منابع و مراجع

1-Programming Languages Design and Implementation 3rd Edition 2003
By: Pratt

2- طراحی و پیاده سازی زبان های برنامه سازی مترجم : جعفرنژاد قمی

3-Fundamentals of Programming Languages Second Edition 2000
By: Horowitz

4- Comparative Programming Languages By: willson

5- Concepts of Programming Languages By: W. Sebesta

طراحی و پیاده سازی زبانهای برنامه سازی

سر فصل مطالب

ویژگی های عمومی زبانهای برنامه‌سازی
مقایسه زبانهای برنامه‌سازی
ویژگی انواع داده ها (Data types) و پیاده‌سازی آنها در زبان های سطح بالا
روشهای تعیین ترتیب اجرای دستورات وپیاده‌سازی آنها (Sequence control)
کنترل داده ها (Data control)
روشهای تخصیص ارگومان های یک تابع وپیاده‌سازی آنها
روشهای مدیریت حافظه در زبان های سطح بالا
انتزاع داده ها (Abstraction)

فهرست مطالب

مقدمه کلی:

نوام چامسکی خالق نظریه زبانها و ماشینها معتقد است زبان بزرگترین معجزه تاریخ بشر است.

تفاوت بین زبانهای برنامه سازی وزبانهای طبیعی ازنظر نحوه پیدایش :

- در زبانهای طبیعی ابتدا خود زبان به وجودآمده است سپس گرامر ان تعریف شده است.
- در زبانهای برنامه سازی ، ابتدا گرامر تعریف شد و بعد خود زبان بوجود امد .
گرامر شامل (تعریف قوانین ،کامپایلر وکلمات کلیدی و...) می باشد.

اهداف درس :

۱- طراحی و پیاده سازی عناصر گوناگون موجود در زبان

۲- بررسی ویژگیهایی که زبانهای برنامه سازی می توانند داشته باشند .

۳- مطالعه تکنیک های پیاده سازی و تکنیک های زمان اجرای ساختارهای مختلف

تعریف زبانهای برنامه سازی :

مجموعه علائم و قوانین برای توصیف الگوریتمها و ساختمان داده های موجود که قابل پیاده سازی روی کامپیوتر باشند .
مثلاً زبان sparc یک زبان توصیف الگوریتمها است که پیاده سازی نشده است .

بررسی واژه طراحی و پیاده سازی زبانهای برنامه سازی :

برنامه سازی : فرآیندی است که طی آن طراح مراحل زیر را طی میکند :

۱- شناخت محیط عملیاتی
۲-شناسایی کلاسها و شناسایی صفات (attributes)
۳-بررسی روابطی که بین کلاسها وجود دارد.
۴-تهیه نمودارها شامل use case ,class diagram, sequence diagram diagram,colobration diagram و....
۵- code نویسی با یک زبان خاص (که معمولا طراح خودش این کار را انجام نمیدهد.)
۶- تست و بازبینی (Test and Verification)

نکته ۱: برنامه نویسی فقط شامل مورد پنجم می شود.
نکته ۲: متد برنامه سازی توسط طراح زبان مشخص می گردد.

طراحی (Design) :

تعریف ویژگی‌ها و امکانات موجود در زبان است همراه با تعریف ساختارهای موجود در زبان .یعنی طراح زبان باید لیست تمام ویژگی هایی را که می خواهد در زبانش موجود باشد (با توجه به هدف زبان) انتخاب و لیست می کند.

مثال : طراحی زبان X این زبان دارای ویژگی های زیر است.

۱- recursive
۲- object oriented
۳- function
۴- compiler
۵- data types شامل int و float

پیاده سازی(Implementation) :

اکنون این سوال مطرح است که این ساختارها چگونه در زبان نمایش داده شود یا به عبارتی چگونه در زبان تعریف و استفاده شوند . ساختارهای موجود تعریف شده در مرحله طراحی باید به نحوی در حافظه نمایش داده شود. یعنی طراح باید کلیه امکاناتی که در مرحله طراحی لیست کرده را بتواند پیاده سازی کند.

مهمترین سوال در طراحی زبان این است که

زبان چیست ؟

مفهوم امروزی زبان با آنچه در ابتدا و جود داشت کاملا متفاوت است.

روزی که می خواستند برای اولین زبان را طراحی کنند چه چیزی را باید طراحی می کردند؟

کامپایلر ؟ در آن زمان کامپایلر اصلا وجود نداشت و مفهومی از آن مطرح نبود.

برنامه ما را چه چیزی **اجرا** می کند؟

سیستم عامل؟ در آن زمان سیستم عامل اصلا وجود نداشت و مفهومی از آن مطرح نبود.

دستورالعمل ها و عملیات را چه چیزی **انجام و محاسبه** می کند؟

سخت افزار؟ در آن زمان سخت افزار به مفهوم امروزی وجود نداشت.

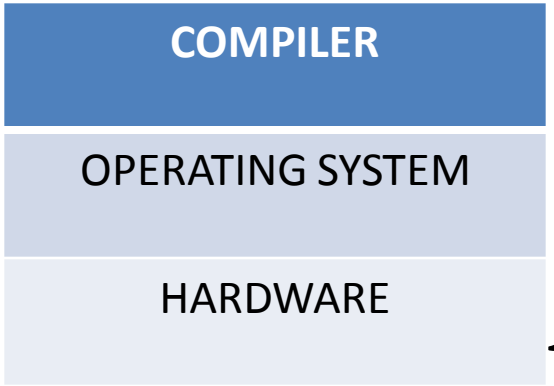
نتیجه : طراحی امروز زبان با طراحی زبان برای اولین بار متفاوت است.

پس طراحان اولین زبان کار بسیار مشکلی داشتند .آنها در حالی که اصلا تصوری از زبان نداشتند شروع به طراحی زبان کردند.

در این درس می خواهیم ببینیم که آنها چه دغدغه ها و مشکلاتی در طراحی داشتند و روش کار آنها چگونه بوده است.

آن چیزی را که ما به عنوان زبان در دروس گذشته یاد گرفتیم فقط بخشی از مفهوم زبان است.

مفهوم امروزی زبان :
زبان یک مفهوم سلسله مراتبی است .



نگاهی به مفهوم ز

زبان دارای یک مفهوم سلسله مراتبی است.

۱- کامپایلر: پوسته زبان است . بخشی است که کاربر با آن کار میکند و ارتباط بین کاربر و سیستم عامل را برقرار می کند.

۲- سیستم عامل: کد آماده شده از کامپایلر را گرفته و اجرا می کند.

۳- سخت افزار: پایین ترین بخش لایه ای زبان است که با برقرای ارتباط با سیستم عامل محاسبات را انجام می دهد.

سیر تکاملی زبانهای برنامه سازی :

- زبانهای سطح پائین(۱۹۴۰) :

۱- زبان ماشین : علائم (1,0)
۲- زبان اسمبلی : علائم (کدهای سمبلیک)

- زبانهای سطح بالا (1950):

- زبانهای نسل اول : Basic – Fortran – Cobol -Algol

- Basic : beginners all purpose symbolic instruction code
- Fortran : formula translation
- Cobol: common business oriented language
- Algol : algorithm language

- زبانهای نسل دوم : PL1 – Pascal – C – Ada – Modula2
- زبانهای نسل سوم : زبانهای پیشرفته : Small talk (شی گرا) Prolog(هوش مصنوعی)
- زبانهای نسل چهارم : زبانهای visual مانند VC++, VB و خانواده .NET

دلایل مطالعه زبانهای برنامه سازی

با توجه به اینکه یادگیری یک یا حداکثر دو زبان برنامه نویسی کافی است این سوال مطرح می شود که مطالعه زبانهای برنامه سازی غیر متداول چه مزایایی دارد؟

۱ – استفاده مناسب از ویژگیهای یک زبان

در صورت مطالعه زبانهای برنامه سازی می توان از امکانات زبان ها به طور مناسب استفاده کرد.بعنی زبان برنامه نویسی مورد استفاده خود را بهتر درک کرد. **زیرا در این درس چگونگی پیاده سازی ویژگی های یک زبان مطالعه می شود.**

سوال۱ : اگر برنامه ای به شما داده شود و زبان شما امکان شی گرایی را داشته باشد آیا از این امکان استفاده می کنید؟

سوال۲: اگر برنامه ای به شما داده شود و زبان شما امکان Recursive و الگوریتم شما بازگشتی تعریف شود آیا از این امکان استفاده می کنید؟

درک بهتر زبان برنامه نویسی منجر به پیاده سازی کارآمد الگوریتم ها می شود.در صورت شناخت ساختارهای موجود در زبان و آشنایی با نحوه پیاده سازی و زمان اجرای آن می توان تصمیم گرفت در چه مواقعی از امکانات خاص زبا ن استفاده کرد .مثلا همیشه برنامه Recursiveننویسیم چون کند است.

تذکر : دلیل فوق باعث می شود که برنامه های نوشته شده به زبان خاص کار آمد باشند زیرا برنامه نویس با نحوه پیاده سازی رشته ها، رکوردها وسایر ساختارها آشنا می شود.

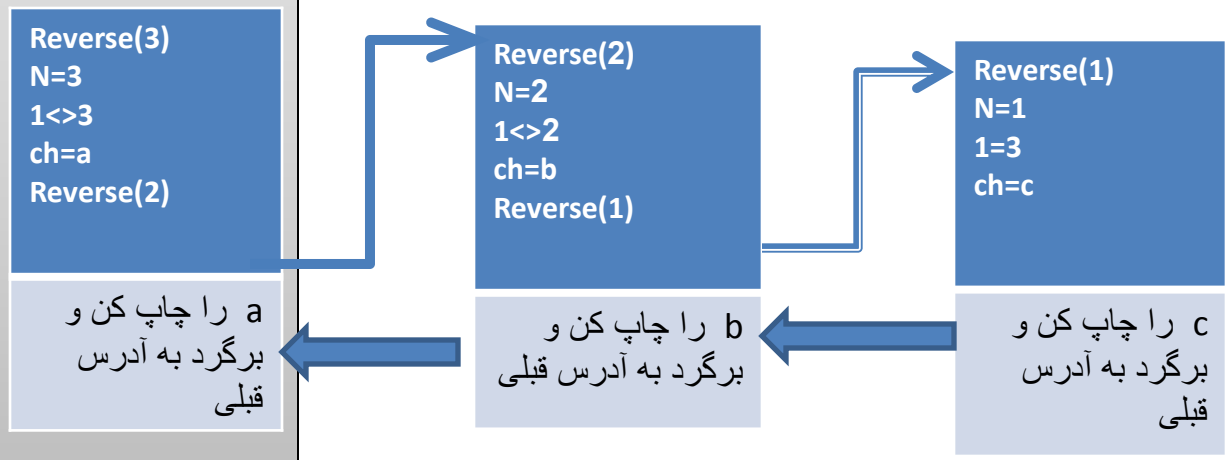
مثال : برنامه ای بنویسید که یک رشته کاراکتری را برعکس ورودی چاپ کند.

```
Program   test;
        Var     n: integer;
 procedure   reverse (n:integer);
    Var
          Ch: char;
Begin
     If  n= 1 then
Begin
   Read(ch);
   Writre(ch);
End;
  Else
Begin
    Read(ch);
    Reverse(n-1);
    Write(ch);
End;
   End;
Begin
          Read(n);
          Reverse(n);
End.
```

توابع بازگشتی در هر بار فراخوانی حافظه جدیدی از سیستم

می گیرند و شناسه های خود را ذخیره می کنند بنابراین ۳ data object جداگانه ایجاد می شود .

که هر یک آدرس بازگشت و سایر موارد مورد نیاز را در خود ذخیره می کنند.



نتیجه : استفاده از متد های برنامه نویسی مانند شی گرایی و برنامه نویسی همزمان نیاز به درک زبانی دارند که آنها را پیاده سازی می کنند.

