به نام خدا

زبان‌های برنامه‌نویسی

آرش شفیعی

کتاب‌های مرجع

- مفاهیم زبان‌های برنامه‌نویسی، از روبرت سبستا [1]
- مفاهیم در زبان‌های برنامه‌نویسی، از جان میچل [2]
- زبان‌های برنامه‌نویسی : اصول و الگوواره‌ها، از ماریتسیو گابریلی [3]
- مستندات زبان‌های برنامه‌نویسی

[1] Concepts of Programming Languages, by Robert W. Sebesta
[2] Concepts in Programming Languages, by John C. Mitchell
[3] Programming Languages : Priciples and Paradigms, by Maurizio Gabbrielli

مقدمه

زبان‌های برنامه‌نویسی

- یک زبان برنامه‌نویسی، یک سیستم نشانه‌گذاری [1] است برای بیان یک برنامهٔ کامپیوتری [2]. یک برنامهٔ کامپیوتری مجموعه‌ای از دستورات است برای انجام محاسبات بر روی داده‌ها.
- یک زبان برنامه‌نویسی ایده‌آل، به برنامه‌نویسان اجازه می‌دهد یک برنامه را به طور مختصر و واضح بیان کنند.
- از آنجایی که یک برنامهٔ کامپیوتری باید در طول دوران حیات خود توسط برنامه‌نویسان مختلف خوانده شود، فهمیده شود، و تغییر داده شود، بنابراین یک زبان برنامه‌نویسی خوب زبانی است که برنامه‌نویسان را قادر می‌سازد، برنامه‌های یکدیگر را بهتر بفهمند.
- معمولا یک برنامهٔ بزرگ از تعداد زیادی اجزا تشکیل شده است که با یکدیگر در ارتباط هستند. یک زبان برنامه‌نویسی خوب، زبانی است که به برنامه‌نویسان کمک می‌کند برنامه‌های پیچیده و ارتباط بین اجزا را به سادگی و به طور بهینه توصیف کنند.

[1] Notation system
[2] Computer program

زبان‌های برنامه‌نویسی

- در طراحی یک زبان برنامه‌نویسی معمولا باید انتخاب‌های مختلف را سبک‌سنگین [1] کرد.
- به طور مثال برخی از ویژگی‌های زبان‌های برنامه‌نویسی به برنامه‌نویسان کمک می‌کنند برنامه‌ها را سریع بنویسند، اما از طرفی همین ویژگی‌ها باعث کاهش بهره‌وری [2] و پیچیده‌تر شدن و کندتر شدن کامپایلر می‌شوند.
- برخی از ویژگی‌های زبانی باعث می‌شوند کامپایلر به طور بهینه پیاده‌سازی شود، اما از طرف دیگر باعث می‌شوند برنامه برای برنامه‌نویسان پیچیده‌تر شود.
- از آنجایی که برنامه‌های متفاوت نیازهای متفاوتی دارند، بنابراین زبان‌های برنامه‌نویسی متفاوتی به وجود آمده‌اند تا این نیازها را برآورده سازند. هر زبان برنامه‌نویسی موفقی در ابتدا برای یک مجموعه از برنامه‌ها با نیازهای مشابه به وجود آمده است. البته این بدین معنی نیست که یک زبان فقط برای انجام یک کار به وجود آمده است، بلکه بدین معنی است که تمرکز یک زبان بر روی حل کردن مجموعهٔ مشخصی از مسائل است.

[1] Trade-off
[2] Performance

زبان‌های برنامه‌نویسی

- مطالعهٔ زبان‌های برنامه‌نویسی به ما کمک خواهد کرد مفاهیم اصلی در زبان‌های برنامه‌نویسی را بهتر بفهمیم و با نقاط قوت و ضعف آنها بهتر آشنا شویم و بتوانیم از زبان‌ها حداکثر بهره را ببریم.
- هر زبان برنامه‌نویسی روشی متفاوت برای حل یک مسئله ارائه می‌کند. در واقع یک زبان برنامه‌نویسی یک چهارچوب فکری برای حل مسئله و طراحی نرم‌افزار ارائه می‌کند. بنابراین مطالعهٔ یک زبان و روش‌های موجود در آن زبان جهت حل یک مسئله، می‌تواند به حل آن مسئله در زبان‌های دیگر نیز کمک کند.

زبان‌های برنامه‌نویسی

- در اینجا ابتدا توضیح می‌دهیم چرا به مطالعه زبان‌های برنامه نویسی احتیاج داریم.
- سپس به طور خلاصه حوزه هایی را توضیح می‌دهیم که در آنها به زبان‌های برنامه نویسی نیاز داریم.
- توضیح می‌دهیم بر اساس چه معیارهایی می‌توان زبان‌های برنامه نویسی را مورد ارزیابی قرار داد.
- در پایان تاریخچهٔ زبان‌های برنامه نویسی و روش‌های پیاده‌سازی آنها را به اختصار شرح می‌دهیم.

دلایل مطالعه زبان‌های برنامه نویسی

- **افزایش توانایی بیان ایده‌ها :** معمولا عمق تفکر افراد وابسته به توان بیان آنها در زبانی است که با استفاده از آن ارتباط برقرار می‌کنند. از آنجایی که زبان ابزاری است برای تفکر و اندیشیدن، هر چقدر یک زبان را بهتر بشناسیم، بهتر می‌توانیم با استفاده از آن ایده‌ها را منتقل و مسائل را حل کنیم. معمولا یک زبان ساختار معینی دارد و با استفاده از آن ساختار می‌توان ارتباط برقرار کرد ولی هر ساختاری محدودیت‌هایی نیز دارد. بنابراین یادگیری یک زبان متفاوت که ساختار متفاوتی دارد کمک می‌کند بتوانیم بهتر بیاندیشیم.

دلایل مطالعه زبان‌های برنامه نویسی

- برنامه نویسان نیز در طراحی نرم‌افزار محدود به زبانی هستند که با استفاده از آن برنامه می‌نویسند. معمولا در برنامه نویسی ساختارهای کنترلی و ساختارهای داده موجود بر روش تفکر ما در حل یک مسئله محدودیت اعمال می‌کنند. یادگیری زبان‌های متفاوت کمک می‌کند که ساختارهای متفاوتی در ذهن ما به وجود بیایند و بتوانیم برای حل مسئله از آن ساختار ها استفاده کنیم.
- برای مثال یک برنامه نویس سی که با پایتون آشنا باشد و از سهولت استفاده از لیست‌ها آگاه باشد، می‌تواند لیست‌هایی مشابه با پایتون در برنامه سی ایجاد کند و از آنها برای حل مسئله خود استفاده کند.

دلایل مطالعه زبان‌های برنامه نویسی

- **انتخاب مناسب یک زبان برای یک کاربرد خاص**: معمولا یک زبان برای کاربردی خاص به وجود می‌آید و ممکن است کسی که در یک حوزه فعالیت می‌کند و زبانی را برای کاربردی خاص استفاده می‌کند، از زبان‌هایی که برای کاربردهای دیگر وجود دارند بی اطلاع باشد. مطالعه زبان‌های برنامه‌نویسی کمک می‌کند نقاط قوت و ضعف و کاربرد زبان‌ها را به خوبی بشناسیم و هر زبان را مناسب با حوزه مرتبط با آن به کار ببریم.

- **افزایش توانایی در یادگیری زبان‌های جدید :** علوم کامپیوتر و تکنولوژی‌های مرتبط با آن همواره در حال پیشرفت هستند و زبان‌های جدید در حال ابداع شدن هستند. یادگیری مفاهیم اصلی در طراحی زبان‌های برنامه نویسی به ما کمک می‌کند تا بتوانیم زبان‌های جدید را بهتر یاد بگیریم. برای مثال وقتی با مفهوم برنامه نویسی شیء گرا آشنا شدیم می‌توانیم از شیء گرایی در همه‌ی زبان‌های شیء گرا استفاده کنیم. این اصل در یادگیری زبان‌های طبیعی نیز صادق است. هرچه با گرامرهای بیشتری در زبان‌های متفاوت آشنا باشیم یادگیری یک زبان طبیعی جدید برای ما آسان‌تر می‌شود.

