

Рефал-5 Version ПЗ

Руководство пользователя

(версия 17-09-2023)

Ниже используется следующая терминология: Подмножество Unicode-символов, абстрактные порядковые номера которых меньше $128 = 256/2$, будем называть *однобайтовыми ASCII-символами*; все остальные Unicode-символы будем называть просто Unicode-символами (или кратко — *и-символами*). Т.е. имя подмножества будет синонимом всего множества. Если не оговорено противное, тогда термин «Unicode-символы» будет обозначать именно это указанное подмножество. Первый Unicode-символ имеет порядковый номер $128 = 0x7F+1 = 0x80$. О последнем см. ниже. В кодировке UTF-8 однобайтовые ASCII-символы представляют сами себя — никак не кодируются, или кодируются тождественно.

«Выходим на простор степного океана.
Воз тонет в зелени, как челн в равнине вод,
Меж заводей цветов, в волнах травы плывет,
Минуя острова багряного бурьяна. »

Иван А. Бунин
по мотивам
А. Мицкевича

Примеры unicode-символов: $\forall, \exists, |, \backslash, \neg, \emptyset, \approx, \neq, _ \stackrel{?}{=} _, \neq, \equiv, \neq, \leq, \geq, \perp, \top, U, \cup, \circ, \in, \notin, \mathbb{N},$
 $\mathbb{Z}, \mathbb{Q}, \times, \rightarrow, \substack{1, 2, \circ}, \oplus, *, \mathbb{B}, A, \mathbf{A}, Z, \mathbf{Z}, \Sigma, \boldsymbol{\Sigma}, \mathbf{0}, \mathbf{9}, \mathbf{9},$ 气, 彗, 塵, 孿, 然, 矸, 劇, 閱, 飢, 馱, 万, 偃, П, ž,
h, , x, , , 0, 0, 9, 9, 0

СОДЕРЖАНИЕ

1. Расширение множества данных.....	3
2. Расширение синтаксиса данных.....	3
2.1. Общий принцип расширения синтаксиса и семантики языка Рефал-5.....	3
2.2. Понятие предслова.....	3
2.3. Расширение синтаксиса экранирования символов.....	4
2.4. Расширение, не относящееся к множеству данных, синтаксиса структуры программ и диалога трассировщика.....	5
2.4.1. Расширение понятия идентификатора.....	5
2.4.2. Расширение синтаксиса переменных.....	6
2.4.3. Синтаксис предслова.....	7
3. Встроенные и библиотечные функции.....	8
3.1. Несколько примеров неожиданностей.....	8
3.2. Встроенные функции преобразования Рефал выражений.....	9
U+2F9F4.....	14
3.3. Встроенные функции и процедуры ввода/вывода.....	15
3.4. Библиотечная функция преобразования Рефал выражений GType.....	22
4. Использование в программе символов, которые не поддерживает клавиатура.....	23



1. Расширение множества данных

Множество данных языка Рефал-5 расширено множеством u-символов.

$\text{Data} ::= \text{Symbol Data} \mid (\text{Data}) \text{ Data} \mid \text{EMPTY}$

$\text{Symbol} ::= \text{Macro-Digit} \mid \text{'Primary-Symbol-Sequence'} \mid \text{Compound-Symbol}$

$\text{Macro-Digit} ::=$ -- неотрицательное целое число в десятичной системе
счисления, не превосходящее $2^{64}-1$

$\text{Primary-Symbol-Sequence} ::= \text{Primary-Symbol Primary-Symbol-Sequence} \mid \text{EMPTY}$

$\text{Primary-Symbol} ::=$ однобайтовый ASCII-символ

| u-символ

| Escape-Sequence

$\text{Escape-Sequence} ::= \backslash \text{hhh}$ -- семантика: однобайтовый символ

| $\backslash \text{uh} \mid \backslash \text{uhh} \mid \backslash \text{uhhh}$ -- семантика: u-символ

| $\backslash \text{uhhhh} \mid \backslash \text{uhhhhh} \mid \backslash \text{u10hhhh}$ -- семантика: u-символ

$\text{Compound-Symbol} ::= \text{"Primary-Symbol-Sequence"}$

$\text{EMPTY} ::=$ -- пустая последовательность

2. Расширение синтаксиса данных

2.1. Общий принцип расширения синтаксиса и семантики языка Рефал-5

Все ранее написанные Рефал-5 программы, которые работают в актуальной (не тестовой) версии системы программирования Рефал-5, должны корректно работать на *нерасширенном* множестве данных в версии, расширенной unicode-символами.

2.2. Понятие предслова

«В русском языке некоторые предлоги могут выступать в роли послелогов, как например, во фразе *не корысти ради, а удовольствия для.*»
Цитата из Википедии

Как известно, термин слово (синоним — «составной символ») в Рефале означает терм, который синтаксически записывается обрамлением двойных кавычек, внутри которых может стоять любая последовательность *первичных* символов, включая пустую последовательность. Слово отождествляется с s-переменной, даже если это пустое слово.

Идентификатор по определению является словом, обрамляющие кавычки над которым можно не писать, а можно и писать, когда конкретное вхождение этого идентификатора является частью данных. Когда этот идентификатор является частью синтаксической структуры программы (именем функции или именем переменной), тогда двойные кавычки его обрамления не допускаются.

Определение. Слово назовем *предсловом*, если оно не является идентификатором, но обрамляющие его кавычки разрешается не писать.

Точные понятия расширенного множества идентификаторов и предслова даны ниже в Разделах 2.4.1., 2.2., 2.4.3..

2.3. Расширение синтаксиса экранирования символов

Реализована возможность указывать Unicode-символ его абстрактным Unicode-номером, т. е. порядковым номером, в данных посредством последовательностей вида '\uhhhhhh', где h — означают шестнадцатеричные цифры. Незначащие нули могут быть опущены, т. е. после \u может стоять от одной до шести шестнадцатеричных цифр. Если за \u нет ни одной шестнадцатеричной цифры — это синтаксическая ошибка; если число hhhhhh больше чем 10FFFF, то это тоже синтаксическая ошибка. Номер 0x10FFFF является последним порядковым номером в множестве u-символов. Семантика символов, следующих за шестой шестнадцатеричной цифрой, определяется равенством:

'\uhhhhhh... characters ...' = '\uhhhhhh' '... characters ...'

