فصل اول

« ساختار کامپایلرها »

مقدمــه

زبان‌های برنامه نویسی ابزاری برای تولید برنامه‌ها و نرم افزارها هستند. زبان‌های برنامه نویسی در سطوح متفاوتی تعریف می‌شوند که هر یک دارای ویژگی‌های خاص خود می‌باشند. پنج سطح متفاوت زبان‌های برنامه نویسی عبارتند از:

۱ـ زبان سطح بالا ۲ـ زبان سطح میانی ۳ـ زبان سطح پایین ۴ـ زبان اسمبلی ۵ـ زبان ماشین

که به ترتیب خصوصیات این زبان‌ها به خصوصیات ماشین نزدیک می‌شود. به عبارت دیگر برنامه نویسی در زبان‌های سطح بالا نسبت به زبان‌های میانی، سطح پایین، اسمبلی و ماشین به مراتب ساده تر است.

چالش اصلی این است که زبان قابل درک توسط ماشین، زبان صفر و یک (زبان ماشین) است. بنابراین هر برنامه که به هر زبان برنامه نویسی نوشته شده باشد نهایتاً باید به زبان ماشین تبدیل شود. از طرف دیگر زبان ماشین دارای محدودیت‌های برنامه‌نویسی است که باعث می‌شود پیچیدگی‌های برنامه نویسی مستقیم به زبان ماشین به وضوح افزایش یابد. از این‌رو برنامه نویسان از زبان‌های سطح بالا که قدرت مورد نیاز برنامه نویسی و همچنین سادگی در نوشتار را در اختیارشان قرار می‌دهند، برای نوشتن برنامه‌ها استفاده می‌کنند و از ابزار دیگری به نام مفسرها و مترجم‌ها در کنار این زبان‌های سطح بالا، بهره می‌گیرند تا برنامه‌ی نوشته شده را به زبان قابل فهم سیستم تبدیل کنند.

به طور کلی ابزارهای ترجمه به چهار دسته تقسیم می‌شوند:

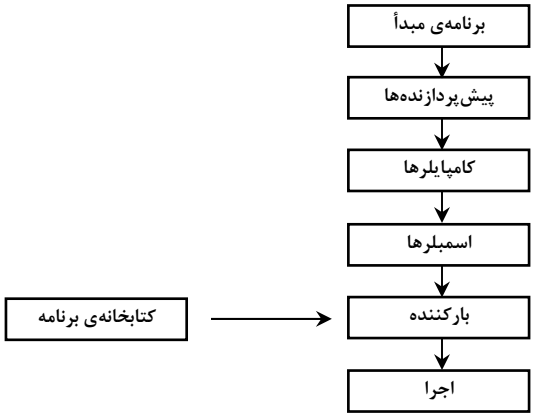
۱ـ کامپایلرها (Compilers) : این ابزار برنامه‌ی منبع را که غالباً به زبان سطح بالا نوشته شده است، از ورودی دریافت کرده و در طی حداقل یک‌بار مرور کل برنامه، خطاهای احتمالی (خطاهای لغوی و نحوی) راکشف و رفع می‌کند. در صورتی‌که دیگر خطایی در برنامه وجود نداشت، کامپایلر کل برنامه را به صورت کامل به زبان مقصد ترجمه می‌کند. (این زبان مقصد غالباً زبان ماشین است.)

۲ـ مفسرها (Interpreters) : این ابزار به صورت خط به خط برنامه منبع را خوانده و خطاهای احتمالی در آن را بررسی می‌کند و در صورت نبودن خطا در هر خط از برنامه، آن خط به زبان مقصد ترجمه می‌شود.

۳ـ اسمبلرها (Assemblers) : این ابزار برنامه‌ی به زبان اسمبلی را از ورودی دریافت و در قالب یک فایل obj به زبان مقصد (زبان ماشین) ترجمه می‌کند.

۴ـ پیش پردازنده‌ها (Preprocessors): این ابزار درواقع یک مترجم اولیه است که داده‌های ورودی را دریافت و یک خروجی تولید می‌کند که این خروجی به عنوان ورودی برای کامپایلر محسوب می‌شود. هدف اصلی استفاده از پیش پردازنده‌ها کاهش بارکاری کامپایلر می‌باشد.

شکل (۱) نحوه‌ی ارتباط مترجم‌ها را نشان می‌دهد.



« شکل ۱ : نحوه‌ی ارتباط مترجم‌ها »

