اصول طراحی کامپایلر

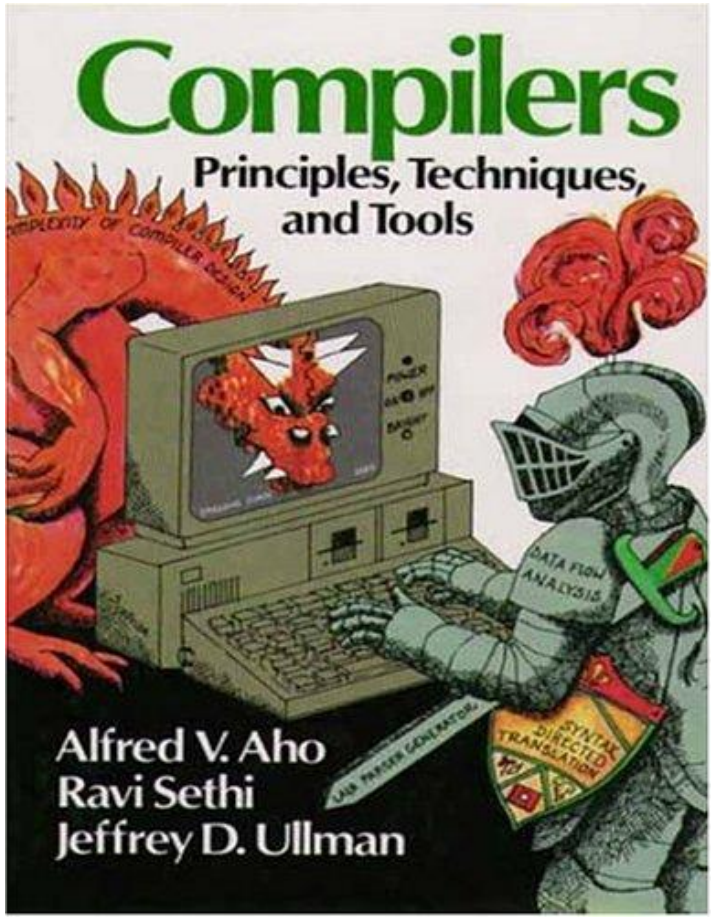
مریم حبیبی

منبع

- **عنوان منبع:** کامپایلرها
  اصول، تکنیک ها و ابزارها

- **منبع اصلی:**

Compilers:
Principles, Techniques, and Tools
Alfred Aho,
Ravi Sethi,
Jeffrey D. Ullman



- **بارم بندی:**
  - ۵ نمره پروژه شامل کدنویسی یکی از بخش های یک کامپایلر
    - زمان ثبت موضوع تا پایان آخرین جلسه اسفند ماه
    - زمان ارسال فایل ارائه، تایید محتوا و ثبت تاریخ ارائه تا پایان آخرین جلسه فروردین ماه
    - ارائه شفاهی در کلاس تا پایان آخرین جلسه خرداد ماه
  - ۵ نمره فعالیت کلاسی شامل حضور و توجه، پاسخ صحیح(!) به پرسش های شفاهی و کوییزها
  - ۱۰ نمره امتحان پایان ترم

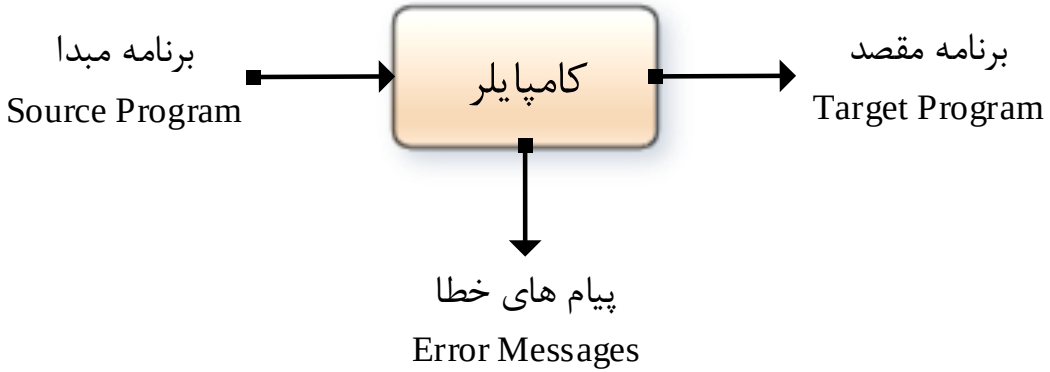
  - جمع نمرات: ۲۰

فصل ۱

آشنایی با کامپایلرها و روش های مختلف کامپایل

۱-۱ تعریف کامپایلر

۱. ترجمه برنامه نوشته شده در یک زبان مبدا (Source language) به برنامه معادلش در یک زبان مقصد (Target language)
۲. گزارش دادن خطاهای برنامه مبدا به کاربر