دلایل مطالعه زبان‌های برنامه نویسی

- **استفاده بهتر از یک زبان:** وقتی با نحوه پیاده سازی یک زبان آشنا شویم، می‌توانیم از آن به طور بهینه‌تر استفاده کنیم. به طور مثال وقتی بفهمیم در زبان‌های شیء گرا، وراثت چه سر باری تحمیل می‌کند متوجه می‌شویم چه جاهایی بهتر است از وراثت استفاده کنیم و چه جاهایی از آن استفاده نکنیم. به عنوان مثالی دیگر، برنامه نویسی که از سربار و پیچیدگی فراخوانی توابع آگاه نباشد، ممکن است از توابع به کثرت استفاده کند که این امر باعث کاهش بهره‌وری برنامه می‌شود. بنابراین با یادگیری مفاهیم پایه در طراحی زبان‌ها می‌توانیم برنامه نویس بهتری شویم. همچنین یادگیری مفاهیم برنامه نویسی به یک برنامه نویس کمک می کند تا از قابلیت‌هایی که تاکنون استفاده نکرده است، استفاده کند.
- **ابداع زبان‌های جدید:** در نهایت با مطالعه مفاهیم زبان‌های برنامه نویسی می‌توانیم دیدگاه روشن‌تری نسبت به گستره‌ی زبان‌های برنامه نویسی به دست آوریم و بنابراین می‌توانیم زبان‌های جدیدی را به فراخور نیازهای خود ابداع و پیاده سازی کنیم.

حوزه‌های برنامه نویسی

- کامپیوترها و برنامه‌های کامپیوتر در حوزه‌های مختلفی استفاده می‌شوند. از سیستم‌های تعبیه شده در هواپیما گرفته تا سیستم های محاسباتی پیچیده در آزمایشگاه‌های تحقیقاتی تا بازی‌های کامپیوتری در کامپیوترهای خانگی یا تلفن‌های همراه.
- **کاربردهای علمی :** اولین کامپیوترهای دیجیتال در دهۀ ۱۹۴۰ و ۱۹۵۰ برای انجام محاسبات علمی و استفاده در آزمایشگاه‌های تحقیقاتی به وجود آمدند. در آن زمان برنامه‌های پیچیده و ساختارداده‌های پیچیده وجود نداشتند، اما به محاسبات پیچیده نیاز بود. معمولا برای انجام چنین محاسباتی به آرایه و ماتریس و یک حلقه برای شمارش نیاز بود. بنابراین زبان‌هایی که در آن زمان ابداع شدند، در جهت رفع نیازها طراحی شده بودند. قبل از آن دوره از زبان اسمبلی استفاده می شد. اولین زبانی که برای محاسبه فرمول‌های ریاضی به وجود آمد زبان فورترن [1] بود. در آن دوره زبان الگول [2] نیز به وجود آمد. اما زبان فورترن به نحوی برای آن کاربرد خاص طراحی شده بود که هنوز هم مورد استفاده است. زبان‌های آن زمان به دلیل محدودیت سخت‌افزار به کارایی قابل توجهی نیاز داشتند.

[1] Fortran
[2] Algol

حوزه‌های برنامه نویسی

- **کاربردهای تجاری**: استفاده از کامپیوترها برای کاربردهای تجاری در ابتدای دهه‌ی ۱۹۵۰ آغاز شد. اولین زبانی که در این حوزه به وجود آمد زبان کوبول [1] بود. در کاربردهای تجاری نیاز به سهولت در گزارش‌گیری و همچنین انجام دقیق محاسبات تجاری است. توسعه برنامه‌های کاربردی برای انجام محاسبات در بانک‌ها یکی از موارد کاربردهای تجاری است.
- **هوش مصنوعی**: یکی از کاربردهای مهم در محاسبات کامپیوتر در حوزه هوش مصنوعی است. در این حوزه از ماتریس‌ها و آرایه‌ها و انجام محاسبات بر روی این ساختارها به کثرت استفاده می‌شود. اولین زبانی که در این حوزه به وجود آمد زبان لیسپ [2] بود که یک زبان تابعی است و در سال ۱۹۵۹ ابداع شد. در دهۀ ۱۹۷۰ زبان دیگری برای کاربردهای هوش مصنوعی به نام زبان پرولوگ [3] به وجود آمد. در دهۀ اخیر بسیاری از برنامه‌های هوش مصنوعی با استفاده از پایتون [4] نوشته می‌شوند.

[1] Cobol
[2] Lisp
[3] Prolog
[4] Python

حوزه‌های برنامه نویسی

- **نرم‌افزارهای وب:** صفحه‌های سادهٔ وب در بدو ابداع آن با استفاده از زبان اچ‌تی‌ام‌ال [1] طراحی می‌شدند. با پیشرفت تکنولوژی وب و نیاز به صفحه های پویا (صفحه‌هایی که محتوای آن قابل تغییر است)، نیاز به زبان‌های دیگری شد که می‌توان در این دسته به جاوااسکریپت [2] یا پی‌اچ‌پی [3] اشاره کرد.

[1] HTML
[2] JavaScript
[3] PHP

ارزیابی زبان‌ها

- برای ارزیابی زبان‌های برنامه نویسی تعدادی معیار را در نظر می‌گیریم.
- **(۱) خوانایی :** یکی از مهم‌ترین معیارها برای ارزیابی زبان‌های برنامه نویسی، خوانایی است، یعنی سهولت خواندن و فهمیدن برنامه‌ای که توسط برنامه نویسان دیگر نوشته شده است. قبل از ۱۹۷۰ به علت محدودیت های سخت‌افزار، آنچه در یک برنامه بیشتر اهمیت داشت کارایی یک برنامه بود. به تدریج برنامه های پیچیده‌تری به وجود آمدند و همچنین سخت‌افزارهای قوی‌تر ابداع شد، بنابراین به مرور زمان آنچه اهمیت بیشتری می‌یافت، خوانایی برنامه‌ها بود. همچنین نگهداری برنامه‌ها با به وجود آمدن برنامه‌های پیچیده و بزرگ اهمیت بیشتری پیدا می‌کرد، بنابراین لازم بود برنامه ها به میزان کافی خوانا باشند. پس زبان‌های برنامه نویسی که تا آن زمان به زبان ماشین شباهت بیشتری داشتند، به مرور زمان به زبان انسان شباهت بیشتری پیدا کردند. خوانایی را می‌توان از جنبه‌های مختلفی بررسی کرد : سادگی، تعامد [1] ، نوع‌های داده‌ای، طراحی نحوی [2] .

[1] Orthogonality
[2] Syntax design

ارزیابی زبان‌ها

- **سادگی:** سادگی یک زبان برنامه نویسی، بر خوانایی آن تأثیر مستقیم دارد. زبانی که تعداد زیادی ساختار کنترلی وکلمات کلیدی دارد طبیعتاً برای یادگیری سخت‌تر است. وقتی یک زبان، بسیار پیچیده می‌شود، معمولا برنامه نویسان تنها از قسمتی از زبان استفاده می‌کنند.
- این سادگی باید در حد متوسط باشد تا برنامه بهترین خوانایی را داشته باشد. اگر یک برنامه بیش از حد ساده باشد خوانایی آن کاهش می‌یابد. برای مثال، زبان اسمبلی زبان بسیار ساده‌ای است اما خواندن یک برنامه در زبان اسمبلی نسبت به زبان سی سخت تر است.

