پروژه‌ی درس کامپایلر

دانشگاه صنعتی اصفهان

استاد درس: زینب زالی

بهار ۹۹

فهرست مطالب

مقدمه:

در این بخش از پروژه، قصد داریم که یک تحلیل‌گر نحوی بسازیم و آن را به خروجی بخش اول(تحلیل گر لغوی) متصل کنیم تا یک برنامه را هم از نظر نحوی و هم از نظر لغوی تحلیل کنیم.

بنابراین تمام قواعدی که در بخش اول گفته شد همچنان در این جا نیز باید رعایت شوند.

به عبارت دیگر در این بخش شما توکن‌های دریافت شده از تحلیل گر لغوی را با توجه به ساختار نحوی و دستوری زبان بررسی می‌کنید که در ادامه به بیان ساختار برنامه‌ی ورودی خواهیم پرداخت.

هدف:

شما باید با استفاده از flex و yacc یک برنامه بنویسید بنویسید و آن را در سامانه آپلود کنید.

یک برنامه به زبانی که توصیف خواهد شد به برنامه شما داده می‌شود، در صورتی که آن برنامه قواعد لغوی و نحوی زبان را رعایت کرده باشد، شما درخت آن را با توجه به گرامر زبان می‌سازید و آن را به صورت pre order(می‌توانید به درس ساختمان داده مراجعه کنید) چاپ می‌کنید اما در صورت وجود هر گونه نقض قواعد زبان، شما باید شماره‌ی خط و ستون و همچنین نام توکن غیرمجاز را ذکر کنید و یک پیام خطای مناسب نشان دهید.

چند نمونه از ورودی و خروجی برنامه برای شما بر روی سامانه قرار داده شده است و برای درک بیشتر می‌توانید به آن‌ها مراجعه کنید.

توکن‌های جدید:

علاوه بر توکن‌هایی که در بخش اول ذکر شده بود، توکن‌های زیر را نیز در نظر بگیرید:

نام توکن	توکن
TOKEN_ELSECONDITION	else
TOKEN_PRFUNC	print

قواعد نحوی زبان:

زبان ورودی دارای قواعد زیر است که هر برنامه‌ی نوشته شده‌ای به این زبان باید آن‌ها را رعایت کند:

۱. تمام متغیرهای گلوبال باید در ابتدای برنامه و به صورت Definition آورده شوند، بنابراین Declaration برای متغیرهای گلوبال وجود ندارد. منظور از Definition این است که مقداردهی

می‌شوند و در همان لحظه برای آن‌ها حافظه در نظر گرفته می‌شود. برای مثال هنگامی که می‌نویسیم int a = 3 در واقع متغیر a را Define کرده‌ایم اما هنگامی که می‌نویسیم ;int a و هیچ مقداردهی انجام نمی‌دهیم، متغیر a را Declare کرده‌ایم و صرفا حضور آن را به برنامه اعلام کرده‌ایم.

با توجه به نکات گفته شده، در این زبان، متغیرهای گلوبال باید به محض تعریف شدن یک مقداردهی اولیه نیز داشته باشند.

۲. متغیرهای گلوبال نمی‌توانند آرایه باشند.

۳. توابع نمی‌توانند دارای prototype باشند و باید به صورت یکجا بدنه‌ی آن‌ها نیز تعریف شود.

۴. برنامه حتما باید دارای یک تابع main باشد که برنامه از آن شروع شود.

۵. سینتکس این زبان دارای حلقه for به صورت زیر است و فرم دیگری ندارد:

```
Foreach i (0..5){
        Body
}
```

دقت کنید که متغیر i در اینجا همان شمارنده‌ی حلقه محسوب می‌شود.

