آموزش برنامه نویسی هوش مصنوعی با

LISP

مهندس رضا مهدی‌هادی

انتشارات پندار پارس

سرشناسه	:	مهدی هادی، رضا، 1367 -
عنوان و نام پدیدآور	:	آموزش برنامه‌نویسی هوش مصنوعی با LISP/ رضا مهدی‌هادی.
مشخصات نشر	:	تهران : پندار پارس، 1390. ناشر همکار: مانلی
مشخصات ظاهری	:	328ص.: مصور، جدول.
شابک	:	978-964-2989-89-8:95000 ریال
یادداشت	:	واژه‌نامه.
یادداشت	:	کتابنامه.
موضوع	:	لیسپ (زبان برنامه‌نویسی کامپیوتر)
موضوع	:	هوش مصنوعی -- برنامه‌های کامپیوتری
رده بندی کنگره	:	QA76/73 1390 9م93ل /
رده بندی دیویی	:	6424/001
شماره کتابشناسی ملی	:	2587658

انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگرجنوبی، کوی رشتچی، شماره 14، واحد 16 **www.pendarepars.com**

تلفن: 66572335 - تلفکس: 66926578همراه: 09122452348 **info@pendarepars.com**

نام کتاب : **آموزش برنامه‌نویسی هوش مصنوعی با LISP**

ناشر : **انتشارات پندار پارس ناشر همکار: مانلی**

ترجمه و تالیف : **رضا مهدی‌هادی**

چاپ اول : **زمستان 90**

شمارگان : **1000 نسخه**

طرح جلد : **محمد اسماعیلی هدی**

لیتوگرافی، چاپ، صحافی : **ترام‌سنج، صالحان، خیام**

قیمت : **9500 تومان به همراه CD** **شابک : 978-964-2989-89-8**

فهرست

مقدمه

همان‌طور که می‌دانید زبان برنامه‌نویسی Lisp یکی از قدرتمندترین زبان‌های برنامه‌نویسی در زمینه هوش مصنوعی است. به‌طوری‌که این زبان به‌همراه زبان برنامه‌نویسی پرولوگ در اکثر دانشگاه‌های معتبر دنیا به‌عنوان یک زبان هوش مصنوعی مورد استفاده قرار می‌گیرد. از جمله ویژگی‌های بارز این زبان، می‌توان به قابلیت نمادگرائی و شئ‌گرائی و محاسباتی آن اشاره کرد.

این کتاب در حقیقت با استفاده از بیانی ساده و قابل فهم، سعی بر آن دارد تا شما را با زبان برنامه‌نویسی لیسپ همراه با تکنیک برنامه‌نویسی هوش مصنوعی آشنا نماید.

از ویژگی‌های بارز این کتاب می‌توان به موارد زیر اشاره نمود:

به‌جای تکیه بر مباحث تئوریک، سعی بر آن است تا با استفاده از نمونه برنامه‌های مختلف و متنوع و هدفمند، مطالب مورد نظر را به‌صورت علمی و کاربردی به خوانندگان منتقل نماید. برای درک بهتر برنامه‌ها، خروجی برنامه‌ها بعد از کد برنامه مورد نظر به‌صورت متنی قرار داده شده است. از اهداف مهم کتاب، بالا بردن حجم نکات مفید و مهم جهت انتقال به خوانندگان و در عین حال پایین آوردن تعداد صفحات کتاب می‌باشد. در این کتاب سعی شده است تا با بیانی مختصر و مفید و به کمک مثال‌های ملموس و قابل درک، خواننده بتواند با صرف وقت کمتر حجم بیشتری از مطلب را فراگیری نماید. فصل آخر کتاب شامل حل مسائل کاربردی است، که خوانندگان با نمونه‌های کاربردی در زمینه برنامه‌نویسی هوش مصنوعی آشنا شده و قادر به نوشتن برنامه‌های جدید باشند. جهت جلوگیری از اتلاف زمان برای تایپ متن برنامه‌های موجود در فصل آخر کتاب، آنها را در CD همراه کتاب گنجانده‌ایم. در آخر هر فصل نیز نکات تاریخی مربوط به آن درج شده است. نکات تاریخی مربوط به پیشنهاداتی است که در طی سالیان توسعه لیسپ برای این زبان پیشنهاد شده است.

امید است توانسته باشم در جهت رشد و تعالی سطح علمی در زمینه تخصصی کتاب فوق گامی مؤثر برداشته باشم. در ضمن اساتید، دانشجویان و سایر خوانندگان محترم می‌توانند هرگونه نظر، انتقاد و پیشنهاد خود را در سایت pro-programing.com مطرح فرمایند.

رضا مهدی هادی

پاییز 1390

فصل 1

تاریخچه

در سال 1950 مشکلات پردازش نمادین در ماشین‌ها در حوزه‌های زبان‌شناسی، روان‌شناسی و ریاضیات در محدوده هوش مصنوعی باعث شد تا نیازهای مشترک آنها، در قالب داده‌های لیست‌ها انجام پذیرد. این ایده باعث شد تا در سال 1950 برای اولین‌بار شرکت IBM در حوزه هوش مصنوعی، پروژه‌ای با نام FLPL بر روی زبان Fortran طراحی و اجرا کند. تصمیم بر این بود تا قابلیت پردازش لیست‌ها به زبان Fortran افزوده شود.

در سال 1958 جان مک کارتی وارد شرکت IBM شد تا مجموعه‌ای برای پردازش و محاسبات نمادین ایجاد کند. لیسپ اولین‌بار به‌وسیله جان مک‌کارتی در سال 1958 در مؤسسه فناوری ماساچوست (MIT) مطرح و به‌عنوان یک مدل پیوسته محاسباتی بر اساس تئوری بازگشتی، معرفی شد. در این خصوص مک کارتی در سال 1960 مقاله‌ای را ارائه کرد که شامل دو بخش بود. بخش اول مقاله شامل:

- ایجاد یک زبان سمبولیک تا یک زبان محاسباتی
- ایجاد زبانی که بتوان از آن به‌عنوان یک مدل محاسباتی بر اساس تئوری بازگشتی استفاده کرد

بخش دوم مقاله هیچ‌گاه منتشر نشد. زبان لیسپ برای اولین‌بار توسط استفان راسل (از دانشجویان مک کارتی در دانشگاه MIT) بر روی کامپیوتر IBM704 پیاده‌سازی و اجرا شد. راسل پس از مطالعه مقاله مک کارتی دریافت که توابع لیسپ توانایی اجرا در کد ماشین را دارد.

