به نام خدا

فاز اول پروژه کامپایلرها و زبان‌های برنامه‌نویسی

پاییز 1400

مهلت تحویل : 17 آبان

در فاز یکم پروژه، شما باید به کمک ابزار ANTLR4 و زبان برنامه‌نویسی جاوا، برای زبان C-- (که سند آن در اختیار شما قرار گرفته است) تحلیلگر لغوی و نحوی بنویسید.

تحلیلگر لغوی[1] :

در این بخش از پروژه، باید به کمک ابزار ANTLR4، تمامی Token های مورد نظر در زبان C-- را مشخص و پیاده‌سازی کنید .

تحلیلگر نحوی[2] :

در این بخش از پروژه، ابتدا با نوشتن قواعد نحوی صحیح، گرامر زبان C-- را به کمک ANTLR4 پیاده سازی می‌کنید. سپس، با اعمال ورودی‌های مورد نظر و با توجه به درخت Parse[3] گرامر خود را تست و اطلاعات خواسته شده را چاپ می‌کنید. بهتر است برای قواعد خود نام‌های مناسب انتخاب کنید تا فهم آن‌ها راحت‌تر باشد. هم چنین توجه داشته باشید که گرامر شما نباید مشکل چپ‌گردی و ابهام داشته باشد.

در این فاز نیازی نیست که هیچ گونه قاعده معنایی [4] را بررسی و پیاده سازی کنید . برای مثال وجود مقدار بازگشتی برای main، وجود دو آرگومان هم‌نام، شرطی بودن عبارت شرطی if،

---

[1] Lexer

[2] Parser

[3] درختی که بر اساس آن قواعد گرامر بررسی میشوند

[4] Semantic rules

ناهماهنگی تایپ‌ها و . . . همگی از مواردی هستند که در فازهای بعدی بررسی می‌شوند و در این فاز نیازی به رسیدگی به آن‌ها نیست.

مواردی که در پیاده سازی گرامر خود باید رعایت کنید، به شرح زیر است:

1) ابتدا ساختمان‌ها، سپس توابع و در انتها main تعریف می شود.
2) برنامه باید داری دقیقا یک main باشد.
3) در هر scope اگر فقط یک گزاره وجود دارد، می‌توان از begin و end استفاده نکرد؛ در غیر این صورت حتما scope باید با این دو کلیدواژه مشخص شود.
4) در هنگام تعریف تابع، بین تعریف تابع و بدنه باید حتما یک خط، فاصله وجود داشته باشد و آخرین گزاره بدنه تابع از بقیه برنامه نیز توسط یک خط جدا می شود. در صورت استفاده از begin و end، کلیدواژه begin باید در همان خط تعریف تابع بیاید و قبل و بعد از کلیدواژه end حتما یک خط، فاصله وجود دارد. برای باقی scopeها نیز به همین صورت است. برای روشن‌تر شدن این موضوع، به مثال زیر توجه کنید:

```
int func1(int arg) begin
   int a = 0; a = a + arg; return a;
end
void func2(int arg)
   if true begin
      arg = arg * arg
   end
main()  begin
   func1(1); func2(2);
end
```

5) در تعریف ساختمان، اگر برای متغیری توابع getter و setter تعریف شود حتما از کلیدواژه‌های begin و end استفاده می‌شود و قوانین برای آن دو تابع همانند مثال بالا خواهد بود.
6) برای متغیر در ساختمان، getter و setter با هم تعریف می‌شوند و امکان تعریف فقط یکی از آن‌ها وجود ندارد؛ تعریف setter قبل از تعریف getter خواهد بود.

7) در برنامه می‌تواند هر تعداد خط خالی از کد وجود داشته باشد و تاثیری در روند اجرا نخواهند داشت.

8) main آرگومان نمی پذیرد.

9) قسمت شرط if و while باید یک expression باشد و نمی توان statement در آن قرار داد.

10) تابع display با بدنه خالی نداریم.

11) همانطور که قبلا گفته شد نیازی به بررسی هیچ گونه ناهماهنگی در تایپ‌ها ندارید . برای مثال یک لیست با عضوهایی که تایپ‌های متفاوت دارند، باید توسط برنامه شما پذیرفته شود.

12) اولویت عملگرها باید رعایت شود.

