اصول طراحی کامپایلر (دکتر ممتازی)

نیم‌سال اول سال تحصیلی ۱۴۰۳-۱۴۰۲

پروژه پایانی

قبل از پیاده سازی پروژه به نکات زیر توجه داشته باشید:

- هدف از انجام پروژه‌ها، یادگیری عمیق‌تر مطالب درسی است و پروژه‌ها به صورت انفرادی انجام می‌شوند. در نتیجه هرگونه تقلب موجب کسر نمره خواهد شد.
- پروژه جهت کمک به پیشروی دانشجویان در ۳ فاز طراحی شده است اما تحویل آن به صورت کامل در انتهای ترم صورت می‌گیرد.

ددلاین تحویل پروژه‌ی کامل اواخر ترم تحصیلی بوده و زمان دقیق آن در آینده اعلام خواهد شد.

- پروژه تحویل انلاین دارد و در صورت حاضر نشدن در ارائه آنلاین نمره پروژه ۰ تلقی می‌شود.
- در صورت وجود سوال می توانید از طریق ایمیل compiler.fall.2023.aut@gmail.com یا کانال تلگرام درس با تدریسیاران در ارتباط باشید.

مقدمه‌ای برANTLR

ANTLR، که مخفف «ANother Tool for Language Recognition» است، یک ابزار قدرتمند برای تولید تجزیه‌گر «Parser» است. با استفاده از یک دستورالعمل گرامری، ANTLR یک تجزیه‌گر را تولید می‌کند که می‌تواند درخت تجزیه را بسازد و آن را پیمایش کند. در واقع، ANTLR یک ابزار است که به شما اجازه می‌دهد تا قوانینی را تعریف کنید که چگونه ورودی‌ها باید تجزیه شوند. بعد از آن، این قوانین را به یک زبان برنامه‌نویسی خاص ترجمه می‌کند تا بتوانید از آن در برنامه‌های خود استفاده کنید.

در مرحله اول پروژه، دانشجویان باید گرامر یک زبان را که در مستندات توضیح داده شده است بنویسند. پس از نوشتن گرامر، با استفاده از تجزیه‌گر درختی ANTLR، باید درخت تجزیه را برای موارد تست داده شده رسم کنند. در فازهای بعدی، دانشجویان باید یک کامپایلر برای این زبان با استفاده از ANTLR بنویسند تا کد سه آدرسی را برای فایل ورودی تولید کنند. کد سه آدرسی در هفته‌های آتی آموزش داده خواهد شد.

نحوه کار کردن با ANTLR

ما در اینجا قصد داریم گام به گام نحوه کار با ANTLR را به شما آموزش دهیم.

زبان :

زبانی که قصد داریم تجزیه کنیم، یک زبان ریاضی ساده شامل عملیات جمع و تفریق است. این زبان از اعداد صحیح و پرانتزها برای تعیین اولویت عملیات پشتیبانی می‌کند.

$$
\begin{aligned}
&< \boldsymbol{Expression} > && ::== && < \boldsymbol{Expression} > + < \boldsymbol{Term} > \\
& && | && < \boldsymbol{Expression} > - < \boldsymbol{Term} > \\
& && | && < \boldsymbol{Term} > ;
\end{aligned}
$$

$$
\begin{aligned}
&< \boldsymbol{Term} > && ::== && < \boldsymbol{Type} > \\
& && | && (< \boldsymbol{Expression} >) ;
\end{aligned}
$$

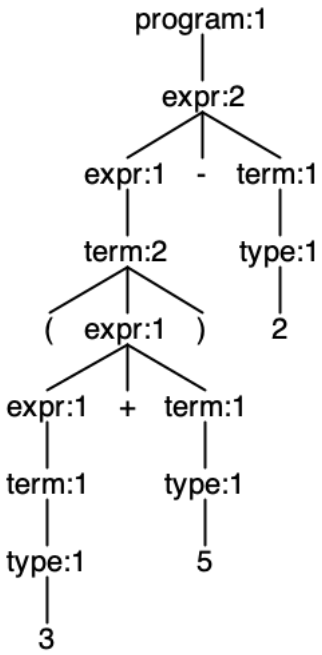
$$
\begin{aligned}
&< \boldsymbol{Type} > && ::== && \boldsymbol{int}
\end{aligned}
$$