اولین کامپایلر تکمیل شده لیسپ، در سال 1960 توسط تامهارت و مایکلوین در کارگاه هوش مصنوعی MIT اجرا شد. زبان به‌کار گرفته شده در این کامپایلر در سبک هارت و لوین نسبت به کدهای ابتدائی مک کارتی به شیوه لیسپ مدرن و جدید نزدیک‌تر است. در حقیقت این مدل برنامه‌نویسی طوری کارآمد بوده که تعدادی از دیگر زبان‌ها، نشأت گرفته از عمل‌کرد برنامه‌نویسی آن بوده است مثل FP، ML و SCHEME.

یک سال بعد نخستین کامپایلر استاندارد با نام Lisp1.5 معرفی شد. پس از آن تعدادی از نسخه‌ها و محیط‌های برنامه‌نویسی لیسپ توسعه یافت؛ مانند MacLisp، FranzLisp، InterLisp،

CommonLisp، و Scheme. هر چند آنها در بعضی جزئیات خاص متفاوتند ولی هسته Syntax (نحو) و Semantic (معنی) آنها اساساً یکسان بودند.

در اویل توسعه لیسپ، محققان می‌خواستند یک ماشین با سخت‌افزار خاص فقط برای اجرای برنامه‌های لیسپ طراحی کنند. این ماشین‌ها "Lisp Machines" نام داشتند. در سال 1973ریچارد گرینبلت و توماس نایت توانستند اولین ماشین لیسپ را در آزمایشگاه هوش مصنوعی MIT طراحی کنند. این طرح در اوایل دهه 1970 رشد زیادی داشت تا اینکه در اواخر دهه 1980 با پیشرفت کامپیوترها، این طرح با شکست مواجه شد.

لیسپ در همان اوایل سریعاً به انجمن تحقیقاتی هوش مصنوعی پیوست، خصوصاً به سیستم PDP . لیسپ یکی از قدیمی‌ترین زبان‌های برنامه‌نویسی هوش مصنوعی محسوب می‌شود و در علوم کامپیوتر بر بسیاری از تفکرات و ایده‌ها پیشگام بوده است.

اسم مستعار LISP از Lost In Stupid Parenthese یا Lost of Irritating Supper fluous parenthese و List Processing گرفته شده‌است .

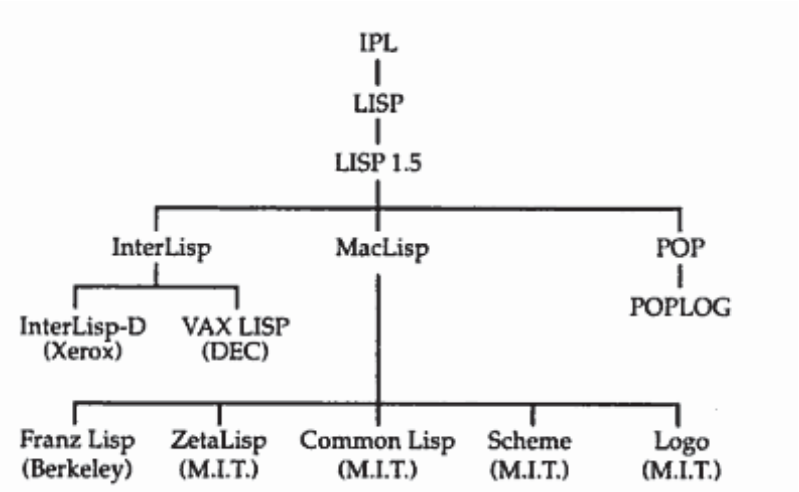
این زبان مانند زبان برنامه‌نویسی پرولوگ، بیشتر برای برنامه‌نویسی هوش مصنوعی مورد استفاده قرار می‌گیرد. با توجه به اینکه زبان لیسپ از سینتکس ساده‌ای برخوردار است تجزیه و پیاده‌سازی آن نسبتاً با سهولت انجام می‌شود.

متن برنامه‌های لیسپ عموماً از نمادها و لیست‌هایی از نمادها تشکیل می‌شود و بدین خاطر است که زبان لیسپ (مخفف پردازش لیست) نامیده شده است. یکی از ویژگی‌های جالب این زبان آن است که خود برنامه‌های لیسپ نیز لیست هستند و بنابرین، می‌توان با برنامه‌ها به‌عنوان داده‌ها رفتار کرد و یا داده‌ها را به‌عنوان برنامه ارزیابی نمود. همچنین این زبان دارای گویش‌های مختلفی است که بعضی از آنها دارای قابلیت‌های شئ‌گرایی نیز هستند. یک زبان لیسپ شامل ساختمان داده درخت، مدیریت نگهداری اتوماتیک، برنامه‌نویسی پویا، برنامه‌نویسی شئ‌گرا و کامپایلر مستقل است.

در لیسپ برخلاف بیشتر زبان‌های دیگر، بین عبارت‌ها و جمله‌ها تمایز و فرقی وجود ندارد. همه کدها و داده‌ها به‌عنوان عبارت‌ها نوشته می‌شوند. تکیه روی عبارت‌ها، قابلیت انعطاف‌پذیری زیادی به زبان می‌دهد، زیرا توابع لیسپ به‌صورت لیست نوشته شده‌اند. آنها دقیقاً مانند داده‌ها می‌توانند پردازش شوند. این قابلیت اجازه می‌دهد برنامه‌های لیسپ به‌سادگی و راحتی نوشته و به نسبت برنامه‌های دیگر به‌راحتی اداره شوند.

