دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

اصول طراحی کامپایلر (دکتر ممتازی)

نیم‌سال اول سال تحصیلی ۱۴۰۱-۱۴۰۲

پروژه پایانی

دانشکده مهندسی کامپیوتر

در این پروژه از شما می‌خواهیم تا به کمک PLY در Python یا Lex/Yacc در C یا JFlex/Cup در Java کامپایلری برای زبانی که در ادامه شرح داده می‌شود بنویسید تا کد زبان C [1] تولید کند. گرامر این زبان در ادامه آورده شده است:

⟨*program*⟩ ::= `program` ⟨*identifier*⟩ ⟨*declarations*⟩ ⟨*compound-statement*⟩

⟨*declarations*⟩ ::= `var` ⟨*declaration-list*⟩
| ⟨*empty*⟩

⟨*declaration-list*⟩ ::= ⟨*identifier-list*⟩ `:` ⟨*type*⟩
| ⟨*declaration-list*⟩ `;` ⟨*identifier-list*⟩ `:` ⟨*type*⟩

⟨*identifier-list*⟩ ::= ⟨*identifier*⟩
| ⟨*identifier-list*⟩ `,` ⟨*identifier*⟩

⟨*type*⟩ ::= `int`
| `real`

⟨*compound-statement*⟩ ::= `begin` ⟨*statement-list*⟩ `end`

⟨*statement-list*⟩ ::= ⟨*statement*⟩
| ⟨*statement-list*⟩ `;` ⟨*statement*⟩

⟨*statement*⟩ ::= ⟨*identifier*⟩ `:=` ⟨*expression*⟩
| `if` ⟨*expression*⟩ `then` ⟨*statement*⟩ `else` ⟨*statement*⟩
| `if` ⟨*expression*⟩ `then` ⟨*statement*⟩
| `while` ⟨*expression*⟩ `do` ⟨*statement*⟩
| ⟨*compound-statement*⟩
| `print (` ⟨*expression*⟩ `)`
| `switch` ⟨*expression*⟩ `of` ⟨*cases*⟩ ⟨*default-case*⟩ `done`

[1] Source-to-source compilers translate a high level language to another high level language.

⟨*default-case*⟩ ::= `default` ⟨*statement*⟩ `;`

| ⟨*empty*⟩

⟨*cases*⟩ ::= ⟨*constant-list*⟩ `:` ⟨*statement*⟩ `;` ⟨*cases*⟩

| ⟨*empty*⟩

⟨*constant-list*⟩ ::= ⟨*constant*⟩

| ⟨*constant-list*⟩ `,` ⟨*constant*⟩

⟨*constant*⟩ ::= ⟨*real*⟩

| ⟨*integer*⟩

⟨*expression*⟩ ::= ⟨*integer*⟩

| ⟨*real*⟩

| ⟨*identifier*⟩

| ⟨*expression*⟩ `+` ⟨*expression*⟩

| ⟨*expression*⟩ `-` ⟨*expression*⟩

| ⟨*expression*⟩ `*` ⟨*expression*⟩

| ⟨*expression*⟩ `/` ⟨*expression*⟩

| `-` ⟨*expression*⟩

| ⟨*expression*⟩ `mod` ⟨*expression*⟩

| ⟨*expression*⟩ `<` ⟨*expression*⟩

| ⟨*expression*⟩ `=` ⟨*expression*⟩

| ⟨*expression*⟩ `>` ⟨*expression*⟩

| ⟨*expression*⟩ `<>` ⟨*expression*⟩

| ⟨*expression*⟩ `<=` ⟨*expression*⟩

| ⟨*expression*⟩ `>=` ⟨*expression*⟩

| ⟨*expression*⟩ `and` ⟨*expression*⟩

| ⟨*expression*⟩ `or` ⟨*expression*⟩

| `not` ⟨*expression*⟩

| `(` ⟨*expression*⟩ `)`

توکن‌هایی که تحلیلگر لغوی برای این زبان تولید می‌کند چهار دسته هستند: کلیدواژه‌ها [۲] ، جدا کننده‌ها [۳] و عملگرها [۴] ، شناسه‌ها [۵] و اعداد. کلید واژه‌ها و جدا کننده‌ها در گرامر **bold** شده‌اند. شناسه‌ها که با identifier مشخص شده‌اند مانند همه زبان‌های برنامه نویسی می‌توانند شامل حروف

[2] keywords

[3] Delimiters are parentheses, semicolons, colons, commas, and periods.

[4] Operators are relational (<, >, <>, =, <=, >=), arithmetic (+, -, *, /), and assignment (:=).

[5] identifiers

کوچک و بزرگ انگلیسی، ارقام و _ باشند اما با ارقام نمی‌توانند شروع شوند. اعداد به صورت صحیح و حقیقی هستند. اعداد صحیح با صفر شروع نمی‌شوند. بخش صحیح و اعشاری اعداد حقیقی با نقطه از هم جدا شده‌اند. دقت کنید اعداد .12 یا 15. معتبر نیستند. همچنین منظور از empty همان رشته به طول صفر است.

دو ابهام مهم در این زبان وجود دارد. یکی مربوط به dangling else است و دیگری مربوط به اولویت عملگرها است. ابهام اول را با روش nearest if رفع کنید. (این استراتژی را به کمک PLY پیاده کنید. لازم نیست گرامر را تغییر دهید.) در این استراتژی که در اکثر زبان‌های برنامه نویسی مانند C استفاده می‌شود، یک else مبهم متناظر با نزدیکترین if می‌شود.

