به نام خدا

# فاز چهارم پروژه کامپایلرها و زبان‌های برنامه‌نویسی

پاییز ۹۹

مهلت تحویل: ۲۴ دی

در این فاز بخش‌های مربوط به تولید کد را به کامپایلر خود اضافه می‌کنید. در انتهای این فاز، کامپایلر شما به طور کامل پیاده‌سازی شده و برنامه‌های نوشته شده به زبان Sophia را به کد قابل اجرا توسط ماشین تبدیل می‌کند. پیاده‌سازی شما باید به ازای هر فایل ورودی به زبان Sophia، بایت کد معادل آن را تولید کند. در تست‌های این فاز، صرفا قابلیت تولید کد کامپایلرتان سنجیده می‌شود و ورودی‌ها دارای خطاهای نحوی و معنایی که در فازهای قبل بررسی کردید نیستند؛ اما توجه کنید که شما برای تولید کد به اطلاعات جمع آوری شده در جدول علائم و اطلاعات مربوط به تایپ نودهای درخت AST نیاز دارید.

## اسمبلر

جهت تولید فایل‌های class. نهایی از شما انتظار نمی‌رود که فایل باینری را مستقیما تولید کنید. برای این کار می‌توانید از اسمبلر jasmin که در کلاس معرفی شده است استفاده کنید.

## تساوی اشیاء

برای تایپ‌های int و boolean آنها را با استفاده از مقادیرشان با دستور if_icmpeq مقایسه می‌کنیم و برای تایپ‌های دیگر از دستور if_acmpeq برای مقایسه استفاده می‌کنیم.

## عملگرهای && و | |

شما باید این عملیات را به صورت short-circuit پیاده‌سازی کنید.

## نکات کلی پیاده‌سازی

- برای آن که لیست multi-type را در جاوا داشته باشیم نیاز داریم که یک لیست از جنس Object (که والد تمام کلاس‌ها است) داشته باشیم تا هر نوعی را بتوان در آن ذخیره کرد. کلاس‌های جاوا مانند Integer و Boolean، از Object ارث می‌برند و بنابراین می‌توان آن ها را در لیستی از Object ذخیره کرد. ولی تایپ int و boolean که تایپ‌های primitive هستند را نمی‌توان در این لیست ذخیره کرد. بدین منظور تایپ‌های int و boolean را در expression ها باید از نوع primitive استفاده کنیم تا بتوان operator ها را روی آنها اعمال کرد و در نهایت آنها را به غیر primitive تبدیل کنیم تا بتوانیم آنها را در لیست‌ها ذخیره کنیم. در ادامه جزئیات این تبدیل توضیح داده می‌شود.
- نوع بازگشتی ویزیتورهای CodeGenerator از نوع String قرار داده شده است. می‌توانید در هر ویزیتور یا command های تولید شده توسط آن ویزیتور را مستقیما با addCommand در فایل اضافه کنید یا اینکه مجموعه command ها را که به صورت string هستند و با \n جدا شده اند return کنید و در تابع دیگری آنها را به فایل اضافه کنید. پیشنهاد می‌شود ویزیتورهای expression مجموعه command هایشان را return کنند و دیگر ویزیتورها با گرفتن آن command ها آنها را در فایل اضافه کنند.
- در کلاس‌ها و متدها، نوع تمام تایپ‌های primitive مانند int یا bool یا string را از نوع‌های غیر primitive جاوا تولید کنید. یعنی در بایت کد تولید شده باید این متغیرها از نوع‌های Integer یا Boolean یا String باشند که در java/lang هستند.
- برای boolean ها در استک، اگر true باشد ۱ و اگر false باشد ۰ اضافه کنید.
- برای اضافه کردن مقادیر primitive به استک، از دستور ldc استفاده کنید. برای string quotation ("") آن را هم در دستور ldc بیاورید.
- برای انجام محاسبات مانند add یا or روی Integer یا Boolean باید آنها از نوع primitive یعنی int یا bool باشند. پس در تمام expression ها از نوع primitive این دو تایپ استفاده کنید و در هنگام نوشتن آنها در یک متغیر یا پاس دادن به توابع یا ریترن شدن آنها، این دو تایپ را از primitive به غیر primitive تغییر دهید. همچنین بعد از خواندن این دو نوع از متغیر یا لیست باید تبدیل انجام شود. دلیل تبدیل‌ها آن است که در تعریف، متغیرها از نوع غیر primitive تعریف شده‌اند و در expression ها ما نیاز به primitive داریم (توابع jasmin برای تبدیل آنها در ادامه آمده است).
- طول stack و locals را در متدها 128 قرار دهید.
- فایل‌های Fptr.j و List.j در اختیارتان قرار گرفته اند و برای کار با لیست‌ها و Fptr ها باید از این دو کلاس آماده استفاده کنید. همچنین معادل java آنها نیز داده شده است تا بتوانید متدهای آنها را مشاهده کنید که چه کاری انجام می‌دهند. برای Fptr در هنگام