توزیع‌های مختلف لیسپ

لیسپ شامل توزیع‌ها و خانواده‌های مختلفی است، که در شکل زیر انواع این توزیع‌ها را مشاهده می‌کنید. پراستفاده‌ترین نسخه‌های لیسپ Common Lisp و scheme هستند. در این کتاب ما Common Lisp را برای نشان دادن جنبه‌های مختلف لیسپ انتخاب کرده‌ایم. هر چند مثال‌هایی که در این نسخه قابل اجرا هستند، به‌راحتی می‌توانند در نسخه‌های دیگر لیسپ سازگار شوند.



لیسپ معمولی یا Common Lisp که معمولاً به‌صورت مخفف CL خوانده می‌شود در سال 1994 به‌منظور منشعب کردن نسخه‌های مختلف لیسپ به‌وسیله سازمان ANSI *30266 استاندارد و گسترش یافت. CL در واقع یک پیاده‌سازی نیست بلکه یک مشخصه زبانی است که پیاده‌سازی‌های مختلف لیسپ با آن مطابقت دارد. لیسپ معمولی در مقایسه با نسخه‌هایی مانند Lisp Emacs و Auto Lisp که زبان‌های جاسازی شده در تولیدات ویژه مانند Auto CAD هستند، یک زبان برنامه‌نویسی همه منظوره است. برخلاف بسیاری از لیسپ‌های اولیه، لیسپ معمولی مانند Scheme از حوزه لغوی برای متغیرها استفاده می‌کند.

لیسپ معمولی یک زبان برنامه‌نویسی چند نمونه‌ای است به‌طوری‌که از مدل‌های برنامه‌نویسی مانند برنامه‌نویسی شئ‌گرا، برنامه‌نویسی تابعی و برنامه‌نویسی امری پشتیبانی می‌کند.

نکات تاریخی

مقاله 1958 مک کارتی دو نوع از ترکیب‌ها را معرفی کرد: S-expressions یا عبارت نمادین که Sexps هم نامیده می‌شود، و M-expressions یا عبارت غیر نمادین. امروز تقریباً همه زبان‌های لیسپ از عبارات نمادین استفاده می‌کنند.

Steele, Guy L در کتاب (.Common Lisp the Language (2nd ed به شماره ISBN 0131524143 اهداف زبان لیسپ معمولی را بیان می کند.

در سال 1986 گروه X3J13 برای استانداردسازی ANSI Common Lisp standard تشکیل شد و در سال 1992 توانست Common Lisp را در موسسه استاندارد ملی امریکا موسوم به ANSI ثبت کند.

در سال 1994 یک نسخه از لیسپ معمولی ANSI X3.226 ارائه شد.

H. Gelernter, J.R. Hansen, C.L. Gerberich در ژانویه سال 1960 قابلیت پردازش لیست‌ها را به زبان Fortran اضافه نمودن.

جان مک کارتی در اکتبر سال 1958 در آزمایشگاه هوش مصنوعی دانشگاه MIT سعی در ایجاد زبانی با قابلیت پردازش نمادین و بازگشتی داشت.

فصل 2

محیط‌های توسعه نرم‌افزاری

محیط‌های توسعه نرم‌افزاری(IDE) عمدتاً یک محیطی گرافیکی هستند، که شماری از ابزارهای لازم برای توسعه نرم‌افزار را در اختیار کاربر می‌گذارد. امکاناتی که به‌طور معمول در محیط‌های توسعه نرم‌افزاری وجود دارد عبارتند از:

- ویرایش و نوشتن کد به‌صورت پیشرفته با استفاده از امکانات پیشنهاد دهنده اتوماتیک که با نوشتن حرف اول یک دستور، نام کامل دستورهایی که وجود دارد لیست می‌شود
- نمایش کدها به‌صورت رنگی
- کمک به رفع عیب‌های نرم‌افزار و حل مشکلات آن (Debug)

محیط‌های توسعه نرم‌افزاری باعث روند سریع برنامه‌نویسی و همچنین تسهیل در امر کامپایل و اجرای برنامه‌ها را به کاربر ارائه می‌دهد.

موارد زیر، محیط‌های توسعه رایگان که برای زبان لیسپ گسترش و پیاده‌سازی شده‌اند را معرفی می‌کند:

Allegro Common Lisp

برای ویندوز، FreeBsd، لینوکس، Apple Mac OS X، یونیکس و انواع مختلف آن. Allegro CL یک محیط توسعه یک‌پارچه (برای ویندوز و لینوکس) فراهم می‌کند و قابلیت‌های گسترده‌ای را برای تحویل درخواست‌های نرم‌افزارها دارد.

Corman Common Lisp

پیاده‌سازی لیسپ معمولی برای ویندوز.

LispWorks

برای ویندوز، FreeBsd، لینوکس، Apple Mac OS X یونیکس و انواع مختلف آن. LispWorks یک محیط توسعه مجتمع (در دسترس برای همه سیستم‌عامل) با قابلیت‌های گسترده را فراهم می‌آورد.

Scieneer Common Lisp

برای محاسبات با کارایی بالا و علمی طراحی شده است.

Armed Bear Common Lisp

نوعی پیاده‌سازی لیسپ معمولی است که بر روی ماشین مجازی جاوا اجرا می‌شود. شامل یک کامپایلر افزون‌بر Java byte code است و به لیسپ اجازه دسترسی به کتابخانه‌های جاوا را می‌دهد. Armed Bear Common Lisp در حال حاضر یک پروژه مستقل است.

CMUCL

در اصل از دانشگاه کارنگی ملون آمده و در حال حاضر به‌وسیله کاربران متعدد استفاده می‌شود. CMUCL از یک کامپایلر اصلی سریع مشهور استفاده می‌کند و شامل مفسر نمی‌باشد، به‌طور گسترده‌ای از پیاده‌سازی CL با کد منبع باز استفاده می‌کند، این محیط توسعه، برای Linux و BSD با پردازنده اینتل x86 موجود است.

GNU CLIPS

نوع دیگری از لیسپ معمولی و به‌صورت متن باز است.

CLIPS

اساس پیاده‌سازی این نسخه بر اساس قاعده است. زبان CLIPS برای طراحی و پیاده‌سازی سیستم‌های خبره طراحی شده است. این زبان برای سیستم‌عامل‌های ویندوز، لینوکس، Apple Mac OS X می‌باشد.

