خروجی قطعه کد۱:

```
1st Hello
2nd Hello
3rd Hello
4th Hello
5th Hello
6th Hello
7th Hello
8th Hello
9th Hello
10th Hello
```

خروجی قطعه کد۲:

```
1st Hello
2nd Hello
3rd Hello
4th Hello
5th Hello
6th Hello
7th Hello
8th Hello
9th Hello
10th Hello
11th Hello
```

تفسیر خروجی برنامه از طریق ارسال آرگومان به برنامه در محیط intellij:

در این حالت خروجی برنامه همان String هایی خواهد بود که در قسمت Program Arguments به برنامه ارسال شده است. بدین معنی که اگر در این قسمت mystrings نوشته شود، در خروجی برنامه همان mystrings چاپ خواهد شد.

انجام دهید ۱)

اشکال زدایی قطعه کد۱:

بهتر است برای انتخاب نام method در قطعه کد۱ از under line استفاده نکرده و بجای نوشتن نام method بصورت my_method از نام myMethod استفاده کنیم.

اشکال زدایی قطعه کد۲:

بهتر است برای انتخاب نام method در قطعه کد۲ از روش نام گذاری Camel Case استفاده کنیم و بجای نوشتن نام method بصورت myAnotherMethod از نام myAnotherMethod استفاده کنیم.

انجام دهید۲)

الف) Error:(4, 43) java: ';' expected

ب) Error:(4, 33) java: ')' expected
Error:(4, 40) java: unclosed string literal
Error:(4, 39) java: not a statement
Error:(6, 2) java: reached end of file while parsing

ج) Error:(6, 2) java: reached end of file while parsing
Error:(1, 18) java: '{' expected
Error:(6, 1) java: class, interface, or enum expected

انجام دهید۳)

علت عدم اجرای کد فوق حذف آرگومان String [] args از مِتُد main می باشد.

انجام دهید۴)

برنامه جاوا در پایان مراحل توسعه همانند یک زبان کامپایلری کامپایل می شود (به Java bytecode) و در زمان اجرا همانند زبان های مفسری interpret می شود. پس باید گفت زبان جاوا هم مفسری است هم کامپایلر.

حال ببینیم زبان مفسری (Interpreter) و کامپایلری (Compiler) چیست؟

مفسر چیست؟

اگر بخواهیم واضح بگویم **مفسر در برنامه نویسی چیست**؟ مفسر یک برنامه کامپیوتری است که دستورهای نوشته شده در یک زبان برنامه نویسی را اجرا می‌کند .**مفسر در واقع یک زبان برنامه نویسی سطح بالا را به یک زبان قابل فهم برای ماشین تبدیل می کند .**

نکته مهم :در زبان های **مفسری** کد ها به صورت خط به خط تفسیر و اجرا می شوند و در اجرا بعدی برنامه نیز به همین صورت ادامه می یابد.

همینطور که می دانید هر دستگاه کامپیوتری (کامپیوترهای رومیزی، نوت بوک ها، گوشی های هوشمند، تبلت ها و ...) از سخت افزار خاصی ساخته شده است که دارای قسمت های مختلفی از جمله پردازنده (Processor) است. حال اگر بخواهید برای این پردازنده برنامه ای بنویسید که این پردازنده کاری را انجام دهد باید دستورات زبان ماشین خاص آن پردازنده را یکجا نوشته و در یک فایل ذخیره کنید و برای اجرا آن فایل را یکباره به پردازنده بدهید تا اجرا کند. چون **نوشتن به زبان ماشین** بسیار نیاز به حوصله و وقت دارد و **اشکال زدایی)** دیباگ) آن نیز کار پر مشقت و وقت گیر است معمولا از سایر زبان ها استفاده می کنند و آن زبان ها را با "برنامه های خاصی" به زبان ماشین تبدیل می کنند.

اگر آن برنامه خاص تبدیل دستورات به زبان ماشین را به جای یکباره و کامل به صورت دستور به دستور انجام دهد و همان لحظه دستور تبدیل شده (فقط و فقط همان دستور را) برای اجرا به پردازنده تحویل دهد و پس از اجرا دوباره دستور بعد را به زبان ماشین تبدیل کند و به پردازنده تحویل دهد و ... این ماجرا تا به انتهای دستورات به صورت خط

به خط تکرار شود و دستورات زبان ماشین توسط پردازنده فقط اجرا شوند و در فایلی ذخیره نشوند به آن برنامه خاص **اینترپرتر یا مفسر** می گویند.

