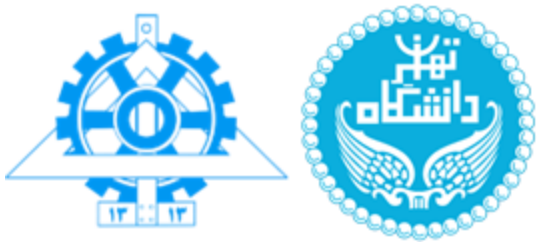

به نام خدا

تمرین کامپیوتری چهارم درس طراحی کامپایلر

بهار ۱۴۰۳

مهلت تحویل:۱۴۰۳/۳/۲۳

طراحان : محمد سوری، علی عطاءاللهی، محمدرضا نعمتی

فهرست مطالب

مقدمه

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 FunctionCraft را به کد قابل اجرا توسط ماشین تبدیل می‌کند. پیاده‌سازی شما باید به ازای هر فایل ورودی به زبان FunctionCraft، لیست bytecode های معادل آن را تولید کند. در تست‌های این فاز، صرفا قابلیت تولید کد کامپایلر شما سنجیده می‌شود و ورودی‌های دارای خطاهای نحوی و معنایی که در فازهای قبل بررسی کردید نیستند؛ اما توجه کنید که شما برای تولید کد به اطلاعات جمع‌آوری شده در جدول علائم و همچنین به اطلاعات مربوط به node-های درخت AST نیاز دارید.

حذفیات

قسمت‌هایی از زبان FunctionCraft برای این فاز حذف شده‌اند که نیازی به نوشتن visitor برای آنها نیستید. این موارد عبارتند از:

1. **Lambda function**
2. **elseif**
3. **two or multi-dimensional lists**
4. **empty lists**
5. **float type**
6. **pattern**
7. **append operator, push**
8. **chomp**
9. **break if, next if**
10. **for loop, range expression**

نکات کلی پیاده‌سازی

- 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 لیست command ها را که به صورت string هستند و با \n جدا شده‌اند return کنید. سپس 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.

- برای متغیر boolean، اگر مقدار آن true باشد 1 و اگر مقدار آن false باشد 0 را در stack اضافه کنید.

- برای اضافه کردن مقادیر primitive به stack، از دستور ldc استفاده کنید. برای string باید double quotation ("") آن را هم در دستور ldc بیاورید.

- طول stack و locals را در متدها ۱۲۸ قرار دهید.

- 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 آن‌ها 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.

- برای انجام محاسبات روی Integer، مقادیر باید از نوع primitive یعنی int باشند. پس در تمام expression ها از نوع primitive این type استفاده کنید و در هنگام نوشتن آن در یک متغیر یا pass دادن به توابع یا return شدن آن، این type را از primitive به non-primitive تبدیل کنید. همچنین بعد از خواندن مقادیر Int از متغیر یا لیست باید تبدیل انجام شود. دلیل تبدیل‌ها آن است که در تعریف، متغیرها از نوع non-primitive تعریف شده‌اند و در expression ها ما نیاز به primitive داریم.

- نام کلاس‌ها (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 به \n ها دقت داشته باشد تا در فایل jasmin ایجاد شده command ها پشت سر هم 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.

- برای پیاده‌سازی AccessExpression، برای function call 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 type مناسب cast شده و در صورت boolean یا int بودن تبدیل به non-primitive می‌شود. 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.

- برای پیاده‌سازی access by index، با استفاده از دستور getElement کلاس List، آن index مورد نظر گرفته شده و سپس به تایپ مناسب cast می‌شود و سپس به primitive تبدیل می‌شود.

  توجه کنید که در این فاز index فقط بر روی لیست استفاده می‌شود.

- تابع function pointer visitor داده شده است. این visitor ابتدا یک object جدید از کلاس Fptr ایجاد می‌کند. سپس با استفاده از متد init، fptr متناسب با آن را ایجاد میکند و بر روی stack قرار می‌دهد.

