فهرست

فصل اول: مقدمه‌ای بر کاپایلر

زبان های برنامه نویسی:

۱- **زبان ماشین:** زبان ماشین، زبانی است که دستور عمل ها بصورت کد های باینری است. این زبان تنها زبانی است که کامپیوتر آن را درک می کند. هر برنامه ای قبل از اجرا باید به زبان ماشین ترجمه بشود. هر نوع کامپیوتر یک زبان ماشین مخصوص به خود را دارد. برنامه نویسی به زبان ماشین مشکل و زمان بر است.

۲- **زبان اسمبلی:** در زبان اسمبلی، بجای کد های باینری از کلمات اختصاری استفاده می شود. خوانایی برنامه های به زبان اسمبلی بیشتر از برنامه های به زبان ماشین است. زبان اسمبلی سخت می باشد اما راحت تر از زبان ماشین است. برای تبدیل برنامه های نوشته شده به زبان اسمبلی به زبان ماشین از اسمبلر استفاده می شود. اسمبلر یک نرم‌افزار مترجم است که کلمات اختصاری را به کد های باینری تبدیل می‌کند.

برنامه های به زبان اسمبلی ← (اسمبلر) ← زبان ماشین

کلمات اختصاری ← (اسمبلر) ← کد های باینری

۳- **زبان سطح بالا:** زبان های سطح بالا به زبان محاوره‌ای نزدیک تر است. دستورات در زبان سطح بالا قدرتمند تر از بقیه ها زبان ها است. برنامه نوشته به این زبان، مستقیما بر روی ماشین اجرا نیستند. برنامه های نوشته شده به زبان سطح بالا برای اینکه ترجمه بشوند به زبان ماشین از کامپایلر استفاده می شود. کامپایلر نرم افزاری است که برنامه های به زبان سطح بالا را به زبان ماشین ترجمه می کند.

برنامه های به زبان سطح بالا ← (کامپایلر) ← زبان ماشین

تفاوت اسمبلر و کامپایلر:

اسمبلر هر دستور زبان اسمبلی را فقط به یک دستور زبان ماشین ترجمه میکند. اما کامپایلر هر دستور زبان سطح بالا را ممکن است به چندین دستور زبان ماشین ترجمه کند. بنابراین ارتباط دستور زبان اسمبلی به زبان ماشین در اسمبلر یک به یک می باشد و در کامپایلر یک به چند است.

اسمبلر: دستور زبان ماشین ← دستور زبان اسمبلی

دستور زبان ماشین ← دستور ۱

دستور زبان ماشین ← دستور ۲

کامپایلر: دستور زبان ماشین ← دستور زبان سطح بالا

دستور زبان ماشین

دستور زبان ماشین

روش های ترجمه و اجرای دستورات زبان های سطح بالا:

۱- **استفاده از مفسر:** در این روش دستورات زبان های سطح بالا یک به یک توسط مفسر خوانده میشود و بعد اجرا میشود. عملکرد مفسر را می توان به عملکرد یک شخص مترجم در مصاحبات و مذاکرات تشبیه کرد. در این روش برنامه مبدا به زبان ماشین ترجمه نمی شود.

مزایا استفاده از مفسر:

۱- سهولت اشکال زدایی: در این روش برای شروع اجرا برنامه نیازی نیست کل برنامه به زبان ماشین ترجمه بشود. مفسر اولین خط را خوانده و اجرا می کند، در نتیجه اجرای برنامه به سرعت شروع می شود. این ویژگی برای برنامه ی در حال ساخت مفید است. زیرا برنامه در حال تغییر است و با هر تغییر نیاز به اجرا مجدد برنامه است. بنابراین این ویژگی اشکال زدایی را تسریع می کند.

۲- قابلیت انعطاف بالا: در این روش مفسر در زمان اجرا برنامه حضور دارد و بعضی از اطلاعات، که فقط در زمان اجرا قابل دستیابی است توسط مفسر جمع آوری می شود و مفسر می تواند بر اساس آن تصمیم گیری کند.
۳- پیاده سازی آسان
۴- قابلیت حمل بالا: برنامه توسط مفسر اجرا می شود، در نتیجه هر جا که مفسر باشد برنامه مستقل از نوع سخت افزار اجرا خواهد شد.