۱-۱ طبقه بندی کامپایلرها بر اساس ساخت و عملیات

- یک گذره
- چند گذره
- بارگزاری و اجرا
- اشکال زدایی
- بهینه سازی

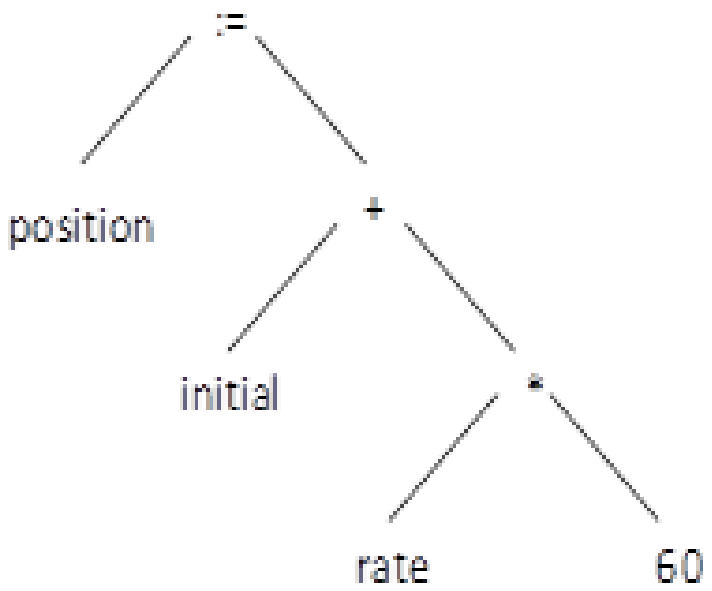
۱-۲ مدل تجزیه و تحلیل و ترکیب در کامپایلر

بخش تحلیل

- تجزیه برنامه مبدا به اجزای تشکیل دهنده اش
- تولید کد میانی از برنامه مبدا

بخش ترکیب

- تبدیل کد میانی به برنامه مقصد در زبان دیگر
- نیاز به بیشترین روش های خاص (مانند درخت ساختار دستور)



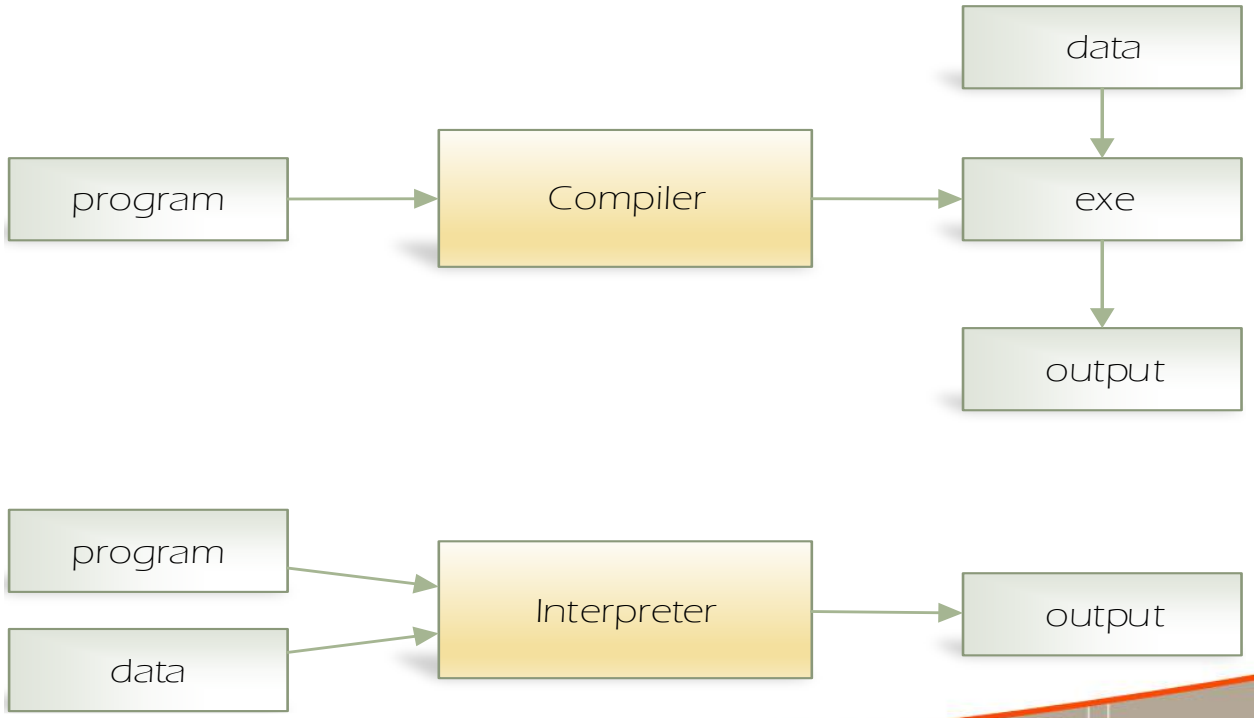
درخت ساختار دستور position : = initial + rate ⁎۶۰

