درس استاد دهقان مباحث ویژه

تمرین(1)

1-طرح مسئله:

ساخت برنامه اندرویدی

2-فرضیه سازی:

آیا این برنامه رو به چه روشی میشه ساخت یا میشه با برنامه های آنلاین برنامه مون رو ساخت.

چطوری برنامه رو با کد آماده و پیدا کردن منابع معتبر برای کپی کردن کد ها ساخت.

3-آزمایش:

حالا با استفاده از مرحله آزمایش می آییم...ایده و فرضیه هایی ساختیم آزمایش میکنیم که آیا به چه نتیجه میرسیم و فایده ای دارد یا ن.

4-نوشن دادن داده ها:

خب برای ساخت این برنامه میتوان از ویژگی های استفاده کرد برای زیبایی یا نمایش بهتر به استفاده کننده یا میتوان شخصی سازی کنیم برای بهبود کردن برنامه.

5-آنالیز داده ها:

میاییم با استفاده از آنالیز نگاه میکنیم و آنالیز میکنیم داده های که برای برنامه به کار برده بدیم وتا نتیجه بدست بیاریم.

6-گزارش کارها:

خب میرسیم به مرحله آزمایش تمام کارهایی که واسه مسئله انجام دادیم تا بتونیم به بهترین نتیجه برسیم.

درس استاد دهقان مباحث ویژه

تمرین(2)

IDEA

1-ایجاد مسئله:

ساخت برنامه اندرویدی

2-راه حل:

برای ساخت برنامه راه حل های زیادی میشه استفاده کرد مثلا سایت هایی که آماده کد بهمون میدن یا برنامه های آماده

3-آزمایش:

با وجود این مراحل خب به یک آزمایش نیازی داریم که به نتیجه خوب رسیم که آیا این کار هایی که میکنیم مناسب هست برای کارمون یا خیر

4-ارائه مطلب:

و در آخر برای کارهایی که انجام دادیم نیاز به یه ارائه داریم و باید یه نتیجه کامل از کارمون رو ارائه بدیم.

درس استاد دهقان مباحث ویژه

تمرین(3)

منبع سایت(tosinso.com)

زمانیکه شما یک برنامه را با یک زبان برنامه نویسی سطح بالا می نویسید ، کامپیوتری که فقط صفر و یک را متوجه می شود درکی از برنامه ای که شما نوشته اید و کدهای درون آن نخواهد داشت. بنابراین شما به چیزی نیاز دارید که این برنامه سطح بالا را به زبانی تبدیل کند که برای کامپیوتر قابل فهم باشد. اینجا درست زمانی است که Compiler و Interpreter ها به کمک ما می آیند و هر دوی آنها یک کار را برای ما انجام می دهند آنها زبان سطح بالا را به زبانی که کامپیوتر متوجه شود تبدیل می کنند.

مهمترین تفاوتی که بین یک Compiler و یک Interpreter وجود دارد روشی است که آنها کد اجرایی برنامه را اجرا می کنند. زمانیکه شما توسط Interpreter یا مفسر کد را اجرا می کند ، کد در همان لحظه بصورت خط به خط اجرا می شود و بصورت خط به خط برای کامپیوتر کدها جهت اجرا ارسال و ترجمه می گردند در مقابل زمانیکه شما یک کد برنامه را توسط Compiler یا کامپایلر اجرا می کنید ، کامپایلر کد شما را یکباره اجرا نمی کند بلکه کد را بصورت کامل بر روی دیسک شما قرار می دهد و به شما این امکان داده می شود که هر زمانی که دوست داشتید کد اجرایی را اجرا کنید. در این مقاله از لفظ کامپایلر به جای Compiler و همچنین مفسر به جای Interpreter استفاده خواهیم کرد.

البته به غیر از بحث ترجمه یکباره و خط به خط کد برنامه ، یکی دیگر از مهمترین تفاوت هایی که بین کامپایلر و مفسر وجود دارد و مهمترین تفاوت این دو نوع مترجم نیز می باشد بحث وابستگی به برنامه است. برنامه یا کد نرم افزاری که توسط یک زبان برنامه نویسی مفسری نوشته شده است برای اینکه بتواند بر روی یک سیستم اجرا شود حتما نیاز به این دارد که مفسر مورد نظر از قبل روی سیستم نصب شده باشد و تا اینکار انجام نشود اجرا برنامه امکانپذیر نیست.