2- شناختن ساختارهاي مناسب ساير زبانها وشبيه سازي آنها

کار کردن فقط با یک زبان باعث محدودیت در کار برنامه سازی می شود .زیرا هر زبان برنامه سازی فقط از یک سری از ساختارها پشتیبانی می کند و همه ساختارها را پیاده سازی نکرده است مطالعه سایر زبان ها باعث می شود برنامه نویس با آشنا شدن با آن ساختارها ، آنها را در زبان مورد استفاده خود شبیه سازی کند زیرا هر زبان برای منظور خاصی تولید شده است. برای شبیه سازی ساختارهای مفیدی که در سایر زبانها وجود دارد برنامه نویس باید ازعناصر اولیه موجود در زبان استفاده کند .

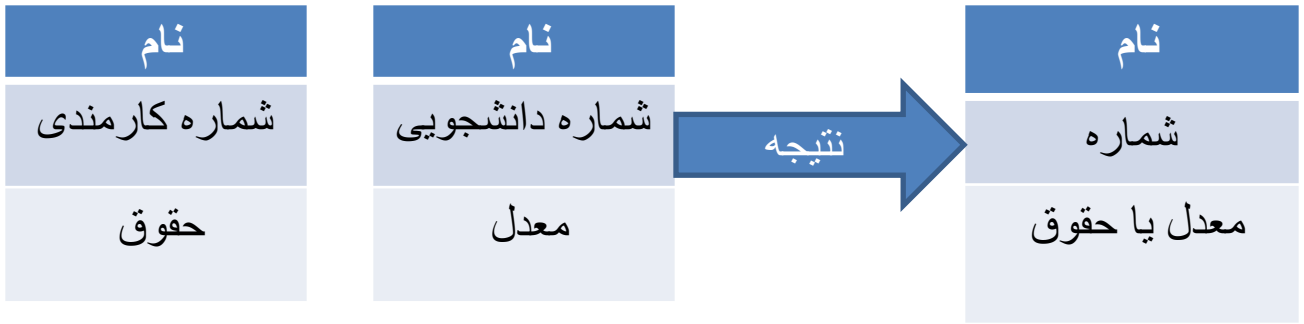
مثال : شناخت ابزار مفیدی به نام variant record و شبیه سازی آن در زبانهای دیگر
نوشتن برنامه سیستم جامع دانشگاه : دارای دو موجودیت کارمند و دانشجو است.

امکان اول : می توان دو رکورد جداگانه برای کارمند و دانشجو در نظر گرفت و برنامه را طراحی و اجرا کرد.

امکان دوم : استفاده از variant record ودر نظر گرفتن یک رکورد شخص برای هر دو موجودیت و ترکیب رکوردها چنین امکانی در زبان پاسکال Ada وجود دارد.

در زبان c می توان آن را با union شبیه سازی کرد.

رکورد شخص (رکورد با طول متغیر)　　رکورد دانشجو　　رکورد کارمند



- **Pascal**
- Person = Record
- Name: string [10] ;
- No: integer;
- Case tag : char of
- ‘a': average :real;
- ‘s': salary: long int;
- End;
- End.

- C/
- Struct person {
- char name [10];
- int n ;
- union{
- float average ;
- long salary;
- } s1;
- }

۳- انتخاب بهترین زبان برنامه نویسی برای یک پروژه خاص :

بررسی امکانات و توانایی زبان ها باعث می شود با نقاط ضعف و قوت آنها آشنا شده و هنگام انجام پروژه ها بهترین انتخاب شودزیرا هر زبانی برای کاربرد مخصوصی طراحی شده است .

مثال : در پروژه های محاسبات عددی بهتر است از Fortran یا c استفاده گردد.
در پروژه های هوش مصنوعی بهتر است از lisp یا prolog استفاده شود.یا به طور کلی برای نوشتن پروژه خاص در شاخه های مختلف کامپیوتر از همان زبان مربوطه استفاده می کنیم .

۴- یادگیری بهتر یک زبان جدید

در صورتی که با زبان آلمانی آشنا باشید یادگیری زبان انگلیسی راحت تر می شود. در صورت آشنایی با ساختارهای همه زبان ها در حالت کلی برای یادگیری زبان جدید زحمت کمتری نیاز است .

۵- طراحی بهتر یک زبان جدید

در صورت آشنایی با ساختارها و متدهای پیاده سازی زبان های برنامه سازی در صورت لزوم برای طراحی یک زبان جدید می توان از امکانات سایر زبان ها استفاده کرد. زیرا pratt معتقد است همه زبان تا ۹۰ درصد دارای ساختارهای یکسان هستند.

تقسیم بندي زبانهاي برنامه سازي از نظر کاربرد :

۱- زبانهای محاسباتی و علمی :

- زبان A-0 : برای کارهای محاسباتی و حل مساله های عددی

- زبان Fortran : (Formula Translation) : اولین زبان سطح بالا در سال ۱۹۵۰ طراحی شده توسط اولین تیم تحقیقاتی bakes & naur در نروژ

خصوصیات :

۱- امکان عبارات ریاضی در دستور انتساب

۲- سادگی ایجاد حلقه های تکرار do

۳- بیان خوب ایده یک زیر برنامه

۴- قابلیت I/o : یعنی برای آن دستورات ورودی / خروجی در نظر گرفتند که باعث مشکل بعدی شد.

۴- قابلیت حمل نداشت.(portability)

- زبان الگول (Algorithm language) : ریشه اکثر زبانهای فعلی است .طراحی شده توسط همان تیم تحقیقاتی زبان fortran

خصوصیات :

۱- ساخت یافته بودن و ساختار بلوکی به مفهوم طراحی ماژولار (بلاکی) است .

ساخت یافته بودن یعنی چه؟(top - down approach)

به زبانی ساخت یافته گفته می شود که به طراح برنامه امکان دهد تا

۱- task های عمده یک برنامه را در ***قالب ماژول*** مشخص و تعریف کند.

۲- برنامه به صورت سلسله مراتبی در داخل ماژولها سازماندهی می شود

۳- هر ماژول یک نقطه ورود و یک نقطه خروج دارد .

شی گرا یی : (Object oriented) به زبانی شی گرا گفته می شود که به طراح برنامه امکان می دهد تا

- ۱- پیچیدگی مساله در دنیای واقعی به صورت ذاتی حل شود.
- ۲- روی تعریف داده تمرکز می کند .
- ۳- توابع یا متد هایی را تعریف می کند که روی آن داده عمل می کنند.
- ۴-توابع یا متدهای فوق را همانند یک قسمت از تعریف ذاتی داده با آن داده ترکیب میکند.

۲- استفاده از نگارش BNF برای طراحی syntax (گرامر) زبان

BNF = Backus Naur Form

<Assignment statement>::=<variable>=<arithmetic expression>

۳- تئوری کامپایلر برای اولین بار به طور جدی مطرح شد .پیاده سازی تقدم عملگر بدون استفاده از پرانتز

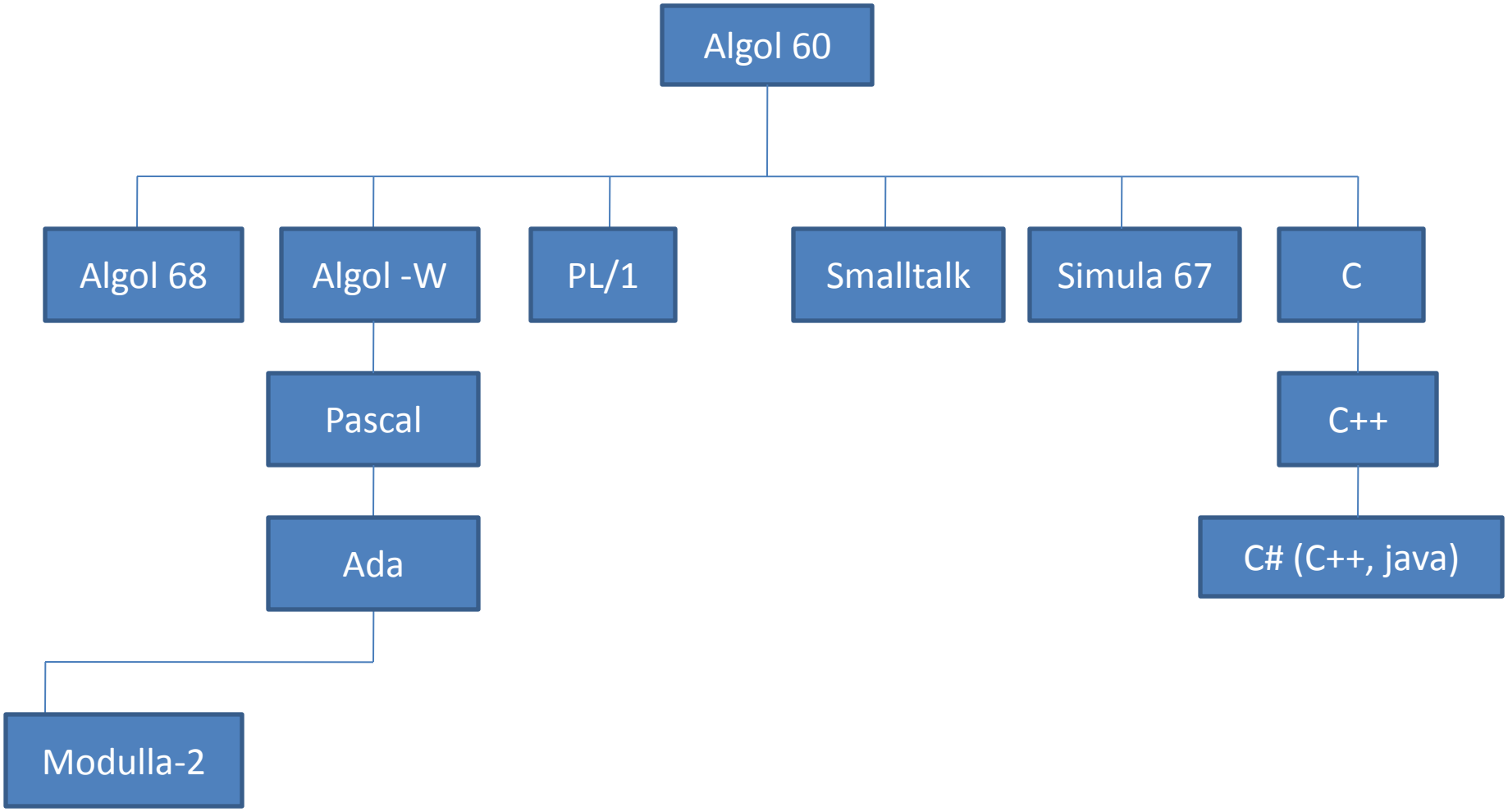
۴- دستورات کنترلی ساخت یافته مانند For ,while, if به طور کامل طراحی شد.

۵- آرایه های با دامنه متغیر : در هنگام طراحی زبان الگول **امنیت** به مفهوم امروزی مطرح نبود به همین دلیل دامنه متغیر مشکلی ایجاد نمی کرد.

۶- پیاده سازی ایده برنامه بازگشتی (Recursive) برای اولین بار (پیشرفت در مدیریت حافظه)

۷- فاقد دستوراتI/O : برای اینکه وابسته به ماشین خاص نباشد .

۸- به علت نزدیک بودن به مفاهیم ریاضی زیر برنامه ها به صورت ماکرو بودند وانتقال پارامترها از طریق call by nameانجام می شد که پیاده سازی آن مشکل است .


Algol 60
Algol 68
Algol -W
PL/1
Smalltalk
Simula 67
C
Pascal
C++
Ada
C# (C++, java)
Modulla-2

- زبان simula 67 : این زبان توسط Christian nygard در یک آزمایشگاه شیمی طراحی شد و اولین بار مفاهیم شی گرایی را مطرح کرد البته این زبان از دید طراحی زبانها اولین زبان شی گرا نیست .