ویژگی‌های کامپایلرها و مفسرها

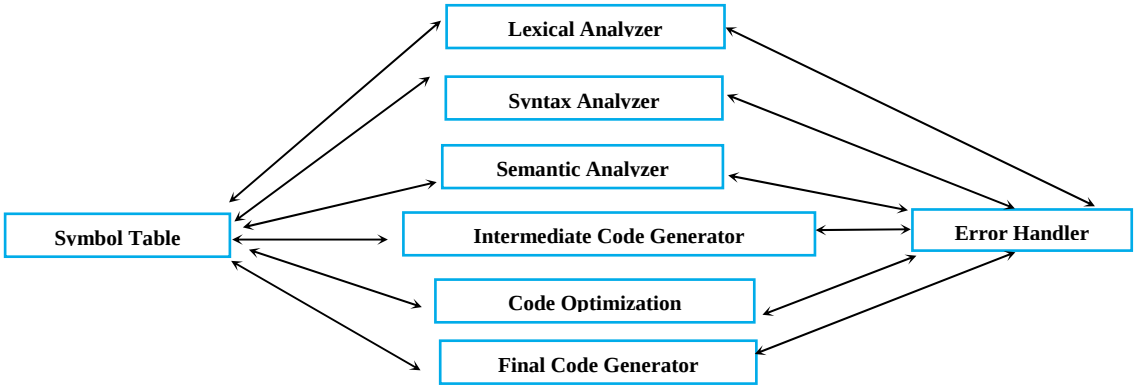
۱ـ در کامپایلر برنامه ابتدا ترجمه می‌شود و بعد اجرا می‌شود ولی در مفسرها برنامه خط به خط ترجمه و اجرا می‌شود.
۲ـ برنامه توسط کامپایلر یک‌بار ترجمه و بارها اجرا می‌شود ولی یک برنامه توسط مفسر یک‌بار ترجمه و یک‌بار اجرا می‌شود. بنابراین سربارکاری زمان ترجمه در مفسرها بالاتر از کامپایلرها است.
۳ـ کامپایلر کد اجرایی تولید می‌کند ولی مفسر کد اجرایی تولید نمی‌کند.
۴ـ برنامه‌ای که توسط کامپایلر ترجمه شود مطمئناً خطای نحوی ندارد ولی برنامه‌ی ترجمه شده توسط مفسر ممکن است دارای خطای نحوی باشد.
۵ـ با هر تغییر در برنامه، کل برنامه باید مجدداً توسط کامپایلر ترجمه شود ولی در مفسر تنها قسمت‌های تغییر یافته مجدداً ترجمه می‌شوند.
۶ـ بهینه‌سازی در برنامه‌ای که کامپایل می‌شود قابل انجام است ولی در برنامه‌ی مفسری این عمل امکان پذیر نیست.
۷ـ برنامه‌ای که توسط کامپایلر ترجمه می‌شود نیازی به کامپایلر برای اجرا ندارد حال اینکه برنامه‌ی مفسری لزوماً نیاز به مفسر برای اجرا دارد.
۸ـ مفسرها کندتر و زمانگیرتر از کامپایلرها هستند.
۹ـ برنامه ای که توسط کامپایلر ترجمه می‌شود خاص یک ماشین است ولی برنامه‌های مفسری روی ماشین‌های گوناگون اجرا می‌شوند.
۱۰ـ تعریف متغیرها در برنامه‌های کامپایلری باید قبلاً انجام شود ولی در برنامه‌های مفسری این‌طور نیست.

مثال۱: مفسرها دارای کدام یک از بخش‌های زیر نیستند؟

۱) Scanner ۲) Parser
۳) Type Checker ۴) Intermediate Code Generator

☑ **پاسخ:** گزینه «۴» مفسرها کدمیانی تولید نمی‌کنند.

مراحل مختلف کامپایل یک برنامه در شکل (۲) نشان داده شده است:



« شکل ۲: مراحل کامپایل یک برنامه »

این مراحل را از دو دیدگاه می‌توان دسته‌بندی نمود:
الف) وابستگی یا عدم وابستگی به سخت‌افزار
ب) تحلیل برنامه و تولید کد

الف) مراحل کامپایل
- Front – end: بخش‌هایی که مستقل از سخت‌افزارند
  - Lexical Analyzer
  - Syntax Analyzer
  - Semantic Analyzer
  - Intermediate Code Generator
- Back- end: بخش‌هایی که وابسته به سخت‌افزارند
  - Code Optimization
  - Final Code Generator

ب) مراحل کامپایل:
- تحلیل کد مبدأ
  - Lexical Analyzer
  - Syntax Analyzer
  - Semantic Analyzer
- تولید کد مقصد
  - Intermediate Code Generator
  - Code Optimization
  - Final Code Generator

مثال۲: کدام مرحله از کامپایل از فازهای اجباری محسوب نمی‌شود؟

۱) Lexical Analysis ۲) Semantic Analysis ۳) Optimization ۴) Code Generation

☑ **پاسخ:** گزینه «۳» فاز بهینه‌سازی کد در مراحل کامپایل جز مراحل اختیاری محسوب می‌شود.

تحلیل لغوی (Lexical Analysis)

اولین فاز از کامپایل برنامه، تحلیل لغوی است که در آن برنامه‌ی ورودی به صورت کاراکتر به کاراکتر دریافت می‌شود.
در این مرحله عملیات زیر انجام می‌شود:
الف) تشخیص لغات
ب) تشخیص Token (توکن‌ها) و ارسال آن‌ها به مرحله‌ی بعد (تحلیل نحوی)
ج) تشخیص و حذف توضیحات (Comments)
د) تشخیص و حذف فضاهای خالی(Spaces)
ه) درج نام لغات کشف شده در جدول نمادها (Symbol Table)

نکته۱: ابزار تحلیل لغوی، اسکنر(Scanner) نامیده می‌شود.

❖ **تعریف:** توکن(Token): به هر لغت تشخیص داده شده در تحلیل لغوی، توکن گفته می‌شود.
توکن‌ها شامل موارد زیر هستند:
۱ـ Identifiers شناسه‌ها
۲ـ Operators عملگرها
۳ـ Key words کلمات کلیدی
۴ـ Constants ثابت‌ها
۵ـ Delimiters جداکننده‌ها

نکته۲: عملیات Scanner، جز عملیات IO محسوب می‌شود و زمانگیر است. به عبارت دیگر زمانگیرترین فاز در کامپایلر، تحلیل لغوی است. Scanner با زمان منظم کار می‌کند.