در گذشته های دور نسخه های مختلف بیسیک جزو معروفترین زبان های اینترپرتری یا مفسری دنیا بودند نسخه هایی نظیر GWbasic و Basica و QBasic و...

بزرگترین مزیت اینترپرتر یا مفسر این است که برنامه نوشته شده اصلی **وابستگی کمتری به نوع سخت افزار** دارد. لذا برنامه ای که به زبان مفسر نوشته شود همانطور که می تواند روی یک کامپیوتر پردازنده اینتل اجرا شود می تواند بر روی یک تبلت با پردازنده آرم نیز اجرا شود یا روی یک موبایل یا یک اسباب بازی مدل.

تفاوت مفسر با کامپایلر

این دو در واقع یک کار را انجام میدهند ولی با هم یکسان نیستند و تفاوت هایی دارند. در مفسر کد ها خط به خط خوانده، برای کامپیوتر ارسال و اجرا می شود. اما کامپایلر تمام کد ها را یک باره ترجمه می کند و به صورت کامل در RAMنگه میدارد و شما میتوانید هر زمانی که خواستید کدهارا اجرا کنید.

تفاوت دیگر این دو، وابستگی به برنامه می باشد. برنامه یا کد های نوشته شده توسط یک **زبان مفسری** برای اجرا روی سیستم نیاز به نصب مفسر همان زبان روی سیستم را دارد. در غیر این صورت برنامه اجرا نمی شود. ولی کامپایلر برنامه را یک بار کامپایل می کند و بعد از آن دیگر نیازی به وجود همان کامپایلر روی سیستم نیست. خروجی یک برنامه نوشته شده با کامپایلر، فایل هایی هستند که روی هر سیستمی اجرا میشوند چه کامپایلر نصب باشد چه نباشد. اما یک مشکل کامپایلر این است که به سیستم وابسته است. یعنی اگر سیستم عامل دستگاه عوض شود امکان اجرای آن برنامه وجود ندارد اما مفسر ها چون خط به خط کد ها را اجرا و ترجمه می کنند امکان استفاده روی سیستم های مختلف را دارند .**برای رفع مشکل کدهای کامپایل شده باید آنها را روی سیستم جدید یکبار کامپایل کرد.**

اینکه کدام یک برتری دارند به نویسنده برنامه بستگی دارد .اگر **قابل حمل بودن** اهمیت داشته باشد، مفسری بهتر است. در غیر این صورت اگر سرعت و درگیر نکردن CPU و RAM اهمیت دارد، کامپایلری بهتر است.

چه زبان های برنامه نویسی، مفسری هستند؟

تعداد زیادی **زبان برنامه نویسی** داریم که مفسری می باشد. که در بخش زیر تعدادی از زبان های معروف که مفسر هستند رو لیست کرده ایم.

- پایتون
- جاوا اسکریپت
- بیسیک
- وی بی اسکریپت
- زبانPHP
- پرل
- روبی
- زبانForth
- و...

کامپایلر چیست؟

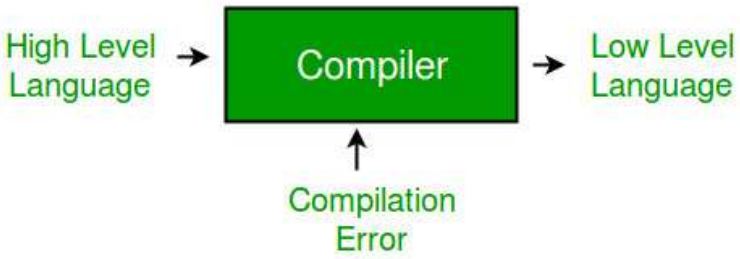
اگر بخواهیم **کامپایلر** رو تعریف کنیم و بگیم واقعا **کامپایلر چیست** باید بگویم مجموعه‌ای از برنامه یا برنامه‌های کامپیوتری است که متنی از زبان برنامه‌نویسی سطح بالا (زبان مبدا) را به زبانی سطح پایین (زبان مقصد)، مثل اسمبلی یا زبان سطح ماشین، تبدیل می‌کند.

