"بسمه تعالی"

درس: اصول طراحی کامپایلر

تمرینات شماره ۱

اتوساطغیانی (۹۶۱۱۴۱۵۰۲۴)

تاریخ: ۹۹/۰۲/۲۹

۱ – تفاوت های بین "Machine language" ، "Assembly language" و " High level languages" را توضیح دهید.

زبان‌های برنامه نویسی برای ایجاد دستورالعمل‌ها به کار می‌روند تا کامپیوترها کارهای خاصی را انجام دهند. این زبانها را می‌توان به عنوان زبان ماشین، زبان اسمبلی و زبان برنامه نویسی سطح بالا طبقه‌بندی کرد.

زبان ماشین (Machine language) چیست؟

زبان ماشین پایین‌ترین و ابتدایی‌ترین سطح زبان برنامه‌نویسی است و اولین نوع زبان برنامه‌نویسی بود که ساخته شد. این زبان فقط از ۰ و ۱ تشکیل شده است؛ تنها زبانی است که یک کامپیوتر قادر به درک آن است. بنابراین از هر زبانی که برای نوشتن کد استفاده می‌شود باید برای فهم کامپیوتر به زبان ماشین تبدیل شود.

اگر چه زبان ماشین برای ترجمه کد نیازی به مترجم ندارد و به طور مستقیم توسط کامپیوتر قابل درک و اجراست، اما استفاده از ان برای کاربر پیچیده است. اصلاح یا یافتن خطاها در برنامه‌ای که به زبان ماشین نوشته شده، دشوار است و تعداد معدودی از افراد هستند که می‌توانند از ان استفاده کنند.

زبان اسمبلی (Assembly language) چیست؟

زبان اسمبلی به زبان سطح پایین گفته می‌شود زیرا به سطح سخت افزار نزدیک‌تر است و تا زمانی که نحوه و اجرای آنها را نشناسید، آنها نیز قابل خواندن نیستند. این زبان برای غلبه بر برخی از ناراحتی‌های زیاد زبان ماشین توسعه یافته است.

به عنوان زبان واسطه بین زبان ماشین و زبان‌های برنامه نویسی سطح بالا عمل می‌کند. در مقایسه با زبان ماشین، درک و استفاده از زبان اسمبلی راحت‌تر است. اما پیچیده‌تر از زبانهای برنامه نویسی سطح بالا است. یافتن و تصحیح خطاها در زبان اسمبلی نسبت به زبان ماشین، بسیار آسان‌تر است ولی از آنجا که به سخت‌افزار وابسته است، برنامه‌نویس نیز باید سخت افزار را درک کند تا بتواند به راحتی اصلاح کند.

این زبان از Mnemonics به جای ۰ و ۱ برای نمایش کدگذاری استفاده می کند؛ Mnemonics در زبان اسمبلی دستورالعمل هایی را برای اجرای دستورات ارائه می دهد.

زبان اسمبلی به عنوان Symbolic Language نیز شناخته می شود.

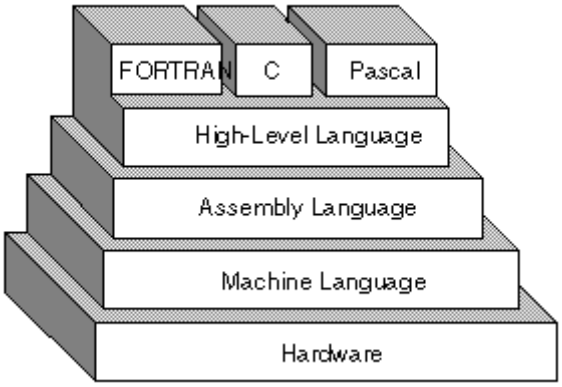
دستورالعمل زبان اسمبلی توسط یک مترجم زبان (اسمبلر) به کدهای ماشین تبدیل می‌شود و سپس توسط کامپیوتر اجرا می‌شود.

زبان سطح بالا (High level languages) چیست؟

زبان سطح بالا زبانی است که از کلمات انگلیسی یا نمادهای ریاضی به جای کد mnemonic استفاده می‌کند، و یادگیری آن آسان‌تر از زبان ماشین و اسمبلی است.

