سیر تکاملی لوا

ما در مورد ظهور و سیر تکاملی لوا بحث خواهیم کرد و اینکه این زبان چگونه از یک ساختار ساده به یک زبان متنوع با کاربرد وسیع که توسعه معنایی ، توابع بدون نام ، دایره لغوی غنی و فراخوانی های مناسب را پشتیبانی می کند ، تبدیل شده است .

معرفی

لوا یک زبان متنی است که در سال ۱۹۹۳ در دانشگاه PUC-RİO ریودوژانیرو برزیل وتوسط ۳ استاد همان دانشگاه متولد شد. از آن زمان تاکنون لوا برای انواع مختلف کاربردهای صنعتی مانند ربوتیک ، فعالیت های توزیع شده ، پردازش تصویر ، ویرایشگرهای متن توسعه ایی ، سوئیچ های اترنت ، بیوانفرماتیک ، بسته های عناصر محدود ، توسعه وب و غیره استفاده می شود . بعلاوه لوا یکی از زبان های برجسته درطراحی و توسعه بازی های کامپیوتری است .

لوا ازحد انتظارات خوش بینانه ما فراتر رفته است . در واقع وقتی که تمام زبان های برنامه نویسی از شمال آمریکا و اروپای غربی تولید می شوند (البته به استثنای RUBY که در ژاپن تولید شده است) ، لوا تنها زبان برنامه نویسی است که در یک کشوردر حال توسعه تولید شده و به صورت گسترده مورداستفاده قرارگرفته است .

درابتدا لوا به صورت ساده ،کوچک ، قابل حمل و باتعبیه آسان درکاربردها طراحی شد . این ویژگی ها و اصول طراحی هنوز بسیار قدرتمند هستند و ما اطمینان داریم که از دلایل موفقیت لوا در صنعت به شمار می روند . خصوصیت اصلی لوا در یک کلام سادگی آن است چرا که لوا فقط یک ساختار داده ارائه می دهد که آن "جدول" است که همان اصطلاح لوا برای آرایه های شرکت پذیر می باشد. اگرچه بسیاری از زبان های برنامه نویسی آرایه های مشترک را نیز ارائه می دهند ولی در هیچ زبان برنامه نویسی دیگری این آرایه ها نقش اصلی را ایفا نمی کنند . جداول لوا پیاده سازی ساده و موثری برای ماژول ها ،عناصرالگو ،کلاس ها ، رکوردها ،آرایه ها ، لیست ها و بسیاری دیگر از ساختار ها هستند .

در اینجا مقدمه ای کوتاه در مورد زبان لوا بیان می کنیم . از لحاظ نحوی از واژگان کلیدی آشنایی استفاده می کند . برای آشنایی با Syntax لوا کد زیر دو نوع از محاسبه فاکتوریل ،

یکی به صورت بازگشتی و دیگری به صورت حلقه را نشان میدهد ، هر کس با اندک معلومات از برنامه نویسی می تواند عبارات و مفهوم این کدها را درک کند .

```
function factorial(n)
  if n == 0 then
    return 1
  else
    return n*factorial(n-1)
  end
end
```

```
function factorial(n)
  local a = 1
  for i = 1,n do
    a = a*i
  end
  return a
end
```

