همه‌ی ما

کامپایلر را دوست داریم

انگیزه و دورنمای درس اصول طراحی کامپایلر

کامپایلر را دوست داریم!

همانگونه که خوانندگان گرامی از این مستند انتظار دارند، هدف این مستند روشن کردن اهمیت کامپایلرها است. با وجود این آمادگی، احتمالا بیشتر آنها بعد از خواندن این متن از اهمیت کامپایلرها شگفت‌زده خواهند شد. مفاهیم و الگوریتم‌هایی که در کامپایلرها استفاده می‌شوند بسیار جالب هستند و به شکل‌های متفاوتی در زندگی حرفه‌ای هر یک از خوانندگان ظاهر می‌شوند و آشنایی با طراحی کامپایلرها می‌تواند برای آنها تعیین کننده باشد.

برای تماس با نویسنده‌ی این متن (علی غلامی رودی) با آدرس gholamirudi@nit.ac.ir مکاتبه نمایید.

مقدمه

قدرت پردازنده‌ها برای پردازش اطلاعات بر کسی پوشیده نیست. اما به دلایلی، زبان این قطعات سریع، بسیار خشک و ناخوانا است. تجربه‌ی نوشتن برنامه‌ای نه چندان بزرگ با استفاده از این زبان برای بیشتر برنامه‌نویسان (از جمله نویسنده‌ی این متن) بسیار ناخوشآیند است[۱]. همین مسئله باعث شد از حدود هفتاد سال پیش کامپایلرها ظهور کنند و به سرعت رشد نمایند. کامپایلرها زبان‌های برنامه‌نویسی سطح بالا را (که خوانایی بسیار خوبی دارند و برنامه‌نویسان را از بسیاری از جنبه‌های مدیریت منابع رها می‌سازند) به زبان سطح پایین پردازنده‌ها ترجمه می‌کنند. بدون کامپایلرها، برنامه‌نویسی، هم رده با دشوارترین حرفه‌های فیزیکی، در دسته‌ی سخت‌ترین شغل‌های بشر قرار می‌گرفت[۲]. نویسنده که رشته‌ی کنونی خود را مدیون کامپایلرها می‌داند، امیدوار است با این مقدمه، اهمیت کامپایلرها برای خوانندگان گرامی روشن شده باشد.

پیچیدگی‌های کامپایلرها

کامپایلرها بسیار جالب ولی پیچیده هستند. کامپایلرها نمایشگاهی از مباحث مختلفی از علوم کامپیوتر هستند: از روش‌های مختلف طراحی الگوریتم و نظریه‌ی گراف گرفته تا مدیریت منابع سیستمی و استفاده مناسب از ویژگی‌های سخت‌افزار. از طرف دیگر، کامپایلرها باید دقیق و کارا باشند. دقیق از این نظر که باید همیشه کد درست را تولید کنند؛ کامپایلری که کد نادرست تولید می‌کند (هر چند با احتمال بسیار کم) برای استفاده مناسب نیست. تصور مشکلاتی که ممکن است از ترجمه‌ی اشتباه برنامه‌ها ایجاد شود، در یک برنامه‌ی تجاری بسیار ترسناک است (برای نمونه، اشکالات ممکن در یک نرم‌افزار بانکداری را تصور کنید).

جالب است که در سال‌های اولیه‌ی ابداع کامپایلرها، آنها خیلی پرطرفدار نبودند؛ به دلیل محدودیت سرعت و حافظه‌ی سخت‌افزارهای آن زمان، کامپایلرها نرم‌افزارهای تجملاتی محسوب می‌شدند. از این رو، جای تعجب نیست که برای افزایش سرعت کامپایلرها بسیار تلاش شده است. اگر این تصور بین خوانندگان گرامی به وجود آمده است که به دلیل افزایش سرعت پردازنده‌ها، سرعت کامپایلرهای امروزی اهمیت چندانی ندارد، با کمال احترام باید بیان کنم که این تصور اشتباه است. در یک برنامه‌ی بزرگ (که تعداد آنها کم نیست) میلیون‌ها خط باید ترجمه شوند و سرعت ترجمه برای چنین برنامه‌هایی بسیار مهم

---

۱ برای این کار، به مستندهایی که مجموعه‌ی دستورات پردازنده و شیوه‌ی کدگذاری آنها را شرح می‌دهند نیاز خواهید داشت. علاوه بر آنها، مستندهایی که رابط برنامه‌نویسی هسته‌ی سیستم عامل را (ABI) توصیف می‌کنند، نیز باید مطالعه کنید.

۲ در آن شرایط، حتی علاقه‌ی زیاد به کامپیوتر نمی‌توانست نویسنده را از انتخاب حرفه‌ی شرافتمندانه‌ی دیگری که در آن احتمال نیاز به نوشتن کد ماشین وجود نداشت، باز دارد.

است. حتی در برنامه‌های کوچک نیز، انتظار چند دقیقه‌ای برای ترجمه به سختی قابل تحمل است.