دسترسی به متد یکی از این کلاس ساخته می‌شود و instance و نام متد در آن قرار داده می‌شود. سپس در هنگام call شدن متد باید تابع invoke از این کلاس را با آرگومان‌های پاس داده شده صدا بزنید. توجه داشته باشید که باید آرگومان‌ها را در یک ArrayList ذخیره کرده و به این تابع پاس دهید.

- تمام value های لیست هنگام پاس داده شدن به تابع یا assign شدن در یک لیست جدید کپی می‌شود. برای این کار از constructor دوم که copy constructor است استفاده کنید. همچنین برای گرفتن یا ست کردن المان می‌توانید از توابع مربوطه استفاده کنید. توجه داشته باشید که خروجی getElement یک Object است و بعد از استفاده از این تابع باید خروجی را به تایپ المانی که گرفته اید cast کنید (دستور cast در jasmin در ادامه آورده شده است).
- در ابتدای هر نوع constructor ای (default constructor یا تعریف شده)، ابتدا فیلدهای آن کلاس باید initialize شوند. مقادیر را به صورت زیر در نظر بگیرید: برای int مقدار صفر، برای string مقدار ""، برای bool مقدار false، برای class یا Fptr مقدار null و برای لیست یک instance از کلاس List که داخل element های آن متناسب با تایپ‌های تعریف شده برای آن لیست، دوباره به صورت بازگشتی مقداردهی می‌شوند.
- در صورتی که ObjectOrListMemberAccess داشته باشیم که عضو یک لیست با نام المان گرفته شده، باید index آن را پیدا کرده و از آن استفاده کنیم.
- نام کلاس‌ها (مثلا در signature ها یا در هنگام cast) به صورت زیر است:

| | |
|---|---|
| IntType → java/lang/Integer | ListType → List |
| BoolType → java/lang/Boolean | FptrType → Fptr |
| StringType → java/lang/String | ClassType → ClassName |

- در اضافه کردن command ها حواستان به \n ها باشد تا command ها پشت هم در فایل jasmin نباشند. همچنین هر command ای که اضافه می‌کنید به طور دقیق بررسی کنید که چه آرگومان‌هایی لازم دارد و چه چیزی ریترن می‌کند؛ زیرا اگر اشتباهی رخ دهد debug کردن آن در فایل‌های jasmin کار دشواری است. برای راحت تر debug کردن و فهمیدن فایل‌های jasmin بهتر است آنها را مرتب بسازید؛ مثلا بین متدها یک خط خالی بگذارید تا مشخص باشند.
- برای مشاهده مجموعه دستورات بایت کد به این لینک می‌توانید مراجعه کنید.

## نکات ویزیتورها و توابع

### slotOf

در این تابع برای متغیرها باید slot آنها را برگردانید. slot صفر به صورت پیش‌فرض برای خود کلاس (this) است و بعد از آن باید به ترتیب آرگومان‌های تابع و local variable ها باشند. طوری این تابع را پیاده‌سازی کنید که اگر ورودی یک string خالی بود، یک slot بعد از تمام slot های مخصوص variable ها برگرداند. این برای استفاده از یک متغیر temp در code generation استفاده می‌شود؛ یعنی یک متغیر که برای تبدیل سوفیا به جاوا اضافه شده است. توجه کنید که ممکن است یک تابع به چند temp variable نیاز داشته باشد.