از لحاظ معنایی لوا با SCHEME شباهت بسیاری دارد اما این شباهت ها به راحتی قابل تشخیص نیست چون این دو زبان از لحاظ لغوی تفاوت های زیادی دارند . رفته رفته در طول تکامل لوا ، برتری های SCHEME بر لوا افزایش یافت .
اساسا SCHEME فقط یک زبان در پس زمینه بود اما بعدها با معرفی توابع بدون نام و دایره لغات کامل اهمیت پیدا کرد .
لوا نیز مانند SCHEME دارای انواع دینامیک است به طوریکه متغیر نوع ندارد فقط مقادیر نوع دارند و دیگر اینکه یک متغیر در لوا هرگز مقدار ساختاریافته ای به خود نمی گیرد فقط به آن اشاره می کند . در SCHEME نام یک تابع هیچ وضعیت خاصی ندارد در لوا نام تابع فقط یک متغیر معمولی است که به مقدار تابع اشاره می کند .
یک تفاوت معنایی بین لوا و SCHEME و شاید اصلی ترین ویژگی متمایز کننده لوا این است که "جداول" تنها مکانیزم ساختار داده در لوا هستند . جداول لوا آرایه های مرتبط به هم هستند ولی با تعدادی خصوصیت مهم مانند تمام مقادیر در لوا . جداول مقادیر کلاس اول هستند . آنها به مشخص کردن نام متغیر محدود نمی شوند مانند چیزی که در PERL و AWK وجود دارد . یک جدول در لوا می تواند هر مقداری برای کلید داشته باشد و هر مقداری نیز ذخیره کند . جداول اجازه یک پیاده سازی ساده و مفید از رکوردها (با استفاده از نام فیلد به عنوان کلید)، مجموعه ها (با استفاده از عناصر مجموعه به عنان کلید) ، ساختارهای عمومی پیوندی و بسیاری از ساختار های داده ای را می دهند .بعلاوه می توان از یک جدول برای پیاده سازی یک آرایه با استفاده از اعداد طبیعی به عنوان اندیس استفاده کرد . این پیاده سازی هوشمندانه تضمین می کند که یک جدول به همان

میزانی که آرایه فضا اشغال می کند حافظه می گیرد و نسبت به آرایه در زبان های مشابه بهتر عمل می کند .

لوا یک Syntax قوی برای تولید جداول به فرم سازند ها ارائه می دهد . ساده ترین سازنده عبارت است از "{}" که یک جدول خالی و جدید تولید می کند . ترکیب سازنده های جداول و توابع ، لوا را تبدیل به یک زبان قدرتمند توصیف داده می کند . برای مثال یک بانک اطلاعاتی فهرست کتاب مانند چیزی که در Bibtex استفاده شده است را میتوان با استفاده از سری جداول مانند زیر نوشت :

```
article{"spe96",
  authors = {"Roberto Ierusalimschy",
             "Luiz Henrique de Figueiredo",
             "Waldemar Celes"},
  title = "Lua: an Extensible Extension Language",
  journal = "Software: Practice & Experience",
  year = 1996,
}
```

اگر پایگاه داده را به عنوان یک فایل داده ای ثابت در نظر بگیریم این برنامه یک برنامه درست لوا است .وقتی بانک اطلاعاتی در لوا بارگذاری می شود هر جزء آن یک تابع را فراخوانی می کند.

گفتیم لوا یک زبان قابل توسعه است چون به توسعه برنامه های کاربردی با استفاده از پیکربندی ماکرو ها و دیگر تنظیمان کاربر نهایی کمک می کند . لوا برای تعبیه در درون برنامه های کاربردی میزبان طراحی شده است بنابراین کاربران می توانند با نوشتن برنامه های لوا که به سرویس های برنامه های کاربردی و محاسبات داده ای برنامه دسترسی دارد رفتار برنامه کاربردی را کنترل کنند.

لوا قابل توسعه است چون داده های کاربر را ارائه می دهد که داده های کاربردی را حفظ می کند و از مکانیزم توسعه معنایی برای انجام محاسبات روی این مقادیر از راه طبیعی استفاده می کند . لوا به عنوان یک هسته کوچک تهیه شده است که با توابع کاربر که در C یا لوا نوشته شده اند قابل توسعه است . بعلاوه ورودی ، خروجی و محاسبات رشته

ایی ، توابع ریاضی و واسط کاربر برای سیستم عامل همگی به صورت کتابخانه های خارجی تهیه شده اند . دیگر خصوصیات متمایز کننده لوا از پیاده سازی آن ناشی می شود.

خصوصیات

۱- **قابل حمل بودن** : لوا ساده است چون در C اجرا می شود و در بسیاری لز محیط های لینوکس ، یونیکس ، ویندوز ، مک و غیره به صورت Out of box (خارج از جعبه) کامپایل می شود و با اندک تنظیماتی در همه محیط ها از جمله دستگاه های موبایل و سیستم عامل های Symbian ، BREW ، Pocket Pc و میکروپروسسورهای ARM وRabbit قابل اجراست . دارای کامپایلر ANCI/ISO C می باشد .