مسئله‌ی مهم دیگر در کامپایلرها، سرعت برنامه‌ای است که ترجمه می‌شود. در گذشته برنامه‌هایی که توسط برنامه‌نویسان باتجربه نوشته می‌شدند، سرعت بسیار بیشتری نسبت به برنامه‌ی ترجمه شده توسط کامپایلر داشتند. از این رو، تلاش زیادی برای بهینه‌سازی کد تولید شده توسط کامپایلرها انجام شده است تا با حفظ درستی کد تولید شده و بدون تحمیل زمان پردازشی زیاد برای ترجمه، این اختلاف کاهش یابد. روش‌های بهینه‌سازی کد تولید شده توسط کامپایلرها بسیار جالب هستند و پژوهش‌گران زیادی برای بهبود روش‌های گذشته تلاش می‌کنند.

بنابراین، هدف کامپایلرها تولید کد سریع است و آنها باید این کار را با سرعت انجام دهند و چه لذتی دارد مطالعه‌ی طراحی کامپایلرهای خوب برای رسیدن به این اهداف و تحسین هنر آنها!

کاربرد و آینده‌ی کامپایلرها

شاید به نظر برسد با وجود کامپایلرهایی که در گذشته نوشته شده‌اند نیازی به نوشتن کامپایلر جدیدی در آینده نیست و افرادی که تخصص آنها طراحی کامپایلر است، باید خود را برای سوء تغذیه و فقر آماده کنند (یا مهارت دیگری را فرا بگیرند). این تصور قطعا اشتباه است. اول آنکه نیاز به طراحی کامپایلرهای جدید یا تغییر کامپایلرهای قدیمی به چند دلیل مثل گذشته باقی خواهد بود: الف) طراحی زبان‌های جدید، ب) اضافه کردن ویژگی‌های جدید به زبان‌های قدیمی (به تغییرات تدریجی زبان‌هایی که استفاده می‌کنید دقت کنید) و ج) بهره بردن از ویژگی‌های جدید سخت‌افزارهای جدید برای بهینه‌سازی کد تولید شده (مخصوصا برای توازی عملیات).

اما این تمام ماجرا نیست. علاوه بر کامپایلرهای خاص منظوره (که بسیار اهمیت و رواج دارند)، تکنیک‌های استفاده شده در کامپایلرها، در بسیاری از نرم‌افزارهای دیگر نیز کاربرد دارند: ابزارهای تولید مستند، موتورهای جستجو، محیط‌های مجتمع برنامه‌نویسی، ابزارهای مهندسی، نرم‌افزارهای کشیدن نمودار و نمونه‌های دیگر. در جلسه‌ی اول درس، برخی این کاربردهای جالب را مرور خواهیم کرد.

تجربه‌ی نویسنده

اهمیت کامپایلر را در گذشته بارها تجربه کرده‌ام. در ادامه، به این تجربیات اشاره می‌کنم.

در سال‌های ۱۳۸۴ تا ۱۳۸۶ کتابخانه‌ی روپ[۱] را برای بازسازی کد (Refactoring) در زبان پایتون نوشتم. یکی از مشکلات بازسازی کد در زبان پایتون این است که نوع متغیرها در زمان اجرا مشخص می‌شود. برای بازسازی کد لازم است تحلیل‌های متفاوتی مشابه تحلیل‌های کامپایلرها روی کد انجام شوند تا نوع متغیرها و تغییرات لازم برای بازسازی محاسبه گردند. در مستند دیگری تاریخچه‌ی این کتابخانه را بازگو خواهم کرد[۲]. پس از این تجربه، برنامه‌ی آیرن‌اوت[۳] را برای بازسازی کد در زبان C نوشتم[۴].

از سال ۱۳۸۸ به عنوان برنامه‌نویس سیستمی در شرکت نسبتا بزرگی همکاری می‌کردم. در این مدت، مشکلاتی در کامپایلر GCC تجربه کردم که بسیار آزاد دهنده بودند[۵]. بر حسب تصادف، برخی از اشکال‌های نسخه‌هایی از این کامپایلر که در هسته‌ی لینوکس خود را نشان دادند، در آن زمان موجب نارضایتی برنامه‌نویسان مشهور هسته‌ی لینوکس شدند[۶]. این مشکلات از یک سو و اعلام بازنویسی قسمت‌هایی از GCC با زبان C++ از سوی دیگر موجب شد به صورت جدی هدف نوشتن یک کامپایلر جدید C را در ذهن پرورش دهم؛ نام این کامپایلر را نیت‌سی‌سی[۷] تعیین کردم. در ابتدا این کامپایلر را برای معماری X86-64 نوشتم. پس از مدتی با هدف سیستم عاملی برای معماری ARM و پس از نوشتن یک اسمبلر برای آن[۸]، نیت‌سی‌سی را به این معماری انتقال دادم و پس از مدتی قابلیت تولید کد برای معماری X86 را نیز به آن اضافه کردم. در کنار این کامپایلر، یک لینکر[۹] و یک کتابخانه‌ی کوچک استاندارد زبان C[۱۰] نیز پیاده‌سازی کردم که نیت‌سی‌سی بتواند خود را بدون وابستگی دیگری ترجمه کند. پس از نوشتن این کامپایلر، بیشتر نرم‌افزارهایی را که روزانه استفاده می‌کنم با آن ترجمه می‌کنم.