Две следующие последовательности символов представляют *разные* Рефал данные:

- '\xC3\x90' и '\uC390'.
- Выполняется семантическое равенство \xhh = \uhh.

Например, '\x0F' = '\u0F' = '\uF'. Согласно определению (решению) В.Ф. Турчина, последовательность '\xF' представляет пример синтаксической ошибки.

См. книгу В.Ф. Турчина «REFAL-5. programming guide & reference manual»

<http://refal.botik.ru/book/html/> .

2.4. *Расширение, не относящееся к множеству данных, синтаксиса структуры программ и диалога трассировщика*

2.4.1. Расширение понятия идентификатора

Выше, в Разделе 2.2., определено содержательное понятие идентификатора. Именем функции может быть только идентификатор. Технические синтаксические детали имени идентификатора определяет грамматика:

```
s.Id ::= `s.first-symbol e.id_name` -- здесь кавычки условные, а не рефальские; в
      рефале это один s.var-символ
s.first- symbol ::= -- символ, во всех поддерживаемых unіcod-ом шрифтах, из одной из
                   следующих (расширенных) азбук: греческой, латиницы, азбуке
                   математических букв (см. https://unicode-table.com/en/blocks/mathematical-
                   alphanumeric-symbols/ );
e.id_name ::= e.id-prefix e.id-postfix
e.id-prefix ::= s.id-prefix-char e.id-prefix | []
s.id-prefix-char ::= `-' | `_' | ascii-десятичная-цифра
                  | -- символ, во всех поддерживаемых unіcode-ом шрифтах, из одной из
                    следующих (расширенных) азбук: греческой, латиницы, азбуке
                    математических букв;
e.id-postfix ::= s.id-postfix-index e.id-postfix | []
s.id-postfix-index ::= -- верхний или нижний* индекс из тех же самых символов, которые
                      допускаются в s.id-prefix-char, кроме символа подчеркивания;
                      *для нижнего индекса, являющегося последним символом имени конкретного
                      вхождения S составного символа, исключением являются три значения этого
                      индекса: s, t, e; если имя вхождения S в контексте конкретных синтаксических
                      структур не может быть однозначно понято как имя некоторой функции, тогда
```

данное вхождение составного символа, по определению, не является идентификатором; примеры таких вхождений см. в Разделе 2.4.2.;

`[] ::= -- пустая последовательность`

Если непосредственно за непустой последовательностью индексов в окончании идентификатора стоит символ, не являющийся *и*-индексом, тогда этот символ отделяется от идентификатора пробелом. Далее синтаксис определяется по индукции.

2.4.2. Расширение синтаксиса переменных

Переменная может либо начинаться именем типа, либо заканчиваться именем типа. В первом случае непосредственно за именем типа следует знак точки, разделяющий имя типа от имени переменной. Во втором случае разделительная точка не допускается. Тип конкретного вхождения данной переменной не может входить дважды в это вхождение.

Уточним данный набросок синтаксиса переменной:

`Variable ::= Type`.`Variable-IdName | Type`.`Variable-DgsName | Variable-Id`

`Variable-Id* ::= Variable-IdName Index-Type`

*если вхождение *S* определяемой здесь синтаксической единицы в контексте других конкретных синтаксических структур может быть однозначно понято как имя некоторой функции *F*, тогда данное вхождение, по определению, есть идентификатор — имя функции *F*; примеры таких вхождений даны ниже определения данной грамматики;

`Type ::= s | t | e`

`Index-Type ::= s | t | e`

`Variable-IdName ::= -- имя идентификатора, с которого снято ограничение, введенное в Разделе 2.4.1.;`

`Variable-DgsName ::= Digit Var-DgsName`

`Var-DgsName ::= Digit Var-DgsName | []`

`Digit ::= ascii-десятичная-цифра`

`[] ::= -- пустая последовательность`

Пример: Здесь выделенные **цветом** вхождения индексированных синтаксических единиц суть идентификаторы, все остальные вхождения индексированных синтаксических единиц суть переменные.

\$EXTERN **Name_s** , Example;

```

.....
Fe{
  Names = <Example Names> <Mu Names 'argument'>;
  Fe = <Names Fe>;
}

```

2.4.3. Синтаксис предслова

pretext — отговорка, повод, предлог;
preword — предслово.

Выше, в Разделе 2.2., дано содержательное определение понятия *предслова*. Синтаксис предслова определяется грамматикой:

```

s.PreWord ::= `s.first-PreWord-symbol e.PreWord_name` -- здесь кавычки
                                     условные, а не рефальские; в рефале это один s.var-символ
s.first- PreWord-symbol ::= -- символ, во всех поддерживаемых unicod-ом шрифтах, из одной
                                     из следующих (расширенных) азбук: Кириллицы, греческой,
                                     латиницы, азбуке математических букв;
e.PreWord_name ::= e.PreWord-prefix e.PreWord-postfix
e.PreWord-prefix ::= s.PreWord-prefix-char e.PreWord-prefix | []
s.PreWord-prefix-char ::= ` ` | ` _ ` | ascii-десятичная-цифра
                                     | -- символ, во всех поддерживаемых unicod-ом шрифтах, из одной
                                     из следующих (расширенных) азбук: Кириллицы, греческой,
                                     латиницы, азбуке математических букв;
e.PreWord -postfix ::= s.id-postfix-index e.id-postfix | []
s.PreWord-postfix-index ::= -- верхний или нижний индекс из тех же самых символов, которые
                                     допускаются в s.PreWord-prefix-char, кроме символа
                                     подчеркивания;
[] ::= -- пустая последовательность

```

Таким образом, синтаксически предслово определяется аналогично идентификатору, в котором нет исключений, наложенных на нижние индексы (см. выше), если в список основных азбук добавить (расширенную) Кириллицу. Например, слово *Привет* является предсловом, но не является ни идентификатором ни именем переменной, даже если к нему приписать нижний s-индекс: *Привет_s*.

