**ВИДЕО ДОКУМЕНТАЦИЯ -** [**https://www.youtube.com/channel/UCvp-MnTxoqh06u7CviDBijA?view\_as=subscriber**](https://www.youtube.com/channel/UCvp-MnTxoqh06u7CviDBijA?view_as=subscriber)

**Официальный сайт RedL** - <https://www.red-lang.ru/>

**Соц. Сеть разработчика -** <https://vk.com/amnots>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Описание RedL :

RedL [Red Language] – свободный язык программирования с открытым исходным кодом.

Информация RedL:

1.Лицензия – GNU GPL v3.0 ([**https://www.gnu.org/licenses/gpl-3.0**](https://www.gnu.org/licenses/gpl-3.0)).

2.Последняя версия — 0.07Betta (Demo).

3.Цель — увеличить скорость написания кода за счёт упрощения системы языки и автоматизации компилятора.

Важно! Лицензия RedL не распространяется на следующие продукты:

1.Node JS.

2.Electron JS.

3.C++/G++/mingw.

4.Geany.

Процессы языка RedL:

1.Препроцессор –

a)Выполняет функцию создание промежуточного файла с извлечёнными файлами и запись их описания. b)Подготовка аргументов программы и последующая передача «»Переводчику»».

2.Переводчик -

a) Определение типа перевода (на какой язык переводить [c++/electron]).

b) Проверка корректности файла.

c) Перевод структур команд в соответствии с выбранным режимом. Вывод ошибок структур команд.

d) Подготовка переведённого файла к компиляции/интерпретации.

e) Проверка на соответствие синтаксиса (RedL) с учётом переменных и функций.

f) Запуск компилятора (c++/electron-packager).

g) Обратный перевод ошибок (c С++/Electron на RedL, если они есть) и последующий их вывод.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Документация к написанию программ на RedL**

**Часть 1 Аргументы RedL**

В языке программирования RedL мало аргументов ,но и в них можно запутаться. Перейдём к делу!

Для вызова RedL необходимо зайти в cmd (если у вас Windows ) или в Terminal (если у вас GNU/Linux).Затем набрать в нём «redl»(Важно! для благоприятного использования необходимо прописать PATH или запустить Makebat.bat в директории с RedL от имени администратора и согласится с предложениями прописать PATH автоматически).Вы увидите ошибку,которая гласит о том ,что файл для компиляции не найден. Вместе с ошибкой вы увидите и предложения аргументов компиляции их для консольных программ использовать не обязательно. Итак давайте я вам объясню,что они значат.

1. «-o» - выходной файл (название или расположение).Важно учитывать ,что указывать на файл и его расположение необходимо ***после* самого аргумента.**

**2.**»-i» - включить режим интерпретации кода (без создание .exe или .out файла).Его расположение — любое. Важно учитывать следующие: a) аргумент «-i» не может работать с аргументом «-d». b) режим интерпретации медленней компиляции.

3.»-d» - включить режим разработчика. Грубо говоря, при данном режиме не удаляется мусор и файлы других языков.

4.»-gui» - включить режим разработки визуальных приложений,но об этом позже.

5.Ваш исходный файл. Требования: a) он должен быть один (в аргументах). b) его расширение должно быть «.rlf».

**Часть 2 Консольная версия RedL**

**Первая программа - «Hello, world!»:**

Переходим к разработке. Для начала создадим файл (у меня это будет «test.rlf»).Открываем его через любой текстовый редактор.И для того ,чтобы нам вывести «Hello, world!» - потребуется лишь одна команда(«print»).

Вот пример первой программы:

Файл - «test.rlf»

Код - «print(“Hello, world!”)»

Важно, без «;». Каждая команда должна быть на своей строке (Есть пара исключений,но о них позже).

Теперь запускаем cmd/terminal и пишем - «redl test.rlf».Ждём и всё! Можем запускать уже появившийся файл - «test.exe»!

**Типы данных:**

В версии 0.07 Demo ещё мало типов данных. Их всего 4 (основных).

1.»INT» - целочисленный тип данных.

2.»STR» - тип строки, значение помещается в кавычках. Можно считать как одномерный массив.

3.»FLOAT» - число с плавающей точкой(например: 10.5 и т.д.).

4.«BOOL» - булево число( либо 0 [false],либо 1 [true] ).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Типы использующиеся только в функциях (на будущие):

5.»TEMP» - шаблонный тип (автоопределяемый).