در زمینه‌ی تولید کد برای سخت‌افزارهای خاص‌منظوره، برای یک سخت‌افزار موازی کامپایلری طراحی کردیم[۱۱] که توصیف‌های یک برنامه‌ی موازی را به دستورات این سخت‌افزار تبدیل نماید[۱۲].

ایده‌هایی که در کامپایلرها مطرح می‌شوند در زمینه‌های مهمی به غیر از تولید کد ماشین نیز کاربرد

---

۱ https://github.com/python-rope/

۲ بعد از بیش از یک دهه، این کتابخانه همچنان طرفداران نسبتا زیادی دارد و اکنون توسط افراد دیگری نگهداری می‌شود.

۳ http://repo.or.cz/ironout.git

۴ اما حقیقت تلخی است که معمولا برنامه‌نویسان زبان C از محیط‌های برنامه‌نویسی ساده‌ای استفاده می‌کنند (این مورد حتی در مورد نویسنده هم صدق می‌کند) و از این رو این برنامه چندان موفق نبود.

۵ کامپایلر GCC یکی از مشهورترین کامپایلرهای کدباز است که بیشتر بسته‌های لینوکس و BSD از آن برای ترجمه‌ی برنامه‌های زبان C و C++ (از جمله هسته‌ی سیستم عامل) استفاده می‌کنند.

۶ تا جایی که Ingo Molnar (یکی از بهترین برنامه‌نویسان هسته‌ی لینوکس) تهدید کرد که با این وضع شاید یک کامپایلر جدید را شروع کند.

۷ http://litcave.rudi.ir/neatcc.pdf

۸ http://repo.or.cz/neatas.git

۹ https://github.com/aligrudi/neatld.git

۱۰ https://github.com/aligrudi/neatlibc.git

۱۱ https://doi.org/10.1016/j.compeleceng.2015.07.015

۱۲ اخیرا نمونه‌ای از این سخت‌افزار ساخته شده است و به عنوان اختراع ثبت گشته است.

دارند. یک مثال از این دسته، نرم‌افزارهای تحلیل کد (مثل روپ) هستند که به آنها اشاره کردم و مثال دیگر برنامه‌های تولید مستند هستند که در ادامه به آنها می‌پردازم. پس از تلاش برای یافتن ابزار حروفچینی مناسب به ویژه برای زبان فارسی، برنامه‌ی نیتراف[۱] را نوشتم. در نیتراف ترجمه به چند شکل انجام می‌شود: ابتدا تعدادی پیش‌پردازشگر توصیف سطح‌بالایی را (مثلا برای شکل‌ها) به کد نیتراف ترجمه می‌کنند، سپس نیتراف کد تولید شده را به کد خروجی نیتراف (مشابه کد میانی در کامپایلرها) ترجمه می‌کند و بعد از آن پس‌پردازشگرها کد خروجی نیتراف را به زبانی برای تولید فایل نهایی ترجمه می‌نمایند.

به عنوان مثالی از پیش‌پردازشگرهای نیتراف، نیت‌ایکیوان[۲] عبارت‌های ریاضی را ترجمه می‌کند و به عنوان مثالی از پس‌پردازشگرهای نیتراف، نیت‌پست[۳] کد خروجی تیراف را به زبان PostScript تبدیل می‌نماید. تاریخچه‌ی نیتراف را در مستند دیگری شرح داده‌ام و علاقمندان را به مطالعه‌ی آن دعوت می‌نمایم[۴].

۱ https://github.com/aligrudi/neatroff.git
۲ http://litcave.rudi.ir/neateqn.pdf
۳ https://github.com/aligrudi/neatpost.git
۴ http://nit.rudi.ir/neatwond.pdf

اطلاعات بیشتر

توضیح	پیوند
صفحه‌ی جدید کتابخانه‌ی روپ	https://github.com/python-rope/
معرفی نیت‌سی‌سی	http://litcave.rudi.ir/neatcc.pdf
تاریخچه‌ی نیتراف (فارسی)	http://nit.rudi.ir/neatwond.pdf
عبارت‌های ریاضی در نیتراف	http://litcave.rudi.ir/neateqn.pdf
کد کامپایلر نیت‌سی‌سی	https://github.com/aligrudi/neatcc.git
کد لینکر نیت‌ال‌دی	https://github.com/aligrudi/neatld.git
کد کتابخانه‌ی استاندارد زبان C نیت‌لیب‌سی	https://github.com/aligrudi/neatlibc.git
معرفی نیتراف	http://litcave.rudi.ir/neatroff.pdf
کد نیتراف	https://github.com/aligrudi/neatroff.git
کد پس‌پردازشگر نیتپست	https://github.com/aligrudi/neatpost.git
کد پیش‌پردازشگر نیت‌ایکیوان	https://github.com/aligrudi/neateqn.git
کد برنامه‌ی آیرن‌اوت	http://repo.or.cz/ironout.git