به عنوان مثال این استراتژی عبارت if E_1 then if E_2 then S_1 else S_2 را به این صورت معنا می‌کند که اگر E_1 برقرار نباشد، هیچ یک از S_1 و S_2 اجرا نخواهند شد.

برای رفع ابهام دوم به این نکته توجه کنید که عملگرهای منطقی یعنی and، or و not فقط روی عبارت‌هایی با ارزش بولی کار می‌کنند. یعنی عبارتی مانند 3 or 5 معنا ندارد. به طور مشابه عملگرهای حسابی فقط روی عبارت‌هایی با ارزش عددی می‌توانند کار کنند. به عنوان مثال عبارت $1 + (2 < 3)$ معتبر نیست. همچنین عملگرهای مقایسه‌ای $<, >, =, <>, <=, >=$ نیز فقط روی مقادیر عددی کار می‌کنند. به عنوان مثال $(1 < 2) = (3 < 4)$ معتبر نیست. دقت کنید عملگرهای مقایسه‌ای شرکت پذیر نیز نیستند.

در نتیجه اولویت عملگرهای $+, -, *, /, \text{mod}$ نسبت به هم معنا دارد و همین طور اولویت عملگرهای and, or, not نسبت به یک دیگر معنا دارند. موارد زیر را در گرامر رفع ابهام شده در نظر بگیرید:

- عملگرهای حسابی همگی شرکت پذیر به راست و اولویت ضرب و تقسیم از جمع و منها بیشتر است. اولویت منفی تک عملوندی نیز از همه آنها بیشتر است.
- عملگرهای منطقی نیز همگی شرکت پذیر به راست و اولویت and از or بیشتر است. اولویت not به عنوان عملگر تک عملوندی از هر دو بیشتر است.
- هر دو عملوند عملگر mod باید صحیح باشند. این عملگرها شرکت پذیر نبوده و اولویتشان با * و / برابر است.

به کمک این قوانین می‌توانید این دو ابهام گرامر را رفع کنید. برای رفع دیگر ابهام‌های احتمالی، تغییرات گرامر را به نحوی انجام دهید که زبان تولید شده توسط گرامر تغییر نکند.

عملگر mod باقی مانده عملوند اول بر دوم را محاسبه می‌کند. عملگر = تساوی دو عدد و عملگر <> عدم تساوی دو عدد را مشخص می‌کند.

عبارت‌هایی که به عنوان شرط در if یا while ظاهر می‌شوند باید دارای ارزش بولی باشند. همچنین در دستور assignment تایپ طرفین باید مطابقت داشته باشند. دستور print مقدار عبارت ورودی (که باید تایپ صحیح داشته باشد.) را در یک خط جداگانه چاپ می‌کند.

دقت کنید می‌توانید فرض کنید این قوانین مربوط به تایپ‌ها رعایت شده‌اند. در نتیجه بررسی تایپ‌ها و اعلام خطاهای معنایی مربوط به تایپ‌ها توسط کامپایلر نمره امتیازی دارد. (به عنوان قاعده کلی در هر کجا که براساس قوانین گفته شده تایپ‌ها تطابق نداشته باشند، کامپایلر باید خطای نحو صادر کند. دقت کنید می‌توانید این بررسی را همزمان با تولید کد و به کمک جدول نمادها انجام دهید و هر کجا که با خطا مواجه شدید کار را متوقف کرده و خطای نحوی صادر کنید.)

دقت کنید گرامر و زبان به صورت کامل برای درک بهتر شما توصیف شده است ولی:

- تولید کد برای switch-case امتیازی است.

- تولید کد با فرض وجود تایپ اعداد حقیقی در زبان امتیازی است. دقت کنید تقسیم دو عدد صحیح مانند زبان C همواره تایپ صحیح دارد و مقدار آن برابر تقسیم صحیح آن دو عدد است. تنها وجود اعداد حقیقی به عنوان یکی از عملوندهای عملگرهای ریاضی موجب تغییر تایپ عبارت می‌شود.

برنامه‌ای که خروجی می‌دهید باید به فرم زیر باشد:

```
#include<stdio.h>
int iid_1, iid_2, ... , iid_n;
float fid_1, fid_2, ... , fid_n;
int temp_int_1, temp_int_2, ... , temp_int_n;
float temp_float_1, temp_float_2, ... , temp_float_n;
int main()
{
    statement_1;
    label_1: statement_2;
    ...
    statement_n;
}
```

هر یک از statement ها باید به یکی از فرم‌های زیر باشند:

1. `x = y op z;` ($op \in \{+, -, *, /, \%, \&\&, ||\}$)
2. `x = op y;` ($op \in \{-, !\}$)
3. `x = y;`
4. `goto label;`
5. `printf("%d\n", x);`
6. `if (x relop y) goto label;` ($relop \in \{<, >, <=, >=, ==, !=\}$)

که x، y، z یکی از متغیرهای تعریف شده در ابتدای برنامه هستند و label نام یکی از برچسب‌های تعریف شده است. دقت کنید هیچ دستور دیگری مجاز نیست و در ابتدای هر statement حداکثر یک label می‌تواند قرار بگیرد. همچنین نام متغیرهای موقت دقیقا باید مانند بالا باشد و صرفا از اندیس می‌توانید استفاده کنید. اما نام متغیرهای اصلی می‌توانند همان نام‌های تعریف شده در برنامه ورودی باشند.