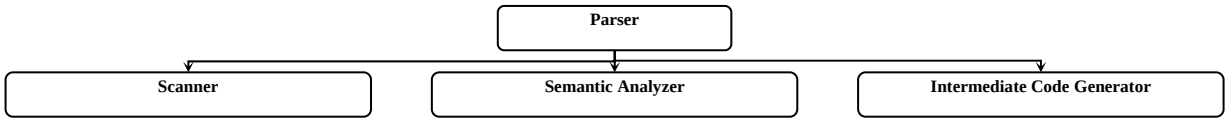
مثال۳: اگر برنامه نویس به جای تایپ کلمه‌ی while در زبان C، کلمه‌ی whil رانوشته باشد، تحلیل‌گر لغوی چه گزارشی در مورد این کلمه می‌دهد؟
۱) خطا ۲) شناسه ۳) صرفاً یک token ناشناخته ۴) هیچکدام

☑ **پاسخ:** گزینه «۲» تشخیص اینکه لغت کلیدی while به صورت whil نوشته شده است در فاز آنالیز نحوی انجام می‌شود و درمرحله‌ی تحلیل لغوی این کلمه به عنوان یک شناسه تشخیص داده می‌شود.

تحلیل نحوی (Syntax Analysis)

پس از تشخیص توکن‌ها و ارسال آن به مرحله‌ی تحلیل نحوی، در این مرحله نحوه‌ی کنار هم قرار گرفتن توکن‌ها مورد بررسی قرار می‌گیرد. معیار بررسی، گرامر مربوط به زبان کامپایلر می‌باشد. به عبارت دیگر در مرحله‌ی تحلیل نحوی بررسی می‌شود رشته‌های تشخیص داده شده عضو زبان کامپایلر مورد نظر هستند یا خیر. در تحلیل نحوی از درخت تجزیه (Parse Tree) برای تعیین عضویت رشته در زبان استفاده می‌شود.

نکته۳: ابزار تحلیل نحوی، پارسر (Parser) نامیده می‌شود و پارسرها با گرامر مستقل از متن کار می‌کنند.
پارسر در واقع مدیر بخش تحلیل کد مبدأ در کامپایلر است که طبق شکل (۳) با بخش Front-end کامپایلر ارتباط دارد.



شکل ۳: ارتباط Parser و بخش Front end

وظایف پارسر عبارت است از:
الف) دریافت دنباله‌ی توکن‌ها از Scanner
ب) بررسی نحوه‌ی قرار گیری توکن‌ها طبق گرامر زبان کامپایلر
ج) تولید درخت تجزیه و سپس درخت نحو
د) تشخیص گزارش و ترسیم خطاهای نحوی به روش‌های مختلف
ه) درج نوع داده‌ای توکن‌ها در جدول نمادها
و) ارسال درخت نحو نهایی به مرحله‌ی بعد (تحلیل‌گر معنایی)

مثال۴: خطای دستور A[5.5] درکدام مرحله از کامپایل مشخص می‌شود؟
۱) تحلیل نحوی ۲) تحلیل لغوی ۳) تولید کد میانی ۴) تحلیل مفهومی

☑ **پاسخ:** گزینه «۱» تخصیص اندیس اعشار به خانه‌ی یک آرایه در زمان آنالیز نحوی بررسی می‌شود.

تحلیل معنایی (Semantic Analysis)

پس از بررسی دستوری برنامه توسط پارسر در این مرحله برنامه از لحاظ معنایی مورد بررسی قرار می‌گیرد. وظایف اصلی تحلیل‌گر معنایی عبارت است از:

الف) دریافت درخت نحو نهایی از پارسر

ب) بررسی نوع متغیرها در محاسبات (Type Checking)

ج) بررسی حدود بالا و پایین آرایه‌ها

د) بررسی تعریف و یا عدم تعریف متغیرها در هنگام استفاده از متغیر در برنامه

ه) بررسی تعداد ، ترتیب و نوع پارامترهای Formal و Actual در فراخوانی تابع

و) تشخیص، گزارش و ترمیم خطاهای معنایی

ز) تبدیل نوع متغیرها به صورت موقت

نکته۴: درمرحله تحلیل معنایی با Warning مواجه می‌شویم. (مانند تقسیم بر صفر) تحلیل‌گر معنایی با زبان‌های وابسته به متن کار می‌کند.

مثال۵: کدام گزینه خطای مفهومی‌محسوب نمی‌شود؟

۱) جمع یک حرف و یک عدد صحیح در زبان پاسکال

۲) تعریف آرایه در زبان C به صورتی که طول آرایه عدد اعشار باشد.

۳) صدا زدن یک تابع با تعداد پارامترهای حقیقی بیش از پارامترهای فرمال

۴) هر سه مورد

☑ **پاسخ:** گزینه «۲» گزینه‌های (۱) و (۳) هر دو خطاهای مفهومی هستند و گزینه (۲) خطای نحوی محسوب می‌شود.

تولید کد میانی (Intermediate Code Generation)

پس از بررسی برنامه از لحاظ خطاهای دستوری، معنایی و رفع خطاها، درخت نحو نهایی شده در مرحله‌ی تحلیل معنایی به مرحله‌ی تولید کد میانی ارسال می‌شود. در این مرحله کدهای سه آدرسه به فرمت مقابل از روی درخت نحو نهایی شده تولید می‌شوند: $(OP, A, B, C) \equiv C = A\ OP\ B$

معمولاً کدهای سه آدرسه به صورتی طراحی می‌شوند که معادل یک، دو یا سه دستور زبان ماشین می‌باشند.