برنامه‌ای که به یک زبان سطح بالا نوشته شده است، توسط کامپایلر به زبان ماشین ترجمه می‌شود و در هر رایانه‌ای که مترجم مناسب برای آن وجود داشته باشد، قابل اجرا است. این زبان مستقل از دستگاهی است که در آن استفاده می‌شود؛ حتی شخصی که معماری ماشین و کد باینری را نمی‌داند، به راحتی می‌تواند از آن استفاده کند.

زبان‌های سطح بالا کاملا به زبان انسان نزدیک هستند و همین امر سبب می‌شود که خواندن و نوشتن ان‌ها اسان‌تر شود. امروزه این زبان ها، بسیار کاربردی، کاربر پسند و پر طرفدار هستند. از جمله این زبان ها می‌توان به: Python , JavaScript , C ,PHP و... اشاره کرد.



تفاوت زبان ماشین، زبان اسمبلی و زبان سطح بالا

۲- در مورد تفاوتهای کلیدی بین compiler و interpreter بحث کنید. کدام یک برای وقتی برنامه‌نویس برنامه خود را اشکال زدایی کند، مناسب تر است؟

compiler و interprete انواع مترجم زبان هستند.

مترجم زبان، نرم افزاری است که برنامه‌ها را از یک زبان مبدا که به صورت قابل خواندن توسط انسان (معمولاً یک زبان برنامه نویسی سطح بالا) است به یک برنامه معادل در زبان ماشین ترجمه می‌کند.

کامپایلر (compiler) چیست؟

کامپایلر یک برنامه کامپیوتری است که کدی را که به زبان برنامه نویسی سطح بالا نوشته شده است به دستورالعمل‌هایی که کامپیوتر قادر به درک آن است (۱ و ۰،باینری) ترجمه می‌کند. کامپیوتر، کد دستگاه را پردازش می‌کند تا کارهای مربوطه را انجام دهد. روند تدوین نسبتاً پیچیده است و زمان زیادی برای تجزیه، تحلیل و پردازش برنامه صرف می‌شود.

یک کامپایلر باید مطابق قانون نحو آن زبان برنامه نویسی باشد که در آن نوشته شده است. کامپایلر، اصطلاحات زبان را برای صحت آن تجزیه و تحلیل می‌کند؛ اگر نادرست باشد، خطایی را وارد می‌کند. خطاها، به صورت همزمان و در انتهای برنامه نمایش داده می‌شوند.

با این حال، کامپایلر فقط یک برنامه است و نمی‌توان خطاهای موجود در آن برنامه را برطرف کند؛ خطاهای برنامه را بدون مراجعه به کد منبع نمی‌توان تغییر داد.

در صورت عدم خطا، کامپایلر کد منبع را به کد دستگاه تبدیل می‌کند.

برنامه خروجی (به صورت exe.) تولید می‌شود که می‌تواند بطور مستقل از برنامه اصلی اجرا شود.

C، C ++، C #، Scala، Java همه از کامپایلر استفاده می‌کنند.

مراحل کامپایلر:

- تحلیلگر لغوی:

متن ورودی خود را در برنامه مورد کامپایل، به‌عنوان یک فایل متن باز گشوده و کاراکتر به کاراکتر می‌خواند و با رسیدن به یک کاراکتر جدا کننده، لغت را تشخیص و تفکیک می‌نماید. در خروجی خود اطلاعات مربوط به لغت را در یک رکورد یا ساختار به نام بسته لغت، قرار می‌دهد که شامل اطلاعاتی در مورد لغت تشخیص داده شده توسط تحلیلگر لغوی مانند: سطر یا ستون و نوع لغت است که آن را در اصطلاح token گویند.

تحلیلگر لغوی در صورت وجود خطا در قالب بندی لغات استفاده شده در متن برنامه مورد کامپایل اعلام خطا لغوی مینماید.

- تحلیلگر نحوی:

تشخیص صحت فرم ظاهری برنامه‌ها از لحاظ دستورالعمل زبان برنامه سازی مربوطه است. با فراخوانی تحلیگر لغوی، لغات را دریافت می‌کند و صحت قرارگرفتن انها نسبت به دستورالعمل را بررسی می‌کند؛ در صورتیکه از نظر دستورالعمل زبان برنامه، مورد کامپایل دارای خطا باشد، پیام خطای نحوی را صادر می‌کند.