بنابراین نرم افزارهایی که به زبان های برنامه نویسی مفسری نوشته می شوند برای اجرا شدن حتما به مفسر مورد نظر نیاز دارند و در واقع وابستگی دارند. اما بر خلاف مفسر ها ، کامپایلر یکبار برای همیشه یک برنامه را به زبان اجرایی ماشین تبدیل می کند و در اصطلاح یکبار برنامه را به همراه کدهای اجرایی آن کامپایل می کند و بعد از آن دیگر نیازی به وجود داشتن نه کامپایلر و نه کد برنامه اجرایی می باشد ، خروجی یک کامپایلر یک یا چند فایل است که فارق از وجود کد اصلی برنامه و یا کامپایلر قادر به اجرا شدن بر

روی هر سیستمی را دارند و در واقع هیچ وابستگی به کامپایلر بعد از تبدیل کد وجود نخواهد داشت.

مقایسه کارایی و انعطاف پذیری

یا دردسر Overhead نقطه منفی استفاده کردن از زبان های مفسری این است که یک اضافه ایجاد می کند. با توجه به اینکه کدهای اجرایی برنامه در نرم افزارهای مفسری و CPU بصورت خط به خط اجرا می شوند اینکار باعث بالا رفتن میزان استفاده از منابع سیستم می شود ، اما زمانیکه یک برنامه کامپایل شد ، بصورت یکباره اجرا می RAM شود و نیازی به اجرا و پردازش هر خط برنامه بصورت جداگانه نخواهد بود.

با توجه به اینکه برنامه مفسری بسیار سنگین و خط به خط اجرا می شود سرعت آن به شدت کمتر از برنامه های کامپایل شده است. همچنین مفسرها قابلیت استفاده از امکانات سیستم عامل را که برای بهینه سازی کدها و اجرای سریعتر برنامه ها در کامپایلرها استفاده می شود را نیز ندارند. انتخاب کردن بین یک کامپایلر و مفسر بستگی به انتخاب شما دارد که کارایی بهتر را انتخاب می کنید یا قابل حمل بودن و انعطاف پذیری مد نظرتان است.

خطایابی کدام راحت تر است؟

همانطور که گفتیم کدهای اجرایی برنامه در برنامه های مفسری بصورت خط به خط یا ترجمه و اجرا می شوند ، این قابلیت به شما به عنوان یک برنامه نویس Line To Line اجازه می دهد که هر جایی از برنامه که به مشکل خوردید متوجه شوید که در کجا مشکل پیش آمده است و در جهت رفع مشکل اقدام کنید. شما می توانید در حین اجرا شدن خط به خط کدها ، آنها را تغییر دهید و بلافاصله تغییرات را مشاهده کنید ، اما مشکل تغییر کد نیز در این است که با هر بار تغییر دادن کد نرم افزار ، نرم افزار مجددا از ابتدا باید تفسیر یا شود Interpret.

اما زمانیکه شما از یک کامپایلر استفاده می کنید ، برای خطایابی ابتدا باید صبر کنید تا کد برنامه بصورت کامل کامپایل شود ، بعد از کامپایل شدن می توانید آن را مشاهده و خطایابی کنید و در صورت پیدا کردن خطا نمی توانید در همان لحظه خطا را تصحیح کنید بلکه باید اصلی برنامه را مجددا تصحیح و نرم افزار مجددا کامپایل کنید تا بتوانید Source Code مشکل به وجود آمده را حل کنید که این خود زمانگیر است.

درس استاد دهقان مباحث ویژه

تمرین(3)

منبع سایت(maktabkhooneh.org)

نقش کامپایلر چیست؟

نقش یک کامپایلر تسهیل تبدیل کدهای نوشته شده در یک زبان سطح بالا به زبان سطح ماشین است که باعث می‌شود به راحتی توسط کامپیوتر قابل درک باشد و این نقطه عطفی برای مسئله فرق بین کامپایلر و مفسر است. در این فرآیند، یک کامپایلر زبان سطح بالا (High Level Language)را به زبان اسمبلی متوسط تبدیل می‌کند. پس از آن، یک اسمبلر برای جمع‌آوری زبان اسمبلی میانی در کد ماشین (Machine language) استفاده می‌شود.