معایب استفاده از مفسر:

۱- تکرار تفسیر: در هر اجرا برنامه مجداد تفسیر می شود.
۲- سرعت اجرا پایین: برنامه به طور مسقیم بر روی سخت افزار اجرا نمی شود؛ زیرا یک لایه به نام مفسر بین سخت افزار و برنامه قرار می گیرد در نتیجه سرعت اجرا برنامه کاهش می یابد.
۳- نیاز به مفسر: در روش برنامه بدون مفسر اجرا نمی شود. در نتیجه هر جا که لازم باشد برنامه اجرا شود، مفسر هم باید وجود داشته باشد. در غیر این صورت برنامه قابل اجرا نخواهد بود.
۴- دسترسی به کد مبدا: برای اجرا برنامه باید کد اصلی منبع موجود باشد تا قابل تفسیر و اجرا باشد. در نتیجه اگر برنامه ای به این روش تولید و توزیع گردد، کد منبع هم باید توزیع شود.

۲- **استفاده از کامپایلر:** در این روش برنامه به وسیله‌ی کامپایلر به زبان ماشین ترجمه می شود. کامپایلر نرم افزاری است که برنامه نوشته شده به زبان مبدا را به برنامه ی معادلی در زبان مقصد ترجمه می کند. اگر خطایی در برنامه وجود داشته باشد، کامپایلر اعلام خطا می کند و در صورت صحیح نبودن برنامه، کامپایلر قادر به ترجمه نخواهد بود. عملکرد استفاده از کامپایلر را تشبیه به عملکرد دوبله یک فیلم خارجی می کنند. (فیلم اول به طور کامل به زبان فارسی ترجمه می شود و بعد در اختیار بینندگان قرار می گیرد.)

مزایا استفاده از کامپایلر:

۱- سرعت اجرا بالا: برنامه اجرایی بدون واسطه روی کامپیوتر اجرا می شود در نتیجه سرعت اجرا بالا خواهد بود.
۲- اجرا مستقل برنامه از کامپایلر: بعد از کامپایل برنامه اجرایی تولید می شود. این برنامه اجرایی مستقل از کامپایلر روی ماشین اجرا می شود و نیازی به وجود کامپایلر در زمان اجرا نیست.
۳- حفاظت از کد منبع: در این روش برای توزیع برنامه بین کاربران، فقط برنامه‌ی اجرایی حاصل از کامپایل توزیع می گردد و نیازی به توزیع به توزیع کد منبع نیست.
۴- عدم تکرار کامپایل: بعد از تولید برنامه اجرایی، برنامه روی ماشین قابل اجرا خواهد بود و نیازی به تکرار کامپایل برای هر بار اجرا نخواهد بود.

معایب استفاده از کامپایل:

۱- زمان بر بودن اشکال زدایی: در روش کامپایل، ابتدا کل برنامه به زبان مقصد ترجمه می شود و سپس اجرای برنامه شروع می شود. بنابراین شروع اجرای برنامه قبل از پایان عمل کامپایل امکان پذیر نیست؛ در نتیجه در مواردی که برنامه در حال تولید است و تغییرات در برنامه زیاد است، زمان زیادی صرف اشکال زدایی و کامپایل خواهد شد.
۲- قابلیت حمل پایین: چون کامپایلر به سخت افزار وابسته است. زمانی که برنامه تولید می شود دارای زبان خاصی می باشد که فقط بر روی کامپیوتر خاصی قابل اجرا خواهد بود.
۳- پیاده سازی دشوار

Back-end و Front-end:

سوال: می خواهیم یک برنامه ای را به زبان c را بر روی کامپیوتر های مختلف اجرا کنیم؛ به چند کامپایلر نیاز خواهیم داشت؟

به تعداد کامپیوتر هایی که وجود دارد، کامپایلر نیاز داریم.

حالا اگر فرض کنیم n زبان، k تا کامپیوتر داشته باشیم. در این صورت به n*k کامپایلر نیاز خواهیم داشت.

به طور مثال اگر ۱۰ زبان و ۲۰ کامپیوتر داشته باشیم، نیاز به ۲۰۰ کامپایلر خواهیم داشت.