- زبان PL/1 : این زبان از نظر گرامر بسیار نزدیک به c است .صفات عددی و محاسباتی فرترن را با صفات تجاری Cobol ترکیب کرد.

-زبان Smalltalk : این زبان توسط Alen key طراحی شد و برای اولین یک زبان کامل شی گرا با مفاهیم امروزی بود خصوصیات این زبان عبارتند از:

۱- مفهوم class و object

۲- مفهوم ارث بری

۳- مفهوم encapsulation

۴- عوض کردن روش طراحی برنامه از ساخت یافته به شی گرایی(تاکید بر روی data)

-از دیگر زبان های شی گرا می توان به زبان Eiffel و cols اشاره کرد. (common lisp object system)

-زبان Modula 2 : مبنای طراحی همان پاسکال است. خصوصیات :

۱- مفهوم ماژول معادل packageدر Ada و زیر برنامه در پاسکال .

۲- امکان ایجاد برنامه های موازی

۳- امکان ایجاد برنامه نویسی سطح پایین در زبان سطح بالا با ایجاد یک ماژول جدید

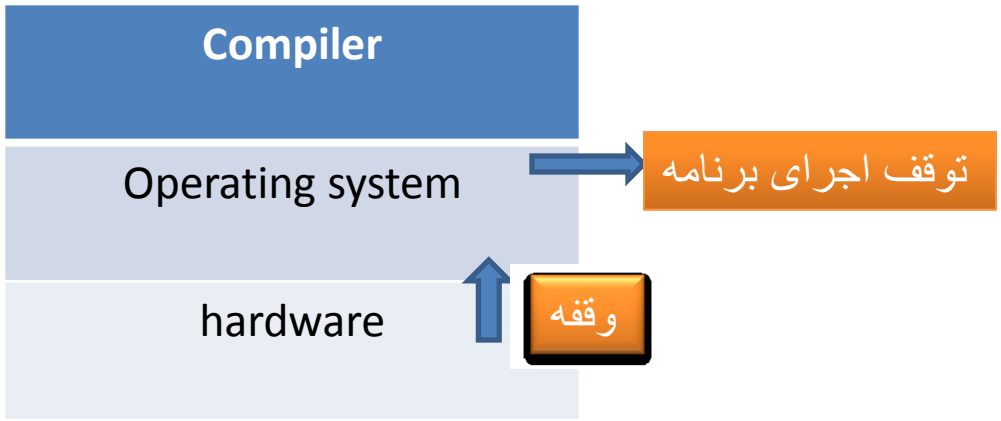
- زبان Ada : این زبان توسط وزارت دفاع آمریکا طراحی شد و به افتخار Ada دستیار چارلزبابیج به نام وی نام گذاری شد. هدف این زبان این بود که اکثر ساختارهای موجود را در کامپایلر خود داشته باشد به همین دلیل کامپایلر آن بزرگ شد .اما بعضی ابداعات و مسائلی که Ada مطرح کرد بسیار مبتکرانه و زمینه ساز پیشرفتها و تغییر دید نسبت به زبان و ایجاد زبان سلسله مراتبی شد.

خصوصیات :

۱- امکانات برای exception Interrupt handling : برای اولین بار ارتباط بین لایه های مختلف زبان ایجاد شد. سوال اینجاست که اگر برنامه از فیلتر کامپایلر بگذرد ولی باعث شود که با یک خطا جامعیت سیستم دچاراشکال شود چگونه باید جلوی آن را گرفت؟ این حالت را exception یا استثنا گویند. در این صورت یک وقفه سخت افزاری صادر می شود و برنامه وقتی در اختیار سیستم عامل است یعنی در حین اجرا متوقف می شود.

مثال : Divided by zero

```
A=10;
B=0;
C=a/b;
```



۲- تعریف ساختارها به صورت ADT (Abstract data type) نوع داده انتزاعی

تعریف ADT : نوع داده ای است که در آن مشخصات (عناصر و عملیات) مستقل از پیاده سازی (نمایش داده) تعریف می شود.

مثال : نوع داده انتزاعی STACK

۱- مشخصات
- 1- تعریف :ساختمان داده ای که سازمان LIFO دارد.
- ۲- عملیات : شامل PUSH وPOP

۲- پیاده سازی در زبان های مختلف متفاوت است و ربطی به مشخصات ندارد.

برای اولین بار در این زبان امکان تعریف داده های جهانی که به کامپیوتر خاصی وابسته نباشد مطرح شد یعنی طراحی انواع داده ای که مشخصات کلی آن کاملا انتزاعی تعریف می شود و به زبان خاصی وابسته نیست.

۳- امکان تعریف نوع داده جدید توسط کاربر

برای اولین بار به کاربر اجازه داده شد که خودش یک نوع داده جدید تعریف و در محدوده برنامه استفاده کند.

```
Type
      Letter = ‘a’..’e’;
Var
      ch : letter;
      X: integer;
```

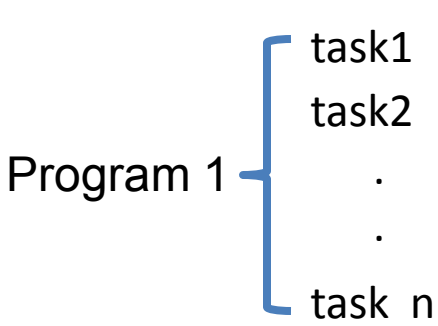
تعریف نوع داده جدید

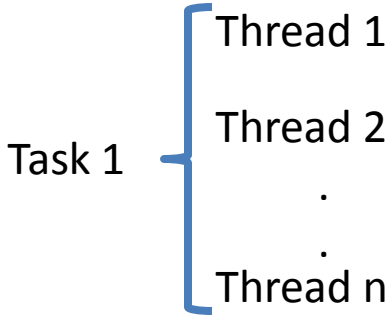
Letter مانند integer است

سوال : تعریف نوع داده جدید توسط کاربر شبیه کدام قابلیت در زبانهای جدید است؟

۴- امکان برنامه نویسی موازی (concurrent)

یک بحث سیستم عاملی است . برای اولین بار این زبان بحث برنامه نویسی موازی وقـرار دادن آن در سلسـله مراتـب زبان را مطرح کرد .امروزه سیستم عامل هایی وجود دارند که قابلیت اجرای چندین برنامه را دارا هستند .





موازی بودن به این معناست که همزمان برنامه ۱ و برنامه ۲ را در نظر بگیرید. Task1 از برنامه ۱ اجرا شده بعد task1 از برنامه ۲ اجرا میشود و این سوئیچ بین برنامه ها ادامه می یابد تا هر دو برنامه به اتمام برسد در این صورت ***چنین به نظر میرسد*** که دو برنامه هجرای همزمان دارند.

سیستم عامل هایی که از multiprogramming پشتیبانی می کنند اجرای موازی را برای برنامه نویسان فراهم میکنند.

زبانهای برنامه سازی معمولا ساختاری برای پشتیبانی از اجرای موازی ندارند اما در زبان Ada ساختار and برای پشتیبانی از اجرای موازی وجود دارد :

Statement 1 **and** statement 2 **and … and** statement n

ساختار به این مفهوم است که دستورات فوق به صورت موازی اجرا شوند.

مثال :

```
x=10 and y=a+b;
```

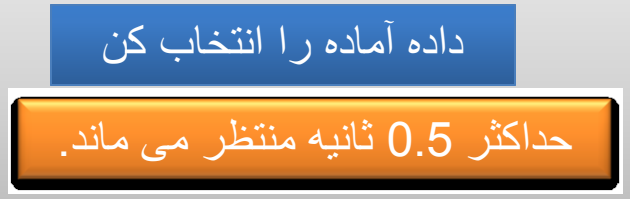
نکته : مسائلی در اجرای همزمان وجود دارد که در صفحه ۳۳۹ کتاب بررسی شده است.

۴- امکان برنامه نویسی بلادرنگ (Real time)

برنامه ای که در یک دوره زمانی خاص باید با دستگاههای I/O یا سایر task ها تعامل داشته باشد.
یعنی زبان به ما اجازه میدهد که به کمک یک ابزار task ها را زمانبندی کنیم که چقدر برای یک task دیگر منتظر بمانند.

مثال : در زبان Ada با ***ابزار زیر*** می توان این کار را کرد:

```
Select    DataReady;
Or    delay    0.5;
End    select;
```



۵- مفهوم package :

همان مفهوم زیر برنامه را در پاسکال دارد.

- **زبان JAVA** : از میان زبانهایی با تکنولوژی جدید زبان java را بررسی می کنیم .

جاوا یک زبان برنامه نویسی است که در سال ۱۹۹۵ توسط شرکت sun معرفی شد. گرامر زبان مشتق از c و شی گرایی آن از c++ گرفته شده است.

با توجه به اینکه طراحی یک زبان جدید با وجود زبان های موجود دلیلی ندارد سوال این است که

چرا java ایجاد شد؟

در سال ۱۹۹۱ شرکت sun سفارشاتی برای نوشتن برنامه برای وسایل الکترونیکی مانند تلویزیون،ماشین لباسشویی، ویدئو و ... دریافت می کرد . زبان c و c++ در دسترس بود اما یک مشکل بزرگ وجود داشت.

مشکل این بود که کامپایلر c برای cpu و یا سخت افزار ادوات فوق طراحی نشده بود و آنها مجبور بودند بعد از تهیه برنامه به زبان c یک کامپایلر مخصوص برای همان وسیله الکترونیکی را تهیه کنند تا بتوانند دستورات زبان c را بر روی آن وسیله اجرا کنند.

ادامه داستان را می توان حدس زد . ایجاد کامپایلرها زمان و هزینه زیادی را مصرف می کند . بنابراین ممکن نیست که برای هر platform کامپایلر مخصوص به خود داشته باشیم.

آیا می توان **ماژولی** تولید کرد که برای همه platformها بتواند کد مقصد تولید کند؟
به نظر میرسد برای تولید این ماژول باید روند زیر را اصلاح کرد.

یعنی مشکل در object program است .اگر قرار باشد کامپایلرکد مقصد تولید کند نمی توان مشکل فوق را حل کرد.

راه حل : اگر بتوانیم زبانی تولید کنیم که بتواند با تکنولوژی دیگری غیر از داشتن ***کامپایلرسنتی*** کدی تولید کند که بتواند روی همه محیط ها و همه cpu ها اجرا شود. آن زبان را platform independent گویند.

Platform independent =any cpu + any environment + any operating system

این مساله زمینه ساز ایجاد جاوا شد. جاوا باید روی همه platform ها ازجمله Macintosh، Unix وحتی web اجرا شود .

Platform independent چیست ؟

توانایی زبان برای انتقال کد از یک کامپیوتر به کامپیوتر دیگر بدون کوچکترین مشکل اشاره می کند.

سطوح platform independent عبارتند از :

۱- source level : اگر بتوانیم برنامه نوشته شده برای یک زبان را از کامپیوتر به کامپیوتر دیگر منتقل،کامپایل و اجرا کنیم در این صورت آن زبان در source level دارای خاصیت platform independent است.

۲-binary level : اگر بتوانیم فایل باینری کامپایل شده را روی کامپیوتر دیگر بدون کامپایل مجدد اجرا کنیم در این صورت آن زبان در binary level دارای خاصیت platform independent است.