به بیان ساده، **کامپایلر** برنامه‌ای است که یک برنامه نوشته شده در یک زبان خاص ساخت‌یافته را خوانده و آن را به یک برنامه مقصد **(Target Language)** تبدیل می‌نماید. در یکی از مهم‌ترین پروسه‌های این تبدیل، کامپایلر وجود خطا را در برنامه مبدأ اعلام می‌نماید.

تعریف یکی از دوستان از کامپایلر، کامپایلر برنامه‌ای رایانه‌ای می‌باشد که به‌منظور انتقال زبان‌های برنامه‌نویسی سطح بالا به زبان‌های سطح پایین مانند زبان اسمبلی و زبان ماشین، **جهت اجرایی شدن**، طراحی وارائه‌شده‌اند، خروجی این نرم‌افزار برای ماشین‌هایی مانند رایانه قابل اجرا هست.

کامپایلرها به عنوان ابتدایی ترین و اصلی‌ترین برنامه، برای برنامه نویسان به شمار می‌آیند، در اولین نگاه ممکن هست کامپایلر ها برنامه‌های ساده و بدون تنوع باشند اما با نگاه دقیق‌تر مشخص می‌شود که آنها در برخی موارد دارای پیچیدگی‌هایی هستند که به علت ویژگی‌های متفاوت آنها پدید آمده است. برخی از این پیچیدگی‌ها به علت دشوار بودن برخی زبان‌های سطح ماشین می‌باشد؛ به عبارتی زبان‌های سطح ماشین مانند زبان‌های برنامه‌نویسی سطح بالا به سادگی قابل‌فهم برای انسان نیستند و برای همین منظور است که انسان به زبان‌های سطح بالا برنامه را می‌نویسد و با استفاده از کامپایلرها آن را به سطح پایین و سطح ماشین تبدیل می‌کند .

برای درک بهتر نحوه کار کامپایلر، بخش زیر را ببینید.



نکته مهم :کامپایلرها معمولاً توسط شرکت‌های متفاوتی تولید می‌شود و همواره شرکت‌هایی که سخت افزار ماشین را تولید می‌کنند، **کامپایلر** مورد نیاز آن ماشین را نیز تولید و ارائه می‌کنند، البته کامپایلر ها دارای استانداردهای جهانی هستند که این امر مانع از آن می‌شود که هر شرکت خود به صورت دلخواه استانداردهایی مشخص کند. برای مثال استاندارد زبان اسمبلی یک استاندارد جهانی می‌باشد و شرکت‌های تولید کننده چیپ و میکروچیپ مانندIntel ، Motorolaو غیره از این زبان استفاده می‌کنند؛ به همین منظور کامپایلرهایی برای تبدیل به این زبان توسط این شرکت‌های ارائه می‌شود .

مهم‌تـــرین علت استفاده از کـــامپایلـــر

تـرجمه برنامه منبع به بـرنامه اجرائی

کامپایلرها دارای انواع متنوعی هستند که هر کدام به منظور استفاده برای کاربرهای خاصی تهیه شده است علی‌رغم این تنوع اعمال اساسی که هر کامپایلر بایستی انجام دهد، مشابه هم می‌باشند .**مهم‌ترین علت استفاده از کامپایلر ترجمه برنامه منبع به برنامه اجرائی می‌باشد** البته در شرایطی برخی کامپایلرها این کار را برعکس نیز انجام می‌دهند به طوری که زبان برنامه نویسی سطح پایین را به زبان برنامه نویسی سطح بالا ترجمه می‌کند.

بـــررسی انـــواع کامپایـــلر

انواع کامپایلر

در بخش زیر **سه مدل کامپایلر** را مورد بررسی قرار داده ایم که می توانید با خواندن این سه مورد با انواع کامپایلر آشنا شوید .

کامپایلرهای محلی و عبوری:

اکثر کامپایلرها به دو دسته **محلی و عبوری** تقسیم می‌شوند. کامپایلرهایی که به منظور اجرای برنامه های باینری هستند، کامپایلرهایی با کد محلی گوییم چرا که تنها در کامپیوترهای یک نوع با سیستم‌عامل های یکسان قابل به کارگیری است. از طرف دیگر ممکن است کامپایلرها کدهای باینری را تولید کنند که در سیستم‌های مختلف قابل اجرا باشد. به این دسته از کامپایلرها که وابستگی به سخت‌افزار ندارند، **کامپایلرهای عبوری** گوییم.

