بسمه تعالی

طراحی کامپایلرها

نیم‌سال دوم ۹۸ـ۹۷
مدرسان: دکتر قاسم‌ثانی و دکتر فیض بخش رانکوه

دانشکده مهندسی کامپیوتر

پروژه پایانی

فاز دوم ـ پیاده‌سازی تحلیل‌گر نحوی

(مهلت تحویل، ۲۷ اردیبهشت)

شما در فاز قبلی، یک اسکنر برای بخشی از زبان C پیاده‌سازی کردید. در این فاز، باید با استفاده از اسکنری که پیاده‌سازی کردید، یک پارسر برای این زبان پیاده‌سازی کنید. توجه کنید که استفاده از کدهای موجود در مرجع درس یا سایر کتب کامپایلر، در صورت تسلط بر آن کد و اعلام مأخذ در مستندات همراه پروژه اشکالی ندارد ولی استفاده از کدها و برنامه‌های موجود در سایت‌ها و کدهای سایر گروه‌ها (در همین نیم‌سال یا سال‌های گذشته) اکیدا ممنوع است و در اکثر موارد سبب مردودی در درس خواهد شد. در این مورد تفاوتی میان گروه دهنده یا گیرنده کد وجود ندارد. دقت کنید این گزاره حتی در صورتی که فاز یک را پیاده‌سازی نکرده‌اید نیز، درست است و شما برای پیاده‌سازی پارسر، نمی‌توانید از اسکنر سایر گروه‌ها استفاده کنید و باید اسکنر مورد نیاز را خودتان پیاده‌سازی کنید.

توضیحات کلی تحلیل‌گر نحوی

شما در این فاز باید با استفاده از روش دیاگرام انتقال، یک پارسر برای زبان NC-Minus پیاده‌سازی کنید. در مورد این روش، توضیحات در کلاس و در انتهای Lecture Note شماره‌ی ۷، داده شده است. علاوه بر این، شما می‌توانید توضیحات بیشتری در مورد این روش را در بخشی از منابع درس که همراه این سند آپلود شده، پیدا کنید.
پارسری که پیاده‌سازی می‌کنید، باید predictive باشد و این یعنی قبل از هر کاری باید گرامر داده شده را به فرم LL(1) در بیاورید.
عملیات پارس باید به صورت همزمان (pipeline) با عملیات اسکن انجام شود؛ به این معنا که پارسر تا زمان رسیدن به توکن EOF و پایان فرایند پارس، در مراحلی که نیاز به خواندن از ورودی دارد، تابع get_next_token در اسکنر را صدا زده و توکن جدید را دریافت کرده و توکن قبلی را دور می‌اندازد.
در این فاز، ورودی شما همانند فاز قبل، یک فایل متن حاوی برنامه‌ای است که باید پارس شود. خروجی در این فاز، دو فایل متنی خواهد بود که در اولی، درخت پارس فایل ورودی نوشته می‌شود و در فایل دومی لیست خطاها شامل خطاهای اسکنر و پارسر. توضیحات مربوط به نحوه‌ی نمایش درخت پارس و دسته‌بندی خطاهای نحوی در ادامه خواهد آمد.
عملیات پارس باید طبق گرامر NC-Minus صورت بگیرد که در ادامه خواهد آمد.

گرامر NC-Minus

عملیات پارس باید طبق این گرامر صورت بپذیرد. در این گرامر، پایانه‌ها پررنگ‌تر از غیر پایانه‌ها نمایش داده شده‌اند. دقت کنید شما نباید تغییراتی در گرامر اعمال کنید که باعث تغییر زبان شود. بنابراین هر گونه تغییری که در گرامر برای تبدیل آن به فرم LL(1) می‌دهید، نباید زبان آن را تغییر دهد.