مثال۶: کدام گزینه صحیح است؟

۱) تولید کد میانی قبل از تحلیل مفهومی انجام می‌شود.

۲) تولید کد میانی باعث افزایش سرعت کامپایل می‌شود.

۳) در زبان‌هایی که عمل بهینه‌سازی کد انجام نمی‌شود، نیازی به تولید کد میانی نیست.

۴) هیچکدام

☑ **پاسخ:** گزینه «۴» تولید کد میانی قبل از بهینه‌سازی کد انجام می‌شود و این عمل باعث افزایش سرعت کامپایل نمی‌شود. از طرفی کد میانی و بهینه‌سازی کد، هیچ وابستگی فازی به یکدیگر ندارند.

بهینه‌سازی کد میانی (Code Optimization)

در این مرحله کدهای سه آدرسه‌ی تولید شده در مرحله قبل برای افزایش سرعت اجرا وکاهش حجم، بهینه می‌شوند.

تولید کد نهایی (Final Code Generation)

آخرین مرحله‌ی ترجمه‌ی یک برنامه توسط کامپایلر است که برنامه‌ی مبدأ، به زبان ماشین تبدیل می‌شود. در این مرحله نیز تکنیک‌های بهینه‌سازی وجود دارد. معروف‌ترین تکنیک‌ها عبارتند از:

الف) تکنیک Peephale Optimization در این تکنیک تا حد ممکن از ثبات‌ها استفاده می‌شود.

ب) تکنیک صرفه جویی در مصرف ثبات‌ها

مثال۷: خطای دستورات زیر در زبان C اگر محدوده‌ی آرایه A از نوع عدد صحیح از 0 تا 100 باشد درکدام مرحله‌ی کامپایل مشخص می‌شود؟

```
a)A[50] = 10.5;
  i = 10;
b)[i + 100] = 2;
```

۱) هر دو در زمان آزمون نوع

۲) خط a در زمان آزمون نوع و خط b در زمان اجرا

۳) هر دو در زمان اجرا

۴) خط a در زمان اجرا و خط b در زمان آزمون نوع

☑ **پاسخ:** گزینه «۲» تخصیص یک مقدار اعشار به یک متغیر از نوع صحیح در زمان آزمون نوع و بررسی محدوده‌ی یک آرایه در زمان اجرا بررسی می‌شود.

مدیر خطا (Error Handler)

ابزاری است که برای کنترل خطا در هر مرحله از کامپایل به کار می‌رود تا بتوان عمل کامپایل را پس از رفع خطا دنبال نمود. ارتباط مدیر خطا با مراحل کامپایل به شرح زیر است.

الف) تحلیل‌گر لغوی: در صورت وجود توکن‌های نامعتبر خطای لغوی رخ داده است. برای مثال یک شناسه با عدد شروع شود.

ب) تحلیل‌گر نحوی: عدم رعایت گرامر مربوط به زبان کامپایلر باعث بروز خطای نحوی می‌شود. برای مثال دو عملگر پشت سر هم قرار بگیرند.

ج) تحلیل‌گر معنایی: خطاهای معنایی و منطقی در این مرحله بررسی می‌شوند. برای مثال: عمل تقسیم بر صفر، حلقه‌ی بی‌نهایت، عدم رعایت حدود آرایه، عدم تطابق پارامترهای Formal و Actual .

د) مراحل تولید کد میانی، بهینه‌سازی کد و تولید کد نهایی: تا رسیدن به این مراحل اگر برنامه خطای لغوی، نحوی یا معنایی داشت رفع می‌گردد و اگر خطایی رخ دهد از جانب برنامه نخواهد بود. برای مثال عدم وجود حافظه‌ی کافی یا مراجعه به آدرس غیر مجاز. این خطاها در زمان اجرا رخ می‌دهند.

مثال۸: درهنگام کامپایل یک برنامه Error- Recovery درچه مرحله‌ای صورت می‌گیرد؟

۱) بهینه‌سازی ۲) تحلیل نحوی ۳) تولید کد میانی ۴) تولید کد ماشین

☑ **پاسخ:** گزینه «۲» پس از مراحل تحلیل لغوی، تحلیل نحوی و تحلیل معنایی، هرگز خطایی در برنامه وجود ندارد.

جدول نمادها (Symbol Table)

ساختمان داده‌ای است که اطلاعات مربوط به هر یک از شناسه‌های موجود در برنامه را در خود نگهداری می‌کند. این اطلاعات شامل: نام، نوع داده‌ای، آدرس، تابع یا متغیر بودن و ... است.

ارتباط مراحل مختلف کامپایل یک برنامه با جدول نمادها به شرح زیر است:

الف) تحلیل‌گر لغوی: اسکنر با تشخیص هر توکن نام آن را در جدول نمادها جستجو می‌کند. در صورت نبودن آن توکن، نام آن را درجدول درج می‌کند.

ب) تحلیل‌گر نحوی: تعیین نوع داده‌ای بعضی توکن‌ها در این مرحله انجام می‌شود.

ج) تحلیل‌گر معنایی: تعیین و تبدیل نوع داده‌ای شناسه‌ها در این بخش انجام می‌شود.

د) تولید کد میانی: درجدول نمادها، به بررسی آدرس شناسه می‌پردازد که قبلاً پرشده است یا خیر.

ه) بهینه‌سازی کد: در اثر بهینه‌سازی کد ممکن است تعداد شناسه‌ها و متقابلاً آدرس شناسه‌ها تغییر کند.