3. Встроенные и библиотечные функции

Встроенной функцией называется функция-подпрограмма, исходный код которой написан на языке программирования Си и её можно вызвать из любой Рефал программы. Встроенные функции изначально доступны пользователю сразу после загрузки интерпретатора *refgo* или трассировщика *reftr* (в этом случае они доступны также из диалога трассировщика).

Библиотечной функцией называется функция, определенная в терминах языка Рефал, исходный код которой находится в стандартном модуле *reflib.ref*.

В данном разделе описываются как расширения областей определений некоторых встроенных и библиотечных функций в системе программирования Рефал-5 Version ПЗ, так и расширение множеств встроенных и библиотечных функций. Имя новой встроенной функции: *UType*. См. также книгу В.Ф. Турчина «REFAL-5. programming guide & reference manual» <http://refal.botik.ru/book/html/> .

3.1. Несколько примеров неожиданностей

Пример №1: $\langle \text{Lower } \Theta \rangle = \theta$, $\langle \text{Upper } \theta \rangle = \Theta$

Пример №2: $\langle \text{Lower } \Theta \rangle = \vartheta$, $\langle \text{Upper } \vartheta \rangle = \Theta$

Замечание к Примерам №1, №2: Большинство текстовых редакторов, в которых редактируются программы, а не пишутся статьи, как и консоль, отображают символы Θ , θ таким образом, что зрительно их невозможно отличить один от другого, также как ASCII

символы С и с (здесь первый символ из Кириллицы, второй — из латиницы). Более того, в этих редакторах и в консоли символ Θ зрительно воспринимается строчным, а символ θ наблюдается прописным.

Пример №3: Ни один из вышеперечисленных символов не совпадает с символами Θ, ө, которые, в отличие от предыдущих — греческих, принадлежат латинской азбуке.

3.2. Встроенные функции преобразования Рефал выражений

- **<Lower e.Expr> ==> e.Expr'**
 - Область определения расширена на определенное выше, расширенное множество данных.
 - Возвращает выражение e.Expr', в котором все символы («буквы») входного выражения переведены в соответствующие им строчные символы. Свойством «быть прописным/строчным» могут обладать не только буквы, но и другие u-символы:
 - например, латинские числа (каждое из которых — один u-символ, а не последовательность латинских букв) существуют в верхнем и нижнем регистрах:
 - <https://unicode-table.com/en/blocks/number-forms/> ,
<https://unicode-table.com/en/sets/roman-numerals/>
 - По определению считаем, что символ u (объ)является строчным (находится в нижнем регистре) тогда и только тогда, когда <Lower u> = u, иначе этот символ (объ)является прописным.
- **<Upper e.Expr> ==> e.Expr'**
 - Область определения расширена на определенное выше, расширенное множество данных.
 - Аналогично функции Lower каждый строчный u-символ входного выражения переводится в верхний регистр, если существует соответствующий ему прописной u-символ.
 - Функция Upper не является «обратной» функцией к функции Lower.

- В общем случае, обычное тождество

$$\langle \text{Lower} \langle \text{Upper } u \rangle \rangle = \langle \text{Lower } u \rangle$$

НЕ выполняется.

- *Пример:* по «спецификации» unicode следующих трёх и-символов S, s, ſ два последних из них суть строчные и оба имеют один и тот же, соответствующий им, прописной символ S. В то же время $\langle \text{Lower } S \rangle = s$. В результате имеем: $\langle \text{Lower} \langle \text{Upper } \text{ſ} \rangle \rangle = s$, $\langle \text{Lower } \text{ſ} \rangle = \text{ſ}$. (см. <https://unicode-table.com/en/>)
- *Пример:* $\langle \text{Lower } \text{İ} \rangle = i$, $\langle \text{Lower } I \rangle = i$, $\langle \text{Upper } i \rangle = I$, $\langle \text{Upper } İ \rangle = I$
 $\langle \text{Lower} \langle \text{Upper } i \rangle \rangle = i$, но $\langle \text{Lower } i \rangle = \text{ı}$
- Тождество $\langle \text{Upper} \langle \text{Lower } u \rangle \rangle = \langle \text{Upper } u \rangle$ также НЕ выполняется в общем случае.
 - *Пример:* по «спецификации» unicode следующих трёх и-символов DŽ Dž dž (это три, а не 6 и-символов, разделенные пробелами) два первых из них находятся в верхнем регистре и оба имеют один и тот же, соответствующий им, строчный символ dž. В то же время $\langle \text{Upper } \text{dž} \rangle = \langle \text{Upper } \text{Dž} \rangle = \text{DŽ}$ (см. <https://unicode-table.com/en/>)
 - *Пример:* $\langle \text{Lower } \text{İ} \rangle = i$, $\langle \text{Lower } I \rangle = i$, $\langle \text{Upper } i \rangle = I$, $\langle \text{Upper } İ \rangle = I$
 $\langle \text{Upper} \langle \text{Lower } \text{İ} \rangle \rangle = I$, но $\langle \text{Upper } \text{İ} \rangle = \text{İ}$
 - *Другой пример:* $\langle \text{Lower } \Sigma \rangle = \sigma$, $\langle \text{Upper } \varsigma \rangle = \langle \text{Upper } \sigma \rangle = \Sigma$. В этом примере нет ни одного символа, выделенного в Word «жирным» шрифтом. Все эти и-символы будут отображаться «жирными» в любом текстовом редакторе, умеющем отображать utf-8. В Рефале-5 Version ПЗ эти символы НЕ будут отождествляться с соответствующими им «нежирными» и-символами.
- Заметим также, что выписанные выше два привычных тождества, однако, *выполняются для «почти всех» и-символов.*
- На данный момент не обнаружено ни одного и-символа, для которого не выполняются следующие тождества:

- $\langle \text{Upper } \langle \text{Lower } \langle \text{Upper } s.u \rangle \rangle \rangle = \langle \text{Upper } \langle \text{Lower } s.u \rangle \rangle$
- $\langle \text{Lower } \langle \text{Upper } \langle \text{Lower } s.u \rangle \rangle \rangle = \langle \text{Lower } \langle \text{Upper } s.u \rangle \rangle$