۱-۲ نمونه ای از برنامه های تحلیل کننده

- **ویرایشگرهای ساخت یافته:** مانند ساختار سلسله مراتبی برنامه، تشخیص کلمات کلیدی، جا افتادن do، جا افتادن پرانتز و ...
- **چاپگرهای زیبا pretty printer**: مانند چاپ توضیحات داخل برنامه با یک قلم خاص، چاپ دستورات برنامه با مقداری تورفتگی متناسب با عمق متداخل بودنشان در ساختار سلسله مراتبی دستورها
- **بررسی کننده های ایستا:** برنامه را می خواند، آن را تحلیل می کند و بدون اجرای برنامه، سعی می کند خطاهای برنامه را پیدا کند. بخش تحلیل بررسی کننده شبیه کامپایلرهای بهینه ساز است.
- **مفسرها:** (توضیح در اسلاید بعد)
- **شکل دهنده های متن:** ورودی آن دنباله ای از کاراکترهای متنی است که باید حروف‌چینی شود. اما بعضی از آن‌ها حاوی دستورهایی برای نمایش پاراگراف‌ها، شکل‌ها یا ساختارهای ریاضی مانند اندیس یا توان هستند.
- **کامپایلرهای سیلیسیومی:** زبان مبدا آن‌ها شبیه یا دقیقاً مانند یک زبان برنامه‌نویسی مرسوم است. ولی برای متغیرهای این زبان، فضایی در حافظه اختصاص داده نمی شود بلکه سیگنال‌های منطقی (0 یا 1) یا مجموعه‌ای از سیگنال‌ها در مدارهای حافظه هستند که خروجی آن، طراحی مدار به یک زبان مناسب است.
- **مفسرهای پرس و جو:** گزاره ورودی را که شامل عملگرهای رابطه‌ای و بولین است به دستورهایی برای جستجوی رکوردها در بانک اطلاعاتی ترجمه می کند.

۱-۲ نمونه ای از برنامه های تحلیل کننده

- **مفسرها:** یکی از ابزارهای نرم افزاری مهم که بر روی برنامه ورودی عملیات تحلیل را انجام می‌دهند، مفسرها (Interpreters) هستند. در این ابزار، به جای این که برنامه هدف به صورت ترجمه شده تولید شود، مفسر، عملیات موجود در برنامه مبدا را یک به یک اجرا می‌کند و خروجی تولید می‌نماید. مثلا برای یک دستور جایگزینی ممکن است مفسر یک درخت ساختار دستور شبیه درخت شکل ۱-۲ بسازد و آنگاه با حرکت (پیمایش) در درخت، عملیات موجود در گره‌ها را اجرا کند. تفاوت بین کامپایلر و مفسر در شکل ۱-۳ واضح‌تر است. به دلیل این که داده ورودی در مفسر از قبل دریافت می‌گردد، می گوییم عملیات مفسر on-line و با این توضیح عملیات کامپایلر off-line می باشد.



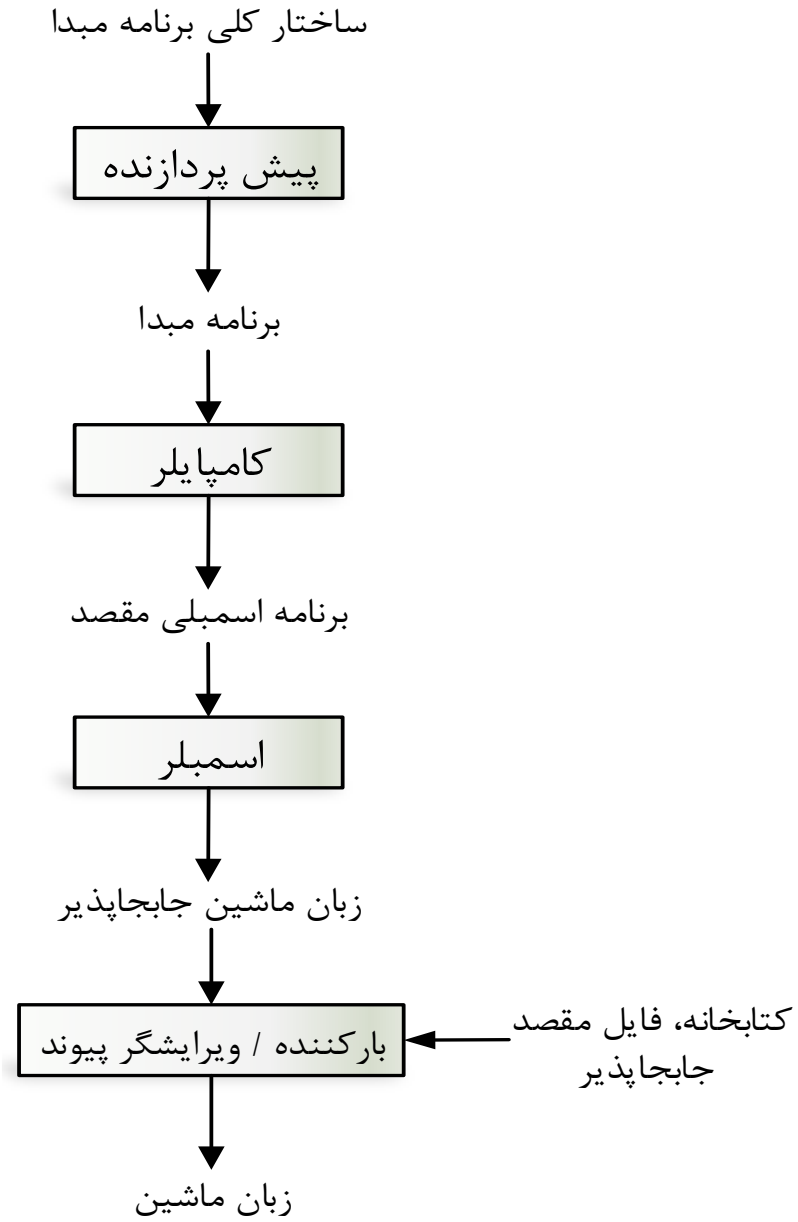
۱-۳ تحلیل برنامه مبدا

- **تحلیل خطی:** که در آن، دنباله کاراکترهای تشکیل دهنده برنامه مبدا از چپ به راست خوانده شده و به نشانه (token) هایی تقسیم می‌شوند که دنباله‌ای از چند کاراکتر بوده و دارای معنی مشخصی هستند.
- **تحلیل سلسله مراتبی:** که در آن، کاراکترها و یا نشانه‌هایی که دارای معنی مشخصی هستند، به صورت سلسله مراتبی در مجموعه‌های تو در تو گروه بندی می‌شوند.
- **تحلیل معنایی:** که در آن بررسی‌های معنایی انجام می‌شود تا اطمینان حاصل شود که اجزای مختلف یک برنامه به طرز معناداری در کنار یکدیگر قرار گرفته‌اند یا خیر.

