پروژه درس اصول طراحی کامپایلر(فاز اول)

دکتر امین طوسی

دستیاران آموزشی: نجمه حبیبی، نسترن احمدی بنکدار

مقدمه

هدف از این پروژه طراحی کامپایلر زبان Cool می باشد . طراحی این کامپایلر به صورت فاز به فاز پیش خواهد رفت بنابراین فاز های بعدی ادامه همین قسمت خواهند بود . 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 Cool است خروجی مورد نظر که توضیحات آن در ادامه است را تولید نمایید .فاز یکم پروژه صرفا جهت آشنایی شما با قواعد زبان Cool و ابزار ANTLR فراگیری چگونگی خروجی گرفتن از توابع طراحی شده است و بسیار ساده می باشد.

توضیحات

- بخش اول:

با توجه به ویدئویی که در اختیارتان قرار داده شده به راه اندازی اولیه پروژه بپردازید. در این ویدئو چگونگی عملکرد گرامر ها و طرز کار با listener ها نیز توضیح داده شده است. (توجه فرمایید ویدئو مربوط به پروژه ترم های گذشته می باشد)

با توجه به ویدئو شما باید پس از ایمپورت کردن یک قطعه کد Cool، 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]
}

main class: [class name]{
}
```

```
class: [class name]/ class parents: ([parent name], )*{
        [class body]
}

field: [field name]/ type=[type]

class method: [method name]/ return type=[return type]{
        (parameters list= [([[parameter type] [parameter name]], )+])?
                [method body]
}

nested statement{
}
```

راهنمایی:

* : باعث مطابقت عبارت منظم با تعداد تکرار 0 و یا بیشتر از بخش مورد نظر از عبارت منظم می شود. برای مثال اگر داریم ab* ، این عبارت منظم با a ، ab ، abb و ... مطابقت دارد.

+ : باعث مطابقت عبارت منظم با تعداد تکرار 1 و یا بیشتر از بخش مورد نظر از عبارت منظم می شود. برای مثال اگر داریم ab+ ، این عبارت منظم با ab ، abb و ... مطابقت دارد. (حداقل یک b لازمه تطابق است.)

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

ورودی:

```
class A {

    var : Int <- 0;
    set_var(num : Int) : SELF_TYPE {
        {
            var <- num;
            self;
        }
    };
    method2(num1 : Int, num2 : Int) : B {  -- plus
        (let x : Int in
            {
                x <- num1 + num2;
                (new B).set_var(x);
            }
        )
    };
};
```

```
class B inherits A {  -- B is a number squared

    method5(num : Int) : C { -- square
        (let x : Int in
            {
                x <- num * num;
                (new C).set_var(x);
            }
        )
    };
};

class C inherits B {

    method6(num : Int) : B { -- negate
        (let x : Int in
            {
                x <- ~num;
                (new B).set_var(x);
            }
        )
    };
};

class Main inherits IO {

    char : String;
    avar : A;
    flag : Bool <- true;

    main() : Object {
        {
            avar <- (new A);
            char <- "a";
            while flag loop
                {
                    if char = "j" then
                        avar <- (new A)
                    else
                        avar <- (new B)
                    fi;
                }
            pool;
```

```
        }
    };
};
```

خروجی:

```
program start{
    class: Main/ class parents: IO, {
        field: char/ type=String
        field: avar/ type=A
        field: flag/ type=Bool

        class method: main/ return type=Object{
            nested statement{
            }
        }
    }
    class: A/ class parents: object, {
        field: var/ type=int
        class method: set_var/ return type=SELF_TYPE{
            parameters list= [int num, ]
        }
        class method: method2/ return type=B{
            parameters list= [int num1, int num2, ]
            field: x/ type=int
        }
    }
    class: B/ class parents: A, {
        class method: method5/ return type=C{
            parameters list= [int num ]
            field: x/ type=int
        }
    }
    class: C/ class parents: B, {
        class method: method6/ return type=B{
            parameters list= [int num ]
            field: x/ 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.

- بخش دوم:

درخت ارث بری برنامه ورودی را ساخته و آن را به روش دلخواهتان نمایش دهید. (ساده ترین راه حل می تواند نمایش درخت از طریق print کردن آن باشد.)

این نکته که در زبان Cool، اگر کلاسی دارای کلاس پدر نبود و از چیزی ارث بری نمی کرد، پدر آن کلاس، کلاس object است را از یاد نبرید! (داخل داک توضیحات کامل آن موجود است.)

شاد و پیروز باشید.