- تحلیلگر معنایی:

وظیفه ان تشخیص صحت مفهوم جملات است. ممکن است یک جمله ازنظر نحوی صحیح ولی از لحاظ مفهومی دارای خطا باشد.

تحلیلگر مفهومی وابسته به نوع اسامی و متغیرهای که در جدول نمادها مشخص شده است؛ صحت استفاده آنها را در جملات و عبارات مختلف مورد آزمون قرار می‌دهد و در صورت وجود خطا، خطای معنایی را صادر می‌کند.

- مولد کد میانی:

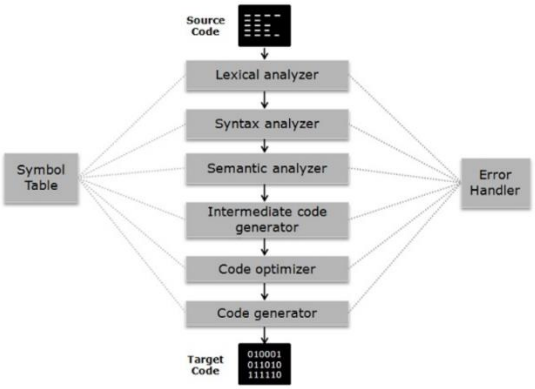
جملات درست تشخیص داده شده توسط تحلیگر معنایی را دریافت می‌کند و در خروجی کد واسطه یا میانی را تولید می‌کند؛ این کد قابل تبدیل و نزدیک به زبان ماشین است اما مستقل از دستگاه است.

- بهینه ساز کد میانی:

در این بخش کد میانی مورد تحلیل قرار می‌گیرد و سعی می‌شود حجم کد دریافتی را با حذف واسطه‌ها و کدهای زائد کاهش داد که سبب افزایش سرعت اجرایی می‌شود.

- مولد کد:

این بخش از کامپایل وابسته به سخت‌افزار است و کد میانی را به کد ماشین تبدیل می‌کند.



مترجم (Interpret) چیست؟

Interpret برنامه ای را که به یک زبان سطح بالا نوشته شده است به زبان دستگاه ترجمه می‌کند.

مترجم اجازه می‌دهد تا هنگام اجرای برنامه، ارزیابی و اصلاح برنامه انجام شود. اجرای برنامه بخشی از فرایند تفسیر است، بنابراین به صورت خطی انجام می‌شود؛ مترجم در حین تفسیر در حافظه وجود دارد و برنامه‌های تفسیر شده می‌توانند برروی رایانه‌هایی که مفسر مربوطه را دارند اجرا شود.

هر دو کامپایلر و مترجمان کار مشابهی را انجام می‌دهند، که تبدیل زبان برنامه نویسی سطح بالاتر به کد دستگاه است. اما، یک کامپایلر قبل از اجرای برنامه، کد را به کد دستگاه تبدیل می‌کند (.exe)، در صورتی که مترجم‌ها هنگام اجرای برنامه، کد را به کد دستگاه تبدیل می‌کنند.

مترجمان کد را به صورت خطی می‌بینند و در صورت وجود خطا، ان را نشان می‌دهند. برای تفسیر خط بعدی باید خطا، تصحیح شود؛ بنابراین بهینه سازی‌ها به اندازه کامپایلرها قوی نیستند.

بهترین گزینه برای محیط برنامه‌نویسی و توسعه است. برخی از زبان های برنامه نویسی که از مترجم استفاده می‌کنند: PHP ، Perl، Ruby .

تفاوت ها:

۱. کد کامپایل شده سریعتر اجرا می‌شود در حالی که کد مفسر کندتر عمل می‌کند.
۲. کامپایلر تمام خطاها را بعد از کامپایل نمایش می‌دهد ، از طرف دیگر Interpreter خطاهای هر خط را یک به یک نشان می‌دهد.
۳. کامپایلر بر اساس مدل بارگیری و ترجمه است ، در حالی که مفسر براساس روش تفسیر است.
۴. کامپایلر برنامه را به صورت کامل می‌گیرد در حالی که مفسر یک خط کد را می‌گیرد.
۵. در کامپایلر کد میانی تولید می شود اما در مقابل مفسر کد واسط ایجاد نمی‌کند.
۶. کامپایلر به دلیل تولید کد شیء به حافظه بیشتری نسبت به مفسر نیاز دارد.
۷. کامپایلر همه خطاها را به طور هم‌زمان ارائه می‌کند، و تشخیص خطاها دشوار است اما در مفسر به دلیل خط به خط نمایش دادن خطاها تشخیص خطاها آسان تر است.
۸. در کامپایلر وقتی خطایی در برنامه رخ می‌دهد ، ترجمه آن متوقف می‌شود و پس از حذف خطا مجدداً کل برنامه ترجمه می‌شود. در مقابل ، وقتی خطایی در مفسر رخ می‌دهد ، از ترجمه آن جلوگیری می‌کند و پس از حذف خطا ، ترجمه از سر گرفته می‌شود.