موارد پیاده‌سازی

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 به شما معرفی شده است استفاده کنید.

کلاس Main

کل برنامه در قالب یک کلاس به نام Main پیاده‌سازی می‌شود و توابع برنامه به عنوان توابع آن کلاس هستند. MainDeclaration هم constructor این کلاس می‌باشد.

slotOf

این تابع slot متغیرها را برمیگرداند. توجه کنید که slot صفر به صورت پیشفرض برای خود کلاس اصلی برنامه است و بعد از آن باید به ترتیب آرگومان‌های تابع اضافه شوند. برای دریافت یا ایجاد slot یک متغیر، کافیست تابع slotOf(varName) صدا زده شود که در صورت وجود نداشتن، آن متغیر به slots اضافه می‌شود و slot مربوط به آن داده شود.
همچنین در ابتدای هر تابع slots خالی می‌شود.

Program

ابتدا هدر های مربوط به کلاس Main اضافه شده است. بعد از آن static main method اضافه شده است. بعد از آن تمام توابع ویزیت می‌شوند. در آخر visit main می‌شود.

FunctionDeclaration

توابعی که در type checker ویزیت شده‌اند به constructor کلاس CodeGenerator داده می‌شود. چون برای ایجاد java bytecode توابع نیاز به type آرگومان‌ها و return آن داریم، فقط توابعی که type check شده‌اند visit می‌شوند.

بررسی تساوی و عدم تساوی اشیاء

برای متغیرهای int، آنها را با استفاده از مقادیرشان با دستور if_icmpeq مقایسه می‌کنیم.
توجه کنید که با توجه به حذفیات و پیاده‌سازی type checker، تساوی یا عدم تساوی فقط برای int تعریف می‌شود.

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.

عملگرهای and و or

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

ifStatement

دستورات مورد نیاز برای شرط ifStatement را اضافه کنید. دقت کنید که در این بخش label های مناسب برای if و else ها را تولید کنید و در ادامه استفاده کنید.

putsStatement

توابع مورد نیاز print را اضافه کنید. با استفاده از typeChecker می‌توانید type ورودی را بگیرید و از signature مناسب برای print استفاده کنید. جهت نوشتن بر روی صفحه‌ی نمایش باید از متد println در کتابخانه‌ی `java.io.PrintStream` استفاده کنید که به صورت

`java/io/PrintStream/println(arg)` می‌باشد. همچنین فقط type های boolean، integer و string به عنوان ورودی print داده می‌شوند و list و fptr نیازی به پیاده‌سازی در این بخش ندارند.

returnStatement

دستورات مربوط به return توابع را اضافه کنید. توجه کنید که اگر expression بازگشتی IntType و یا BoolType بود، باید از primitive به non-primitive تبدیل شود. دستور return مربوط به هر type در زیر آمده است:

```
return int: ireturn
return bool, string, list: areturn
return empty: return
```

BinaryExpression

ابتدا عملوند سمت چپ و سپس عملوند سمت راست visit شوند و مقادیر آنها روی stack قرار بگیرند. سپس عملگر مورد نظر اعمال شود.
توجه کنید که چون float در این فاز حذف شده است و Binary Expression ها فقط با IntType کار می‌کنند، آن‌ها باید از non-primitive به primitive تبدیل شوند.

UnaryExpression

ابتدا عملوند آن visit شود و مقدار آن روی stack قرار بگیرد، سپس عملگر مورد نظر اعمال شود.
توجه کنید که چون float در این فاز حذف شده است و Unary Expression ها فقط با IntType و BoolType کار می‌کنند، آن‌ها باید از non-primitive به primitive تبدیل شوند.

Identifier

از slot متناسب با آن identifier باید مقدار load شود با `aload`، سپس اگر نوع آن IntType یا BoolType بود تبدیل به primitive شود. اگر identifier type ای که داریم آن را ویزیت می کنیم FptrType بود، باید fptr مخصوص به آن ساخته شود و روی stack گذاشته شود. برای نحوه ساختن fptr می‌توانید به function pointer visitor مراجعه کنید.