از آنجایی تولید این تعداد کامپایلر زمان بر و هزینه بر می باشد، کامپایلر را به دو بخش Back-end و Front-end تقسیم می کنند. (عقب بندی و جلوبندی)

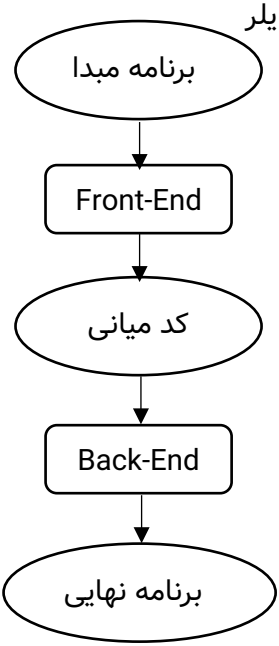
مثال: فرض کنید یک متن خبری از زبان روسی را می خواهیم ترجمه کنیم به همه زبان های زنده دنیا؛ اگر فرض کنیم n تا زبان در دنیا وجود داشته باشد به $n-1$ مترجم نیاز خواهیم داشت.

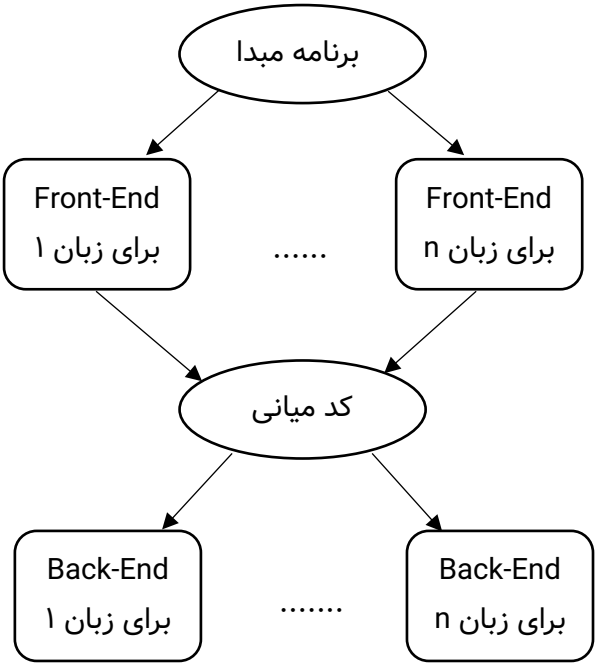
حال فرض کنیم آن متن خبری را از هر زبانی به هر زبان دیگری ترجمه کنیم به $\frac{n(n-1)}{2}$ نیاز خواهیم داشت.

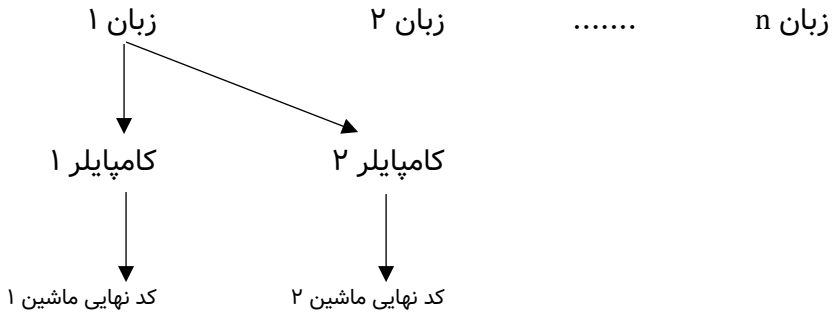
حالا اگر ما ۱۰۰ زبان (n) داشته باشیم نیاز به ۴۹۵۰ مترجم خواهیم داشت.

برای اینکه جلوگیری شود از این کار که زمان بر و هزینه بر است؛ از یک زبان میانی استفاده می شود که اصطلاحا به آن زبان بین المللی می گویند. که در این صورت اول متن را از زبان مبدا به زبان انگلیسی ترجمه می شود و بعد از زبان انگلیسی به زبان های مقصد ترجمه می شود. در این صورت به ۱۰۰ مترجم یا همان n مترجم نیاز خواهیم داشت.

