

ЛИСП один из древнейших языков программирования, по возрасту второй после **ФОРТРАНа**. Он был разработан в конце 50-х годов прошлого века в **MIT**, и впервые опубликован **Джоном Маккарти** в 1960г. С тех пор язык развивался независимыми группами разработчиков и дошел до наших дней в виде нескольких диалектов. Наиболее известные из них **Scheme** и **CommonLisp**. Хотя оба диалекта имеют общие корни, они весьма существенно отличаются друг от друга.

На наших занятиях мы будем использовать диалект **Scheme** по двум причинам. Во-первых, в нашем распоряжении имеется свободно распространяемая интегрированная среда разработки программ - **DrRacket**, поддерживающая диалект **Scheme**. Во-вторых, в переводе на русский язык имеется очень хорошая книга **Абельсона** и **Сассманов** «Структура и интерпретация компьютерных программ», в которой понятно и увлекательно изложены основы диалекта **Scheme**.

Актуальную версию **DrRacket** можно найти по ссылке <http://racket-lang.org>, а первую главу книги в файле **AbelsonChapter1.pdf**.

Хочу подчеркнуть, что **ЛИСП** интересует нас не как инструмент для разработки реальных программ. Нам не нужна вся мощь этого языка, во многом отличающегося от других языков программирования. В фокусе нашего внимания будут те конструкции **ЛИСПа**, которые сближают его с другими языками. Из этих конструкций мы составим подмножество **ЛИСПа** и будем отрабатывать на нем методы компиляции, общие для разных языков программирования.

Учебное подмножество **ЛИСПа** мы будем называть **МИКРОЛИСП**.

МИКРОЛИСП – собственное подмножество языка **ЛИСП**. Это означает, что любая конструкция, допустимая в **МИКРОЛИСПе**, допустима в **Лиспе**. Любая конструкция недопустимая в **ЛИСПЕ**, недопустима и в **МИКРОЛИСПе**.

Имеется множество конструкций, допустимых в ЛИСПе, но недопустимых в МИКРОЛИСПе.

Такое отношение между языками позволяет выстроить удобную и эффективную систему верификации компилятора.

- 1. Программа на МИКРОЛИСПе подготавливается и выполняется в среде DrRacket.**
- 2. Затем эта программа компилируется в C++.**
- 3. Полученная целевая программа выполняется в системе программирования на C++.**

Если результаты вычислений в ЛИСПе и C++ совпадут, то можно с уверенностью утверждать, что компилятор работает правильно и выдает целевую программу, эквивалентную исходной.

На последующих занятиях, после того как вы освоите необходимый математический аппарат, мы составим точное формальное описание языка. Сегодня я ограничусь кратким неформальным введением в МИКРОЛИСП.

Краткое введение в МИКРОЛИСП.

Алфавит языка включает все символы, имеющие изображение на клавиатуре компьютера, в том числе русские буквы.

Минимальная значимая единица текста называется атомом.

Атом – это цепочка символов алфавита, ограниченная пробелом или знаком пунктуации.

***Пробелом* считается класс символов, включающий собственно символ пробела, символы TAB(табуляция) и LF (переход на новую строку). Последовательность пробелов эквивалентна одному пробелу.**

В тексте программы можно записать *комментарий*. Он начинается с ; и простирается до конца строки текста. Комментарий эквивалентен пробелу.

К знакам пунктуации относятся () “

Все атомы делятся на несколько крупных классов.

1. Числовые десятичные литералы. Правила записи те же, что в C++, например, 1,123, 2.5, 1e-5 и т.п.
2. #t , #f обозначают булевские константы true и false.
3. Строковые литералы записываются как в C++, например, "abc", "()"'\'", "c:\\Program Files".
4. Операторы: + - * / = < > <= >=
5. Идентификаторы – это символические имена, составленные из латинских букв, цифр и знаков ! ? -
6. Все прочие атомы считаются ошибочными.

Фразы и предложения МИКРОЛИСПа называются **выражениями**.

Элементарное выражение – атом.

Составное выражение – *список* вида

(e1 e2 ...), где e_i – атом или список.

В силу приведенного рекурсивного определения правильное выражение содержит сбалансированный набор скобок.

Программа на МИКРОЛИСПе – это последовательность выражений.

Имеется общий механизм вычисления выражений.

1. Значение атома-литерала содержится в его записи.
2. Атом-идентификатор рассматривается как имя переменной. В процессе выполнения программы он должен быть *связан* со значением перед тем, как будет выполнено обращение к переменной.
В языке есть предопределенные константы e и pi.
3. Если выражение – список, то «голова»(первый элемент) списка задает имя процедуры, а его «хвост» - последовательность аргументов. Сначала вычисляются значения аргументов, а затем к ним применяется процедура.

Для удобства записи программ в язык включены *особые формы* – выражения, к которым применяются специальные правила вычисления.

См. AbelsonChapter1.pdf стр. 4-17.

Рассмотрим некоторые из них на примерах.
Форма `(define x 1)` определяет(создает) переменную с именем `x` и связывает ее со значением `1`.

