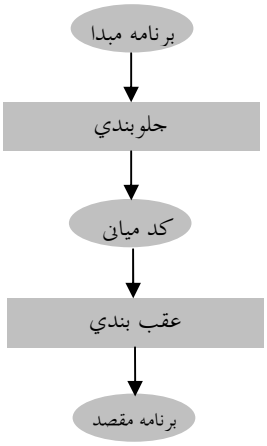
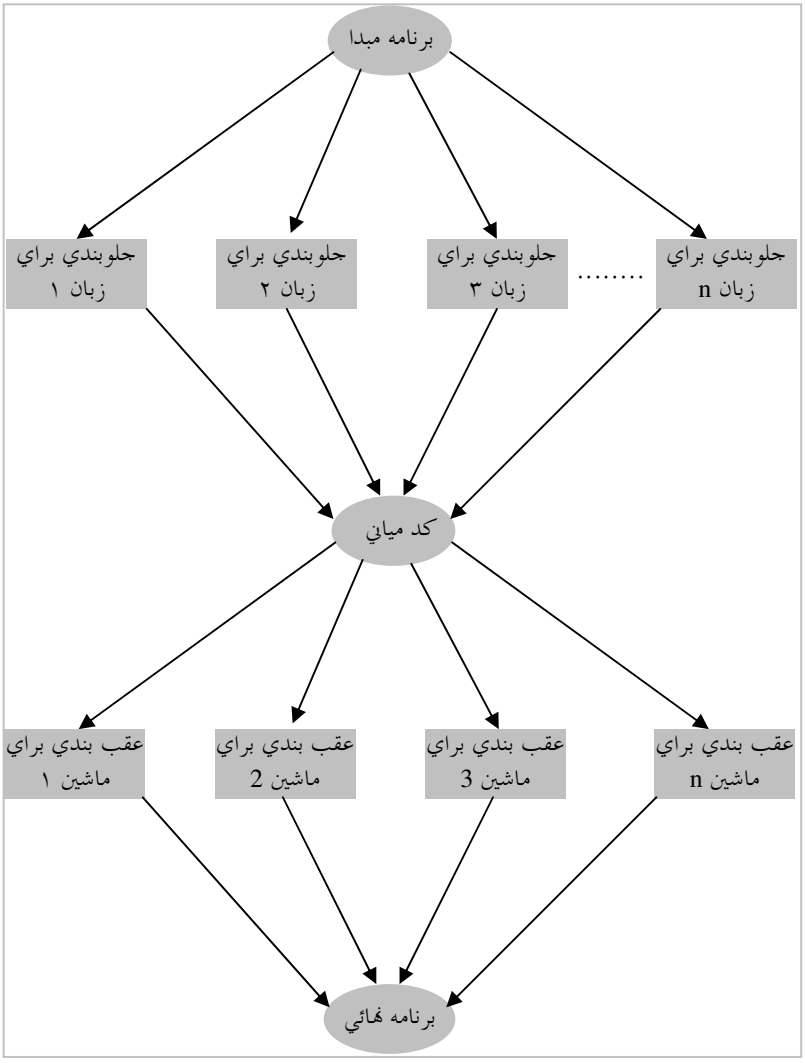
جلسه دوم

جلوبندی (Front End) و عقب بندی(Back End) کامپایلر

می خواهیم برنامه ای را به زبان C روی انواع مختلف کامپیوتر(مانند Vax، Mainfram، IBM) اجرا کنیم در این صورت برای هر نوع کامپیوتر، باید یک کامپایلر جداگانه بسازیم. اگر n تعداد زبان های برنامه سازی(مانند C و پاسکال و...) و K تعداد انواع مختلف کامپیوتر ها باشد در این صورت به nk کامپایلر نیاز است. ایجاد این تعداد کامپایلر بسیار زمانبر و پر هزینه است. برای حل این مشکل از تقسیم کامپایلر به جلوبندی و عقب بندی استفاده می کنیم. به این شکل که یک زبان میانی در نظر می گیریم در ابتدا برنامه مبدا را به این زبان میانی ترجمه کرده سپس از زبان میانی به زبان مقصد ترجمه می کنیم. **بخشی از کامپایلر که وظیفه ترجمه برنامه مبدا به برنامه زبان میانی را بر عهده دارد و وابسته به زبان ماشین نیست را جلوبندی و بخشی از کامپایلر که وظیفه ترجمه برنامه از زبان میانی را به زبان مقصد را بر عهده دارد و وابسته به زبان ماشین است را عقب بندی کامپایلر می گوئیم(شکل زیر).**