مزایا و معایب کامپایلر

در این بخش از مطلب تفاوت کامپایلر و مفسر به بررسی مزایا و معایب کامپایلر پرداخته‌ایم که از مهم‌ترین آن‌ها می‌توان به موارد زیر اشاره کرد:

مزایای کامپایلرها:

1. **افزایش سرعت اجرا :کد کامپایل شده در مقایسه با کد تفسیر شده سریع‌تر اجرا می‌شود، زیرا قبل از اجرا مستقیماً به کد ماشین ترجمه خواهد شد و اینجاست که تفاوت کامپایلر و مفسر خودش را نشان می‌دهد.**
2. **افزایش امنیت برنامه‌ها :کامپایلرها به بهبود امنیت برنامه‌ها کمک می‌کنند. فرآیند کامپایل شامل بهینه‌سازی کد و شناسایی آسیب‌پذیری‌های بالقوه است و در نتیجه خطر نقض امنیت را کاهش می‌دهد.**
3. **قابلیت‌های اشکال زدایی :کامپایلرها اغلب ابزارهای اشکال زدایی را ارائه می‌دهند که به شناسایی و اصلاح خطاهای کد کمک می‌کند. این ابزارها فرآیند اشکال‌زدایی را ساده کرده و پیدا کردن و رفع مشکلات را برای توسعه‌دهندگان آسان‌تر می‌کنند.**

معایب کامپایلرها:

1. **تشخیص خطای محدود :در حالی که کامپایلرها می‌توانند خطاهای نحوی و برخی از خطاهای معنایی را شناسایی کنند، ممکن است انواع خطاها یا مسائل منطقی**

را در کد تشخیص ندهند. برخی از خطاها ممکن است فقط در زمان اجرا آشکار شوند.

2. افزایش زمان کامپایل برای پایگاه‌های کد بزرگ :کامپایل کدهای حجیم می‌تواند زمان‌بر باشد. با افزایش اندازه پایگاه کد، فرآیند کامپایل ممکن است بیشتر طول بکشد و منجر به تأخیرهای احتمالی در چرخه توسعه شود.

توجه به این نکته مهم است که در حالی که این مزایا و معایب به طور کلی برای کامپایلرها صادق است، ویژگی‌ها و عملکرد خاص می‌تواند بسته به پیاده‌سازی کامپایلر و زبان برنامه‌نویسی مورد استفاده متفاوت باشد.

مفسر چیست؟

برای درک بهتر تفاوت کامپایلر و مفسر آشنایی با مورد دوم یعنی مفسر نیز خالی از لطف نیست. مفسر یا مترجم (interpreter) یک برنامه نرم‌افزاری است که یک زبان برنامه‌نویسی را به فرمی قابل درک و اجرا توسط کامپیوتر تبدیل می‌کند. بر خلاف کامپایلرها، مفسرها به طور مستقیم کد ماشین را تولید نمی‌کنند. در عوض، آن‌ها زبان سطح بالا را به یک زبان میانی یا بایت کد ترجمه می‌کنند. مفسرها معمولاً با کد از پیش کامپایل شده، کد منبع یا ترکیبی از هر دو کار خواهند کرد.

یکی از ویژگی‌های قابل توجه مفسرها این است که آن‌ها به جای ترجمه کل برنامه قبل از اجرا، یک دستور برنامه را در یک زمان پردازش و اجرا می‌کنند. این امکان یک فرآیند توسعه تعاملی و تکراری‌تر را فراهم می‌کند. از نظر اندازه، مفسرها معمولاً از نظر اندازه کوچک‌تر از کامپایلرها هستند. آن‌ها یک فایل اجرایی کامپایل شده جداگانه تولید نمی‌کنند، بلکه مستقیماً با کد منبع یا یک نمایش میانی از کد کار می‌کنند.

توجه به این نکته مهم است که در حالی که مفسرها مزایایی مانند چرخه‌های توسعه سریع‌تر و قابل‌حمل بودن را ارائه می‌دهند، ممکن است از نظر سرعت اجرا در مقایسه با کد کامپایل شده کندتر باشند. با این حال، مفسران مدرن اغلب از تکنیک‌هایی مانند کامپایل کردن در زمان (JIT) برای کاهش تفاوت‌های عملکرد و دستیابی به زمان‌های اجرایی سریع‌تر استفاده می‌کنند.

مزایای مفسر:

1. اشکال‌زدایی آسان‌تر :برنامه‌هایی که به زبان تفسیر شده نوشته شده‌اند، عموماً اشکال‌زدایی آسان‌تری دارند. از آنجایی که مفسرها کد را خط به خط اجرا می‌کنند،

بازخورد فوری ارائه کرده و به توسعه‌دهندگان اجازه می‌دهند تا خطاها را به طور مؤثرتری شناسایی و تصحیح کنند.

