«به نام خدا»

کامپایلر زبان C

پروژه درس اصول طراحی کامپایلر

مقدمه

هدف از این پروژه طراحی کامپایلر زبان C می باشد. طراحی این کامپایلر به صورت فاز به فاز پیش خواهد رفت بنابراین فاز های بعدی ادامه همین قسمت خواهند بود. گرامر ساده و خلاصه شده زبان C 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 C است خروجی مورد نظر که توضیحات آن در ادامه است را تولید نمایید. فاز یکم پروژه صرفا جهت آشنایی شما با ابزارANTLR 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 C، 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 method{

parameters list: [([[parameter type] [parameter name]],)+])?

Fanction cal: function name/ function params (param index)

[method body]

}

normal method: [method name]/ return type=[return type]{

parameters list: [([[parameter type] [parameter name]],)+])?

Fanction cal: function name/ function params (param index)

[method body]

}

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

nested statement{

}

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

Input:

```
void myFunction(int myNumbers[5], char isOdd) {
  for (int i = 0; i < 5; i++) {
      if(isOdd == ‘y’){
          printf("%d\n", myNumbers[i]);
      }
  }
}

int main() {
```

```
  int myNumbers[5] = {10, 20, 30, 40, 50};
  myFunction(myNumbers);
  return 0;
}
```

Output:

```
program start{
    normal method: name: myFunction/ return type : int {
      parameter list: [myNumbers int, isOdd char]
      nested statement : {
      }
    }
    main method: return type: void(no return) {
        field: myNumbers/ type: int/ length: 5
       function call: name: myFunction/ params: myNumbers (index: 0)
    }
}
```

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.

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.

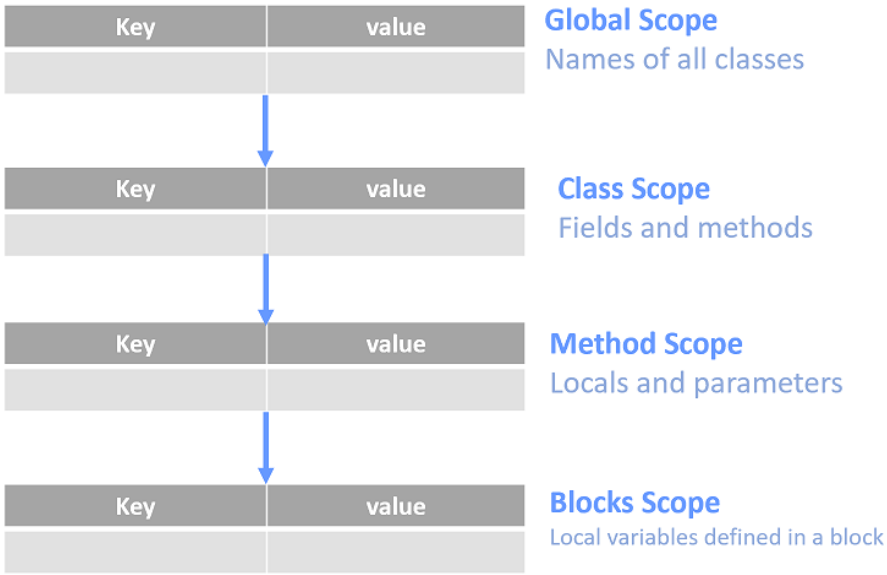
```
insert (idefName, attributes)
lookup (idefName)
```

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

- Scopes:
هر یک از موارد زیر در زبان C یک اسکوپ به حساب می‌آیند:
  - تعریف برنامه
  - تعریف متد main
  - 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 C 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 C آمده است.

Input:

```
1. void myFunction(int myNumbers[5], char isOdd) {
2.  for (int i = 0; i < 5; i++) {
     3.if(isOdd == ‘y’){
     4. float a;
     5.   printf("%d\n", myNumbers[i]);
     6.}
  7.}
8.}

10.int main() {
  11.int myNumbers[5] = {10, 20, 30, 40, 50};
  12.myFunction(myNumbers);
  13.return 0;
14.}
```

Output:

```
---------program: 1 ---------
Key : Method_main | Value : Method (name : main) (return type:
int)
Key : Method_myFunction | Value : Method (name : myFunction)
(return type: void] [parameter list: [type: int array, index:
0], [type: char, index: 2]))


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


---------myFunction: 1---------
Key : Field_myNumbers | Value : methodParamField(name:
myNumbers) (type: int array, length= 5)
Key : Field_isOdd | Value : methodParamField (name : isOdd)
(type : char)

================================================================
---------main: 10---------
Key : Field_myNumbers | Value : methodField(name: myNumbers)
(type: int array, length= 5)

================================================================
---------nested: 3---------

Key : Field_a | Value : MethodField (name: a) (type: int)
```

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 C 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_set در قسمت key ذخیره کنید.

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

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.

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

۱. **خطای تعریف دوباره متد/خصیصه**

- تعریف دوباره متد:

```
Error102 : in line [line:column] , method [name] has been defined already
```

- تعریف دوباره خصیصه:

```
Error104 : in line [line:column], field [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. خطای استفاده از متغیر تعریف نشده:

- استفاده از متغیر یا خصیصه تعریف نشده در اسکوپ:

```
Error106 : in line [line:column], Can not find Variable [name]
```

۳. عدم تطابق نوع بازگشتی

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

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

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]
```

موفق باشید.

تیم حل پروژه: کیارش وثوق، محمدرضا تشکری، الهه متقین