نام و نام خانوادگی:
شماره دانشجویی:

پاسخ تمرین سری سوم — اسکوپ، Type Checking ، SDT — موعد تحویل: ۹۹/۹/۲۱

مسئله‌ی ۱. اسکوپ

در این سوال بصورت کلی روند اجرا را توضیح می‌دهیم و مرحله به مرحله پیش می‌رویم:
۱. چون اسکوپ $static$ می‌باشد، از جدول نماد به سبک اسپاگتی استفاده می‌کنیم. ابتدا هریک از اسکوپ‌ها را در زیر معرفی می‌کنیم و نحوه عملکرد آن‌ها را شرح می‌دهیم:

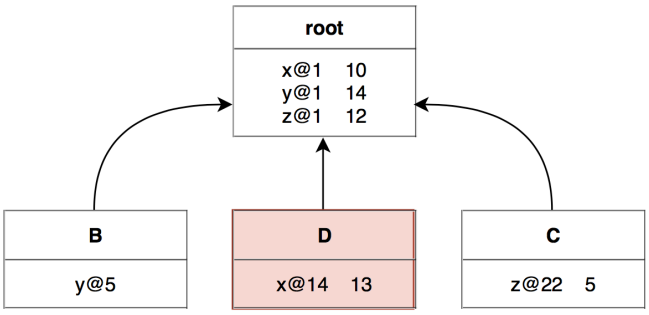
- در ابتدا $rootscope$ را داریم که شامل سه متغیر x, y, z می‌باشد.
- اسکوپ B که شامل یک متغیر به نام y می‌باشد و سپس آن را به صفر مقداردهی می‌کند. سپس متغیر x مربوط به $rootscope$ را برابر ۱ واحد بیشتر از مقدار متغیر z مربوط به $rootscope$ قرار می‌دهد. سپس متغیر z مربوط به $rootscope$ را برابر ۲ واحد بیشتر از مقدار متغیر y مربوط به اسکوپ خود قرار می‌دهد بعبارتی دیگر، مقدار متغیر گلوبال z را برابر ۲ قرار می‌دهد. و درنهایت از چپ به راست، مقدار متغیر x مربوط به اسکوپ ریشه، y اسکوپ خود و z اسکوپ ریشه را چاپ می‌کند.
- اسکوپ D که شامل یک متغیر به نام x می‌باشد و مقدار آن را ۱ واحد بیشتر از مقدار متغیر z مربوط به اسکوپ ریشه قرار می‌دهد. سپس مقدار متغیر y اسکوپ ریشه را برابر ۱ واحد بیشتر از مقدار متغیر x مربوط به اسکوپ خود قرار می‌دهد و درنهایت نیز تابع(یا همان اسکوپ) B را فرا می‌خواند.
- اسکوپ C که شامل یک متغیر z می‌باشد که مقدار آن را برابر ۵ قرار می‌دهد و سپس تابع D را فرا می‌خواند.
- اسکوپ $main$ که مقدار متغیرهای گلوبال را بصورت روبه‌رو مشخص می‌کند: $x = ۱۰, y = ۱۱, z = ۱۲$ و درنهایت تابع C را فرا می‌خواند.

حال با توجه به توضیحات بالا در مورد اسکوپ‌های استاتیک موجود خروجی برنامه ما بصورت زیر خواهدبود:

۲ ۰ ۱۳
۲ ۱۴ ۱۳

در هنگام اجرای خط ۱۷، مقدار متغیر z در اسکوپ C برابر ۵ است و مقدار متغیر x در اسکوپ D برابر ۱۳ است و مقدار متغیر x اسکوپ ریشه برابر ۱۰ ، متغیر y اسکوپ ریشه برابر ۱۴ و متغیر z اسکوپ ریشه برابر ۱۲ می‌باشد. و پس از اجرای اسکوپ B مقدار y اسکوپ B برابر ۰، متغیر x اسکوپ ریشه برابر ۱۳ و متغیر z اسکوپ ریشه برابر ۲ خواهد بود.