کامپایلرهای تک فاز و چند فاز:

کامپایلرها از نظر فاز به تک فاز و چند فاز تقسیم بندی می شوند. فاز بندی کامپایلرها در عمل به محدودیت‌های منابع سخت‌افزاری وابسته‌است. در نتیجه کامپایلرها به مجموعه برنامه‌های کوچکتر تقسیم می‌شوند هر یک بخشی از عمل ترجمه یا آنالیز را برعهده می‌گیرند.

کامپایلرهای تفسیری و کامپایلی:

زبانهای سطح بالا را به دو دسته تفسیری و کامپایلی تقسیم می‌کنند .**کامپایلرها و مفسرها** روی زبان‌ها عمل می‌کنند نه زبانها روی آنها! مثلاً این تصور وجود دارد که الزاماً BASIC تفسیر می‌شود و C کامپایل. اما ممکن است نمونه‌هایی از BASIC یا C ارائه شود که به ترتیب کامپایلری و تفسیری باشد. البته استثناهایی نیز وجود دارد.

بـــررسی نحوه کار کامپایـــلر

کامپایلر چگونه کار می کند؟

بعد از اینکه گفتیم کامپایلر چیست و معنی کامپایلر رو مورد بررسی قرار دادیم حال باید ببینم اصلا کامپایلر چطور کار می کند و به صورت دقیق به آن بپردازیم.

اگر بپذیرید که کامپیوتر تنها قادر به درک مفهوم سیگنال های پذیرش و عدم پذیرش و یا همان سیگنال ها و اعداد صفر و یک است می توانید راحت تر به جواب برسید **در واقع سیستم کامپیوتر شامل** مدارهایی است که این مدارها فقط به دو سیگنال صفر و یک و یا فعال و غیر فعال و یا روشن و خاموش حساس است و به هیچ وجه قادر به درک الفاظ و زبان طبیعی نمی باشد و حتی از کاری که قرار است انجام بدهد نیز خبر ندارد و مدارهای الکتریکی بر اساس کدهایی که در حافظه قرار می گیرد) *کلمات حافظه* (و در نهایت پردازش هایی که توسط پردازنده در واحد کنترل و ALUبر روی آن ها صورت می دهد اعمالی انجام می شود. اما آن چه که در این بخش مورد توجه است همان شکل گیری صفر و یک ها در نتیجه یک برنامه به زبان فرضا سی شارپ می باشد. این کاری است که کامپایلرها انجام می دهند.

مکانیسم کلی کار کامپایلرها به این صورت است که برنامه مبدا را خوانده و یک شکل میانی از آن ایجاد نموده و سرانجام آن را به زبان دیگری مانند اسمبلی تبدیل می کند و زبان اسمبلی نیز از شکل میانی برنامه شکل قابل فهم سیستم و یا همان صفر و یک ها را ایجاد و آن ها را در قالب Memory Word برای سیستم و سخت افزار مهیا می نماید. لذا تبدیل شکل ابتدایی برنامه مقصد به یک شکل اجرایی سیستمی از **وظایف کامپایلر ها** می باشد. البته باید توجه کنیم که کامپایلرها بر اساس قواعد و گرامر زبان مبدا اقدام به تولید زبان مقصد می نمایند.

بـررسی اجزای کامپایلـر

کامپایلر نویسان برای سهولت در طراحی، اجزای کامپایلر را به بخش های زیر تقسیم بندی می کنند که هر یک عملی را انجام می دهد:

الف) تحلیل گر لغوی :(Lexer) در واقع طولانی ترین پروسه را انجام می دهد، با زبان مبدا مستقیما در تعامل بوده و مستقل از زبان مقصد می باشد. تحلیل گر لغوی با خواندن زبان ورودی آن را **به مجموعه ای از نشانه های قابل فهم برای تجزیه کننده تقسیم بندی می کند** .میدانیم که جملات یک زبان از رشته هایی از نشانه ها تشکیل شده است و دنباله ای از این کاراکترهای ورودی که یک نشانه را تشکیل می دهند یک لغت (Lexeme) نامیده می شوند.

