پروژه زبان‌های برنامه‌نویسی

معرفی زبان سی

ارائه دهندگان:

امیرحسین فتحی

حسین شائمی

تاریخ:

۱۴۰۲/۰۹/۲۰

۱. مقدمه

- زبان برنامه نویسی C، در سال 1972 توسط فردی به نام Dennis Ritchie در آزمایشگاه Bell برای استفاده در سیستم عامل های UNIX توسعه داده شد. هدف کلیدی طراحی زبانی بود که برای برنامه نویسی سیستمی مناسب باشد زبانی که دسترسی به حافظه سطح پایین را فراهم کند و در عین حال مستقل از ماشین، انعطاف پذیر و کارآمد باشد. این زبان به عنوان جانشین زبان برنامه نویسی B در نظر گرفته شده بود و هدف آن بهبود آن با پشتیبانی از متغیرهای محدوده محلی، توسعه پذیری نوع داده و کامپایل آسان تر با استفاده از یک کد میانی قابل حمل برای چندین هدف ماشین بود. زبان C یک زبان دستوری رویه ای می باشد.

در سال‌های نوپای محاسبات، چشم‌انداز، مملو از زبان‌هایی بود که انتزاعات سطح بالا یا کنترل سطح پایین را ارائه می‌کردند، اما به ندرت هر دو را ارائه می‌کردند. هدف C این بود که این شکاف را پر کند و زبان برنامه نویسی را ارائه دهد که بتواند بهترین های هر دو دنیا را به طور یکپارچه ترکیب کند. دنیس ریچی و همکارانش به دنبال ایجاد ابزاری بودند که برنامه‌نویسان را قادر می‌سازد تا پیچیدگی‌های برنامه‌نویسی در سطح سیستم را بدون به خطر انداختن وضوح و بیان مورد نیاز برای توسعه نرم‌افزار هدایت کنند.

منشأ C به طور پیچیده در پروژه یونیکس بافته شده است، جایی که این زبان هدف اولیه خود را پیدا کرد. یونیکس، یک تلاش بلندپروازانه در آزمایشگاه‌های بل، به زبان برنامه‌نویسی نیاز داشت که بتواند کارایی، سادگی و توانایی دستکاری منابع سیستم را در سطح پایین ارائه دهد. C با این هدف خاص ساخته شد تا به عنوان سنگ بنای زبانی برای ساخت سیستم عامل یونیکس عمل کند. نحو و ویژگی های آن برای پاسخگویی به نیازهای برنامه نویسی سیستم طراحی شده است و سطحی از کنترل بر منابع سخت افزاری را فراهم می کند که قبلاً با زبان های موجود مشابه نبود.

C به طور گسترده ای برای سیستم عامل ها و نرم افزارهای سیستمی استفاده می شود که در آن برنامه نویسی سیستم های کارآمد نزدیک به سخت افزار و حافظه مورد نیاز است. همچنان برای سیستم‌های جاسازی شده، درایورهای دستگاه، کامپایلرها، سیستم‌های پایگاه داده، ارتباطات شبکه، مترجمان زبان و کامپایلرها استفاده می‌شود، و همچنین به عنوان یک زبان آموزشی رایج در برنامه‌های علوم کامپیوتر است. انعطاف‌پذیری C به انواعی مانند C++ و Objective-C اجازه می‌دهد تا در برنامه‌های کاربردی و برنامه‌نویسی شی‌گرا بر روی آن بنا شوند.

مقایسه با زبان‌های مشابه:

- نحو و معنای C از زبان‌های قبلی BCPL و B مشتق شده است. در مقایسه با زبان‌های مشابه، قابلیت حمل و نقل را بدون زمان‌های اجرا سربار قابل‌توجه به دست آورد. برخلاف زبان‌های اسمبلی، C انتزاعات

سطح بالایی مانند انواع داده‌های پیچیده و عملگرهای ساخته شده برای برنامه‌نویسی سیستم‌ها را ارائه می‌کند. برخلاف Fortran یا BASIC، C انعطاف پذیرتر، توسعه پذیرتر، مدولار است و دسترسی مستقیم سخت افزاری را فراهم می کند. از نظر قابلیت اطمینان، C برای بررسی‌های ایمنی داخلی محدود مورد انتقاد قرار می‌گیرد که مسئولیت برنامه‌نویس را افزایش می‌دهد اما کد سیستم را با کارایی بالاتر می‌دهد. معاوضه هایی مانند این به همراه کارایی و در دسترس بودن گسترده کامپایلرها، C را به یکی از پرکاربردترین زبان های رویه ای تبدیل کرده است.