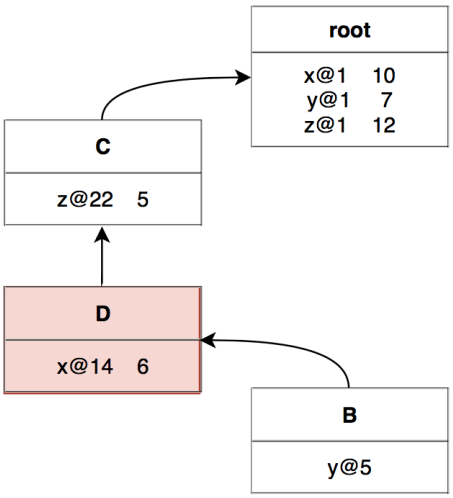
symbol table ها به شکل زیر است و هنگام اجرای خط ۱۷، در جدول مشخص شده هستیم.
هر متغیر با خطی که در آن تعریف شده مشخص و مقدار آن نیز هنگام اجرای خط ۱۷ روبروی آن نوشته شده است.



۲. در حالتی که اسکوپ ما حالت پویا داشته باشد مرحله به مرحله نیز ساخته خواهد شد و مانند حالت قبل در تمام مرحله شکل ساختاری یکسانی نخواهد داشت، لذا ما نیز به ترتیب اجرا آن را توضیح می‌دهیم:
ابتدا اسکوپ ریشه را داریم در بالا که شامل متغیرهای x, y, z سپس می‌باشد سپس در تابع $main$ مقادیر این ۳ متغیر اسکوپ ریشه به ۱۰، ۱۱ و ۱۲ تغییر می‌یابد، حال اسکوپ C اضافه می‌شود که در آن ابتدا یک متغیر z تعریف می‌شود با مقدار ۵، سپس اسکوپ D افزوده می‌شود که در آن ابتدا یک متغیر x تعریف می‌شود و مقدار آن برابر ۶ لحاظ می‌شود و سپس مقدار متغیر y اسکوپ ریشه به ۷ تغییر می‌یابد. بعد اسکوپ B افزوده می‌شود که در ابتدای آن متغیر y تعریف می‌شود و مقدار ۰ به خود می‌گیرد، بعد مقدار متغیر x مربوط به اسکوپ D برابر ۶ می‌شود و مقدار z اسکوپ C برابر ۲ می‌شود و بعد مقادیر x اسکوپ D، y اسکوپ B و z اسکوپ C چاپ می‌شوند سپس اسکوپ B حذف می‌شود و بعد اسکوپ D حذف می‌شود و بعد از آن نیز اسکوپ C حذف می‌شود و در نهایت مقادیر x, y, z مربوط به اسکوپ ریشه چاپ می‌شوند.
پس در نتیجه خروجی برنامه ما بصورت زیر خواهدبود:

۶ ۰ ۲
۱۰ ۷ ۱۲

هنگام اجرای خط ۱۷ در جدول مشخص شده هستیم. (جدول B وجود ندارد)
مشابه بالا هر متغیر با خطی که در آن تعریف شده مشخص و مقدار آن نیز هنگام اجرای خط ۱۷ روبروی آن نوشته شده است.



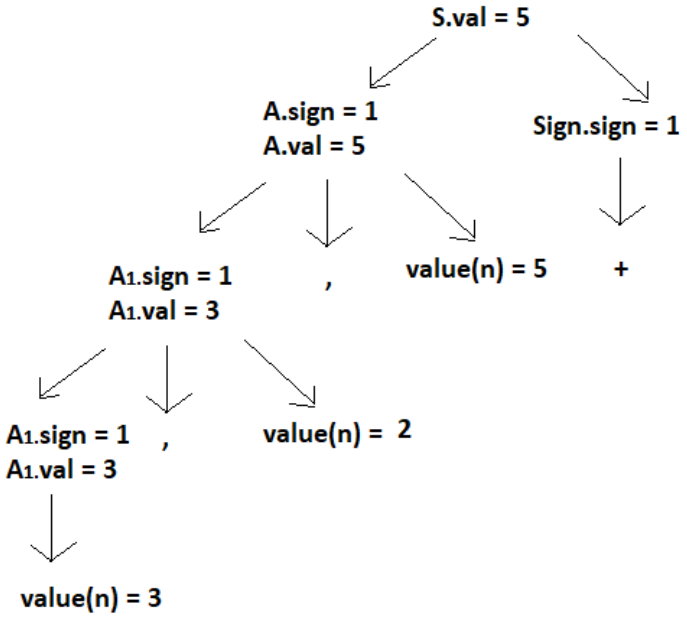
مسئله‌ی ۲. SDT

۱. این SDD یک دنباله از اعداد (چندین عدد و یک کاما(،) در میان آن‌ها) و یک علامت + یا ـ در پایان دنباله دریافت می‌کند. اگر در پایان دنباله + باشد، بیشترین عدد را از دنباله اعداد محاسبه می‌کند و اگر در پایان دنباله ـ باشد، کمترین عدد را از دنباله محاسبه می‌کند.

