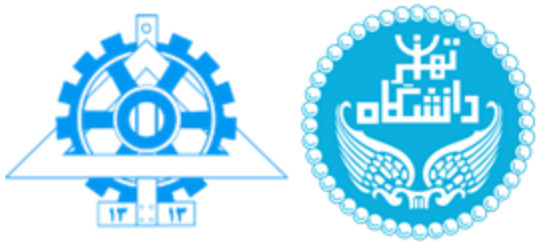

به نام خدا

تمرین کامپیوتری دوم درس طراحی کامپایلر

بهار ۱۴۰۳

مهلت تحویل: ۱۴۰۳/۰۲/۱۵

طراحان : محمدجواد پسرکلو

فهرست مطالب

مقدمه

در فاز اول پروژه، تحلیل‌گر لغوی و نحوی (lexer and parser) زبان FunctionCraft را پیاده‌سازی کردید. همانطور که در درس مطرح شد، فاز بعدی کامپایلر تحلیل‌گر معنایی (semantic analyzer) است. در این فاز از پروژه از شما انتظار می‌رود یک name analyzer برای زبان طراحی کنید. برای این کار لازم است موارد زیر انجام شود:

- درخت نحو انتزاعی (Abstract Syntax Tree) را با قرار دادن قواعد معنایی در گرامر بسازید. توجه کنید که AST به صورت کلاس‌های جاوا در پکیج main.ast.nodes در اختیار شما قرار گرفته است. برای این قسمت، توصیه می‌شود ابتدا کلاس های مربوط به ast nodes را به دقت بررسی کنید، سپس با قرار دادن قواعد معنایی مناسب، attribute های لازم برای تشکیل دادن این node ها را از گرامر حین parse استخراج کنید.
- با پیمایش AST، اطلاعات مربوط به توابع، pattern ها و متغیرها را در جدول علائم (Symbol Table) ذخیره کنید. برای این کار لازم است تا متد visit از visitor interface در ویزیتور NameAnalyzer، برای همه node ها override شود و بررسی‌های لازم برای هر node انجام شود. در صورت تشخیص خطا، از کلاس خطای مربوطه، یک instance به آرایه nameErrors اضافه کنید؛ توجه کنید که خطاهای name analysis به صورت کلاس‌های جاوا در پکیج main.compileError.nameErrors در اختیار شما قرار گرفته است.
- در گام آخر برای node های لازم از ویزیتور DependencyDetector، متد visit را override کنید (برای تشخیص خطای وابستگی حلقوی که در بخش بعد توضیح داده شده است).

برای داشتن درک بهتر از نحوه کارکرد کد، می‌توانید در مورد visitor pattern مطالعه کنید. همچنین برای دیدن نمونه کارکرد visitor pattern، می‌توانید ویزیتور AstPrinter از پکیج main.visitor.astPrinter را بررسی کنید. این ویزیتور پیمایش preorder درخت AST را چاپ می کند و می‌توانید از آن برای debug کردن برنامه خود استفاده کنید؛ بدین صورت که مسیرهایی که در AST طی شده را مشاهده کنید.

خطاهای name analysis:

برنامه زیر را در نظر بگیرید:

```
pattern fib(n)
    | (n == 0) = 1
    | (n == 1) = 1
    | (n > 2) = fib.match(n-1) + fib.match(n-2)
;

pattern fib(n)
    | (n == 0) = 2
    | (n == 1) = 3
    | (n > 2) = fib.match(n-1) + fib.match(n-2)
;

def main()
    fib_5 = fib.match(5);
end
```

در این برنامه fib pattern دوبار تعریف شده و خطای زیر ایجاد می‌شود:

```
Line:7-> Redefinition of pattern fib
```

برای آشنایی بیشتر شما با نحوه کار کردن با Item ها و SymbolTable، کد مربوط به خطای Redefinition of pattern به شما داده شده است. خطاهایی که انتظار می‌رود پیاده‌سازی کنید به شرح زیر است:

1-تعریف دوباره توابع (redefinition of functions)

در scope برنامه، تعریف توابع مختلف با نام‌های یکسان مجاز نیست. مثال:

```
1   def f()
2       return;
3   end
4
5
6   def f(a, b, c)
7       return a+b+c;
8   end
9
10
11  def main()
12      puts("Salam!");
13  end
```

