به نام خدا

فاز چهارم پروژه کامپایلرها و زبان‌های برنامه‌نویسی

بهار 1400

مهلت تحویل : 4 تیر

در این فاز بخش های مربوط به تولید کد را به کامپایلر خود اضافه می کنید. در انتهای این فاز، کامپایلر شما به طور کامل پیاده سازی شده و برنامه های نوشته شده به زبان Jepeto را به کد قابل اجرا توسط ماشین تبدیل می کند. پیاده سازی شما باید به ازای هر فایل ورودی به زبان Jepeto ، بایت کد معادل آن را تولید کند. در تست های این فاز، صرفا قابلیت تولید کد کامپایلر تان سنجیده می شود و ورودی ها 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 نمی رود که فایل باینری را مستقیما تولید کنید. برای این کار می توانید از اسمبلر jasmin که در کلاس درس معرفی شده است استفاده کنید.

تساوی اشیاء

برای تایپ های int و boolean آن ها را با استفاده از مقادیرشان با دستور if_icmpeq مقایسه می کنیم و برای تایپ های دیگر از دستور if_acmpeq برای مقایسه استفاده می کنیم.

عملگرهای and و or

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

کلاس Main

کل برنامه در قالب یک کلاس به نام Main پیاده سازی میشود و توابع برنامه به عنوان توابع آن کلاس هستند . mainDeclaration هم constructor این کلاس می باشد.

نکات کلی پیاده سازی

- برای پیاده سازی لیست را در جاوا، 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 می برند و بنابراین می توان آن ها را در لیستی از Object ذخیره کرد ولی تایپ int و boolean که تایپ های primitive هستند را نمی توان در این لیست ذخیره کرد. بدین منظور تایپ های 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 لیست ها 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 آن ها را در فایل اضافه کنند.

- در فانکشن ها، نوع تمام تایپهای primitive مانند int یا bool یا string را از نوع های غیر primitive جاوا تولید کنید. یعنی در بایت کد تولید شده باید این متغیرها از نوع های Integer یا Boolean یا String باشند که در java/lang هستند.

- برای boolean ها در استک، اگر true باشد 1 و اگر false باشد 0 اضافه کنید.

- برای اضافه کردن مقادیر primitive به استک، از دستور ldc استفاده کنید. برای string کوتیشن ("") آن را هم در دستورldc بیاورید.

- طول stack و locals را در متدها 128 قرار دهید.

- برای انجام محاسبات 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 Returnشدن آن ها، 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 تبدیل ها آن است که در تعریف، متغیرها از نوع غیر primitive تعریف شده اند و در expression ها ما نیاز به Primitive داریم (توابع jasmin برای تبدیل آنها در ادامه آمده است)

- فایلهای Fptr.j و List.j در اختیارتان قرار گرفته اند و برای کار با لیست ها و Fptr ها باید از این دو کلاس آماده استفاده کنید. همچنین معادل java آنها نیز داده شده است تا بتوانید متدهای آن ها را مشاهده کنید که چه کاری انجام میدهند. Fptr در هنگام دسترسی به یکی از تابع ها ساخته می شود و instance و نام تابع در آن قرار داده میشود. سپس در هنگام call شدن باید تابع invoke از این کلاس را با آرگومان های پاس داده شده صدا بزنید. توجه داشته باشید که باید آرگومان ها را در یک ArrayList ذخیره کرده و به این تابع پاس دهید.

- تمام value های لیست هنگام پاس داده شدن به تابع 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) .

❖ نام کلاس ها (مثلا در signature ها یا در هنگام cast) به صورت زیر است:

ListType → List

IntType → java/lang/Integer

FptrType → Fptr

BoolType → java/lang/Boolean

StringType → java/lang/String

❖ 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 آرگومان هایی لازم دارد و چه مقداری را باز می گرداند؛ زیرا اگر اشتباهی رخ دهد debug کردن آن در فایلهای jasmin کار دشواری است.

❖ توجه کنید که دستور اضافه کردن به arraylist که در جلوتر آمده است، یک مقدار bool برمیگرداند و روی استک میگذارد. دقت کنید در صورت لزوم این مقدار باید pop شود.

نکات ویزیتورها و توابع

slotOf

در این تابع برای متغیرها باید slot آنها را برگردانید. توجه کنید که slot صفر به صورت پیشفرض برای خود کلاس اصلی برنامه است و بعد از آن باید به ترتیب آرگومانهای تابع باشند (در واقع باید slot ها از 1 شروع شوند). طوری این تابع را پیاده سازی کنید که اگر ورودی یک string خالی بود، یک slot بعد از تمام slot های مخصوص آرگومان ها برگرداند. این برای استفاده از یک متغیر temp در code generation استفاده میشود؛ یعنی یک متغیر که برای تبدیل Jepeto به جاوا اضافه شده است.