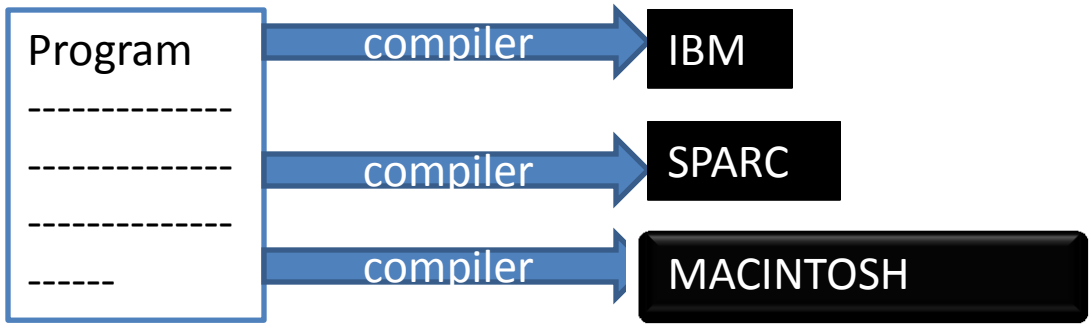
برای رسیدن به binary level دوشرط لازم است :

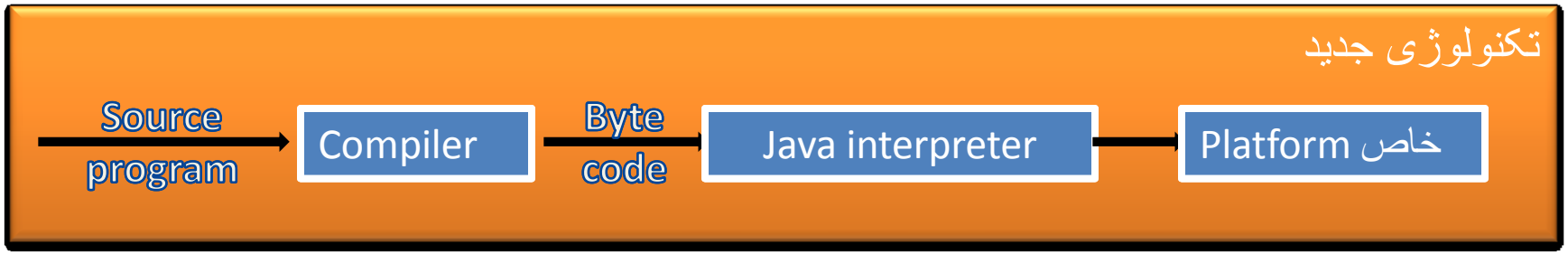
۱- انواع داده ای java باید در میان همه platformها جهانی و سازگار باشد .

۲- تکنولوژی پیاده سازی زبان باید تغییر نماید یعنی دیگر نمی توان از کامپایلر سنتی با همان ماژولها و حتی مفسر سنتی با همان ماژولها استفاده کرد .بلکه نیاز به یک java virtual machine داریم که حالت خاصی از یک مفسر است .

کامپایلر مجموعه دستورات را به زبان ماشین یا به دستورات پردازنده تبدیل می کند که مخصوص همان cpu است
.اگر بخواهیم این کد را در سایر سیستم ها اجرا کنیم باید
۱- یک کامپایلر برای آن سیستم بنویسیم .
۲- برنامه را مجددا کامپایل کنیم .

شکل زیر اجرای برنامه در زبانهای غیرجاوا را روی سیستم های مختلف نشان می دهد .





۱-کامپایلر جاوا : source code را به byte code تبدیل می کند که machine independent (مستقل از ماشین) است byte code ها مجموعه ای از دستورالعمل های جاوا هستند که به byte شکسته شده اند و به وسیله هر cpu می توانند رمز گشایی شوند .مشخص است که این کامپایلر با کامپایلر های زبان های کامپایلری متفاوت است .

۲- مفسر جاوا : که JVM یا مفسر زمان اجرای جاوا نام دارد byte code ها را اجرا می کند .البته لازم به ذکر است که مفسر جاوا با مفسر زبان های مفسری متفاوت است.

Jvm چییست ؟

مفسر جاوا یا jvm یک شبه سیستم عامل است که byte code را دریافت کرده و آن را به مجموعه دستورالعملهای مخصوص همان platform برای اجرا تبدیل می کند یعنی jvm یک سیستم زمان اجرا ایجاد می کند که باعث اجرای کد در زمان اجرا می شود.

برای آشنایی با تکنولوژی جدید ماژولهای کامپایلر و مفسر را در جاوا با کامپایلرها و مفسرهای سنتی از نظر مراحل کار ***مقایسه*** می کنیم.

۱- کامپایلر سنتی :

کامپایلر ها source code را به object code تبدیل می کنند و فقط بر روی همان platform اجرا می شوند .

فرآیند کامپایل شامل ۴ مرحله عمده است .

ویژگی های برنامه های کامپایل شده

۱- سرعت اجرای بالا

۲- عدم نیاز به کد منبع در زمان اجرا

۳-عدم نیاز به کامپایلر در زمان اجرا

۴-امکان بهینه سازی کد کامپایل شده

۲- مفسر سنتی (interpreter) :

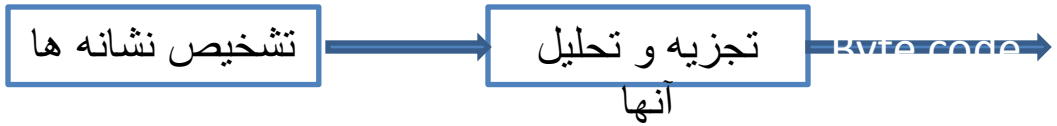
مفسرها دستورات برنامه را خط به خط تفسیر می کنند . فرآیند تفسیر برای هر دستور شامل مراحل زیر است :

مشکل مفسر : بهینه سازی کد وجود ندارد و کند است.

نکته مهم : در زبان جاوا به دلیل اینکه هم کامپایلر و هم مفسر دارد کار ۴ مرحله ای کامپایلرسنتی شکسته شده و در دو زمان کامپایل و اجرا به ترتیب توسط کامپایلر جاوا و مفسر جاوا انجام می شود .

۳- کامپایلر جاوا :

کامپایلری ۲ مرحله ای است . در زمان کامپایل از مراحل ۴ گانه کامپایل فقط ۲مرحله **تشخیص نشانه ها** و **تجزیه و تحلیل** آنها انجام شده و تا حد امکان(مختصر) بهینه می شود . و محصول آن **byte code است** .



۴- مفسر جاوا :

بر روی byte code عمل می کنند . بنابراین نسبت به مفسر های معمولی (که برای هر دستور ۳ مرحله تشخیص نشانه،تجزیه و تحلیل و اجرا را دارد) *فقط مرحله اجرا* را دارد.

یعنی **تشخیص نشانه** و **تجزیه و تحلیل** آنها در مرحله کامپایل و در زمان ترجمه صورت می گیرد. به همین دلیل مفسر جاوا از مفسرهای سنتی سریع تر است و روی هر platform ی اجرا می شوند.

مفسر جاوا هر byte code را به دنباله ای از دستورالعمل های زبان ماشین تبدیل کرده و اجرا می کند.

ویژگی های برنامه های تفسیر شده

۱- سرعت کم اجرا نسبت به کامپایلر

۲- قابلیت حمل

۳- نیاز به ماشین مجازی برای اجرا

نکته : وقتی از تکنولوژی جاوا استفاده کنیم به دلیل platform independent بودن سرعت اجرای بالا را از دست می دهیم .چون byte code کد اجرایی مانند کد object نیست و مفسر برای اجرا باید byte code را در ***حین اجرا*** به دستوالعمل های زبان ماشین تبدیل کند و این زمان گیر است.

کامپایلر JIT(just in time)

برای افزایش سرعت اجرای برنامه ها در جاوا کامپایلر جاوا طوری تغییر کرده است که بتواند در صورت لزوم ***بعضی از Byte code ها*** را در حین کامپایل به دستورالعمل های زبان ماشین تبدیل کند.البته برای باقی ماندن خاصیت platform independent این دستورالعمل ها در حافظه موقت ذخیره میشوند.

مثلا در توابعی که چند بار فراخوانی می گردد اولین باری که تابع فراخوانی می شود به وسیله کامپایلر JIT ترجمه شده و در حافظه موقت ذخیره می شود و هر گاه تابع فراخوانی شود کدهای زبان ماشین اجرا میشوند و سرعت بالا می رود.(۵۰ بار سریع تر)

این دسته از کامپایلر ها برای برنامه هایی که دنباله ای از دستورالعمل ها را بارها تکرار می کنند مناسب هستند.

این دسته از کامپایلر ها نمی توانند بهینه سازی را برای برنامه ها درست اعمال کنند.

نکته مهم : مفسری یا کامپایلری بودن یک زبان جزو ویژگی های ذاتی یک زبان نمی باشد.مثلا برای زبان کامپایلری C می توان مفسر نوشت.

شعار جاوا : یک بار کد بنویسید ولی آن را در هر محیط ، هر زمان و n مرتبه اجرا کنید .

خصوصیات زبان جاوا :

۱- simple :

- یادگیری زبان آسان است.گرامر آن مشتق شده از زبان c و شی گرایی آن از c++ گرفته شده است.
- خصوصیات پیچیده c حذف شده است . Pointer، goto ، header file، struct و union ندارد.

۲- object oriented

۳- platform independent

۴- Robust : (قوی)

- در زبان جاوا انواع داده ای نیاز به اعلان صریح دارند و این قدرت زبان را از نظر سرعت بالا می برد.

-هم مفسر دارد و هم کامپایلر بنابراین کد برنامه را ۲ بار چک می کند و انواع مشخصی از error ها را حذف می کند.

- فاقد pointer است و تمام دسترسی ها به آرایه و رشته ها را در زمان اجرا بررسی می کند که bind آنها صحیح باشد .(امنیت)

-مدیریت حافظه به طور اتوماتیک انجام می شود و در اختیار برنامه نویس نیست تا مشکلات حافظه زباله و ارجاع سرگردان به وجود آید در ضمن جاوا به طور اتوماتیک حافظه زباله را در اختیار سایر اشیا قرار میدهد.

۵-secure :

جاوا دارای چندین لایه کنترل امنیت است .

-اولین لایه : data و method در کلاس ها encapsulate شده اند که فقط از طریق همان کلاس قابل دسترسی هستند.

- لایه دوم : فاقد اشاره گر است و اجازه دسترسی مستقیم به حافظه را نمی دهد. سرریز آرایه را مجاز نیست و در نتیجه از خوانده شدن حافظه خارج از محدوده جلوگیری می کند.

-لایه سوم : کامپایلر ایمنی کد را بررسی می کند.

- لایه چهارم : مفسر جاوا byte code را قبل از اجرا برای بار آخر بررسی می کند.

۶-distributed :

جاوا می تواند در یک محیط شبیه اینترنت که platform های متفاوتی دارد استفاده شود .

۷- multithreading :

جاوا برای اجرای همزمان task ها از multithreading پشتیبانی می کند .

۸- dynamic :

جاوا برای افزایش انعطاف مقدار زیادی ساختمان داده زمان اجرا برای معتبرسازی و دسترسی به object های زمان اجرا با خود حمل می کند .

۲- زبان های تجاری :

این زبان ها برای کارکرد در دنیای تجاری معرفی شدند تا بتوانند نیازهای دنیای تجارت ازجمله نرم افزارهای مالی را پوشش دهند.

-زبان Cobol : (common business oriented language)

خصوصیات :

۱- برای پردازش فایل ها به وجود آمد و هم اکنون برای کارهای تجاری با حجم زیاد مورد استفاده قرار میگیرد.

۲- قادر به انجام عملیات روی کاراکتر های الفبایی مثل(نام-آدرس و سایر مشخصات)می باشد.

۳- محدودیت ان این است که برای انجام عملیات پیچیده ی ریاضی مناسب نیست.

۳- زبان های پیشرفته :

- زبان های هوش مصنوعی : مانند prolog ، xip ، snobol و clips که با داشتن گزاره های منطقی برای انجام پروژهای هوش مصنوعی مناسب است .

۴- زبان های سیستمی

مانند زبان c و c++

عوامل تکامل زبانهای برنامه سازی :

۱- پیشرفت در سخت افزار و سیستم عامل :
با توجه به مفهوم سلسله مراتبی زبان با پیشرفت در لایه های درونی زبان بعضی از وظایف زبان برنامه نویسی به سیستم عامل واگذار می شود که باعث می شود زبان با تکیه بر سیستم عامل راحتتر طراحی شود .