و) تولید کد نهایی: از جدول نمادها تنها برای خواندن آدرس استفاده می‌کند.

نکته۵: مراحل تولید کد میانی و کد نهایی از جدول نمادها برای خواندن آدرس استفاده می‌کنند و مراحل تحلیل لغوی، تحلیل نحوی، تحلیل معنایی و بهینه‌سازی کد، محتوای جدول نمادها را تغییر می‌دهند.

مثال۹: مراحل کامپایل دستور $x = y - z / 20$ نشان داده شده است.

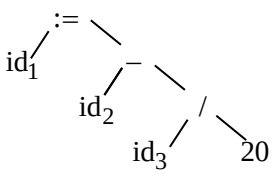
☑ **پاسخ:**

***Expression :** $x := y - z / 20$

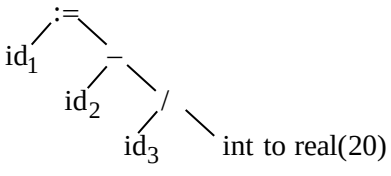
***Scanner** $:< id,1 >< op,:=>< id,2 >< op,- >< id,3 >< op,/ >< num,\#20 >$

$id_1 := id_2 - id_3 / 20$

*** Parser:**



*** Semantic Analyzer**



Intermediate Code Generator

(inttoreal,#20,100,)

(/,52,100,101)

(−,51,101,102)

(:=,102,50,)

*** Optimization :**

Symbol table

	Name	Type	.....	Address
1	x	Real		50
2	y	Real		51
3	z	Real		52
...				

(/,52,#20.0,100)
(−,51,100,50)
* **Final Code Generator**

MOV $20.0, R_1$
LD $52, R_2$
DIV R_2, R_1
LD $51, R_2$
SUB R_2, R_1
STORE $R_1, 50$

مثال۱۰: اطلاعات لازم برای جدول نمادها، درکدام فاز جمع‌آوری و در کدام فاز استفاده می‌شود؟

۱) ترجمه ـ ترکیب ۲) تحلیل ـ تولید ۳) تحلیل ـ ترکیب ۴) ترجمه ـ تولید

☑ **پاسخ:** گزینه «۲» اطلاعات جدول نمادها در فازهای اول، دوم و سوم که معروف به فاز تحلیل هستند جمع‌آوری و در فازهای چهارم، پنجم و ششم که فاز تولید نامیده می‌شوند استفاده می‌شوند.

دسته‌بندی کامپایلرها

کامپایلرها از نظر تعداد دفعات مرور کد برنامه به دو دسته‌ی تک گذره و چند گذره تقسیم می‌شوند.

کامپایلرهای تک گذره: کامپایلرهایی هستند که تنها با یک‌بار خواندن برنامه‌ی ورودی از ابتدا تا انتها کد نهایی به زبان ماشین را تولید می‌کنند.

کامپایلرهای چند گذره: کامپایلرهایی هستند که حداقل دوبار کد برنامه را برای تولید کد نهایی بررسی می‌کنند.

نکته۶: اگر n برنامه‌ی مبدأ و m تعداد کامپیوترهای متفاوت با سخت‌افزارهای متفاوت باشد آنگاه تعداد کامپایلرهای مورد نیاز برای تولید کد مقصد برابر است با:

الف) درحالتی که مراحل کامپایل به دو بخش Front-end و Back-end تقسیم نشوند: $n \times m$

ب) در حالتی که مراحل کامپایل به دو بخش Front- end و Back- end تقسیم شوند: $n + m$

پیش پردازنده‌ها و اسمبلرها (Preprocessors and Assemblers)

پیش پردازنده‌ها به طور کلی ورودی یک کامپایلر را فراهم می‌کنند این بخش عملیات زیر را انجام می‌دهد.:

۱ـ امکان تعریف ماکرو برای کوتاه سازی ساختارهای برنامه نویسی طولانی‌تر

۲ـ امکان درج سر فایل‌ها در متن یک برنامه

۳ـ امکان تلفیق زبان‌های قدیمی و جدید برای دستیابی به ساختارهای برنامه‌نویسی پیچیده‌تر

۴ـ امکان توسعه‌ی قابلیت‌های یک زبان برنامه نویسی

بعضی از کامپایلرها به عنوان خروجی، کد اسمبلی تولید می‌کنند و لازم است که این کد برای پردازش به واحد اسمبلر داده شود. کد اسمبلی یک شبه کد ماشین است که به جای ارقام باینری در آن از اسامی‌استفاده شده است که هر یک معرف یک خانه‌ی حافظه است. برای مثال به دستورات اسمبلی زیر توجه می‌کنیم.

MOV X, R_1
ADD $\#10, R_1$
MOV R_1, Y

این دستورات به ترتیب مقدار متغیر X را در ثبات R_1 قرار می‌دهد، سپس ده واحد به ثبات R_1 اضافه کرده و حاصل را در متغیر Y قرار می‌دهد.
ساده‌ترین فرم اسمبلر، یک اسمبلر دو گذره است. در گذر اول همه‌ی متغیرهایی که مکان‌های ذخیره سازی مقادیر را نشان می‌دهند پیدا شده و در جدول نمادها درج می‌شوند. درگذر دوم اسمبلر بار دیگر ورودی را بررسی کرده و در این مرحله هر کد عملیاتی را به رشته‌ای از بیت‌ها که معرف معادل آن کد عملیاتی در زبان ماشین هستند تبدیل می‌کند. همچنین هر متغیر را به آدرس معادلش در جدول نمادها تبدیل می‌کند. مجدداً برای مثال معادل دستورات اسمبلی مذکور به شکل زیر هستند.

