پروژه درس اصول طراحی کامپیوتر(فاز دوم و سوم)

دکتر امین طوسی

تدریس یاران : الهه متّقین، محمدرضا تشکری، امیرعلی وجدانی فرد

مهلت تحویل : ۱۴۰۱/۱۱/۱۲

مقدمه :

هدف این فاز : ۱) پیاده‌سازی تحلیل‌گر معنایی ۲)تشخیص خطا می‌باشد.

در مرحله اول اطلاعاتی را جمع آوری و در جدول علائم ذخیره می‌کنیم و در آخر جدول را نمایش می‌دهیم.

در مرحله دوم خطاها را توسط تحلیل‌گر معنایی بررسی و سپس چاپ می‌کنیم.

توضیحات:

- جدول علائم (Symbol Table)

ساختار داده‌ای است که برای نگه‌داری شناسه های(علائم) تعریف شده در کد ورودی استفاده می‌شود.

- طراحی جدول علائم:

برای طراحی این جدول می‌توان از روش‌های مختلفی(List, Linked List, Hash Table, …) استفاده کرد که با توجه به نیاز، نوع زبان، پیچیدگی و نظر طراح انتخاب می‌شود.

ساده‌ترین نوع پیاده‌سازی این جدول استفاده از Hash Table می‌باشد. به این‌صورت که key آن نام شناسه و value آن مقدار(مجموعه مقادیر) ویژگی‌های مربوط به شناسه است.

هر جدول علائم دو متد اصلی دارد که اطلاعات مربوط به شناسه از طریق این دو متد در جدول ذخیره یا از جدول بازیابی می‌شوند.

```
insert (idefName, attributes)
lookup (idefName)
```

در زبان Toorla هر Scope یک جدول علائم مخصوص به خود دارد.

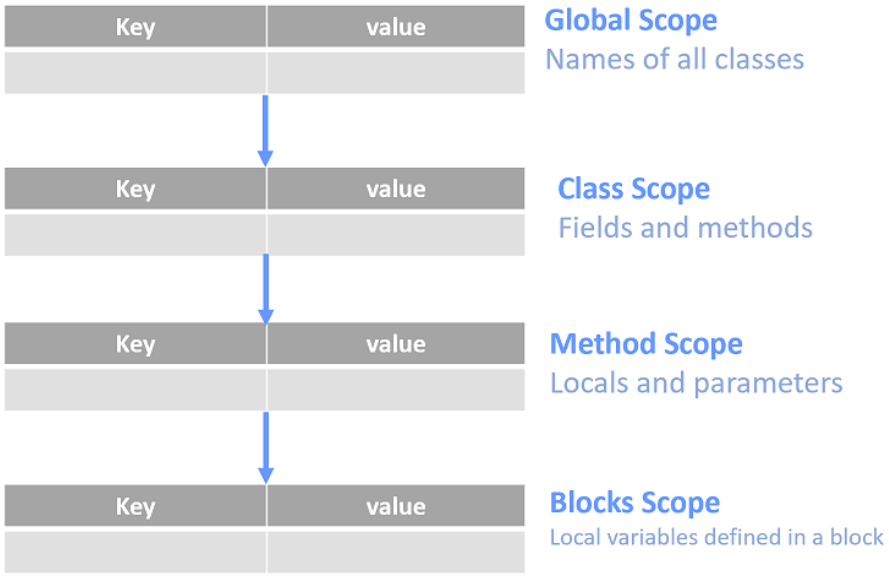
- Scopes:

هر یک از موارد زیر در زبان Toorla یک اسکوپ به حساب می‌آیند:

- تعریف برنامه
- تعریف کلاس
- تعریف constructor
- تعریف متد
- ساختار تصمیم گیری(شروع if و elif و else)
- ساختار تکرار(while و for)

اسکوپ ها و جداول علائم (صرفا جهت اطلاع)

همانطور که پیش‌تر گفته شد، هر اسکوپ شامل یک جدول علائم می‌باشد. بنابرین علائمی(شناسه‌هایی) که در هر اسکوپ تعریف می‌شوند در جدول علائم این اسکوپ ذخیره می‌شوند. از آنجایی که اسکوپ‌ها می‌توانند تو در تو باشند، جداول علائم اسکوپ ها با یکدیگر رابطه درختی دارند.