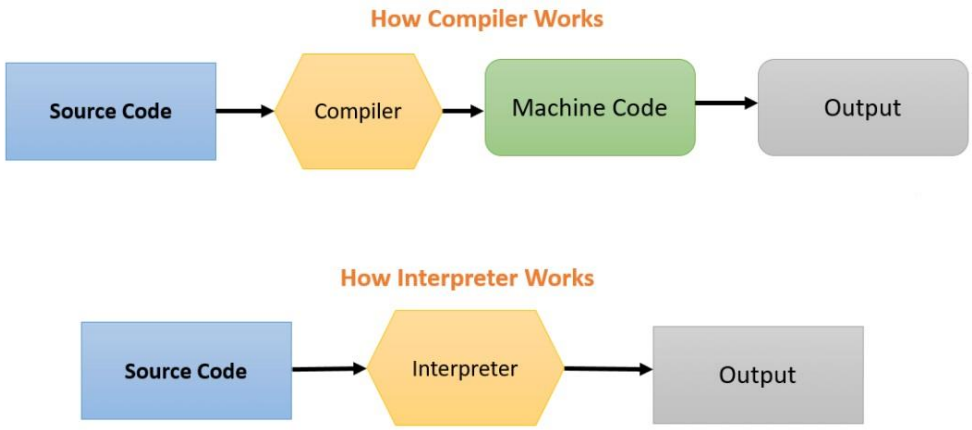
نتیجه گیری:

مهمترین تفاوتی که بین یک Compiler و یک Interpreter وجود دارد روشی است که آنها، کد اجرایی برنامه را اجرا می‌کنند.

برنامه یا کد نرم‌افزاری که توسط یک زبان برنامه‌نویسی مفسری نوشته شده است برای اینکه بتواند بر روی یک سیستم اجرا شود حتما نیاز به این دارد که مفسر روی سیستم نصب شده باشد و تا اینکار انجام نشود اجرا برنامه امکان‌پذیر نیست ؛ اما بر خلاف مفسرها ، کامپایلر یکبار برای همیشه یک برنامه را به زبان اجرایی ماشین تبدیل می‌کند و بعد از ان قابلیت اجرا شدن بر روی هر سیستمی را دارد و در واقع هیچ وابستگی به کامپایلر بعد از تبدیل کد وجود نخواهد داشت.

نقطه منفی استفاده کردن از زبان های مفسری این است که یک Overhead یا دردسر اضافه ایجاد می‌کند. چون کدهای اجرایی برنامه به‌صورت خط به خط اجرا می‌شوند و اینکار باعث بالا رفتن میزان استفاده از منابع CPUو RAM سیستم می‌شود ، اما زمانیکه یک برنامه کامپایل شد ، بصورت یکباره اجرا می‌شود و نیازی به اجرا و پردازش هر خط برنامه بصورت جداگانه نخواهد بود.

همان طور که گفته شد خطایابی در مفسرها به دلیل خط به خط خواندن(Line To Line) کدها اسان‌تر است، این قابلیت به برنامه‌نویس اجازه می‌دهد که هر جایی از برنامه که به مشکل خورد، متوجه شود که در کجا مشکل پیش آمده است و در جهت رفع مشکل اقدام کند. 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 Interpret شود. اما در کامپایلر برای خطایابی باید برنامه به صورت کامل، کامپایل شود تا بتوان خطا را مشاهده کرد، بعد از کامپایل شدن در صورت وجود خطا باید Source Code را تصحیح و نرم‌افزار مجدد کامپایل شود که این پروسه زمانگیر است.