۶. شرط if ممکن است در کدها وجود داشته باشد که ساختار آن به شکل زیر است:

```
If ( condition ) {
        Body
}
```

۷. ممکن است که این شرط با else نیز همراه شود:

```
If ( condition ) {
    Body
}else{
    Body
}
```

۸. فراخوانی تابع نیز وجود دارد که این عمل همانند شکل زیر انجام می‌پذیرد:

```
myFunc(arg1, arg2, arg3)
```

۹. اگر برنامه دارای تابعی به جز تابع main است، حتما باید قبل از تابع main قرار بگیرد.

۱۰. توابع می‌توانند آرگومان ورودی داشته باشند و یا نداشته باشند اما در صورت وجود آرگومان ورودی نباید تعداد آن‌ها از ۴ عدد تجاوز کند. یعنی اگر یک تابع ۵ عدد آرگومان ورودی داشته باشد به منزله‌ی نقض قواعد زبان است.
۱۱. در مورد تمام عملگرها، قوانین زبان c باید رعایت شود، با این تفاوت که عملگرهای +، ==، =!، >، <، برای داده‌های string نیز قابل اعمال هستند (به شرطی که عملوندهای شرکت کننده هر دو از نوع string باشند).
۱۲. در این زبان، تنها برای int و char عملیات auto casting را در نظر می‌گیریم؛ یعنی اگر یک متغیر از نوع int با یک متغیر از نوع double جمع بسته شود، منجر به تولید خطا می‌شود.
۱۳. آرایه‌ها در هنگام تعریف نیاز نیست که مقداردهی شوند، یعنی تضمین می‌شود که آرایه‌ها به شکل int a[3]; تعریف خواهند شد و اگر به شکل int a[2] = {0, 1} تعریف شوند به منزله‌ی نقض قواعد زبان است.
۱۴. تمام برنامه در قالب یک فایل داده خواهد شد.
۱۵. در صورت وجود خطا، تنها همان را چاپ شود و کامپایلر بدون تولید هر گونه کدی خارج شود.(کامپایل ادامه‌ی برنامه و پیدا کردن خطاهای دیگر همراه با نمره‌ی اضافه است، پیاده‌سازی این بخش خارج از حل‌تمرین‌های برگزار شده در مورد bison و flex است و خود شما باید در مورد آن تحقیق کنید.).
۱۶. قوانین وجود ’;‘ (که نشان دهنده‌ی پایان دستورات است) همانند زبان c است.

نکته‌ی بسیار مهم: چنانچه نوشتن بخشی از کامپایلر در توانتان نبود، لطفا پروژه را رها نکنید و بقیه‌ی قسمت‌ها را انجام دهید و در یک فایل به اسم notImplemented.txt توضیح دهید که چه بخش‌هایی را نتوانستید بنویسید تا کد شما متناسب با آنچه که نوشته‌اید تصحیح شود(تست کیس های مناسب با چیزی که تحویل داده‌اید به کامپایلر شما داده خواهد شد) و تمام نمره را از دست ندهید.

در صورتی که ابهامی در قوانین وجود دارد و یا قانونی ذکر نشده است با تی‌ای‌های خود در میان بگذارید.

رعایت کردن هر قاعده‌ی اضافه‌ای مستلزم هماهنگی با تی ای است وگرنه موجب کسر نمره می‌گردد برای مثال اگر زبان شما توانایی تعریف prototype را نیز داشته باشد ممکن است منجر به کسر نمره شود.

گرامر زبان ورودی:

همان‌گونه که می‌دانید اساسی‌ترین بخش یک تحلیل‌گر نحوی، گرامر آن است.

در زیر یک گرامر برای زبان ورودی آورده شده است که هدف از آن صرفا نشان دادن شمای کلی زبان است و ممکن است یک گرامر مبهم باشد و یا حتی نواقصی نیز داشته باشد، بنابراین تولید یک گرامر غیرمبهم بر عهده-ی شما است که می‌توانید از قابلیت‌های زبان بایسون(برای اولویت‌بندی) نیز استفاده کنید:

PROGRAM → GLOBAL_DECLARE PGM

GLOBAL_DECLARE → TYPE ID ASSIGN EXP GLOBAL_DECLARE | epsilon

PGM → TYPE ID '(' F_ARG ')' '{' STMTS '}' PGM | epsilon

F_ARG → TYPE ID ARRAY_VAR ',' F_ARG| TYPE ID ARRAY_VAR | epsilon

STMTS → STMT STMTS | epsilon

STMT → STMT_DECLARE | STMT_ASSIGN | STMT_RETURN | LOOP | CONDITION | CALL | EXP | PRINTFUNC | ';' STMT | epsilon

PRINTFUNC → PRINT '(' EXP ')' STMT

EXP → EXP '<' EXP

EXP → EXP '<=' EXP

EXP → EXP '>' EXP

EXP → EXP '>=' EXP

EXP → EXP '!-' EXP

EXP → EXP '==' EXP

EXP → EXP '+' EXP

EXP → EXP '-' EXP

EXP → EXP '*' EXP

EXP → EXP '/' EXP

EXP → EXP '&&' EXP

EXP → EXP '||' EXP

EXP → EXP ‘<’ EXP

EXP → EXP ‘<=’ EXP

EXP → EXP ‘|’ EXP

EXP → EXP ‘&’ EXP

EXP → EXP ‘^’ EXP

EXP → ‘-‘ EXP

EXP → ID ‘[‘ EXP ‘]’

EXP → ‘!’ EXP

EXP → ‘(‘ EXP ‘)’

EXP → ID

EXP → NUM

EXP → STRINGCONST

EXP → CHARCONST

EXP → FLOATCONST

LOOP → FOREACH ID ‘(‘ EXP ‘..’ EXP ‘)’ ‘{‘ STMT ‘}’ STMT

CONDITION → IF ‘(‘ EXP ‘)’ ‘{‘ STMT ‘}’ STMT | ELSECON

ELSECON → ELSE ‘{‘ STMT ‘}’ STMT

STMT_RETURN → RETURN EXP ‘;’

STMT_DECLARE → TYPE ID ARRAY_VAR IDS

ARRAY_VAR → ‘[‘ EXP ‘]’ | ‘[‘ ‘]’ |epsilon

IDS → ‘;’ STMT | ‘,’ ID ARRAY_VAR IDS

CALL → ID ‘(‘ ARGS ‘)’ STMT

ARGS → ID ARRAY_VAR ARGS | ‘,’ ID ARRAY_VAR ARGS | epsilon

```
STMT_ASSIGN → ID ARRAY_VAR '=' EXP STMT
```

اصلی‌ترین بخش پروژه، تهیه گرامر مناسب است و جزییات این بخش به راحتی قابل اعمال و پیاده سازی است. اگر گرامر را به درستی انتخاب کنید می‌تواند حجم کار شما را تا حد زیادی کاهش دهد بنابراین توصیه می‌شود که به صورت روزانه، وقتی را صرف نوشتن یک گرامر غیر مبهم برای این زبان کنید و آن را به دقایق آخر موکول نکنید.

آرگومان‌های ورودی تحلیل‌گر:

تحلیل‌گر شما باید یک آرگومان ورودی همانند زیر دریافت کند:

```
$ ./semanticParser      printByTokenName
```

در صورتی که مقدار ۱ به برنامه داده شد، باید نام توکن برای ترمینال‌ها نمایش داده شود و در صورتی که ۰ داده شد، باید مقدار آن چاپ شود. برای مثال: با ورودی عدد ۱ باید به ازای دیدن کلمه‌ی ورودی if عبارت TOKEN_IFCONDITION هنگام چاپ درخت نمایش داده شود و در صورتی که ورودی ۰ باشد باید خود مقدار آن، یعنی if، نمایش داده شود.

نمره‌ی اضافه:

نمایش درخت به صورت دوبعدی (همانند آنچه که در کلاس آموخته‌اید) دارای نمره اضافه است.

نمایش دوبعدی باید به صورت عمودی باشد، برای مثال:

PROGRAM

|

| |

GLOBAL_DECLARE PROGRAM