«Мир информатики»
Журнал для тех, у кого информатика – любимый школьный предмет

Выпуск № 57, июнь 2021 г.

Школа программирования

Работа со строками. Общие вопросы

Как отмечалось в статье [1], одним из основных типов данных в языках программирования является строковый тип. К нему относятся величины, значением которых является последовательность символов. Это может быть конкретная строка символов, которая указывается в кавычках (одинарных или двойных, в зависимости от языка программирования) – `'карандаш'`, `"Дмитрий"`, `'h'`, `''` и т. п.[1]), переменная (`s`, `фам` и т. п.) или выражение, результатом которого является последовательность символов (`имя + " " + фамил` и т. п.). В устной речи и при письме, как правило, этот тип данных называют «строковый тип», а сами данные – «строки».

Способы задания значений величинам строкового типа такие же, как и величинам других типов:

– с помощью оператора ввода данных:

```
вывод нс, "Введите строку символов "
ввод стр
```

– с помощью оператора присваивания:

```
имя := "Дмитрий"
```

В правой части оператора присваивания можно указывать также имя переменной величины строкового типа или имя функции (метода) для работы со строками. Примеры:

```
клуб := "Спартак"
мой_клуб := клуб
```

Можно также указать несколько таких значений, между которыми записывается знак «+»:

```
кто := имя + фамил
об_мне := "Я – болельщик команды " + клуб
текст := "Меня зовут" + " " + имя
```

В данном случае операция с знаком «+» называется «конкатенация».

Другие операции[2] над строками выполнять нельзя (напомним [1], что перечень операций определяется типом величины).

---

[1] Если строка ограничена одинарными кавычками, внутри нее могут быть двойные кавычки:

```
s2 := 'Я болельщик клуба "Спартак"'
```

и наоборот.

[2] В некоторых языках программирования возможна также операция «умножения» величины строкового типа на целое число.

При хранении значения величины строкового типа в памяти компьютера символы строки расположены подряд (в соседних ячейках). Модель памяти при этом можно представить в виде:

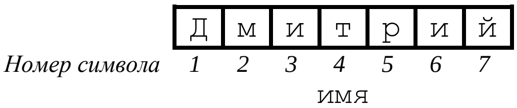


Рис. 1

В языке программирования Python и ряде других нумерация отдельных символов в памяти начинается с нуля. Здесь же напомним, что каждый символ в памяти компьютера хранится в виде двоичного числа – его кода.

Строки можно сравнивать между собой, и не только на предмет их равенства или неравенства:

```
если derevo > "дуб"
  то …
```

Вот несколько примеров сравнения строк (во всех случаях результат является истинным):

```
"а123" > "а"
"а2" > "а123"
"а123" < "аб"
"аб" < "б"
"б" < "б2"
"б2" < "б2д"
"б2д" < "бб"
"б" > "а"
```

Если выписать перечисленные строки в виде:

```
"а"
"а123"
"а2"
"аб"
"б"
"б2"
"б2д"
"бб"
```

то можно установить правило, по которому происходит сравнение двух строк: посимвольно слева направо сравниваются коды соответствующих символов до тех пор, пока не нарушится равенство кодов или не закончится одна из строк (или обе сразу), при этом сразу делается вывод о неравенстве и определяется, какая из строк меньше, а какая – больше. Две величины строкового типа являются равными, если они равны по длине и совпадают посимвольно.

Именно по такому правилу происходит сравнение строк при их сортировке. Порядок, в котором окажутся строки в результате сортировки, называют «алфавитным» (так расположены, например, слова в словарях).

Но как компьютер «знает», что такое «алфавитный порядок»? И как сравнивать слова, в которых есть и строчные, и прописные буквы, а также цифры и другие символы? Оказывается, при сравнении строк используются коды символов.

В кодовой таблице[3] прописные буквы стоят раньше строчных и поэтому имеют меньшие коды. Цифры стоят в кодовой таблице по порядку, причем раньше, чем латинские буквы; латинские буквы – раньше, чем русские; заглавные буквы (русские и латинские) – раньше, чем соответствующие строчные.

СОВЕТ № 10

Указанные особенности (цифры и буквы алфавита стоят в кодовой таблице по порядку) позволяют вместо многократного сравнения переменной величины со всеми буквами или цифрами использовать сложное условие со знаками сравнения.

Пример задачи. Дан символ. Определить, является ли он прописной латинской буквой.

Решение

Вместо такого, громоздкого, варианта:

```
ввод симв
если симв = "A" или симв = "B" или симв = "C" или … или симв = "Z"
  то
    вывод нс, "Да, является"
  иначе
    вывод нс, "Нет, не является"
все
```

можно использовать следующий:

```
ввод симв
если симв >= "A" и симв <= "Z"
  то
    …
```

Величина строкового типа рассматривается как массив, элементами которого являются отдельные символы. Это значит, что для того чтобы использовать в программе значение отдельного символа строки, нужно указать имя величины и в скобках (круглых или квадратных) – номер символа в памяти. Примеры на школьном алгоритмическом языке:

```
вывод стр[5]
вывод стр[k]
если стр[к + 1] = "Д"
  то
    …
```

В любой момент выполнения программы длину строки (количество символов в ней) можно определить с помощью стандартной функции. В школьном алгоритмическом языке имя этой функции – `длин`. Например, так в переменную `всего` записывается длина строки `стр`:

```
всего := длин(стр)
```

Эта функция используется, например, когда в операторе цикла с параметром требуется рассмотреть все символы величины строкового типа.

