RBasic 2.0

Целью задания является создание интерпретатора языка RBasic - упрощенной версии самого популярного языка статистической обработки данных R http://www.r-project.org/.

Пример программы на RBasic:

```
x <- c(1,2,4,8,16) # создание вектора чисел, присваивание переменной x
y <- c(1,3,5) # создание вектора чисел, присваивание переменной y
z <- x[y] # в переменную z попадут элементы вектора x с индексами 1, 3, 5
z > 1 # Значение выражения FALSE TRUE TRUE
```

Лексический анализ

Синтаксис допускает использование \n, пробелов и символов табуляции между идентификаторами, символами табуляции круглыми и квадратными скобками, операциями и т.д.

RBasic поддерживает однострочные комментарии, начинающиеся с символа #, например:

```
x <- 1  # This is a comment
```

На этапе лексического анализа текст программы должен быть преобразован в набор лексем (ключевые слова, идентификаторы, отдельные терминалы, например, значки операций) .

Идентификаторы в RBasic аналогичны идентификаторам Си, но допускают использование символа «.»

Базовая версия RBasic должна поддерживать типы значений: булевый (logical), число с плавающей точкой (numeric), строковый (character). Формат задания констант типа число с плавающей точкой – такой же, как неэкспоненциальная форма записи числа в Си.

Строковые константы аналогичны таковым в Си. Обеспечить поддержку эскейп-последовательностей: \n, \t, \”

NULL – ключевое слово, означающее не заданное значение.

Примечание знатокам R: RBasic NULL не является аналогом NULL из R. Его семантика является смесью семантики оригинальных NULL и NAN. С целью упрощения мы не стали вводить две сущности.

Ключевые слова

TRUE, **FALSE**, **NULL**

Грамматика

Program → ε | Expression {[; | \n] Expression}
Expression → {Variable **<-**} Exp1
Exp1 → Exp2 {[**&**|**|**] Exp2}
Exp2 → Exp3 | **!** Exp3
Exp3 → Exp4 | Exp4 [**>**|**<** |**>=**|**<=**|**==**| **!=**] Exp4
Exp4 → Exp5 {[**+** | **-**] Exp5}

Exp5 → Exp6 {[*| **/**] Exp6}
Exp6 → Exp7 {**:** Exp7}
Exp7 → **(**Expression**)** | FunctionCall | Variable | Constant
Variable → Identifier | Identifier **[** Expression **]**
Constant → Numeric| Character | **TRUE** | **FALSE** | **NULL**
FunctionCall→Identifier **(** ArgList **)**
ArgList → ε | ArgListElem {**,** ArgListElem }
ArgListElem → Expression | Identifier **=** Expression

[a|b] означает выбор одной из альтернатив. {a} означает повторение а ноль или более раз.

Семантика операций

x<-3 #присваивание переменной x значения 3

Примечание: выражение, вызванное в глобальной области видимости (не внутри блока или функции для соответствующего варианта) и имеющее значение, и последняя операция которого не присваивание, выводит значение результата в stdout. Элементы вектора должны выводиться через пробел. Пустые векторы выводятся как number(0), logical(0), character(0), NULL(0).

> x # вывод выражения x на экран
3
Примечание: в RBasic нельзя создать переменную скалярного типа. Переменная x из примера – это вектор чисел с плавающей точкой длины 1.

Примечание: вектор может содержать значения только одного типа и значение NULL.

Элементы векторов нумеруются с 1, таким образом:

> x[1]
3

При присваивании вектора создается копия:
> y <- x
> y[1] <- 4
> x
3

Операция “:” генерирует вектор значений в рамках заданных границ с шагом 1 либо -1:

> 1:3
1 2 3
> 3:1
3 2 1
> 1.1:3.5
1.1 2.1 3.1
В случае, если аргумент ":" содержит более одного элемента, остальные игнорируются.