۲. به شرح زیر است:

- Sign.sign synthesized
- A.sign inherited
- Sign.val synthesized
- A.val synthesized

۳. به شرح زیر است:



۴. می‌توانیم غیرپایانه‌ی $Sign$را برداریم و دو حالت مثبت و منفی را جدا کنیم:

$$S \to B + \qquad S.val = B.val; print(B.val);$$

$$S \to C - \qquad S.val = C.val; print(C.val);$$

$$C \to C_1, n \qquad C.val = min(C_1.val, value(n))$$

$$B \to B_1, n \qquad B.val = max(B_1.val, value(n))$$

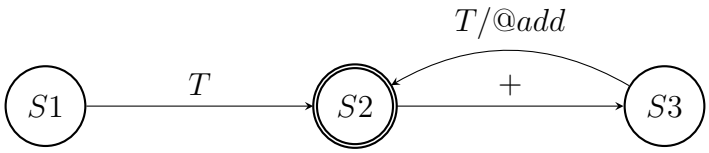
$$B \to n \qquad B.val = value(n);$$

$$C \to n \qquad C.val = value(n);$$

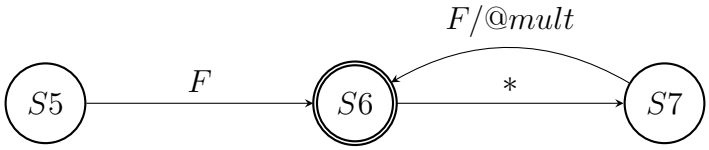
مسئله‌ی ۳. Syntax Graph

حل. ▷

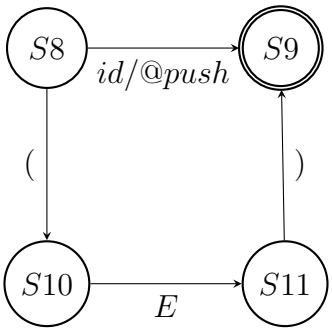
گراف‌ها به صورت زیر می‌باشند در ابتدا گراف E را داریم:



T:



F:



حال به توضیح دادن نحوه‌ی جلو رفتن و پر شدن استک می‌پردازیم رشته‌ی داده شده به $id + id$ تبدیل می‌شود حال ابتدا id اول خوانده می‌شود با توجه به اینکه در گره S1 می‌باشیم و هیچ یالی با برچسب id نداریم به داخل گراف T می‌رویم حال در گره S5 می‌باشیم و از آنجا نیز با توجه به اینکه توکن مورد نظر id می‌باشد و چنین یالی نداریم به داخل گراف F می‌رویم حال در گره S8 قرار داریم با توجه به اینکه توکنی که داریم id می‌باشد توسط یال آن به S9 می رویم و عملیات پوش را انجام می‌دهیم و a را به استک اضافه می‌کنیم. در این حالت استک به صورت زیر می‌باشد:

a

با دیدن a و پایان گراف F از این گراف خارج شده و به گراف T بر می‌گردیم و حال در گره S6 قرار داریم اکنون توکنی که در حال پردازش هستیم برابر با + می‌باشد از آنجایی که چنین یالی وجود ندارد و در حالت پایانی گراف می‌باشیم به گراف E باز می‌گردیم و در S2 قرار می‌گیریم. حال با دیدن + به گره S3 می‌رویم و از آنجا با دیدن توکن id دوباره مانند قبل عمل می‌کنیم وقتی دوباره به S8 رسیدیم و b را به استک اضافه کنیم استک به صورت زیر بدست می‌آید:

a
b

حال با توجه به اینکه رشته پایان یافته از گراف F خارج می‌شویم و از گراف T هم خارج می‌شویم حال در حالت S3 قرار داریم از اینجا نیز با توجه به اینکه گراف T را تمام کرده‌ایم با یال مربوط به آن به S2 می‌رویم و در این یال عملیات add@ نیز داریم حال دو عنصر آخر را از استک پاپ کرده و جمع می‌کنیم و در استک پوش می‌کنیم، استک به صورت زیر بدست می‌آید:

a+b

مسئله‌ی ۴.