۲- مطرح بودن کاربردهای خاص :
پیشرفت در تکنولوژی کامپیوتر باعث پیشر فت در زبانها شد به عنوان مثال شاخه هوش مصنوعی باعث به وجود آمدن زبان prolog شد.

۳- پیدایش متد های مختلف برنامه نویسی :
به عنوان مثال متدلوژی شی گرایی باعث به وجود آمدن زبان small talk شد.

۴- مطالعات تئوری : تحقیق درباره مفاهیم طراحی زبانها باعث پیشرفت در زبانها می شود.

دلایل موفقیت یا عدم موفقیت یک زبان :

۱- دلایل خارجی : (یعنی مربوط به خصوصیات ذاتی زبان نیست)

الف- اجبار در استفاده از یک زبان خاص تحت تاثیر قوانین به عنوان مثال Cobol در امریکا

ب- حمایت شرکتهای بزرگ : مانند حمایت IBM از زبان Fortran.

ج -استفاده دانشجویان از یک زبان خاص مثل Pascal

۲- دلایل داخلی : (مشخصات یک زبان برنامه نویسی خوب)

الف - وضوح ، سادگی، خوانایی، قابلیت اعتماد (clarity , simplicity ,readability ,Reliability)

- شکل ظاهری یک دستور بیانگر نوع عملی باشد که انجام می دهد .مثلا شکل عمل جمع + باشد

C=a+b; c=a#b;

- ساختارهایي که معاني آنها متفاوت است شکل ظاهري آنها با هم فرق داشته باشد یعنی دو جمله مشابه ، معنایی متقاوت نداشته باشند . مثلا + هم علامت جمع و هم علامت xor نباشد .

تذکر- زبانهایی که گرامر مختصر و رمزی دارند برنامه نویسی آسان است ولی قابلیت خوانایی پایین است.

۲- امکان ترکیب (orthogonality) : یعنی بتوان ویژگی های مختلف یک زبان را با هم ترکیب کرد و ترکیب حاصل با معنا باشد . مثلا بتوان به کمک < ، > ،= ، + و... یک عبارت محاسباتی نوشت.
مثال :

Int I;	ساختار اول
For (I=0; I<10 ; I++)	ساختار دوم
For (int I=0; I<10 ; I++)	ترکیب دو ساختار

نکته ۱: زبانهایی مانند جاوا و c++ که اجازه declaration را در بین جملات قابل اجرا می دهد orthogonal تر از زبانی مانند پاسکال است که این امکان را نمی دهد.

نکته ۲: زبانهای تابعی مانند lisp (مرتب از توابع تودرتو استفاده می کنند)و زبانهای شی گرا(که امکان استفاده مشترک از متدها وجود دارد) از درجه orthogonality بالایی برخوردار هستند.

مزیت : یادگیری زبان ساده می شود زیرا حالت های استثنا کم میشوند.

عیب : ممکن است برنامه شامل ترکیبی از ویژگیهایی باشد که از نظر منطقی روشن نباشد ولی برنامه بدون خطا کامپایل شود.

۳- پشتیبانی از انتزاع (support for Abstraction) :

انتزاع به مفهوم ***پنهان سازی پیچیدگی*** از دید کاربر است . هر چه ابزارهای انتزاع بیشتر باشد زبان برنامه سازی انتزاعی تر است .

به عنوان مثال توابع کتابخانه ای باعث سهولت در نوشتن برنامه و پنهان سازی پیچیدگی می شوند هرقدر که این توابع تنوع بیشتری داشته باشند زبان انتزاعی تر است.

Sqrt (x);

۴- قابلیت حمل (Portability) :

زبانی که به ماشین خاصی وابسته نباشد و روی هر platform اجرا شود .این مورد قبلا بررسی شده است .

۵- سهولت در تست برنامه :

الف – به صورت دستی (manual)

ب – با داده های آزمایشی (sample data)

۶- محیط برنامه نویسی (programming environment) :

یک محیط برنامه نویسی قدرتمند کاربرد یک زبان را نسبت به زبان دیگری که قدرت تکنیکی بالاتری دارد بیشتر می کند.

منظور ازمحیط برنامه نویسی

۱- پیاده سازی قابل اعتماد و کارآمد

۲- راحت بودن تست و عیب یابی

۳-امکانات برای اصلاح برنامه

۷- هزینه (cost) :

الف – هزینه اجرا (Run) : یعنی تا جایی که ممکن است برنامه سریع تر اجرا شود .سرعت همیشه در اولویت نیست.

ب- هزینه ترجمه (compile) : یعنی تا جایی که ممکن است برنامه سریع تر کامپایل شود .به عنوان مثال در یک زبان آموزشی برنامه دانشجویان چندین بار کامپایل و فقط یک بار اجرا می شود.پس کامپایل در این جا در اولویت است.

ج- هزینه نگهداری برنامه (maintenance) :
زبان باید ابزاری برای نگهداری برنامه ها بعد از اجرا داشته باشد .
به عنوان مثال در صورت تغییر در سخت افزار و سیستم عامل برنامه همچنان قابل اجرا باشد .
در صورت تغییر در نیازمندیهای برنامه بتوان با هزینه کم از برنامه نوشته شده استفاده کرد.

مفهوم استاندارد سازی (standardization) : برنامه نوشته شده به یک زبان برنامه سازی بعد از انتقال به کامپیوتر دیگر باید :
۱- بدون مشکل اجرا شود. (portability)
۲- همان نتیجه را تولید کند .
در صورتی که ***همان نتیجه را تولید کند*** گفته می شود هر دو کامپایلر دارای ***پیاده سازی های یکسان*** بوده اند.

تعریف استاندارد سازی : بین پیاده سازی ها ی مختلف زبان یکنواختی ایجاد می کند .

سوال – اگر بتوانیم به کامپایلر زبان c استاندارد یک قانون اضافه یا کم کنیم آیا زبان حاصل همان زبان c است؟

پاسخ منفی است .

مفهوم زبان برنامه سازی به یک ***پیاده سازی خاص*** مربوط می شود. ممکن است کامپایلرهای مختلف از یک برنامه تفسیرهای متعددی ارائه دهند .یعنی یک برنامه واحد روی کامپایلر های مختلف دو نتیجه مجزا داشته باشند . در این صورت زبان استاندارد نشده است .

مسائلی که در استاندارد سازی باید در نظر گرفت :
۱- بعد از مدت کوتاهی از اولین پیاده سازی، زبان استاندارد شود.
۲- برنامه های سابق نوشته شده روی استاندارد جدید اجرا شود یعنی کامپایلر پیرو باشد .(conformance)

استانداردهای زبان Fortran : (پیاده سازیهای زبان Fortran)

Fortran 1	**۱۹۵۸**
Fortran 2	**۱۹۶۰**
Fortran 4	**۱۹۶۶**
Fortran 77	**۱۹۷۷**

چه عواملی بر طراحی زبان اثر می گذارند ؟

۱- محیط عملیاتی (سیستم عامل)

۲- محیط برنامه نویسی (کامپایلر)

۳- سخت افزار ماشین

برنامه ۱- محیط عملیاتی : محیطی که ***اجرای*** برنامه را پشتیبانی می کند . قبل از طراحی زبان طراح باید بداند های این زبان قرار است روی چه ***محیط عملیاتی اجرا*** شود.

محیط عملیاتی روی ویژگیهای زیر در طراحی زبان اثر می گذارد :

۱- ویژگی I/O
۲- ویژگی پردازش خطا
۳- امکانات زمان بندی (زمان اجرا)
۴- امکانات ساختار برنامه

انواع محیط های عملیاتی

۱- محیط پردازش دسته ای (Batch processing)
۲- محیط پردازش محاوره ای (Interactive)
۳- محیط پردازش ادغام شده (Embedded)

۱- محیط پردازش دسته ای (Batch processing)
تعدادی از برنامه ها به صورت گروهی و پشت سر هم اجرا می شوند .
تاثیر عوامل فوق را در این محیط بررسی می کنیم .

ویژگی I/O : فایلهای متداولترین وسیله I/o هستند .

ویژگی پردازش خطا : امکان کنترل خطا توسط کاربر وجود ندارد زیرا پردازش دسته ای خواهد بود .

امکانات زمان بندی : زمان اجرای برنامه بحرانی نیست .

امکانات ساختار برنامه : هر برنامه شامل یک Main اصلی و تعدادی زیر برنامه است .

۲- محیط پردازش محاوره ای (Interactive) : برنامه در حال اجرا با کاربران تعامل مستقیم دارد .

ویژگی I/O : ورودی مستقیما از کاربر گرفته می شود.

ویژگی پردازش خطا : امکان کنترل خطا در زمان اجرا توسط کاربر وجود دارد .

امکانات زمان بندی : زمان اجرای برنامه بحرانی است .

امکانات ساختار برنامه : در محیط های ***محاوره ای کامل*** معمولا مفهوم برنامه اصلی وجود ندارد.

۳- محیط پردازش ادغام شده (Embedded)

محیط هایی هستند که برنامه های نوشته شده در آن باید قادر به کنترل سخت افزار مثلا هواپیما،ماشین آلات و ... باشند .

ویژگی I/O : چون برنامه باید سخت افزار خاص را کنترل کند معمولا **I/O** آنها استاندارد نیست یعنی وابسته به دستگاه خاص است .

ویژگی پردازش خطا : هر برنامه باید بتواند در حضور خطا به عملکرد صحیح خود ادامه دهد یعنی fault tolerance باشد و بعد از رخداد خطا فعالیت مناسبی برای ترمیم خطا و ادامه کار انجام دهد.

امکانات زمان بندی : زمان اجرای برنامه فوق بحرانی است .

امکانات ساختار برنامه : مفهوم برنامه اصلی وجود ندارد زیرا برنامه بر روی ***یک سیستم توزیع شده*** که هر کدام بخشی از سیستم را کنترل و نظارت می کنند اجرا می شود.

۲- محیط برنامه نویسی (کامپایلر) :

محیطی است که برنامه در آن تست و اشکال زدایی می شود . شامل امکاناتی از قبیل debugger ، editor و ...می باشد .

قبل از طراحی زبان طراح زبان باید به طور کامل کامپایلر را شناخته و بتواند با توجه به هدف زبان به همه ریزه کاری های کامپایلر آشنا باشد به عنوان مثال چند مورد را بررسی می کنیم .

سوال ۱ :کامپایلر چگونه یک برنامه بزرگ را کامپایل می کند؟

یعنی ترتیب ترجمه چگونه است؟ چون یک برنامه دارای ماژول های زیادی است . به برنامه زیر دقت کنید .

```
main() {
    .....
}
f1(a,b) {
int a,b;
F2(a)
}
f2(c) {
int c;
return (c*c);
}
```

۱- کامپایل به صورت ترتیبی باشد و هر زیر برنامه به ***طور مستقل*** کامپایل شود .

مشکلات:

الف- کامپایلر به اطلاعاتی درباره زیربرنامه ها و متغیرهای مشترک نیاز دارد.

ب- پارامترهای مشترک و متغیرهایی که در *زیر برنامه های بعدی* تعریف شده اند برای کامپایلر ناشناخته هستند .

ج- تخصیص حافظه برای زیربرنامه های بعدی انجام نشده است .