Семантика сравнений и логических операций. Сравнение:

> x<-c(1,2,3)
> y<-c(2,0,0)
> x>y

FALSE TRUE TRUE

Логические операции:

!, &, | - поэлементное НЕТ, И, ИЛИ

Примечание: в случае, если в **любой бинарной операции** один вектор оказывается короче другого, более короткий вектор циклически применяется к «хвосту» длинного:

```
> x<-c(1,2,3, 4,5)
> y<-c(1,2)
> x – y
0 0 2 2 4
```

Индексация вектора

1. Индекс вектора – вектор чисел – означает доступ к элементам вектора по индексу:
   ```
   > x<-c(1,2,4,8)
   > y<-c(1,3)
   > x[y]
   1 4
   ```
2. Индекс вектора – логический вектор – означает фильтрацию элементов вектора, остаются значения, для которых значения вектора-индекса равны TRUE:
   ```
   > x<-c(1,2,4,8)
   > y<-c(TRUE, FALSE, TRUE, TRUE)
   > x[y]
   1 4 8
   ```
   Если логический вектор-индекс короче чем вектор значений вектор-индекс циклически применяется:

   ```
   > x<-c(1,2,4,8)
   > y<-c(TRUE, FALSE)
   > x[y]
   1 4
   ```
   Если вектор значений короче чем логический вектор-индекс в результирующий вектор добавляются NULL:

   ```
   > x<-c(1,2)
   > y<-c(TRUE, FALSE, TRUE, TRUE)
   > x[y]
   1 NULL NULL
   ```

Удобно вписывать условие непосредственно в индекс:

```
> x<-c(1,2,4,8)
> x[x > 2]
4 8
```

При модификации элемента за границей вектор должен расширяться. Не заполненные элементы должны получать значение NULL:

```
> x<-c(1,2)
> x[5]<-3
> x
1 2 NULL NULL 3
```

Встроенные функции

c(значение1, ..., значениеN)

Создает вектор со значениями значение1, ..., значениеN:

```
> x<-c(1,2,4,8)
> x
1 2 4 8
```

length(переменная)

Возвращает длину вектора:

```
> x<-c(1,2,4,8)
> length(x)
4
```

mode(переменная)

Возвращает строку (вернее как обычно вектор из одной строки), содержащую тип переменной: «numeric», «logical», «character», либо «NULL» (пустой вектор не известного типа)

```
> x<-c(TRUE,FALSE)
> mode(x)
“logical”
```

Приведение типов

Для бинарных операций в ситуации “numeric” op “logical” и для унарной логической операции, применяемой к числу, реализовать автоматическое приведение типов:

```
> x <- 0
> y <- TRUE
> x + y
1
> x & y
FALSE
> !x
TRUE
```

Сроки сравнивать лексикографически.

При сравнении чисел либо логических переменных со строками приводить числа и логические переменные к строкам.

Любые унарные операции и бинарные операции с участием NULL (в которых другой аргумент м.б. использован в операции) должны возвращать NULL, кроме логических операций, в которых NULL не влияет на результат:

```
> 1+NULL
NULL
> TRUE & NULL
NULL
> TRUE | NULL
TRUE
> ! NULL
NULL
> 1 > NULL
NULL
> NULL == NULL
NULL
> “1” * NULL
2 Wrong types of args of operation ‘*’
```

Примечание: при этом сокращенное вычисление выражений не применять!

x<-TRUE | (z<-2) | NULL

x получит значение TRUE, z появится в пространстве имен со значением 2

При попытке модифицировать вектор значением типа, отличного от типа элементов вектора:

1. Приводить тип нового значения к типу значений в векторе
2. В случае добавления строки в вектор не строк приводить существующие в векторе значения к строкам

Обработка ошибок

При обнаружении ошибок синтаксиса или семантики интерпретатор должен выводить ошибку в поток stderr в формате:

<Положительный_код_ошибки> <Сообщение на англ.языке>

Предопределенные коды ошибок

1 Variable not defined

2 Wrong types of args of operation %s

Командная строка

1. Интерактивный интерпретатор языка RBasic д.б. доступен при запуске исполняемого файла RBasic. Выход Ctrl+C. В начале очередной строки для ввода должен отображаться символ >. Если до завершения оператора нажат Enter, в начале следующей строки **интерпретатор** должен отображать “ + ”, сигнализирующий о том, что оператор не завершен:

```
> x<-
 + y*
 + z/8
>
```

2. При запуске RBasicScript <имя скрипта> должен интерпретироваться соответствующий скрипт (код возврата программы 0 в случае отсутствия ошибок, код возврата 1 в случае ошибки). Интерпретатор скрипта точно также как интерактивный интерпретатор должен писать в stdout и stderr.

Дополнительные требования

1. Хороший C++
2. В архитектуре программы должен быть продуман механизм добавления в интерпретатор своих функций, написанных на C++
3. В состав отчетных материалов должны войти автоматические тесты
4. В состав отчетных материалов должна войти документация, сгенерированная по документирующим комментариям Doxygen
5. Программа должна содержать Makefile
6. По ходу интерпретации выражений они должны переводиться в ПОЛИЗ.