۲- **تعبیه ساده و مناسب :** لوا طوری طراحی شده است که به راحتی درون برنامه های کاربردی تعبیه می شود یک قسمت مهم لوا این است که یک API خوش تعریف دارد که اجازه یک ارتباط کامل را بین کد لوا و کد خارجی می دهد . توسعه لوا با استفاده از صدور توابع C از برنامه کاربردی میزبان به راحتی قابل انجام است . API به لوا این امکان را می دهد که علاوه بر C و C++ با دیگر زبان های برنامه نویسی مانند Fortran ، Java ، SmallTalk ، Ada ، Perl ، Ruby ارتباط داشته باشد.

۳-**اندازه کوچک :** اضافه کردن لوا به یک برنامه حجم آن را افزایش نمی دهد تمام توزیع لوا شامل کد منبع ، مستندسازی باینری ها برای بعضی محیط ها بر روی یک فلاپی دیسک جا می شود (۸۶۰کیلو بایت) چون کتابخانه های کمی دارد .

۴-**کارایی :** شاخصه های استقلال ، لوا را به عنوان یکی از سریعترین زبان های برنامه نویسی در حوزه زبان های متنی مفسری نشان می دهد . این خصوصیت به تولید کنندگان برنامه های کاربردی این امکان را می دهد که قسمتی از برنامه خود را در لوا بنویسند . برای مثال بالغ بر۴۰درصد از Adobe Light room در لوا نوشته شده است (تقریبا ۱۰۰هزار خط کد را شامل می شود)

۵-**قدرتمند :** مفهوم اصلی در طراحی لوا استفاده از مکانیزم های پیشرفته برای پیاده سازی خصوصیات است . با این وجود که لوا یک زبان کاملا شی گرا نیست ولی از مکانیز م های پیشرفته ای برای پیاده سازی کلاس ها و وراثت استفاده می کند .

۶-**رایگان :** می توان از آدرس www.lua.org به صورت رایگان دانلود کرد .

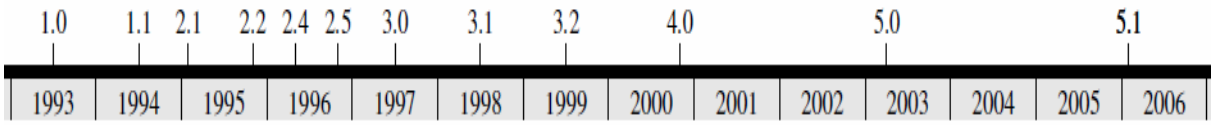
اگرچه این خصوصیات یک پیاده سازی بخصوص است که به دلیل طراحی لوا امکان پذیر شده است ، سادگی لوا یک فاکتور کلیدی در پیاده سازی کوچک و موثر است . لوا یک زبان برنامه نویسی همه منظوره کوچک و تعبیه شده است که جهت برنامه نویسی روالی با امکان توصیف داده طراحی شده است . و درکامپیوتر میزبان به صورت تعبیه شده کار میکند مانند کتابخانه ای از توابع C برای اتصال به برنامه اصلی .
برنامه اصلی می تواند توابع درون کتابخانه را جهت اجرای قسمتی از دستورات لوا ، خواندن ، نوشتن متغیرهای لوا و ثبت توابعC برای فراخوانی توسط کد لوا صدا زد .

تاریخچه

از سال ۱۹۹۲تا ۱۹۹۷ در برزیل سیاست "صرفه اقتصادی" در رابطه با سخت افزار و نرم افزار در پیش گرفته شد که برزیل می تواند و باید سخت افزار و نرم افزار خودرا خود تولید کند . در چنین جوی مشتریان TECGRAF نمی توانستند از لحاظ مالی و سیاسی نرم افزار های مورد نیاز خود را از خارج تهیه کنند. با قوانین صرفه اقتصادی آنها مجبور بودند برای رفع نیازهایشان از یک پروسه سخت اداری که شرکت های برزیلی آن را نمی پسندیدند عبور کنند . علاوه براین جدایی طبیعی و جغرافیایی برزیل از دیگر مراکز تحقیقاتی گروه TECGRAF را به سوی پیاده سازی ابزارهای پایه که نیاز داشت سوق داد .