(Steel Bank Common Lisp) SBCL

یک شاخه از CMUCL است. SBCL با تأکید بیشتر روی قابلیت نگهداری، از CMUCL باز شناخته می‌شود. SBCL روی محیط CMUCL اجرا می‌شود به‌جز HP/UX. به‌علاوه روی لینوکس برای PowerPC، SPARC و MIPS و روی سیستم‌عامل Mac توانایی اجرا دارد.

(GNU Common Lisp) GCL

این کامپایلر متن باز بوده و شامل ابزارهای ریاضی Maxima و ACL2 است.

OpenMCL

یک کامپایلر متن باز دیگر. همان‌طور که از اسم آن برمی‌آید ویژه مکینتاش است و روی سیستم‌عامل Mac، Darwin و یونیکس برای PowerPC اجرا می‌شود.

Movitz

محیط Lisp را برای کامپیوترهای x86 بدون تکیه بر سیستم‌عامل فراهم می‌کند.

Eclipse-cusp plugin

یک پلاگین رایگان برای محیط eclipse می‌باشد. این محیط بر روی تمام سیستم‌عامل‌ها قابل اجرا است.

پیاده‌سازی‌هایی همچون Franz، Xanalys، Digitool، Corman و Scieneer وجود دارند، که به‌صورت غیر رایگان توسط شرکت‌های سازنده عرضه می‌شود.

با وجود انتظارات بزرگ از کمیته استاندارد (CL گاهی به‌عنوان جایگزینی برای C معرفی می‌شود)، لیسپ معمولی یک زبان برنامه‌نویسی است که اغلب در دانشگاه و یا محیط‌های کاربردی ویژه که در ارتباط با هوش مصنوعی فعالند به‌کار می‌رود.

لیسپ معمولی در بسیاری از برنامه‌های تجاری مورد استفاده قرار گرفته است، از جمله مهم‌ترین این پروژه‌ها عبارتند از:

- بازی jak and daxter در کنسول پلی استیشن.
- OpenMusic یک محیط برنامه‌نویسی شئ‌گرا و تصویری مبتنی‌بر لیسپ معمولی.
- موتور جستجوی نرم‌افزار ITA که توسط سایت‌های گردشگری مانند orbitz و kayak.com و خطوط هواپیمایی مانند Continental Airlines، American Airlines و US Airlines استفاده می‌شود.

همچنین پروژه‌های متن بازی که با لیسپ پیاده‌سازی شده‌اند:

- **Maxima**، یک سیستم پیچیده جبر کامپیوتری.
- **Stumpwm** یک window manager که تماماً به وسیله لیسپ معمولی نوشته شده است.
- **acl2**، کاربردی، یک ثابت کننده نظریه با تمام خصیصه‌ها برای زیر مجموعه‌ای از لیسپ معمولی است.

- **Compo**، زبانی که به ساختارهای موسیقی پیچیده اجازه توصیف یک روش طبیعی را می‌دهد.
- **Lisa**، سیستم تولید قانون برای ساختن نماینده‌های نرم‌افزاری "هوش".

2-1 ویژگی‌های محیط‌های توسعه

تمام محیط‌های توسعه زبان لیسپ شامل مشترکاتی هستند. از اشتراکات بین محیط‌ها می‌توان به نحوه اجرای برنامه‌های لیسپ اشاره کرد.

روش‌های تعبیه شده برای اجرای برنامه‌های لیسپ در محیط‌های توسعه نرم‌افزاری به دو روش امکان‌پذیر می‌باشد:

1. حالت محاورهای
2. حالت اسکریپتی

حالت محاوره‌ای ساده‌ترین روش برای اجرای کد لیسپ است. در این روش کافیست عبارت خود را در قسمت مفسری لیسپ موسوم به "command, interpreter, terminal" وارد کنیم. نتیجه هر عبارت بعد از عمل REPL نشان داده می‌شود.

```
>> (+ 6 2)
8
>> (print 'Hello-World!)
HELLO-WORLD!
```

در حالت اسکریپتی کدهای برنامه لیسپ را در یک فایل با پسوند lisp. نوشته و بر روی محلی در حافظه ذخیره می‌کنیم. برای کامپایل و اجرای محتوای فایل در مفسر لیسپ کافی است آن‌را فراخوانی کنیم.

```
(load "test.lisp")
(test-compile)
```

عمل فراخوانی به‌وسیله دستور load انجام می‌شود. پس از عمل فراخوانی، باید فایل را کامپایل کنیم.

Eclipse – Plugin cusp 2-2

شاید با محیط توسعه قدرتمند eclipse آشنایی داشته باشید. این محیط توسعه رایگان که تحت پلتفرم جاوا توسعه داده شده است قابل استفاده در تمام سیستم‌عامل‌ها می‌باشد. این محیط توسعه دارای پلاگین‌های زیادی می‌باشد که امکان برنامه‌نویسی به زبان‌های مختلفی را در اختیار کاربر قرار

می‌دهد. یکی از پلاگین‌های eclipse که برای لیسپ پیاده‌سازی شده است cusp نام دارد. این پلاگین به کاربران امکان توسعه کدهای لیسپ را می‌دهد. بر اساس ساختار محیط eclipse، این برنامه قابلیت بهتری نسبت به سایر محیط‌های توسعه دارد که در ادامه با آنها بیشتر آشنا خواهیم شد.

