На этом шаге мы поговорим немного об основах языка программирования **LISP**.

Вначале заметим, что в основе языка **LISP** лежит лямбда-исчисление. Заглянем в словарь [1]: "***Лямбда-исчисление (Lambda calculus)*** - формализм для представления функций и способов их комбинирования. Вместе со своим эквивалентом - ***комбинаторной логикой***, в которой не используются переменные, - предложено около 1930 г. логиками Черчем, Шейнфинкелем и Карри.

Примеры выражений:

- **lambda x.x** - тождественная функция со значением просто равным ее аргументу;
- **lambda x.c** - постоянная функция со значением, равным c, вне зависимости от аргумента;
- **lambda x.f(f(x))** - композиция функции f с самой собой, т.е. функция со значением, равным f(f(x)) для аргумента x.

Широкие возможности рассмотренного способа обозначения во многом объясняется возможностью представления с его помощью функций более высоких порядков.

Например, запись **lambda f.lambda x.(f(x))** соответствует функции (высшего порядка), значение которой для аргумента x равно функции, получаемой путем композиции функции f с самой собой.

Лямбда-исчисление дает не просто форму записи, но также правила эквивалентных преобразований лямбда-выражений. Наиболее важным является правило **betta-*редукции***, при помощи которого можно упрощать выражения вида:

lambda (x.e1)(e2)

Например, выражение (lambda x.f(x,x))(a) betta-редуцируется в f(a,a).

В сочетании с некоторыми простейшими функциями лямбда-исчисление дает оригинальный способ определения множества всех эффективно вычислимых функций от неотрицательных целых чисел - следовательно, в этом смысле ***оно эквивалентно машине Тьюринга и аппарату теории рекурсивных функций***.

Лямбда-исчисление используется в информатике для разработки целого класса языков программирования (в частности, языков LISP, PAL, POP-2). Более того, математическая теория, разработанная в качестве первой теоретико-множественной модели лямбда-исчисления, послужило основой так называемой ***денотационной семантики*** языков программирования".

Х.Барендрегт пишет [2,с.16]: "Анализ, проведенный Тьюрингом, показывает, что, несмотря на свой очень простой синтаксис ***лямбда-исчисление*** способно изобразить все механически вычислимые функции. Поэтому на лямбда-исчисление можно смотреть как на парадигматический язык программирования. Отсюда, конечно, не следует, что мы должны писать на нем настоящие программы. Подразумевается лишь, что многие проблемы, возникающие в программировании, особенно в связи с вызовами процедур, предстают в лямбда-исчислении в чистом виде. Их изучение может принести пользу при проектировании и анализе языков программирования.

Например, ряд языков программирования обладает (быть может, без прямого намерения своих создателей) чертами, навеянными лямбда-исчислением. В алголе 60, алголе 68, Паскале процедуры могут быть аргументами процедур. В Лиспе, кроме того, процедура может быть результатом работы процедуры".

Дж.Саммит, известный специалист по языкам программирования, как-то сказала, что все языки программирования можно грубо разбить на два класса. В одном находится **LISP**, в другом - все остальные языки программирования [3,с.6].

Язык **LISP** разработан в Стэнфорде под руководством ***Дж.Маккарти*** в начале 60-х годов. По первоначальным замыслам он должен был включать наряду со всеми возможностями Фортрана средства работы с матрицами, указателями, структурами из указателей и т.п. Предполагалось, что первые реализации будут интерпретирующими, но в дальнейшем будут созданы компиляторы, транслирующие **LISP**-конструкции в машинный код. К счастью, для такого проекта не хватило средств. К тому же к моменту создания первых **LISP**-интерпретаторов в практику работы на ЭВМ стал входить диалоговый режим, а режим интерпретации естественно вписался в общую структуру диалоговой работы. Примерно тогда же окончательно сформировались и принципы, положенные в основу языка **LISP**: использование единого спискового представления для программ и данных, применение выражений для определения функций, скобочный синтаксис языка. Процесс разработки языка завершился созданием версии **LISP** 1.5, которая на многие годы определила путь его развития и совершенствования [4].

Таким образом, **LISP** представляет собой интерпретирующую систему, а это позволяет значительно облегчить и ускорить процесс создания сложных комплексов программ в интерактивном режиме, так как обеспечивает немедленную реакцию системы на изменения, вносимые пользователем, и предоставляет мощные средства отладки и редактирования программ.