یکی از بزرگترین شرکای TECGRAF، Petrobras بوده و هنوز هم هست ، یک شرکت نفتی برزیلی است. بسیاری از تولیدات TECGRAF برنامه های گرافیکی جذاب برای کاربردهای مهندسی در Petrobras بود . در سال ۱۹۹۳ TECGRAF زبان های کوچک را برای دو نمونه از آن کاربردها توسعه داد : ۱- برنامه ایی برای ورود داده (DEL) ۲-یک تولید کننده گزارشات قابل ویرایش برای نمودار های سنگ شناسی (SOL) که اجداد لوا به حساب می آیند .

نمودار زیر نمودار رشد لوا را نشان میدهد همانطور که دیده می شود زمان بین تولید ورژن های مختلف لوا اساسا از لوا ۳ به بعد افزایش می یابد .



لوا به یک محصول کامل تبدیل شد و نیازمند استحکام برای برآورد منافع جامعه در حال رشد خود بود با این حال نیاز به ا ستحکام هیچ وقت مانع روند توسعه نشد. نسخه های بزرگ لوا (۴و۵) بعد از آن موقع رشد پیدا کردند .

	1.0	1.1	2.1	2.2	2.4	2.5	3.0	3.1	3.2	4.0	5.0	5.1
constructors	●	●	●	●	●	●	●	●	●	●	●	●
garbage collection	●	●	●	●	●	●	●	●	●	●	●	●
extensible semantics	○	○	●	●	●	●	●	●	●	●	●	●
support for OOP	○	○	●	●	●	●	●	●	●	●	●	●
long strings	○	○	○	●	●	●	●	●	●	●	●	●
debug API	○	○	○	●	●	●	●	●	●	●	●	●
external compiler	○	○	○	○	●	●	●	●	●	●	●	●
vararg functions	○	○	○	○	○	●	●	●	●	●	●	●
pattern matching	○	○	○	○	○	●	●	●	●	●	●	●
conditional compilation	○	○	○	○	○	○	●	●	●	○	○	○
anonymous functions, closures	○	○	○	○	○	○	○	●	●	●	●	●
debug library	○	○	○	○	○	○	○	○	●	●	●	●
multi-state API	○	○	○	○	○	○	○	○	○	●	●	●
`for` statement	○	○	○	○	○	○	○	○	○	●	●	●
long comments	○	○	○	○	○	○	○	○	○	○	●	●
full lexical scoping	○	○	○	○	○	○	○	○	○	○	●	●
booleans	○	○	○	○	○	○	○	○	○	○	●	●
coroutines	○	○	○	○	○	○	○	○	○	○	●	●
incremental garbage collection	○	○	○	○	○	○	○	○	○	○	○	●
module system	○	○	○	○	○	○	○	○	○	○	○	●

فاصله زمانی زیاد مابین نسخه های مختلف مدل رشد لوا را منعکس می کند . برخلاف دیگر پروژه های open source ، نسخه های اولیه یا آلفا پایا بودند و نسخه های بتا اساسا پایانی هستند. بجز برای ایرادات آشکار ثابت شده است این مدل انتشاری برای استحکام لوا مناسب است . محصولات متنوعی از نسخه های آلفاد و بتا از لوا منتشر شده و به خوبی کار می کنند. تعداد کتابخانه های استاندارد لوا دارای حجم کوچکی هستند چون بسیاری از توابع توسط برنامه کاربردی میزبان و کتابخانه های سطح سوم تامین می شود . تا لوا ۳ کتابخانه های استاندارد فقط برای ورودی ، خروجی ، محاسبات رشته ایی ، توابع ریاضی و یک کتابخانه مخصوص برای توابع داخلی که از C API استفاده نمی کنند اما مستقیما به ساختمان داده های داخلی دسترسی دارند . از آن موقع به بعد کتابخانه هایی برای اشکال زدایی (لوای ۳,۲) ، رابط بین سیستم عامل (لوا ۴) ، جداول و روال ها (لوا ۵) ، ماژول ها (لوا ۵,۱) اضافه کردیم . اندازه C API زمانیکه برای لوای ۴ طراحی شد به طور قابل ملاحظه ایی تغییر کرد . از آن به بعد به سمت کامل شدن حرکت کرد . نتیجتا توابع داخلی بزرگتری وجود ندارند تمام کتابخانه های استاندارد بر روی C API پیاده سازی شده است بدون دسترسی به توابع داخلی لوا . ماشین مجازی که برنامه های لوا را اجرا می کند به صورت پشته طراحی شده بود تا لوای ۴. در لوا ۳,۱ ما متغیر ها را برای بسیاری از دستورالعمل ها و ارتقاء کارایی اضافه کردیم .