0001 01 00 00010110
0011 01 10 00001010
0010 01 00 00011010

در این کد، چهار بیت اول معرف کد ماشین دستورات ADD, LOAD و STORE هستند و دو بیت بعد معرف ثبات R_1 هستند. دو بیت بعدی معرف نحوه‌ی آدرس‌دهی است و هشت بیت آخر معرف آدرس متغیرها یا مقدار عدد می‌باشد. (فرض بر این است که آدرس متغیر X از خانه‌ی 22 حافظه شروع شده است)

پیوند دهنده‌ها و بارکننده‌ها (Linkers and Loaders)

یکی دیگر از واحدهایی که یک کامپایلر را همراهی می‌کند واحد بارکننده است که از دو بخش بار کردن (Loading) و ویرایش کردن پیوندی (Link Editing) تشکیل شده است.

وظیفه‌ی اصلی بار کننده عبارت است از :

Loading:
۱ـ دریافت کد ماشین قابل جابه‌جایی
۲ـ تعویض آدرس‌های موجود در کد
۳ـ قرار دادن دستوارت و داده‌های تغییر یافته در حافظه

Link- Editing:
۴ـ امکان ایجاد یک برنامه‌ی واحد از فایل‌های مختلف کد ماشین

این فایل‌های مختلف ممکن است از کامپایل‌های متفاوت حاصل شده باشد و یا اینکه فایل‌های کتابخانه‌ای باشند.

مثال۱۱: کدام گزینه تعریف صحیح یک کامپایلر است؟

۱) برنامه‌ای است که یک برنامه به زبان منبع را به معادل آن در زبان مقصد ترجمه می‌کند.
۲) میان افزاری است برای تبدیل برنامه‌های سطح بالا به زبان ماشین.
۳) مجموعه‌ای از نرم‌افزار و سخت‌افزار که یک برنامه‌ی ورودی را دریافت کرده ویک سری عملیات سخت‌افزاری متناسب با آن را تولید می‌کند.
۴) ۱ و۳

☑ **پاسخ:** گزینه «۱» کامپایلر یک نرم‌افزار مترجم است و یک میان افزار نیست.

کاهش تعداد گذرهای کامپایلر

کاهش تعداد گذرهای کامپایلر از این نظر که تعداد دفعات خواندن و نوشتن فایل‌های میانی کاهش پیدا می‌کند بسیار مناسب است. از طرف دیگر برای ترکیب چند فاز در یک گذر کامپایل، باید کل برنامه درحافظه‌ی اصلی نگهداری شود. زیرا ممکن است یک فاز به اطلاعات خاص و به فرم دیگری که در فاز قبل از خود تولید شده است نیاز داشته باشد.

ترکیب بعضی از فازهای کامپایل، در یک گذر ممکن است چالش‌هایی نظیر موارد زیر را ایجاد کند:

۱ـ واسط بین فازهای تحلیل لغوی و تحلیل نحوی اغلب یک توکن است ونه چیز دیگر!
۲ـ اغلب بسیار دشوار است که قبل از تولید کد میانی بطور کامل، کد نهایی حاصل شود.
۳ـ بعضی از زبان‌ها ساختار goto را پشتیبانی می‌کنند که پرش‌های غیر شرطی حاصل از آن یک چالش در کاهش گذرهای کامپایل‌است.

در بعضی موارد می‌توان دو فاز آخر کامپایل را که عبارتند از تولید کد میانی و تولید کد نهایی با یکدیگر ترکیب کرد و این دو فاز را در یک گذر اعمال کرد. به این تکنیک اصطلاحاً اصلاح کد به عقب (Back Patching) می‌گویند. که در فصل مربوط به تولید کد میانی (فصل پنجم) به آن اشاره خواهد شد.

تست‌های طبقه‌بندی شده فصل اول

۱ـ اگر A یک متغیر از نوع آرایه‌ی یک بعدی با اندیس‌ها ی 1 تا 10 و I یک عدد صحیح باشد، خطای اندیس خارج از محدوده (subscript out of range) که در دستورات: A[I] := B; I := 11 وجود دارد، در حالت کلی در چه زمانی و توسط چه برنامه‌ای قابل کشف است؟ (مهندسی کامپیوتر ـ سراسری ۸۲)

۱) زمان اجرا توسط سیستم عامل
۲) زمان کامپایل توسط تجزیه‌کننده‌ی دستوری (Parser)
۳) زمان کامپایل توسط تجزیه‌کننده‌ی مفهومی (Semantic Analyzer)
۴) زمان اجرا توسط خود برنامه‌ی حاوی دستورات فوق

۲ـ در برنامه‌ی زیر هر دو کاربرد عدد 10.5 خطا است. خطای اول در Declaration و خطای دوم در عبارت به ترتیب در کدام یک از بخش‌های Compiler کشف می‌شوند؟ (مهندسی کامپیوتر ـ سراسری ۸۳)

```
    ⋮
int A[10.5];
    ⋮
...A[10.5]+B...
    ⋮
```

۱) Parser و Parser
۲) Scanner و Parser
۳) Parser و کد خطایاب در زمان اجرا
۴) Parser و یکی از Semantic Routineها

۳ـ قواعد گرامری روبرو را در نظر می‌گیریم. کدام یک از گزاره‌های زیر صحیح است؟ (مهندسی کامپیوتر ـ سراسری ۸۴)

