«به نام خدا»

کامپایلر زبان جایتون

پروژه درس اصول طراحی کامپایلر

مقدمه

هدف از این پروژه طراحی کامپایلر زبان جایتون می باشد. طراحی این کامپایلر به صورت فاز به فاز پیش خواهد رفت بنابراین فاز های بعدی ادامه همین قسمت خواهند بود. سند زبان جایتون در فایل ضمیمه در اختیار شما قرار گرفته است. در این فاز از شما انتظار می رود پس از مطالعه سند این زبان و آشنایی با قواعد آن، برای یک ورودی که قطعه کدی به زبان جایتون است خروجی مورد نظر که توضیحات آن در ادامه است را تولید نمایید. فاز یکم پروژه صرفا جهت آشنایی شما با قواعد زبان جایتون و ابزار ANTLR و فراگیری چگونگی خروجی گرفتن از توابع طراحی شده است و بسیار ساده می باشد.

توضیحات

با توجه به ویدیویی که در اختیارتان قرار داده شده است به راه اندازی اولیه پروژه بپردازید. در این ویدئو چگونگی عملکرد گرامر ها و طرز کار با listener ها نیز توضیح داده شده است.

با توجه به ویدئو شما باید پس از ایمپورت کردن یک قطعه کد جایتون، با استفاده از Listener ها یک خروجی تولید نمایید. این خروجی نمایگر اجزای مختلف قطعه کد ورودی و جزئیات آن است.

شکل کلی خروجی مورد نظر به صورت زیر است. مواردی که داخل [ ] قرار ندارند نشان دهنده اجزای مختلف یک برنامه در حالت کلی می باشد (کلاس، اینترفیس، متغیر و ...) و باید عینا در خروجی نوشته شوند. موارد داخل [ ] وابسته به قطعه کد ورودی می باشد و در واقع توضیحی برای هر جزء هستند(نام کلاس ها، نام اینترفیس ها، نام متغیرها، نوع متغیر ها و.....) که باید توسط شما با توجه به قطعه کد ورودی تکمیل شوند. کد های خروجی شما تست خواهند شد بنابراین حتما مطابق فرمت داده شده خروجی را تعیین کنید، در غیر این صورت بخش زیادی از نمره را از دست خواهید داد.

```
program start {
        [program body]
}
import class: [class name]
class: [class name]/ class parents: ([parent name], )*{
        [class body]
}
class constructor: [constructor name]{
        parameters list: [  ([[parameter type] [parameter name]], )+])?
        [method body]
}
main method{
        parameters list: [  ([[parameter type] [parameter name]], )+])?
        [method body]
}
class method: [method name]/ return type=[return type]{
        parameters list: [  ([[parameter type] [parameter name]], )+])?
        [method body]
}
field: [field name]/ type=[type]
nested statement{
}
```

در ادامه یک نمونه ورودی و خروجی برای درک بهتر آورده شده است.

Input:

```
import Nothing
import Nothing2
class Human(Nothing, Nothing2){
    Nose nose
    Hand[2] hands
    Leg[2] legs
    int calories
    bool isHungry

    def Human(Nose n){
        self.nose = n
    }
    def Voice speak(){
        Voic voice
        return voice
    }
    def void eat(Food food, int c){
        calories += c
        newFood = food
        while(self.isHungry){
            Food newFood = Food()
            eat(newFood)
            self.isHungry = self.checkIsHungry()
        }
    }
    def bool checkIsHungry(){
        return true
    }
}
```

Output:

```
program start{
    import class: Nothing1
    import class: Nothing2
    class: Human/ class parents: object, {
        field: nose/ type= Nose
        field: legs/ type= Leg
        field: hands/ type= Hand
        field: calories/ type= int
        field: isHungry/ type= bool
        class constructor: Human{
```

```
            parameter list: [Nose n]
        }
        class method: speak/ return type: Voice{
            field: voice/ type= Voice
        }
        class method: eat{
            parameter list: [Food food, int c, ]
        }
        class method: checkIsHungry/ return type: bool{

        }
    }
    class: Woman/ class parents: Human, {
        field: age/ type= int
        class constructor: Woman{
            parameter list: [int a]
        }
        main method{

        }
    }

}
```

توجه داشته باشید از شما خواسته شده است همانند مثال بالا دندانه گذاری (Indentation) بلاک های کد را در خروجی برآورده سازید. به این معنی که خطوط خروجی می بایستد با توجه جایگاهشان در ساختار کد با فاصله مناسب از ابتدای خط چاپ شوند. هر indent level چهار عدد space می باشد.

موفق باشید.

تیم حل پروژه: الهه متقین، عادل جلال احمدی