ارزیابی زبان‌ها

- **تعامد:** تعامد در یک زبان برنامه نویسی بدین معناست که مجموعه‌ای از ساختارهای ابتدایی در یک زبان بتوانند به چندین روش محدود با یکدیگر ترکیب شوند و همهٔ ساختارهای داده و کنترلی مورد نیاز را بسازند. اگر همهٔ ساختارهای ابتدایی بتوانند با یکدیگر ترکیب شوند و ترکیب‌های معناداری بسازند، می‌گوییم یک زبان تعامد بالایی دارد. تعامد کم در یک زبان بدین معناست که برخی ترکیب‌ها معنادار نیستند. بنابراین یک زبان با تعامد کم استثناهای بیشتری دارد. تعداد استثناهای زیاد در یک زبان یادگیری زبان را پیچیده تر می‌کند.

- برای مثال در کامپیوترهای IBM دو دستور برای جمع وجود داشت. دستور `A Reg, Memory` محتوای یک رجیستر و یک خانه حافظه را جمع می‌کند و دستور `AR Reg1, Reg2` محتوای دو رجیستر را جمع می‌کند. بنابر این برخی از ترکیب‌ها در این زبان بی‌معنا هستند. مثلا توسط دستور A نمی‌توان دو رجیستر را با هم جمع کرد که این یک استثنا در زبان است. اما در کامپیوترهای سری VAX تنها یک دستور `ADDL op1, op2` برای جمع طراحی شده بود. بنابراین رجیسترها و حافظه‌ها به شکل‌های مختلف می‌توانند با استفاده از این دستور با هم جمع شوند و در نتیجه زبان این نوع کامپیوترها تعامد بیشتری دارد.

ارزیابی زبان‌ها

- اگر تعامد کم باشد، تعداد استثناها افزایش می‌یابد و در نتیجه نوشتن و خواندن برنامه سخت می‌شود. از طرفی دیگر اگر تعامد زیاد باشد، ترکیب‌های پیچیده‌ای توسط زبان به وجود می‌آیند و باز هم خوانایی برنامه کاهش می‌یابد.
- در زبان سی برای مثال یک ساختمان (استراکت) را می‌توانیم توسط یک تابع بازگردانیم ولی یک آرایه را نمی‌توانیم بازگردانیم. اعضای یک ساختمان میتوانند از هر نوعی باشند غیر از `void` . مقادیر به توابع با مقدار داده می‌شوند اما اگر آرایه به تابع ارسال کنیم، با ارجاع به تابع داده می‌شود. نوع `long long int` وجود دارد ولی `long long double` وجود ندارد. این استثناها از تأثیرات تعامد کم است که زبان را برای یادگیری و استفاده پیچیده تر می‌کند.
- از طرفی اگر تعامد خیلی زیاد باشد نیز زبان پیچیده می‌شود. مثلا در زبان الگول نوع‌ها می‌توانند با هم ترکیب شوند و نوع‌های بسیار پیچیده‌ای بوجود آورند. در این حالت هیچ استثنایی وجود ندارد و بنابراین تعامد بسیار بالا است ولی همین امر خوانایی برنامه را کاهش می‌دهد.

ارزیابی زبان‌ها

- **نوع داده:** وجود امکانات کافی برای نوع داده‌ها و ساختمان داده‌ها در یک زبان به خوانایی آن زبان کمک می‌کند. برای مثال در زبانی که نوع منطقی وجود ندارد، مجبوریم برای تعریف مقادیر درست و نادرست از اعداد صفر و یک استفاده کنیم که این امر موجب کاهش خوانایی کد می‌شود.

- **طراحی قوائد نحوی:** قوائد نحوی در یک زبان تأثیر مهمی بر روی خوانایی برنامه در آن زبان دارد. برای مثال در زبان فورترن برای اتمام یک بلوک از کد از یک کلمهٔ کلیدی متناسب با آن بلوک استفاده می‌شود و بدین ترتیب خوانایی کد در آن نسبت به زبان سی که همهٔ بلوک‌ها در آن با آکولاد پایان می‌یابند بالاتر است. مثلا برای خاتمه بلوک IF در فورترن از END IF استفاده می‌شود. در اینجا می‌بینیم که برای بالا بردن خوانایی در یک زبان می‌توان کلمات کلیدی را افزایش داد ولی از طرفی افزایش کلمات کلیدی، زبان را پیچیده تر و برای یادگیری سخت تر می‌کند، بنابراین رعایت حد اعتدال در اینجا نیز بسیار دارای اهمیت است.

ارزیابی زبان‌ها

- همچنین برای خوانایی بهتر، قوائد نحوی باید به گونه‌ای طراحی شوند که صورت و معنی یک عبارت همخوانی داشته باشند. برای مثال در زبان سی اگر متغیری درون یک تابع با استفاده از کلمهٔ کلیدی `static` تعریف شود، آن متغیر در اولین اجرای تابع بر روی حافظه در بخش داده تعریف می‌شود و مقدار خود را در فراخوانی‌های پی در پی حفظ می‌کند. اما اگر متغیری به صورت عمومی با کلمهٔ `static` تعریف شود، آن متغیر فقط در آن فایل قابل استفاده است. پس این کلمه کلیدی معانی متعددی دارد که این امر از خوانایی برنامه می‌کاهد ولی از طرف دیگر تعداد کلمات کلیدی کم باعث سادگی زبان می‌شود.

ارزیابی زبان‌ها

- **(۲) قابلیت اطمینان** [1]: یک برنامه قابل اطمینان است اگر همیشه همان کاری را انجام دهد که برنامه‌نویس انتظار دارد. به عبارت دیگر برای افزایش قابلیت اطمینان خطاهای برنامه نویس باید توسط کامپایلر تشخیص داده شود.
- از جمله عواملی که بر قابلیت اطمینان تأثیر می‌گذارند، عبارتند از بررسی نوع [2]، مدیریت استثنا [3]، نام‌های مستعار [4]، و خوانایی

[1] Reliability
[2] Type checking
[3] Exception handling
[4] Aliasing

ارزیابی زبان‌ها

- **بررسی نوع**: بررسی نوع بدین معناست که خطاهای نوع داده‌ای در یک برنامه توسط کامپایلر یا در هنگام اجرا تشخیص داده شوند. بررسی نوع در زمان اجرا هزینه بالایی دارد بنابراین در زبانی که به سرعت اجرای زیاد نیاز دارد، بررسی نوع توسط کامپایلر انجام می‌شود. همچنین اگر خطاها در زمان کامپایل شناخته شوند، می‌توان آنها را زودتر رفع کرد و نیازی به اجرای برنامه برای رفع خطا نیست. برای مثال در نسخه‌های اولیه زبان سی، هنگام ارسال یک متغیر به تابع، لزومی نداشت نوع آن متغیر با نوع داده‌ای که تابع دریافت می‌کند همخوانی داشته باشد. این امر موجب خطاهای احتمالی توسط برنامه نویس می‌شد و قابلیت اطمینان زبان را پایین می‌آورد.

ارزیابی زبان‌ها

- **مدیریت استثنا:** توانایی یک برنامه برای تشخیص خطاها در زمان اجرا و ادامه برنامه در صورت بروز خطا قابلیت اطمینان برنامه را بالا می‌برد. به این قابلیت مدیریت استثنا گفته می‌شود. برای مثال وجود مدیریت استثنا در زبان سی++ این زبان را نسبت به سی قابل اطمینان‌تر می‌کند.
- **نام‌های مستعار:** با استفاده از قابلیت ایجاد نام‌های مستعار، یک خانه حافظه می‌تواند دو یا چند نام داشته باشد. برای مثال اشاره‌گرها و متغیرهای ارجاعی در سی++ می‌توانند به یک متغیر چندین نام را منتسب کنند. وجود چنین قابلیتی امکان بروز خطا را بیشتر و در نتیجه قابلیت اطمینان را کمتر می‌کند.
- **خوانایی:** هرچه خوانایی یک زبان بیشتر باشد، برنامه نویس راحت‌تر می‌تواند برنامه‌هایی بنویسد که درست‌تر و در نتیجه قابل اطمینان‌تر باشند.