در این دو فاز چه باید انجام دهیم؟

فاز دوم:

در این فاز ابتدا چند برنامه به زبان Toorla بنویسید؛ سپس هر قطعه کد را به عنوان ورودی دریافت و اسکوپ‌های آن را پردازش کنید و جدول علائم مربوط به آن را بسازید و همه جداول را در یک خروجی و به ترتیب شماره خط شروع اسکوپ چاپ کنید. در ادامه مثالی از ورودی و خروجی به زبان Toorla آمده است.

Input:

```
class Operator inherits Test:
    private field result int;
    public function Operator(a: int) 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:
         var counter = 0;
         while(counter < arr.length)
             self.result = self.result + arr[counter];
         return self.result;
     end
    public function comparator(a:int , b: int) returns string:
        if(a<b)
        begin
            var alaki = 3;
            return "b is bigger than a";
        end
        elif(a>b)
        begin
            if(b<0)
                print("b is negative");
            return "a is bigger than b";
        end
        else
            return "a and b are equal";
    end
end
entry class MainClass:
    function main() returns int:
        var a = 5;
        var b = 6;
        arr = new int[4];
        var sum=0;
        var bigger="";
        operator = new Operator();
        sub = operator.subtractor();
        sum = operator.arrCollector(arr);
        bigger = operator.comparator(a,b);
        return 1;
    end
end
```

Output:

```
---------program: 1 ---------
Key: Class_ Operator | Value : Class (name: Operator) (parent: Test) (isEntry:
False)

Key: Class_ MainClass | Value : Class (name: MainClass) (parent: []) (isEntry:
True)


================================================================================


--------- Operator: 1 ---------
Key : Field_result | Value : ClassField (name: result) (type: int, isDefiend:
True)

Key : Constructor_Operator | Value : Constructor (name : Operator) (return type:
[int]) (parameter list: [name: a, type: int, index: 1])

Key : Method_ subtractor | Value : Method (name : subtractor) (return type:
[int]) (parameter list: [name: a, type: int, index: 1], [name: b, type: int,
index: 2])

Key : Method_ arrCollector | Value : Method (name : arrCollector) (return type:
[int]) (parameter list: [name: arr, type: int[], index: 1])


Key : Method_ comparator | Value : Method (name : comparator) (return type:
[string]) (parameter list: [name: a, type: int, index: 1], [name: b, type: int,
index: 2])


================================================================================

--------- Operator: 3---------
Key : Field_a | Value : ParamField (name: a) (type: int, isDefiend: True)


================================================================================

--------- subtractor: 6---------
Key : Field_a | Value : ParamField (name: a) (type: [ classtyped= int, isDefiend:
True)

================================================================================

--------- arrCollector: 10---------
Key : Field_arr | Value : ParamField (name: arr) (type: [ classtyped= int[],
isDefiend: true)

Key : Field_ counter | Value : MethodVar (name: counter) (type: [ loacalVar=
int, isDefiend: True)


================================================================================

--------- comparator: 16 ---------
Key : Field_a | Value : ParamField (name: a) (type: int, isDefiend: True)
```

```
Key : Field_b | Value : ParamField (name: btest) (type: int, isDefiend: True)


--------- while: 12 ---------


================================================================================

--------- if: 17 -----------
Key : Field_ alaki | Value : MethodVar (name: counter) (type: [ loacalVar= int,
isDefiend: True)

================================================================================

--------- nested: 24 -----------


================================================================================

--------- MainClass: 32 -----------
Key : Method_ main | Value : Method (name : main) (return type: [int]) (parameter
list: []

================================================================================

--------- MainClass: 33 -----------

Key : Field_ a | Value : MethodVar (name: a) (type: [ loacalVar= int, isDefiend:
True)

Key : Field_ b | Value : MethodVar (name: b) (type: [ loacalVar= int, isDefiend:
True)

Key : Field_ arr | Value : MethodVar (name: arr) (type: [ loacalVar= int[],
isDefiend: True)

Key : Field_ sum | Value : MethodVar (name: sum) (type: [ loacalVar= int,
isDefiend: True)

Key : Field_ bigger | Value : MethodVar (name: bigger) (type: [ loacalVar=
string, isDefiend: True)

Key : Field_ Oprator | Value : MethodVar (name: Oprator) (type: [ loacalVar=
Classtype:[Oprator], isDefiend: True)
```

مراحل گرفتن خروجی :

۱. برای هر SymbolTable باید دو تابع زیر فراخوانی شوند. تابع toString برای چاپ کردن مقادیر symbolTable و تابع getValue برای دریافت مقادیر از Hashmap استفاده می‌شوند.

```
 public String toString() {

  return "-------------  " + name + " : " + scopeNumber + "  -------------\n" +
            printItems() +
           "----------------------------------------\n";
}
```

```
public String printItems(){
   String itemsStr = "";
   for (Map.Entry<String,SymbolTableItem> entry : items.entrySet()) {
       itemsStr += "Key = " + entry.getKey() + "  | Value = " + entry.getValue()
+ "\n";
   }
   return itemsStr;
}
```

۲. برای چاپ هر item نیز باید متد toString بنویسیم.