زبان برنامه نویسی C برای رسیدگی به چندین مشکل و چالش کلیدی موجود در چشم انداز برنامه نویسی زمان خود طراحی شده است. در اینجا برخی از مسائل اولیه ای که C قصد داشت آنها را حل کند آورده شده است:

- عدم قابلیت حمل:
- مشکل:
- بسیاری از زبان‌های برنامه‌نویسی موجود نزدیک به معماری‌های سخت‌افزاری خاص مرتبط بودند، و نوشتن کدهایی که به راحتی می‌توانستند بر روی ماشین‌های مختلف اجرا شوند، چالش برانگیز بود.
- راه حل C:
- C سطحی از انتزاع را معرفی کرد که امکان حمل بیشتر را فراهم می کرد. استفاده از یک کامپایلر و مفهوم ماشین مجازی (از طریق استفاده از کتابخانه استاندارد C و بعداً کتابخانه الگوی استاندارد C) باعث شد تا برنامه های C بر روی پلتفرم های مختلف بدون تغییر کامپایل و اجرا شوند.
- ناکارآمدی در زبان اسمبلی:
- مشکل: زبان‌های اسمبلی، در حالی که قدرتمند بودند، مختص پلتفرم بودند و اغلب به دانش پیچیده‌ای از سخت‌افزار اساسی نیاز داشتند. نوشتن برنامه های پیچیده در اسمبلی زمان بر و مستعد خطا بود.
- راه حل C: یک انتزاع سطح بالاتر را ارائه می کند در حالی که کنترل سطح پایین را حفظ می کند. این تعادل بین بیان زبان های سطح بالا و کنترل دقیق اسمبلی ایجاد کرد و آن را برای کارهای برنامه نویسی در سطح سیستم کارآمدتر کرد.
- عدم استانداردسازی زبان:
- مشکل: بسیاری از زبان‌های برنامه‌نویسی فاقد مشخصات استاندارد بودند، که منجر به تغییرات در پیاده‌سازی و مانع از انتقال کد می‌شود.

- راه حل C: توسعه C شامل ایجاد یک مشخصات رسمی، استاندارد ANSI C در سال 1989 بود. این استانداردسازی ثبات را در بین پیاده سازی ها تضمین می کرد و محیط برنامه نویسی یکنواخت تر و قابل پیش بینی را تقویت می کرد.
- پیچیدگی برنامه نویسی سیستم:
- مشکل: نوشتن نرم‌افزاری که با سخت‌افزار تعامل نزدیک دارد، مانند سیستم‌عامل‌ها یا درایورهای دستگاه، به تعادلی بین انتزاع و کنترل نیاز دارد که توسط زبان‌های موجود به خوبی پشتیبانی نمی‌شود.
- راه حل C: به طور خاص برای رسیدگی به چالش های برنامه نویسی در سطح سیستم طراحی شده است. این ویژگی هایی مانند اشاره گرها و دستکاری مستقیم حافظه را ارائه می دهد که به برنامه نویسان اجازه می دهد تا به طور موثر با منابع سخت افزاری تعامل داشته باشند.
- نیاز به اجرای کارآمد کد:
- مشکل: زبان های برنامه نویسی سطح بالا اولیه اغلب کدهایی تولید می کردند که کارایی کمتری نسبت به کد اسمبلی دست ساز داشتند.
- راه حل C: طراحی C بر تولید کد ماشین کارآمد تاکید داشت. این باعث شد آن را برای طیف گسترده ای از برنامه ها، از سیستم های تعبیه شده کوچک با منابع محدود گرفته تا محیط های محاسباتی در مقیاس بزرگ، مناسب کند.
- تقاضا برای زبان جهانی برای یونیکس:
- مشکل: توسعه سیستم عامل یونیکس به زبانی نیاز داشت که بتواند کنترل سطح پایین سیستم را فراهم کند و از اجرای طیف وسیعی از ابزارها پشتیبانی کند.
- راه حل C: زبان C به زبان منتخب برای توسعه یونیکس تبدیل شد و امکان ایجاد یک سیستم عامل قابل حمل و کارآمد را فراهم کرد. این همکاری بین C و Unix نقش مهمی در پذیرش گسترده این زبان ایفا کرد.
- با پرداختن به این چالش ها، زبان برنامه نویسی C نقشی اساسی در شکل دهی مجدد چشم انداز برنامه نویسی ایفا کرد، زمینه را برای زبان های بعدی فراهم کرد و بر رویکرد توسعه دهندگان به توسعه نرم افزار تأثیر گذاشت.

ارزیابی زبان برنامه نویسی C در مقایسه با سایر زبان های برنامه نویسی شامل در نظر گرفتن جنبه های مختلفی مانند عملکرد، قابلیت حمل، سهولت استفاده، تطبیق پذیری و حوزه های خاصی است که هر زبان در آن برتری دارد. در اینجا یک ارزیابی مقایسه ای از C در برابر برخی معیارهای کلیدی آورده شده است:

- کارایی:
- راه حل C: که به دلیل عملکرد بالای خود مشهور است، امکان کنترل دقیق بر منابع سیستم را فراهم می کند که در نتیجه کدهایی کارآمد و سریع اجرا می شود. عدم وجود مدیریت خودکار حافظه، مانند جمع آوری زباله، به برنامه نویسان کنترل مستقیم بر تخصیص و تخصیص حافظه می دهد.
- مقایسه: C اغلب از نظر سرعت اجرای خام از زبان‌های سطح بالاتر بهتر عمل می‌کند و آن را به یک انتخاب ترجیحی برای برنامه‌نویسی سیستم‌ها و برنامه‌های کاربردی حیاتی تبدیل می‌کند.
- قابلیت حمل:
- راه حل C: با در نظر گرفتن قابلیت حمل و نقل طراحی شده است که به برنامه ها اجازه می دهد به راحتی با معماری های سخت افزاری مختلف سازگار شوند. با این حال، فقدان کتابخانه های استاندارد شده برای عملکردهای خاص می تواند بر قابلیت حمل در سیستم های مختلف تأثیر بگذارد.
- مقایسه: در حالی که C قابلیت حمل را ترویج می کند، زبان هایی مانند جاوا و پایتون اغلب سطح بالاتری از انتزاع و استقلال پلت فرم را از طریق ویژگی هایی مانند ماشین های مجازی و کتابخانه های استاندارد شده ارائه می دهند.
- راحتی در استفاده:
- راه حل C: زبان C که به دلیل سادگی و نحو ساده اش شناخته شده است، در مقایسه با زبان های سطح پایین مانند اسمبلی نسبتاً آسان است. با این حال، مدیریت حافظه دستی و دستکاری اشاره گر صریح می تواند پیچیدگی هایی را برای مبتدیان ایجاد کند.
- مقایسه: زبان‌های سطح بالاتر مانند پایتون یا جاوا اسکریپت انتزاع بیشتری را ارائه می‌کنند و اغلب به کد دیگ بخار کمتری نیاز دارند، که باعث می‌شود برای مبتدیان در دسترس‌تر باشند.
- تطبیق پذیری:
- راه حل C: تطبیق پذیری C در کاربردهای آن در طیف وسیعی از دامنه ها، از سیستم های جاسازی شده و درایورهای دستگاه گرفته تا سیستم عامل ها و محاسبات با کارایی بالا، مشهود است.
- مقایسه: در حالی که زبان C همه کاره است، زبان هایی مانند پایتون به دلیل انتزاعات سطح بالاتر و کتابخانه های گسترده در حوزه هایی مانند توسعه وب و علم داده برتری دارند.
- مدیریت حافظه:

- راه حل C: امکان مدیریت دستی حافظه را از طریق اشاره گرها فراهم می کند و کنترل صریح بر تخصیص و تخصیص حافظه را فراهم می کند. در حالی که این به انعطاف پذیری می دهد، همچنین احتمال خطاهای مرتبط با حافظه را نیز معرفی می کند.
- مقایسه: زبان‌هایی مانند جاوا و سی شارپ مدیریت خودکار حافظه را از طریق جمع‌آوری زباله‌ها ارائه می‌کنند که خطر نشت حافظه و خطاهای بخش‌بندی را کاهش می‌دهد، اما به قیمت هزینه‌های سربار زمان اجرا.
- جامعه و اکوسیستم:
- راه حل C: زبان برنامه نویسی C دارای یک جامعه بالغ و تثبیت شده است، با اکوسیستم وسیعی از کتابخانه ها و ابزار. با این حال، در دسترس بودن بسته های شخص ثالث ممکن است به اندازه برخی از زبان های مدرن تر نباشد.
- مقایسه: زبان‌هایی مانند پایتون و جاوا اسکریپت دارای جوامع پر رونق و اکوسیستم‌های گسترده، با کتابخانه‌ها و چارچوب‌های گسترده هستند که طیف گسترده‌ای از برنامه‌ها را پوشش می‌دهند.
- امنیت:
- راه حل C: دستکاری مستقیم حافظه و اشاره گرها در C می تواند چالش های امنیتی مانند سرریز بافر را در صورت عدم دقت به همراه داشته باشد. آسیب پذیری های امنیتی در برنامه های C می تواند عواقب شدیدی داشته باشد.
- مقایسه: زبان‌های ایمن برای حافظه مانند جاوا یا Rust ویژگی‌هایی را ارائه می‌کنند که خطر مشکلات امنیتی رایج مرتبط با مدیریت دستی حافظه را کاهش می‌دهند.

زبان برنامه نویسی C با چندین ویژگی کلیدی متمایز می شود که آن را از سایر زبان های برنامه نویسی متمایز می کند. این ویژگی ها به تطبیق پذیری، عملکرد و پذیرش گسترده C در حوزه های مختلف کمک می کند. در اینجا برخی از ویژگی های خاص که زبان C را از سایر زبان ها متمایز می کند آورده شده است.

- کنترل سطح پایین:
- تمایز: C دسترسی مستقیم به حافظه را از طریق نشانگرها فراهم می کند و امکان کنترل دقیق بر منابع سیستم را فراهم می کند. این کنترل سطح پایین برای کارهایی مانند برنامه نویسی سیستم ها و توسعه سیستم عامل ها، که در آن دستکاری مستقیم سخت افزار ضروری است، بسیار مهم است.