• $\langle \text{Type } e.\text{Expr} \rangle \implies s.\text{first-term-type } s.\text{subtype } e.\text{Expr}$

- Область определения расширена на определенное выше, расширенное множество данных. Здесь, кроме возможных выходов, описанных в книге В.Ф. Турчина, добавлены следующие:

- $e.\text{Expr} ::= t.\text{first-term } e.\text{Expr} \mid []$ -- пустое выражение

$s.\text{first-term-type} ::= 'U'$

$s.\text{subtype} ::=$ -- однобуквенный код «азбуки», к которой принадлежит и-символ:

- 'c' — (расширенная) Кириллица (включая церковно-славянскую азбуку), 'g' — (расширенная) греческая (включая устаревшие варианты греческой азбуки), 'l' — (расширенная) латиница, 's' — алфавит символов, 'M' — азбука математических букв, 'r' — арабская «вязь», 'h' — иврит, 'a' — армянская, 'm' — грузинская, 'j' — объединение японских азбук (их несколько), 'k' — корейская, 'z' — китайская слоговая (современные и традиционные иероглифы), 'e' — эфиопская, 'i' — полуслоговая индийская (объединённая Хинди/Санскрит), 'o' — объединение некоторых исторических азбук (включая глаголицу), 'u' — объединение всех иных азбук;

- семантика значения подтипа 'u' в будущем может быть сужена выделением некоторых азбук в отдельные новые значения подтипа.

Трудно сопоставить без пересечения адекватные символы, указывающие азбуку. Например, какие были бы предложения для грузинской азбуки? Или какие символы выбрать для армянского алфавита и арабской вязи?

$s.\text{first-term-type} = 'W'$ – в этом случае расширена семантика подтипа:

$s.\text{subtype} ::= 'i'$ – этому подтипу принадлежит любое возможное имя

функции (см. определение выше);

- | 'q' – этому подтипу принадлежит любой составной символ из дополнения к множеству элементов подтипа 'i'.

- $\langle \text{UType } e.\text{Expr} \rangle \implies \text{s.first-term-type } (e.\text{Alpha-properties}) e.\text{Expr}$

- Причина введения новой функции UType, вместо более широкого доопределения функции Type, состоит в необходимости совместимости всех, ранее написанных программ на Рефале-5, с новой версией Version ПЗ. Ранее зафиксированный В.Ф. Турчиным выходной формат функции Type является слишком жестким/узким и не позволяет описывать свойства и-символов, которыми не обладают ascii-символы.

$e.\text{Expr} ::= t.\text{first-term } e.\text{Expr}' \mid [] \text{ -- пустое выражение}$

$\text{s.first-term-type} ::= \text{UChar} \mid \text{ASCII} \mid \text{Empty} \mid \text{Other}$

$e.\text{Alpha-properties} ::= \text{s.Alphabet-compound-symbol } e.\text{properties} \mid \text{Unknown } e.\text{properties} \mid []$

$e.\text{properties} ::= \text{s.Register-case } e.\text{Case } e.\text{still-undefined-properties}$

$\text{s.Register-case} ::= \text{Lower} \mid \text{Upper}$

$e.\text{Case} ::= e.\text{Numeric} \mid e.\text{Index} \mid e.\text{WhiteSpace}$

$e.\text{Numeric} ::= \text{Numeric } s.\text{NType}$

$s.\text{NType} ::= s.\text{ArabicDigit} \mid s.\text{OtherNumeric}$

$s.\text{ArabicDigit} ::= \text{Digit} \text{ -- десятичная (арабская) цифра, включая ее варианты в разных шрифтах}$

$s.\text{OtherNumeric} ::= \text{Fraction}$

| -- цифры (и некоторые числа) в других числовых системах - римской, еврейской, китайской, славянской, и т. д.; здесь под римскими «цифрами» понимаются числа в римской системе счисления, которые представлены одним и-символом; например, II — это число два отождествляется с s-переменной

$e.\text{Index} ::= \text{Index } s.\text{IndexCase } s.\text{IndexAlphabet}$

$s.\text{IndexCase} ::= \text{Top} \mid \text{Bottom}$

s.IndexAlphabet ::= Digit | Latin | Greek | Cyrillic | Parenthesis | Operation | Relation | Other

e.WhiteSpace ::= WhiteSpace e.WhiteSpaceCase

e.WhiteSpaceCase ::= e.Non-breaking | EN s.ENCase | EM s.EMCase | e.Punctuation | e.Thin
| e.Hair | e.Zero_width | e.Separator | e.Format_char | e.Medium_math
| e.Invisible_oper | e.Deprecated | e.Ideographic

s.ENCase ::= QUAD | Space

s.EMCase ::= QUAD | Space | $\frac{1}{3}$ | $\frac{1}{4}$ | $\frac{1}{6}$ -- ширина (доля) пробела от ширины M

e.Non-breaking ::= Non-breaking e.Other

e.Punctuation ::= Punctuation e.Other -- ширина пробела равна ширине точки

e.Thin ::= Thin e.Other

e.Hair ::= Hair e.Other

e.Separator ::= Separator e.Other

e.Format_char ::= Format_char e.Other

e.Medium_math ::= Medium_math e.Other

e.Invisible_oper ::= Invisible_operator e.Other

e.Deprecated ::= Deprecated e.Other

e.Ideographic ::= Ideographic e.Other -- ширина пробела равна ширине иероглифа

e.Other ::= Other | e.still-undefined-properties

Примеры unicode-индексирования:⁵ Φ_A^n $B^{n^{\wedge}}$ T^{\wedge} $N^{\wedge^5}_{\beta^5}$ I^5_6 I_{β^z} U^z

<https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B1%D0%B5%D0%BB>

<https://unicode-table.com/en/blocks/general-punctuation/>

Библиотечная функция GType (см. Раздел 3.4.) «объединяет» встроенные функции Type и UType, т. е. расширяет выходной формат этих функций и приводит его к виду выходного

формата функции UType.