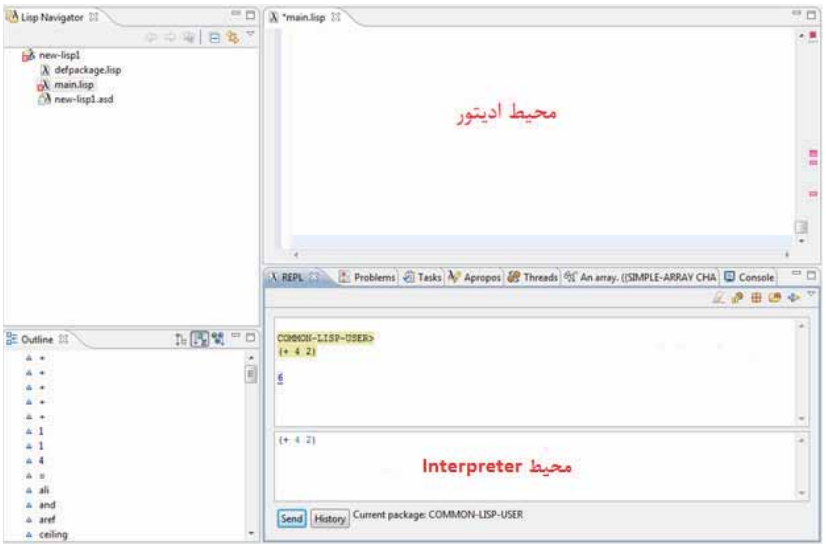
پلاگین cusp را می‌توانید از آدرس اینترنتی زیر به‌صورت رایگان دریافت کنید:

http://bitfauna.com/projects/cusp/index.html

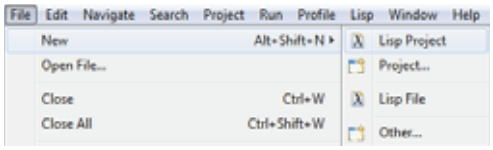
در CD همراه کتاب، محیط توسعه eclipse به‌همراه پلاگین cusp در دسترس است.

برای شروع کار ابتدا باید پلاگین cusp را به محیط eclipse اضافه کنیم. پس از اضافه کردن پلاگین، eclipse را اجرا کنید.

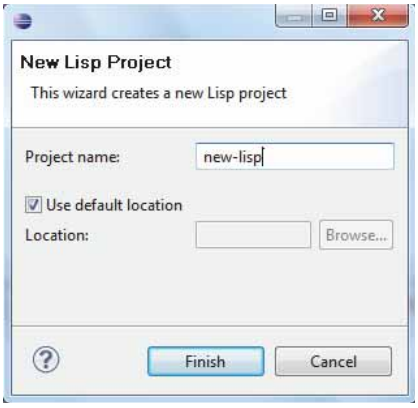
شکل زیر یک نمای کلی از محیط برنامه را نشان می‌دهد.



برای شروع کار، در مرحله اول نیاز است تا یک پروژه جدید ایجاد کنیم. برای ایجاد یک پروژه جدید، مسیر File > New > Lisp Project را دنبال کنید:



شکل2-1



شکل 2-2

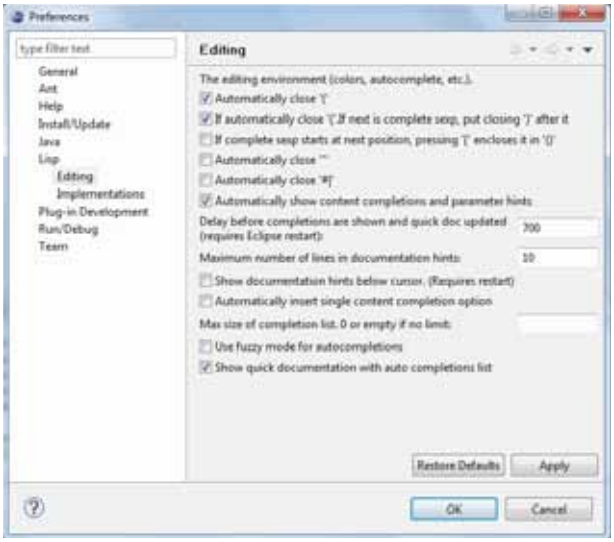
در پنجره باز شده، نام پروژه را وارد، و با پذیرش تنظیمات پیش فرض، بر روی Finish کلیک کنید.

در پروژه ایجاد شده، به‌صورت پیش‌فرض سه نوع فایل همراه پروژه ایجاد می‌گردد، که شامل فایل‌های: new-lisp1.asd، main.lisp ، و defpackage.lisp می‌باشد.

- **new-lisp.asd**: این فایل شامل توضیحاتی در مورد نحوه ایجاد پروژه است.
- **defpackage.lisp**: این فایل یک رابط می‌باشد (تقریباً شبیه فایل‌های هدر در c/c++).
- **main.lisp**: این فایل در برگیرنده کدهای اصلی برنامه است.

در هنگام اجرای پروژه، عمل بارگذاری و کامپایل نیز به‌طور اتوماتیک صورت می‌پذیرد.

پلاگین cusp قابلیت سفارشی‌سازی محیط را به کاربران ارائه می‌دهد. عملیات سفارشی‌سازی باعث تسهیل در امر تایپ و کدنویسی می‌شود. برای سفارشی‌سازی، مسیر Window > Preferencesرا دنبال کنید.

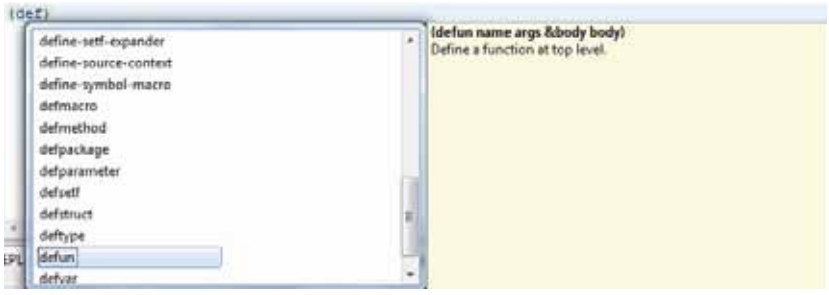


شکل 2-3

در پنجره ظاهر شده، تنظیمات محیط ادیتور برنامه را مشاهده می‌نمایید.