- تأثیر: در حالی که این ویژگی به C مزیت قدرتمندی از نظر کارایی و انعطاف‌پذیری می‌دهد، همچنین از توسعه‌دهندگان می‌خواهد که حافظه را به طور صریح مدیریت کنند و در صورت عدم دقت، احتمال خطا را افزایش دهند.
- اجرای کارآمد و سریع:
- تمایز: C به دلیل کارایی و اجرای سریع آن مشهور است. این زبان برای تولید کد ماشین فشرده و بهینه طراحی شده است که آن را برای برنامه‌های کاربردی و محیط‌های حیاتی با محدودیت منابع مناسب می‌سازد.
- تأثیر: توانایی تولید کد بسیار کارآمد، C را به عنوان یک انتخاب ارجح برای برنامه نویسی سیستم ها، سیستم های جاسازی شده و برنامه هایی که حداکثر کارایی را می طلبند، قرار داده است.
- محاسبات اشاره گر:
- تمایز: C اجازه می دهد تا محاسبات اشاره گر، امکان دستکاری مستقیم آدرس های حافظه را فراهم کند. این ویژگی برای کارهایی مانند دستکاری آرایه، مدیریت رشته ها و تخصیص حافظه پویا بسیار مهم است.
- تاثیر: در حالی که محاسبات اشاره گر انعطاف پذیری و کارایی را فراهم می کند، نیاز به درک دقیق مدیریت حافظه برای جلوگیری از مشکلات احتمالی مانند سرریز شدن بافر و خطاهای تقسیم بندی دارد.
- نحو ساده و رسا:
- تمایز: C یک نحو ساده و گویا دارد که بر خوانایی و سهولت درک تأکید دارد. این زبان از انتزاع غیر ضروری جلوگیری می کند، آن را در دسترس توسعه دهندگان قرار می دهد و ترجمه مستقیم مفاهیم را به کد تسهیل می کند.
- تأثیر: سادگی نحو C به پذیرش گسترده و سهولت یادگیری آن کمک کرده است، به ویژه برای کسانی که برای اولین بار از زبان هایی مانند اسمبلی یا یادگیری برنامه نویسی در حال گذار هستند.
- دستورالعمل های پیش پردازنده:
- تمایز: C شامل یک پیش پردازنده است که اجازه می دهد تا از دستورالعمل ها برای دستکاری کد منبع قبل از کامپایل استفاده شود. این ویژگی کارهایی مانند تعاریف ماکرو، گنجاندن فایل و کامپایل شرطی را فعال می کند.
- تأثیر: پیش پردازنده ماژولار بودن کد را افزایش می دهد و ایجاد کدهای قابل استفاده مجدد و قابل تنظیم را از طریق استفاده از ماکروها امکان پذیر می کند. با این حال، برای جلوگیری از مشکلات احتمالی و حفظ وضوح کد نیاز به استفاده دقیق دارد.
- بدون مدیریت حافظه خودکار:

- تمایز: C شامل مکانیسم های مدیریت خودکار حافظه، مانند جمع آوری زباله نمی شود. تخصیص و تخصیص حافظه به صراحت توسط برنامه نویس انجام می شود.
- تأثیر: در حالی که این رویکرد کنترل استفاده از حافظه و توزیع را فراهم می کند، همچنین مسئولیت مدیریت حافظه را به طور موثر بر عهده برنامه نویس می گذارد و خطر خطاهای مربوط به حافظه را افزایش می دهد.
- پارادایم زبان رویه ای:
- تمایز: C از پارادایم برنامه‌نویسی رویه‌ای پیروی می‌کند و بر رویه‌ها یا توابع برای سازماندهی و ساختار کد تأکید می‌کند. این شامل پشتیبانی داخلی برای مفاهیم برنامه نویسی شی گرا، مانند کلاس ها و وراثت نمی شود.
- تأثیر: الگوی رویه‌ای در C یک رویکرد مدولار و ساختاریافته را به کد ارتقا می‌دهد، و آن را برای برنامه‌های مختلف مناسب می‌سازد، به‌ویژه برنامه‌هایی که ویژگی‌های شی‌گرا یک نیاز اولیه نیستند.
- کتابخانه استاندارد کوچک:
- تمایز: C دارای یک کتابخانه استاندارد نسبتا کوچک در مقایسه با برخی از زبان های سطح بالاتر است. کتابخانه استاندارد شامل توابع ضروری برای ورودی/خروجی، دستکاری رشته و عملیات اساسی است.
- تأثیر: کتابخانه استاندارد کوچک رویکردی سبک و کارآمد را تشویق می‌کند، و توسعه‌دهندگان اغلب در مواقعی که عملکردهای اضافی مورد نیاز است، به کتابخانه‌های خارجی تکیه می‌کنند که به یک سبک توسعه مدولار کمک می‌کند.

خوانایی

نقاط قوت:

سادگی: سینتکس C نسبتا ساده و مختصر است و به کدی کمک می کند که خواندن و درک آن آسان باشد.

ساختار رویه ای: ماهیت رویه ای C یک جریان مستقیم و خطی در کد را ترویج می کند و به خوانایی کمک می کند.