۳- نحوه تهیه و اجرای کدهای C را توضیح دهید.

هر پرونده ای که حاوی یک برنامه به زبان C است باید با پسوند .c ذخیره شود. این امر برای درک کامپایلر ضروری است که این یک فایل برنامه C است.

فرض کنید یک فایل از برنامه ، first.c نامگذاری شده است. پرونده first.c به فایل منبع گفته می‌شود که کد برنامه را نگه می‌دارد. حال ، هنگام تهیه فایل ، کامپایلر C به دنبال خطا است. اگر کامپایلر C خطایی را گزارش ندهد ، آن پرونده را به صورت یک پرونده obj. با همین نام ، به نام فایل شیء ذخیره می‌کند. بنابراین ، در اینجا آن first.obj را ایجاد می‌کند. این پرونده obj. قابل اجرا نیست. این فرآیند توسط Linker ادامه دارد که در نهایت یک پرونده exe. را اجرا می‌کند که قابل اجرا است.

Linker: اول از همه ، باید فهمید که توابع کتابخانه بخشی از هیچ برنامه C نیست بلکه از نرم افزار C است. بنابراین ، کامپایلر عملکرد هیچ عملکردی را نمی‌داند ، چه از نوع چاپ باشد ، چه از نوع اسکن. تعاریف این توابع در کتابخانه مربوطه ذخیره می‌شود که کامپایلر باید بتواند آن را پیوند بدهد. این کاری است که Linker انجام می دهد. پیوند دهنده پرونده‌های شی را به توابع کتابخانه پیوند می‌دهد و برنامه تبدیل به یک پرونده exe. می‌شود. در اینجا، first.exe ایجاد خواهدشد که در یک قالب اجرایی است.

Loader: هروقت دستور اجرای برنامه خاصی را می دهیم ، لودر وارد کار می شود. لودر پرونده exe. را در RAM بارگذاری می‌کند و CPU را با نقطه شروع آدرسی که این برنامه بارگذاری شده است ، مطلع می کند.

هر زمان که یک فایل برنامه C کامپایل و اجرا می‌شود ، کامپایلر برخی از فایل‌ها را با همان نام فایل برنامه C اما با پسوندهای مختلف تولید می‌کند. برای تبدیل شدن به یک برنامه C، چهار مرحله وجود دارد:

۱. پیش پردازش
۲. تلفیقی
۳. مونتاژ
۴. ربط دادن

در اولین مرحله، کد منبع منتقل می‌شود و شامل مراحل زیر است:

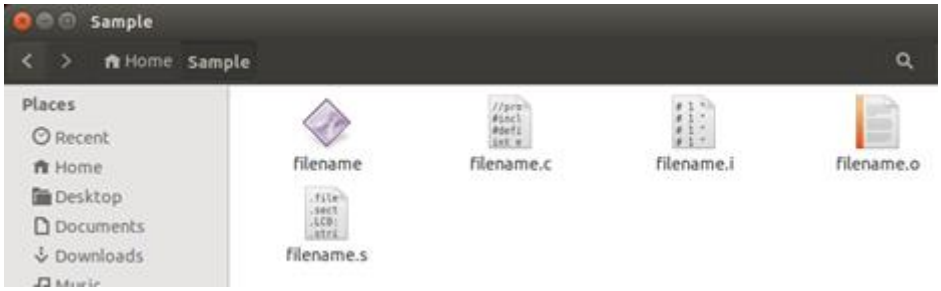
- حذف نظرات
- گسترش ماکرو
- گسترش پرونده های درج شده
- تدوین شرطی

خروجی این مرحله در فایلی به نام filename.i ذخیره می‌شود.