- $\langle \text{Ord } e.\text{InpExpr} \rangle \implies e.\text{OutExpr}$

- Функция заменяет все ascii-символы и u-символы во входном выражении на соответствующие им макроцифры — порядковые номера абстрактного Unicode.

Номер реально существующего на данный момент последнего визуально наблюдаемого в таблицах u-символа 𐄂: 0x2F9F4 = 195060 .

U+10FFFF = 285212671 -- теоретически возможный максимальный порядковый номер u-символа

U+2FA1F = 195103 -- порядковый номер реально существующего последнего u-символа □ - на данный момент этот символ в таблицах неопределен

U+2F9F4 = 195060 -- на данный момент порядковый номер существующего последнего визуально наблюдаемого в таблицах u-символа 𐄂

- $\langle \text{Chr } e.\text{InpExpr} \rangle \implies e.\text{OutExpr}$

- Функция «обратная» к функции Ord на множестве своих значений. Заменяет все макроцифры во входном выражении, взятые по модулю 285212672, на соответствующие им u-символы. Т.е. эти входные макроцифры по указанному модулю суть порядковые номера абстрактного Unicode получаемых u-символов.
- Выполняются следующие тождества:

$$\langle \text{Chr } \langle \text{Ord } e.\text{InpExp} \rangle \rangle = \langle \text{Chr } e.\text{InpExp} \rangle$$

$$\langle \text{Ord } \langle \text{Chr } \langle \text{Ord } e.\text{InpExp} \rangle \rangle \rangle = \langle \text{Ord } e.\text{InpExp} \rangle$$

- **Пример:**

$\langle \text{Prout } \langle \text{Chr } 'a' (\langle \text{Ord } 'a' \rangle) 'п' (\langle \text{Ord } 'п' \rangle) 'Ð' (\langle \text{Ord } 'Ð' \rangle) '𐄂' (\langle \text{Ord } '𐄂' \rangle)$

$'𐄂' (\langle \text{Ord } '𐄂' \rangle) 'o' (\langle \text{Ord } 'o' \rangle) '𐄂' (\langle \text{Ord } '𐄂' \rangle) \rangle \rangle \implies$

$a(a) п(п) Ð(Ð) 𐄂(𐄂) 𐄂(𐄂) o(o) 𐄂(𐄂)$

3.3. Встроенные функции и процедуры ввода/вывода

К простейшим функциям ввода/вывода можно отнести и встроенные функции `Explode_Ext`, `Implode_Ext` (см. Приложение к книге В.Ф. Турчина, <http://refal.botik.ru/>), если «поток» или «файлом» считать имя (текст) составного символа или unicode-последовательность байт, представляющих и-символ.

- `<Explode_Ext s.symbol e.option> ==> e.string`

- Функция читает из входного потока `s.symbol`.

`s.symbol ::= s.word | s.UTF-symbol`

- `s.word ::= s.compound_symbol-string_name`

- Область определения этой функции расширена относительно как аргумента `s.word` (Рефал слов стало больше — они могут содержать и-символы), так и аргумента `e.option`. Первый аргумент в предыдущей версии Рефала-5 допускал только `s.word`, второй аргумент — пустое значение. Логически текст первого аргумента этой функции можно рассматривать, как содержимое входного потока; `e.option` задает режим чтения из этого потока. По-умолчанию, режим чтения (преобразования) совпадает с используемым режимом в предыдущей версии Рефала-5.

`e.option ::= Flat | Nonflat | []`

`[]` ::= -- пустое выражение, опция «по-умолчанию», есть `Nonflat`

-- Если выбрана опция `Nonflat`, тогда выходная строка `e.string` может включать как байтовые ASCII символы, так и любые и-символы. Функция выдаёт свой первый аргумент, если он является и-символом.

-- Если выбрана опция `Flat`, тогда `e.string` состоит только либо из байтовых ASCII символов, либо из и-символов, посредством которых закодированы однобайтовые ASCII символы с порядковыми номерами больше 127.

Визуально эти и-символы совпадают с обычными однобайтовыми символами с соответствующими порядковыми номерами, если их

отображение на выходе не зависит от одно-байтовой кодировки. Например, KOI8-RU или Win-1251.

- Пример результатов двух вызовов функции вида:

- $\langle \text{Explode_Ext } s.\text{UTF-symbol } e.\text{option} \rangle$

- $\langle \text{Explode_Ext 'П'} \rangle = \text{'П'}$, где $\langle \text{Lenw } \langle \text{Explode_Ext 'П'} \rangle \rangle = 1$

- $\langle \text{Explode_Ext 'П' Flat} \rangle = \text{'Д'}$, где $\langle \text{Lenw } \langle \text{Explode_Ext 'П' Flat} \rangle \rangle = 2$

- Здесь второй символ результата есть управляющий символ, визуально — «пробел нулевой ширины».

- $\langle \text{Implode_Ext } e.\text{string } e.\text{Expr} \rangle \Rightarrow s.\text{compound_symbol-string_name}$

- Функция производит запись в выходной поток $s.\text{compound_symbol-string_name}$.

- Здесь $e.\text{string}$ есть приставка входного аргумента, состоящая только из ASCII и u-символов и имеющая наибольшую длину из всех таких приставок, включая приставку нулевой длины. Для всех *таких* строк выполняется тождество:

$$\langle \text{Explode_Ext } \langle \text{Implode_Ext } e.\text{string} \rangle \rangle = e.\text{string}$$

- $\langle \text{Open } s.\text{Mode } s.D \text{ } e.\text{File-name} \rangle \Rightarrow e.\text{OutExpr}$

- Открывает имя файла и сопоставляет ему макроцифру $s.D$ — уникальный дескриптор.
- Ниже будем предполагать, что существуют две текстовых кодировки: utf-8, которая является на данный момент актуальной кодировкой; вторая кодировка уточнена ниже.