Аналогичные функции в языках Python и Паскаль:

Язык	**Имя функции**
Python	`len`
Паскаль	`length`

---

[3] Кодовая таблица – таблица, сопоставляющая каждому символу некоторое двоичное число, его код.

Приведем еще ряд стандартных функций для работы с величинами строкового типа.

`верхний регистр`

Возвращает строку, все символы которой приведены к верхнему регистру символов. Например, в результате выполнения оператора:

```
вывод нс, верхний регистр("Мастер и Маргарита")
```

на экран будет выведено:

```
МАСТЕР И МАРГАРИТА
```

Аналогично, функция `нижний регистр` возвращает строку, все символы которой приведены к нижнему регистру.

В языке Паскаль соответствующие функции называются `UpCase` и `LowCase`.

В языке Python для изменения регистра всех букв строки применяются методы `upper` и `lower`:

```
str = "Мастер и Маргарита"
str_AllUpper = str.upper
print(str_AllUpper)
str2 = "Московские Улицы"
str2_AllLower = str2.lower
print(str2_AllLower)
```

СОВЕТ

Функции, преобразующие все символы строки в их написание в верхнем или нижнем регистре, удобно использовать при сравнении двух значений, которые могут (по ошибке пользователя программы) отличаться регистром одной-двух букв. Как правило, это фамилии, имена и т. п. Например, если пользователь должен ввести фамилию и инициалы (имя переменной – `фио`) и какие-то действия должны выполняться, если это «Пупкин В.И.», то соответствующий оператор может быть оформлен в виде:

```
если верхний регистр(фио) = верхний регистр("Пупкин В.И.")
  то
    …
```

или

```
если нижний регистр(фио) = нижний регистр("Пупкин В.И.")
  то
    …
```

В этих случаях действия будут выполняться, даже если пользователь при вводе ошибочно введет не прописные, а строчные буквы или наоборот.

Здесь же заметим, что прописные и строчные буквы не различаются при проверке правильности ввода имени пользователя (логина) при входе; в то же время регистр букв введенного пароля при проверке учитывается.

Во всех языках программирования имеется возможность получить так называемый «срез» строки – некоторую последовательность её символов.

В школьном алгоритмическом языке для получения среза нужно после имени величины строкового типа указать в квадратных скобках номера первого и последнего символа требуемой части строки через двоеточие.

```
слово := "информатика"
срез := слово[3 : 7]
вывод срез
```

В языке Python в срезе в квадратных скобках указываются:
– номер первого символа требуемой части строки в памяти компьютера;
– число, на 1 большее номера последнего символа требуемой части строки в памяти компьютера.

Пример. Программа для получения слова `"menu"` (6–9-й символы строки `"File menu"`):

```
s = "File menu"
s2 = s[5:9]  # Срез строки s
print(s2)
```

В программах на языке Паскаль срез строки возвращает функция `Copy`. Её общий вид:

`Copy(`*`<строка>, <номер начального символа подстроки>,<количество символов>`*

В языке Паскаль можно также по-особому получить срез из нескольких первых или нескольких последних символов величины строкового типа. Это можно сделать, соответственно, с помощью функций `LeftStr` и `RightStr`. Первая возвращает первые `k` символов строки `st`, вторая – последние `k` символов:

```
LeftStr(st, k)
RightStr(st, k)
```

Выше уже отмечалось, что в памяти компьютера каждый символ хранится в виде двоичного числа – его кода. Десятичный эквивалент кода символа можно получить с помощью функций, указанных в таблице:

Язык программирования	**Имя функции**
Школьный алгоритмический язык	`код`
Паскаль	`ord`
Python	`ord`

Символ, код которого возвращается, указывается в кавычках в качестве аргумента функции (в скобках).

Имеются и другие функции и процедуры для работы со строками (их частями, в том числе с отдельными символами). О некоторых из них вы можете узнать ниже, а об остальных – по другим источниками.

Задания для самостоятельной работы

1. Дано слово. Вывести его буквы в «столбик».

2. Даны два слова. Определить, какое из них в словаре, в котором слова расположены в алфавитном порядке, будет записано раньше.

3. Даны две величины строкового типа. Определить, в какой из них больше символов или количество символов в них одно и то же.

4. Дан символ. Определить, является ли он цифрой.

5. Даны фамилия, имя и отчество человека. Сформировать из них:
а) одну общую величину;
б) величину в виде: «фамилия и инициалы» (например, так: «Иванов Н.А.»).

6. Даны два слова. Получить из них общую величину, в которой первое слово будет записано прописными буквами, а второе (после пробела) – строчными.

7. Дано слово из четного количества букв. Определить, одинаковы ли его первая и вторая половины.

8. Дана строка. Найти сумму десятичных кодов ее первого и последнего символов.

9. Дано слово. Проверить, является ли оно палиндромом («палиндромом» называется слово, которое одинаково читается как слева направо, так и справа налево, например, такими словами являются «потоп» и «дед»).

Дополнение

В данном дополнении расскажем о функциях, которые преобразовывают числовые величины в величины строкового типа. В школьном алгоритмическом языке это функции `цел_в_лит` и `вещ_в_лит`. Например, чтобы преобразовать целое число `n` в величину строкового типа `n_стр` следует записать:

```
n_стр := цел_в_лит(n)
```

Для преобразования вещественного числа:

```
n_2_стр := вещ_в_лит(n)
```

Аналогичные функции языка Паскаль – `IntToStr` и `FloatToStr`.