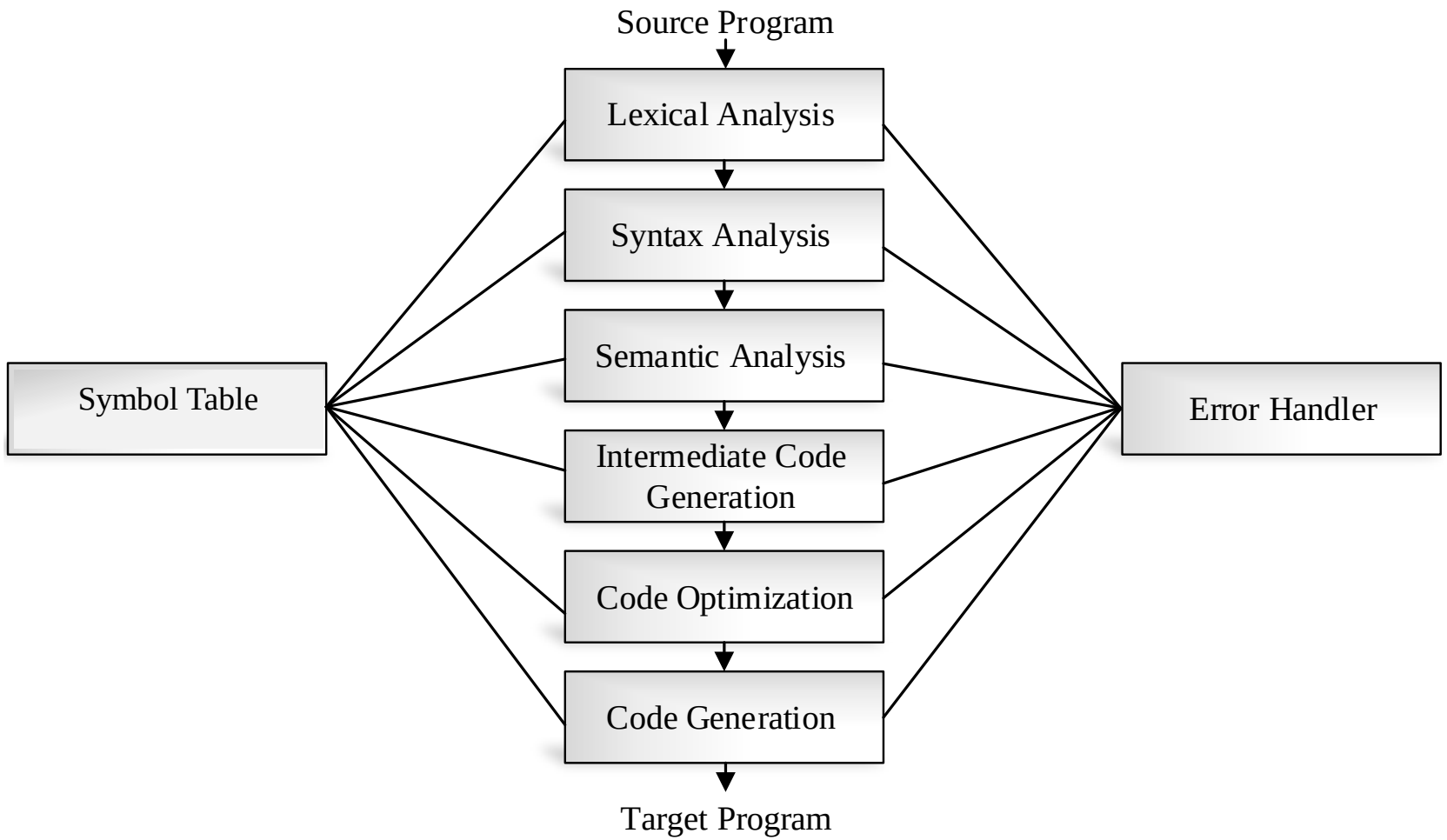
نوع تحلیل	عملکرد
تحلیل خطی (تحلیل لغوی یا اسکن)	تشخیص نشانه ها یا توکن ها
تحلیل سلسله مراتبی (تحلیل نحوی یا تجزیه)	گروه بندی توکن های برنامه مبدا در سلسله مراتب
تحلیل معنایی	بررسی خطاهای معنایی

۱-۴ اجزای یک سیستم پردازش زبان

- پیش پردازنده
  - جمع آوری ماژول های برنامه مبدا موجود در فایلهای جداگانه
  - تبدیل بخش های خلاصه شده بنام درشت دستورات به احکام زبان مبدا
- کامپایلر
- اسمبلر
- بارکننده / ویرایشگر پیوند



۱-۵ مراحل یا فازهای کامپایل



۱- تحلیل گر لغوی (واژه ای)

برنامه ورودی را کاراکتر به کاراکتر می خواند (scan).