گرامر:

```
grammar ArithGrammer;

// Starting rule
program: expr ;

expr    : expr '+' term
        | expr '-' term
        | term
        ;

term    : type
        | '(' expr ')'
        ;

type    : INT ;

INT     : [0-9]+ ;

WS      : [ \t\r\n]+ -> skip ;
```

این گرامر به شما این اجازه را می‌دهد تا ورودی‌هایی که عملیات جمع/تفریق انجام می‌دهند را تجزیه کنید. با کمک دستور زیر خروجی درخت پارس شده را رسم می‌کنیم.

```
antlr4-parse ArithGrammer.g4 program -tree test/test003.txt -gui
```



در نهایت با کمک دستور زیر فایل‌هایی برایمان ساخته می‌شود که حاوی کد Lexer ، Parser و Listener می‌باشد. در فازهای بعدی به این فایل‌ها پرداخته می‌شود.

```
antlr4 -o src ArithGrammer.g4
```

پیاده سازی

در این پروژه باید به کمک ابزار ANTLR که در بخش قبل یاد گرفتید، گرامر زبانی که در ادامه شرح داده می‌شود را بنویسید تا در فازهای آینده کامپایلری جهت تولید کد سه آدرسه در زبان C پیاده سازی کنید. گرامر زبان پروژه در ادامه آورده شده است. توجه کنید این گرامر را باید در فرمت antlr4 پیاده‌سازی کنید و اطمینان حاصل کنید که به درستی فایل‌های تست را تجزیه می‌کند. به عنوان مثال باید ترتیب عملیات ریاضی حفظ شود. همچنین ابهاماتی در گرامر وجود دارد که باید رفع شوند.

$$
\begin{array}{lll}
< prog > & ::= & < func - list > \\
\\
< func - list > & ::= & < func - def >< func - list > \\
 & & | < func - def > \\
\\
< func - def > & ::= & < data - type >< id > (< param - list >) < code - block > \\
\\
< param - list > & ::= & < param >, < param - list > \\
 & & | < param > \\
 & & | \epsilon \\
\\
< param > & ::= & < data - type >< id > \\
\\
< data - type > & ::= & \text{int} \mid \text{double} \mid \text{boolean} \mid \text{void} \\
\\
< code - block > & ::= & \{< stmt - list >\} \\
\\
< stmt - list > & ::= & < stmt >< stmt - list > \\
 & & | \epsilon \\
\\
< stmt > & ::= & ; \\
 & & | < code - block > \\
 & & | < data - type >< var - list >; \\
 & & | < id >=< expr >; \\
 & & | < id > ++; \\
 & & | < id > --; \\
 & & | \text{return} ; \\
 & & | \text{return} < expr >; \\
 & & | < loop - stmt > \\
 & & | \text{decide} (< expr >) < stmt > \\
 & & | \text{decide} (< expr >) < stmt > else < stmt > \\
 & & | < expr >; \\
\\
< loop - stmt > & ::= & \text{loop} (< expr >) < stmt > \\
 & & | \text{do-loop} < stmt > \text{until} (< expr >); \\
 & & | \text{for-loop} (< init - stmt >; < expr >; < post - stmt >) < stmt > \\
\\
< init - stmt > & ::= & < data - type >< id >=< expr > \\
 & & | < id >=< expr > \\
 & & | \epsilon \\
\\
< post - stmt > & ::= & < id >=< expr > \\
 & & | < id > ++ \\
 & & | < id > -- \\
 & & | \epsilon
\end{array}
$$

$$
\begin{array}{lll}
< var - list > & ::= & < var - list >< var > \\
& & | < var > \\
\\
< var > & ::= & < id > \\
& & | < id >=< expr > \\
\\
< expr > & ::= & < number > \\
& & | < id > \\
& & | \text{true} \\
& & | \text{false} \\
& & | < string - lit > \\
& & | < id > (< args >) \\
& & | (< expr >) \\
& & | < unop >< expr > \\
& & | < expr >< binop >< expr > \\
\\
< args > & ::= & < expr >, < args > \\
& & | < expr > \\
\\
< unop > & ::= & \text{- | !} \\
\\
< binop > & ::= & \text{+ | - | * | / | == | != | ¿ | ¿= | ¡ | ¡= | and | or} \\
\\
< number > & ::= & < integer > | < double >
\end{array}
$$