Соответствующие задачи в языке Python выполняет функция `str`, которая переводит целое или вещественное число, указанное в скобках в качестве ее аргумента, в строку.

Такое преобразование позволяет просто решить многие задачи по обработке натурального (и не только) числа. Решение основано на том, что, как уже отмечалось, величина строкового типа рассматривается как массив, элементами которого являются отдельные символы.

В приведенных далее фрагментах программ, основанных на таком подходе, использованы следующие основные величины:

`n` – заданное число;
`n_стр` – его строковое представление;
`ном` – номер символа в строковом представлении;
`m` – номер цифры в числе (при счете справа налево или слева направо).

Д1. Определение количества цифр заданного натурального числа

```
ввод n
кол_цифр := длин(n)
| Вывод ответа
…
```

Без преобразования числа в строку нужно использовать оператор цикла с условием, операции определения остатка и др.

Д2. Выделение цифр

Задача формулируется так: «Дано натуральное число. Вывести его цифры в “столбик”».

Без преобразования числа в строку задача решается так, как показано в пункте 2.3.2. А вот как – с использованием такого преобразования:

```
ввод n
| Преобразовываем число в строку
n_стр := цел_в_лит(n)
| Выводим каждый символ строки
нц для ном от 1 до длин(n_стр)
  вывод нс, n_стр[ном]
кц
```

Д3. Определение m-й справа цифры натурального числа

Без преобразования числа в строку задачу можно решить так. Последовательно в цикле выделяем цифры, использую операции определения остатка и целочисленного деления, одновременно считаем выделенные цифры, используя переменную-счетчик. Когда значение счетчика станет равно m, выводим последнюю выделенную цифру. Соответствующий фрагмент:

```
ввод n
к := 0                        | Переменная-счетчик
нц пока n > 0:
  посл := n mod 10
  к := к + 1                  | Увеличиваем счетчик
  если к = m                  | Встретилась m-я цифра
    то
      вывод нс,  посл         | Выводим ее
   все
   n := n div 10
кц
```

Можно уменьшить размер программы, учитывая, что мы знаем, сколько раз нужно выделять цифры (m раз). Поэтому можно использовать цикл с параметром:

```
…
нц для к от 1 до m
  посл := n mod 10
  n := n div 10
кц
| Выводим последнюю выделенную (m-ю) цифру
вывод нс, посл
```

А теперь – вариант с преобразованием числа в строку:

```
ввод n
ввод m
n_стр := цел_в_лит(n)
вывод нс, "Это цифра ", n_стр[длин(n_стр) - m + 1]
```

Д4. Определение m-й слева цифры натурального числа

Без преобразования числа в строку данную задачу за один проход решить нельзя. Сначала надо определить общее количество цифр. Потом установить, какой номер имеет m-я слева цифра при счете справа налево и еще раз «пройти по числу» и выделить нужную цифру.

А вот программа с преобразованием числа в строку:

```
ввод n
ввод m
n_стр := цел_в_лит(n)
вывод нс, "Это цифра ", n_стр[m]
```

Как вам она, уважаемый читатель?

Д5. Определение цифры с заданным номером вещественного числа

Программу решения задачи предлагаем читателю разработать самостоятельно.

Можно также находить сумму цифр, максимальную/минимальную цифру и ее номер и т. д. Одним из наиболее эффектных (и эффективных) примеров использования описанного приема являются задачи так называемой «длинной арифметики», то есть задачи, связанные с расчетами с очень большими числами, значения которых превышают допустимый формат их хранения, например, расчеты со 100-значными числами.

Типовые задачи обработки строк

1. Обработка отдельных символов строк

Во всех рассмотренных далее задачах заданная строка имеет имя `стр`.

1.1. Определить, сколько раз в заданной строке встречается некоторый символ симв

Решение

Нужно сравнить с заданным символом каждый символ строки и в случае их равенства увеличить значение переменной-счетчика на 1:

```
    ...
    кол := 0                      | Переменная-счетчик
    нц для к от 1 до длин(стр)    | Рассматриваем все номера символов
строки
      если стр[к] = симв          | Сравниваем соответствующий символ
        то                        | с заданным
          кол := кол + 1
      все
    кц
    | Выводим ответ
    если кол > 0
      то
        вывод нс, "Заданный символ встречается в строке ", кол, " раз"
      иначе
        вывод нс, "Такого символа нет"
    все
```

В программе на языке Python можно также рассматривать не номера символов, а сами символы:

```
kol = 0
for c in str:  # Рассматриваем все символы c строки str
    if c == simv:
        k = k + 1
…
```

В языках программирования Паскаль и Python для решения рассматриваемой задачи может быть использован функция `count` (с учетом особенностей её применения).

1.2. Определить позицию (номер) первого вхождения некоторого символа в заданную строку

Сначала обсудим вариант задачи, в котором заведомо известно, что заданный символ в строке имеется.

Данная задача может быть решена путем использования оператора цикла с параметром и переменной-флага:

```
встретился_первый := нет
нц для к от 1 до длин(стр)
  если стр[к] = симв
    то | Проверяем, это первый символ?
      если встретился_первый = нет  | Или    если не встретился_первый
        то
          искомый_номер := к
          встретился_первый := да
    все
  все
```

```
кц
вывод нс, искомый_номер
```

или (без проверки всех символов строки):

```
нц для к от 1 до длин(стр)
  если стр[к] = симв
    то
      искомый_номер := к
      | Прекращаем обработку строки
      выход
  все
кц
вывод нс, искомый_номер
```

Если же допускается, что искомого символа в строке может не быть, то задача решается так же, как решалась задача аналогичная задача применительно к массиву. Разработайте соответствующую программу самостоятельно.