به دنباله‌ای از نشانه‌ها یا Tokenها (کلمات کلیدی، عملگر، جداکننده، ثوابت و شناسه) که دارای معنی هستند، تبدیل می کند.

۱- تحلیل گر لغوی

- **مثال:** در تحلیل واژه‌ای، کاراکترهای تشکیل دهنده دستور جایگزینی

$$position := initial + rate * 60$$

به نشانه‌های زیر دسته بندی می‌شوند:

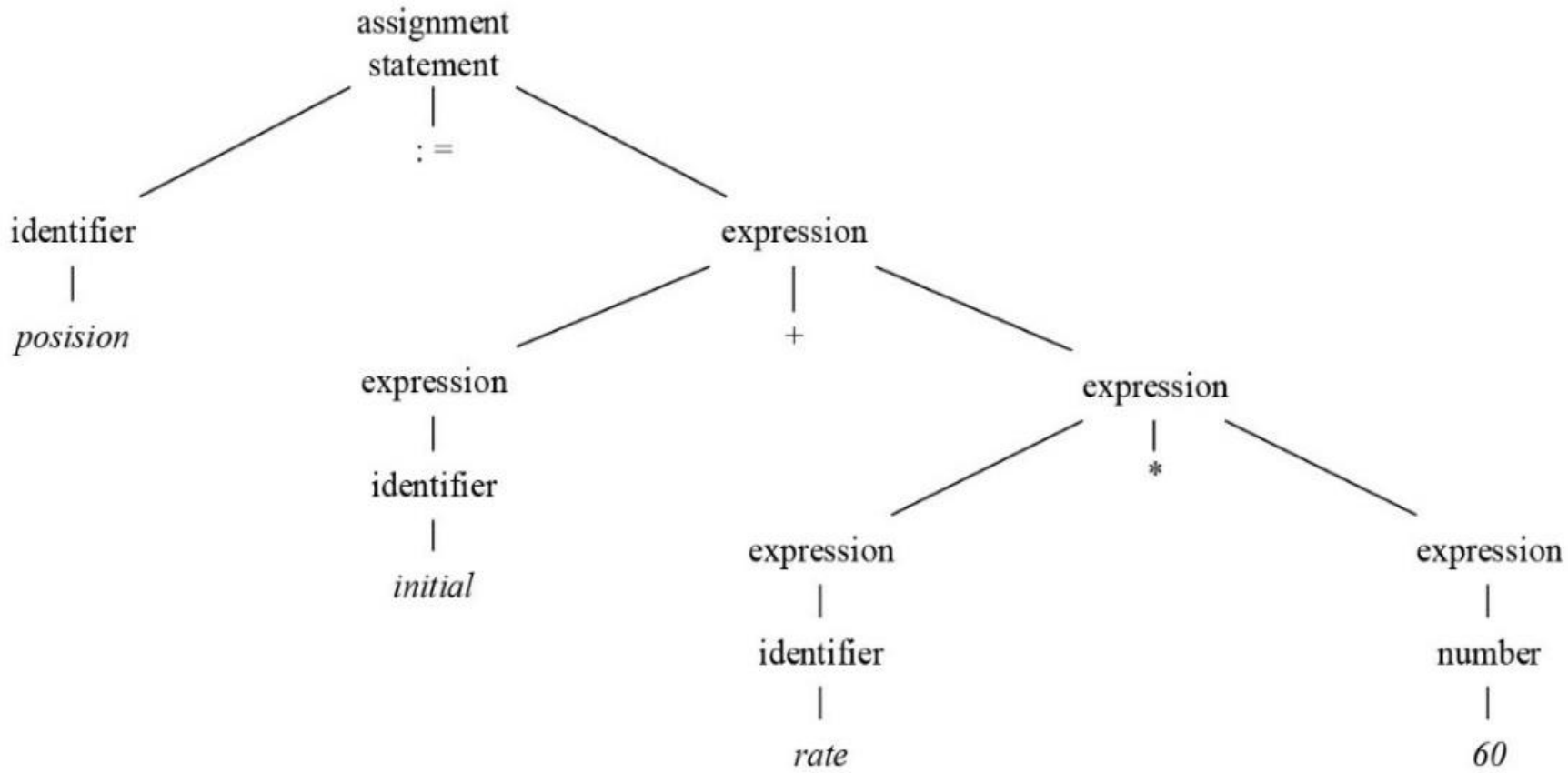
- شناسه position
- علامت جایگزینی :=
- شناسه initial
- علامت جمع +
- شناسه rate
- علامت *
- عدد 60

۲- تحلیل گر نحوی (دستوری)

نشانه‌های تولید شده (Tokens) در مرحله قبل در عبارت‌های گرامری دسته بندی می‌شوند.