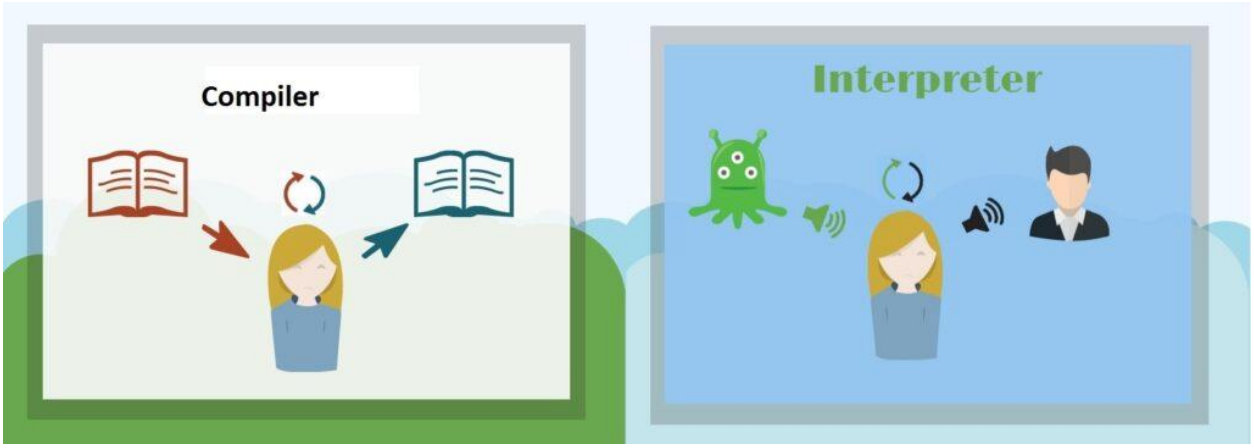
2. مدیریت خودکار حافظه :مفسرها اغلب مدیریت حافظه را به طور خودکار انجام داده و خطر خطاهای مربوط به حافظه مانند نشت حافظه یا سرریز شدن بافر را کاهش می‌دهند. این می‌تواند فرآیند توسعه را ساده کرده و ثبات کلی برنامه را بهبود بخشد.
3. انعطاف‌پذیری :زبان‌های تفسیر شده انعطاف‌پذیری بیشتری نسبت به زبان‌های کامپایل ارائه می‌دهند. مفسرها امکان تایپ پویا، تخصیص حافظه پویا و اصلاح کد زمان اجرا را فراهم می‌کنند و آن‌ها را برای نمونه‌سازی سریع، اسکریپت‌نویسی و محیط‌های توسعه تعاملی مناسب می‌سازد.

معایب مفسر:

1. اجرای برنامه محدود :مفسرها فقط می‌توانند برنامه‌هایی را اجرا کنند که به طور خاص برای مفسر مربوطه طراحی شده‌اند. این بدان معناست که کد نوشته شده برای یک مفسر ممکن است مستقیماً توسط مفسر یا کامپایلر دیگر بدون تغییرات قابل‌اجرا نباشد.
2. سرعت اجرای کمتر :کد تفسیر شده در مقایسه با کد کامپایل شده کندتر اجرا می‌شود زیرا در طول زمان اجرا خط به خط ترجمه و اجرا می‌شود. مفسرها از مزیت بهینه‌سازی پیش از زمان انجام شده توسط کامپایلرها برخوردار نیستند که می‌تواند بر عملکرد در وظایف محاسباتی فشرده تأثیر بگذارد.

بررسی تفاوت مفسر و کامپایلر

تفاوت بین کامپایلر و مفسر (مترجم) را همان‌طور که گفتیم می‌توان در ابعاد مختلفی بررسی کرد. در پایین به بررسی انواع تفاوت مفسر و کامپایلر پرداخته شده است.