В языке Python существует функция для поиска позиции (номера символа), в которой начинается первое вхождение подстроки (и отдельного символа) в строке. Эта функция называется `find()`. В скобках нужно указать образец для поиска:

```
str = input("Введите строку ")
simv = input("Введите символ ")
ind = str.find(simv)        | Функция find() вызывается как метод
```

Эта функция возвращает номер заданного символа в памяти компьютера; если заданного символа в строке нет, – возвращается значение –1. Поэтому вывод ответа оформляется следующим образом:

```
if ind != -1:
  print(ind + 1)
else:
  print("Такого символа в строке нет")
```

Например, если `str` = "Информатика", а `simv` == "а", то будет выведен ответ 7 (номер заданного символа в памяти компьютера, увеличенный на 1; такой ответ в данном случае более привычен пользователю программы).

Имя аналогичной функции в языке Паскаль – `Pos`. Ее первый аргумент – искомая подстрока, второй – строка, в которой происходит поиск. Если заданного символа в строке нет, – возвращается значение 0.

Обратим внимание на то, что во всех случаях строчные и прописные буквы при поиске отличаются.

1.3. Определить, есть ли в заданной строке некоторый символ `симв`

Прежде всего, заметим, что такой вариант решения:

```
нц для к от 1 до длин(стр)
  если стр[к] = симв
    то
      вывод "Заданный символ в строке есть"
    иначе
      вывод "В строке нет заданного символа"
  все
кц
```

– ошибочный – на экран ответ будет выводиться многократно.

Неправильный результат дает также такой способ, использующий переменную-флаг `есть`, которая фиксирует факт наличия заданного символа в заданной строке:

```
нц для к от 1 до длин(стр)
  если стр[к] = симв
    то
      есть := да
    иначе
      есть := нет
  все
кц
если есть  | Или          если есть = да
  то
    вывод "Заданный символ в строке есть"
  иначе
    вывод "В строке нет заданного символа"
все
```

Его особенности проанализируйте самостоятельно.

Наиболее простой правильный способ решения задачи такой:
1) подсчитать число вхождений заданного символа в заданную строку (см. задачу 1);
2) по найденному значению получить ответ.

Можно также не рассматривать всю строку, а прекратить проверки, как только встретится заданный символ:

```
к := 1
найден := нет
нц пока к <= длин(стр) и не найден
  если стр[к] = симв
    то
      найден := да
    иначе | Переходим к следующему символу
      к := к + 1
  все
кц
если найден
  то
    вывод нс, "Заданный символ в строке есть"
  иначе
    вывод нс, "В строке нет заданного символа"
все
```

В программе на языках Паскаль и Python задача также решается просто благодаря использованию функциям, соответственно, `find()`и `Pos` (в них предусмотрен вывод результатов поиска при отсутствии в строке заданного символа).

В программе на Python лучше всего использовать оператор проверки принадлежности `in`, который проверяет, имеется ли символ в заданной строке:

```
if simv in str:
  print("Заданный символ в строке есть")
else:
  print("В строке нет заданного символа")
```

1.4. Определить, является ли символ строки с заданным номером цифрой

Задача может быть решена разными способами.
Во-первых, можно сравнить заданный символ с каждой цифрой:

```
| к – заданный номер символа
если стр[к] = "0" или стр[к] = "1" или … или стр[к] = "9"
  то
    вывод "Да, заданный символ – цифра"
  иначе
    вывод "Нет, заданный символ цифрой не является"
все
 …
```

Во-вторых, можно учесть, что коды символов-цифр расположены в кодовой таблице по порядку. Их десятичные эквиваленты: 48, 49, …, 57:

```
если код(стр[к]) >= 48 и код(стр[к]) <= 57
  то
    …
```

СОВЕТ

Если вы забудете коды цифр, то можно вместо конкретных значений кода нуля и цифры 9 использовать функцию, возвращающую эти коды:

```
если код(стр[к]) >= код("0") и код(стр[к]) <= код("9")
  то
    …
```

В языке Python код цифры 0 можно определить в интерактивном режиме или в программе так:

```
print(ord("0"))
```

В языке Python возможны два оригинальных способа решения задачи. Один из них заключается в использовании:

1) строки с именем, например, `vse`, со всеми цифрами:

2) оператора проверки принадлежности **`in`**:

```
vse = "0123456789"
if str[nom] in vse:
  print("Да …")
else:
  print("Нет …")
```

А второй, дающий самое краткое решение, использует метод `isdigit()`. Этот метод проверяет, состоит ли строка из цифр, и возвращает в качестве результата логическое значение (`True` или `False`):

```
if str[nom].isdigit(): # В данном случае
                       # проверяется строка из одного символа
  print("Да …")
else:
  print("Нет …")
```

Задача для самостоятельной работы

Определить, есть ли в заданной строке цифры?

2. Обработка подстрок

Термином «подстрока» условимся называть заданную последовательность символов, представляющую часть строки.

Во всех рассмотренных далее задачах заданная строка имеет имя `стр`, а подстрока – `подстр`.