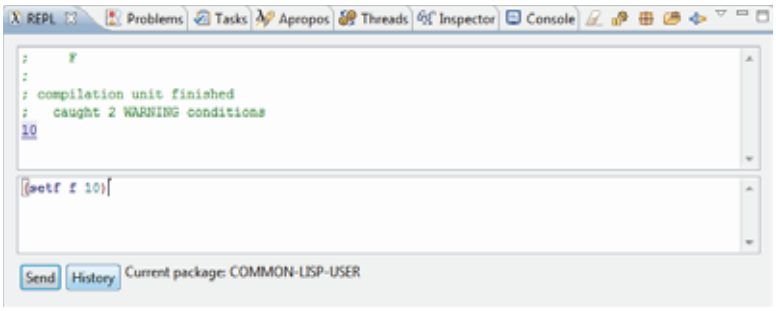
زبان لیسپ به‌علت پرانتزهایی که در هر عبارت به‌کار برده می‌شود دارای سینتکس نسبتاً پیچیده‌ای است، و این خود موجب بروز مشکلات عمده در خطاهای سینتکس (نحوی) در آن می‌گردد. برای جلوگیری از خطای نحوی در گذاشتن پرانتزها در کدهای لیسپ، کافیست در بخش سفارشی‌سازی گزینه ')' Automatically Close را فعال کنیم. در این‌صورت ادیتور لیسپ هر پرانتزی که باز می‌شود را به‌طور اتوماتیک می‌بندد. تمام گزینه‌های Automatically Close می‌تواند در نوشتن سریع کدها و همچنین جلوگیری از خطاهای نحوی مؤثر باشند.

گزینه Automatically Show Content، ...show quick documentation باعث می‌شود تا در هنگام تایپ لیستی از دستورات، توضیحاتی در مورد آن دستور برای ما نشان داده شود. عملکرد این دو گزینه را می‌توانید در شکل زیر مشاهده کنید.



شکل 2-4

در شکل زیر برگه‌های مختلفی وجود دارد که مهم‌ترین آنها REPL است که از دو ناحیه مجزا تشکیل می‌شود:

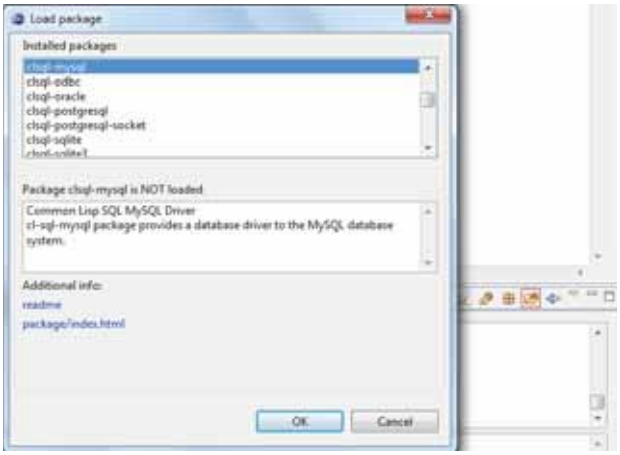


شکل 2-5

در ناحیه بالایی پنجره، حاصل مقداریابی عبارت‌ها نمایش داده می‌شود. در ناحیه پایینی که در اصطلاح Interpreter گفته می‌شود می‌توانیم عبارت‌های خود را به مفسر لیسپ ارسال کنیم.

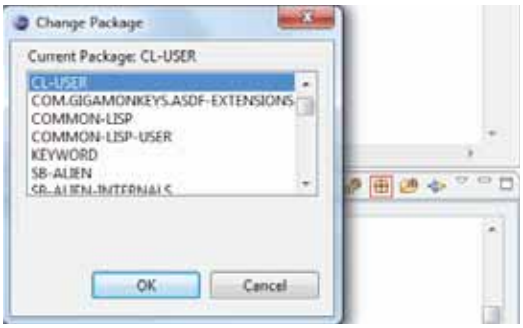
هر عبارتی که در لیسپ کامپایل می‌شود در package جاری پروژه یعنی در new-lisp اضافه می‌گردد. Packegeها در اصل بسته‌هایی شامل توابع و دستورات از پیش تعیین شده هستند که به برنامه‌های لیسپ قابلیت استفاده از توابع جاسازی شده یا بیرونی را می‌دهد. پکیج جاری پروژه، به‌طور پیش‌فرض common-lisp-user است.

در cusp برای تغییر داده پکیج جاری پروژه در برگه REPL، از گزینه Change Package استفاده می‌شود. همچنین برای اضافه کردن پکیج جدید در همان برگه، از گزینه load install package استفاده می‌شود.



شکل 2-6

شکل بالا نحوه تغییر package جاری سیستم را نشان می‌دهد. همچنین شکل زیر نحوه اضافه کردن package جدید به پروژه را نشان می‌دهد.

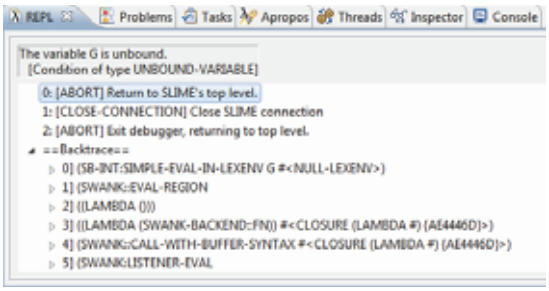


شکل 2-7

با تغییر package دوباره محتویات آن توسط REPL مقداریابی می‌شود و نتیجه در برگه outline، که حاوی توابع package مورد نظر است نشان داده می شود. مثال بالا Packege برنامه را از common-lisp-user به cl-user تغییر داده است.

خطایابی

خطایابی در اصطلاح برنامه‌نویسی به‌عمل شناسایی خطای سینتکس و ساختاری در یک زبان برنامه‌نویسی گفته می‌شود. اگر هر نوع عبارت نادرست را در REPL اجرا کنیم، خطایاب cusp علاوه‌بر اینکه خطای مورد نظر را برای ما نشان می‌دهد، می‌تواند راه‌حلی برای برون رفت از مشکل نیز به‌ما ارائه دهد.

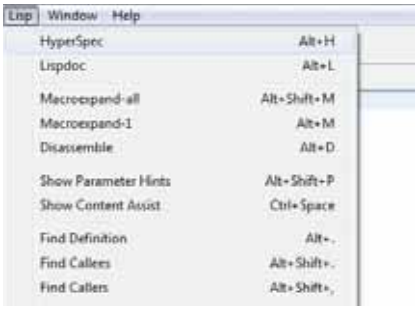


شکل 2-8

پلاگین cusp همچنین دارای راهنمای آنلاین است، که کاربر با مراجعه به این راهنما می‌تواند اطلاعات کافی در مورد دستورات لیسپ داشته باشند. برای به‌دست آوردن اطلاعات در مورد دستور مورد نظر کافی است بعد از انتخاب دستور مورد نظر، از منویLisp بر روی گزینه HyperSpec کلیک کنید.