$e.\text{File-name} := (e.\text{string}) \mid s.\text{word}$

$s.D :=$ -- натуральное число по модулю FILE_LIMIT — дескриптор потока/файла, где FILE_LIMIT — константа конкретной версии интерпретатора Рефала-5

-- Правила преобразования байт из потока в Рефал выражение или наоборот зависят от s.Mode — режима, в котором открыт этот поток. Подробное описание этих правил дано ниже при «спецификации» конкретных процедур чтения и записи.

```
s.Mode := "e.m,ccs=e.Encode" | 's.Read' | 's.Write' | 's.Add'

e.m := s.m e.type | e.other -- другие режимы, поддерживаемые языком Си

s.m := s.Read | s.Write | s.Add

s.Read := R | r -- открыть на чтение

s.Write := W | w -- открыть на запись

s.Add := A | a -- открыть на дополнение

e.type := t | b | []

t := -- построчн-ое(ая) текстов-ое(ая) чтение/запись/дополнение

b := -- побайтн-ое(ая) бинарн-ое(ая) чтение/запись/дополнение

[] := -- опция по-умолчанию: построчн-ое(ая) текстов-ое(ая)
      чтение/запись/дополнение

e.Encode := utf-8 | Nonflat | Flat | [] | e.Other

utf-8 := -- если файл открыт, следовательно, зафиксирована кодировка, с которой
          будут работать процедуры обращения к этому файлу. Кодировкой «по-
          умолчанию» является кодировка utf-8.

Nonflat := -- синоним utf-8

Flat := -- побайтное чтение/запись; уточнение см. ниже в описаниях функций
         ввода/вывода.

[] := -- пустое выражение -- опция по-умолчанию: utf-8
```

- `<Close s.D> ==> []`
 - Удаляет файл, которому приписан дескриптор `s.D`, из списка открытых файлов.
- `<Card e.option> ==> e.OutExpr`
 - Чтение строки символов из стандартного входного потока `stdin`. Конец строки есть один из символов `'\n'`, `'\r'` или их сочетание, и зависит от операционной системы. На выходе строка `u`-символов, видимая пользователю, как строка абстрактных `u`-символов. Следующее решение позволяет не вводить понятие ошибки чтения из потока.
 - Любая корректная последовательность байт `utf-8` кода, длина которой больше 1, преобразуется (читается слева направо) в Рефал Unicode символ, представленный в реализации `utf-8` кодом — натуральным числом и видимый пользователем как абстрактный Unicode символ с абстрактным порядковым номером этого символа (а не натуральным числом — `utf-8` кодом). ASCII символы, однобайтовые Unicode-символы, с визуальной точки зрения пользователя, преобразуются, грубо говоря, «сами в себя»: в термы — символы, визуально представляющие эти ASCII символы (каждый такой символ — в один терм).
 - Если очередной байт потока не входит (слева направо) в корректную последовательность байт `utf-8` кода, и не является признаком начала или конца всего потока, тогда этот байт преобразуется (кодируется) в `u`-символ, который визуальное отображается на консоли точно также, как и считываемый байт, если понимать его как классический ASCII символ (из всего множества от 0 до 255).

`e.option := utf-8 | Nonflat | Flat | []`

`utf-8 := --` Любая корректная последовательность байт `utf-8` кода преобразуется (читается слева направо из потока `stdin`) в Рефал Unicode символ (см. выше).

`--` Заметим, что Linux-консоль отображает эту последовательность также единственным Unicode символом. Следовательно, с точки зрения пользователя, визуальное при чтении происходит «тождественное преобразование». Но «в

действительности всё не так, как на самом деле».

-- Правило преобразования байта потока stdin, который не входит (слева направо) в корректную последовательность байт utf-8 кода описано выше.

Flat := -- Любой байт потока stdin преобразуется в u-символ, который визуально отображается на консоли точно так же, как и считываемый байт, если понимать его как ASCII символ (из всего множества от 0 до 255).

-- Другими словами, в некотором смысле, вызов <Card Flat> при чтении игнорирует понятие utf-8 последовательности в потоке stdin.

Nonflat := -- синоним utf-8

[] := -- Опцией «по-умолчанию» является кодировка, в которой открыт поток stdin. При старте интерпретатора refgo/reftr поток stdin открывается на чтение в кодировке utf-8.

Пример: пусть входной поток визуально является последовательностью из пяти символов, два из которых суть пробелы: a ⌘ b,

- если поток stdin открыт в режиме «utf-8», тогда результатом вызова <Card> является последовательность 'a ⌘ b', которая состоит из пяти термов — u-символов;
 - если поток stdin открыт на побайтное чтение «Flat», тогда результатом вызова <Card> является последовательность 'a ê® b' из семи символов, один из которых (средний) есть «пробел нулевой ширины»; три средних символа *визуально* составляют utf-8 последовательность, которая представляет u-символ ⌘.
- Вызов <Card Flat> позволяет открыть stdin на побайтное чтение только для данного вызова. После чего восстанавливается кодировка потока stdin, которая была до этого вызова. Аналогично для любой другой кодировки.

- `<Get s.D e.Encode> ==> e.OutExpr`

`s.D` := -- натуральное число по модулю `FILE_LIMIT` — дескриптор входного потока, где `FILE_LIMIT` — константа конкретной версии интерпретатора Рефала-5

`e.Encode` := `utf-8` | `Nonflat` | `Flat` | `[]`

`utf-8` := -- Любая корректная последовательность байт `utf-8` кода преобразуется (читается слева направо из потока `s.D`) в Рефал Unicode символ.

-- Правило преобразования байта потока `stdin`, который не входит (слева направо) в корректную последовательность байт `utf-8` кода, описано выше.

`Flat` := -- Любой байт входного потока `s.D` преобразуется в `u`-символ, который визуально отображается текстовым редактором, поддерживающим кодировку UTF-8, точно так же, как и считываемый байт, если понимать его как ASCII символ (из всего множества от 0 до 255).

`Nonflat` := -- синоним `utf-8`

`[]` := -- Опцией «по-умолчанию» является кодировка, в которой открыт поток `s.D`.

- Вызов функции `<Get s.D s.Encode>` с явным указанием кодировки `s.Encode`, позволяет открыть входной поток `s.D` на чтение в кодировке `s.Encode` только для данного вызова. После чего восстанавливается кодировка потока `s.D`, которая была до этого вызова.