ارزیابی زبان‌ها

- **(۳) هزینه:** انگیزه ابتدایی طراحی زبان‌های برنامه نویسی پایین آوردن هزینهٔ نوشتن آنها بود. برنامه نویسان می‌توانستند با استفاده از یک زبان برنامه نویسی سطح بالا یک برنامه را در زمان کمتری بنویسند. حال هر چه یک زبان ساده‌تر باشد یادگیری آن نیز نیازمند زمان کمتری است و نوشتن برنامه توسط آن ساده‌تر است و در نتیجه هزینه مالی کمتری برای آن صرف می‌شود.
- هزینهٔ زمانی اجرای یک برنامه نیز یکی از معیارهای ارزیابی یک زبان است. هرچه یک زبان در زمان کمتری یک برنامهٔ معین را اجرا کند، آن برنامه بهتر است. اما گاهی این معیار با معیارهای دیگر در تضاد است. برای مثال اشاره‌گرها گرچه قابلیت اطمینان را پایین می‌آورند، اما از هزینه اجرا نیز می‌کاهند و بهبود می‌دهند.

ارزیابی زبان‌ها

- گاهی (برای مثال در یک برنامهٔ حسابداری) خوانایی یک برنامه برای ما مهم‌تر است پس زبانی را انتخاب می‌کنیم که خواناتر باشد و هزینه اجرا برای ما در اولویت نیست. اما گاهی (برای مثال در سیستم‌های تعبیه شده در هواپیما) زمان اجرا بسیار پر اهمیت است و خوانایی اهمیت زیادی ندارد.
- هزینه می‌تواند با قابلیت اطمینان به گونه‌ای دیگر نیز در ارتباط باشد. قابلیت اطمینان پایین ممکن است باعث بروز خطا شود و در یک سیستم حساس مانند هواپیما، ممکن است باعث تحمیل هزینه‌های مالی زیادی شود.
- هزینه نگهداری یک نرم‌افزار نیز معیار مهمی است. هرچه خوانایی یک برنامه بالاتر باشد، هزینه نگهداری آن پایین‌تر است چرا که برنامه نویسان آتی می‌توانند برنامه را به آسانی فرا بگیرند و تغییر دهند.

ارزیابی زبان‌ها

- در بسیاری مواقع یک زبان برنامه‌نویسی قسمتی از وظیفهٔ برنامه‌نویس را به طور خودکار انجام می‌دهد. به طور مثال یک برنامه‌نویس، وظیفه دارد حافظه را مدیریت کند، اما برخی از زبان‌های برنامه‌نویسی مدیریت حافظه را به طور اتوماتیک و خودکار انجام می‌دهند و حافظه‌های تخصیص داده شده را وقتی به آنها دیگر نیازی نیست به طور خودکار آزادسازی می‌کنند. گرچه چنین خودکارسازی‌هایی باعث می‌شود برنامه‌نویس نیازی به فکر کردن نداشته باشد، اما از طرف دیگر باعث می‌شود از سرعت اجرای برنامه‌ها در یک زبان برنامه‌نویسی کاسته شود. وجود ساختارها و نوع‌های داده‌ای متنوع قدرت بیان را بیشتر می‌کند.
- قدرت بیان در مقابل کارایی [1]: هر چه قدرت بیان یک زبان بیشتر باشد (کارهای بیشتری توسط زبان به طور خودکار انجام شوند)، برنامه‌نویس نیاز به زمان کمتری برای نوشتن برنامه دارد، اما برنامه با سرعت کمتری اجرا می‌شود و کارایی کمتری دارد. برخی مواقع نیاز است برنامه قدرت بیان بیشتری داشته باشد تا پیچیدگی برنامه کمتر شود و خطاهای برنامه کاهش یابد و برخی مواقع نیاز است که یک برنامه کارایی بیشتری داشته باشد.

[1] Expressiveness versus efficiency

ارزیابی زبان‌ها

- **معیارهای دیگر**: یکی از معیارهای مقایسه زبان‌ها می‌تواند قابلیت اجرای برنامه‌های آن بر روی سیستم‌ها و معماری های سخت‌افزاری مختلف یا قابلیت جابجایی [1] باشد. هرچه یک زبان دارای استانداردهای بهتری باشد، کامپایلرهای متنوع در سیستم عامل‌های مختلف همگون‌تر پیاده سازی می‌شوند و بنابراین یک کد واحد را می‌توان بر روی سیستم‌های مختلف کامپایل و اجرا کرد. به عنوان یک معیار دیگر می‌توانیم عمومی بودن یا اختصاصی بودن یک زبان را در نظر بگیریم. برخی از زبان‌ها برای یک استفاده خاص به کار می‌روند و برخی از زبان‌ها در کاربردهای متنوع‌تری می‌توانند استفاده شوند.

- در طراحی زبان‌های برنامه نویسی معمولا معیارهای زیادی وجود دارد که این معیارها با هم در تضادند و بدست آوردن حد وسط مناسب برای یک کاربرد خاص بسیار حائز اهمیت است. برای مثال در زبان جاوا بررسی می‌شود که دسترسی به اندیس‌های آرایه در محدوده تعریف شدهٔ آرایه باشد. بدین ترتیب قابلیت اطمینان جاوا از سی بیشتر است ولی این بررسی اجرای برنامه جاوا را کندتر می‌کند، پس هزینه اجرا افزایش پیدا می‌کند.

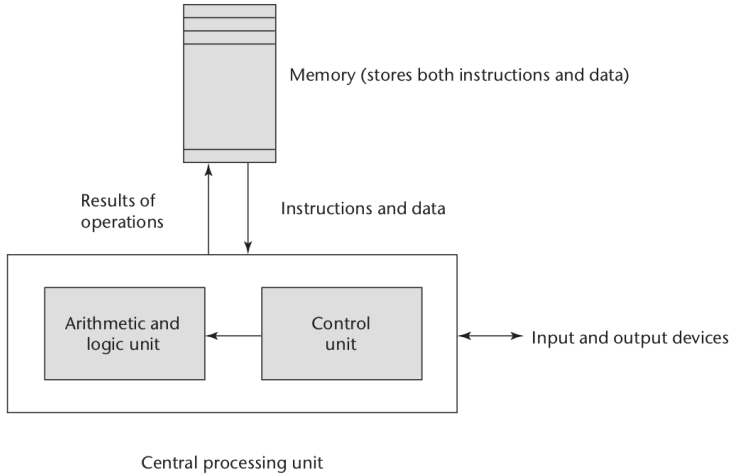
[1] Portability

عوامل تأثیرگذار بر طراحی زبان

- دو عامل مهم تأثیرگذار بر طراحی زبان‌های برنامه‌نویسی عبارتند از معماری کامپیوتر و متودهای طراحی.
- **معماری کامپیوتر**: معماری سخت‌افزار عامل تأثیرگذار مهمی در طراحی زبان‌های برنامه‌نویسی است. همهٔ زبان‌های ۶۰ سال گذشته تحت تأثیر معماری وان نویمان [1] بوده‌اند. در معماری وان نویمان کد برنامه و داده‌ها هر دو بر روی حافظهٔ اصلی قرار می‌گیرند. سپس واحد پردازنده مرکزی، دستورات برنامه را یک به یک از حافظه می‌خواند و اجرا می‌کند. نتیجه محاسبات دوباره بر روی حافظه قرار می‌گیرد. همه کامپیوترها از دهه ۱۹۴۰ تا کنون بر اساس این معماری ساخته شده‌اند که این معماری نیز بر اساس ماشین تورینگ طراحی شده و ماشین تورینگ نیز از موتور تحلیلی چارلز بابج الهام گرفته است. بنابراین همه زبان‌ها نیز طبیعتاً تحت تأثیر این معماری بوده‌اند.

[1] Van Neumann

عوامل تأثیرگذار بر طراحی زبان



شکل: معماری وان نویمان

عوامل تأثیرگذار بر طراحی زبان

- به دلیل استفاده از این معماری اکثر زبان‌های برنامه نویسی از متغیرها استفاده می‌کنند که معرف خانه‌های حافظه است، از عملیات انتساب استفاده می‌کنند که همان عملیات تغییر مقادیر توسط پردازنده است و از حلقه‌ها استفاده می‌کنند که ساده‌ترین روش برای تکرار دسته‌ای از دستورات است.