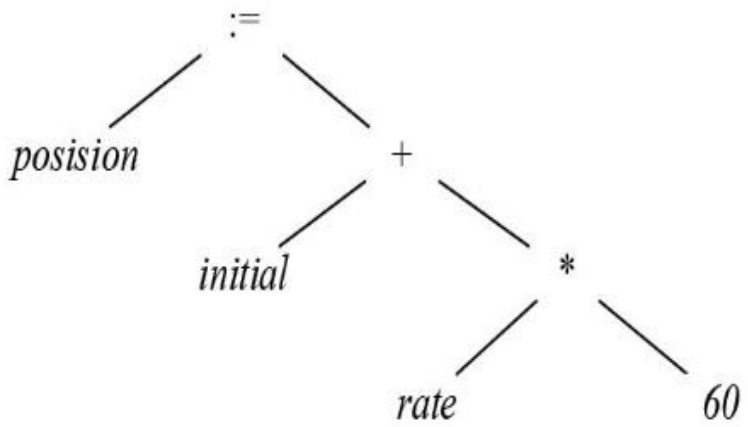
معمولاً عبارت‌های گرامری برنامه مبدا با درخت تجزیه (درخت pars) نشان داده می‌شوند

۲- تحلیل گر نحوی (دستوری)



ساختار سلسله مراتبی درخت تجزیه

۲- تحلیل گر نحوی (دستوری)

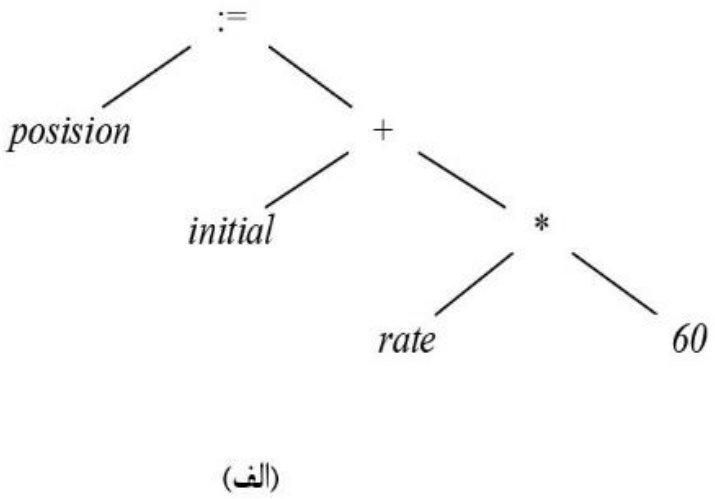


۳- تحلیل گر معنایی

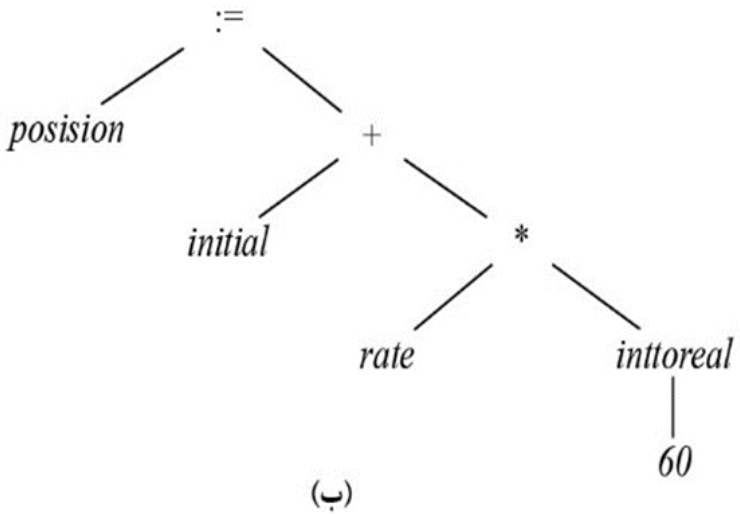
بررسی برنامه مبدا برای یافتن خطاهای معنایی

جمع آوری اطلاعات مربوط به نوع داده ها

۳- تحلیل گر معنایی



(الف)



(ب)

الف) درخت ساختار دستور، ب) تحلیل معنایی

۴- تولید کد میانی

خواندن برنامه ورودی

یک نمایش میانی صریح از برنامه مبدا به صورت برنامه ای برای یک ماشین انتزاعی تولید می کنند.

۴- تولید کد میانی

- در انتخاب زبان میانی باید موارد زیر در نظر گرفته شوند:
  ۱- تولید و بهینه سازی کد میانی باید ساده باشد.
  ۲- ترجمه آن به برنامه مقصد نیز به راحتی صورت پذیرد.
- نمایش میانی به شکل های مختلفی می تواند وجود داشته باشد که یکی از آنها "کد سه آدرسی" است. این کد شبیه زبان اسمبلی ماشینی است که تمام خانه های حافظه آن، ثبات(register) هستند. کد سه آدرسه از تعدادی دستور تشکیل شده که هر دستور حداکثر سه عملوند دارد.
- مثال: دستور position : = initial + rate ❊ ۶۰ با کد سه آدرسه بدین صورت بیان می شود:

```
temp1 := inttoreal (60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

- این کد میانی دارای سه ویژگی زیر است:
  ۱- هر دستور سه آدرسه علاوه بر عملگر جایگزینی، حداکثر یک عملگر دارد. بنابراین کامپایلر قبل از تولید این کدها باید در مورد اولویت عملگرها تصمیم بگیرد.
  ۲- کامپایلر باید برای ذخیره مقدار محاسبه شده توسط هر دستور، یک نام موقت تولید کند.
  ۳- برخی از دستورات سه آدرسه مانند دستورات اول و آخر، کمتر از سه عملوند دارند.

۵- بهینه سازی کد

بهینه کردن کد میانی (حذف متغیرهای میانی غیر ضروری)

تولید کد ماشین سریع تر برای اجرا

۵- بهینه سازی کد

- مثال: در مثال قبل راه بهتر انجام همان محاسبات به صورت زیر است:

```
temp1 := id3 * 60.0
id1 := id2 * temp1
```

- این الگوریتم ساده با اعمال دو تغییر ایجاد شده و کاملا درست است. اول این که عمل inttoreal حذف شده و از نمایش اعشاری ۶۰.۰ به جای نمایش صحیح ۶۰ استفاده گردیده است. دوم این که در مثال ۱-۳ از temp3 فقط یک بار استفاده شده بنابراین می توان idl را جایگزین آن کرد تا از تعداد دستورات کاهید.

۶- تولید کد نهایی

تبدیل کد میانی بهینه به کد ماشین جابجاپذیر یا کد اسمبلی

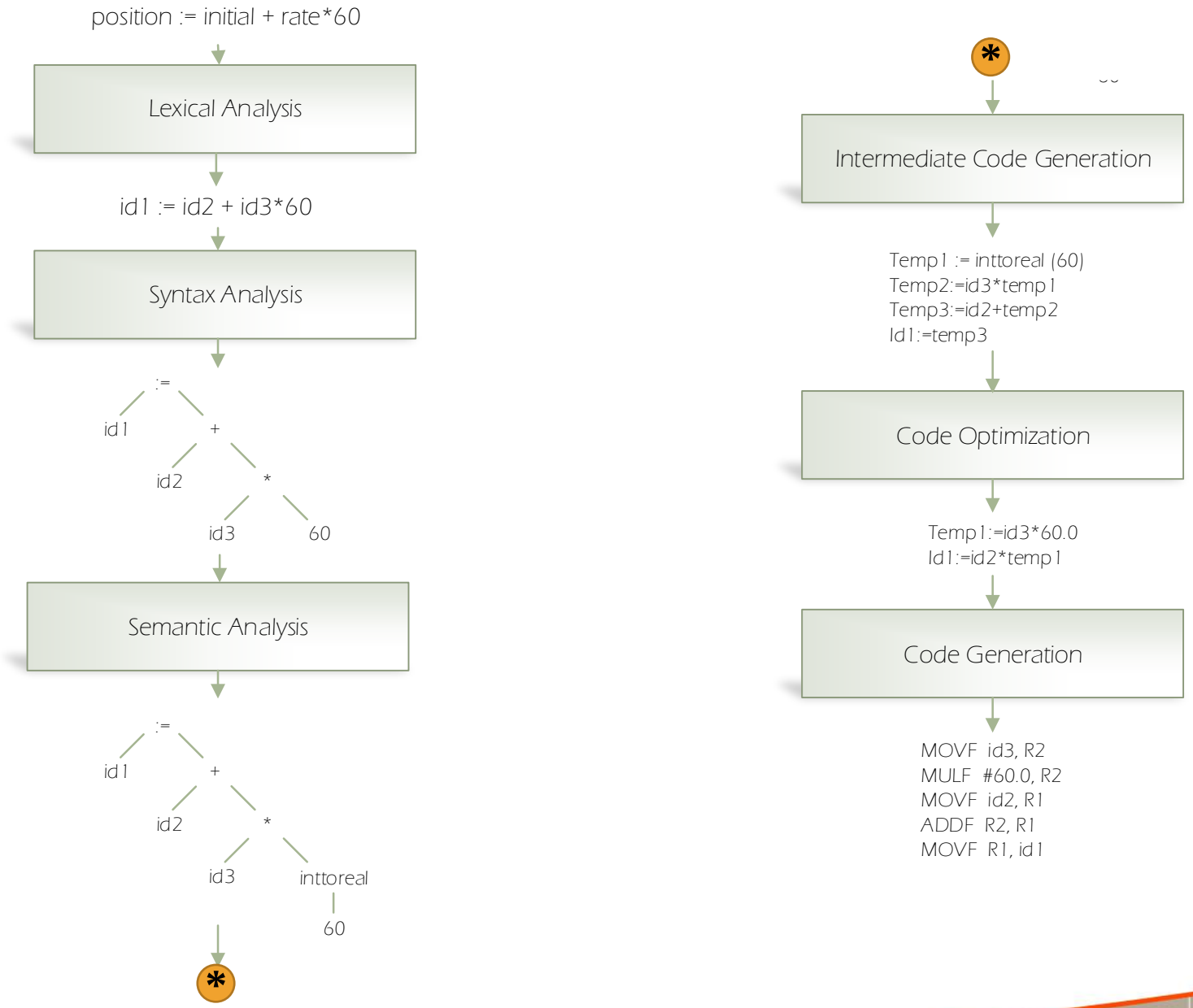
تعیین مکانهای حافظه برای متغیرهای برنامه

انتساب متغیرها به ثبات ها (رجیسترها)

۶- تولید کد نهایی

- **مثال**: ترجمه کد مثال قبل به صورت زیر است:

MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
M0VF R1, id1


position := initial + rate*60
Lexical Analysis
id1 := id2 + id3*60
Syntax Analysis
:=
id1
+
id2
*
id3
60
Semantic Analysis
:=
id1
+
id2
*
id3
inttoreal
60
Intermediate Code Generation
Temp1 := inttoreal (60)
Temp2:=id3*temp1
Temp3:=id2+temp2
Id1:=temp3
Code Optimization
Temp1:=id3*60.0
Id1:=id2*temp1
Code Generation
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1

۷- مدیر جدول نمادها

- **تعریف:**
  - جدول نمادها، ساختمان داده‌ای است که شامل رکوردی برای هر شناسه می باشد که فیلدهای آن، خصیصه‌های آن شناسه هستند.
  - فیلدها شامل اطلاعات مربوط به حافظه اختصاص یافته به هر شناسه، نوع آن، دامنه فعالیت آن بخشی از برنامه که شناسه در آن معتبر است و در مورد نام زیر برنامه‌ها (زیرروال‌ها) مواردی از قبیل تعداد و نوع آرگمان‌های آن، روش انتقال هر آرگمان (مثلا از طریق ارجاع با نام یا ارجاع با آدرس) و نوع مقدار بازگشتی در صورت وجود، می‌باشند.

- **هدف:**
  - با استفاده از این ساختمان داده به سرعت می توان رکورد هر شناسه را پیدا کرد و داده ها را در آن رکورد به سرعت ذخیره یا از آن بازیابی نمود.

۸ – خطا پرداز

- **وظیفه:**
  - هر فاز از کامپایل ممکن است با خطاهای برنامه روبرو شود. بعد از کشف خطا، هر فاز باید به گونه‌ای با آن برخورد کند که امکان ادامه کامپایل و کشف خطاهای بیشتر در برنامه مبدا میسر باشد.
  - کامپایلری که با یافتن اولین خطا متوقف شود، کامپایلر خوب و کارآمدی نیست.
- **بیشترین خطاها:**
  - فازهای تحلیل ساختار دستور و معنایی معمولا بخش عمده‌ای از خطاهایی را که توسط کامپایلر کشف می‌شود اداره می‌کنند.
  - فاز واژه‌ای می تواند خطاهایی را کشف کند که در آن، رشته کاراکترهای ورودی، مطابق الگوی هیچ یک از نشانه (token) های زبان نباشد.