LoopDoStatement

ابتدا label های مورد نظر را اضافه کنید. سپس statement های آن را visit کنید و در صورت ویزیت کردن BreakStatement یا NextStatement، دستورات goto مناسب را اضافه کنید.

LenStatement

با استفاده از تابع `List/getSize()` سایز یک لیست را گرفته و بر روی stack قرار دهید. برای string از تابع `java/lang/String/length()` استفاده شود.

ChopStatement

ابتدا ورودی آن را visit کنید و مقدار آن را روی stack قرار دهید. سپس با استفاده از دستورات مورد نیاز، آخرین کاراکتر string را حذف و دوباره مقدار باقی‌مانده را بر روی stack قرار دهید. توجه کنید که مقدار قبلی را از pop stack کرده باشید.

ListValue

در این قسمت ابتدا باید یک ArrayList ساخته شود و عناصر لیست به ترتیب visit شده و تبدیل به non-primitive شوند. سپس با استفاده از این ArrayList یک List ایجاد شود.

برای ایجاد List میتوان از دستور زیر استفاده کرد:

```
invokespecial List/<init>(Ljava/util/ArrayList;)V
```

توجه کنید که عناصر لیست شامل int یا string یا boolean می‌باشند. برای پیاده‌سازی لیست در جاوا نیاز داریم که یک لیست از جنس Object که superclass 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 آن‌ها را در لیستی از Object ذخیره کرد، ولی int و boolean که type های primitive هستند را نمی‌توان در این لیست ذخیره کرد. بدین منظور type 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 non-primitive تبدیل کنیم تا بتوانیم آنها را در لیست‌ها ذخیره کنیم.

IntValue

با استفاده از دستور `ldc` مقدار آن روی stack قرار داده شود.

BoolValue

با استفاده از دستور `ldc` مقدار 0 یا 1 متناسب روی stack قرار داده شود.

StringValue

با استفاده از دستور `ldc` مقدار آن (به همراه quotation) روی stack قرار داده 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 فایل‌های class. (فایل Main.class باید اجرا شود):

```
java Main
```

- تبدیل فایل (j.) jasmin bytecode به فایل class.:

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

- تبدیل فایل class. به java bytecode که خروجی هم در ترمینال نمایش داده شود:

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

- تبدیل فایل class. به jasmin bytecode که خروجی هم در ترمینال نمایش داده شود:

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

برای تبدیل فایل class. به کد java می‌توانید فایل class. را به drag-and-drop intellij کنید.

با استفاده از این دستورات می‌توانید کدهای زبان FunctionCraft را به معادل jasmin آن تبدیل کنید. به این صورت که ابتدا معادل جاوای کد FunctionCraft را بنویسید. یعنی منطق کدی که با زبان FunctionCraft می‌خواهید بنویسید را با زبان جاوا بنویسید و هر دو دارای مسیرهای یکسان در کد نوشته شده‌شان باشند. سپس فایل جاوای بدست آمده را کامپایل کنید که "class." تولید شود. سپس این فایل را با classFileAnalyzer به بایت‌کد 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

- تمامی فایل‌ها و کدهای خود را در یک فایل فشرده به صورت studentID1_studentID2.zip آپلود نمایید.
- در صورت کشف هرگونه تقلب، نمره صفر لحاظ می‌شود.
- دقت کنید که خروجی‌ها به صورت خودکار تست می‌شوند؛ پس نحوه چاپ خروجی باید عیناً مطابق موارد ذکر شده در بالا باشد.
- بهتر است سوالات خود را در فروم درس یا در گروه اسکایپ مطرح نمایید تا دوستانتان نیز از آنها استفاده کنند؛ در غیر این صورت به مسئولان پروژه ایمیل بزنید.