### addDefaultConstructor

یک default constructor به فایل اضافه کنید. فیلدهای کلاس باید initialize بشوند و constructor والد نیز صدا شود.

### addStaticMainMethod

یک متد static main به فایل اضافه کنید. این متد توسط جاوا شناسایی شده و ابتدا این تابع در پروژه اجرا می‌شود. در این تابع باید از کلاس Main یک instance بگیرید و constructor آن را صدا بزنید (خود این تابع در فایل کلاس Main قرار می‌گیرد).

### Program

همه‌ی کلاس‌ها ویزیت شوند. current class را در Code Generator و Expression Type Checkerست کنید.

### ClassDeclaration

فایل کلاس متناظر را با تابع createFile بسازید و سپس header مربوط به کلاس را اضافه کنید. اگر کلاس parent نداشت، parent آن را java/lang/Object قرار دهید. سپس فیلدها را ویزیت کنید. اگر constructor دارد آن را ویزیت کنید و در غیر این صورت یک default constructor اضافه کنید. در نهایت متدها را ویزیت کنید. قبل از ویزیت کردن متد یا constructor، current method را در Code Generator و Expression Type Checker ست کنید.

### ConstructorDeclaration

اگر constructor حداقل یک آرگومان می‌گیرد باید یک default constructor علاوه بر آن constructor به کلاس اضافه شود. یعنی تمام کلاس‌ها دقیقا یک constructor بدون آرگومان و یک یا صفر constructor با آرگومان خواهند داشت. اگر کلاس main است باید یک static main method هم به کلاس اضافه شود.

### MethodDeclaration

header های مربوطه را متناسب یا متد یا constructor بودن اضافه کنید. اگر constructor است، constructor والدش را صدا زده و فیلدهایش را نیز initialize کنید (اگر والدی نداشت constructor کلاس Object باید صدا زده شود). سپس متغیرهای local را ویزیت کنید و initialize کنید (آرگومان‌ها را initialize نکنید). سپس statement ها را ویزیت کنید. دقت کنید که اگر تابع ریترن نمی‌کند، نیاز است که حتما یک دستور return در انتهای command های متد قرار داده شود. برای اینکه بفهمید متد ریترن می‌کند یا خیر از getDoesReturn روی methodDeclaration استفاده کنید که در فاز قبل (در بخش امتیازی) ست شده است.

### FieldDeclaration

دستورات مربوط به field اضافه می‌شوند.

### VarDeclaration

دستورات مربوط به initialize کردن متغیر با توجه به تایپ آن اضافه می‌شوند.

### AssignmentStmt

در این قسمت می‌توانید از روی assignment statement یک node از جنس assignment expression ساخته و آن را ویزیت کنید. توجه داشته باشید که باید در انتهای ویزیت مقداری که assignment expression روی stack قرار می‌دهد را pop کنید.

### BlockStmt

تمام statement ها را ویزیت کنید.

### ConditionalStmt

دستورات مورد نیاز برای یک شرط را اضافه کنید.

### MethodCallStmt

می‌توانید methodCall داخل آن را ویزیت کنید و خروجی آن را pop کنید. توجه داشته باشید که قبل و بعد از ویزیت متد کال،
expressionTypeChecker.setIsInMethodCallStmt را صدا بزنید و در ابتدا آن را true و در انتها آن را false کنید که هنگام استفاده از expressionTypeChecker در ویزیتورها، مشکلی پیش نیاید.

### PrintStmt

توابع مورد نیاز print را اضافه کنید. با استفاده از expressionTypeChecker می‌توانید تایپ آرگومان را بگیرید و از signature مناسب برای print استفاده کنید. جهت نوشتن بر روی صفحه‌ی نمایش باید از print در کتابخانه‌ی PrintStream.io.java استفاده کنید.

### ReturnStmt

دستورات مربوط به ریترن را اگر void نیست اضافه کنید. توجه کنید که اگر expression جلویreturn از نوع IntType یا BoolType است، ابتدا باید از primitive به غیر primitive تبدیل شود.

### BreakStmt

به label خروج حلقه‌ی فعلی بروید.

### ContinueStmt

به label شروع حلقه‌ی فعلی بروید.