Lua ۱

پیاده سازی اصلی و اولیه لوا پیشرفت در TECGRAF بود و لوا از دیگر پروژه های TECGRAF کاربران را مجذوب کرد .کاربران جدید سفارشات جدید می دهند . کاربران مختلف می خواهند لوا را به عنوان زبان پشتیبان در محیط های گرافیکی که در TECGRAF محدود هستند استفاده کنند . طبیعی است که برای مدل کردن عناصر پیچیده باید قطعات کد را با عبارات ترکیب کرد. برای مثال VRML باید یک زبان دیگر را برای مل سازی اجزاء روالی استفاده کند . استفاده از لوا برای این نوع از توصیف داده مخصوصا فایل های گرافیکی بزرگ مشکلاتی را برای زبان های متنی به همراه دارد مثلا برای یک دیاگرام که در برنامه ورود داده استفاده میشود داشتن چندین هزار قسمت که توسط یک سازنده جدول که شامل قسمت های مختلف بوده غیرعادی نبود . به این معنی است که لوا مجبور بود با برنامه های بزرگ واصطلاحات بزرگ سروکارداشته باشد چون تمام

برنامه ها را به بایت کدها برای ماشین مجازی کامپایل می کرد باید سرعت بالایی داشت حتی برای برنامه های بزرگ .

با عوض کردن اسکنر Lex-generated که در نسخه اولیه لوا استفاده می شد ، با یک نسخه دست نویس سرعت کامپایل لوا را افزایش دادیم . ما همچنین با اضافه کردن جفت مقادیر کلیدی درجداول ما شین مجازی لوا را برای کارکردن با سازنده های طولانی مجهز کردیم .این تغییرات تقاضاهای اصلی برای بهبود کارایی را مرتفع کرد از آن زمان ما تلاش کرده ایم که زمانی که برای مراحل قبل از کامپایل مصرف می شود به حداقل برسد . در جولای ۱۹۹۴ نسخه جدیدی از لوا را با آن بهینه سازی ها منتشر کردیم . این کار با انتشار اولین مقاله که در توصیف لوا بود همزمان شد که طراحی و پیاده سازی لوا را شرح می داد . نسخه جدید را Lua ۱,۱ نامیدیم که به عنوان یک نرم افزار دردسترس و به صورت نسخه باز توسط ftp منتشر شد ، در حالیکه جنبش نسخه های باز به درجه و حرکت کنونی خود نرسیده بود .

Lua ۱,۱ برای استفاده های دانشگاهی به صورت رایگان در دسترس بود ولی برای موارد تجاری باید مذاکره می شد .

Lua ۲

ما نمی خواهیم لوا به یک زبان شی گرا تبدیل شود چون آن را با برنامه نویسی نمونه تطبیق نمی دهیم و فکر میکنیم که لوا به عناصر وکلاس ها به عنوان مفاهیم اصلی نیاز ندارد مخصوصا زمانی که می توان آن را به وسیله جدول پیاده سازی کرد . لوا برنامه نویس را مجبور به استفاده از کلاسها نمی کند . بسیاری از مدل های اجزا توسط خود کاربر پیشنهاد و پیاده سازی می شود. از سویی دیگر می خواهیم برنامه نویسی شی گرا رابوسیله لوا مقدور سازیم . به جای ثلبت کردن یک مدل تصمیم گرفتیم که مکانیزم های قابل انعطافی که به برنامه نویس اجازه ایجاد یک مدل مناسب برای برنامه کاربردی میدهد را تامین کنیم . Lua ۲,۱ در فوریه ۱۹۹۵ منتشر شد با تکیه بر معرفی این مکانیزم های توسعه معنایی که به صورت فزآینده ایی پرمعنایی لوا را افزایش داد . توسعه معنایی به یک نماد از لوا تبدیل شد که یکی از اهداف آن استفاده از جداول به عنوان مبنا برای پیاده سازی اجزا و کلاس هاست . برای این منظور ما نیازمند توسعه ارث بری برای جداول بودیم . هدف دیگر تغییر داده کاربر به داده های کاربردی طبیعی است . ما می خواهیم قادر به شاخص گذاری داده های کاربر

