به نام خدا

فاز چهارم پروژه کامپایلرها و زبان‌های برنامه‌نویسی

پاییز 1400

مهلت تحویل : ۲۱ 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 --C را به کد قابل اجرا توسط ماشین تبدیل می‌کند. پیاده‌سازی شما باید به ازای هر فایل ورودی به زبان --C ، بایت کد معادل آن را تولید کند. در تست‌های این فاز، صرفا قابلیت تولید کد کامپایلرتان 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 نمی‌رود که فایل باینری را مستقیما تولید کنید. برای این کار می‌توانید از اسمبلر jasmin که در کلاس درس معرفی شده است استفاده کنید.

تساوی اشیاء

برای تایپ‌های int و boolean آن ها 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 پیاده سازی کنید.

نکات کلی پیاده سازی

- برای پیاده_سازی لیست 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 می‌برند و بنابراین می توان آن‌ها 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 non-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 commandها را که به صورت string هستند و با \n جدا شده‌اند 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 ساختمان‌ها و توابع، نوع تمام تایپ‌های primitive مانند int یا bool را از نوع‌های non-primitive جاوا تولید کنید. یعنی در بایت‌کد تولید شده باید این متغیرها از نوع‌های Integer یا Boolean 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.

- 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 expressionها از نوع primitive این دو تایپ استفاده کنید و در هنگام نوشتن آن‌ها در یک متغیر یا پاس دادن به توابع یا return شدن آن‌ها، این دو تایپ را از primitive به non-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 non-primitive تعریف شده‌اند و در expressionها ما نیاز به primitive داریم (توابع jasmin برای تبدیل آن‌ها در ادامه آمده است)

- فایل‌های Fptr.j و List.j در اختیارتان قرار گرفته‌اند 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 آن ها را مشاهده کنید که چه کاری انجام می‌دهند. Fptr در هنگام دسترسی به یکی از تابع‌ها 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 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 valueهای لیست هنگام پاس داده شدن به تابع یا assign شدن در یک لیست جدید کپی می-شود. 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 گرفته‌اید cast کنید (دستور cast در jasmin در ادامه آورده شده است) .

- در ابتدای هر کلاس که برای یک struct ایجاد می‌شود باید یک default constructor اضافه شود و باید در آن فیلد های آن ساختمان initialize شوند. برای int مقدار صفر ، برای bool مقدار false ، برای fptr مقدار null ، برای struct یک شی جدید از آن ساختمان و برای لیست یک شی از کلاس لیست که یک array list خالی از object ها دارد. (اگر default value داشتند باید آن مقدار لحاظ شود)

- 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

StructType → “struct_name”

- 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 مقداری را باز می‌گرداند؛ 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.

- برای مشاهده دستورات بایت کد میتوانید به این لینک مراجعه کنید.

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 slotهای مخصوص آرگومان‌ها برگرداند. این برای استفاده از یک متغیر temp در code generation استفاده می‌شود؛ یعنی یک متغیر که برای تبدیل --C به جاوا اضافه شده است.

Program

به ازای هر ساختمان یک فایل j. بسازید و ساختمان را ویزیت کرده و دستور های مربوط را در آن بنویسید و برای توابع و بخش main یک فایل Main.j بسازید و static main method را به آن اضافه و سپس آن ها را ویزیت کنید.

StructDeclaraion

فایل کلاس متناظر را با createFile بسازید و سپس header مربوط به کلاس را اضافه کنید. Parent آن را java/lang/Object قرار دهید. سپس متغیرها را ویزیت کنید.(متغیر ها را به عنوان یک فیلد به این کلاس اضافه کنید.) یک default constructor اضافه کنید. برای سادگی فرض کنید متغیرهایی که برای آن‌ها getter و setter تعریف شده‌اند نداریم.

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.

VariableDeclaration

اگر که در struct هستیم آن متغیر به عنوان یک فیلد به کلاس struct اضافه شود در غیر این صورت دستورات مربوط به initialize کردن متغیر با توجه به تایپ آن اضافه شود.

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 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.setInFunctionCallStmt 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.

DisplayStmt

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.

** این تابع برای شما پیاده سازی شده است.

ReturnStmt

دستورات مربوط به return را اگر void نیست اضافه کنید. توجه کنید که اگر expression جلوی return از نوع IntType یا BoolType است، ابتدا باید از primitive به non-primitive تبدیل شود.

LoopStmt

با توجه به نوع حلقه (یک فیلد isDoWhile در کلاس loopStmt تعریف شده است) دستورات و label های مورد نیاز برای آن را اضافه کنید.

VarDecStmt

تمام variable declaration ها را ویزیت کنید.

ListAppendStmt

listAppend را ویزیت کنید. توجه داشته باشید که قبل و بعد از ویزیت، expressionTypeChecker.setInFunctionCallStmt 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.

ListSizeStmt

listSize را ویزیت کنید. دقت کنید که مقداری که روی استک می‌گذارد را pop کنید .

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 LisAccessByIndex یا StructAccess است، دستورات assign را اضافه کنید. توجه داشته باشد که assign باید مقدار حاصل از دستورات operand سمت راست را که داخل operand سمت چپ ذخیره می‌شود در نهایت در stack قرار دهد. همچنین توجه کنید که اگر عملیات از نوع IntType یا BoolType است، ابتدا باید از non-primitive به primitive تبدیل شود.

UnaryExpression

نیازی به پیاده سازی این بخش نیست.

StructAccess

فیلد مورد نظر از کلاسی که برای struct ساخته شده است لود شود و اگر نوع آن int و یا bool بود تبدیل به primitive شود. (نیازی به ویزیت کردن element آن نیست و صرفا از اسم آن برای لود کردن استفاده کنید)

Identifier

اگر identifier اسم یکی از توابع بود آن فیلد باید fptr مخصوص به آن ساخته و روی استک گذاشته شود. در غیر این صورت از slot متناسب با آن identifier باید مقدار load شود (با aload) سپس اگرلازم بود تبدیل به primitive شود.

ListAccessByIndex

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

FunctionCall

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 می شود. در نهایت خروجی آن به تایپ مناسب cast شده و در صورت boolean یا int بودن تبدیل به non-primitive می شود. توجه داشته باشید آرگومان‌ها بعد از visit شدن و قبل از اضافه شدن به ArrayList ، اگر int یا bool هستند باید به non-primitive تبدیل شوند.

ListSize

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

ListAppend

با استفاده از دستور addElement کلاس لیست عضو جدید به آن لیست اضافه شود .

IntValue

با ldc باید مقدار آن روی stack گذاشته شود.

BoolValue

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

ExprInPar

فقط کافی است که اولین عضو آن ویزیت و دستوراتش اضافه 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 فایل class.

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

- اجرای فایل class. ها (در کل 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 C-- که میخواهید معادل jasmin آن را پیدا کنید. به این صورت عمل کنید که ابتدا معادل java آن کد C-- 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.)

نکات مهم:

- در این فاز شما باید ویزیتور Code Generator را تکمیل کنید. تنها فایل CodeGenerator.java را به صورت یک فایل P4_<studentID1>_<studentID2>.zip آپلود کنید(از تغییر قسمت های دیگر پروژه خودداری فرماید).
- توجه شود که تنها یک نفر از هر گروه باید پروژه را آپلود کند.
- در صورت کشف هرگونه تقلب، نمره 0 لحاظ می‌شود.
- دقت کنید که خروجی‌های شما به صورت خودکار تست می‌شوند.
- بهتر است سوالات خود را در 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 nazaninyousefian79@gmail.com