### ForeachStmt

باید foreach را به for تبدیل کنید. برای این کار یک slot برای یک temp variable که متغیر حلقه خواهد بود بگیرید (این متغیر از جنس int است). سپس معادل foreach را که یک forروی یک لیست است بیابید و command های قسمت‌های مربوط به initialization متغیر حلقه، شرط حلقه (متغیر temp گرفته شده از طول لیست پیمایش شونده کمتر باشد)، update (اندیس temp از آرایه گرفته شده با تابع getElement در کلاس List و در پیمایش کننده ریخته شود و cast کردن بعد از آن فراموش نشود و temp هم باید آپدیت شود) و body را باید اضافه کنید.

### ForStmt

مانند قسمت قبل command های مناسب را اضافه کنید. توجه داشته باشید که هر یک از initialization یا condition یا update می‌تواند وجود نداشته باشد.

### BinaryExpression

برای هر یک از عملگرها دستورات مناسب را اضافه کنید. برای assign، ابتدا بررسی کنید اگر list دارد assign می‌شود، بعد از مجموعه دستورات operand سمت راست، دستوراتی اضافه کنید که از این لیست که توسط دستورات operand دوم به stack اضافه شده، یک لیست کپی ساخته شود (با copy constructor در کلاس List). سپس با توجه به اینکه سمت چپ identifier یا ListAccessByIndex یا ObjectOrListMemberAccess است، دستورات assign را اضافه کنید. در حالت ObjectOrListMemberAccess دوباره دو حالت اینکه instance آن ListType باشد یا ClassType داریم که باید جداگانه پیاده‌سازی شوند. توجه داشته باشید که assign باید مقدار حاصل از دستورات operand سمت راست را که داخل operand سمت چپ ذخیره می‌شود در نهایت در stack قرار دهد.

### UnaryExpression

برای هر یک از عملگرها دستورات مناسب باید اضافه شوند. توجه کنید که برای predec، postdec، preinc و postinc دوباره همان تقسیم‌بندی‌های assign را داریم؛ زیرا این دستورات یک مقدار جدید به آن متغیر می‌دهند.

### ObjectOrListMemberAccess

بررسی می‌شود که اگر نوع instance آن کلاس است، متناسب با اینکه آن ممبر field یا متد است کد مناسب ساخته شود. اگر هم که لیست است کد مناسب باید ساخته شود.

### Identifier

از slot متناسب با آن identifier باید مقدار load شود (با aload). سپس اگر نوع آن int type یا bool type است تبدیل به primitive شود.

### ListAccessByIndex

با استفاده از getElement آن اندیس مورد نظر گرفته شده و سپس به تایپ مناسب cast می‌شود و اگر int یا boolean است دوباره به primitive تبدیل می‌شود.

### MethodCall

یک ArrayList ابتدا new شده و مقادیر آرگومان‌ها بعد از visit، به این لیست add می‌شود (با java/util/ArrayList/add) و سپس با استفاده از این لیست تابع invoke از instance صدا زده می‌شود. در نهایت خروجی آن به تایپ مناسب cast شده و در صورت boolean یا intبودن تبدیل به غیر primitive می‌شود. توجه داشته باشید آرگومان‌ها بعد از visit شدن و قبل از اضافه شدن به ArrayList، اگر int یا bool هستند باید به غیر primitive تبدیل شوند.

### NewClassInstance

بعد از قرار دادن آرگومان‌های constructor روی stack، آن constructor صدا زده می‌شود. توجه داشته باشید آرگومان‌ها بعد از visit شدن، اگر int یا bool هستند باید به غیر primitive تبدیل شوند (زیرا نوع آرگومان تابعی که صدا زده می‌شود مثلا Integer است نه int).

### ThisClass

reference به خود کلاس باید روی stack قرار داده شود.

### ListValue

در این قسمت باید یک ArrayList جدید ساخته شده و آرگومان‌ها پس از visit شدن و تبدیل شدن به غیر primitive به آن اضافه شوند. سپس با استفاده از این ArrayList یک List ساخته شود (با استفاده از constructor اول کلاس List).

### NullValue

مقدار null باید روی stack گذاشته شود.