برای استفاده از راهنمای آنلاین باید به شبکه اینترنت متصل باشید.

این راهنما اطلاعاتی جامع و کامل را از پایگاه اینترنتی HyperSpec در اختیار کاربر می‌گذارد.



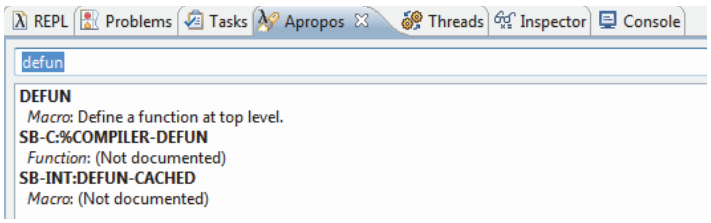
شکل 2-9

شکل 2-10

همچنین گزینه lispdoc اطلاعات جامع و کاملی در مورد دستور مورد نظر از پایگاه اینترنتی lispdoc.com در اختیار کاربر می‌گذارد.

شکل 2-11

همچنین برگه Apropos می‌تواند اطلاعاتی مختصر در مورد دستورات، که به‌صورت یک راهنمای آفلاین در پلاگین cusp تعبیه شده است، را به‌ما ارائه دهد.



شکل 2-12

فصل 3

ساختار دستوری و معناشناسی در لیسپ

زبان‌های برنامه‌نویسی برای ایجاد برنامه‌هایی به‌کار می‌روند که رفتار یک ماشین را مشخص می‌کنند، الگوریتم دقیق را بیان می‌کنند، و یا روشی برای ارتباط انسان با ماشین هستند. با بیان ویژگی زبان‌های برنامه‌نویسی، این نکته حائز اهمیت می‌شود که هر زبان باید دارای ساختار دستوری (syntax) و معناشناسی (semantics) باشد.

ساختار دستوری یک زبان برنامه‌نویسی، اسکلت برنامه شما را تشکیل می‌دهد. متن زیر یک گرامر ساده، به زبان lisp است:

```
expression ::= atom | list atom  ::= number | symbol number  ::= [+-]?['0'-'9']+ symbol  ::= ['A'-'Za'-'z'].* list  ::= '('
expression* ')'
```

این گرامر موارد ذیل را مشخص می‌کند:

- یک عبارت یا atom است و یا یک لیست
- یک atom یا یک عدد است و یا یک سمبل
- یک عدد دنباله ناشکسته‌ای از یک یا تعداد بیشتری اعداد دهدهی است، که یک علامت مثبت و یا منفی می‌تواند پیش از آن بیاید.
- یک سمبل حرفی است که بعد از هیچ یا تعدادی کاراکتر (جز فاصله) می‌آید.
- یک لیست، تعدادی پرانتز است که می‌تواند صفر یا چند عبارت در خود داشته باشد.

"12345"، "()"، "((232 a b c) 1)" مثال‌هایی هستند از دنباله‌های خوش فرم در این گرامر.

شاید یک عبارت از لحاظ دستوری درست باشد، ولی ممکن است از نظر نحوی یا معنایی درست نباشد. این امر باعث می‌شود تا عبارت‌ها رفتار نامشخصی را انجام دهند. به مثال زیر توجه کنید:

"او یک مجرد متأهل است." از نظر دستوری درست است، ولی معنایی را بیان می‌کند که نمی‌تواند درست باشد. یعنی کسی که مجرد است نمی‌تواند متأهل باشد و همچنین کسی که متأهل است نمی‌تواند مجرد باشد.

وقتی که داده مشخص شد، ماشین باید هدایت شود تا عملیاتی را روی داده انجام بدهد. معناشناسی یک زبان، اجرای ماشین را این‌گونه تعیین می‌کند که چگونه و چه زمانی ساختارهای گوناگون باید رفتار برنامه را ایجاد کنند. معناشناسی یک برنامه، ساختار ثابتی ندارد بلکه ساختاری تصادفی دارد مانند دستورات مختلف و متغیر مجاز که می‌توانند در هر کجای برنامه مورد استفاده قرار گیرند.

دستورات زبان لیسپ در قالب عباراتی که S-expressions نامیده می‌شود به interpreter لیسپ ارسال می‌گردد. در نسخه‌های اولیه لیسپ این عبارات به‌صورت M-expressions نامیده می‌شدند car[cons[A,B]]، و بعدها به‌دلیل استفاده از لیست‌ها در پردازش نمادین، به S-expressions تغییر داده شد (car (cons A B)). در این تغییر، لیست‌ها ساختار مرکزی لیسپ شدند که برای نشان دادن داده‌ها و برنامه‌ها به‌کار برده می‌شود.

تمام عبارات در لیسپ از دو نوع ساختار اتم (Atom) و لیست (List) تشکیل شده‌اند:

اتم‌ها می‌توانند یک عدد، یک کاراکتر یا یک نماد باشند. در حالی‌که لیست‌ها می‌توانند دنباله‌ای از اتم‌ها و لیست‌های دیگر باشند.

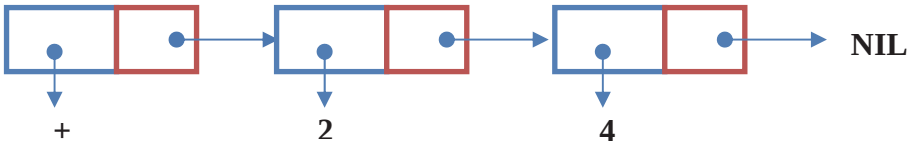
```
20          ;Atom
16.6452     ;Atom
a           ;Atom
(2 5 7 8 10 14 20)      ;List
(hello 3 why? 4.56 "bla" bla)     ;List
```

تکیه روی عبارت‌ها، قابلیت انعطاف‌پذیری زیادی به زبان می‌دهد، زیرا توابع لیسپ به‌صورت لیست نوشته شده‌اند و دقیقاً مانند داده‌ها می‌توانند پردازش شوند. این قابلیت اجازه می‌دهد برنامه‌های لیسپ به سادگی و راحتی نوشته و به‌نسبت برنامه‌های دیگر به‌راحتی اداره شوند.