عوامل تأثیرگذار بر طراحی زبان

- اجرای یک برنامه بر روی این معماری بدین شکل است که پردازنده دستورات را از حافظه می‌خواند و یک به یک اجرا می‌کند. آدرس آخرین دستور برای اجرا در یک رجیستر به نام رجیستر شمارنده برنامه [1] ذخیره می‌شود.
- می‌توان الگوریتم اجرای برنامه را به صورت زیر نوشت :

```
initialize the program counter
repeat forever
    fetch the instruction pointed to by the program counter
    increment the program counter to point at the next instruction
    decode the instruction
    execute the instruction
end repeat
```

[1] program counter

عوامل تأثیرگذار بر طراحی زبان

- برخی از زبان‌های برنامه نویسی عملیات انتساب متغیر ندارند. این زبان‌ها که زبان‌های تابعی نامیده می‌شوند خروجی را براساس اعمال تعدادی توابع بر ورودی محاسبه می‌کنند. این زبان‌ها به طور مستقیم از معماری وان نویمان پیروی نمی‌کنند، اما کامپایلری که برای آنها نوشته می‌شود باید نهایتاً برنامه را به زبان ماشین که براساس معماری وان نویمان است ترجمه کند.
- **متودهای طراحی**: در اوایل دهۀ ۱۹۷۰ نرم‌افزارها برای برنامه‌های بزرگتری به کار می‌رفتند و بنابراین نیاز به سازماندهی بهتر نرم‌افزارها بود. این امر سبب شد روش‌ها و متودهایی برای طراحی برنامه‌های پیچیده ابداع شود که این متودها منجر به ابداع زبان‌هایی شد که تمرکزشان بر روی ساخت نوع داده‌ها و روش‌های سازماندهی برنامه بود. اولین زبانی که برای حل چنین مشکلاتی به وجود آمد زبان سیمولا [1] بود که شروعی برای طراحی زبان‌های شیء گرا بود. پس از آن زبان‌های اسمالتاک [2]، سی++ و جاوا به وجود آمدند.

[1] Simula
[2] Smalltalk

تاریخچهٔ زبان‌های برنامه‌نویسی

- در طول نیم قرن اخیر صدها زبان برنامه‌نویسی طراحی و پیاده‌سازی شده‌اند. بسیاری از زبان‌های برنامه‌نویسی از مفاهیم مشابهی استفاده می‌کنند، بنابراین در اینجا بر روی تعدادی از آنها تمرکز می‌کنیم که مفاهیم متفاوتی را ارائه می‌کنند.
- هر زبان برنامه‌نویسی بر اساس تعدادی الگوواره [1] ساخته شده است.
- یک الگوواره یا پارادایم به طور کلی یک چهارچوب فکری یا مجموعه ای از مفروضات و مفاهیم و ارزش‌هاست که الگوهای مشابهی را می سازند.
- یک پارادایم یا الگوواره برنامه‌نویسی مجموعه‌ای از ساختارهای برنامه‌نویسی برای دسته‌ای از زبان‌های برنامه‌نویسی است که الگوهای مشابه و ویژگی‌های همسانی را می‌سازند. به عبارت دیگر یک پارادایم ویژگی‌های اصلی دسته‌ای از زبان‌های برنامه‌نویسی را تعیین می‌کند.

[1] paradigm

تاریخچهٔ زبان‌های برنامه‌نویسی

- یکی از سؤالاتی که همیشه ذهن منطق‌دانان را مشغول کرده بوده است این بوده که آیا می‌توان یک مدل محاسباتی ارائه کرد که هر چه قابل محاسبه است را محاسبه کند؟ در دههٔ ۱۹۴۰ دو مدل توسط آلن تورینگ و آلونزو چرچ ارائه شد. حساب لامبدا توسط چرچ و ماشین تورینگ توسط دانشجوی چرچ یعنی تورینگ ارائه شد و بعدها هر دو اثبات کردند که این دو مدل از نظر قدرت محاسبات برابرند. با این وجود کامپیوترهای امروزی بر اساس ماشین تورینگ طراحی شده‌اند و نه حساب لامبدا. با این حال دو دسته زبان‌های برنامه‌نویسی به وجود آمد. دسته زبان‌های رویه‌ای بر اساس ماشین تورینگ و دستهٔ زبان‌های تابعی بر اساس مدل چرچ یعنی حساب لامبدا شکل گرفتند.

تاریخچهٔ زبان‌های برنامه‌نویسی

- دو دسته مهم از روش‌های برنامه‌نویسی که بر اساس دو الگوواره و دو چارچوب فکری به وجود آمده‌اند، عبارتند از: برنامه‌نویسی دستوری [2]، و برنامه‌نویسی اعلانی [3].
- در برنامه‌نویسی دستوری، مراحل اجرای یک برنامه گام به گام توسط برنامه‌نویس بیان می‌شود، در حالی که در برنامه‌نویسی اعلانی تنها هدف انجام محاسبات بدون شرح چگونگی انجام آن توصیف می‌شود.

[2] imperative programming
[3] declarative programming

تاریخچهٔ زبان‌های برنامه‌نویسی

- برای مثال در زبان پایتون [1]، به روش برنامه‌نویسی دستوری برای محاسبهٔ عدد فیبوناچی n ام برنامه‌ای به صورت زیر می‌نویسیم:

```
۱ def fib(n) :
۲     i,j = 0,1
۳     for k in range(1,n + 1):
۴         i,j = j, i + j
۵     return j
```

- همین برنامه در زبان هسکل [2]، به روش برنامه‌نویسی اعلانی به صورت زیر نوشته می‌شود:

```
۱ fib 0 = 0
۲ fib 1 = 1
۳ fib n = fib (n-1) + fib (n-2)
```

[1] Python
[2] Haskell

تاریخچهٔ زبان‌های برنامه‌نویسی

- دو پارادایم (الگووارهٔ) مهم را می‌توان زیرمجموعهٔ پارادایم برنامه‌نویسی دستوری، به حساب آورد: برنامه‌نویسی رویه‌ای [1] و برنامه‌نویسی شیءگرا [2].

[1] Procedural programming
[2] Object-oriented programming

تاریخچهٔ زبان‌های برنامه‌نویسی

- در برنامه‌نویسی رویه‌ای، یک برنامه از تعدادی رویه تشکیل شده است. هر رویه مقادیری را به عنوان ورودی می‌گیرد و پس از انجام محاسبات مقادیری را باز می‌گرداند. یک رویه می‌تواند در محاسبات خود از تعدادی متغیر عمومی استفاده کند. زبان‌های مهم در این دسته عبارتند از: فورترن [1]، الگول [2]، کوبول [3] و سی.
- در برنامه‌نویسی شیءگرا، یک برنامه از تعدادی از اشیاء که با یکدیگر در ارتباط هستند تشکیل شده است. هر شیء نمونه‌ای از یک کلاس است و یک کلاس ویژگی‌ها و رفتارهای معینی دارد. زبان‌های مهم در این دسته عبارتند از: سیمولا [4]. اسمالتاک [5]، سی++، و جاوا.

[1] Fortran
[2] Algol
[3] Cobol
[4] Simula
[5] Smalltalk

تاریخچهٔ زبان‌های برنامه‌نویسی

- دو پارادایم مهم را می‌توان زیرمجموعهٔ پارادایم برنامه‌نویسی اعلانی، به حساب آورد: برنامه‌نویسی تابعی [1] و برنامه‌نویسی منطقی [2].

[1] Functional programming
[2] Logic programming

تاریخچهٔ زبان‌های برنامه‌نویسی

- در برنامه‌نویسی تابعی، یک برنامه از تعدادی تابع تشکیل شده است. هر تابع مقادیری را به عنوان ورودی می‌گیرد و مقادیری را باز می‌گرداند. یک تابع به ازای یک ورودی معین همیشه خروجی ثابتی را بازمی‌گرداند. زبان‌های مهم در این دسته عبارتند از: لیسپ [1]، ام‌ال [2]، اوکمل [3]، هسکل [4].
- در برنامه‌نویسی منطقی، یک برنامه از عبارات منطقی تشکیل شده است. زبان پرولوگ [5] در این دسته قرار دارد.