۲- هر زیر برنامه باید self-contained (خودکفا) باشد یعنی هر چیزی که نیاز دارد را با خود داشته باشد یعنی در هر زیربرنامه عناصر مشترک و تعاریف نوع مجددا تعریف شوند . این کار به عهده برنامه نویس است . مانند Fortran استاندارد

۳- قبل از کامپایل شدن هر زیربرنامه تمام داده های مشترک مربوط به همان زیربرنامه کامپایل شوند . یعنی ابتدا زیربرنامه ها و داده های مشترک کامپایل شوند. مانند Ada و پاسکال

۴- با توجه به اینکه هر برنامه دارای دو بخش است:

الف – مشخصات (Specification)

ب - بدنه (Body)

کامپایلر کتابخانه ای ایجاد می کند و ابتدا کل مشخصات همه زیربرنامه ها را کامپایل کرده و در کتابخانه زمان اجرا قرار می‌دهد و بعد بدنه برنامه کامپایل می‌شود. (مثل: C , C++)

```
Var

    a : integer;
```

سوال۲ - کامپایلر چگونه با اسامی مشترک برخورد می‌کند. دامنه شناخت متغیرها تا چه اندازه است؟

۱- هر نام در زیربرنامه باید منحصر به فرد باشد .برنامه نویس مسوول این کار است.

۲- استفاده از قوانین خاص (scoping rules) مانند تعریف متغیر های local و global در پاسکال

۳- استفاده از کلاسهای حافظه مانند زبان c

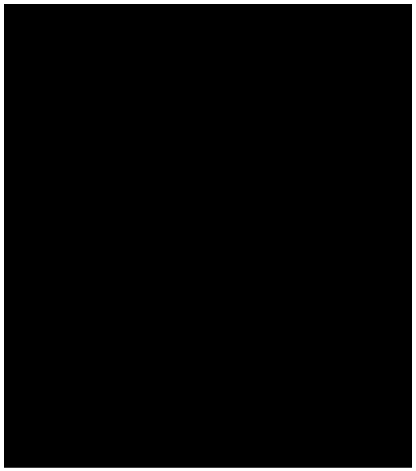
-auto : متغیرها فقط در همان تابع قابل شناسایی هستند.

-Extern : خارج از main تعریف می شوند.

-Static :
۱- محلی : فقط یک بار مقدار اولیه می گیرند.
۲- عمومی : از محلی که تعریف شده اند به بعد قابل دسترسی هستند.

- Register : مقدار متغیرها در رجیسترها قرار می گیرد.

۴-استفاده از external library در زبانهای شی گرا



الف- وقتی در زیربرنامه ای از کلاسی استفاده می شود که تعریف شده است سایر اشیای تعریف شده آن کلاس شناخته شده می شوند.

ب- متغیرها می توانند public یا private باشند .

تمرین- زبان با دستورات زیر را در نظر بگیرید :

```
a = b;
a = a + 1;
a = a – 1;
If  a = 0 then goto l;
If  a>0   then goto l;
Goto   l;
Halt;
```

۱- برنامه ای بنویسید که a=a+b را شبیه سازی کند.

۲ – برنامه ای بنویسید که a=a*b را شبیه سازی کند .

تمرین - آیا در زبان C این ویژگی که می‌توان بسیاری از مفاهیم را به روشهای گوناگون بیان کرد ویژگی مناسبی است یا خیر؟

پایان فصل اول

جواب شبیه سازی

a= a+b;

```
L : a= a+ 1;
    b= b-1 ;
    if    b>0 goto    l;
    halt;
```

فصل دوم
اصول طراحی زبان

اصول طراحی زبان
از میان عوامل سه گانه محیط عملیاتی، محیط برنامه نویسی و سخت افزار ماشین که در طراحی زبان تاثیر دارند مورد سوم را بررسی می کنیم .

سخت افزار ماشین
طراحی زبان باید با علم به سخت افزار کامپیوتر انجام شود زیرا در نهایت امر این سخت افزار است که ***عملیات*** را انجام میدهد.

تذکر- باید توجه داشت عملیات تعریف شده توسط زبان هم می تواند به کمک امکانات سخت افزاری انجام شود و هم می تواند به صورت ***نرم افزاری شبیه سازی*** شده باشد .

ادعا این است که مراحل طراحی زبان با مراحل طراحی یک کامپیوتر مشابه است . هدف این است که با مقایسه مراحل طراحی کامپیوتر و زبان صحت این ادعا را بررسی کنیم .

مراحلی که در طراحی یک کامپیوتر انجام می شود

۱- طراحی داده ها : چگونگی نگهداری داده ها به وسیله سخت افزار در حافظه مانند نگهداری int و نوع داده اعشاری با ممیز شناور

۲- تعریف عملیات :
الف- شامل طراحی الگوریتم های چهار عمل اصلی ریاضی و مقایسه ای و مدارات مربوطه

ب – طراحی مراحل چرخه اجرای دستورالعمل :

نکته مهم – فرآیند رمزگشایی پیچیده و زمان بر است .

۳- **کنترل ترتیب اجرا** : به وسیله رجیستر cs : ip در پردازنده ترتیب اجرا کنترل می شود .در حالت عادی دستورات به ترتیب اجرا می شود که دلیل آن همین رجیستر است . دستورات شرطی باید به نوعی محتوای همین رجیستر را تغییر دهند .

مراحلی که در طراحی یک زبان برنامه سازی انجام می شود

۱- **طراحی داده ها** : یک زبان برنامه سازی باید مجموعه انواع داده ای ممکن برای زبان را تعریف کند مانند نوع داده صحیح ، نوع داده اعشاری و ....

۲- **عملیات تعریف شده روی انواع داده ای :** این عملیات شامل چهار عمل اصلی ریاضی و عملیات مقایسه ای

نکته : این عملیات با استفاده از امکانات سخت افزاری قبلا تعریف شده است برای عملیات پیچیده تر با استفاده از توابع کتابخانه ای به صورت نرم افزاری انجام می شود .

۳- **کنترل ترتیب اجرا :** زبان های برنامه سازی باید مکانیزم هایی را برای کنترل ترتیب اجرا و فراهم کنند.

۴- مکانیزمی برای تخصیص حافظه : زبان های برنامه سازی باید روشی برای ارتباط زبان با حافظه را فراهم کنند.

همانطوری که ملاحظه می شود مراحل طراحی کامپیوتر با مراحل طراحی زبان یکسان است . از این رو وقتی طراح کامپیوتر در حال طراحی کامپیوتر است عملا دارد زبانی تولید می کند که بر روی سخت افزار کار کند .

از طراح کامپیوتر می خواهیم که همان موقعی که در حال طراحی کامپیوتر است زبان سطح بالا را روی آن تولید کند به عبارت دیگر زبان ماشین کامپیوتر یک زبان سطح بالا باشد.

سوال - آیا می توان کامپیوتری تولید کرد که زبان ماشین آن زبان سطح بالا باشد؟

پاسخ: بله، مثلاً کامپیوترهایی طراحی شده که زبان ماشین آن زبان **C** است.

از دیدگاه بالا دو نوع کامپیوتر وجود دارد :

۱- کامپیوتر واقعی : زبان ماشین آن زبان اسمبلی است.
۲- کامپیوتر مجازی : زبان ماشین آن زبان سطح بالا است.

تمرین: در مورد مزایای کامپیوتر واقعی بر کامپیوتر مجازی بحث کنید.

سوال – چرا روش پیاده سازی زبان ها با استفاده از روش فوق یعنی عدم استفاده از کامپایلر نیست ؟

پیچیدگی سخت افزاری افزایش می یابد .

روشهای پیاده سازی زبانها

۱- ترجمه (کامپایل) :

-کامپایلر : یعنی زبان سطح بالا به زبان ماشین تبدیل شود .
-اسمبلر : یعنی زبان اسمبلی به زبان ماشین تبدیل شود.
- loader : آدرس های زبان ماشین را تنظیم می کند .
- preprocessor : دستورات سطح بالای توسعه یافته را به دستورات سطح بالای استاندارد تبدیل می کند .

۲- تفسیر (شبیه سازی) : به جای ترجمه کامل برنامه برنامه خط به خط تفسیر می شود .

۳- کامپیوتر مجازی : ساخت ماشینی که زبان ماشین آن زبان سطح بالا است .

۴- استفاده از تکنولوژی کامپایلر تغییر یافته و مفسرتغییر یافته با هم در زبان های جدید

مقایسه ترجمه و شبیه سازی

از دیدگاه سرعت : یک برنامه با یک حلقه تکرار را در نظر بگیرید .

```
While   a>10   do
Begin
b = b + 1 ;
-------------
-------------
-------------
End.
```

چون فرآیند رمزگشایی پیچیده است در حالت ترجمه دستورات داخل حلقه فقط یک بار رمزگشایی می شوند و به تعداد دفعات مشخص تکرار می شوند .یعنی سرعت افزایش می یابد . ولی در حالت شبیه سازی این کار امکان پذیر نیست .

از دیدگاه انعطاف : شبیه سازی انعطاف را افزایش می دهد .

از دیدگاه مصرف حافظه : شبیه سازی کدی را در حافظه نگهداری نمی کند و مصرف حافظه کمتری دارد .

سوال – برای پیاده سازی زبان و اجرای کارآمد برنامه ها چه اعمالی را در زمان ترجمه و چه اعمالی را در زمان اجرا انجام دهیم ؟

۱- اگر کارها را به کامپایلر واگذار کنیم باعث پیچیدگی کامپایلر می شود .

۲- اگر کارها در زمان اجرا انجام شود نیاز به ساختمان داده زمان اجرا داریم و کارها کند انجام می شود.

Binding and Binding time

تعریف binding : (محدود کردن) انتخاب یک ویژگی از میان مجموعه ای از خواص برای هر یک از عناصر برنامه نویسی را binding یا انقیاد گویند .

مثال ۱: `int x;`

یعنی x به نوع صحیح bind شده است یعنی از میان مجموعه انواع داده ای ممکن در این زبان نوع integer برای این متغیر استفاده شده است .

مثال ۲ : `x = 5;`

یعنی از میان مجموعه مقادیر ممکن برای متغیر x که می تواند شامل بازه قابل قبول برای int که بین (۳۲۷۶۸- تا ۳۲۷۶۷+) است مقدار ۵ انتخاب شده است به عبارت دیگر x به مقدار ۵ bind شده است .

تعریف binding time : زمان هایی که می توانیم بر روی زبان محدودیتهای فوق را انجام دهیم زمان های binding گویند.

نکته مهم – انتخاب زمان های binding بر روی رفتار زبان تاثیر مستقیم دارد .

انواع زمان های **binding**

۱- زمان تعریف زبان (define time) : یعنی محدودیت بر روی عناصر زبان هنگام تعریف زبان توسط طراح زبان صورت می گیرد . مانند تعریف مجموعه انواع داده ای ممکن در زبان

۲- زمان پیاده سازی زبان (implement time) : یعنی محدودیت بر روی عناصر زبان هنگام پیاده سازی زبان توسط طراح صورت می گیرد. زمان پیاده سازی زمانی است که طراح خصوصیات زبان را در **حافظه نمایش** می دهد . مانند مشخص کردن نمایش نوع داده اعشاری در زبان

۳- زمان ترجمه (compile time) : یعنی محدودیت بر روی عناصر زبان هنگام ترجمه زبان توسط برنامه نویس به کمک ابزاری که طراح قرار داده است صورت می گیرد . مانند محدود کردن متغیر x به نوع int هنگام نوشتن برنامه

۴- زمان اجرا (Run time) : یعنی محدودیت بر روی عناصر زبان هنگام اجرا صورت می گیرد یعنی طراح زبان را طوری طراحی کرده است که محدودیت در دیرترین زمان یعنی اجرا انجام شود .مانند محدود شدن متغیر به مقدار مشخص x =18;

نکته ۱ : دو زمان تعریف و پیاده سازی مربوط به طراح زبان است و دو زمان ترجمه و اجرا هنگام نوشتن برنامه معنا دارد .

نکته ۲ : نمودار زمانی binding ها از زمان تعریف شروع و در زمان اجرا به پایان می رسد .

نکته ۳: واژه binding زودهنگام برای زمان ترجمه و واژه binding دیرهنگام برای زمان اجرا به کار می رود . مگر اینکه ذکر گردد که منظور از binding زود هنگام زمان تعریف است .