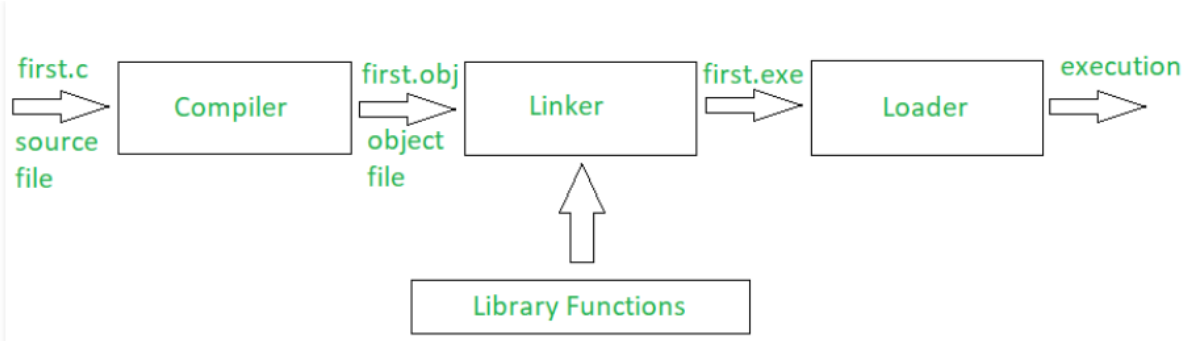
مرحله دوم ، کامپایل فایل filename.i است و خروجی این مرحله به نام فایل filename.s ذخیره می‌شود؛ این پرونده به زبان مونتاژ است که اسمبلر می‌تواند ان را درک کند.

مرحله سوم ، فایل filename.s را به عنوان ورودی دریافت کرده و توسط اسمبلر به filename.o تبدیل می‌شود؛ این پرونده شامل دستورالعمل های سطح دستگاه است.در این مرحله کد به زبان ماشین تبدیل می‌شود.

در مرحله اخر ، تمام پیوند تماس‌های عملکردی با تعاریف آنها انجام می‌شود.



مراحل اجرای کد c



۴- ۱۰ زبان برنامه نویسی را بنویسید و بحث‌کنید که کدام یک از complier، interpreter یا هر دو استفاده می کنند.

در سوال ۲ در مورد کامپایلر و مفسر توضیحاتی ارائه شده است و باید به ان توضیحات این نکته را نیز افزود؛ کامپایلرهای امروزی انواع مختلفی دارند که زبان‌های برنامه نویسی با توجه به حافظه مصرفی، زمان لازم برای اجرا، و همچنین توانایی پلت فرم‌های خاص در اجرای کدها و غیره از هر کدام از ان‌ها بهره می‌برند. در ادامه به معرفی انها می‌پردازیم.

کامپایلر AOT :

در این کامپایلرها تمام کدها قبل از رسیدن به سیستم عاملی که ان‌ها را اجرا می‌کند، ترجمه می‌شوند که به این کامپایل مقدماتی ahead-of-time compilation (AOT) گفته می‌شود. بسیاری از زبان‌های برنامه نویسی از این نوع کامپایلر استفاده می‌کنند که می‌توان به C, C++, Swift, Assembly Language اشاره کرد.

کامپایلر JIT :

این نوع کامپایلر ترکیبی از کامپایلر AOT و interpreted است. کامپایلر JIT به جای تبدیل کد به دستورات قابل اجرا توسط یک پلت فرم خاص، آن را به bytecode تبدیل می‌کند. bytecode از پلت‌فرم‌ها مستقل است و می‌تواند روی هر پلت‌فرمی که از ان زبان برنامه نویسی پشتیبانی می‌کند، ارسال و اجرا شود. در این حالت برنامه در دو مرحله ترجمه می‌شود. کامپایلر JIT در مرحله دوم بایت‌کدها را ترجمه می‌کند. تا زمانی که تبدیل بایت کدها برای پلت فرم قابل فهم باشد، برنامه اجرا می‌شود.

زبان هایی که از این نوع کامپایلر استفاده می‌کنند: Java, C#

Interpreted code، تمام کدها را به صورت خط به خط در برنامه ترجمه و اجرا می‌کند؛ یعنی توسط یک ماشین مجازی در زمان اجرای کد ان را به زبان ماشین ترجمه می‌کند. زبان‌هایی که از مفسر استفاده می‌کنند، شامل: JavaScript, Python, BASIC, MATLAB, PHP .