2.1. Определить, сколько раз в заданной строке встречается некоторая подстрока

Решение задачи во многом аналогично решению задачи 1.1. Единственное отличие в том, что проверять надо не по одному символу, а по несколько. При анализе нам понадобится получать и сравнивать с заданной подстрокой несколько последовательных символов строки – срез (см. начало данного выпуска). При этом возникает вопрос – чему равен номер последнего символа строки, начиная с которого следует формировать срез?

Обозначим количество символов в заданной строке – `длина1`, в подстроке – `длина2`. Рассмотрим несколько возможных значений:

длина1	**длина2**	**Номер последнего символа строки для формирования «среза»**
10	2	9
10	3	8
15	5	11
15	6	10

Из таблицы можно установить, что номер последнего символа строки для формирования среза определяется по формуле: `длина1 – длина2 + 1`.

Итак, фрагмент программы решения задачи:

```
длина1 := длин(стр)
длина2 := длин(подстр)
кол := 0
нц для к от 1 до длина1 – длина2 + 1
если стр[к : (к + длина2 – 1)] = подстр  | Срез совпадает с подстрокой
то
кол := кол + 1
все
кц
```

2.2. Определить позицию (номер) первого вхождения некоторой подстроки в заданную строку

Если заведомо известно, что заданная подстрока в строке имеется, то задача решается аналогично задаче 1.2:

```
…
длина1 := длин(стр)
длина2 := длин(подстр)
встретилась_первая := нет
нц для к от 1 до длина1 – длина2 + 1
  если стр[к : (к + длина2 – 1)] = подстр
    то | Проверяем, это первый срез, совпадающий с подстрокой?
      если встретилась_первая = нет   | или
                                      | если не встретилась_первая
        то
          искомый_номер := к
          встретилась_первая := да
    все
```

```
      все
    кц
вывод нс, искомый_номер
```

или (без проверки всех символов строки):

```
…
длина1 := длин(стр)
длина2 := длин(подстр )
нц для к от 1 до длина1 – длина2 + 1
  если стр[к : (к + длина2 – 1)] = подстр
    то
      искомый_номер := к
      | Прекращаем обработку строки
      выход
  все
кц
вывод нс, искомый_номер
```

Если же допускается, что искомой подстроки в строке может не быть, то задача решается аналогично тому, как решалась подобная задача в главе 8. Разработайте соответствующую программу самостоятельно.

В программе на языках Паскаль и Python обсуждаемая задача также решается просто благодаря использованию функциям `find` и `Pos`, соответственно.

2.3. Определить, есть ли в заданной строке некоторая подстрока

Различные способы решения данной задачи аналогичны описанным применительно к отдельному символу строки.

2.4. Удалить из заданной строки все вхождения некоторой подстроки

В программе на школьном алгоритмическом языке удалить из строки подстроку можно, присвоив срезу строки (см. выше) «пустое» значение, например, для подстроки из четырех символов, начиная с 7-го:

```
стр[7:10] := ""
```

Еще две важные особенности программы решения задачи.

1. Поскольку в ходе обработки строки ее длина меняется, использовать оператора цикла с параметром, в конечном значении которого применяется величина `длина1` (количество символов в заданной строке) – нельзя. Следует применить оператор цикла с условием следующего вида:

```
к <= длин(стр) - длина2 + 1
```

– где `длина2` – количество символов в удаляемой подстроке.

2. При рассмотрении `к` начальных номеров строки после нахождения искомой подстроки и ее удаления величина `к` не меняется (убедитесь в этом!), а в случае, когда искомая подстрока не встретилась, – увеличивается на 1.

С учетом сказанного фрагмент программы, решающей задачу, имеет вид:

```
к := 1
нц пока к <= длин(стр) - длина2 + 1
если стр[к : (к + длина2 - 1)] = подстр
  то | Встретилась искомая подстрока
     | Удаляем ее
     стр[к : (к + длина2 - 1)] := ””
```

```
    | Величина к - не меняется
  иначе
    | Переходим к следующему символу для проверки
    к := к + 1
  все
кц
```

В программе решения задачи на языке Python возникает проблема, связанная с тем, что в этом языке величина строкового типа является так называемым «неизменяемым объектом». Имеется в виду, что нельзя изменить отдельный символ строки, удалить отдельный символ (или подстроку)[4] и нельзя что-то вставить в строку[5]. Но все эти операции можно провести, разработав специальные программы. Например, для удаления части строки `str` нужно составить новую строку `str2`, объединив в ней части исходной строки до и после удаляемого участка (см. рис. 11.1). Нужные части можно получить с помощью среза:

```
str2 = str[0 : nach – 1] + str[kon : len(str)]
```

*Примечание***.** Использованы номера `nach` и `kon`, соответствующие номерам символов в заданной строке (напомним, что в языке программирования Python нумерация отдельных символов в памяти начинается с нуля).

В данном случае новой строке можно присвоить имя исходной строки:

```
str = str[0 : nach – 1] + str[kon : len(str)]
```

Удаляемый фрагмент

Символы										
Номер	*1*	*2*	*3*	*...*	*nach*	*...*	*kon*	*...*	*...*	*len(str)*

Рис. 1

Итак, как же удалить из строки все вхождения некоторой подстроки? Вспомним решение задачи 2.2. В ней многократно формировались срезы заданной строки некоторой длины, которые сравнивались с заданной подстрокой.