1. program → declaration-list **EOF**
2. declaration-list → declaration-list declaration | ϵ
3. declaration → var-declaration | fun-declaration
4. var-declaration → type-specifier **ID ;** | type-specifier **ID** [**NUM**] **;**
5. type-specifier → **int** | **void**
6. fun-declaration → type-specifier **ID** (params) compound-stmt
7. params → param-list | **void**
8. param-list → param-list **,** param | param
9. param → type-specifier **ID** | type-specifier **ID** [ ]
10. compound-stmt → **{** declaration-list statement-list **}**
11. statement-list → statement-list statement | ϵ
12. statement → expression-stmt | compound-stmt | selection-stmt | iteration-stmt | return-stmt | switch-stmt
13. expression-stmt → expression **;** | **continue ;** | **break ;** | **;**
14. selection-stmt → **if** (expression) statement **else** statement
15. iteration-stmt → **while** (expression) statement
16. return-stmt → **return ;** | **return** expression **;**
17. switch-stmt → **switch** (expression) **{** case-stmts default-stmt **}**
18. case-stmts → case-stmts case-stmt | ϵ
19. case-stmt → **case NUM :** statement-list
20. default-stmt → **default :** statement-list | ϵ

21. expression → var = expression | simple-expression

22. var → **ID** | **ID** [expression]

23. simple-expression → additive-expression relop additive-expression | additive-expression

24. relop → < | ==

25. additive-expression → additive-expression addop term | term

26. addop → + | -

27. term → term * signed-factor | signed-factor

28. signed-factor → factor | + factor | - factor

29. factor → (expression) | var | call | **NUM**

30. call → **ID** (args)

31. args → arg-list | ϵ

32. arg-list → arg-list , expression | expression

ملاحظات پیاده‌سازی تحلیل‌گر نحوی

- پس از تبدیل دستوارت گرامر به دیاگرام انتقال، نباید دیاگرام‌ها را ساده‌سازی کنید.
- در دیاگرام به دست آمده، باید به ازای هر غیر پایانه، یک زیر دیاگرام انتقال وجود داشته باشد.
- هر زیر دیاگرام تنها یک حالت نهایی دارد. حالت‌های نهایی در زیر دیاگرام‌ها یال خروجی ندارند و هنگام رسیدن به حالت‌نهایی، کنترل برنامه از زیر دیاگرام فعلی گرفته شده و به زیر دیاگرام فراخواننده بازگردانده می‌شود.
- فرایند پارس زمانی که برنامه به وضعیت نهایی زیر دیاگرام مربوط به غیرپایانه شروع گرامر (در این گرامر program) برسد، به اتمام می‌رسد.
- در هنگام پیمایش یک زیر دیاگرام، یال‌های با برچسب پایانه در صورتی پیمایش می‌شوند که توکن ورودی مطابق با پایانه‌ی روی آن یال‌ها باشد و با پیمایش این یال‌ها، توکن جدیدی از اسکنر گرفته می‌شود.
- در مورد یال‌های با برچسب غیرپایانه‌ای مانند A، کنترل برنامه در صورتی که توکن جاری عضو First(A) باشد، به حالت شروع زیر دیاگرام A می‌رود. همچنین اگر ϵ عضو First(A) ، و در عین حال، توکن جاری عضو Follow(A) باشد، وارد حالت شروع زیر دیاگرام A می‌شویم.
- یال‌های با برچسب ϵ در یک زیر دیاگرام تنها زمانی طی می‌شوند که توکن جاری عضو Follow غیرپایانه‌ای باشد که زیر دیاگرام به آن تعلق دارد. دقت کنید با توجه به اینکه گرامر قبل از تبدیل شدن به دیاگرام انتقال، به فرم LL(1) در آمده است، تمامی این نوع از یال‌ها، مستقیما از حالت شروع یک زیر دیاگرام به حالت نهایی آن متصل شده‌اند.
- در صورتی که در حین فرایند پارس، به حالتی برسیم که نتوانیم از هیچ‌کدام از یال‌های آن، با استفاده از سه مورد بالا گذر کنیم، وارد فرایند خطاپردازی می‌شویم. فرایند خطاپردازی در ادامه‌ی این سند به صورت مفصل توضیح داده شده است.
- برای پیدا کردن مجموعه‌های First و Follow هر غیرپایانه می‌توانید از سایت‌ها و برنامه‌هایی که این کار را با گرفتن گرامر، به صورت اتوماتیک انجام می‌دهند، با ذکر منبع از طریق اضافه کردن کامنت به کد پارسر، استفاده کنید.
- در طراحی سیستم ماشین‌حالت، بهتر است امکان فراخوانی توابعی در زمان ورود و خروج از هر حالت را فراهم کنید تا در هنگام پیاده‌سازی فاز نهایی پروژه، راحت‌تر بتوانید تحلیل‌گر معنایی و مولد کد میانی را روی پارسر سوار کنید.