"Лисп был для нас не просто языком, который используют для определенных целей, - говорил Пол Абрахамс, бывший дипломником у Маккарти в период разработки нового языка, - им можно было любоваться, как прелестной вещью. Поэтому существовала постоянная напряженность в отношениях между теми, кто восхищался LISPом за его чистоту, и теми, кто стремился использовать его для различных вычислений. Безусловно, с помощью LISPа проделано множество вычислений. Но в самом начале было не так. Часто говорили, что главная цель LISPа - делать больше LISPа."[3]

Если исходить из базисного набора примитивов (CAR, CDR, CONS, COND, ATOM, EQ), **LISP** является языком низкого уровня. И с этой точкой зрения его можно рассматривать как ассемблер, ориентированный на работу со списковыми структурами. Поэтому на протяжении всего существования

языка было много попыток его усовершенствования за счет введения дополнительных базисных примитивов и управляющих структур. Однако все изменения, как правило, не прививались в качестве самостоятельных языков. И причин здесь несколько. С одной стороны, в большинстве случаев создатели новых языков оставались в "лисповской" парадигме, не предлагая нового взгляда на программирование. С другой - новые языки, как правило не имели собственной программной среды, а "жили" в **LISP**-среде и поэтому воспринимались как часть этого языка. В новых своих редакциях **LISP** быстро "усваивал все ценные изобретения конкурентов". Наконец, немаловажную роль в распространении языка **LISP** и утверждении его в качестве основного языка интеллектуальных систем сыграли авторитет Стэнфордской школы и лично Дж.Маккарти в области искусственного интеллекта, а также введение его как обязательного для изучения студентами во всех учебных заведениях США, связанных с проблематикой искусственного интеллекта.

LISP как представитель языков функционального программирования обладает воистину удивительными чертами.

Первая наиболее поразительная черта - это эквивалентность представления программ и данных в языке, что позволяет интерпретировать структуры данных как программы и модифицировать программы как данные. Метод программирования, в котором внешние к программе данные используются с целью управления работой программы или сами интерпретируются в качестве программы, называется ***программированием, управляемым данными***. В программировании, управляемом данными, программы хранятся вместе с данными или с отображениями их типов. Таким образом, необходимые в текущий момент функции можно определить или найти, исходя из данных. В языке **LISP** программирование, управляемое данными, легко применимо в связи с единообразной формой представления данных и программы.

Второй удивительной особенностью является применение в качестве основной управляющей структуры не итерации (цикла), как в языках императивного программирования, а рекурсии.

Третья особенность состоит в широком использовании структуры данных "связанный список", поэтому обработка списков лежит в основе большинства алгоритмов языка **LISP**.

Простота синтаксиса LISPa являются одновременно и достоинством и недостатком. Начинающий программист может выучить основные правила синтаксиса за несколько минут, и тогда написание программы в основном сводится (синтаксически) к правильной записи аргументов для каждого вызова функции. Проблемой этого простого синтаксиса являются скобки. В каждом выражении должны быть расставлены все необходимые скобки, а поскольку тело любой функции представляет собой одно гигантское выражение, в определении функции часто накаплвается до 10-15 уровней вложенных скобок. Такие конструкции чрезвычайно трудно читать и отлаживать, и ошибка в расположении скобок является, по-видимому, наиболее типичной синтаксической ошибкой в языке **LISP**. К сожалению, неправильное расположение скобок во многих случаях формально не рассматривается, как синтаксическая ошибка (т.е. оно не может быть выявлено во время трансляции), ошибочное выражение часто выглядит "осмысленно", но его семантика (смысл) не совпадает с тем, которые в него хотели вложить. Существует даже шутка: LISP - "Lots of Idiotic Silly Parentheses" (переведите самостоятельно). Логические ошибки такого рода очень трудно найти, а плохая читаемость определений функций удваивает трудность этой задачи.

Одним из основных недостатков языка **LISP** традиционно считалась относительно невысокая скорость выполнения программ. Однако с появлением мощных **LISP**-машин и с разработкой эффективных компиляторов с языка **LISP** скорость исполнения программ значительно увеличилась. Относительное неудобство в освоении языка **LISP** состоит в том, что существует много диалектов и, к сожалению, ни один из них не принят в качестве стандарта. Однако сейчас есть надежда, что таким стандартом станет **Common Lisp**.