В нашей задаче вместо переменной `k` будем использовать переменную `ykaz`, которая будет хранить значение номера символа, начиная с которого формируется срез для сравнения с заданной подстрокой `podstr`. Будем называть эту переменную «указатель».

Схема действий следующая:

```
Повторение действий, начиная с 1-го символа (ykaz == 0):
  Если срез не совпадает с заданной подстрокой:
    Добавляем текущий символ в новую строку
    Смещаем указатель на 1 позицию вправо
  иначе:    # Встретилась искомая подстрока
    Пропускаем ее (смещаем указатель на len(podstr) позиций вправо)
```

Возникает вопрос: «Сколько раз надо повторить указанные действия?» Неизвестно. Значит, оператор цикла с параметром применить нельзя. Но мы знаем (см. раздел 2.1), что последнее значение переменной `ykaz` равно `len(str) – len(podstr)`[6]. Значит, можем использовать оператор цикла с условием:

---

[4] Можно удалить один или несколько начальных и/или конечных символов (пробелов и т. п.) с помощью стандартных методов `lstrip()`, `rstrip()` и `strip()`.

[5] В то же время величины типа строкового типа можно «удлинять»:

```
st = "Волк"
st = st + "и"
```

[6] С учетом особенностей нумерации символов в программе на языке Python.

```
str2 = ""    # Начальное значение новой строки
ykaz = 0     # Начальное значение указателя
while ykaz <= len(str) - len(podstr):
  # Сравниваем срез и заданную подстроку
  if str[ykaz : ykaz + len(podstr)] != podstr:
    # Добавляем текущий символ str[ykaz] в новую строку
    str2 = str2 + str[ykaz]
    # Смещаем указатель на 1 позицию вправо
    ykaz = ykaz + 1
  else:   # Встретилась искомая подстрока
    # Пропускаем ее
    # (смещаем указатель на len(podstr) позиций вправо)
    ykaz = ykaz + len(podstr)
```

После того как «перемещение» указателя прекратится, в конце заданной строки могут остаться символы. Необходимо их также добавить к значению `str2`:

```
# Добавляем оставшуюся часть (используя срез)
str2 = str2 + str[ykaz : len(str)]
```

Подход к решению задачи, описанный применительно к языку Python (с использованием новой строковой переменной), может быть использован и в программах на других языках программирования.

Отметим интересный момент. В школьном алгоритмическом языке, языке Паскаль и ряде других имеется процедура, удаляющая из строки заданное количество символов, начиная с символа с некоторым номером:

```
удалить(<строка>, <начальный номер>, <количество символов>)
Delete(<строка>, <начальный номер>, <количество символов>)
```

Если вы попытаетесь применить эту процедуру для решения обсуждаемой задачи, то убедитесь, что использование стандартной процедуры (или функции) в ряде случаев значительно усложняет программу.

Задание для самостоятельной работы

Разработайте программы решения обсуждаемой задачи:

а) по описанной методике;

б) с использованием процедуры удаления подстроки (при ее наличии в используемом языке программирования).

2.5. Заменить в заданной строке все вхождения некоторой подстроки на другую подстроку

Используем в программе величины:

`стр` – заданная строка;
`подстр1` – заменяемая подстрока;
`подстр2` – новая подстрока;
`длина1` – количество символов в величине `подстр1`;
`длина2` – то же, в величине `подстр2`.

Приводя фрагмент программы решения задачи, обращаем внимание на условие в операторе цикла (см. предыдущую задачу), а также на особенности изменения величины `к`:

```
…
    длина1 := длин(подстр1)
    длина2 := длин(подстр2)
    к := 1
```

```
нц пока к <= длин(стр) - длина1 + 1
 если стр[к : (к + длина1 - 1)] = подстр1
   то | Встретилась искомая подстрока
    | Заменяем ее
    стр[к : (к + длина1 - 1)] := подстр2
    | Изменяем значение величины к
    к := к + длина2
  иначе
     | Переходим к следующему символу для проверки
     к := к + 1
 все
кц
```

В школьном алгоритмическом языке имеется процедура, заменяющая в строке все вхождения заданной подстроки на другую подстроку. Аналогичные функции с именем `replace` в языках Паскаль и Python используются как метод.

Задание для самостоятельной работы

Разработайте программы решения обсуждаемой задачи:

а) по описанной методике;

б) с использованием процедуры замены подстроки (при ее наличии в используемом языке программирования).

11.3. Выделение слов предложения

В рассмотренных в данном разделе задачах обрабатывается задаваемое во время выполнения программы предложение `пред`, имеющее в общем случае следующую структуру:

слово	пробел	слово	пробел	…	слово

3.1. Выделение первого слова

Сначала заметим, что первые `к` символов заданной строки `стр` можно объединить в подстроку `подстр`, рассмотрев каждый из них с помощью оператора цикла с параметром и используя конкатенацию (см. начало данной статьи 10):

```
подстр := ""  |  Начальное значение формируемой подстроки
нц для ном от 1 до к | ном – номер символа
  подстр := подстр + предл[ном]
кц
```

В нашей же задаче количество символов для конкатенации заранее неизвестно. Но мы знаем, что после первого слова находится пробел. Значит, для решения задачи нужно проводить конкатенацию всех начальных символов предложения, отличающихся от пробела. Для таких повторяющихся действий следует применить оператор цикла с предусловием:

```
вывод нс, "Введите предложение "
ввод предл
слово1 := ""     | Формируемое слово (начальное значение)
ном := 1         | Номер очередного символа строки (начальное значение)
нц пока предл[ном] <> " " | Пока не встретится 1-й пробел
  | Добавляем текущий символ к "старому" значению величины слово1
  слово1 := слово1 + предл[ном]
  | Переходим к следующему символу
  ном := ном + 1
кц
| Встретился пробел, то есть первое слово закончилось
```

```
| Выводим его
…
```