بدون سربار انتزاع: C از انتزاع غیر ضروری جلوگیری می کند و به توسعه دهندگان این امکان را می دهد تا ترجمه مستقیم منطق خود را به کد مشاهده کنند.

ملاحظات:

حساب اشاره گر: استفاده از اشاره گرها و مدیریت حافظه دستی می تواند پیچیدگی ایجاد کند و برای حفظ خوانایی نیاز به توجه دقیق دارد.

دستورالعمل‌های پیش‌پردازنده: اگرچه قدرتمند، استفاده بیش از حد از دستورالعمل‌های پیش‌پردازنده می‌تواند پیروی از کد را سخت‌تر کند.

قابلیت اطمینان

نقاط قوت:

رفتار قابل پیش بینی: رفتار C به خوبی تعریف شده و قابل پیش بینی است و به اجرای کد قابل اعتماد کمک می کند.

کنترل سطح پایین: کنترل مستقیم بر حافظه و منابع سخت افزاری امکان مدیریت دقیق و کاهش رفتارهای غیرمنتظره را فراهم می کند.

ملاحظات:

مدیریت حافظه دستی: عدم مدیریت خودکار حافظه خطر خطاهای مرتبط با حافظه مانند نشت یا سرریز را به همراه دارد.

رفتار تعریف نشده: C اجازه می دهد تا برخی رفتارها تعریف نشده باشند و اگر با احتیاط رفتار نشود منجر به مشکلات احتمالی می شود.

هزینه (کدگذاری)

نقاط قوت:

کارایی: C به توسعه دهندگان اجازه می دهد تا کدهای بسیار کارآمد بنویسند و سربار زمان اجرا را به حداقل برسانند.

بدون انتزاع در زمان اجرا: فقدان انتزاعات زمان اجرا منجر به کدهایی می شود که با عملیات سطح ماشین همسو می شوند.

ملاحظات:

مدیریت حافظه دستی: تخصیص و واگذاری حافظه به صورت دستی می تواند زمان بر و مستعد خطا باشد.

نحو پرمخاطب: برخی وظایف ممکن است در مقایسه با زبان های سطح بالاتر به خطوط کد بیشتری نیاز داشته باشند که بر سرعت توسعه تأثیر می گذارد.

هزینه (زمان اجرا)

نقاط قوت:

کارایی: کد کامپایل شده C اغلب از نظر عملکرد زمان اجرا در مقایسه با زبان های تفسیر شده یا مدیریت شده کارآمدتر است.

کنترل سطح پایین: دستکاری مستقیم منابع سخت افزاری امکان استفاده بهینه از منابع را فراهم می کند.

ملاحظات:

بدون مدیریت حافظه خودکار: در حالی که مدیریت دستی حافظه می تواند به استفاده کارآمد از حافظه منجر شود، همچنین برای جلوگیری از نشت یا تکه تکه شدن حافظه، به توجه دقیق نیاز دارد.

سربار کامپایل: مرحله تدوین ممکن است زمان بیشتری را در طول توسعه ایجاد کند.

هزینه (نگهداری)

نقاط قوت:

کنترل مستقیم: کنترل سطح پایین بر منابع سیستم می تواند وظایف تعمیر و نگهداری مربوط به برنامه نویسی سیستم و بهینه سازی عملکرد را تسهیل کند.

استاندارد پایدار: استانداردهای پایدار C به سازگاری طولانی مدت و سهولت نگهداری کمک می کند.

ملاحظات:

چالش‌های کد قدیمی: نگهداری از کد C قدیمی می‌تواند چالش‌برانگیز باشد، به‌ویژه اگر فاقد مستندات مناسب باشد یا از شیوه‌های قدیمی پیروی کند.

مدیریت حافظه دستی: اشکال زدایی و نگهداری مسائل مربوط به حافظه در طول زمان سخت تر می شود.

چندین کامپایلر C موجود است که هر کدام ویژگی ها، مزایا و وضعیت توسعه خاص خود را دارند. در اینجا چند کامپایلر قابل توجه C به همراه اطلاعاتی در مورد توسعه دهندگان و وضعیت فعلی آنها آورده شده است:

GCC (مجموعه کامپایلر گنو):

توسعه دهنده: GCC توسط پروژه گنو، به رهبری بنیاد نرم افزار آزاد (FSF) توسعه یافته است.

وضعیت فعلی: GCC به طور فعال توسط جامعه منبع باز توسعه یافته و نگهداری می شود.

مزایای:

به طور گسترده استفاده می شود و بسیار قابل حمل در پلت فرم های مختلف است.

پشتیبانی از چندین زبان برنامه نویسی، آن را به یک ابزار همه کاره تبدیل می کند.

قابلیت های بهینه سازی قوی برای تولید کد کارآمد.

Clang

توسعه دهنده: Clang توسط پروژه LLVM که یک همکاری بین سازمان ها و افراد مختلف است، توسعه یافته است.

وضعیت فعلی: Clang به طور فعال توسعه یافته است و LLVM به طور گسترده در پروژه های مختلف مرتبط با کامپایلر استفاده می شود.