[1] Lisp
[2] ML
[3] Ocaml
[4] Haskell
[5] Prolog

تاریخچهٔ زبان‌های برنامه‌نویسی

- یک پارادایم مهم دیگر در زبان‌های برنامه‌نویسی برنامه‌نویسی همروند [1] و توزیع‌شده [2] است. در برنامه‌نویسی همروند و توزیع‌شده محاسبات به صورت موازی توسط تعدادی پردازنده انجام می‌شود. زبان‌های مهم در این دسته عبارتند از: گو [1]، و ارلنگ [2].

[1] Concurrent programming
[2] Distributed programming
[1] Go
[2] Erlang

تاریخچهٔ زبان‌های برنامه‌نویسی

- برخی از زبان‌ها تعدادی از پارادایم‌ها را پشتبانی می‌کنند. برای مثال زبان پایتون هم یک زبان رویه‌ای است، هم شیءگرا، و هم تابعی. همچنین کتابخانه‌ای برای برنامه‌نویسی همروند ارائه می‌کند، بنابراین می‌توان برای برنامه‌نویسی همروند هم از آن استفاده کرد.
- به طور مشابه جاوا و سی++ گرچه در گروه زبان‌های شیءگرا قرار می‌گیرند، اما کتابخانه‌هایی را ارائه می‌کنند که با استفاده از آنها می‌توان به صورت تابعی و همروند نیز استفاده کرد.
- برخی از زبان‌ها تنها یک پارادایم را پشتیبانی می‌کنند. برای مثال هسکل فقط برای برنامه‌نویسی تابعی به کار می‌رود.

تاریخچهٔ زبان‌های برنامه‌نویسی

- در دههٔ ۱۹۵۰ تعدادی از زبان‌های برنامه‌نویسی برای تسهیل نوشتن دستورات کامپیوتری که تا قبل از آن به زبان اسمبلی نوشته می‌شدند به وجود آمدند. قبل از به وجود آمدن زبان‌های برنامه‌نویسی، به ازای هر نوع معماری سخت‌افزاری یک زبان اسمبلی وجود داشت. با این که زبان اسمبلی زبانی است که به زبان ماشین شباهت بیشتری دارد تا زبان انسان، و بنابراین نوشتن برنامه با استفاده از این زبان نسبتا دشوار است اما به علت کنترلی که برنامه‌نویس بر روی سخت‌افزار دارد، با استفاده از آن برنامه‌های کارآمدی می‌توان نوشت.
- اولین زبان برنامه‌نویسی در یک رسالهٔ دکتری در سال ۱۹۵۱ توسط کورادو بوهم [1] در دانشگاه ای‌تی‌اچ زوریخ توصیف و به همراه یک کامپایلر عرضه شد.
- دو زبان مهم تجاری که در این دهه به وجود آمدند، عبارتند از فورترن و کوبول.
- فورترن در بین سال‌های ۱۹۵۴ و ۱۹۵۶ توسط تیمی به رهبری جان باکوس [2] در آی‌بی‌ام به وجود آمد.

[1] Corrado Bohm
[2] John Backus

تاریخچهٔ زبان‌های برنامه‌نویسی

- نوآوری جدید فورترن این بود که به برنامه نویس کمک می‌کرد تا بتواند فرمول‌های ریاضی را به همان صورتی که بر روی کاغذ نوشته می‌شود بنویسد. در واقع کلمهٔ فورترن مخفف کلمهٔ ترجمه فرمول [1] بود. برای مثال برنامه‌نویسان فورترن می‌توانستند فرمولی مانند $i + 2 * j$ را بنویسند. تا قبل از آن نیاز بود که برنامه نویس متغیر i را در یک رجیستر ذخیره کند و j را در یک رجیستر دیگر. سپس j را دو برابر کند و سپس مقدار این دو رجیستر را با هم جمع کند. بنابراین فورترن به برنامه‌نویسان کمک می‌کرد که فرمول‌های ریاضی را به زبان خود بنویسند و نه به زبان کامپیوتر و کامپایلر عملیات مورد نیاز برای تبدیل فرمول به زبان اسمبلی را انجام می‌داد.

[1] Formula Translation

تاریخچهٔ زبان‌های برنامه‌نویسی

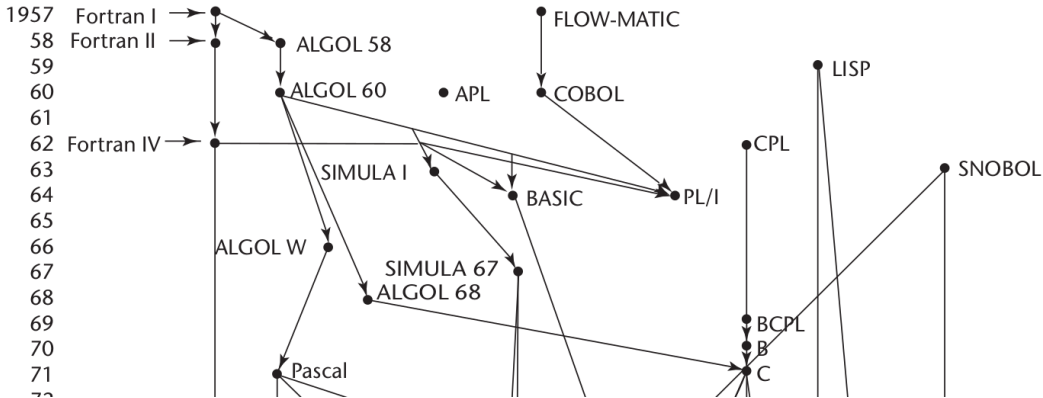
- فورترن همچنین دارای زیربرنامه و همچنین آرایه بود و بدین ترتیب برنامه‌نویسان می‌توانستند برنامه‌های قابل فهم‌تری بنویسند. البته فورترن دارای محدودیت‌هایی نیز بود. به طور مثال با استفاده از فورترن یک تابع نمی‌توانست خود را فراخوانی کند، زیرا برای این کار به تکنیک‌هایی نیاز بود که تا آن زمان به وجود نیامده بودند.
- در همان دوره زبان کوبول نیز برای استفاده در برنامه‌های تجاری توسط گریس هاپر [1] به وجود آمد. دستورات کوبول به زبان انسان شباهت زیادی داشت.