Форма `(define(f x)(* 2 x))` определяет процедуру с именем `f` и параметром `x`. В теле процедуры `x` выступает в качестве локальной переменной.

При вызове `(f 3)` значение аргумента связывается с параметром, вычисляются тело и его значение возвращается в качестве значения процедуры.

Тело процедуры может состоять из нескольких выражений. Все они последовательно вычисляются, и процедура возвращает значение последнего.

Условная форма
`(cond ((< a b) 1)
 ((= a b) 0)
 ((> a b) 1))`

состоит из списка ветвей – клауз.

Вначале клаузы записано булевское выражение – предикат, принимающий значения `#t`(истина) или `#f`(ложь). За предикатом следует тело ветви.

При вычислении формы последовательно просматриваются ветви и выбирается та, у которой предикат имеет значение `#t`. Для выбранной ветви вычисляется тело и его значение становится значением формы.

Последующие ветви не просматриваются.

Форма `cond` корректно определена, если хотя бы один из предикатов имеет значение `#t`. В приведенном выше примере это условие выполняется.

В следующем примере корректность обеспечивается клаузой, в которой явно прописан предикат `#t`.

```
(cond ((= a b) 0)  
      (#t 1) )
```

Вместо константы `#t` можно записать ключевое слово `else`.

```
(cond ((= a b) 0)  
      (else 1) )
```

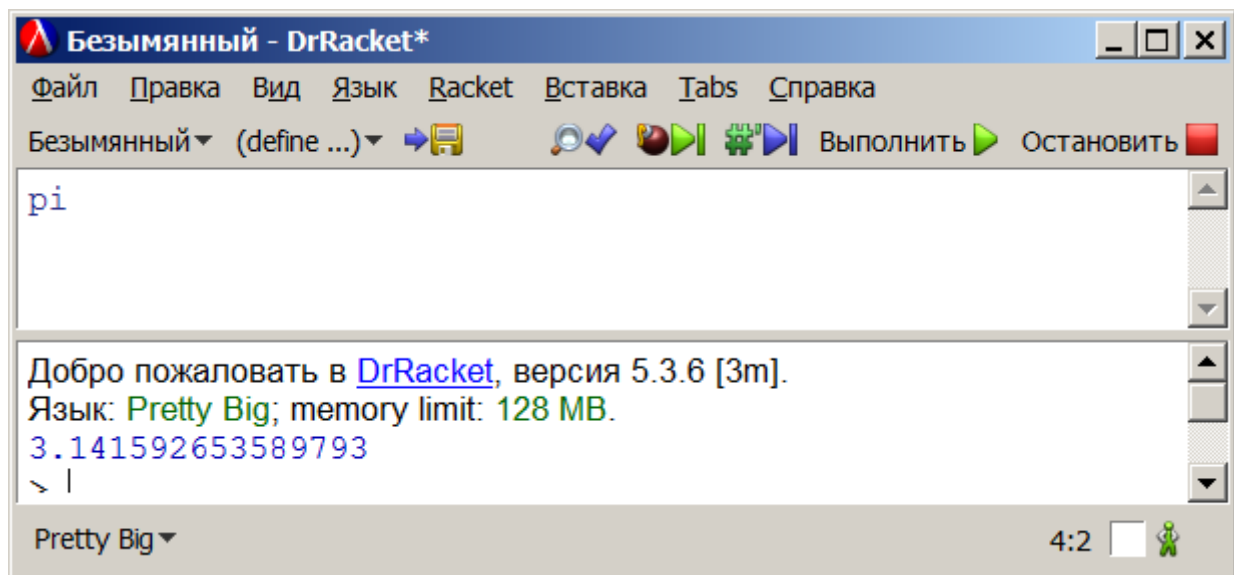
Условная форма `(if(= a b) 0 1)` имеет три аргумента – предикат, следствие и альтернативу.

Если предикат принимает значение #t, вычисляется следствие, в противном случае – альтернатива.

NB!!! Для выполнения заданий практикума DrRacket следует настроить на язык **Pretty Big!**

Главное>Язык>Выбрать язык>Pretty Big

Ниже приведен скриншот окна DrRacket. На панели определений записан атом-идентификатор, обозначающий встроенную константу π . После применения команды **Выполнить** на панели интерпретатора отображено значение константы.



В следующем примере на панель определений командой **Файл->Открыть** загружена программа из файла **kpg.ss**. Ранее эта программа была сохранена командой **Файл->Сохранить определения как**. После применения команды **Выполнить** на панели интерпретатора отображены значения выражений, составляющих программу.

```
kpg.ss - DrRacket
Файл  Правка  Вид  Язык  Racket  Вставка  Tabs  Справка
kpg.ss (define ...)
-0
1e-5
e
(exp 1)

0
1e-005
2.718281828459045
2.718281828459045
>

Pretty Big 7:2
```

Загрузите и выполните программы из файлов: L01.ss, L02.ss, L03.ss, L04.ss, L05.ss, factorial.ss. Используя полученные результаты, разберитесь в правилах вычисления выражений.