باشیم همچنان که اگر جدول بودند و متدها را برای آنها فراخوانی کنیم . این به لوا امکان برآورده کردن اصلی ترین اهداف طراحی اش را که توسعه برنامه های کاربردی با دسترسی متنی به سرویس ها و داده هاست را تامین می کند . به جای اضافه کردن مستقیم این مکانیزم ها برای پشتیبانی همه این خصوصیات ما تصمیم گرفتیم که ساده تر است مقالات دانشگاهی بیشتری تعریف شوند . در دسامبر ۱۹۹۶ کمی بعد از Lua ۲,۵ در مجله Dr.Dobbs که لوا را به صورت برجسته نشان داد مقاله به چاپ رساندیم . این مجله یک انتشار محبوب برای برنامه نویسان است و آن مقاله لوا را در صنعت نرم افزار مورد توجه قرار داد . بعد از آن مقاله از شرکت Lucas Art با ما تماس گرفته شد و گفتند که می خواهند زبان برنامه نویسی خودرا به Lua تغییر دهند . بعد از آن لوا در میان توسعه دهندگان بازی های کامپیوتری پخش شد و به صورت قاطع به عنوان یک مهارت قابل عرضه در صنعت بازی های کامپیوتری شناخته شد.

Lua ۳

مکانیزم fallback در Lua ۲,۱ برای توسعه معنایی معرفی شد . این یک مکانیزم جهانی بود. یک قلاب برای هر رویداد وجود داشت . این خصوصیت اشتراک یا دوباره استفاده کردن از کدها و ماژول ها را سخت می کرد چون ماژول هایی که برای fallback های یک رویداد همسان در نظر گرفته شده بود به راحتی با هم کار نمی کردند .

شرکت هایی که از لوا استفاده می کنند :

Adobe
Bombardier
Disney
Intel
Microsoft
Electronic Arts
Lucas Art
Nasa
…

بازی های کامپیوتری مشهور طراحی شده در لوا :

Far cry
Grim Fandango
The sims
World of Warcraft
Home world
Imposible creatures
Escape from monkey island
...

کلمات کلیدی در لوا :

and break do else elseif
end false for function if
in local nil not or
repeat return then true until while

عملگرها در لوا :

```
+     -     *     /     %     ^     #
==    ~=    <=    >=    <     >     =
   (     )     {     }     [     ]
  ;     :     ,     .     ..    ...
```

نامها درلوا می توانند هر رشته از حروف شامل اعداد باشند ولی متغیرها نباید با عدد آغاز شوند. لوا یک زبان حساس به حروف است .اعداد مبنای ۱۶ در لوا قابل پذیرش است با اضافه کردن ox به ابتدای آن . لوا یک زبان دینامیک است به این معنی که متغیر ها در لوا نوع ندارند و نوعی برای متغیرها مشخص نمی شود مقادیرخود بیانگر نوع داده می باشند .
۸ نوع داده اصلی در لوا وجود دارد :nil ، عددی ، رشته ایی ، تابعی ، داده کاربر ، نخ ، جداول و داده های منطقی

Userdata : این نوع از داده برای داده های اختیاری C که می توان در متغیرها ذخیره کرد در نظر گرفته شده است . این نوع از داده مربوط به یک بلاک از حافظه است و هیچ عملگر از پیش تعیین شده ای ندارد . با استفاده از Meta table ها برنامه نویس می تواند عملیات را تعریف کند این نوع در لوا تولید یا مدیریت نمی شوند فقط از طریق C API این کارانجام می شود.

Thread : این نوع داده نمایانگر نخ های مستقل اجرا هستند و برای پیاده سازی روال های مرتبط به هم استفاده می شوند. نباید نخ های لوا را با نخ های سیستم عامل اشتباه گرفت ، لوا نخها را بر روی تمامی سیستم عامل ها پشتیبانی می کند حتی آنهایی که نخ را پشتیبانی نمی کنند .

Table : نوع داده جدول آرایه های شرکت پذیر را پیاده سازی می کند . این آرایه های می توانند نه تنها با اعداد بلکه با هر مقدار دیگری بجز تهی شاخص گذاری شوند . جداول می توانند ناهمگن باشند و هر مقداری را در خود نگه داری کنند . جدول ساختمان داده منحصر به فرد در لواست . می توان برای نمایش آرایه های عادی ، جداول علامت ، مجموعه ها ، رکوردها ، گراف ها و درخت ها از آن استفاده کرد . حتی جداول می توانند شامل توابع نیز باشند. برای نمایش رکوردها از یک فیلد به عنوان شاخص استفاده می کند .