[1] Grace M. Hopper

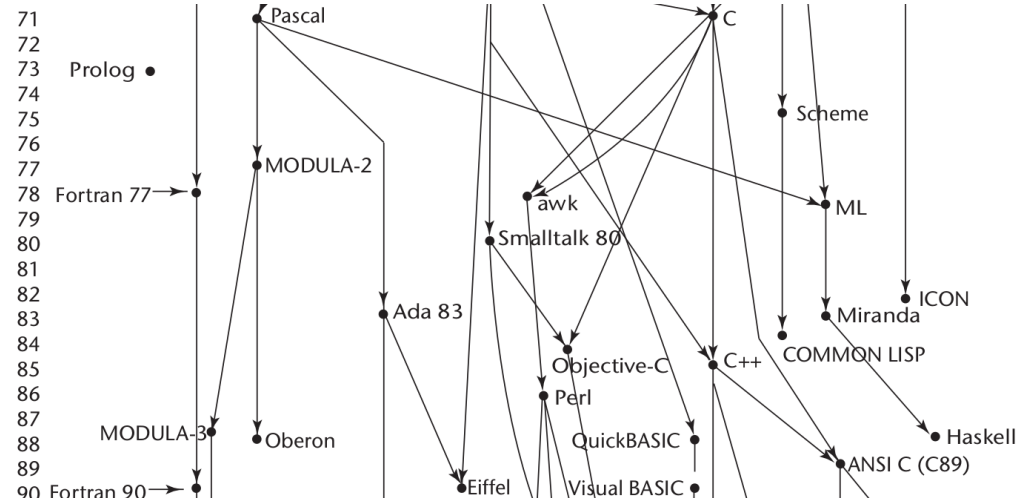
تاریخچهٔ زبان‌های برنامه‌نویسی

- در اواخر دههٔ ۱۹۵۰ و اویل دههٔ ۱۹۶۰ زبان‌های الگول و لیسپ به وجود آمدند. در این زبان‌ها امکان فراخوانی تابع توسط خودش و بنابراین نوشتن توابع بازگشتی وجود داشت.
- در دههٔ ۱۹۷۰ روش‌هایی برای ساختاربندی داده‌ها و ایجاد نوع داده‌های انتزاعی به وجود آمدند. با استفاده از این روش‌ها برنامه‌های پیچیده نظم بیشتری پیدا می‌کردند.
- با افزایش سرعت سخت‌افزار به تکنیک‌هایی برای استفاده بهینه از آنها نیاز بود و بنابراین برنامه‌نویسی همروند برای اجرای قسمت‌های مختلف یک برنامه به طور همزمان و یا اجرای چند برنامه به طور همزمان به وجود آمد. همچنین با به وجود آمدن شبکه‌های کامپیوتری به روش‌هایی برای بهره‌گیری از چندین کامپیوتر به طور همزمان نیاز بود و بنابراین برنامه‌نویسی توزیع شده به وجود آمد.

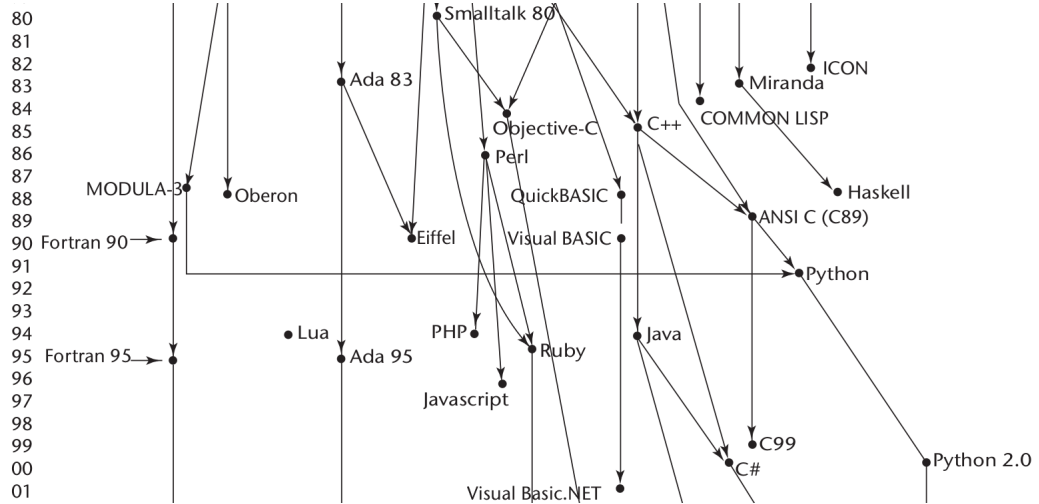
تاریخچهٔ زبان‌های برنامه‌نویسی



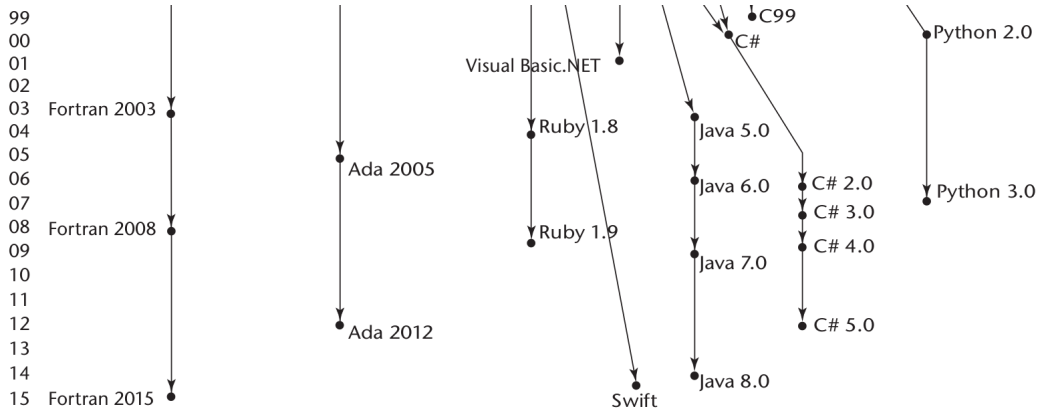
تاریخچهٔ زبان‌های برنامه‌نویسی



تاریخچهٔ زبان‌های برنامه‌نویسی



تاریخچهٔ زبان‌های برنامه‌نویسی



روش‌های پیاده‌سازی زبان‌ها

- زبان‌های برنامه‌نویسی میتوانند به یکی از سه روش زیر پیاده سازی شوند : ترجمه [1]، تفسیر [2]، پیاده‌سازی ترکیبی [3]

[1] complitation
[2] interpretation
[3] hybrid implementation

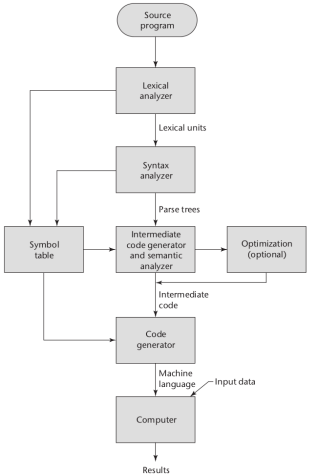
ترجمه

- یک برنامه کامپیوتری می‌تواند مستقیماً به زبان ماشین ترجمه شده، و بر روی کامپیوتر مقصد اجرا شود. این روش را پیاده سازی توسط کامپایلر [1] یا مترجم می‌نامیم.
- مزیت این روش این است که برنامه پس از کامپایل شدن با سرعت بالایی اجرا می‌شود.
- پیاده‌سازی زبان‌هایی مانند کوبول، سی و سی++ با استفاده از کامپایلر صورت گرفته است.
- زبانی را که یک کامپایلر ترجمه می‌کند زبان مبدأ [2] یا زبان منبع نامیده می‌شود.

[1] compiler
[2] source languege

ترجمه

- در شکل زیر مراحل انجام ترجمه نشان داده شده است.



ترجمه

- تحلیل‌گر لغوی [1] متن نوشته شده در برنامه منبع [2] را به واحدهای لغوی یا واژگانی تبدیل می‌کند. این واحدهای لغوی که توکن نامیده می‌شوند می‌توانند شناسه‌ها [3]، کلمات کلیدی [4]، عملگرها [5] و علائم نشانه گذاری [6] باشند. تحلیل‌گر واژگانی، از توضیحات [7] چشم پوشی می‌کند.

[1] lexical analyzer
[2] source program
[3] identifier
[4] key word
[5] operator
[6] punctuation symbol
[7] comment

ترجمه

- تحلیل‌گر نحوی [1] توکن را از تحلیل‌گر لغوی دریافت می‌کند و با استفاده از آنها خطاهای نحوی برنامه را تشخیص می‌دهد و همچنین یک ساختار به نام درخت تجزیه [2] می‌سازد. این درخت‌های تجزیه ساختار نحوی یک برنامه را نشان می‌دهند.
- تولید کنندهٔ کد واسط [3] یک برنامه به یک زبان دیگر تولید می‌کند که حد واسط برنامه منبع و برنامه زبان ماشین [4] است. این زبان میانی معمولا کمی انتزاعی‌تر از زبان اسمبلی است.