- `<Putout s.D e.Expr> ==> []`

- Вывод выражения `e.Expr` в выходной поток `s.D`. Вывод `u`-символов производится в виде последовательностей байт, которые определяются кодировкой, в которой открыт данный выходной поток `s.D`.

- Если поток открыт на запись в кодировке `utf-8`, тогда любой `u`-символ

выводится соответствующей ему utf-8 последовательностью байт, которыми он закодирован в кодировке utf-8.

- Правила вывода u-символов, длина utf-последовательности которых больше 2, при открытии выходного потока на запись в кодировках utf-8 и Flat, совпадают.
- Если поток открыт на запись в кодировке Flat и u-символ является utf-8 кодировкой байта, значение которого > 127 , тогда этот u-символ декодируется и указанный байт — результат декодирования выводится в поток.

○ Выполняется нижеследующее отношение:

- Пусть даны два файла (потока) inp.txt и out.txt. Пусть они открыты следующим образом: `<Open s.mode1 s.Dinp 'inp.txt'>` `<Open s.mode2 s.D 'out.txt'>`, где `s.mode1 = "r,ccs=e.encode1"` и `s.mode2 = "w,ccs=utf-8"` и определяют кодировки (utf-8 или Flat), тогда для каждой unicode строки потока s.Dinp равенство

$$\langle \text{Get } s.\text{Dinp} \rangle = \langle \text{Get } s.D \langle \text{Putout } s.D \langle \text{Get } s.\text{Dinp} \rangle \rangle$$
$$\langle \text{Open "r,ccs=e.encode3" s.D 'out.txt'} \rangle$$

- выполнено тогда и только тогда, когда `e.encode3 = utf-8` или `e.encode1 = e.encode3 = Flat`.
- Если `s.mode2 = "w,ccs=Flat"`, тогда для каждой unicode строки потока s.Dinp равенство

$$\langle \text{Get } s.\text{Dinp} \rangle = \langle \text{Get } s.D \langle \text{Putout } s.D \langle \text{Get } s.\text{Dinp} \rangle \rangle$$
$$\langle \text{Open "r,ccs=e.encode3" s.D 'out.txt'} \rangle$$

- выполнено тогда и только тогда, когда `e.encode1 = e.encode3 = utf-8`.

- **Замечание о бинарной кодировке**

- Бинарный вывод/ввод нужно воспринимать, как еще один вид кодировки. Правило приоритета кодировок: если файл открыт как `<Open "wb" имя-файла>`,

тогда бинарная кодировка на вывод имеет старший приоритет, т. е. вывод будет происходить бинарный, не взирая ни на какие уловки пользователя.

3.4. Библиотечная функция преобразования Рефал выражений *GType*

- $\langle GType\ e.Expr \rangle \implies (e.Alpha-properties)\ e.Expr$

- Эта функция определена на Рефале, в модуле `reflib.ref`. Она объединяет встроенные функции `Type` и `UType`, т. е. расширяет выходной формат этих функций и приводит его к виду выходного формата функции `UType`. Ниже показано это расширение выходного формата, опущенные определения совпадают с определениями, данными выше для функции `UType`.

`e.Expr ::= t.first-term e.Expr' | []` -- вычисляем свойства первого терма

`e.Alpha-properties ::= s.Alphabet-compound-symbol e.properties | ...`

`e.Case ::= ... | Parenthesis | e.Printable | e.Word | e.NonPrintable`

`e.Printable ::= Printable ASCII e.Other`

`e.Word ::= Word s.Identifier | Word e.Other`

`s.Identifier ::= -- имя функции; см. выше определение имени функции`

`e.NonPrintable ::= s.NonPrintableASCII e.Other`

`s.NonPrintable BEL | BS | ESC | Control`

`e.WhiteSpaceCase ::= ... | s.SpaceAscii ASCII e.Other`

`s.SpaceAscii ::= Space | TAB | LF | Zero_width`

| CR -- (#13, на ubuntu ставит курсор в первую позицию текущей строки; далее печать продолжается с первой позиции этой строки, стирая шаг за шагом содержимое необходимых позиций до перевода строки, после которого оставшееся окончание

прерванной строки остается на экране) -- возврат каретки на пишущей машинке -- механическом «принтере»!

e.Separator ::= ... | s.SeparatorAscii ASCII e.Other

s.SeparatorAscii ::= ... | SOH | STX | ETX | FF | ETB | EM | FS | GS | RS | US

SOH ::= -- начало заголовка

STX ::= -- начало текста

ETX ::= -- конец текста

FF ::= -- новая страница (#12, на ubuntu просто переводит строку и вставляет на ней n пробелов, где n — длина строки, которую прервал символ FF)

ETB ::= -- конец текстового блока

EM ::= -- конец носителя

FS ::= -- разделитель файлов

GS ::= -- разделитель групп

RS ::= -- разделитель записей

US ::= -- разделитель юнитов

Пример идентификатора — имени функции: 608B15%. Здесь все шрифты поддерживает unicode, а LibreOffice только косвенно отображает их, согласно кодировке utf-8.

Пример слова, не являющегося идентификатором: "6086B15%".

4. Использование в программе символов, которые не поддерживает клавиатура

1. Об опыте использования текстового редактора Bluefish версии 2.2.11.

- Этот редактор автоматически проявил своё существование при попытке редактирования файлов с именем вида *.c в операционной системе Ubuntu. Т.е. его не нужно было устанавливать в системе Ubuntu. Откуда можно предположить, что этот редактор не только свободно распространяется, но и входит в дистрибутив Ubuntu.
- В рабочем окне этого редактора, в левом нижнем углу, есть кнопка, на которой нарисованы четыре символа. Эта кнопка позволяет выбрать нужный алфавит, в котором можно выбрать нужный символ, который мышкой можно перенести в текст программы, не копируя его, а захватив левой кнопкой мышки. Замечу, что он иногда вставляется в неожиданные места: не пугайтесь, если не увидели этот символ в вашем тексте. Посмотрите внимательно на текст — символ где-то там.
- Скорее всего, такие возможности есть в любом текстовом редакторе, поддерживающем unicode символы.