با توجه به توضیحات فوق می‌توان گفت که:

زبان‌های مفسری: JavaScript, Python, BASIC, MATLAB, PHP

زبان‌های کامپایلری:,C++, Swift ,Assembly Language, C

زبان هایی که از هردو استفاده می‌کنند: Java, C#

۵- کدهای زیر را به کد سه آدرس تبدیل کنید.

a= 110 + (b*c);

t1 = b * c

t2 = int to real (110)

a = t1 + t2

b=2*c+2*b;

t1= int to real (2)

t2= t1 * c

t3 = int to real (2)

t4 = t3 * b

b = t2 + t4

c=42+2*b+2*c;

t1=int to real (2)

t2 = t1 * b

t3 = int to real (2)

t4 = t3 * c

t5 = int to real (42)

c = t5 + t4 + t2

۶- ابزارهایی برای "Lexical Analyzer Generator" وجود دارد. یکی از این ابزارها "Flex" است. در مورد این ابزار (نرم افزار) در اینترنت جستجو کنید و یک سند ۱ صفحه ای را برای این ابزار گزارش دهید.

همان طور که در سوال ۲ گفته شد، تحلیلگر لغوی فاز اول کامپایل کردن یک برنامه است. در این مرحله متن ورودی، کاراکتر به کاراکتر خوانده و توکن‌بندی می‌شود. (توکن‌ها (Token) در زبان‌های برنامه نویسی، شناسه-هایی هستند که معنای خاصی دارند). توکن‌ها در جدول نمادها ذخیره می‌شوند تا در مراحل بعدی مورد استفاده قرار گیرند. (جدول نمادها (Symbol Table)، یک نوع ساختمان داده است که برای ذخیره کردن توکن‌ها در مراحل کامپایل مورد استفاده قرار می‌گیرد).

برنامه‌ای که تجزیه و تحلیل واژگانی را انجام می‌دهد، لغت نامه (Lexer) ، نشانه‌گذاری (tokenizer) یا اسکنر(scanner) خوانده می‌شود؛ اگرچه اسکنر نیز اصطلاحی برای اولین مرحله از یک مترجم است. یک لغت نامه معمولاً با یک تجزیه‌کننده ترکیب می‌شود ، که در کنار یکدیگر نحو زبان‌های برنامه نویسی ، صفحات وب و غیره را تجزیه و تحلیل می‌کنند.

واژه (lexeme) : تعریف واژه در علم کامپیوتر با تعریفی که در زبان شناسی استفاده و شناخته شده است، متفاوت است. واژه در اینجا به رشته‌ایی از کاراکترها گفته می‌شود که یک واحد نحوی را تشکیل می‌دهند.

توکن سازی(Tokenization) : تحلیلگر واژه‌ای، lexems ها را پردازش می‌کند و آن‌ها را با توجه به کاربردشان دسته‌بندی می‌کند و به آن‌ها معنا می‌دهد؛ این انتساب معنا tokenization نامیده می‌شود. توکن‌ها معمولاً به وسیله محتوای کاراکتر یا محتویات درون یه جریان داده دسته‌بندی می‌شوند. این دسته‌ها به وسیله قوانین lexer تعریف می‌شوند.

Lexers و تجزیه کنندگان (parsers) بیشتر برای کامپایلرها مورد استفاده قرار می‌گیرند ، اما می‌توانند برای سایر ابزارهای زبان رایانه‌ای ، مانند برنامه‌نویسی یا لاینر نیز استفاده شوند.

Lexing را می توان به دو مرحله تقسیم کرد:

- **اسکن:** که رشته ورودی را به واحدهای نحوی موسوم به lexemes تقسیم می‌کند و آنها را به کلاس های نشانه گذاری طبقه بندی می‌کند.
- **ارزیابی:** lexemes را به مقادیر پردازش شده تبدیل می کند.

Lexical Analysis: این اولین فرایندی است که کامپایلر جریانی از کاراکترها را می خواند (معمولاً از یک فایل کد منبع) و یک جریان از نشانه های واژگانی تولید می‌کند. به عنوان مثال کد C ++ ؛ معمولاً فقط تعداد کمی نشانه برای یک زبان برنامه نویسی وجود دارد: ثابت (integer, double, char, string, etc) ، اپراتورها (arithmetic, relational, logical) ، نگارشی و کلمات محفوظ.

آنالایزر واژگانی یک برنامه منبع را به عنوان ورودی در نظر می‌گیرد و یک جریان از نشانه ها را به عنوان خروجی تولید می‌کند.

Syntactical Analysis:

این خروجی از Lexical Analyzer به قسمت Syntactical Analyzer کامپایلر می‌رود. با استفاده از این قوانین دستور زبان بررسی می‌کند که ورودی صحیح است یا خیر؛ در صورت اشتباه بودن کامپایلر هشدار صادر می‌کند و نباید هشدارها را نادیده گرفت و باید ان‌ها را برطرف کرد.