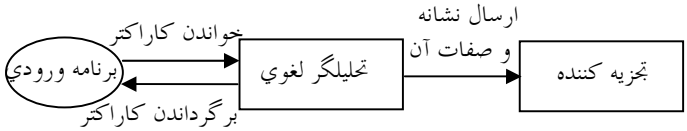
با استفاده از این روش برای n زبان مبدا و k کامپیوتر مختلف به n جلوبندی و k عقب بندی نیاز است که در مجموع n+k برنامه(کامپایلر) احتیاج است(شکل زیر این موضوع را نشان می دهد).



تقسیم بندی کامپایلر به عقب بندی و جلو بندی چه مزایائی دارد؟

- سادگی طراحی
- استقلال جلوبندی از زبان مقصد
- استقلال عقب بندگی از زبان مبدا
- کاهش پیچیدگی
- افزایش قابلیت استفاده مجدد
- افزایش سرعت تولید کامپایلر برای سخت افزار جدید و زبان های جدید.

تحلیگر لغوی(scanner) :واسط بین کامپایلر و برنامه مبدا می باشد و مهمترین وظیفه آن خواندن برنامه ورودی به صورت کـاراکتر بـه کاراکتر و تشخیص نشانه ها یا **token** ها میباشد(شکل زیر نحوه قرار گرفتن تحلیلگر لغوی بین ورودی و تجزیه کننده را نشان می دهد).



Scanner میتو اند به صورت هم روال یا زیرروال با **parser** پیاده سازی شود، **Scanner** به محض درخواست **token** بعدی از **parser** ،توکن بعدی را به پارسر می دهد.

وظایف دیگر scanner

- حذف فضا های خالی و توضیحات(comment)
- گزارش خطا ها

از آنجا که **scaner** برنامه را به صورت کاراکتر به کاراکتر می خواند، می تواند تعداد کاراکتر های هر خط و به تبع آن شماره خطوطی را که توکن ها در آن قرار دارد را نیز مشخص کند. بنابر این هنگام اعلام خطا شماره خطی را که خطا رخ داده است را نیز گزارش می کند.

به چه دلایلی بهتر است که Scanner **و**parser **به صورت مجزا پیاه سازی شوند؟.**

- سادگی
- کارائی بالاتر
- قابلیت حمل

سادگی:

پارسری که دربردارنده قواعد مربوط به حذف فضا های خالی و توضیحات می باشد پیچیده تر از پارسری است که فرض می کند این اعمال را **scanner** انجام می دهد.

کارائی بالاتر:

خواندن کاراکتر به کاراکتر یک عمل ورودی است قرار دادن این اعمال در**Scanner** و در نظر گرفتن تمهیداتی برای بالا بردن سرعت در **scanner** سبب افزایش کارائی می شود.

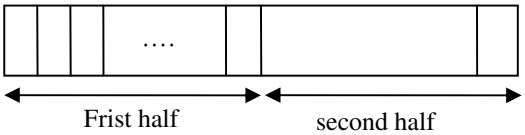
قابلیت حمل:

اعمال قرارداد های مربوط به کاراکتر های ویژه(مثلا علامت ^ در پاسکال) در **scanner** و همین طور اعمال مدیریت ورودی وابسته به هر زبان در **scanner** قابلیت حمل تحلیلگر را بالا می برد.

☐ وقت گیرترین کار کامپایلر همان کاری است که **scanner** انجام می دهد پس باید سرعت را بالا ببریم، باید از بافر استفاده شود چون بافر باعث بالا رفتن سرعت می شود.

افزایش سرعت:

تحلیلگر لغوی یا **scanner** تنها فازی از کامپایلر است که عمل ورودی را انجام می دهد بنابراین به دلیل این که به ازای هر کاراکتر می بایست یک عمل ورودی انجام داد استفاده از روشی جهت افزایش سرعت مفید می باشد این روش استفاده از بافر می باشد.