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

header های مربوط به کلاس اصلی (Main) و همچنین static main method اضافه شوند.
سپس همه ی تابع ها ویزیت شوند. دقت کنید که current function را در Code Generator و Expression Type Checker ست شود.

FunctionDeclaration

header های مربوط به تابع اضافه شوند و سپس بدنه آن تابع ویزیت شود.

MainDeclaration

header های مربوط به یک کانستراکتور اضافه شوند و instance آن کلاس (کلاس اصلی Main) روی استک گذاشته شود و سپس کانستراکتور پدر که همان object جاوا میباشد صدا زده شود
(invokespecial java/lang/Object/<init>()V) و سپس بدنه main ویزیت شود.

BlockStmt

تمام statement ها را ویزیت کنید.

ConditionalStmt

دستورات و label های مورد نیاز برای یک شرط را اضافه کنید.

.

FunctionCallStmt

می توانید functionCall داخل آن را ویزیت کنید و خروجی آن را pop کنید. توجه داشته باشید که قبل و بعد از ویزیت، expressionTypeChecker.setFunctioncallStmt 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 صفحه ی نمایش باید از print در کتابخانه ی PrintStream.io.java استفاده کنید.

چاپ لیست ها **امتیازی** است . مثال چگونگی چاپ یک لیست:

[member1, member2,...,member n]
[1,2,3,4,5,6]

ReturnStmt

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

BinaryExpression

ابتدا عملوند سمت چپ و سپس سمت راست ویزیت می شوند و سپس عملوند مورد نظر اعمال میشود.
توجه کنید که اگر عملیات از نوع IntType یا BoolType است، ابتدا باید از غیر primitive به primitive تبدیل شود.

برای عملگر append باید از تابع addElement در کلاس لیست استفاده کنید .

UnaryExpression

ابتدا عملوند ویزیت شود؛ سپس برای هر یک از عملگرها دستورات مناسب باید اضافه شوند.

AnonymousFunction (امتیازی)

مانند یک functionDeclaration با آن برخورد می شود . اسم هر AnonymousFunction در نود مربوطه آن آمده است و باید فانکشن را با این نام ساخت. همچنین در انتها باید یه instance از کلاس fptr که مربوط به آن است را روی استک گذاشت.

Identifier

از slot متناسب با آن identifier باید مقدار load شود (با aload) سپس اگر نوع آن intType یا boolType است تبدیل به primitive شود.
اگر identifier ای که داریم آن را ویزیت می کنیم اسم یکی از توابع بود , باید fptr مخصوص به آن ساخته و روی استک گذاشته شود.

ListAccessByIndex

با استفاده از دستور getElement کلاس لیست آن اندیس مورد نظر گرفته شده و سپس به تایپ مناسب cast میشود و سپس به primitive تبدیل میشود.

ListSize

با استفاده از دستور getSize کلاس لیست سایز آن لیست گرفته شود .

FunctionCall

یک ArrayList ابتدا new شده و مقادیر آرگومان ها 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 int بودن تبدیل به غیر primitive می شود. توجه داشته باشید آرگومان ها بعد از visit شدن و قبل از اضافه شدن به ArrayList ، اگر int یا bool هستند باید به غیر primitive تبدیل شوند.

ListValue

در این قسمت باید یک ArrayList جدید ساخته شده و آرگومان ها 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 .)
توجه کنید که در این فاز فقط لیستی از عدد ها (int) داریم .

IntValue

با ldc باید مقدار آن روی stack گذاشته شود.

BoolValue

با ldc باید مقدار آن (0 یا 1) روی stack گذاشته شود.

StringValue

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

VoidValue

کاری نباید صورت بگیرد.

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 فایل class.

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

- اجرای فایل .class ها (در کل باید Main.class اجرا شود).

```
java Main
```

- تبدیل فایل بایت کد (j.) jasminبه فایل class.

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

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

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

- تبدیل فایل class. به بایت کد jasmin که خروجی در ترمینال نمایش داده می شود.

```
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 Jepeto که میخواهید معادل jasmin آن را پیدا کنید. به این صورت عمل کنید که ابتدا معادل java آن کد Jepeto 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 نمی دهد و باید بررسی شود (در اکثر موارد خروجی درست میدهد مگر چند مورد خاص.)

نکات مهم:

- در این فاز شما باید ویزیتور Code Generator را تکمیل کنید. کلیه فایل ها را به صورت یک فایل P4_<studentID1>_<studentID2>.zip آپلود کنید. توجه شود که تنها یک نفر از هر گروه باید پروژه را آپلود کند.
- در صورت کشف هرگونه تقلب، نمره 100- لحاظ می‌شود.
- دقت کنید که خروجی‌های شما به صورت خودکار تست می‌شوند.
- 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:

آرش رسولی arash3908@gmail.com

نازنین یوسفیان nazaninyousefian@ut.ac.ir