[1] syntax analyzer
[2] parse tree
[3] intermediate code generator
[4] machine languege

ترجمه

- تحلیل‌گر معنایی [1] که بخشی از تولید کننده کد میانی یا کد واسط است، خطاهای معنایی برنامه را بررسی می‌کند.
- کد تولید شده توسط تولید کنندهٔ کد میانی، در برخی موارد به یک واحد بهینه سازی [2] داده می‌شود تا کد تولید شده را در صورت امکان کوچک‌تر و سریع‌تر کند. بسیاری از بهینه سازی‌ها را به سختی می‌توان بر روی کد زبان ماشین انجام داد بنابراین این بهینه سازی‌ها بر روی کد واسط انجام می‌شوند.
- در نهایت تولید کنندهٔ کد [3] ، کد بهینه سازی شده در زبان میانی را به برنامه‌ای در زبان ماشین ترجمه می‌کند.

[1] semantic analyzer
[2] optimization
[3] code generator

ترجمه

- جدول علائم [1] اطلاعات مورد نیاز برای فرایند کامپایل را نگهداری می‌کند. محتوای این جدول همهٔ نمادها و نوع آنهاست که برای فرایند تولید کد مورد نیاز است. این گونه اطلاعات توسط تحلیل‌گر لغوی و نحوی در جدولی نگهداری می‌شوند و توسط تحلیل‌گر معنایی و تولیدکننده کد استفاده می‌شوند.
- دقت کنید که برنامه‌ی نوشته شده تقریبا هیچگاه به تنهایی قابل استفاده نیست بلکه معمولاً نیاز به برنامه‌های جانبی است که توسط برنامه‌های دیگر یا سیستم عامل نوشته شده‌اند. برای مثال برای استفاده از ورودی و خروجی، برنامه نیاز به برنامه‌های جانبی دارد که توسط سیستم عامل برای استفاده از ورودی و خروجی مهیا شده‌اند.

[1] symbel table

ترجمه

- بنابراین قبل از این که کد تولید شده به زبان ماشین بتواند اجرا شود، برنامه‌های جانبی باید به برنامه مورد نظر پیوند [1] داده شوند. در این فرآیند پیوند دادن یا لینک کردن، برنامه‌های مورد نیاز دیگر (که توسط برنامه نویسان دیگر پیاده سازی شده‌اند) و یا برنامه‌های سیستمی (که توسط سیستم عامل مهیا شده‌اند) باید به کد ماشین تولید شده لینک شوند. برای این کار آدرس کد ماشین برنامه‌های جانبی به برنامه مورد نظر برای اجرا داده می‌شود.
- فرایند پیوند کدهای جانبی به کد تولید شده، توسط یک پیوند دهنده [2] یا لینکر انجام می‌شود.
- بنابراین لینکر وظیفه دارد کد ماشین تولید شده را به کد ماشین برنامه‌های جانبی که به صورت کتابخانه‌ها با ارائهٔ توابع پر استفاده توسعه داده شده‌اند و کد ماشین برنامه‌های سیستمی که توسط سیستم عامل مهیا شده‌اند، پیوند دهد.

[1] Link
[2] Linker

ترجمه

- یک پیش پردازنده [1] برنامه‌ای است که یک برنامه را قبل از کامپایل شدن پردازش می‌کند. در مرحله پیش پردازش، معمولاً دستوراتی که میانبر برای دستورات دیگر هستند حذف می‌شوند و با دستورات اصلی جایگزین می‌شوند.
- برای مثال در زبان سی با استفاده از دستور `#include "lib.h"` محتوای فایل `lib.h` در ابتدای برنامه قرار می‌گیرد. در مرحله پیش پردازش محتوای فایل مورد نظر به محتوای فایل کد منبع الحاق می‌شود.

[1] preprocessor

ترجمه

- به عنوان مثال دیگر، در زبان سی می‌توانیم با استفاده از دستور `#define` نام‌های نمادین ایجاد کنیم. به قواعد و دستوراتی که مشخص می‌کنند چگونه یک الگوی خروجی بر اساس یک الگوی ورودی تولید شود، ماکرو [1] گفته می‌شود.

- برای محاسبه ماکزیمم دو عدد می‌توانیم ماکرویی به صورت زیر تعریف کنیم.
`#define max(A,B) ((A)>(B) ? (A):(B))`. حال اگر در برنامه منبع داشته باشیم `max(x+2,y)` ، در مرحله پیش پردازش این دستور به دستور `((x+2)>y ? (x+2):(y))` تبدیل می‌شود.

- در پیاده‌سازی زبان به روش ترجمه، سرعت انتقال کد از حافظه به پردازنده معمولا کمتر از سرعت اجرای کد در پردازنده است و بنابراین سرعت اجرای برنامه را سرعت انتقال کد تعیین می‌کند. به این پدیده تنگنای معماری وان نویمان [2] می‌گوییم.

[1] macro
[2] Van Neumann bottleneck

تفسیر

- تفسیر روش دیگری برای پیاده سازی زبان برنامه‌نویسی است. با استفاده از این روش، برنامه‌های زبان منبع توسط یک مفسر [1] خط به خط خوانده می‌شوند، به زبان ماشین تبدیل شده و اجرا می‌شوند.
- مزیت این روش این است که کد برنامه نیاز به کامپایل ندارد و همه جا قابل استفاده است. همچنین با سرعت بیشتری می‌توان برنامه‌های کامپیوتری تولید کرد چرا که کد برنامه را می‌توان به راحتی با استفاده از مفسر خط به خط اجرا و تست نمود.
- از طرفی دیگر عیب این روش این است که معمولاً زمان اجرای برنامه‌ها در آن نسبت به روش ترجمه پایین‌تر است.

[1] interpreter

تفسیر

- در روش تفسیر تنگنای زمان اجرا، کدگشایی دستورات است که بسیار زمان‌بر است و نه انتقال دستورات از حافظه به پردازنده.
- زبان‌هایی مانند ای‌پی‌ال [1] و لیسپ از زبان‌های دهۀ ۱۹۶۰ توسط مفسر پیاده سازی شدند. در سال‌های اخیر زبان‌های وب مانند پی‌اچ‌پی و جاوااسکریپت نیز توسط مفسر پیاده سازی می‌شوند.

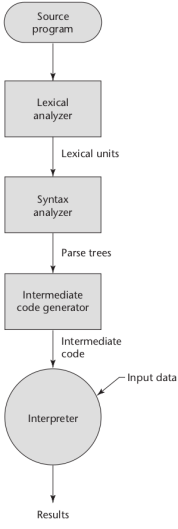
[1] APL

پیاده سازی ترکیبی

- برخی از زبان‌های برنامه‌نویسی توسط یک روش ترکیبی بین تفسیر و ترجمه پیاده سازی شده‌اند. در این نوع پیاده سازی، زبان سطح بالا ابتدا به یک برنامه در یک زبان میانی ترجمه می‌شود. سپس تفسیر از زبان میانی به زبان ماشین آسان‌تر می‌تواند اجرا شود.

پیاده سازی ترکیبی

- روند پیاده سازی زبان در این روش در شکل زیر نشان داده شده است.



پیاده سازی ترکیبی

- زبان پرل [1] و پایتون [2] به این روش پیاده سازی شده‌اند.
- همچنین زبان جاوا به این روش پیاده سازی شده است. زبان میانی بایت کد [3] (کد بایتی) نامیده می‌شود. با استفاده از این روش می‌توان بایت کد را بر روی هر سیستمی که دارای ماشین مجازی جاوا باشد اجرا نمود.
- زبان‌های برنامه‌نویسی به روش‌های مختلفی پیاده سازی شده‌اند. با استفاده از مترجم، سرعت اجرای برنامه بهبود پیدا می‌کند. پیاده‌سازی توسط مفسر برای قابلیت جابجایی برنامه منبع و همچنین استفاده و خطایابی آسان‌تر و پیاده‌سازی سریعتر توسط برنامه نویس ارائه می‌شود. یک پیاده سازی ترکیبی برای فراهم کردن امکان جابجایی کد و سرعت اجرای بالاتر نسبت به مفسر استفاده می‌شود.

[1] Perl
[2] Python
[3] byte code