هر یک ار انواع بافر ها دارای دو نوع اشاره گر مکی باشند

Forward : با خواندن هر کاراکتر یک واحد جلو می رود .

Lexeme_beginning : ابتدای توکنی را که می خواهیم مشخص کنیم نشان می دهد

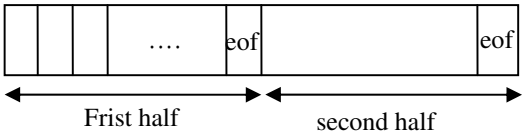
کد مربوط به پیشروی اشاره گر پیش رو(forward) به شکل زیر خواهد بود.

```
if  forward at the end of  frist  half  then begine
reload  second  half;
forward = forward +1
end
else if  forward at the end of  second  half  then begine
reload  frist  half;
move  forward  to  beginning  of  frist  half
end
else  forward = forward +1
```

به ازای خواندن هر کاراکتر به دو تا مقایسه نیاز داریم می خواهیم تعداد مقایسه ها را کمتر کنیم(یک مقایسه) راه حل آن است که از نگهبان استفاده کنیم

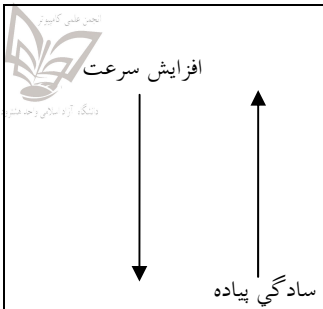
کاراکتر نگهبان:eof

کاراکتر نگهبان انتهای هر نیمه بافر قرار می گیرد و کاراکتر ویژه ای است که جزء متن نیست.



کد مربوط به پیشروی اشاره گر پیش رو(forward) همراه با کاراکتر نگهبان به شکل زیر خواهد بود.

```
forward = forward +1;
if  forward ↑= eof  then  begine
if  forward at the end of  frist  half  then begine
reload  second  half;
forward = forward +1
end
else if  forward at the end of  second  half  then begine
reload  frist  half;
move  forward  to  beginning  of  frist  half
end
else/*eof  within  a  buffer  signifiying  end  of  input8/
end
```



روش های پیاده سازی scaner

۱- استفاده از مولد تحلیلگر لغوی، lex در سیستم عامل unix

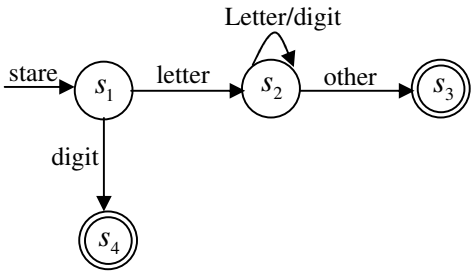
۲- پیاده سازی به کمک زبان برنامه سازی سیستم و استفاده از امکانات مدیریت ورودی در آن زبان

۳- پیاده سازی به کمک زبان اسمبلی و مدیریت مستقیم ورودی

☐به ابزار هائی که خودشان کامپایلر تولید می کنند کامپایلر کامپایلرها می گویند مثلlex در سیستم عامل unix

☐ الگو های توکن ها را به وسیله عبارت منظم نمایش می دهند و هر عبارت منظمی قابل تبدیل به dfa است.

شکل زیر نمودار تغییر حالت برای شناسه ها را نشان می دهد، قاعده تشخیص شناسه عبارت است از دنباله ای از حروف و ارقام که اولین نماد آن حرف باشد .



حال table ها را مشخص می کنیم.

جدول char-class

class	letter	letter	digit	other
char	a-z	A-Z	0-9	other

جدول state

state	letter	digit	other
s_1	s_2	s_4	s_4
s_2	s_2	s_2	s_3
s_3	–	–	–
s_4	–	–	–

هر token از دو مولفه تشکیل می شود یکی نوع token و دیگری مقدار token

بعد از یافتن هر توکن می بایست توکن در جدول نماد ها ذخیره شود.

در جدول نماد ها نوع و نام و شماره اولین سطر ذخیره می شود.

< token type , token value >