$D \rightarrow TL$
$T \rightarrow int / real$
$L \rightarrow L, id$
$L \rightarrow id$

۱) اسم و نوع متغیرها در حین تحلیل معنایی وارد جدول نمادها می‌شوند.
۲) اسم و نوع متغیرهای معرفی شده به راحتی در مرحله‌ی تحلیل لغوی در جدول نمادها وارد می‌شوند.
۳) اسم متغیرها در حین تحلیل لغوی وارد جدول نمادها می‌شود ولی نوع آن‌ها در حین تولید کد است که تشخیص و وارد جدول می‌شود.
۴) با اجرا کردن قواعد معنایی (Semantic Rules) صحیح در حین یک تجزیه‌ی پایین به بالا می‌توانیم نوع کلیه‌ی متغیرهای معرفی شده را مشخص و در جدول نمادها وارد کنیم.

۴ـ کدام یک از گزاره‌های زیر در مورد زبان‌های برنامه‌نویسی C و پاسکال صحیح است؟ (مهندسی کامپیوتر ـ سراسری ۸۴)

۱) هر دو نوع حساس به متن هستند و به کمک گرامرهای حساس به متن تعریف شده‌اند.
۲) هر دو نوع حساس به متن هستند ولی به وسیله‌ی گرامرهای مستقل از متن تعریف شده‌اند.
۳) هر دو نوع مستقل از متن هستند و به همین دلیل آن‌ها را به وسیله‌ی گرامرهای مستقل از متن تعریف کرده‌اند.
۴) زبان پاسکال از نوع مستقل از متن و زبان C از نوع حساس به متن است و هر کدام توسط گرامری از نوع مربوطه تعریف شده‌اند.

۵ـ مزایای تولید کد آرایه‌ها با آدرس شروع مجازی در زبان‌هایی که حد پایین اندیس آرایه‌ها لزوماً صفر نیست کدام است؟
(مهندسی کامپیوتر ـ سراسری ۸۶)

۱) سرعت ترجمه افزایش می‌یابد.
۲) سرعت اجرا افزایش می‌یابد.
۳) حجم کد قابل اجرا کاهش می‌یابد.
۴) هر سه مورد

۶ـ کامپایلرهای A و B برای زبان X مفروضند. کامپایلر A به ازای هر ارجاع به آرایه‌های یک بعدی دو کد زبان ماشین و کامپایلر B، سه کد تولید می‌کند. کامپایلر A برای اصلاح توصیف‌کننده‌ی هر آرایه، محاسباتی انجام می‌دهد ولی B کار خاصی نمی‌کند. کدام گزینه صحیح است؟
(مهندسی کامپیوتر ـ سراسری ۸۸)

۱) فضایی که کامپایلر A برای نگهداری اطلاعات آرایه‌ها در زمان اجرا اختصاص می‌دهد از B کمتر است.
۲) کامپایلر A در مجموع برای آماده‌سازی توصیف‌کننده‌ی آرایه و ترجمه‌ی هر ارجاع به آرایه یک بعدی وقت بیشتری می‌گیرد. ولی B کدی تولید می‌کند که اجرای آن سریع‌تر است.
۳) هر دو کامپایلر برای یک برنامه‌ی واحد با هر تعداد ارجاع به آرایه‌های یک بعدی، به یک اندازه وقت می‌گیرند. ولی کدی که A تولید می‌کند سریع‌تر اجرا می‌شود.
۴) کامپایلر B در مجموع برای آماده‌سازی توصیف‌کننده‌ی آرایه و ترجمه‌ی هر ارجاع به آرایه یک بعدی وقت بیشتری می‌گیرد. ولی A کدی تولید می‌کند که اجرای آن سریع‌تر است.

پاسخنامه تست‌های طبقه‌بندی شده فصل اول

۱ـ گزینه «۴» این خطا در زمان اجرا و توسط برنامه‌ی حاوی دستورات فوق کشف می‌شود. علی‌رغم اینکه یکی از وظایف تحلیل‌گر معنایی بررسی حدود بالا و پایین اندیس‌های آرایه می‌باشد، اما در این حالت خاص چون مقدار اندیس صریحاً ذکر نشده است، این خطا در مرحله‌ی تحلیل معنایی ایستا قابل تشخیص نیست.

۲ـ گزینه «۱» به طور کلی هر رشته یا ورودی که در گرامر زبان نباشد در مرحله‌ی تحلیل نحوی شناخته می‌شود. هر دوی این خطاها باعث می‌شود رشته‌ی ورودی عضو گرامر زبان نباشد. زیرا در گرامر، اندیس آرایه اعشار نمی‌تواند باشد.

۳ـ گزینه «۴» اسم متغیرها در مرحله‌ی تحلیل لغوی و نوع متغیرها در مرحله‌ی تحلیل نحوی و معنایی وارد جدول نمادها می‌شود. همچنین با اجرای قواعد معنایی دقیق در پارس پایین به بالا می‌توان نوع کلیه‌ی متغیرهای برنامه را در جدول نمادها وارد کرد.

۴ـ گزینه «۲» زبان‌های C و پاسکال از نوع زبان‌های نوع اول (حساس به متن هستند) ولی اغلب قواعد توسط گرامرهای مستقل از متن پیاده‌سازی شده است و تنها حالات خاص آن‌ها مانند بررسی نوع، ترتیب و تعداد پارامتر و ... که در گرامرهای مستقل از متن وجود ندارد را توسط گرامرهای حساس به متن پیاده‌سازی کرده‌اند.