2. Более удобным инструментом копирования u-символов является таблица

<https://old.unicode-table.com/en/blocks/cyrillic/> .

- Находите (можно визуально, можно поиском (части) названия нужного множества символов; строка для введения названия — в верхней части экрана). Далее просто копируете мышкой нужный символ и переносите его в Рефал программу.
3. В самом крайнем случае можно использовать препроцессор `gefrr`, который можно найти в дистрибутиве Рефала-5. Этот препроцессор написан на Рефале-5. `gefrr` заменяет имена макросов на значения этих макросов, если эти значения существуют. Имя макроса обрамляется скобками вида `@/имя/`. Никаких синтаксических ограничений, кроме того, пустое имя не допускается, нет. Нет никаких ограничений на синтаксический контекст использований имен макросов в тексте программы. Любое вхождение имени макроса заменяется его значением, если этот имя не является частью комментариев и значение макроса с таким именем существует.

Препроцессор `gefrr` должен использоваться перед вызовом компилятора, который ответственен за корректность синтаксиса результата подстановки.

- (Почти) Каждый unicode-символ имеет формальное уникальное имя, которое может состоять из нескольких слов. Имя макроса должно быть именем unicode-символа. Таблица названий имен этих символов доступна по ссылке:

view-source:https://www.babelstone.co.uk/Unicode/unicode.js .

- Формальные имена unicode-символов также можно найти по ссылке, уже данной выше. Но там это сделать труднее.
- Для некоторых символов достаточно знать их HTML кодировки. Они суть синонимы формальных имен.

■ **Примеры макросов:** @/LATIN CAPITAL LETTER AE/, @/OGONEK/, @/Ψ/.

- Ниже дана «рефал-программа» bak-macros_h.ref, в которой используются макросы и которая может быть подана на вход препроцессора refpp.

```
$ENTRY Go { = <Hello>; }

Hello {

= <Prout <Examples>>;

}

Examples { /* '@/&psi;/' */

/* /* '@/&psi;/' */

e.1 = "Integral-@/INTEGRAL AROUND A POINT OPERATOR/f(z)dz"\n' @/LATIN CAPITAL LETTER AE/

('@/LATIN CAPITAL LETTER O WITH GRAVE/') \n'

'(s.@/LATIN CAPITAL LETTER O WITH GRAVE/q)\n' @/LATIN CAPITAL LETTER O WITH GRAVE/

<@/&Sigma;/'"@/GREEK CAPITAL LETTER OMEGA/Привет_@/LATIN CAPITAL LETTER

AE/_28@/OGONEK/\n">;

}

/* Здесь /* еще одно предупреждение */

@/&Sigma;/ {

* '@/OGONEK/' и '@/HEBREW LETTER WIDE ALEF/' *

s.x 123 = 78;

/* '@/OGONEK/' и '@/HEBREW LETTER WIDE ALEF/'

@/GREEK CAPITAL LETTER OMEGA/ @/LATIN CAPITAL LETTER AE

*/

'@/&Psi;/' e.0, e.@/&psi;/' : e.1 '@/OGONEK/' e.2 = '@/HEBREW LETTER WIDE ALEF/' e.0;

e.x = ;

}
```

После нижеследующего диалога с пользователем, который может быть упрощен подбором ключей запуска препроцессора:

```
ref5_220602_untested> ./refgo refpp -m /w bak-macros_h.ref test_h.ref
Copyright (C): RefalScope Project, 2023.
Предупреждение:  удалить файл "test_h.ref"? (Да/Нет) >
Да
Предупреждение:  '/*' внутри пояснений: "/* ... /* '@/&psi;/' */" на строке 7 .
Предупреждение:  '/*' внутри пояснений: "/* ... /* еще одно предупреждение */" на строке 14 .
ref5_220602_untested> _
```

на выходе получаем программу test_h.ref:

```
$ENTRY Go { = <Hello>;}

Hello {

= <Prout <Examples>>;

}

Examples { /* '@/&psi;/' */

/* /* '@/&psi;/' */

e.1 = "Integral-∫f(z)dz"\n' Æ

('Ò') \n'

'(s.Òq)\n' Ò

<Σ \"ΩПривет_Æ_28\">;

}

/* Здесь /* еще одно предупреждение */

Σ {

* '@/OGONEK/' и '@/HEBREW LETTER WIDE ALEF/' *

s.x 123 = 78;

/* '@/OGONEK/' и '@/HEBREW LETTER WIDE ALEF/'

@/GREEK CAPITAL LETTER OMEGA/ @/LATIN CAPITAL LETTER AE

*/

"Ψ" e.0, e.ψ : e.1 ' ' e.2 = 'κ' e.0;

e.x = ;

}
```

Рекомендуется использовать следующую командную строку для вызова refpp:

```
linux> refgo refpp --m /w input.ref output.ref
```

О значениях выбранных здесь ключей – опций запуска расскажет вызов:

```
linux> refgo refpp /help
```

Чтобы препроцессор можно было использовать из любого каталога, нужно присвоить переменной среды REF5RSL путь к каталогу, в котором находится модуль refpp.rsl. В тот же каталог нужно скопировать и подкаталог unicode, который находится в головном каталоге дистрибутива. Переменной REF5RSL можно присвоить список путей, тогда поиск *.rsl модулей будет происходить согласно присвоенному списку до первой удачи.

Пример командного batch модуля, который показывает, как это делается в Linux, в среде оболочки bash:

```
REF5RSL=:test:/home/andrei/refal-5/refal-5_work/ref5_220602_untested_09082022-Terukha/refal5-26-08-2022/  
ref5_220602_untested  
export REF5RSL  
echo  
../refgo $1 $2 $3 $4 $5 $6 $7 $8 $9 || exit 1  
echo
```

Выше в первой строке (в присваивании) переноса нет. Знак двоеточия стоит после знака равенства, а не перед ним. Он формально говорит о том, что поиск нужно начинать с текущего каталога, хотя, по умолчанию он с текущего каталога и начинается.