مزایای:

تاکید بر ارائه تشخیص بهتر و پیام های خطا.

طراحی مدولار، امکان استفاده مجدد از اجزا برای اهداف مختلف.

پشتیبانی از تجزیه و تحلیل استاتیک و ادغام با IDE های مختلف.

Microsoft Visual C++

توسعه دهنده: توسط مایکروسافت طراحی شده است.

وضعیت فعلی: Visual C++ به طور فعال توسعه یافته است و بخشی از محیط توسعه یکپارچه Visual Studio (IDE) مایکروسافت است.

مزایای:

ادغام با ابزارها و محیط های توسعه مایکروسافت.

پشتیبانی از ویژگی ها و کتابخانه های خاص ویندوز.

سازگاری قوی با فناوری های مایکروسافت.

کامپایلر C Intel (ICC)

توسعه دهنده: توسط شرکت اینتل توسعه یافته است.

وضعیت فعلی: کامپایلر Intel C به طور فعال توسعه یافته و نگهداری می شود.

مزایای:

تاکید بر تولید کد بسیار بهینه شده برای معماری های اینتل.

پشتیبانی از محاسبات موازی از طریق ویژگی هایی مانند OpenMP.

ادغام با ابزارهای توسعه اینتل.

TinyCC (TCC)

توسعه دهنده: TCC توسط Fabrice Bellard و جامعه ای از مشارکت کنندگان توسعه داده شده است.

وضعیت فعلی: TCC به طور فعال توسعه یافته است و به دلیل سبک وزن و کامپایل سریع آن شناخته شده است.

مزایای:

اندازه باینری کوچک و سرعت کامپایل سریع.

مناسب برای سناریوهایی که تدوین سریع در اولویت است.

می تواند در برنامه های کاربردی برای کامپایل به موقع جاسازی شود.

پلس سی

توسعه دهنده: توسعه دهنده توسط Pelle Orinius.

وضعیت فعلی: Pelles C به طور فعال توسعه یافته است و برای سیستم عامل های ویندوز در دسترس است.

مزایای:

طراحی شده برای توسعه ویندوز و شامل یک محیط توسعه یکپارچه است.

پشتیبانی از توسعه Win32 و Win64.

طیف وسیعی از ابزارها را برای ویرایش منابع و اشکال زدایی فراهم می کند.

محبوبیت

GCC به طور گسترده ای محبوب است و معمولاً در پروژه های منبع باز و تجاری استفاده می شود. پشتیبانی بین پلتفرمی و بهینه سازی قوی آن را به یک انتخاب ترجیحی تبدیل می کند.

Clang به ویژه در زمینه طراحی مدولار LLVM و تمرکز بر ارائه تشخیص بهتر محبوبیت پیدا کرده است.

Microsoft Visual C++ به طور گسترده در توسعه ویندوز، به ویژه با Visual Studio IDE مایکروسافت استفاده می شود.

کامپایلر Intel C در سناریوهایی که بهینه سازی برای معماری های اینتل بسیار مهم است مورد علاقه است.

TinyCC به دلیل اندازه کوچک و جمع‌آوری سریع آن مورد قدردانی قرار می‌گیرد که آن را برای موارد خاص استفاده از طاقچه مناسب می‌کند.

Pelles C در میان توسعه دهندگان ویندوز محبوب است، به ویژه برای محیط توسعه یکپارچه آن که برای توسعه Win32 و Win64 طراحی شده است.

انتخاب یک کامپایلر C اغلب به عواملی مانند پلتفرم هدف، محیط توسعه، الزامات بهینه سازی و ویژگی های خاص ارائه شده توسط هر کامپایلر بستگی دارد. بسیاری از پروژه ها از چندین کامپایلر برای سازگاری و اهداف بهینه سازی استفاده می کنند.

۲. نحو و معناشناسی

گرامر:

```
<program>::= <declaration-list>

<declaration-list>::= <declaration-list> <declaration>
        | <declaration>

<declaration>::= <type-specifier> <identifier> ';'
        | <type-specifier> <identifier> '=' <expression> ';'

<type-specifier>::= 'int'
        | 'float'
        | 'char'
        | 'void'
```

<identifier>::= <letter> <identifier>*
<letter>::= 'a' | 'b' | … | 'z' | 'A' | 'B' | ... | 'Z'

<expression>::= <term>
| <expression> '+' <term>
| <expression> '-' <term>

<term>::= <factor>
| <term> '*' <factor>
| <term> '/' <factor>

<factor>::= <identifier>
| <number>
| '(' <expression> ')'

<number>::= <digit> <number>*
<digit>::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

