
 if(second\_operator=="Integer" or second\_operator=="Float"):  
 self.result = "Float"  
 self.stack.append(self.result)  
 else:  
 self.error = "yes"  
 self.result = "Type error: String cannot be divided"  
 elif(first\_operator=="Float"):  
 if (second\_operator == "Integer" or second\_operator == "Float"):  
 self.result = "Float"  
 self.stack.append(self.result)  
 else:  
 self.error = "yes"  
 self.result = "Type error: String cannot be divided"  
  
 print("result = ", self.result)  
 print("stack = ", self.stack)  
 pass

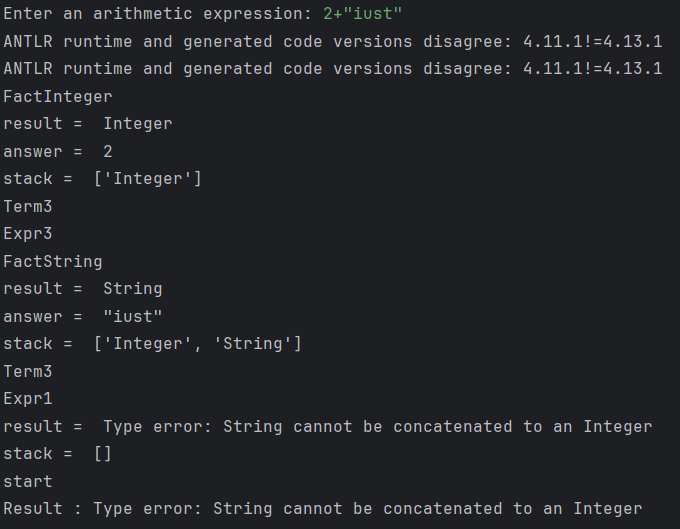
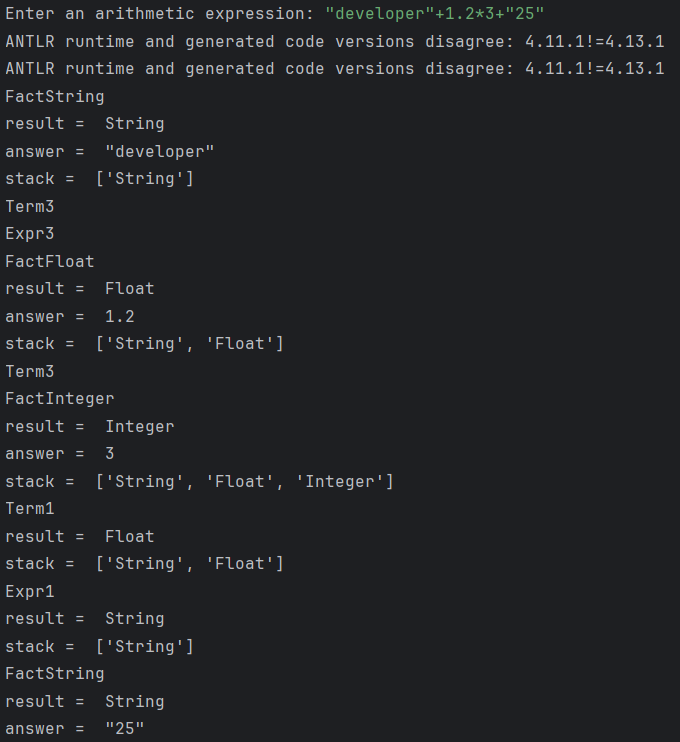
1. در مرحله آخر نیز به عنوان نتیجه گیری مثال های موجود را بررسی کردیم و برای هر کدام نه تنها نتیجه نهایی بلکه مراحل میانی را هم نشان داده ایم.

* 2+3\*1.2
* “hello”+2
* “compiler”\*2



* “students”/1.5



* 2+3\*4
* 2+”iust”
* “developer”+1.2\*3+”25”