### IntValue

با ldc باید مقدار آن روی stack گذاشته شود.

### BoolValue

با ldc باید مقدار آن (۰ یا ۱) روی stack گذاشته شود.

### StringValue

با ldc باید مقدار آن (همراه quotation) روی stack گذاشته شود.

## دستورات کاربردی jasmin

تبدیل int به Integer

```
invokestatic java/lang/Integer/valueOf(I)Ljava/lang/Integer;
```

تبدیل bool به Boolean

```
invokestatic java/lang/Boolean/valueOf(Z)Ljava/lang/Boolean;
```

تبدیل Integer به int

```
invokevirtual java/lang/Integer/intValue()I
```

تبدیل Boolean به bool

```
invokevirtual java/lang/Boolean/booleanValue()Z
```

اضافه کردن به ArrayList

```
invokevirtual java/util/ArrayList/add(Ljava/lang/Object;)Z
```

گرفتن سایز ArrayList

```
invokevirtual java/util/ArrayList/size()I
```

تبدیل (cast) یک Object به یک کلاس A

```
checkcast A
```

## دستورات تبدیل و اجرای کدها

کامپایل کردن فایل java. به به فایل class.

```
javac -g *.java
```

اجرای فایلclass. در نهایت باید Main.class اجرا شود

```
java Main
```

تبدیل فایل بایت کد (.j) jasmin به فایل class.

```
java -jar jasmin.jar *.j
```

تبدیل فایل class. به بایت کد جاوا (نه jasmin) که خروجی در ترمینال نمایش داده می‌شود

```
javap -c -l A
```

تبدیل فایل class. به بایت کد jasmin که خروجی در ترمینال نمایش داده می‌شود

```
java -jar classFileAnalyzer.jar A.class
```

تبدیل class. به کد جاوا

```
drag the .class file to intellij window
```

می‌توانید با استفاده از دستورات بالا برای هر کد سوفیا که می‌خواهید معادل jasmin آن را پیدا کنید به این صورت عمل کنید که ابتدا معادل java آن کد سوفیا را بنویسید. سپس آن فایل جاوا را کامپایل کنید که class. تولید شود. سپس این فایل را با classFileAnalyzer به بایت کد jasmin تبدیل کنید. فقط به این نکته توجه کنید که این classFileAnalyzer یک پروژه از github بوده و لزوما خروجی صحیحی نمی‌دهد و باید بررسی شود (در اکثر موارد خروجی درست می‌دهد مگر چند مورد خاص).

## نمونه خروجی

نمونه خروجی حاصل از کامپایل و اجرای صحیح کد نمونه شماره ۲ (که در اختیارتان گذاشته شده است) به صورت زیر است:

```
-------------------------Compiling-------------------------
Compilation successful

------------------Generating Class Files------------------
Generated: Fptr.class
Generated: List.class
Generated: Main.class
Generated: WordProcessor.class

--------------------------Output--------------------------
the total number of names: 100
```

---

## نکات مهم:

- در این فاز شما باید کد visitor مربوط به CodeGenerator که بخشی از آن به شما داده شده را تکمیل کنید. در نهایت **تنها یک فایل** **CodeGenerator.java** (بدون تغییر نام) آپلود کنید. توجه شود که تنها **یک نفر** از هرگروه باید پروژه را آپلود کند. در صورت عدم رعایت این موارد از شما نمره **کسر خواهد شد**.
- در صورت کشف هر گونه تقلب، نمره 100- لحاظ می‌شود.
- دقت کنید که خروجی برنامه شما به صورت خودکار تست می‌شود.
- در صورتی که قبل از کلاس‌های رفع اشکال سوالی برایتان پیش آمد، می‌توانید زودتر آنها را بپرسید تا ابهامات شما در ابتدای زمان پروژه برطرف شود و به ساعات پایانی موکول نشود. بهتر است سوالات خود را در فروم یا گروه درس مطرح نمایید تا دوستانتان نیز از آنها استفاده کنند؛ در غیر این صورت به مسئولان پروژه ایمیل بزنید:

امیر پورمحمدعلی amir.pma1378@gmail.com

مبینا شاه‌بنده shbmobina@gmail.com