«به نام خدا»

کامپایلر زبان mini java

پروژه درس اصول طراحی کامپایلر

مقدمه

هدف از این پروژه، طراحی یک کامپایلر ساده برای زبان mini Java است. طراحی این کامپایلر به صورت فاز به فاز پیش خواهد رفت.زبان Mini Java زیرمجموعه و نسخه ساده شده ای از زبان Java میباشد و دستورات و قواعد آن بسیار شبیه Java است.

از آنجایی که شما در درس نظریه با طراحی گرامر با استفاده از ANTLR آشنا شده اید فایل گرامر mini Java برای شما قرار داده شده است. البته توجه داشته باشید گرامری که در اختیار شما قرار گرفته است نسخه توسعه یافته زبان mini Java است و برخی قواعد به آن اضافه شده است و یا تغییر یافته است.

توجه : فاز بندی های پروژه صرفا به جهت تسهیل پیاده سازی قدم به قدم و نمره دهی دقیق تر انجام شده است.

فاز اول

در این فاز، لازم است که شما یک نمونه ورودی کامل به زبان mini Java آماده کرده و سپس با ایجاد تغییرات در listener ها، این قطعه کد را به زبان Java ترجمه کنید. همانطور که گفته شد گرامر مینی جاوا بسیار شبیه به جاواست اما تفاوت هایی دارد. در این فاز لازم است یک نمونه ورودی کامل آماده کرده و در خروجی با ایجاد تغییرات در listener ها قطعه کد معادل ورودی در زبان جاوا را چاپ کنید.

نکات قابل توجه:

- 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 .

- رعایت دندانه گذاری (Indentation)در کد خروجی بسیار مهم است؛ پس دندانه گذاری بلاک های کد را حتما انجام دهید. هر indent level معادل ۴ عدد Space می باشد.در ادامه یک نمونه ورودی و خروجی برای درک بهتر آورده شده است.

Input:

```
class Pikachu inherits Pokemon {
    @Override
    public number damage() {
    if(<>(health < 2)) {
        health = health - 2;
    }
    print(“new health”);
    ret health;
    }
}
```

Output:

```
class Pikachu extends Pokemon {
    @Override
    public number damage() {
    if(<>(health < 2)){
        health = health - 2;
    }
    System.out.print(“new health”);
    return health;
    }
}
```

فاز دوم و سوم

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 شناسه‌های(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 mini java هر Scope یک جدول علائم مخصوص به خود دارد.

- Scopes:

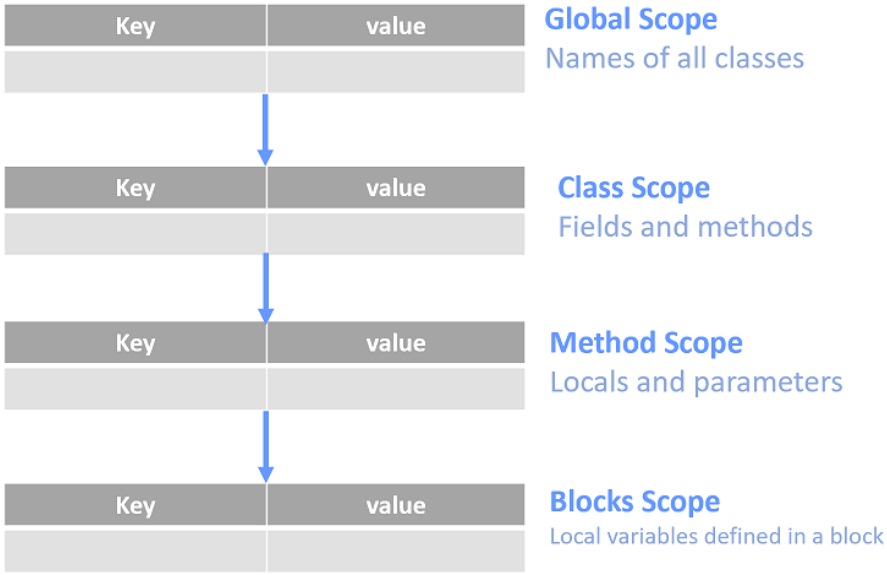
هر یک از موارد زیر در زبان C یک اسکوپ به حساب می‌آیند:

هر یک از موارد زیر زبان MiniJava یک حوزه به حساب می آیند.

- تعریف برنامه
- تعریف کلاس
- تعریف توابع
- شروع while, if , else و nestedStatement ها

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 mini java 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 mini java آمده است.