خروجی:

```
1   Line:6-> Redefinition of function f
```

2-استفاده از متغیر تعریف نشده (variable not declared)

اگر در scope هر تابع یا pattern، از متغیری استفاده می‌شود، حتما باید آرگومان تابع یا pattern باشد، یا اینکه با استفاده از assignment، تعریف شده باشد. مثال:

```
1  def f(b, c)
2      a = 5;
3      return a+b+c+d;
4  end
5
6
7  def main()
8      puts("Salam!");
9  end
```

خروجی:

```
1  Line:4-> variable d is not declared
```

3-استفاده از تابع تعریف نشده (function not declared)

اگر در تابعی function call داشته باشیم، تابع call شده باید حتما تعریف شده باشد (توجه کنید که لزومی ندارد که تابع call شده، حتما قبل از تابع call کننده تعریف شده باشد). مثال:

```
1
2  def f(b, c)
3      return g(multiply(b, c));
4  end
5
6  def multiply(a, b)
7      return a*b;
8  end
9
10 def main()
11     puts("Salam!");
12 end
```

خروجی:

```
1   Line:3-> function g is not declared
```

4-یکسان بودن نام تابع با آرگومان

در تعریف توابع، نمی‌توان از آرگومانی هم‌نام با اسم تابع استفاده کرد. مثال:

```
1
2  def compare(compare, b)
3      return compare > b;
4  end
5
6  def main()
7      a = 10;
8      b = 5;
9      compare(a, b);
10 end
```

خروجی:

```
1  Line:2-> argument compare has same name with function
```

5-عدم استفاده از تعداد صحیح آرگومان ورودی در هنگام فراخوانی توابع

تعداد آرگومان‌هایی که هنگام فراخوانی تابع استفاده می‌شود، باید با تعداد پارامترها در تعریف آن تابع همخوانی داشته باشد. توجه کنید که در FunctionCraft، می‌توان برای آرگومان تابع یک default value در نظر گرفت؛ پس به این نکته دقت کنید که در هنگام فراخوانی تابع، وجود آرگومان دارای default value الزامی نیست. مثال:

```
1
2   def f(a, b, [c = 2])
3       return a + b + c;
4   end
5
6
7   def g(a, b)
8       return;
9   end
10
11  def main()
12      a = 10;
13      b = 5;
14      c = 10;
15      f(a, b);
16      f(a, b, c);
17      f(a);
18      g(a);
19      g(a, b);
20  end
```

خروجی (برای تحلیل خروجی این بخش به شماره خط خطا دقت کنید):

```
1   Line:17-> number of arguments provided for function f does not match with its declaration
2   Line:18-> number of arguments provided for function g does not match with its declaration
```

6-وجود وابستگی حلقوی در تعریف توابع

فراخوانی توابع نباید حلقه ایجاد کند. مثلا در صورتی که در تعریف تابع f، تابع g فراخوانی شود و در تعریف تابع g، تابع f فراخوانی شود، یک حلقه ایجاد می‌شود. مثال:

```
1
2  def f()
3      return g();
4  end
5
6
7  def g()
8      return h();
9  end
10
11 def h()
12     return f();
13 end
14
15 def main()
16     puts("Loop!");
17 end
```

خروجی:

```
1  *-> defenition of functions f, h, g contains circular dependency
```

نکات مهم

- تمامی فایل‌ها و کدهای خود را در یک فایل فشرده به صورت studentID1_studentID2.zip آپلود نمایید.
- در صورت کشف هر گونه تقلب، نمره صفر لحاظ می‌شود.
- دقت کنید که خروجی‌ها به صورت خودکار تست می‌شوند؛ پس نحوه چاپ خروجی باید عیناً مطابق موارد ذکر شده در بالا باشد. علاوه بر آن، درخت parse ساخته شده نیز مورد بررسی قرار می‌گیرد.
- بهتر است سوالات خود را در فروم درس یا در گروه اسکایپ مطرح نمایید تا دوستانتان نیز از آنها استفاده کنند؛ در غیر این صورت به مسئولان پروژه ایمیل بزنید.