مثال : فرض کنید دستور زیر در یک زبان برنامه سازی نوشته شده است انواع binding ها را بررسی نمایید .

X=x+10 ;

۱- مجموعه انواع داده ای ممکن برای x :

در زمان تعریف زبان صورت می گیرد .

نکته – در صورتی که زبان امکان تعریف نوع داده جدید را به کاربر بدهد این کار می تواند در زمان ترجمه یا کامپایل هم صورت گیرد .

۲- نوع متغیر x :

در زبان هایی که declaration دارند در زمان ترجمه انجام می شود .

نکته – زبان هایی که declaration ندارند اجازه می دهند نوع متغیر ها در زمان اجرا از روی مقدار آنها مشخص شوند .

۳- مجموعه مقادیر ممکن برای x (بازه قابل قبول برای x) : فرض شود نوع x مشخص شده است .

زمان پیاده سازی مشخص می شود .

۴- مقدار متغیر x :

در زمان اجرا صورت می گیرد .

۵- نحوه نمایش عدد ثابت ۱۰ :

انتخاب نمایش دهدهی برای عدد ثابت ۱۰ در زمان تعریف زبان صورت می گیرد .

۶- نحوه نمایش نماد جمع و انتساب (+ و=) :

در زمان تعریف زبان انجام می شود .

۷- انجام عمل جمع

در زمان اجرا انجام می شود .

۸- type checking : درست بودن نوع متغیرها از نظر نوع عملی که روی آنها انجام می شود.

در زبان هایی که declaration دارند در زمان ترجمه صورت می گیرد.(ایستا)
در زبان هایی که declaration ندارند در زمان اجرا صورت می گیرد .(پویا)

۹- مدیریت حافظه برای متغیر x :

در نظر گرفتن حافظه : به دست آوردن آدرسی نسبی برای اشیا داده

تخصیص واقعی حافظه : به دست آودن آدرس واقعی متغیر بر روی سیستم مقصد

سوال – اگر تخصیص واقعی حافظه در زمان ترجمه صورت گیرد چه اتفاقی می افتد؟

در این صورت زبان ها خاصیت بسیار مهم قابلیت حمل را از دست می دهند به این دلیل که هنگام فروش نرم افزارهای تجاری نسخه اجرایی آن به فروش می رسد که قبلا در جایی کامپایل شده است و عقلانی نیست که حافظه اصلی آن سیستم همراه نسخه تجاری آن به فروش برسد !

بنابراین تخصیص واقعی حافظه قطعا در زمان اجرا خواهد بود .اگر چه با این کار سرعت را از دست می دهیم .

سوال – در هنگام ترجمه چه عملیاتی از نظر تخصیص حافظه انجام می شود ؟

پاسخ - در نظر گرفتن حافظه انجام می شود .
برای آشنا شدن با چگونگی این فرآیند وظایف کامپایلر را بررسی می کنیم .

وظایف کامپایلر

۱- بررسی صحت برنامه
- ۱- lexical analyzer(scanner) :شناسایی token ها یا نشانه ها را انجام میدهد.
- parser-۲ یا به عبارتی بررسی املایی برنامه را انجام می دهد .
  - ۱-syntatic analyzer: بررسی گرامری برنامه
  - ۲- semantic analyzer : بررسی مفهومی برنامه (تعبیر عملگرها)

۲- code generation : تبدیل برنامه به کد اجرایی

در نظر گرفتن حافظه به این شکل انجام می شود که در بخش scanner کامپایلر جدولی به نام symbol table می سازد . این جدول در هنگام ترجمه تشکیل شده تا **در هنگام اجرا وقت کمتری برای تخصیص واقعی حافظه** انجام شود .
این جدول دارای ساختار زیر است :

Symbol table

Name	Type	Size	------	Offset address
a	Int	2	-	@
b	Int	2	-	@+2
c	float	4	-	@+4
d	char	1	-	@+8
e	int	2	-	@+9

```
Var
int     a,b;
float   c;
char    d;
int     e;
```

نکته – خیلی از تفاوت بین زبان ها به دلیل تفاوت در زمان های binding است .

سوال – binding زودهنگام بهتر است یا دیر هنگام ؟

۱- **از دیدگاه سرعت :** هر قدر binding ها زودتر انجام شوند برنامه وقت کمتری را در زمان اجرا برای ایجاد و حذف binding انجام می دهد در نتیجه سرعت افزایش می یابد .

۲- از دیدگاه انعطاف پذیری : هر قدر binding ها دیرتر انجام شوند به دلیل اینکه در زمان اجرا باید برنامه کار ایجاد و حذف binding ها را انجام دهد که باعث افزایش انعطاف می شود .

نمودار زیر نسبت این دو پارامتر را نشان می دهد .

انواع زبانها بر اساس زمان های binding

۱- **زبان های early binding** : زبانهایی که حجم عمده ای از عملیات binding را در زمان ترجمه انجام می دهند . مانند c ، c++ ، pascal

خصوصیات زبان های early

۱- کامپایلر دارند.
۲- declaration دارند .
۳- type checking ایستا
۴- تخصیص حافظه در زمان اجرا
۵- سرعت بالا، انعطاف کم

۲- زبانهای later binding : زبانهایی که حجم عمده ای از عملیات binding را در زمان اجرا انجام می دهند مانند lisp ، ml

خصوصیات زبانهای later

۱- مفسر دارند.

۲- declaration ندارند .

۳- type checking پویا

۴- سرعت کم ، انعطاف بالا

تمرین – چرا زبانهای later قادر به تولید کد exe نیستند ؟

انواع زبان ها از نظر ساختار دستورات

۱- زبان های روالی (procedural) :در این زبان ها یک برنامه دنباله ای از عبارت هاست که اجرای هر عبارت باعث تغییر حالت (state) برنامه می شود .

تعریف حالت (state) : مقدار متغیرها ،رجیسترها و ثابت ها در یک زمان خاص را حالت برنامه در آن لحظه گویند .

با این تعریف برنامه تغییر حالتهای متوالی تا رسیدن به حالت نهایی است .

```
Statement 1 -------→ state 1
Statement 2 ------→  state 2
…
…
Statement n ------→ state n
```

۲- زبانهای تابعی **(functional)** : در این زبانها به جای در نظر گرفتن حالتهای متوالی محاسبات ماشین ،تبدیلات متوالی تابعی را در نظر گرفته می شود که باید بر روی داده های اولیه انجام شوند تا به پاسخ نهایی برسیم. مانند **lisp** یعنی :

الف- هر عمل یک تابع است.

ب- هر تابع به عنوان پارامتر تابع بعدی در نظر گرفته می شود.

fn (---------- f2 (f1 (data))----------)

۳- زبانهای مبتنی بر قاعده **(Rule base)** : این زبانها دارای خصوصیات زیر هستند :

این زبانها شرایطی را بررسی می کنند و در صورت برقراری آنها فعالیتی را انجام می دهند.

زبان دارای فیلترهایی است که بر روی داده ها عمل می کند.

دستورات در این زبانها ترتیبی اجرا نمی شوند .بلکه فعال شدن شرط ها ترتیب اجرا را مشخص می کند.مانند **prolog** و **gpss**

```
Filter1 ----------→ action1
Filter2 ----------→ action 2
----
----
Filter n ---------→action n
```

۴- زبانهای شی گرا **(object oriented)**

این زبانها بر اساس تعریف داده ها در یک ساختار داده به نام کلاس و مشخص کردن عملیاتی که بر روی این داده ها مجاز است اجرا می شود .

```
Class   test {
Int       a,b;
 show();
 get    ();
}
```

پایان فصل دوم

فصل چهارم
اشیا داده و انواع داده ای

عوامل تمایز بین زبانهای برنامه سازی

۱- زمانهای **binding** : ساختارهای اکثر زبانها یکسان است آنچه زبانها را از هم متمایز می کند تفاوت بین زمان انجام کارهاست که بر روی رفتار زبان تاثیر مستقیم دارد . دو زبان که دارای ساختارهای ۱۰۰ درصد یکسان هستند در صورت تفاوت در زمانهای **binding** متفاوت می شوند .

۲- انواع داده ای **(data types)** : دو جنبه در این مورد قابل بحث است

الف- وجود یا عدم وجود یک نوع داده در زبان برنامه سازی رفتار زبان را تغییر می دهد .

ب- نحوه پیاده سازی یک نوع داده در یک زبان قابلیت آن نوع داده و در نهایت قابلیت زبان را تغییر می دهد .

۳-عملیات **(operations)** : انتخاب عملیاتی که بر روی انواع داده ای فوق انجام می شود باعث تغییر در قابلیت زبان می شود مثلا وجود عملیات بیتی خاص در زبان **c** باعث افزایش قابلیت این زبان نسبت به زبانهایی که این عملیات را ندارند می شود.

تعریف Data object (شی داده ای): مکانی از حافظه است که برای نگهداری مقدار داده به کار می رود. به عبارت دیگر مجموعه ای از داده ها که در زمان اجرا به آن حافظه تخصیص یافته است .به شی داده ظرف داده هم گفته می شود.
منظور از حافظه می تواند حافظه اصلی،رجیستر و یا حافظه جانبی باشد.

انواع data object از دیدگاه چگونگی تعریف

۱- programmer defined : این اشیا داده در هنگام اجرای برنامه وجود دارند و توسط برنامه نویس تعریف شده اند.تعریف آنها به وسیله declaration انجام می شود.مانند متغیرها ، آرایه ها و....

۲- system defined : اشیا داده ای هستند که در حین اجرای برنامه توسط سیستم به طور خودکار ایجاد می شوند و در اختیار برنامه نویس نیستند و برنامه نویس از وجود آنها با خبر نیست .مثلا اشیا داده ای که سیستم از آنها برای ذخیره موقت داده ها استفاده می کند مثلا بافر،stack برای اجراب برنامه بازگشتی و ....

temp

A= b+(k*d)

صفات اشیا داده (attributes)

۱- نام ، نوع و مقدار یک شی داده

10000010101

۲- چرخه حیات (life cycle) : از زمانی که به شی داده حافظه داده می شود تا زمانی که حافظه از شی داده گرفته می شود طول عمر یک شی داده را تشکیل می دهد.

انواع شی داده مورد استفاده در یک برنامه

۱- متغیر (variable) : شی داده ای است که binding آن به مقدار در طول دوره حیاتش قابل تغییر است .

۲- ثابت (constant) شی داده ای است که binding آن به مقدار در طول دوره حیاتش غیر قابل تغییر است .

سوال – تفاوت بین متغیر و ثابت از دید کامپایلر چیست ؟

متغیر برای کامپایلر یک شی داده است که در زمان اجرا به آن حافظه تخصیص می یابد ولی ثابت یک شی داده است که کامپایلر در ***زمان ترجمه*** به جای آن مقدار آن را قرار می دهد .یعنی کار کامپایلر ساده می شود .

نکته- اگر کامپایلر در زمان ترجمه مقادیر ثابت را بداند از اطلاعات آنها استفاده کرده و از تولید کد برای یک عبارت پرهیز می کند .یعنی ثابت به کامپایلر در ترجمه کمک می کند .

مثال –
a ,b ثابت و c متغیر است.
کامپایلر قطعا مقدار c را می داند و از تولید کد برای شاخه then جلوگیری می کند.

```
C =a+b+10;
If  c > 10 then
…….
```

۳- لیترال (literal) : نامی است که مقدار عددی یک شی داده را مشخص می کند .یعنی به کمک آن می توان به شی داده حاوی مقدار مراجعه کرد .
Const a=20; "۲۰" لیترالی است که از آن می توان برای مراجعه به شی داده حاوی مقدار ۲۰ در برنامه استفاده کرد.