6.»NNUM» - шаблонный тип (автоопределяемый, но только из чисел).

7.»LINT» - целочисленный тип данных. Ссылка на переменную (для изменение переменной в функции).

8.»LSTR» - тип строки. Ссылка на переменную (для изменение переменной в функции).

9.»LFLOAT» - число с плавающей точкой(например: 10.5 и т.д.).Ссылка на переменную (для изменение переменной в функции).

10.«LBOOL» - булево число( либо 0 [false],либо 1 [true] ).Ссылка на переменную (для изменение переменной в функции).

11.LTEMP - шаблонный тип (автоопределяемый).Ссылка на переменную (для изменение переменной в функции).

Объявление:

«INT name = 53» или «INT name» ,или «STR name = “Hello, world!”» и т.д. .

Где name – название переменной.

Также в языке программирования RedL есть массивы. Их объявление выглядит примерно так:

«INT[5] name».Где «[5]» - количество ячеек массива

Важно! ***В*** функциях запрещено размечать массив (можно — «INT[] name», нельзя - «INT[5] name»), а ***ВНЕ*** функциях - наоборот (можно — «INT[5] name», нельзя -«INT[] name»).

-Присваивание:

«INT name»

«name = 532»

«STR name»

«name = “Hello”»

«STR[5] name»

«name[0] = “Hello”»

Важно! Указывать на ячейку массива нужно в «[...]» и отсчёт ячеек начинается с «0»! Т.е. если у нас массив из 5-и ячеек,то их отчёт будет таким - «[0],[1],[2],[3],[4]».

-Обращение к типу «STR» (как к одномерному массиву):

«STR name = “Hello, world!”»

«print(STR[0])»

В результате получим - «H» (Первый символ переменной «name» ).

**Не много о команде «print»:**

Ранее (в «**Первая программа - «Hello world»**») уже шла речь о данной команде,однако я показал не все её возможности.

1)Команда «print» не переносит строку («\n») автоматически. Т.е. для того, чтобы перенести стоку нам потребуется написать спец. Символ «\n» самим:

«print(“Hello world\n”)».

Данный символ не будет показан в командной строке ,но зато перенесёт строку на новый уровень. Также есть и другие спец символы. Например:

«\t» - табуляция или же длинный пробел.

«\0» - показатель того ,что это конец строки (используется только в типе «STR» и «LSTR»).

«\b» - удаляет предыдущий символ ,а на его место следующий символ (за «\b»).

Это основные спец. Символы ,которые нужно знать.

2)Команда «print» может быть поточной. Имеется ввиду ,что текст команды может быть разделён (не зависимости от типа).Для

этого используем символ «$».Например:

«print(“Hello, world\n” $ “2 + 2 = ” $ 2+2)».

В результате получим - «Hello world»

«2 + 2 = 4».

У некоторых возникнет вопрос — откуда появилась «4»,ведь мы писали «2+2»? Это операторы (математические).Они работают. само собой, только с числами. Их всего 4:

«+» - прибавить. Только с типами «INT» и «FLOAT».

«-» - отнять. Только с типами «INT» и «FLOAT».

«/» - делить. Только с типами «INT» и «FLOAT».

«\*» - умножить. Только с типами «INT» и «FLOAT».

«%» - остаток при деление (например: 10 % 3 = 1 ).Только с типом «INT»

Хочется отойти от темы и рассказать про операторы присваивания (для переменных):

«=» - известный всеми оператор «равно».Во всех типах данных.

«+=» - «добавить что-то».Например:

«INT name = 5»

«name += 10» - «name» теперь равен 15.

«STR name = “Hello, ”»

«name += “world!”» - «name» теперь равен “Hello, world!”.

«-=» - отнять от переменной (в «STR» не работает).

«/=» - поделить переменную (в «STR» не работает).

«\*=» - умножить переменную (в «STR» не работает).

«%=» - остаток при деление переменной (в «STR» и «FLOAT» не работает).

**Проверки «if, else, elseif»:**

В языке программирования RedL особые проверки с так называемым «сахаром».Проверки нужны ,само собой, чтобы проверить что-либо. Основная команда проверки «if».О их структуре позже. Например:

«STR name = “Hello, world!”»

«if name == “Hello, world!”» - условия и вызов проверки

« print(“Yes!\n”)» - код (если условие выполнено он сработает).

«end» - конец выполнение кода (если условие выполнено).