خروجی

پس از نوشتن تحلیلگرهای لغوی و نحوی , باید به کمک Action هایی که با زبان java می‌نویسید و به گرامر خود اضافه می‌کنید خروجی‌های زیر را بر اساس پیمایش Pre-order درخت parse چاپ کنید :

1) هنگام رسیدن به تعریف هر ساختمان، نام ساختمان را به صورت زیر (قبل از مشاهده‌ی دستورات داخل آن) چاپ کنید :

StructDec : #name_of_struct

2) هنگام مشاهده توابع get و set در ساختمان، قبل از مشاهده دستورات درون‌شان عبارات زیر را چاپ کنید:

Getter

Setter

3) هنگام رسیدن به تعریف هر تابع، نام تابع را به صورت زیر (قبل از مشاهده‌ی دستورات داخل آن) چاپ کنید :

FunctionDec : #name_of_function

4) در صورت مشاهده بخش main، نام آن را به صورت زیر (قبل از مشاهده‌ی دستورات داخل آن) چاپ کنید :

Main

5) هنگام رسیدن به تعریف آرگومان‌های یک تابع، نام آرگومان را به صورت زیر چاپ کنید :
ArgumentDec : #name_of_argument

6) با رسیدن به تعریف هر متغیر، نام آن را به صورت زیر چاپ کنید:
VarDec: #name_of_variable

7) در صورت مشاهده دستورات if و else، به صورت زیر آن‌ها را چاپ کنید :
Conditional : if

Conditional : else

8) در صورت مشاهده دستور while و do…while، آن‌ها را به صورت زیر چاپ کنید:
Loop : while

Loop : do…while

9) در صورت مشاهده دستور return آن را به صورت زیر چاپ کنید :
Return

10) در صورت مشاهده دستور display آن را به صورت زیر چاپ کنید :
Built-in : display

11) در صورت مشاهده فراخوانی یک تابع به عنوان یک گزاره (نه در expression) ، آن را به صورت زیر چاپ کنید:
FunctionCall

12) در صورت مشاهده append یا size، آن‌ها را به صورت زیر چاپ کنید :
Size

Append

عبارت ها را در درخت عبارت به صورت Post-order پیمایش کنید. یعنی برای یک عملگر دوتایی ابتدا عملگرهای اپرند اول و بعد عملگرهای اپرند دوم و در نهایت خود آن عملگر چاپ شود .(عملگر یگانه مانند دوگانه‌ای است که یک فرزند دارد)

در صورت مشاهده یک عملگر، تنها خود عملگر را به صورت زیر چاپ کنید. عملگر های [] ، () و . نباید چاپ شوند.

Operator : OperatorSymbol

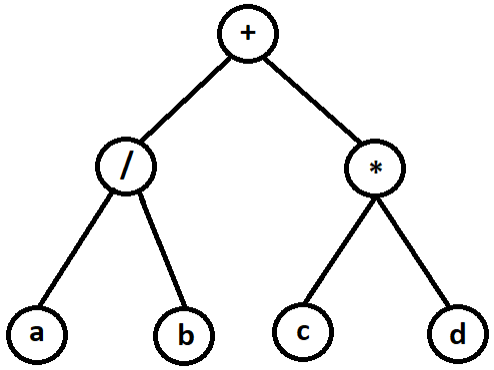
به عنوان مثال برای عبارت a / b + c * d عملگرها را به صورت زیر چاپ کنید :

Operator : /

Operator : *

Operator : +

به درخت عبارت آن توجه کنید :



همچنین دقت کنید که عبارت‌های قبل از : همگی کلیدواژه هستند و آنها را عینا چاپ کنید. تنها موارد خواسته شده را در فایل خروجی نمایش دهید و از قرار دادن خط‌های خالی و فاصله و ... نیز خودداری کنید.

در ادامه دو نمونه کد به همراه خروجی‌شان آمده است. توجه داشته باشید که کدها در این فاز بدون خطای لغوی و نحوی هستند اما می‌توانند خطای معنایی داشته باشند که در فازهای بعد بررسی می-شوند. نمونه کد اول بدون خطا و نمونه کد دوم دارای خطاهای معنایی است.