مثال – نام اشیا داده زیر را مشخص نمایید.

```
const    int  max= 30;
 Int      n ;
N=27;
N=n+ max;
```

n یک متغیر است.
max،۲۷ و ۳۰ ثابت ها هستند.
N ، max ،"۲۷"،"۳۰" نامهایی یرای اشیای داده هستند .
Max نام صریحی برای مقدار ۳۰ است .
"۲۷" لیترالی است که شی داده حاوی ۲۷ را نام گذاری می کند.
ثابت ۳۰ دو نام دارد : نام تعریف شده توسط توسط برنامه نویس یعنی MAX و نام لیترال "۳۰"که از هر دو می توان برای مراجعه به شی داده حاوی مقدار ۳۰ در برنامه استفاده کرد.

نکته- بین مقدار ۲۷ و نام "۲۷" تفاوت وجود دارد .۲۷ مقداری صحیح است که در زمان اجرا در حافظه وجود دارد ولی نام "۲۷"دنباله ای از دو کاراکتر "۲"و "۷ "است که آن عدد را به صورت دهدهی در برنامه نمایش می دهد .

تمرین –تفاوت بین دوجمله زیر چیست؟

```
#define     max     30
Const       int   max=30;
```

دستور اول یک عمل زمان ترجمه است و باعث می شود تمام ارجاعات **max** به ۳۰ تبدیل شوندیعنی این دستور یک فرمان است که باعث می شود مترجم **max** را برابر ۳۰ قرار دهد در حالی که **const** راهنمای مترجم است .

در حوزه آنها می باشد . وقتی از **define** استفاده می کنید کنترل ثابت ها را از حوزه کامپلر خارج می کنید **type checking** روی اسم انجام نمی شود و آدرس آن قابل بازیابی نیست و نمی تواند از نوع **user define** باشد اما اگر ثابت توسط **const** تعریف شده باشد می تواند از هر نوع داه استاندارد یا **user define** باشد .آدرس آن هم قابل بازیابی است و مانند یک متفیر دارای حوزه دستزسی است .بنابراین ثابنی که در درون یک تابع تعریف شده باشد در سایر نقاط برنامه شناخته شده نیست به عبارت دیگر

Define ثابت عمومی

Const ثابت خصوصی

دستور **define** یک دستور پیش پردازتده است .پیش پردازنده یک برنامه سیستمی است که قبل از ترجمه برنامه توسط کامپایار تغییراتی را در برنامه ایحاد می کند .پیش پردارنده مقدار ثابت را که دستور **define** آمده را به جای تام ثابت قرار می دهد و این دستور در زمان اجرا و جود ندارد . به این ثوابت ماکرو نیز می گویند .

چه مسائلی در DECLARATION مشخص می شوند ؟

۱- نوع،مقدار اولیه و ویژگی شی داده
۲- بعضی جزئیات پیاده سازی شامل
الف – نحوه نمایش در حافظه مثلا در COBOL تعیین متغیر صحیح با صفت COMPUTATIONAL نمایش حافظه دودویی را به جای کاراکتری مشخص می کند.
ب- آدرس متغیر در حافظه

تمرین – آیا زبانی را می شناسید که هنگام تعریف متغیر امکان تعیین آدرس آن وجود داشته باشد؟

```
INT     X[FB00];
```

اهداف DECLARATION

۱-بالا بردن سرعت اجرا : با توجه به اینکه اعلان باعث BINDING زمان ترجمه می شود در نتیجه در هنگام اجرا برنامه ها سریع تر اجرا می شوند.

۲-کمک به مدیریت حافظه – با اعلان طول عمر اشیای داده مشخص می شود در نتیجه مدیر حافظه با دسته بندی آنها اشیا داده با طول عمر یکسان را در یک بلوک ذخیره می کند .بنابراین تخصیص و آزاد سازی اشیا داده راحت تر صورت می گیرد (مثلا متغیر های یک زیربرنامه طول عمر یکسان دارند)

چگونگی تعریف DATA OBJECT

توسط DECLARATION تعریف می شود .

DECLARATION (اعلان) در زبان های برنامه سازی

تعریف – دستوری است که توسط آن نام و نوع شی داده را در حین اجرای برنامه مشخص می کند .

```
INT     I, J ;
```

انواع DECLARATION

۱- صریح – توسط برنامه نویس اعلان می شود .

```
FLOAT   A,B;
```

۲- ضمنی – صراحتا توسط برنامه نویس تعریف نمی شود بلکه قبلا برای کامپایلر تعریف شده است .مثلا یک متغیر با نام مشخص بدون اعلان نوع همیشه برای کامپایلر صحیح تعریف شده است .(برای کمک به کامپایلر در ترجمه)

نکته- وجود شی داده در زمان ترجمه دلیل وجود اعلان است .زبان هایی که DECLARATION ندارند تعریف اعلان را در زمان ترجمه غیر ممکن می سازند .

۳- عملیات POLYMORPHISM

تعریف- به یک OPERATION اجازه می دهد رفتار متفاوتی روی OBJECT های مختلف داشته باشد.

در بسیاری از زبان ها بعضی از نماد ها عملیات مختلفی انجام می دهند مثلا

+ : جمع صحیح + : جمع اعشاری + :
الحاق رشته ها

در صورتی که اعلان وجود داشته باشد و نوع اشیا داده معلوم باشد نوع عملیات صورت گرفته روی آن نیز در هنگام ترجمه مشخص می شود (یعنی BINDING را زود هنگام می کتد) و در هنگام اجرا لازم نیست کنترل شود چه عملی باید صورت گیرد.

```
INT   A,B,C;
C=A+B;
```
جمع صحیح

```
FLOAT   A,B,C;
C=A+B;
```
جمع اعشازی

```
STRING   A,B,C;
C=A+B;
```
الحاق رشته

۴- TYPE CHECKING : چک کردن درست بودن نوع متغیر ها از نظر نوع عملی که روی آنها انجام می شود .

تذکر- اعلان باعث می شود کنترل نوع ایستا انجام شود.

انواع TYPE CHECKING

۱-static type checking (ایستا) : در زمان ترجمه توسط کامپایلر انجام می شود .مستلزم اعلان است و در زبان های early binding انجام می شود.مانند زبان های خانواده c

2- dynamic type checking (پویا) : در زمان اجرا انجام می شود و مربوط به زبان هایی است که اعلان ندارند یعنی later binding هستند و اجازه می دهند نوع متغیر ها در زمان اجرا عوض شوند .

نحوه پیاده سازی : برای بررسی نوع در زمان اجرا اشیا داده نیاز به یک type tag دارند این برچسب نوع ساختمان داده زمان اجرا نامیده می شود .که هر data object باید آن را با خود حمل کند .در هنگام اجرا به محض اینکه مقدار شی داده مشخص می شود type tag پر شده و نوع را نشان می دهد .بنابراین بلافاصله قبل از اجرای عملیات نوع شی داده مشخص می شود و بعد از آن با توجه به سیاست زبان type checking انجام می شود .مشخص است که در این حالت در هر عملیات کنترل نوع صورت می گیرد. مانند lisp ,prolog

Type tag	value
Int	16

مزیت روش پویا

قابلیت انعطاف دارد زیرا نوع شی داده تا قبل از اجرا مشخص نیست .

معایب روش پویا

۱- اشکال زدایی و حذف تمام خطا ها مشکل است زیرا نوع داده در زمان اجرا مشخص می شود و ایراد زمان کامپایل نداریم .

۲- افزایش مصرف حافظه : اطلاعات مربوط به نوع در زمان اجرا باید نگهداری شوند(type tag در زمان اجرا حافظه مصرف می کند.) در کنترل نوع ایستا این حافظه مصرف نمی شود .

۳- کاهش سرعت برنامه : زیرا کنترل نوع در زمان اجرا قبل از هر دستور باید انجام شود و این کار زمان بر است.

مثال – بررسی کنید که بعد از اجرای دستورات زیر چه اتفاقی می افتد.

```
Var

X: integer;
Y: real;
Y:=2.5;
X:=y;
```

تبدیل انواع

در برخورد با ناسازگاری داده ها چندین راه زیر برای کامپایلر متصور است :

۱- اعلام خطا (type mistake) : کامپایلر بعد از اعلام خطا اجازه ادامه کار را نمی دهد.

۲- تبدیل نوع ضمنی (conversion) : کامپایلر نوع شی داده را به درستی تغییر می دهد .یعنی نوع شی داده را گرفته و آن را به نوع دیگر تبدیل می کند .

```
X,y :integer;
Z: char;
Y=z+x;
```

کامپایلر char 

integer 1byte  byte

انواع تبدیل نوع ضمنی

۱- تبدیل ضمنی گسترده کننده : (expanding) در این نوع تبدیل ضمنی طول یک شی داده افزایش می یابد مثلا در زبان c هر نوع short int ۲ بایتی می تواند به صورت long int ۴ بایتی نمایش داده شود .یعنی مقادیر کوچک به بزرگ تبدیل شوند .

۲-تبدیل نوع کوتاه کننده (narrowing) : طول یک شی داده در این حالت کاهش می یاند .مثلا برای تبدیل نوع float (۴ بایت) به int (۲ بایت) طول اطلاعات کاهش می یاند و data از بین می رود.

سوال- تبدیل ضمنی کوتاه کننده در چه صورتی ممکن است ؟

باید اشیا داده مقدار مناسبی داشته باشند یعنی داده با ارزش از بین نرود مثلا ۲/۰ اعشاری می تواند به ۲ صحیح تبدیل شود ولی ۳/۵ اعشاری غیر قابل تبدیل به صحیح است .

مزایا و معایب تبدیل نوع ضمنی

تبدیل نوع ضمنی آزادی برنامه نویس را بیشتر می کند ولی خطاها را پنهان می کند که کشف آن دشوار است .مثلا c دارای type checking ضعیف است یعنی از تبدیل نوع ضمنی در بسیاری از موارد استفاده می کند .ولی Ada فاقد تبدیل نوع ضمنی است .

انواع زبان ها از دیدگاه فوق

۱- strongly typed checking : زبان هایی که دارای type checking قوی هسنند .یعنی هیچ تبدیلی را انجام نمی دهند و هنگام ناسازگاری داده ها اعلام خطا می کنند .مانند pascal , Ada, modulla 2

۲-weakly typed checking : زبان هایی که دارای type checking ضعیف هسنند یعنی تا جایی که بتوانند تبدیل نوع انجام می دهند .مانند c ,lisp , pl/1

انواع داده ای Data types

تعریف – اگر یک شی داده را به همراه مجموعه عملیاتی که باید بر روی آن انجام شود در نظر بگیریم نوع داده به وجود می آید .

Int a ; Data object + operations (+ ,- , * , /)

File

Data object

\+ operation(search,open,close,save)

DT = (D.O +OPERATIONS)

انواع Data types از نظر ساختار

۱- مقدماتی (Elementary) : دارای طول ثابت هستند و دستیابی به آنها همواره به صورت یک واحد(unit) است.مانند Boolean, chat ,int , float

۲- ساختاری (structured) : مجموعه ای از سایر اشیا داده است که هم می توانند دارای طول ثابت باشند مانند Array و Record و هم می توانند دارای طول متغیر باشند مانند list , tree , queue , stack

انواع Data types از نظر طول

۱- static data type : انواع داده ای که دارای طول ثابت هستند مانند int , float از مقدماتی و آرایه و رکورد از ساختاری

۲- dynamic data type : انواع داده ای که دارای طول متغیر هستند مانند صف و پشته

انواع Data types از نظر زمان تعریف

۱- primitive data type : انواع داده ای که هنگام تعریف زبان ایجاد شده اند .تمام static ها از این دسته هستند.

۲-programmer defined data type : انواع داه ای که هنگام تعریف زبان ایجاد نشده اند و برنامه نویس برحسب نیازش آنها را تعریف می کند.تقریبا تمام dynamic ها از این دسته اند.

تمرین – تابعی به زبان c اضافه کنید که درخت را هم به عنوان یک ساختمان داده primitive قبول کند.