خطاپردازی در تحلیل نحوی

خطاپردازی در این تحلیل‌گر نحوی، با استفاده از روش Panic Mode می‌باشد. در هنگام برخورد با خطا، یکی از دو حالت زیر پیش آمده است:

۱. اگر در فرایند طی کردن دیاگرام، در حالتی که یال خروجی آن یک پایانه است، خطا رخ دهد، با فرض اینکه پایانه‌ی مذکور از ورودی جا افتاده است، یک پیغام خطا به صورت:

#LINE_NUM : Syntax Error! Missing #TERMINAL_NAME

در فایل مربوط به خطاها چاپ می‌شود.

۲. در حالت دوم اگر در هنگام بروز خطا، در حالتی با یال خروجی غیرپایانه‌ای مانند A باشیم، شروع به حذف کردن توکن‌های ورودی به همراه چاپ پیام‌هایی با مضمون:

#LINE_NUM : Syntax Error! Unexpected #TERMINAL_NAME

در فایل خطاها می‌کنیم تا زمانی که به یک توکن در First یا Follow غیرپایانه A برسیم. حال اگر بدانیم ϵ عضو مجموعه‌ی First(A) است، می‌توانیم فرایند پارس را بدون مشکل ادامه دهیم. اما در صورتی که توکن جاری عضو Follow(A) باشد اما ϵ عضو First(A) نباشد، برای جلوگیری از ادامه‌ی فرایند حذف کردن ورودی‌ها، فرض می‌کنیم که دیاگرام مربوط به این غیرپایانه را به درستی طی کرده‌ایم و با چاپ کردن پیامی به فرم:

#LINE_NUM : Syntax Error! Missing #NON_TERMINAL_DESCRIPTION

در خروجی مربوط به خطاها، یال مربوط به غیرپایانه‌ی A را طی می‌کنیم و فرایند پارس را ادامه می‌دهیم. توضیحاتی که در پیام خطا در مورد غیرپایانه باید چاپ شود، می‌تواند توصیف ساده‌ترین عبارتی باشد که از این غیرپایانه نتیجه می‌شود.

روش‌هایی که در بالا ذکر شده‌اند، فقط در حالت‌هایی با تنها یک خروجی قابل استفاده هستند، ولی با توجه به روش پارس، نباید در هیچ وضعیت دیگری با خطا مواجه شویم. دقت کنید که فقط و فقط در حالت شروع زیر دیاگرام‌ها ممکن است بیش از یک یال خروجی داشته باشیم، ولی با توجه به روش طی شدن دیاگرام‌های انتقال، هیچ‌گاه در حالت شروع یک زیر دیاگرام، به خطا بر نمی‌خوریم. [۱]

همچنین برای اطمینان از تمام شدن فرایند پارس، در دو وضعیت خاصی که مربوط به توکن **EOF** می‌شوند، باید به نحوه‌ی دیگری عمل کنیم:

وضعیت اول زمانی رخ می‌دهد که در حال خطاپردازی هستیم و توکن **EOF** از اسکنر دریافت می‌شود.

وضعیت دوم در مقابل زمانی است که در حالتی با تنها یک یال خروجی با یال **EOF** قرار داریم و وارد فرایند خطاپردازی شده‌ایم.

در هردوی این وضعیت‌ها، فرایند پارس باید متوقف شود و در وضعیت اول، پیام:

#LINE_NUM : Syntax Error! Unexpected EndOfFile

---

[۱] غیرپایانه‌ی شروع از این قاعده مستثنی است اما با توجه به اینکه غیرپایانه‌ی شروع گرامر تنها یک قاعده دارد، مشکلی برای فرایند پارس ایجاد نمی‌شود.

و در وضعیت دوم، پیام:

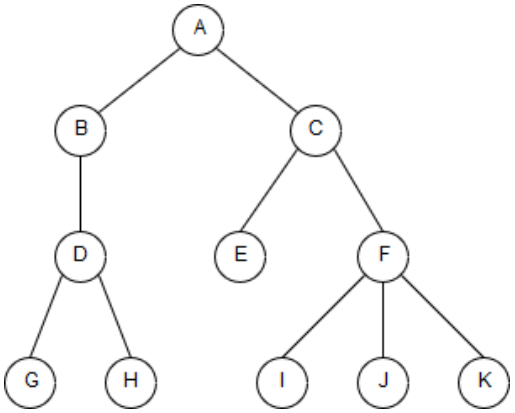
#LINE_NUM : Syntax Error! Malformed Input

در فایل مربوط به خطاها نوشته شود.