В результате мы получим - «Yes!».Давайте теперь подробнее.

-Значение/операторы проверок:

«==» - равно (Важно именно 2 «=» ,а не 1!).

«!=» - не равно.

«>» - больше (только в «INT» и «FLOAT»).

«<» - меньше (только в «INT» и «FLOAT»).

«>=» - больше или равно (только в «INT» и «FLOAT»).

«<=» - меньше или равно (только в «INT» и «FLOAT»).

«or» - или.

«and» - и.

«in» - содержания текста в …

«!in» - содержания текста c учётом слов содержащих искомый текст.

-Структура проверок:

1)«STR name = “Hello, world!”»

«if name == “Hello, world!”»

« print(“Yes!\n”)»

«end»

2)«STR name = “Hello, world!” $ print(“Yes!\n”)» - Если условие выполнено, то выполнит **1** команду.

3)«STR name = “Hello, world!”»

«if name == “Hello, world!”»

« print(“Yes!\n”)»

«else» - иначе

« print(“No!\n”)»

«end»

4)«STR name = “Hello, world!”»

«if name == “Hello, world!”»

« print(“Yes!\n”)»

«elseif name == “TEST”» - иначе если...

« print(“No!\n”)»

«end»

Ранее я упоминал о «сахарном» синтаксисе проверок. Вот пара примеров:

1)«STR name = “Hello, world!”»

«if name == “Hello, world!” and :== “TEST” $ print(“unreal!\n”)»

2)«STR name = “Hello, world!”»

«if “world” in name or “ello” in: $ print(“real!\n”)»

3)«STR name = “Hello, world!”»

«if “world” !in name or “ello” !in: $ print(“unreal!\n”)»

**Циклы:**

В отличие от других языков программирования ,циклы в RedL выполняются одной командой - «cycle».Примеры:

1)«INT name = 0»

«cycle name<=20» - условие цикла

« print(name)»

« name++»

«end»

2)«INT name = 0»

«cycle name<=20 or :< 21» - условие цикла

« print(name)»

« name++»

«end»

Как вы поняли условия «cycle» идентичны условиям команды «if».Но что такое - «name++»? Это инкремент (т.е. прибавить переменную на 1 ),но не единственное ,что о них нужно знать. Примеры:

«name++» - инкремент «не важный»

«++name» - инкремент «важный» выполняется в любом случае и идёт в приоритете.

«name--» - декремент «не важный»

«--name» - декремент «важный» выполняется в любом случае и идёт в приоритете.

Что будет, если их не делать? Очень просто — цикл будет бесконечен.

-Команды для «cycle»:

«break» - остановить цикл.

«skip» - пропустить шаг цикла.

**Функции:**

Что такое функции? Это с одной стороны — способ сделать из множеств команд сделать одну, с другой — создание своей команды. Эти две стороны не взаимоисключаемы.Примеры:

1)«FUNCTION test()»

« print(“Hello, world!\n”)»

«END» - важно именно «END», а не «end»!

«test()» - вызов нашей функции

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2)«FUNCTION test(INT x)»

« print(“Hello, world!\n” $ x)»

«END» - важно именно «END», а не «end»!

«INT name = 25»

«test(name)» - вызов нашей функции с учётом аргумента «INT x»

-Типы функций и «return»:

«FUNCTION test(INT name) INT» - тип функции INT, соответственно функция должна вернуть любое число.

« if name == 5 »

« return 1» - возвращаем «1»

« else»

« return 0» - возвращаем «0»

« end»

«END»

-»C++» вставки в функциях и подключение библиотек «C++»:

«cpplib(“iostream”)» - только в библиотеке

«FUNCTION test()»

« cpp(“cout << \“Hello, world!\” << endl;”)» - только внутри функции библиотеки

«END»

-Расположение папки с библиотеками и их подключение:

1)Каталог: «../redl/library».

2)Подключение библиотеки: «library(“название файла библиотеки”)».

3Подключение любых файлов RedL с кодом: «import(“путь и название файла с кодом и расширением .rlf”)».

**Аргументы в своей программе:**

Для создание аргументов в свое программе необходимо использовать две переменные — «argc» и «argv».

Где «argc» - количество аргументов, а «argv» - массив из этих аргументов. Пример:

«STR name = “hello”»

«if argc > 2»

« if argv[1] == name»

« print(“Hello, world!”)»