نمونه کد اول

```
struct House begin
    int price;
    int area(int _area) begin
        set begin
            area  = _area
```

```
            price = 1000 * area
        end
        get
            return area
    end
    struct Room room
end

struct Room begin
    int area;
end

list#int f1(int num) begin
    list#int a
    int i; i=0;
    while(i < num) begin
        append(a, i)
        i = i + 1
    end
    return a;
end

list#int f2(int num)begin
    list#int a
    int i; i=0
    while(i < num) begin
        append(a, i*i)
        i = i + 1
    end
    return a
end

fptr<int -> list#int> f3(bool choice, int dummy, struct House h1) begin
    display(dummy)
    display(h1.price)
    if choice
        return f1
    else
        return f2
end
```

```
main()begin
    fptr<int -> list#int> functionPointer
    struct House h1,h2,h3,h4
    struct Room r1

    r1.area = 12
    h1.room = r1
    h1.area = (90)

    functionPointer = f3(true, 5*2+3-2*2+(-1) ,h1)
    list#int numbers = functionPointer(15)

    list#list#struct House houses

    list#struct House alley1
    list#struct House alley2

    append(alley1, h2)
    append(alley1, h3)
    display(size(alley1))
    append(houses, alley1)
    append(houses, alley2)
    append(houses[1], h4)
end
```

```
StructDec : House
VarDec : price
VarDec : area
ArgumentDec : _area
Setter
Operator : *
Getter
Return
VarDec : room
StructDec : Room
VarDec : area
FunctionDec : f1
ArgumentDec : num
```

```
VarDec : a
VarDec : i
Loop : while
Operator : <
Append
Operator : +
Return
FunctionDec : f2
ArgumentDec : num
VarDec : a
VarDec : i
Loop : while
Operator : <
Append
Operator : *
Operator : +
Return
FunctionDec : f3
ArgumentDec : choice
ArgumentDec : dummy
ArgumentDec : h1
Built-in : display
Built-in : display
Conditional : if
Return
Conditional : else
Return
Main
VarDec : functionPointer
VarDec : h1
VarDec : h2
VarDec : h3
VarDec : h4
VarDec : r1
Operator : *
Operator : +
Operator : *
Operator : -
Operator : -
Operator : +
```

```
VarDec : numbers
VarDec : houses
VarDec : alley1
VarDec : alley2
Append
Append
Built-in : display
Size
Append
Append
Append
```

نمونه کد دوم

```
struct S1 begin
    int var1 , var2 = 3;
     int id (int _id) begin
        set begin
            id = _id;
        end
        get
            return id
     end
end

fptr <int -> int> f1(int arg1, bool arg1, list# list# struct S1 arg2)  begin
    fptr <void -> void> ptr1 = f1; return ptr1;
end


int sample()
    if a | b * c + d / g
        if ~(a & b)
            return append(li, 8 * 9 / func())[1][0];

        else
            return -size(li) + a - n;
```

```
main() begin
    f1()();
     do
        i = i + 1 - 3;
     while i < n & i > k
end
```

```
StructDec : S1
VarDec : var1
VarDec : var2
VarDec : id
ArgumentDec : _id
Setter
Getter
Return
FunctionDec : f1
ArgumentDec : arg1
ArgumentDec : arg1
ArgumentDec : arg2
VarDec : ptr1
Return
FunctionDec : sample
Conditional : if
Operator : *
Operator : /
Operator : +
Operator : |
Conditional : if
Operator : &
Operator : ~
Return
```

```
Append
Operator : *
Operator : /
Conditional : else
Return
Size
Operator : -
Operator : +
Operator : -
Main
FunctionCall
Loop : do...while
Operator : +
Operator : -
Operator : <
Operator : >
Operator : &
```

نکات مهم:

- **کد خود را به صورت یک فایل Cmm.g4 آپلود کنید و نام اولین قانون زبان را cmm بگذارید.**
- **در صورت کشف هرگونه تقلب، نمره 100- لحاظ میشود.**
- **دقت کنید که خروجی‌های شما به صورت خودکار تست می شوند؛ پس نحوه چاپ خروجی شما باید عیناً مطابق موارد ذکر شده در بالا باشد. علاوه بر آن، درخت parse شما نیز بررسی می‌شود.**
- **بهتر است سوالات خود را در فروم یا گروه درس مطرح نمایید تا دوستان‌تان نیز از آن‌ها استفاده کنند؛ در غیر این صورت به مسئولان پروژه ایمیل بزنید:**

آرش رسولی arash3908@gmail.com

نازنین یوسفیان nazaninyousefian79@gmail.com