کار هسته هر مفسر لیسپ، محاسبه مقدار برای یک عبارت نمادین داده شده است و آن‌را طی فرآیند ارزیابی که REPL نامیده می‌شود انجام می‌دهد. نتیجه یا مقدار یک عبارت نمادین، خود نیز یک عبارت نمادین دیگر است، که پس از کامل شدن ارزیابی برگردانده می‌شود. توضیح اینکه در واقع لیسپ دارای معناشناسی عملیاتی است و با یک تعریف ریاضی دقیق از نظریه تابع بازگشتی به‌دست می‌آید. عمل REPL (مخففRead-Eval-Print) پردازش یا ارزیابی مقدار عبارت‌ها را در سه مرحله انجام می‌دهد؛ در مرحله نخست، هر نوع عبارت یا S-expression در لیسپ، پیش از پردازش ابتدا به لیست‌های پیوندی تبدیل می‌گردد، سپس نمادها مقداریابی شده و در آخر، پردازش و چاپ نتیجه در قالب لیست پیوندی انجام می‌گیرد. عبارت زیر را در نظر بگیرید:

مثال:

```
(+ 2 4)
```



در عبارت بالا در مرحله Read عبارت به لیست پیوندی که نشان داده شده است تبدیل، سپس در مرحله Eval مقدار هر عنصر لیست ارزیابی و در مرحله Print حاصل عبارت در قالب داده‌ای لیست بازگشت داده خواهد شد.

به‌طور کلی مفسر لیسپ عبارت‌ها را در دو حالت، شناسایی و تفسیر می‌کند؛ حالت Code mode و حالت Data mode. در حالت Data mode عناصر هر عبارت به‌عنوان یک داده تفسیر و مقداریابی می‌شود. اما در حالت Code mode مفسر لیسپ عنصر اول را به‌عنوان تابع و عناصر بعدی را به‌عنوان پارامتر های ورودی آن تفسیر و مقداریابی می‌کند.

```
(min 5 6 2 8 1)
```

```
Function    A Forn
```

مثال بالا حالت Code mode را بیان می‌کند، که عنصر اول یعنی min را به‌عنوان تابع مقداریابی می‌کند و بقیه عناصر را پارامتر ورودی آن در نظر می‌گیرد. در نهایت پارامترها را به تابع ارسال و نتیجه را بازگشت می‌دهد.

در مثال زیر عبارات به‌صورت Data mode تفسیر و مقداریابی می‌شود. دلیل این امر استفاده از نمادهای list و " ' " در عنصر اول عبارات است.

```
(list hello 4 "hello" 6.78)
'(hello 4 "hello" 6.78)
```

نکته حائز اهمیت دیگر در مورد عبارات در لیسپ، نحوه پیمایش آنهاست. روشی که لیسپ از آن برای پیمایش لیست‌ها استفاده می‌کند، روش prefix است. پس برای اجرای حالت درست عبارات باید آنها را به‌صورت prefix نوشت. در ادامه با این موضوع بیشتر آشنا خواهید شد.

به‌دلیل ساختار هسته لیسپ، هر عبارت باید با پرانتز باز لاتین "(" شروع و با پرانتز بسته ")" خاتمه یابد. در غیر این‌صورت عبارت نادرست است.

یک ویژگی دیگر زبان لیسپ نداشتن بدنه اصلی برنامه‌ها می‌باشد. در اکثر زبان‌های برنامه‌نویسی، اجرای کلی برنامه‌ها در یک تابع که بدنه اصلی برنامه خوانده می‌شود انجام می‌پذیرد، این در حالی

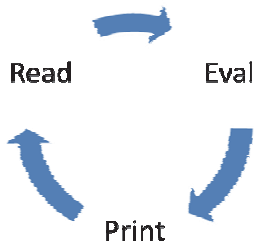
است که در لیسپ هر تابع در صورت فراخوانی می‌تواند به‌صورت مستقل اجرا شود و حتی می‌تواند توابع دیگر را در داخل بدنه خود نیز فراخوانی و اجرا نماید.

یک نکته مهم دیگر در زبان لیسپ، این است که داده‌ها نیازی به تعریف نوع ندارند. این در حالی است که داده‌ها بر اساس نوع مقدار و عملی که بر روی آن انجام می‌پذیرد، نوع داده‌ای (Data Type) آنها توسط لیسپ تعیین می‌شود. ممکن است یک قالب داده‌ای در طی روند اجرای برنامه، انواع مختلف نوع داده‌ای را در خود نگهداری کند. مشابه این عمل‌کرد را می‌توان در زبان PHP مشاهده کرد.

1-3 مقداریابی در لیسپ

مقداریابی یا REPL، سه مرحله پردازش در مفسر لیسپ را بیان می‌کند. این مراحل به ترتیب و طبق قوانین زیر مورد پردازش قرار می‌گیرند:

1. نوع ورودی تعیین می‌شود.
2. یک معنا به‌عبارت انتساب داده می‌شود.
3. چاپ مقدار ارزیابی شده یا چاپ پیغام خطا در صورتی‌که عبارت نامعتبر باشد.



شکل 1-3

همان‌طور که در شکل بالا مشاهده می‌کنید این فرآیند به‌صورت چرخشی تا رسیدن به انتهای عبارت ادامه می‌یابد.

در مرحله اول یعنی Read، مفسر پس از خواندن عبارت، آن را به عبارت‌های مجاز که در ساختار حافظه به‌صورت لیست‌های پیوندی پیاده‌سازی می‌شود تبدیل می‌کند. اگر نتیجه عبارت صحیح باشد، انجام این فرآیند به مرحله بعدی که Eval است انتقال می‌یابد. در این مرحله مقدار هر عبارت ارزشیابی می‌شود، که شامل موارد زیر است:

1. اگر عبارت یک اتم باشد:
   - اگر عبارت عدد باشد حاصل مقداریابی آن خود، عدد است.