سه نوع متغیر در لوا وجود دارد : متغیرهای عمومی ، محلی و فیلدهای جداول

مکانیزم های پیشر فته

Fallback ها : هنگامی که لوا نمی داند چگونه دستورات را اجرا کند از این توابع استفاده می کند. هنگامی که در شرایط غیرعادی قرار گیرد مثلا هنگامیکه عملیات ریاضی با عملوند های غیر عددی استفاده شود ، چون خروج از سیستم در این شرایط برای برنامه مناسب نیست لوا این اجازه را می دهد که برنامه نویس توابع خود را در شرایط وقوع خطا تنظیم کند این توابع توابع Fallback نامیده می شوند که برای قرار گرفتن در موقعیتی که دقیقا شرایط خطا نیست استفاده می شوند مانند دسترسی به یک فیلد خالی در جدول .

یکی از کاربردهای Fallback ها در پیاده سازی وراثت است . یک وراثت ساده به یک شی اجازه می دهد که مقدار یک فیلد خالی را در فیلد والدش پیدا کند این مکانیزم یک نوع از وراثت است مانند وراثتی که در C++ وSmallTalk استفاده می شود .
تنظیم Fallback ها توسط متخصصین لوا انجام میگیرد آن هم زمانی که لوا را به یک برنامه خاص متصل می کنند .

آرایه های شرکت پذیر : سازنده قوی داده های متعدد هستند که الگوریتم های کاراتری نسبت به دیگر سازنده ها مانند رشته ها و لیست ها دارند . برخلاف دیگر زبان ها که ـرایه های شرکت پذیر را پیاده سازی می کنند ، جداول در لوا به صورت پویا ساگخته می شوند .

مدیریت اتوماتیک حافظه (Garbage Callector) : لوا حافظه را به صورت اتوماتیک مدیریت میکند به این معنی که در مورد اختصاص حا فظه به عناصر جدید یا آزادسازی حافظه مصرفی آنها زمانیکه دیگر موردنیاز نیستند نگران نباشید. این کار با استفاده از امکان Garbage Callector انجام می شود که تمام حافظه مصرفی توسط عناصر مرده را جمع آوری می کند .

دستورات کنترلی :

```
stat ::= while exp do block end
      stat ::= repeat block until exp
      stat ::= if exp then block {elseif exp then block}
[else block] end
```

حلقه :

```
stat ::= for Name `=´ exp `,´ exp [`,´ exp] do block end
```

برای مثال:

```
for v = e۱, e۲, e۳ do block end
```

که برابراست با دستورات زیر :

```
                                    do
       local var, limit, step = tonumber(e۱), tonumber(e۲),
tonumber(e۳)
       if not (var and limit and step) then error() end
       while (step > ۰ and var <= limit) or (step <= ۰ and
var >= limit) do
         local v = var
         block
         var = var + step
       end
     end
```

عملیات منطقی :

```
    ۱۰ or ۲۰              --> ۱۰
  ۱۰ or error()       --> ۱۰
  nil or "a"          --> "a"
  nil and ۱۰          --> nil
  false and error()   --> false
  false and nil       --> false
  false or nil        --> nil
  ۱۰ and ۲۰           --> ۲۰
```

تعریف توابع :

```
 function ::= function funcbody
funcbody ::= `(´ [parlist] `)´ block end


stat ::= function funcname funcbody
      stat ::= local function Name funcbody
      funcname ::= Name {`.´ Name} [`:´ Name]
```

متغیرها عمومی :

```
x = ١٠                  -- global variable
     do                      -- new block
       local x = x           -- new 'x', with value ١٠
       print(x)              --> ١٠
       x = x+١
       do                    -- another block
         local x = x+١       -- another 'x'
         print(x)            --> ١٢
       end
       print(x)              --> ١١
     end
     print(x)                --> ١٠  (the global one)
```

متغیرها محلی :

```
a = {}
     local x = ٢٠
     for i=١،١٠ do
       local y = ٠
       a[i] = function () y=y+١; return x+y end
     end
```