تکلیف Yacc
درس کامپایلر، ترم دوم ۹۸-۹۷
استاد درس: زینب زالی

همراه فایل تکلیف، فایل‌های جانبی که در صورت تکلیف بدان‌ها اشاره شده و همچنین شاخه testcase که شامل نمونه‌های ورودی و خروجی است، ارائه شده است.
برنامه‌های شما باید ورودی را از ورودی استاندارد گرفته و خروجی را در خروجی استاندارد نشان دهد.
لطفاً هر تکلیف را به صورت جداگانه آماده کرده و در فایل‌هایی با نام mathExp_stdno و Decaf_yacc_stdno فشرده کرده در تکلیف مرتبط آپلود نمایید

۱- فایل simple-expr.y حاوی یک کد yacc برای محاسبه عبارات ساده ریاضی است. همان‌طور که می‌دانید، در بخشی که بین ٪{ ... ٪} قرار دارد هر کد c/c++ موردنظر می‌تواند نوشته شود. همچنین توکن‌هایی که تحلیلگر لغوی تشخیص می‌دهد با ٪token مشخص می‌شود. با کمک bison یا yacc می‌توانید با دستور زیر، از این فایل، یک تجزیه‌گر بسازید:

```
bison -o simple-expr.tab.c -d simpleexpr.y
```

آپشن d- باعث تولید فایل هدر simple-expr.tab.h می‌شود.
اگر کد lex هم نوشته شده باشد و با lex یا flex به صورت زیر کامپایل شود:

```
flex -o simple-expr.lex.c simple-expr.lex
```

فایل باینری نهایی از کامپایل خروجی flex و bison بدست می‌آید:

```
gcc -o ./simple-expr simple-expr.tab.c simple-expr.lex.c -ly -lfl
```

این مراحل در فایل makefile نوشته شده که با اجرای make به صورت اتوماتیک انجام می‌شوند.
کد را به صورتی تکمیل کنید که بتواند عبارات چند خطی را هم در نظر بگیرد. مثلاً برنامه حاضر b=5+10-4 را حساب می‌کند ولی از مقدار b برای محاسبه خط بعد که به شکل a=b+5 باشد نمی‌تواند استفاده کند.

کد lex و yacc داده‌شده تا اینجا، نمی‌تواند انتساب به متغیرها را در نظر بگیرد. برای در نظر گرفتن انتساب، فرض می‌کنیم برنامه lex دو نوع متغیر برمی‌گرداند: یکی برای اعداد و دیگری برای نام‌های متغیرها. کد simple-varexpr.lex این کار را کرده است. این کد lex برای اعداد، yylval.rvalue و برای نام متغیرها yylval.lvalue را برمی‌گرداند. این دو نوع در برنامه yacc با نام simple-varexpr.y با استفاده از ٪union تعریف شده‌اند. Union می‌تواند شامل انواع داده پیچیده باشد. کد yacc فقط برای توکن‌ها، type مشخص نمی‌کند بلکه برای غیرترمینال‌ها هم با ٪type مشخص می‌کند. این به yacc اجازه می‌دهد که بتواند نوع غیرترمینال‌ها را که در rvalue هستند در نظر بگیرد.
کد را به صورتی تکمیل کنید که انتساب مقادیر به متغیرها را هندل کند.
سپس کد قبل را تکمیل کنید به صورتی که انواع ثابت (constant) از نوع اعشاری و متغیرهایی که می‌توانند هم صحیح و هم اعشاری باشند را پوشش دهد. همچنین عملیات +، *، /، () پشتیبانی شوند و در نهایت توابع sqrt، expr و log را اضافه کنید به صورتی که عباراتی مانند ورودی زیر قابل محاسبه باشند:

```
a = 2.0
b = exp(a)
b
```

می‌توانید از گرامری شبیه گرامر فایل expr-grammar.y استفاده کنید یا از گرامر مبهم استفاده کرده و الویت‌های عملگرها را برای آن مشخص کنید.

۲- با استفاده از گرامر رفرنسی که برای Decaf در فایل مشخصات Decaf تعریف شده است، گرامر مستقل از متنی بنویسید که LR(1) باشد و توسط Yacc قابل استفاده باشد. سعی کنید در حد امکان غیرترمینالها را مطابق با تعریف گرامر رفرنس نامگذاری کنید مثلاً برای عبارت زیر

$\langle \text{start} \rangle \rightarrow A \langle \text{start} \rangle B \mid \epsilon$

می‌توان چنین نوشت:

```
start: A start B
     | /* empty string */
     ;
```

نوشتن گرامر مستقل از متنِ، بخش مهمی از این تکلیف است. برای مثال برای چنین سینتکسی در Decaf

```
class foo {
  int bar
```

کد بالا، قسمتی از یک کد کامل است. بعد از این قسمت ممکن است field declaration یا method declaration داشته باشیم. این دو گزینه ممکن، مشکلی برای گرامر LR ایجاد نمی‌کند. دو نمونه گرامر در فایل‌های decafSmallGrammar.y و decafSmallLRGrammar.y نوشته شده‌است. این دو گرامر را بررسی کنید. خروجی‌ها برای هر یک از دو گرامر در فایل‌های out آمده است. گرامر اول منجر به s/r confilict می‌شود اما گرامر دوم صحیح کار می‌کند.
سپس یک برنامه yacc بنویسید که AST (Abstract Syntax Tree) را برای هر برنامه Decaf چاپ کند. اگر برنامه ورودی دارای خطا باشد، هیچ درختی چاپ نشود و پیغام خطایی چاپ شود. چاپ پیغام خطا الزامی است ولی کنترل خطا به صورتی که تجزیه ادامه یابد و در حد امکان همه خطاهای برنامه گرفته شود اختیاری و دارای نمره اضافه است! از فرمت asdl که در فایل Decaf.asdl آمده است جهت نمایش درخت پارس استفاده کنید.