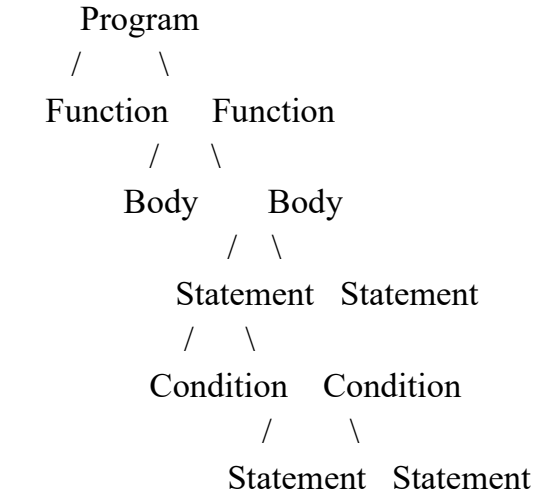
<statement>::= <expression> ';'
| 'if' '(' <expression> ')' '{' <statement-list> '}'
| 'if' '(' <expression> ')' '{' <statement-list> '}' 'else' '{' <statement-list> '}'
| 'while' '(' <expression> ')' '{' <statement-list> '}'

<statement-list>::= <statement-list> <statement>
| <statement>

کلمات کلیدی:

C دارای 32 کلمه کلیدی از جمله اصول اولیه، return plus ،while ،else ،if انواع (double ،float ،char ،int و غیره)، کلاس های ذخیره سازی (استاتیک، خارجی، register، خودکار(auto))، کنترل جریان (case ،switch ،break ،continue). default،) و واجد شرایط (volatile ،const). هر کدام قوانین و معنای خاصی در مشخصات زبان دارند.

```
int main()
{
   int x = 5;
   int y;
   if (x > 3)
   {
      y = 10;
   }
   else
   {
      y = 0;
   }
}
```



پیاده‌سازی if-statement به وسیله Assembly

```
   LOAD condition, R1 ; Load the condition into register R1
    CMP R1, 0; Compare R1 with 0
    JZ elseLabel; Jump to elseLabel if condition is false
//Code to execute if the condition is true
elseLabel:
    //Code to execute if the condition is false or after the if statement
```

Binding در C به مرتبط کردن شناسه هایی مانند نام متغیرها یا تابع ها با تعریف یا اجرای آنها در نقاط مختلف در طول فرآیند کامپایل و اجرا اشاره دارد. C از هر دو نوع انقیاد ثابت و انقیاد پویا پشتیبانی میکند.

انقیاد استاتیک:

در C ، اتصال برای اکثر نام ها در طول زمان کامپایل اتفاق می افتد. انواع متغیرها، اعلان‌های تابع، و غیره قبل از اجرای برنامه به معنای خود محدود می‌شوند. این به عنوان انقیاد ایستا نامیده می شود زیرا اتصال هر بار که برنامه اجرا می شود ثابت می ماند.

چند نمونه از انقیاد استاتیک:

- انواع داده متغیرها، پارامترها، انواع برگشتی و غیره
- توابع بدون اشاره گر تابع
- اعضای ساختار

این امکان بررسی نوع و تشخیص زودهنگام خطاها را فراهم می‌کند، اما انعطاف‌پذیری کمتری دارد زیرا اتصال‌ها در زمان اجرا نمی‌توانند تغییر کنند.

اتصال پویا:

C با استفاده از ویژگی‌هایی مانند اشاره‌گرهای تابع، تماس‌های برگشتی و دسترسی غیرمستقیم از طریق اشاره‌گرها، برخی از امکانات را برای اتصال پویا یا دیرهنگام می‌دهد.

چند نمونه از انقیادهای پویا به شکل زیر میباشند:

- نشانگرهای تابع - آدرس تابع اختصاص داده شده در زمان اجرا
- کتابخانه های بارگذاری شده پویا - نمادها را در زمان بارگذاری پیوند می دهند

این نوع اتصال، انتساب معنی نام ها را تا زمان اجرا که زمینه کامل در دسترس است به تعویق می اندازد. به انعطاف پذیری بیشتری اجازه می دهد، اما خطر خطاهای کشف نشده را تا بعداً به خطر می اندازد.

متغیر های ایستا: این متغیرها در زمان کامپایل در سگمنت داده تخصیص داده می شوند و در طول مدت اجرای برنامه باقی می‌مانند.

```
static int count = 0;
```

متغیر های پشته (خودکار): این متغیرها در فراخوانی تابع در پشته زمان اجرا تخصیص داده می شوند و با بازگشت تابع آزاد می شوند.

```
void func() {
  int x = 0; //allocated and freed each call
}
```

متغیرهای هیپ (صریح): به صورت پویا در طول اجرا با استفاده از malloc/calloc/realloc که حافظه را از هیپ تخصیص می دهد. این حافظه به صورت دستی توسط برنامه نویس آزاد می‌شود.

```
int* p = malloc(sizeof(int)); //allocated on heap
...
free(p); //manual de-allocation
```

متغیرهای هیپ (ضمنی): لفظ‌ها و ثابت‌های رشته‌ای ممکن است از فضای ذخیره‌سازی ثابت استفاده کنند یا به طور ضمنی در بخش‌های فقط با قابلیت خواندن هیپ قرار گیرند.

```
"hello world"; //stored in heap or data segment
```

مقایسه سرعت در این سه حالت:

- از آنجایی که تخصیص حافظه پشته خودکار است سریعتر است.
- تخصیص فضای هیپ با توجه به اینکه باید تابع malloc فراخوانی شود سرعت کمتری دارد.
- تخصیص های ثابت در زمان کامپایل صورت میگیرند و در نتیجه سرباری در زمان اجرا ندارند.