تولید کد ماشین:

پس از طی موفقیت امیز مراحل Lexical Analysis و Syntactical Analysis ، مرحله اخر تولید کد ماشین است؛ این فرایند پیچیده است، به خصوص با توجه به cpu های مدرن امروزی .

Flex and lexical analysis:

در حوزه کامپایلرها ، ابزارهای بسیاری برای تبدیل پرونده‌های متنی به برنامه ها وجود دارد؛ بخش اول آن فرآیند اغلب lexical analysis نامیده می شود ، به خصوص برای زبانهایی مانند C .

یک ابزار خوب برای ایجاد آنالایزر واژگانی ، flex است. یک پرونده مشخصات را می‌گیرد و یک آنالایزر ایجاد می کند ، معمولاً lex.yy.c .

Flex(The Fast Lexical Analyzer):

یک نرم افزار ازاد برای تولید آنالایزر واژگانی (اسکنر یا واژگان) است که توسط ورن پاکسون در سال ۱۹۸۷ در C نوشته شده است؛ همراه با ژنراتور تجزیه کننده Berkeley Yacc یا GNU Bison استفاده می‌شود. Flex و Bison هر دو از Lex و Yacc انعطاف پذیرتر هستند و کد سریع تری تولید می کنند. Flex یک اجرای رایگان (اما غیر GNU) از برنامه اصلی UNIX lex است.

Flex یک فایل منبع C به نام "lex.yy.c" تولید می‌کند ، که تابع yylex() را که تابع اصلی ان محسوب می‌شود، تعریف می‌کند.

پرونده "lex.yy.c" قابل کامپایل است؛ این فایل می‌تواند توسط کاربر کامپایل شود تا یک فایل اجرایی به وجود آید که همان برنامه تحلیل‌گر واژگانی است. وقتی این برنامه اجرا شد، متنی را از ورودی می‌گیرد و سعی می‌کند که الگوهای مورد نظر کاربر را در این متن تشخیص دهد. وقتی که یکی از این الگوها پیدا شد، یک کد به زبان سی (که از قبل تعریف شده) اجرا می‌شود.

ساختار برنامه:

۱. بخش تعریف: شامل اعلام متغیرها ، تعاریف منظم ، ثابت‌های آشکار است. در این بخش ، متن در براکت ها به صورت زیر محصور شده است. هر آنچه در این براکت ها نوشته شود، مستقیماً در پرونده lex.yy.c کپی می‌شود.

```
%{
   // Definitions
%}
```

۲ .بخش قوانین: شامل یک سری قوانین در فرم است: الگوی عمل باید ناخواسته باشد و عمل باید در براکت شروع شود. قسمت قانون مانند زیر است.

```
%%
    Pattern action
%%
```

۳ .بخش کد کاربر: این بخش شامل عبارات C و توابع اضافی است. ما همچنین می توانیم این توابع را به طور جداگانه کامپایل کرده و با آنالایزر واژگانی بارگذاری کنیم.

ساختار اصلی برنامه:

```
%{
  // Definitions
%}

%%

    Rules

%%
```

مقایسه:

Lex ، جدول ورودی عبارات و اعمال منظم کاربر را به تابعی به نام yylexn () تبدیل می‌کند. تابع yylex() وقتی در برنامه میزبان زبان منبع گنجانیده می‌شود ، هر عمل را با شناسایی الگوی مرتبط انجام می‌دهد. TFlex قادر به تولید خروجی خود به عنوان کد منبع C ، C++ یا FORTRAN است. در هر دو حالت ، عملکرد yylex() شامل روال‌های بسیار کارآمد تطبیق رشته ای Aho و Corasick است.

عملکرد yylex () تولید شده توسط Lex معمولاً نیاز به زمان متناسب با طول جریان ورودی دارد. این عملکرد با توجه به ورودی و مستقل از تعداد قوانین خطی است. با افزایش تعداد و پیچیدگی قوانین ، yylex() فقط سایز ان افزایش می‌یابد. وقتی قوانین به اسکن گسترده ورودی ها نیاز دارد، سرعت باید کاهش یابد.