تحلیل‌گر لغوی

شما باید توکن‌های زبان را با توجه به گرامر مشخص کنید. هر رشته‌ای که بین دو جهت‌نما نوشته نشده است یک توکن است. همچنین موارد زیر نیز توکن‌های زبان هستند:

- اعداد صحیح: اعداد صحیح با صفر شروع نمی‌شوند.
- اعداد ممیز شناور: به صورت integer.integer تعریف می‌شوند.
- شناسه‌ها: با ارقام شروع نمی‌شوند و شمال حروف کوچک و بزرگ انگلیسی، ارقام و کاراکترهای خاص مانند – و _ هستند. همچنین کلیدواژه‌ها نمی‌توانند شناسه باشند.
- کلیدواژه‌ها: شامل ... ,if, else, while, for هستند.
- رشته‌ها: دنباله‌ای از مجموعه کاراکترهای پایه‌ای به جز Quotation mark و Reverse solidus و Line feed همچنین رشته‌ها می‌توانند شامل Simple escape sequences نیز باشند. دقت کنید رشته‌ها فقط به تابع اولیه printString ورودی داده می‌شوند. (در فازهای بعدی بیشتر با این تابع آشنا می‌شوید)
- کامنت: در این زبان می‌توان با کمک // یا /* */ کامنت تعریف کرد.

تحلیلگر نحوی

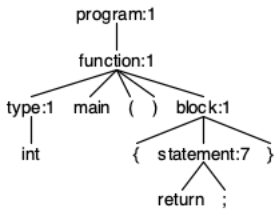
دقت کنید ANTLR یک تجزیه‌گر top-down است. از درس می‌دانیم ملاحظاتی برای گرامری که به این تجزیه‌گرها داده می‌شود وجود دارد. با توجه به قابلیت‌های ANTLR آنها را در صورت لزوم اعمال کنید. همچنین چند ابهام در گرامر وجود دارد. ابهام اول درباره dangling else است که به روش nearest if باید رفع شود. ابهام دوم مربوط به اولویت عملگرهاست. از جدول زیر برای اولویت‌دهی و رفع این ابهام استفاده کنید:

Precedence	Operator	Associativity	Description
1	- !	Right-to-left	Unary minus and logical NOT
2	* / %	Left-to-right	Multiplication, division, and remainder
3	+ -	Left-to-right	Addition and subtraction
4	< > <= >=	Non-associative	Relational operators
5	== !=	Non-associative	Equality operators
6	&&	Left-to-right	Logical AND
7	||	Left-to-right	Logical OR

خروجی

در فایل زیپ همراه پروژه چند فایل test قرار داده شده که برای تمامی آنها می‌بایست درخت تجزیه شده را رسم کنید. برای مثال یک برنامه و خروجی مد نظر آن در ادامه قابل مشاهده می‌باشند.

```
int main() {
    return 0;
}
```



بعد از بدست آوردن خروجی‌های مدنظر، با کمک دستور آموزش داده شده در بخش‌های قبلی کد مربوط به Lexer, Parser, Listener را تولید کنید و با مطالعه کد Lexer, Parser بخش‌های متفاوت را توضیح دهید.

گزارش

برای گزارش فاز اول پروژه، فایلی خلاصه و کوتاه تهیه کنید که شامل اطلاعات ذیل باشد.

۱. بخش اولویت عملگرها و روش استفاده شده برای رفع dangling else را توضیح دهید.

۲. خروجی دو درخت تجزیه شده را به دلخواه انتخاب کنید و به صورت خلاصه توضیح دهید.

۳. فایل‌های تولید شده توسط antlr را به صورت خلاصه توضیح دهید.

امتیازی

در پیاده‌سازی گرامر در صورتی که موارد ذیل را لحاظ کنید نمره امتیازی کسب خواهید کرد.

- به گرامر، دستورات goto را اضافه کنید.
- به گرامر، تایپ struct را اضافه کنید.
- به گرامر، عملگر XOR را اضافه کنید.