کامپایلر:

- مراحل برنامه‌نویسی: ایجاد برنامه، تجزیه‌وتحلیل توسط کامپایلر، تبدیل به کد ماشین، پیوند فایل‌های کد و اجرای برنامه.
- کد ماشین تولید می‌کند که به عنوان یک فایل اجرایی ذخیره می‌شود.
- کد کامپایل شده سریع‌تر از کدهای تفسیر شده اجرا می‌شود.
- از مدل بارگذاری پیوند به عنوان مدل اولیه کار پیروی می‌کند.
- هرگونه تغییر در برنامه منبع مستلزم کامپایل مجدد کل کد است.
- خطاها پس از کامپایل کل کد نمایش داده می‌شوند.
- بهینه‌سازی کدهای اولیه را برای اجرای سریع‌تر انجام می‌دهد.
- برای اجرای بعدی به کد منبع نیاز ندارد.
- زمان بیشتری برای تجزیه‌وتحلیل کد منبع می‌گیرد.
- استفاده بالاتر ازCPU
- معمولاً در محیطهای تولید استفاده می‌شود.
- کد شی به طور دائم برای استفاده در آینده ذخیره می‌شود.
- نمونه‌هایی از زبان‌های برنامه‌نویسی مبتنی بر کامپایلر یا زبان‌های کامپایلرC :، C++، C#و غیره هستند.

مفسر:

- مراحل برنامه‌نویسی: ایجاد برنامه، اجرای دستورات و بررسی خطا.
- کد ماشین تولید نخواهد کر یا خروجی را ذخیره نمی‌کند.
- کدهای تفسیر شده نسبت به کدهای کامپایل شده کندتر اجرا می‌شوند.
- از مدل تفسیری به عنوان مدل کار اساسی پیروی می‌کند.
- تغییرات در برنامه منبع در حین ترجمه نیازی به ترجمه مجدد کل کد ندارد.
- خطاها برای هر خط کد نمایش داده می‌شود.
- تفسیر خط به خط را بدون بهینه‌سازی اولیه انجام می‌دهد.
- برای اجرای بعدی به کد منبع نیاز دارد.
- زمان کمتری برای تجزیه‌وتحلیل کد منبع می‌گیرد.
- استفاده کمتر ازCPU
- معمولاً در محیطهای برنامه‌نویسی و توسعه استفاده می‌شود.
- هیچ کد شی برای استفاده در آینده ذخیره نمی‌شود.

- نمونه‌هایی از زبان‌های برنامه‌نویسی مفسری: پایتون، روبی، پرل، SNOBOL، متلب و غیره هستند. همچنین می‌توان تفاوت زبان برنامه‌نویسی کامپایلر و مفسری را نیز به صورت مجزا بررسی کرد.

مفسر بهتر است یا کامپایلر؟

انتخاب بین کامپایلر و مفسر بستگی به شرایط و زمینه خاص دارد. یک مفسر برای کارهایی مانند اشکال زدایی و ارائه بازخورد فوری مفید است. از طرف دیگر، یک برنامه کامپایل شده پس از کامپایل شدن، سریع‌تر اجرا می‌شود. گزینه بهتر به نیازهای خاص کاربر بستگی دارد.

مفسر سریع‌تر است یا کامپایلر؟

به طور کلی، یک مفسر کندتر از یک برنامه کامپایل شده است و تفاوت کامپایلر و مفسر را می‌توان از جنبه سرعت هم بررسی کرد. با این حال، اگر یک برنامه قبلاً کامپایل شده باشد، اجرای آن معمولاً سریع‌تر از یک برنامه تفسیر شده است. مفسرها در ترجمه و اجرای کد خط به خط هزینه زیادی را متحمل می‌شوند، در حالی که کامپایلرها کد را از قبل برای اجرای کارآمد بهینه می‌کنند.

انواع کامپایلرها

کامپایلرها انواع مختلفی دارند که فهرست زیر مهم‌ترین آن‌ها را نشان می‌دهد:

- کامپایلر متقابل(Cross-Compiler)
- کامپایلر بومی(Native Compiler)
- کامپایلر بوت استرپ(Bootstrap Compiler)
- دیکامپایلر(Decompiler)
- کامپایلر منبع به منبع(Source-to-Source Compiler)
- بازنویسی زبان(Language Rewriter)
- کامپایلر بایت کد(Bytecode Compiler)
- کامپایلر به موقع(Just-in-time Compiler)

انواع مفسر

مفسرها هم انواع مختلفی دارند که فهرست زیر مهم‌ترین آن‌ها را نشان می‌دهد:

- **مفسر بایت کد(Bytecode Interpreter)**
- **مفسر کد رشته‌ای(Threaded code Interpreter)**
- **مفسر درخت نحو انتزاعی(Abstract syntax tree Interpreter)**
- **گردآوری به موقع(Just-in-time compilation)**