توضیحات، فضاهای سفید و فاصله ها توسط تحلیل گر لغوی نادیده گرفته می شود، سپس نشانه های تولیدی را در جدولی به نام جدول نمادها قرار می دهد و اشاره گری برای دسترسی پارسر به آن ها را بر می گرداند. تشخیص شناسه ها و کلمات کلیدی نیز از جمله مواردی است که Lexer باید آن ها را نیز تشخیص دهد و از سایر نشانه ها تمیز دهد. بنابراین به طور خلاصه **Lexer**کاراکترها را از ورودی می خواند، آن ها را به صورت لغت دسته بندی می کند و نشانه های ایجاد شده توسط لغت ها را به همراه مقادیر خصیصه آن ها به مرحله های بعدی کامپایلر منتقل می کند. نشانه های تولید شده در پارسر و یا تجزیه کننده مورد استفاده قرار می گیرد.

ب) تحلیل گر ساختار دستور :(Parser) پارسر (Parser) بررسی می کند که آیا می توان دنباله ای از نشانه های ایجاد شده را توسط گرامر زبان مورد نظر تولید نمود یا نه .**در واقع پارسر مهم ترین عمل را در طراحی کامپایلر انجام می دهد** و آن تولید دنباله از از رشته ها توسط گرامر زبان مبدا می باشد. به طور کل هنگام تحلیل ساختار دستور، نشانه هایی که در زبان مبدا قرار دارند را به عبارت های گرامری دسته بندی می کنیم به طوری که کامپایلر بتواند مجددا با استفاده از آن ها خروجی را ترکیب بندی کند. عبارت های تولید شده را توسط درخت تجزیه نمایش می دهند که درختی است که فرزندان هر گره عبارات سمت راست قوانین گرامر و پدر آن ها سمت چپ هر گره می باشد و گره های برگ مشتمل بر پایانی ها می باشند و یک دنباله از آن ها یک رشته از گرامر را نشان می دهند.

ج) تحلیل گر معنایی :(Syntax Analyzer) فاز تحلیل معنایی برنامه مبدا را برای پیدا کردن خطاهای معنایی بررسی کرده و اطلاعات مربوط به نوع داده ها را در درخت تجزیه حاشیه نویسی می کند مثلا بررسی می کند که یک رشته حرفی با یک عدد جمع نزده شده باشد و مانند آن. مجاز بودن نوع داده ها نیز در این بخش بررسی می شود. در واقع در زمان تحلیل معنایی، **کامپایلر ساختارهایی را کشف می کند** که از نظر ساختار دستوری صحیح هستند اما در رابطه با عملی که انجام می دهند بی معنی هستند.

د) تولید کننده کد میانی :د راین بخش یک شکل میانی قابل فهم اسمبلر و یا یک نمایش میانی صریح از برنامه مبدا تولید می کند که می توان آن را برنامه ای برای یک ماشین انتزاعی در نظر گرفت) **مفهوم انتزاعی به معنای قابل فهم بودن برای انسان است نه ماشین** (و باید دارای دو ویژگی سهولت تولید و سهولت تبدیل به برنامه مقصد باشد.

نمایش میانی می تواند شکل های گوناگونی داشته باشد مانند صفر، یک، دو، سه و چهار آدرسه. در ساختار فرضا سه آدرسه در هر ثبات می توان سه آدرس را شامل عملگر، عملوند اول و عملوند دوم قرار داد. محاسبه عبارات، نظارت بر ساختارهای جریان کنترلی و احضار رویه ها نیز در این بخش توسط مولد کد میانی صورت می گیرد.

هـ) بهینه ساز کد :(Optimizer)کدی که تولید می کنیم باید دو شرط صرفه جویی در حافظه و در زمان اجرا را برآورده کند. گاهی وقت ها کد ما بسیار پیچیده است و با اعمال جایگزینی ها و حذف و درج ها می توان آن را ساده

تر و کاراتر نمود. تغییر ساختارهای آدرس دهی نیز می تواند کد را بهینه تر سازد. سرعت اجرا نیز باید در نظر گرفته شود.

و) تولید کد :آخرین فاز کامپایلر، تولید کد مقصد می باشد که معمولا شامل کد ماشین جابه جا پذیر یا کد اسمبلی است. تخصیص حافظه به هر یک از متغیرهای برنامه در این فاز انجام می شود. آن گاه هر یک از دستورهای میانی به یک دنباله از دستورهای ماشین که همان کار را انجام میدهند ترجمه می شوند .*یک جنبه مهم آن، جایگزینی متغیرها در ثبات ها می باشد*.

پیروز و پرتلاش باقی بمانید

یاحق