« end»

«end»

Теперь если мы скомпилируем программу и через командную строку запустим нашу программу с аргументом «hello» ,то у программы выведет «Hello, world!».

**Повседневные команды:**

-Список:

1) «input(name)» – команда позволяет разработчика спросить (о чём либо) у пользователя.»name» - переменная (любой тип данных).

2) «console(“echo Hello, world!” $ 123)» – написать команду в BASH или в CMD. Идентична по структуре с командой «print».

3) «col(name)» - оставляет после себя число (количество ячеек в **массиве**).Работает только с массивами.

4) «len(name)» - оставляет после себя число (длину строки).Работает только с типом «STR» или его массив с указанной ячейкой.

**Нынешние библиотеки в RedL:**

На данный момент (при создании данной документации) их всего 2:

1) «files.rlf» - библиотека для работы с файлами.

2) «format.rlf» - библиотека для вспомогательного упрощения работы со строками.

**Библиотека «files.rlf»:**

-Перечень функций:

1) «filestr(name)» - Оставляет после себя число (количество строк в файле).»name» - название файла(переменная).

2) «readfile(file , array)» - Чтение файла и запись каждой строки в массив(array).»file» - название файла(переменная) , «array» - массив в который будет записан весь файл(тип только «STR[…]…»).

3) «readstr(file,3,var)» - Прочитать определённую строку (здесь это «3») файла. »file» - название файла(переменная), «var» - переменная «STR».

4) «fprintstr(file,3)» - Вывести строку (здесь это «3»). »file» - название файла(переменная).

5) «fprint(file)» - Вывести весь файл. »file» - название файла(переменная).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6) «outfile(file,”text”)» - Напечатать текст (“text”) в файл «file»(переменная).

7) «outfilestr(file,5,”text”)» - Напечатать текст (“text”) в файл «file»(переменная) на 5-ю сточку.

8) «outfileend(file,”text”)» - Напечатать текст (“text”) в конец файла «file»(переменная)

9) «outfilestart(file,”text”)» - Напечатать текст (“text”) в начало файла «file»(переменная).

Пример программы с этой библиотекой:

«library("files.rlf")»

«STR[filestr("123.txt")] x»

«readfile("123.txt", x)»

«INT z = 0»

«cycle z < col(x)»

« print(x[z] $ "\n")»

« z++»

«end»

**Библиотека «format.rlf»:**

-Перечень функций:

1) «int\_str(intvar, strvar)» - конвертировать значение «INT» («intvar») в тип «STR» и передать в переменную («strvar»).

2) «str\_int(strvar, intvar)» - конвертировать значение «STR» («strvar») в тип «INT» и передать в переменную («intvar»).

3) «flow(name,1)» - поток из слов (отделяемые пробелом - « »).Где «name» - само предложение («STR»), «1» - второе слово предложения. ! Команда возвращает результат на своё место !

**Примечание для разработчиков библиотек:**

Вспомним команды - «print/console...». Их объединяет структура («$»). Как сделать подобную команду ? Есть так называемый шаблон функций ,где при соблюдении условия (одного) — команда будет подобна «print/console...».Вот это условие:

! В аргументах функции должна быть одна переменная типа - «STR» или «TEMP» , или их ссылки !

! Тип функции может быть любым !

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Часть 3 Разработка графических приложений.**

Графическая разработка приложений в RedL очень простая. Ведь нам потребуется лишь «html/c++/js».В свою очередь RedL выступает в роли сборщика. Пример программы:

«build.rlf»

«icon("app\_ico.ico")» - Иконка нашего приложения. Обязательная команда. Расширение на windows должно быть «.ico», на linux «.png».

«width(300)» - Длина окна. Обязательная команда.

«height(370)» - Ширина окна. Обязательная команда.

«index("index.html")» - Файл резметки. Обязательная команда.

«resizable(1)» - Возможность изменять длину и ширину окна пользователю (1 — да, 0 — нет). Обязательная команда.

«execute("mkdir TESTREDL")» - Произвести команду в консоли (здесь — создать папку с названием «TESTREDL»). **НЕ о**бязательная команда.

«index.html»

«<!DOCTYPE html>»

«<html lang=”ru”>»

« <head>»

« <meta charset=”UTF-8”>»

« <title>TEST</title>»

« </head>»

« <body>»

« <h1>Hello, world!</h1>»

« </body>»

«</html>»

Теперь компилируем: «redl build.rlf -gui» ( «-gui» - обязателен ).