Дополнительные варианты

Блочный оператор (код варианта “block”)

Блочные оператор - последовательность выражений, заключенных в { }. Значением блочного оператора является значение последнего выражения. Значением пустого блочного оператора является NULL. Блочный оператор не приводит к созданию локальной области видимости.

Оператор ветвления (код варианта “if”)

Дополнительно требуется блочный оператор

Реализовать оператор ветвления:

if(условие) оператор1 [else оператор2]

В условии может присутствовать логический или числовой вектор ненулевой длины. На истинность проверяется его первое значение.

Оператором м.б. выражение или блочный оператор.

Примечание: в отличие от Си-подобных языков, конструкция if имеет значение! (отсутствующая ветка else имеет значение NULL)

Цикл for (код варианта “for”)

Дополнительно требуется блочный оператор

Реализовать цикл for.

for(тек_элемент in вектор) оператор

Оператором может быть выражение либо блочный оператор

Выход из цикла также возможен с помощью оператора break

Оператор цикла не имеет значения.

Цикл repeat (код варианта “repeat”)

Дополнительно требуется блочный оператор, if

Реализовать цикл repeat. Синтаксис:

repeat оператор

Оператором может быть выражение либо блочный оператор

Выход из цикла с помощью оператора break

Оператор цикла не имеет значения.

Цикл while (код варианта “while”)

Дополнительно требуется блочный оператор, if

Реализовать цикл while. Синтаксис:

while(условие) оператор

В условии может присутствовать логический или числовой вектор ненулевой длины. На истинность проверяется его первое значение.

Оператором может быть выражение либо блочный оператор

Выход из цикла с помощью оператора break

Оператор цикла не имеет значения.

Поддержка матриц (код варианта “matrix”)

Создание матрицы:

```
> m <- matrix(c(1,2,3,4,5,6), nrow=2, byrow=TRUE)
> m
1 2 3
4 5 6
```

При этом:

```
> mode(m)
“numeric”
> length(m)
6
```

Но реализовать функции is.vector и is.matrix, которые будут возвращать:

```
> is.matrix(m)
TRUE
> is.vector(m)
FALSE
```

Доступ к элементам матрицы:

```
> m[1,1]
1
```

Доступ к строке:

```
> m[1,] # в реализации не должно быть копирования!
1 2 3
> is.vector(m[1,])
TRUE
```

Доступ к столбцу:

```
> m[,2]
2 5
```

Как и со списками возможен синтаксис:

```
> m[1, c(2,3)]
2 3
```

Прототип функции matrix:

matrix(data = NULL, nrow = 1, ncol = 1, byrow = FALSE)

Стандартные векторные операции: + - * / > < >= <= == != ! & | применяются к матрицам поэлементно. При инициализации из data действует правило циклического повторения более “короткого” объекта (в данном случае data).

%*% - операция умножения матриц
Функция t - транспонирование:
> t(m)
1 4
2 5
3 6

Механизм описания функций (код варианта “function”)

Дополнительно требуется блочный оператор
Примеры функций:

```
f <- function(a=1, b, c) a+b
g <-function() {
y <- 1
x <- y-1
x
}
> f(2,3)
5
> g()
0
```

В примере f - вектор из одного элемента, содержащий функцию. Вектор функций имеет тип “function”:

```
> mode(f)
“function”
```

Тело функции - либо выражение, либо блочный оператор. Значением функции является значение этого оператора. Как видно из примера, при обращении к функции не обязательно использовать все формальные аргументы. Значение поумолчанию могут иметь не обязательно последние аргументы. Значением по умолчанию м.б. только константа. При интерпретации операции обращения к функции сначала значения получают все аргументы, вызванные по имени, затем происходит обработка оставшихся аргументов (вне зависимости от наличия значения по умолчанию) в порядке их следования, если остались не затронутые аргументы со значениями по умолчанию они получают соответствующие значения.
К функции f можно обратиться так:
f(b=2, 7)
В этом случае a получит значение 1.
Функция м.б. определена внутри тела другой функции.
Внутри функции “видны” глобальные переменные. Попытка их модификации из тела функции приведет к созданию локальной переменной.
Допустимо создание функций без присваивания вектору:

```
> (function(x) {x}) (8)
8
```

Может быть создан вектор из функций: f <- c(function {}, function {})