Input:

```
class Classes {
    public static void main(String[] args) {
        Base b;
        Derived d;
        number x;
        b = new Base();
        d = new Derived();
        b = d;
        print(b.set(1));
        print(b.set(3));
    }
}

class Rest implements Nothing{

}

class Base inherits Rest implements Face,MyFace{
    number[] data;
    private Rest[] d;
    private number set(number[] x) {
        data = x;
        ret data;
    }
    public number get() {
        ret data;
    }
    private number test(){

        while(1)
            number x;

        number b ;
        {
            number t;
            a = 0;
            b = 5;
        }
        if(true){
            boolean n;
            {
            }

        }

        ret a + b;
    }

}

class Derived inherits Base {

    public number set(number x, C d) {
        data = x * 2;
        {
```

```
            Base[] t;

        }
        if (a<0)
            number a;
        ret data;
    }
}

class Hi {
    private void sayHi(number one, boolean two, Hi three, number four){
    }
}

interface Face{
    final number[] a = {1,2};
    number getFace(number s);
}

interface MyFace{
    final String  myFace= ":)";
}
```

Output:

```
-------------  program : 1  -------------
        Key = class Base  | Value = Class: (name: Base) (extends : Rest |
implements: Face, MyFace)
        Key = interface_Face  | Value = Interface: (name: Face)
        Key = interface_MyFace  | Value = Interface: (name: MyFace)
        Key = class_Rest  | Value = Class: (name: Rest) (extends : Object |
implements: Nothing)
        Key = mainClass_Classes  | Value = MainClass: (name: Classes)
        Key = class_Hi  | Value = Class: (name: Hi) (extends : Object)
        Key = class_Derived  | Value = Class: (name: Derived) (extends :
Base)
        ----------------------------------------

        -------------  Classes : 1  -------------
        Key = method_main  | Value = Method: (name: main) (returnType: void)
(accessModifier: ACCESS_MODIFIER_PUBLIC) (parametersType: [array of
[classType = String, isDefined = true] , index: 1] )
        ----------------------------------------

        -------------  mainMethod : 2  -------------
        Key = var_d  | Value = LocalVar: (name: d) (type: [classType =
Derived, isDefined = true])
        Key = var_b  | Value = LocalVar: (name: b) (type: [classType = Base,
isDefined = true])
        Key = var_x  | Value = LocalVar: (name: x) (type: int)
        Key = var_args  | Value = Parameter: (name: args) (type: array of
[classType = String, isDefined = true]) (index: 1)
        ----------------------------------------
```

```
        -------------  Rest : 14  -------------
        ----------------------------------------

        -------------  Base : 18  -------------
        Key = var_d  | Value = Field: (name: d) (type: array of [classType =
Rest, isDefined = true]) (accessModifier: ACCESS_MODIFIER_PRIVATE)
        Key = method_get  | Value = Method: (name: get) (returnType: int)
(accessModifier: ACCESS_MODIFIER_PUBLIC)
        Key = method_set  | Value = Method: (name: set) (returnType: int)
(accessModifier: ACCESS_MODIFIER_PRIVATE) (parametersType: [array of int ,
index: 1] )
        Key = var_data  | Value = Field: (name: data) (type: array of int)
(accessModifier: ACCESS_MODIFIER_PUBLIC)
        Key = method_test  | Value = Method: (name: test) (returnType: int)
(accessModifier: ACCESS_MODIFIER_PRIVATE)
        ----------------------------------------

        -------------  set : 21  -------------
        Key = var_x  | Value = Parameter: (name: x) (type: array of int)
(index: 1)
        ----------------------------------------

        -------------  get : 25  -------------
        ----------------------------------------

        -------------  test : 28  -------------
        Key = var_b  | Value = LocalVar: (name: b) (type: int)
        ----------------------------------------

        -------------  while : 31  -------------
        Key = var_x  | Value = LocalVar: (name: x) (type: int)
        ----------------------------------------

        -------------  nested : 34  -------------
        Key = var_t  | Value = LocalVar: (name: t) (type: int)
        ----------------------------------------

        -------------  nested : 39  -------------
        Key = var_n  | Value = LocalVar: (name: n) (type: bool)
        ----------------------------------------

        -------------  nested : 41  -------------
        ----------------------------------------

        -------------  Derived : 51  -------------
        Key = method_set  | Value = Method: (name: set) (returnType: int)
(accessModifier: ACCESS_MODIFIER_PUBLIC) (parametersType: [int , index: 1]
[[classType = C, isDefined = false] , index: 2] )
        ----------------------------------------

        -------------  set : 53  -------------
        Key = var_d  | Value = Parameter: (name: d) (type: [classType = C,
isDefined = false]) (index: 2)
        Key = var_x  | Value = Parameter: (name: x) (type: int) (index: 1)
        ----------------------------------------
```

```
        -------------  nested : 55  -------------
        Key = var_t  | Value = LocalVar: (name: t) (type: array of [classType
= Base, isDefined = true])
        ----------------------------------------


        -------------  if : 60  -------------
        Key = var_a  | Value = LocalVar: (name: a) (type: int)
        ----------------------------------------
        -------------  Hi : 65  -------------
        Key = method_sayHi  | Value = Method: (name: sayHi) (returnType:
void) (accessModifier: ACCESS_MODIFIER_PRIVATE) (parametersType: [int ,
index: 1]  [bool , index: 2]  [[classType = Hi, isDefined = true] , index: 3]
[int , index: 4] )
        ----------------------------------------


        -------------  sayHi : 66  -------------
        Key = var_four  | Value = Parameter: (name: four) (type: int) (index:
4)
        Key = var_three  | Value = Parameter: (name: three) (type: [classType
= Hi, isDefined = true]) (index: 3)
        Key = var_one  | Value = Parameter: (name: one) (type: int) (index:
1)
        Key = var_two  | Value = Parameter: (name: two) (type: bool) (index:
2)
        ----------------------------------------


        -------------  Face : 70  -------------
        Key = var_a  | Value = Field: (name: a) (type: array of int)
(accessModifier: ACCESS_MODIFIER_PUBLIC)
        Key = method_getFace  | Value = Method: (name: getFace) (returnType:
int) (accessModifier: ACCESS_MODIFIER_PUBLIC) (parametersType: [int , index:
1] )
        ----------------------------------------


        -------------  MyFace : 75  -------------
        Key = var_myFace  | Value = Field: (name: myFace) (type: [classType =
String, isDefined = true]) (accessModifier: ACCESS_MODIFIER_PUBLIC)
        ----------------------------------------
```