۵ـ گزینه «۴» از مزایای تولید مستقیم کد آرایه‌ها که آدرس شروع مجازی آن‌ها نیز لحاظ شده است، در زبان‌هایی که اندیس شروع از صفر می‌تواند نباشد، اولاً افزایش سرعت اجرا، افزایش سرعت ترجمه و همچنین به دلیل انجام پیش محاسبات، کاهش حجم کد قابل اجرا می‌باشد.

۶ـ گزینه «۴» گزینه (۱) نادرست است زیرا کامپایلر A برای اصلاح توصیف کننده در زمان اجرا محاسباتی را انجام می‌دهد که باعث می‌شود فضای اشغالی آن از B بیشتر شود گزینه (۲) و (۳) نادرست هستند. زیرا کامپایلر B در زمان کامپایل (ترجمه) به اصلاح توصیف کننده‌های آرایه می‌پردازد و همچنین تعداد کد سه آدرسه‌ی تولید شده توسط کامپایلر B بیشتر از A است. در نتیجه کد تولید شده توسط B در مجموع برای آماده‌سازی توصیف کننده‌ی آرایه و ترجمه‌ی هر ارجاع، زمان بیشتری می‌گیرد و کد تولید شده‌ی A سریع‌تر است.

تست‌های تألیفی فصل اول

۱ـ پیام زیر در چه مرحله‌ای از کامپایل تشخیص داده می‌شود؟
" متغیر X تعریف شده است ولی استفاده نشده است "
۱) تحلیل لغوی ۲) تحلیل نحوی ۳) تحلیل معنایی ۴) بهینه‌سازی کد

۲ـ کدام موارد از مزایای کد میانی است؟
الف) قابلیت بهینه‌سازی ب) قابلیت ترجمه به انواع کد اجرایی ج) قابلیت جابه‌جایی کد د) افزایش سرعت کامپایل
۱) ب و د ۲) الف و ب و ج و د ۳) الف و ب و ج ۴) ج و د

۳ـ عمل آزمون نوع (Type Checking) در زبان‌های مفسری درچه مرحله‌ای صورت می‌گیرد؟
۱) مرحله‌ی اجرا ۲) مرحله‌ی ترجمه ۳) مرحله‌ی پیاده سازی زبان ۴) مرحله‌ی ترجمه و مرحله‌ی اجرا

۴ـ با توجه به قطعه کد زیر کدام گزینه درست است؟

```
var
x : array[1...20] of integer;
a, b : integer;
begin
a = 40;
i)x[a]= 3;
read(b);
j)x[b]= 10;
```

۱) خط i تولید خطا کرده و در مرحله‌ی بهینه‌سازی مشخص می‌شود.
۲) خط i تولید خطا می‌کند که در زمان اجرا مشخص می‌شود.
۳) خط j در صورتی که b در بازه‌ی 1 تا 20 نباشد در زمان اجرا تولید خطا می‌کند.
۴) اگر b دربازه‌ی 1 تا 20 نباشد دو خط i و j دارای خطا می‌باشند و در زمان اجرا مشخص می‌شوند.

۵ـ محاسبه‌ی آدرس متغیر i در دستورهای i*=2 و و i =i*2 به ترتیب چند بار انجام می‌شود؟
۱) 1 و 1 ۲)2 و 1 ۳) 1 و 2 ۴) 2 و 2

۶ـ اگر قطعه کد زیر در زبان C موجود باشد وعدد n از ورودی برابر 100 خوانده شود، آنگاه وجود خطای "array reference out of bounds" چه نوع خطایی است؟

```
int A[100];
cin >> n;
A[n] = 8.5;
```

۱) Lexical Error
۲) Static Semantic Error
۳) Dynamic Semantic Error
۴) Syntax Error

۷ـ در هنگام کامپایل کردن یک برنامه، Tokenها در چه مرحله‌ای شناسایی می‌شوند؟
۱) آنالیز لغوی ۲) تولید کد ۳) آنالیز معنایی ۴) آنالیز نحوی

۸ـ کدام مورد توسط Scanner تشخیص داده نمی‌شود؟
۱) سطر و ستون شناسه‌ها ۲) نوع شناسه‌ها ۳) گروه Tokenها ۴) هر سه مورد

۹ـ انواع مختلف Tokenها عبارت است از:
۱) کلمات کلیدی، کلمات غیر کلیدی، جدا کننده‌ها ۲) Operation, Identifier , Literal
۳) BR/CR , Keyword , Operation ۴) ۱و۲

۱۰ـ در دومین مرحله از کامپایل چه ابزاری استفاده می‌شود؟
۱) Optimizer ۲) Scanner ۳) Parser ۴) Recovery

۱۱ـ در مورد دخالت خطا پرداز در مرحله اول کامپایل چه می‌توان گفت؟
۱) دخالت ندارد زیرا تا مشخص نشدن اهداف برنامه نمی‌توان خطاها را تعیین نمود.
۲) دخالت دارد، زیرا خطاپرداز سعی می‌کند خطاها را به نحوی برطرف کند که امکان ادامه مراحل کامپایل فراهم شود.
۳) دخالت ندارد زیرا اولین مرحله برای گزارش خطا، مرحله‌ی تولید کد میانی است.
۴) دخالت دارد زیرا چنانچه خطایی باشد باید قبل از ادامه‌ی مراحل کامپایل، عملیات متوقف نشده و خطا بر طرف شود.