Программа на языке Паскаль может быть также улучшена за счет того, что вводимые символы предложения можно считывать сразу во время их ввода, используя оператор `read` с параметром символьного типа:

```
var slovo1: string; c: char;
BEGIN
write('Введите предложение ')
repeat
  {Считываем очередной введенный символ}
  read(c);
  {и добавляем его к "старому" значению величины slovo1}
  slovo1 := slovo1 + c
until c = ' ';
…
```

Это позволяет отказаться от использования переменных, соответствующих обрабатываемому предложению и номеру символа, и не вводить все предложение. Однако в этом случае пробел будет включен в состав первого слова.

Можно также решить задачу так:

1) найти позицию (номер в строке) первого пробела. Это можно сделать, используя соответствующую функцию (см. раздел 1.2);

2) получить искомое слово, используя срез символов от первой буквы предложения до буквы с номером, на 1 меньшим найденного на этапе 1.

Соответствующую программу разработайте самостоятельно.

3.2. Выделение второго слова

Для решения задачи сначала следует пропустить первое слово:

```
ном := 1
нц пока предл[ном] <> " "
  | Переходим к следующему символу
  ном := ном + 1
кц
| Величина ном указывает на первый пробел
```

После этого можно сформировать второе слово:

```
...
слово2 := ""   | Формируемое слово (начальное значение)
| Переходим к 1-му символу 2-го слова
ном := ном + 1
нц пока строка[ном] <> " "    | Пока не встретится 2-й пробел
  | Добавляем текущий символ к "старому" значению величины слово2
  слово2 := слово2 + строка[ном]
  | Переходим к следующему символу
  ном := ном + 1
кц
…
```

В программе языке Паскаль, как и в программе решения предыдущей задачи, вводимые символы строки можно считывать сразу во время их ввода, используя оператор `read` с параметром символьного типа:

```
{Пропускаем первое слово}
    repeat
      read(c);
    until c = ' ';
    {Формируем второе слово}
    repeat
      {Считываем очередной введенный символ}
      read(c);
      {и добавляем его к "старому" значению величины slovo1}
      slovo1 := slovo1 + c
    until c = ' ';
```

Обращаем внимание на тот факт, что и здесь второй пробел будет включен в состав второго слова.

Можно также решить задачу, определяя позиции первого и второго пробелов и используя срез. Для определения позиции второго пробела можно в функциях, возвращающих позицию первого вхождения подстроки, указать номер символа, начиная с которого проводить поиск. В школьном алгоритмическом языке поиск начиная с символа с номером `начало` проводит функция `поз после`:

```
позиц_пробела := поз после(начало, " ", предл)
```

Поиск с определенной позиции возможен и в программе на языке Паскаль. Для этого в функции `Pos` следует указать номер символа, начиная с которого проводить поиск, в качестве третьего аргумента функции.

Соответствующий вариант программы разработайте самостоятельно.

Другие задачи для самостоятельной работы

1. Получить два первых слова заданного предложения.
2. Получить фразу, состоящую из двух первых слов заданного предложения и пробела между ними.
3. Вывести на экран последнее слово заданного предложения.

3.3. Выделение всех слов

Рассмотрим сначала случай, когда количество слов в предложении известно и равно `n`. В этом случае надо выделить первые (`n - 1`) слов, после которых стоит пробел. Такие отдельные слова мы выделять научились (см. предыдущие задачи). Чтобы применить оператор цикла с параметром и провести (`n - 1`) повторений, надо вместо имен величин `слово1` и `слово2` использовать одно имя – `слово`.

Начальная часть соответствующей программы, формирующая и печатающая первые (`n - 1`) слов с использованием конкатенации:

```
n := …
вывод нс, "Введите предложение из ", n, " слов"
ввод предл
ном := 1
нц для к от 1 до n - 1      | (n - 1)  раз
  | Формируем очередное слово
  слово := ""  | Начальное значение
  нц пока предл[ном] <> " "
    | Если символ - не пробел
    | Добавляем его к величине slovo
    слово := слово + предл[ном]
    | Переходим к следующему символу
    ном := ном + 1
  кц
```

```
    | Встретился пробел – слово закончилось
    | Печатаем его
    вывод слово, " "
    | Переходим к следующему символу – началу следующего слова
    ном := ном + 1
  кц
```

Можно также выделять слова, используя срез, предварительно определяя позицию очередного пробела. Если последнюю переменную величину обозначить `позиц`, то фрагмент, в котором выделяются и выводятся на экран `n - 1` слов предложения, оформляется так:

```
    n := …
вывод нс, "Введите предложение из ", n, " слов"
ввод предл
начало := 1  | Начальный номер символа, начиная с которого определяется
                    | позиция ближайшего пробела
нц для к от 1 до n - 1    | (n - 1)раз
  | Получаем очередное слово
  | Определяем позицию ближайшего пробела
  позиц := поз после(начало, " ", предл) | начиная с позиции начало
  | Определяем слово, используя срез
  слово := предл[начало : позиц - 1]
  вывод слово, " "
  | "Переходим" на первую букву следующего слова
  начало := позиц + 1
кц
```