Язык **LISP** входит сегодня в программное обеспечение почти всех ЭВМ, выпускаемых за рубежом. Большие компьютеры IBM снабжаются интерпретаторами и компиляторами с диалектов **InterLISP** и **Common LISP**,

компьютеры DEC - системами **Franz LISP**, **InterLISP, XLISP**. На ПЭВМ наибольшее распространение получили системы **Golden Common LISP, muLISP, IQ-LISP**.

Если же говорить о глобальной тенденции развития самой идеологии языка **LISP**, то очевидно, что она связана с созданием объектно-ориентированных версий языка как наиболее пригодных для реализации ***систем искусственного интеллекта***.

Хотелось бы отметить ещу один раздел информатики, в которой широко используется язык **LISP**.

Компьютерная алгебра есть та часть информатики, которая занимается разработкой, анализом, реализацией и применением алгебраических алгоритмов.

От других алгоритмов алгебраические алгоритмы отличаются наличием простых формальных описаний, существованием доказательств правильности и асимптотических границ времени выполнения, которые можно получить на основе хорошо развитой математической теории. Кроме того, алгебраические объекты можно точно представить в памяти вычислительной машины, благодаря чему алгебраические преобразования могут быть выполнены без потери точности и значимости. Обычно алгебраические алгоритмы реализуются в программных системах, допускающих ввод и вывод информации в символьных алгебраических обозначениях.

Выбор языка **LISP** в качестве ***языка реализации алгоритмов компьютерной алгебры*** является естественным. Это обусловлено, во-первых, необходимостью динамического управления памятью в связи с проблемой разбухания промежуточных выражений. Во-вторых, рекурсивность и списковая структура являются естественными инструментами при автоматизации работы с математическими объектами и операциями [5, с.282].

Дальше мы рассмотрим **muLISP** - один из самых удачных диалектов языка **LISP**, созданный фирмой Soft Warehouse Inc (США) [6]. Отметим, что

существуют несколько реализаций: **muLISP81, muLISP83, muLISP85, muLISP87**.

muLISP87 - система, имеющая относительно немного встроенных функций (порядка 400), но зато весьма компактная и обеспечивающая создание эффективных программ. Предусмотрен встроенный текстовый редактор (правда, не очень удачный), интерфейс с Ассемблером, поставляются исходные тексты системы объектно-ориентированного программирования Flavors. Средства отладки системы более чем скромные. Некоторые оригинальные решения и новые функции системы позволяют создавать очень эффективные, хотя и не вписывающиеся в классические принципы языка **LISP** программы. Система снабжена очень полной и хорошо написанной документацией. К сожалению, по синтаксису пакет **muLISP** не совместим со стандартом **Common LISP** даже при включении в него поставляемой фирмой библиотеки функций **Common LISP**. Это серьезный недостаток в настоящее время. Однако **muLISP** - очень мощный и удобный пакет, позволяющий эффективно решать не только задачи искусственного интеллекта, но и нетрадиционные для языка **LISP** задачи, например, позволяет разработать собственную оконную систему, организовать обмен с программами на языке C++ и т.д.

Система программирования **muLISP-90** является "маленьким LISPом", который работает на IBM PC (или на HP 95LX palmtop), используя операционную систему MS-DOS версии 2.1 или более поздние.

Конечно, система **muLISP90** - это не **Common Lisp**, хотя в **muLISP** имеется пакет для совместимости с **Common Lisp**, содержащий более 450 специальных форм, макросов, функций и управляющих переменных **Common Lisp**.

Система включает экранный редактор, отладчик (debugger), оконную систему, интерпретатор и компилятор. Среди многочисленных примеров программ имеется DOCTOR ("Eliza-подобная" программа). Система времени выполнения (run-time system) позволяет создавать небольшие EXE- или COM-файлы. Она использует компактное внутреннее представление кода,

обеспечивающее минимизацию памяти и увеличение скорости выполнения. Ядро занимает всего 50K.

Дальше мы рассмотрим вопросы обучения функциональному программированию с использованием диалектов **muLISP81, muLISP83** и **muLISP85**.