فرمت مثال زده شده صرفا یک نمونه فرمت قابل قبول برای خروجی زبان Toorla می باشد و دیگر فرمت‌های خوانا، مرتب و نمایش‌دهنده تمام اجزای هر بخش قابل قبول می‌باشند. (اگر فرمت شما خلاقانه،مرتب و بسیار کامل باشد و به طور کاملا واضح و زیبا نمایانگر تمام اجزا جدول علائم اسکوپ باشد می‌تواند شامل نمره اضافه شود.)

نکات:

- شماره خط شروع هر اسکوپ را در ابتدا به همراه نام آن نمایش دهید:

```
-------------  Base : 18  -------------
```

- در صورت خالی بودن یک جدول باز هم نیاز به نمایش دادن آن می‌باشد:

```
------------  nested : 39  ------------
----------------------------------------
```

- در هنگام ذخیره سازی هر یک از اجزا در Symbol table نیاز است نوع آن را در کنار نام آن ذخیره کنید به عنوان مثال در قطعه کد زیر نیاز است کلاس Base را به صورت class_Base و متد set را به صورت method_setدر قسمت key ذخیره کنید.

```
class Base{
  private int set() {
  }
}
```

** توجه کنید تشخیص نوع متغیرهای محلی از نوع Classtype دارای نمره اضافه است اما تشخیص نوع سایر انواع از پیش تعیین شده از متغیرهای محلی مانند int، string، []int و ... الزامی است.

فاز سوم:

در این فاز می‌خواهیم با استفاده از جدول علائم به بررسی خطاهای معنایی موجود در برنامه بپردازیم.

فرمت گزارش خطا:

خطاهای موجود در برنامه را بر اساس فرمت زیر گزارش دهید:

line شماره خط ارور و column مکان آن را در یک خط نشان می‌دهد.

شما باید دو نوع خطایی که در ادامه آورده شده است پیاده‌سازی کنید.

۱. **خطای تعریف دوباره متد/خصیصه/ کلاس/ متغیر محلی**

- تعریف دوباره کلاس:

```
Error۱۰۱: in line [line:column] , method [name] has been defined already
```

- تعریف دوباره متد:

```
Error۱۰۲ : in line [line:column] , method [name] has been defined already
```

- تعریف دوباره خصیصه:

```
Error۱۰۳ : in line [line:column], field [name] has been defined already
```

- تعریف دوباره یک متغیر محلی در یک حوزه:

```
Error۱۰۴ : in line [line:column], var [name] has been defined already
```

- نکته: دو نوع متفاوت میتوانند هم نام باشند به عنوان مثال اگر یک فیلد و متد هم اسم باشند مشکلی نیست.
- نکته: در صورت تعریف دوباره یک کلاس، متد ویا فیلد اسم آن را عوض میکنیم و به سیمبل تیبل اضافه میکنیم و اسم آن را به این صورت ذخیره میکنیم: name_line_column .

  بعنوان مثال اگر متغیر d دوباره تعریف شود آن را به صورت d_۳۴_۴۸ ذخیره می نماییم.

- نکته: هر کدام از موارد ذکر شده اگر دوبار تعریف شوند مورد دوم مطرح نیست و فرض میکنیم اصلا وجود ندارد و تنها ازمورد اول استفاده میشود. به عنوان مثال اگر یک کلاس دوبار تعریف شده باشد تنها میتوان از کلاس اول استفاده کرد.

۲. خطاهای مربوط به ارث بری

- وجود دور در ارث بری :

```
Error410 : Invalid inheritance [classname1] -> [classname2] -> [classname3] …
```

۳. نمره اضافه

- عدم تطابق نوع بازگشتی متد با نوع بازگشتی تعریف شده توسط متد:

```
Error210 : in line [line:column], ReturnType of this method must be [MethodReturnType]
```

- متدهای پرایوت یک کلاس در کلاس های دیگر قابل دسترسی نمی باشد.

```
Error310: in line [line:column], private methods are not accessible outside of class.
```

موفق باشید