  - خطایی که در آن، دنباله‌ای از نشانه‌ها، قوانین ساختار دستوری (syntax) زبان را نقض می‌کنند در فاز تحلیل ساختار دستور تشخیص داده می‌شود.
  - در زمان تحلیل معنایی، کامپایلر، ساختارهایی را کشف می‌کند که از نظر ساختار دستوری صحیح هستند اما در رابطه با عملی که باید انجام دهند بی معنی هستند. برای مثال جمع دو شناسه که یکی از آنها نام آرایه و دیگری نام رویه است عملی بی‌معنی می‌باشد و از این رو پیغام خطا داده می‌شود.

۱-۷- دسته بندی فازها

- بحث فازهای کامپایلر به ساماندهی منطقی کامپایلر مربوط می‌شد.
- در یک پیاده سازی از کامپایلر، اغلب اوقات، فعالیت‌های مربوط به بیش از یک فاز، با هم در یک دسته قرار می‌گیرند.
  - Front-end و Back-end
  - گذرها

۱ – ۷ – ۱ – front-end و back-end کامپایلر

- **بخش ابتدایی یا جلوبندی یا Front-end:**
  - شامل فازهایی است که به زبان مبدا وابسته هستند و عمدتا از زبان مقصد مستقل می‌باشند.
  - این بخش‌ها معمولا شامل تحلیل واژه‌ای و ساختار دستور، ایجاد جدول نمادها، تحلیل معنایی و تولید کد میانی است. علاوه بر این، قسمتی از بهینه سازی کد توسط بخش ابتدایی انجام می‌شود و نیز بررسی خطاهای هر یک از این فازها در همین بخش ابتدایی انجام می‌شود.
- **بخش انتهایی یا عقب بندی یا Back-end:**
  - شامل آن قسمت‌هایی از کامپایلر است که وابسته به ماشین مقصد می‌باشند و معمولا به زبان مبدا وابسته نیستند و فقط به زبان میانی وابسته‌اند.
  - در بخش انتهایی، قسمتی از فاز بهینه سازی کد و تولید کد و نیز عملیات لازم برای بررسی و گزارش خطاها و عملیات روی جدول نمادها وجود دارد.

۱ – ۷– ۲ – گذرها

- چند فاز مختلف کامپایلر معمولا به صورت یک گذر که شامل خواندن فایل ورودی و نوشتن در فایل خروجی است پیاده سازی می‌شوند. در یک گذر این فازها به صورت سلسله مراتبی فعال می‌شوند. برای نمونه، تحلیل لغوی، تحلیل ساختار دستور، تحلیل معنایی و تولید کد میانی ممکن است در یک گذر قرار داده شوند.
- به طور کلی به علت زمان بر بودن عملیات خواندن و نوشتن فایل های میانی، علاقمندیم تا تعداد فازهای کامپایلر را کاهش دهیم. از طرف دیگر اگر چند فاز را در یک گذر قرار دهیم احتمالا مجبور به نگهداری کل برنامه در حافظه می‌شویم زیرا این امکان وجود دارد که ترتیب نیاز به اطلاعات در یک فاز با ترتیب تولید آن اطلاعات در فاز قبل متفاوت باشد.
- در اسمبلر دو گذره، در گذر اول، تمام شناسه‌هایی که معرف خانه‌های حافظه هستند و نیز آدرس آنها شناسایی می‌گردد و در گذر دوم، آدرس‌ها جایگزین شناسه‌ها می‌شوند.

پایان فصل ۱

آشنایی با کامپایلرها و روش های مختلف کامپایل