«به نام خدا»

کامپایلر زبان Toorla

پروژه درس اصول طراحی کامپایلر

مقدمه

هدف از این پروژه طراحی کامپایلر زبان Toorla می باشد. طراحی این کامپایلر به صورت فاز به فاز پیش خواهد رفت بنابراین فاز های بعدی ادامه همین قسمت خواهند بود. سند زبان Toorla 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 Toorla است خروجی مورد نظر که توضیحات آن در ادامه است را تولید نمایید. فاز یکم پروژه صرفا جهت آشنایی شما با قواعد زبان Toorla 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 Toorla، 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.

توجه کنید لازم است تا یک نمونه فایل ورودی به منظور تست کد ارائه شده خود آماده کنید. این قسمت بخشی از نمره را در زمان تحویل پروژه شامل می شود.

```
program start {
        [program body]
}
class: [class name]/ class parent: [parent name]/ isEntry: boolean{
        [class body]
}
class constructor: [constructor name] / type: public | privaite{
        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] / type: public | privaite {
        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:

```
class Operator inherits Test:
    private field result int;
    public function Operator() returns int:
        return 1;
    end

    public function subtractor(a:int , b:int) returns int:
        self.result = a - b;
        return self.result;
    end

    public function arrCollector(arr:int[]) returns int:
        int counter = 0;
        while(counter < arr.length):
            self.result = self.result + arr[counter];
        end
        return self.result;
    end

    public function comparator(a:int , b: int) returns string:
        if(a<b)
            int alaki = 3;
            if(a<0)
                print("a is negative")
            return "a is bigger than b";
        elif(a>b)
            if(b<0)
                print("b is negative");
                return "b is bigger than a";
            else
                return "a and b are equal";
            end
         end
```

```
entry class MainClass:
    function main() returns int:
        int a = 5;
        int b = 6;
        arr = new int[4];
        int sum;
        int sub;
        string bigger;
        operator = new Operator();
        sub = operator.subtractor();
        sum = operator.arrCollector(arr);
        bigger = operator.comparator(a,b);
        return 1;
    end
end
```

Output:

```
program start{
    class: Operator / class parent: Test / isEntry: false {
        field: result / type: int
        class constructor: Operator / return type: int/ type: public{
            parameter list: []
        }
        class method: subtractor / return type: int/ type: public {
            parameter list: [type: int / name: a, type: int / name: b]
        }
        class method: arrCollector / return type: int/ type: public {
            parameter list: [type: int[] / name: arr]
        }
        class method: comparator / return type: string/ type: public {
            parameter list: [type: int / name: a, type: int / name: b]
             nested {
                    field: alaki / type: int
                    nested{
                     }
                     nested{
                     }
              }
         }
    }
```

```
    class: MainClass / class parents: none / isEntry: true {
        main method / type: int {
            field: a / type: int
            field: b / type: int
            field: arr / type: int[]
            field: sum / type: int
            field: sub / type: int
        }
    }
}
```

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، محمدرضا تشکری، امیرعلی وجدانی فرد