فرمت خروجی و نحوه‌ی چاپ درخت پارس

همانطور که در ابتدای این سند ذکر شده، در این فاز، ورودی شما یک فایل متنی حاوی کدی است که باید پارس شود. خروجی پارسر شما دو فایل متنی است که یکی از آن‌ها برای نمایش درخت پارس برنامه‌ی ورودی و دیگری برای نمایش لیست خطاهای لغوی و نحوی موجود در برنامه است. نحوه‌ی تولید درخت پارس، بر عهده‌ی خود شما است، اما برای نمایش این درخت در یک فایل متنی، می‌توانید از فرمت نمایشی که در ادامه معرفی می‌شود، استفاده کنید:

در این نمایش، در هر خط از خروجی، یک گره (node) از درخت چاپ می‌شود. تعداد tab هایی که در پشت هر گره چاپ می شود، نمایانگر عمق گره مذکور در درخت است. با رسیدن به هر گره و چاپ کردن آن، گره‌های فرزند آن از چپ به راست طی شده و در خطوط بعدی چاپ می‌شوند. به عنوان یک نمونه‌ی ورودی و خروجی می‌توانید به درخت و خروجی مربوطه‌ی آن در ادامه توجه کنید. دقت کنید در صورتی که از روش گویای دیگری برای چاپ درخت استفاده کنید نیز، مشکلی نخواهد بود.



درخت نمونه

```
1   A
2   |   B
3   |   |   D
4   |   |   |   G
5   |   |   |   H
6   |   C
7   |   |   E
8   |   |   F
9   |   |   |   I
10  |   |   |   J
11  |   |   |   K
```

نمایش درخت نمونه

نمونه‌ی ورودی

با توجه به اینکه به شکل‌های مختلفی می‌توان یک گرامر را تبدیل به $LL(1)$ کرد و همه این تبدیل‌ها لزوما یک درخت پارس یکتا به ازای هر ورودی بدست نمی‌آورند، نمی‌توانیم مثالی که همراه با درخت پارس و خطایابی باشد در اختیار شما قرار دهیم. اما در نمونه زیر کدی قرار دارد که تعداد زیادی از ویژگی‌ها و ساختارهای مهم زبان را در بر دارد که می‌توانید از آن به عنوان نمونه تست استفاده کنید.

```
int b;
int foo(int d, int e){
    int f;
    void foo2(int k[2]) {
        return k[0] + k[1];
    }
    int fff[2];
    fff[0] = d;
    fff[1] = d + 1;
    f = foo2(fff);
    b = e + f;
    while (d > 0) {
        f = f + d;
        d = d - 1;
        if (d == 4)
            break;
    }
    // comment1
    return f + b;
}
int arr[3];
void main(void){
    int a;
    a = -3 + +11;
    b = 5 * a + foo(a, a);
    arr[1] = b + -3;
    arr[2] = foo(arr[0], arr[1])
    if (b /* comment2 */ == 3) {
        arr[0] = -7;
    }
    else
    {   switch (arr[2]) {
            2:
                b = b + 1;
            3:
                b = b + 2;
                return;
            4:
            {   u = 5;
                b = u * -123;
                break;}
            default:
                b = b - -1;
        }}
    return;
}
```

نمونه فایل ورودی

انجام پروژه

- امکان تغییر گروه‌ها برای انجام این فاز از پروژه وجود ندارد. سعی کنید در انجام پروژه به صورت گروهی، هر دو عضو گروه، مشارکت کافی داشته باشند. در صورت اختلاف قابل توجه مشارکت دو عضو گروه در انجام پروژه، فردی که مشارکت کمتری داشته، نمره‌ی کمتری نسبت به دیگری می‌گیرد.
- در صورتی که هر گونه سوالی در رابطه با تعریف پروژه دارید، آن را از طریق کوئرا مطرح نمایید.
- مهلت بارگذاری سورس کد پروژه برای فاز دوم، ساعت ۲۳:۵۹ روز جمعه ۲۷ اردیبهشت است.
- تحویل حضوری دو فاز اول و دوم همزمان بوده و زمان‌بندی دقیق آن متعاقبا اعلام خواهد شد. (توجه کنید که حضور هر دو عضو گروه‌های دو نفره در جلسه‌ی تحویل الزامی است.)

موفق باشید.