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 mini java 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() {
  }
}
```

- مقدار isDefined بیانگر این است که کلاس در برنامه تعریف شده است یانه.

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 پوزیشن آن را در یک خط نشان می‌دهد.

خطاهایی که لازم است بررسی کنید به صورت زیر است:

- وجود دور در ارث بری

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

- در صورت implement کردن یک کلاس باید تمامی متد های این کلاس پیاده سازی شوند :

```
Error420: Class [className] must implement all abstract methods
```

- سطح دسترسی تابع نمی تواند محدودتر از سطح دسترسی تابع override شده باشد:

```
Error320: in line [line:column], the access level cannot be more
restrictive than the overridden method's access level
```

نمره اضافه

- تعداد و نوع پارامترهای متد در هنگام فراخوانی با تعداد پارامترهای رسمی در هنگام تعریف برابر نباشد:

```
Error220: in line [line:column], Mismatch arguments.
```

- در دستورات انتساب نوع عملوند چپ و راست یکی نباشد:

```
Error 230 : in line [line:column], Incompatible types : [LeftType] can not be
converted to [RightType]
```

فاز چهار

در این فاز لازم است به کمک کامپایلر خود کد ورودی را بهینه سازی کنید.

مواردی که باید در نظر گرفته شود مطابق زیر است.

- حذف متغیر استفاده نشده.
- Constant Folding

نمونه اولیه: number a = 3 + 4

نمونه بهینه شده: number a = 7

- Dead Code Elimination:

نمونه اولیه:

```
if (false) {

print("Dead Code");}
```

نمونه بهینه: حذف کد.

نمره اضافه

- Code Motion (Loop Invariant Code Motion):

بیرون گذاشتن از حلقه در صورتی که در حلقه تغییری ایجاد نمی شود.

نمونه اولیه:

```
for (int i = 0; i < n; i++) {

int x = y + z;

a[i] = x * i;

}
```

نمونه بهینه شده:

```
int x = y + z;

for (int i = 0; i < n; i++) {

a[i] = x * i;      }
```

شما می توانید یک نمونه دیگر بهینه سازی به عنوان نمره اضافه و به دلخواه خود انجام دهید. تنها در صورتی نمره اضافه به شما تعلق می گیرد که مورد انتخابی شما به تایید تیم حل پروژه رسیده باشد. پس قبل از پیاده سازی تایید تیم حل پروژه را بگیرید.

موفق باشید.

تیم حل پروژه: عادل جلال احمدی، محمدرضا تشکری، الهه متقین