۱. در کامپایلرهای single-pass ، کد تکه تکه خوانده می‌شود و برای هر تکه جداگانه عملیات کامپایل به تمامی انجام می‌شود (عملیات پارس، type checking، ایجاد کد و ...). اما در کامپایلرهای multi-pass کد چندین بار (برای هر مرحله کامپایل) خوانده می‌شود تا عملیات کامپایل به تمامی انجام بگیرد.
مزیت کامپایلر single-pass در سرعت فرآیند کامپایل می‌باشد. یک مشکل این کامپایلر این است که بخاطر کمبود اطلاعات در هر اسکوپی که در حال بررسی است، نمی‌تواند در برخی موارد کدهای کارآمدی تولید کند؛ مثلا در زبان java که توابع می‌توانند پیش از تعریف خود استفاده شوند، برای کامپایلر single-pass مشکل پیش می‌آید.
مزیت کامپایلر multi-pass این است که به دلیل اینکه در هر مرحله از کامپایل می‌تواند تمامی کد را ببیند، کدهای نهایی سریع‌تر و کارآمدتر ایجاد می‌کند. یکی از مشکلات این کامپایلر زمان‌بر بودن فرایند کامپایل می‌باشد.

۲. type checking در زمان کامپایل، خیلی قوی‌تر می‌تواند پیش از اجرای برنامه بسیاری از خطاهای type را تشخیص بدهد. حتی اگر که آن بلوک از کد در موارد نادری اجرا شود، همچنان با استفاده از این ویژگی می‌توان بسیاری از خطاهای type را پیدا کرد. مشکل این است که بررسی سلامت بسیاری از روش‌های برنامه نویسی نوین را نمی‌توان با این روش به خوبی انجام داد مانند downcasting و reflection .
runtime type checking به برنامه‌نویس اجازه می‌دهد بتواند روش‌های برنامه‌نویسی که type checking در زمان کامپایل نمی‌تواند به درستی آزمایش کند، بررسی کند. مشکل این روش این است که ممکن است برنامه در حین اجرا به مشکل برخورد کند و از کار بیفتد که خطرناک و مشکل‌زا خواهد بود.

مسئله‌ی ۵.

۱.

قاعده 1- صرفا یک فیچر زبان است.

قاعده ۰ درواقع توصیف بخشی از گراف وراثت است و در عمل یک type rule نیست.

-1

$$\frac{}{List = List < Object >}$$

۰

$$\frac{T \text{ is a class}}{ArrayList \leqslant List < T >}$$

۱

$$\frac{T\ is\ a\ class}{new\ T() : T}$$

۲

$$\frac{\begin{array}{c} e_۱ : T_۱ \\ e_۲ : T_۲ \\ T_۲ \leqslant T_۱ \end{array}}{WF(e_۱ = e_۲;)}$$

۳

$$\frac{expr : T}{WF(expr;)}$$

۴

$$\frac{\begin{array}{c} e. : List < T > \\ e_۱ : int \end{array}}{e..get(e_۱) : T}$$

۵

$$\frac{\begin{array}{c} e. : List < T_۱ > \\ e_۱ : T_۲ \\ T_۲ \leqslant T_۱ \end{array}}{e..add(e_۱) : bool}$$

۶

$$\frac{\begin{array}{c} T_۱\ is\ a\ type \\ e. : T_۲ \\ \cancel{T_۲ \leqslant T_۱} \end{array}}{(T_۱)e. : T_۱}$$

در خط ۷ از قاعده های ۰ و ۱ و ۲ استفاده شده است.

در خط های ۸ تا ۱۰ از قاعده های ۵ و ۳ استفاده شده است.

در خط های ۱۲ تا ۱۴ از قاعده های ۴ و ۶ و ۲ استفاده شده است.

.۲

به جای قاعده 1- از قاعده زیر استفاده می شود

-1

$$\frac{T\ is\ a\ class}{List < T >= List < T >}$$

با این کار در خط ۱۰ که می خواهیم از قاعده ۵ استفاده کنیم دچار مشکل می شویم زیرا $int \leqslant String$ را نمی توانیم نتیجه بگیریم و در نتیجه type این عبارت را نمی توانیم محاسبه کنیم و به خطای زمان کامپایل از نوع semantic برخورد می کنیم که ناشی از static type checking است.

در قسمت قبل کامپایل به درستی انجام می شد زیرا static type checker جاوا sound نیست و خط ۱۴ را قبول می کند(به قسمت خط خورده در قاعده ۶ دقت کنید). اما در زمان اجرا حین dynamic type checking متوجه می شویم که انجام خط ۱۴ امکان پذیر نیست و دچار cast exception می شویم.