تخصیص پشته برای داده های کوچک و موقت سریع ترین و کارآمدترین نوع تخصیص حافظه است. هیپ در تخصیص اندازه پویا انعطاف پذیر است و پایداری را فراهم می کند. تخصیص استاتیک برای داده های ثابت منطقی است. به صورت کلی بازدهی در این سه نوع این چنین خواهد بود. Stack < Static < Heap

دامنه در زبان C استاتیک/واژگانی است، به این معنی که دامنه در زمان کامپایل بر اساس قرار دادن متغیرها و توابع و مستقل از فراخوانی‌های زمان اجرا تعیین می‌شود.

```
int x = 10;

void func() {
  int y = 20;
}

void otherFunc() {
  printf("%d", y);
}
`
```

دامنه به صورت ایستا بر اساس ساختار کد تصمیم گیری می شود.
برای افزودن دامنه پویا، C باید برای جستجوی متغیرها باید از پشته فراخوانی در زمان اجرا به جای استفاده از دامنه واژگانی استفاده کند. این امر مستلزم ذخیره سازی context متغیر و انجام جستجوی نام به صورت پویا به جای binding ایستا است.
یک مثال:

```
int x = 10; // global

void func() {
  dynamic int y = 20; // dynamic scoped
}

void otherFunc() {
  printf("%d", y); // Now works by search call stack
}
```

بلوک ها در C با { و } تعریف می شوند. متغیرهای محدوده بلاک فقط در داخل بلوک قابل مشاهده است. کلمه کلیدی ویژه static می تواند پیوند متغیر را برای محدود کردن دید تغییر دهد. بنابراین تعاریف دامنه در C به طور کلی غیر از موارد خاص مانند بارگذاری پویا، ثابت هستند.

یک نمای کلی از نوع داده‌ها در سی و پیاده سازی‌ها:

Primary types:

- int - Integer (whole number). Typically 4 bytes. Supports math operators (+, -, *, /). Used for numeric calculations.
- float - Single precision floating point. 4 bytes. Math operators. Fractional values.
- double - Double precision float. 8 bytes. More range than float.
- char - Character. 1 byte. Holds ASCII values. '%c' printing.

Qualifiers:

- Signed/Unsigned - Allow/disallow negative numbers.
- Short/Long - Less/more precision 2/4/8 bytes.

Enumerated types:
Custom integers values named by programmer.

Derived types:

- Array - Contiguous allocation. `int arr[10];`
- Pointer - Holds memory address. `int* p;`
- Struct - Custom aggregate type.

Memory allocation:

- Stack - Local scopes, function calls
- Heap - Dynamic allocation
- Static - Globals, persistent storage

Strings: Null-terminated char arrays:

```
c
Copy code
char str[6] = {'H','e','l','l','o','\0'};
```

Pointers reference addresses in memory:

```
c
Copy code
int x = 5;
int* p = &x; //pointer to x
```

زبان سی هیچ مدیریت کننده حافظه در داخل خود ندارد. بنا به همین موضوع خطر نشتی حافظه و پوینتر های سرگردان وجود دارد. کد نویسی منظم، استفاده از پوینترهای هوشمند، شمارش مرجع ها و ... میتواند از این مسائل جلوگیری کند.

زبان های دیگر مانند جاوا و سی‌شارپ از زباله روبی خودکار جهت ایمنی حافظه استفاده میکنند. این مسئله روی بازدهی تاثیر میگذارد اما مسئولیت و خطاهای برنامه نویس را کاهش میدهد.

۳. منابع

-[The C Programming Language (Second Edition)](https://www.goodreads.com/book/show/515601.The_C_Programming_Language

- [Dennis M. Ritchie's Home Page](https://www.bell-labs.com/usr/dmr/www/)
- [The Development of the C Language](https://www.bell-labs.com/usr/dmr/www/chist.html)
- [POSIX Threads Programming (pthreads)](https://computing.llnl.gov/tutorials/pthreads/)
- [GeeksforGeeks - Mutex in C/C++](https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/)
- [OpenMP](https://www.openmp.org/)
- [Introduction to OpenMP](https://www.openmp.org/wp-content/uploads/OpenMP-4.0-1115-CPP-web.pdf)
- [libuv Documentation](https://libuv.org/)
- [An Introduction to libuv](https://nikhilm.github.io/uvbook/)
- [GNU Compiler Collection (GCC)](https://gcc.gnu.org/)
- [CMake](https://cmake.org/)
- [OpenMP](https://www.openmp.org/)
- [libuv Documentation](https://libuv.org/)
- [Debian APT](https://wiki.debian.org/Apt)
- [Red Hat RPM](https://rpm.org/)
- [Hunter C](https://hunter.readthedocs.io/)
- [vcpkg](https://vcpkg.io/)