در زبان های برنامه نویسی هم به همین صورت می باشد یعنی زبان مبدا را به زبان میانی ترجمه می شود و بعد از زبان میانی به زبان مقصد ترجمه می شود.


برنامه مبدا
Front-End
کد میانی
Back-End
برنامه نهایی


برنامه مبدا
Front-End
برای زبان ۱
.....
Front-End
برای زبان n
کد میانی
Back-End
برای زبان ۱
.....
Back-End
برای زبان n



کامپایلری نرم افزاری است که برنامه نوشته شده به زبان مبدا را به برنامه ی معادلی در زبان مقصد ترجمه میکند؛ در نتیجه با توجه به تعداد بالا زبان های مبدا و مقصد، کامپایلر های بسیاری می توان ایجاد کرد.

جهت جلوگیری از ایجاد تعداد زیادی کامپایلر که بسیاز زمان بر و پرهزینه است از تقسیم کامپایلر به Back-end و Front-end استفاده می کنیم. آن بخش از کامپایلر که وظیفه ترجمه برنامه مبدا به کد میانی را بر عهده دارد Front-End و آن بخش از کامپایلر که وظیفه ترجمه کد میانی به برنامه نهایی را برعهده دارد Back-End می گویند.

مزایا تقسیم کامپایلر به Back-end و Front-end:

۱- سادگی طراحی
۲- کاهش پیچیدگی
۳- استقلال Front-end از کد نهایی
۴- استقلال Back-end از زبان مبدا
۵- افزایش قابلیت استفاده مجدد
۶- افزایش سرعت تولید کامپایلر

ساختار کامپایلر کمی پیچیده است به همین دلیل کامپایلر را به بخش های مختلفی تقسیم کرده اند. که هر بخش از کامپایلر یک وظیفه مخصوص به خودش را دارد که در کنار هم قرار گرفتن این بخش ها کار کامپایلر را انجام می دهد.

هر کدام از بخش های کامپایلر، خروجی هر بخشش، ورودی بخش بعدی است که به این بخش های کامپایلر، فاز های کامپایلر می گویند.

فاز های کامپایلر:

۱- تحلیلگر لغوی
۲- تحلیلگر نحوی
۳- تحلیلگر معنایی
۴- مولد کد میانی
۵- بهینه سازی
۶- مولد کد پایانی

مزایا تقسیم کامپایلر به فاز ها:

۱- سادگی طراحی
۲- افزایش کارایی
۳- افزایش قابلیت حمل
۴- افزایش قابلیت انعطاف پذیری

تحلیلگر لغوی:

تحلیلگر لغوی تنها فازی است که مستقیما به برنامه مبدا دسترسی دارد. برنامه مبدا را به صورت جریانی از کارکتر ها از ورودی دریافت می کند و لغات تشکیل دهنده برنامه و نوع آن ها را تشخیص داده و برای تحلیلگر نحوی ارسال می کند. تحلیگر لغوی در صورت وجود خطا در قالب بندی لغات مورد استفاده در قالب بندی لغات مورد استفاده در متن برنامه ی مورد کامپایل اعلام خطای لغوی می کند.

مثال:

K := H + 12 B ;

شناسه — علامت انتساب — شناسه — علامت جمع — عدد — شناسه — علامت

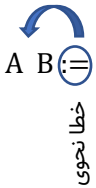
var 7temp; integer;

خطا لغوی

تحلیلگر نحوی:

وظیفه تحلیلگر نحوی تشخیص فرم ظاهری برنامه ها از نظر دستور عمل زبان برنامه نویسی مربوطه است. تحلیلگر نحوی با فراخوانی تحلیلگر لغوی، لغات را از متن برنامه ی مورد کامپایل دریافت کرده و صحت قرار گرفتن آن ها در مجاور یکدیگر را بر اساس دستور عمل زبان مربوطه مورد آزمون و تحلیل قرار می دهد؛ و اگر خطایی وجود داشته باشد، اعلام خطای نحوی می کند. اگر ترتیب لغات برنامه مبدا صحیح باشد از آن یک ساختار درختی تولید می کند که این درخت نشان دهنده ساختار برنامه مبدا است که اصطلاحا به آن درخت تجزیه می گویند.

مثال:



عدم رعایت ترتیب انتساب