Осталось получить и вывести на экран последнее слово. При использовании среза это можно сделать, учитывая, что мы знаем порядковый номер первой буквы последнего слова (`начало`) и номер последней буквы предложения:

```
    …
    | Последнее слово
    слово := предл[начало + 1 : длин(предл)]
    вывод слово
```

В программе с конкатенацией завершающая часть, в которой формируется и выводится последнее слово, будет такой («складываем» буквы от буквы с номером `ном` до последней):

```
    …
    слово := ""  | Начальное значение
    нц пока ном <= длин(предл)
        слово := слово + предл[ном]
        | Переходим к следующему символу
        ном := nom + 1
    кц
    | Закончилось последнее слово
    | Печатаем его
    вывод слово
```

Но, как правило, количество слов в предложении заранее неизвестно. Как решить задачу в таком случае? Ответ – при многократном выделении отдельных слов использовать только оператор цикла с предусловием.

В программе с конкатенацией условие, которое записывается в таком операторе, должно быть аналогичным тому, которое использовалось при выделении последнего слова (см. выше):

```
    вывод нс, "Введите предложение "
    ввод предл
    ном := 1
    слово := ""
```

```
нц пока ном <= длин(предл)
  | Формируем очередное слово
  если предл[ном] <>  " "    | Если символ - не пробел
    то
      | Добавляем его к величине слово
      слово := слово + предл[ном]
      | Переходим к следующему символу
      ном := ном + 1
      иначе | Встретился пробел - слово закончилось
            | Печатаем его
        вывод слово, " "
        | Готовимся формировать новое слово
        слово := ""
        | Переходим к следующему символу - началу следующего слова
        ном := ном + 1
  все
кц
| В конце работы оператора цикла было сформировано,
| но не напечатано последнее слово
| Печатаем его
вывод слово
```

В программе, использующей срез, рассуждения немного сложнее. Они приведены в комментариях:

```
вывод нс, "Введите предложение "
ввод предл
начало := 1
нц пока начало <= длин(предл)
  | Получаем очередное слово
  | Если есть ближайший пробел, начиная с позиции начало
  если поз после(начало, " ", предл) <> 0
   то
    | Определяем его номер
    позиц := поз после(начало, " ", предл)
    | Определяем очередное слово
    слово := предл[начало : позиц]
    | Печатаем его
    вывод слово, " "
    | "Переходим" на первую букву следующего слова
    начало := позиц + 1
   иначе | Больше пробелов нет - осталось последнее слово
         | Определяем его
    слово := предл[начало : длин(предл)]
    | и печатаем
    вывод слово
    | Меняем значение начало так,
    | чтобы оператор цикла больше не выполнялся
    начало := длин(предл) + 1
  все
кц
```

Какой вариант решения задачи лучше, решите сами.

Задача для самостоятельной работы

Дан набор слов, отделенных друг от друга запятой и пробелом. Вывести на экран все слова.

Литература

1. Основные операторы языков программирования // «Мир информатики», выпуск № 45 (май 2020 г.). https://infojournal.ru/2020/04/25/v-45/

Древние языки программирования набирают популярность

Язык программирования Fortran, созданный в 1957 г., переживает резкий всплеск популярности. В рейтинге TIOBE он еще летом 2020 г. занимал самое последнее 50 место, но к апрелю 2021 г. оказался на 20 строчке. Несмотря на почтенный возраст, Fortran продолжает развиваться и использоваться в различных сферах – его самая актуальная версия вышла в конце 2018 г.

Живее всех живых

Язык программирования Fortran после многих лет забвения вновь стал интересовать разработчиков. В рейтинге языков TIOBE за апрель 2021 г. он вошел в топ-20, опередив многие известные языки, включая Objective-C и Rust (см. рис. 1).

Apr 2021	Apr 2020	Change	Programming Language	Ratings	Change
1	2	⌃	C	14.32%	-2.40%
2	1	⌄	Java	11.23%	-5.49%
3	3		Python	11.03%	+1.72%
4	4		C++	7.14%	+0.36%
5	5		C#	4.91%	+0.16%
6	6		Visual Basic	4.55%	-0.18%
7	7		JavaScript	2.44%	+0.06%
8	14	⏫	Assembly language	2.32%	+1.16%
9	8	⌄	PHP	1.84%	-0.54%
10	9	⌄	SQL	1.83%	-0.34%
11	19	⏫	Classic Visual Basic	1.54%	+0.71%
12	22	⏫	Delphi/Object Pascal	1.47%	+0.77%
13	13		Ruby	1.23%	-0.02%
14	12	⌄	Go	1.22%	-0.13%
15	11	⏬	Swift	1.19%	-0.32%
16	10	⏬	R	1.12%	-0.42%
17	48	⏫	Groovy	1.04%	+0.86%
18	16	⌄	Perl	0.99%	+0.03%
19	18	⌄	MATLAB	0.99%	+0.06%
20	34	⏫	Fortran	0.91%	+0.58%

Рис. 1

TIOBE – это один из самых авторитетных рейтингов языков программирования. Он существует с 2003 г., и его развивает одноименная компания (**T**he **I**mportance **O**f **B**eing **E**arnest, отсылка к пьесе Оскара Уайлда «Как важно быть серьезным»). Рейтинг создается на основе результатов поиска информации о тех или иных языках на популярных сайтах, включая «Википедию» и YouTube, а также на основе запросов в поисковиках Google, Bing и др.

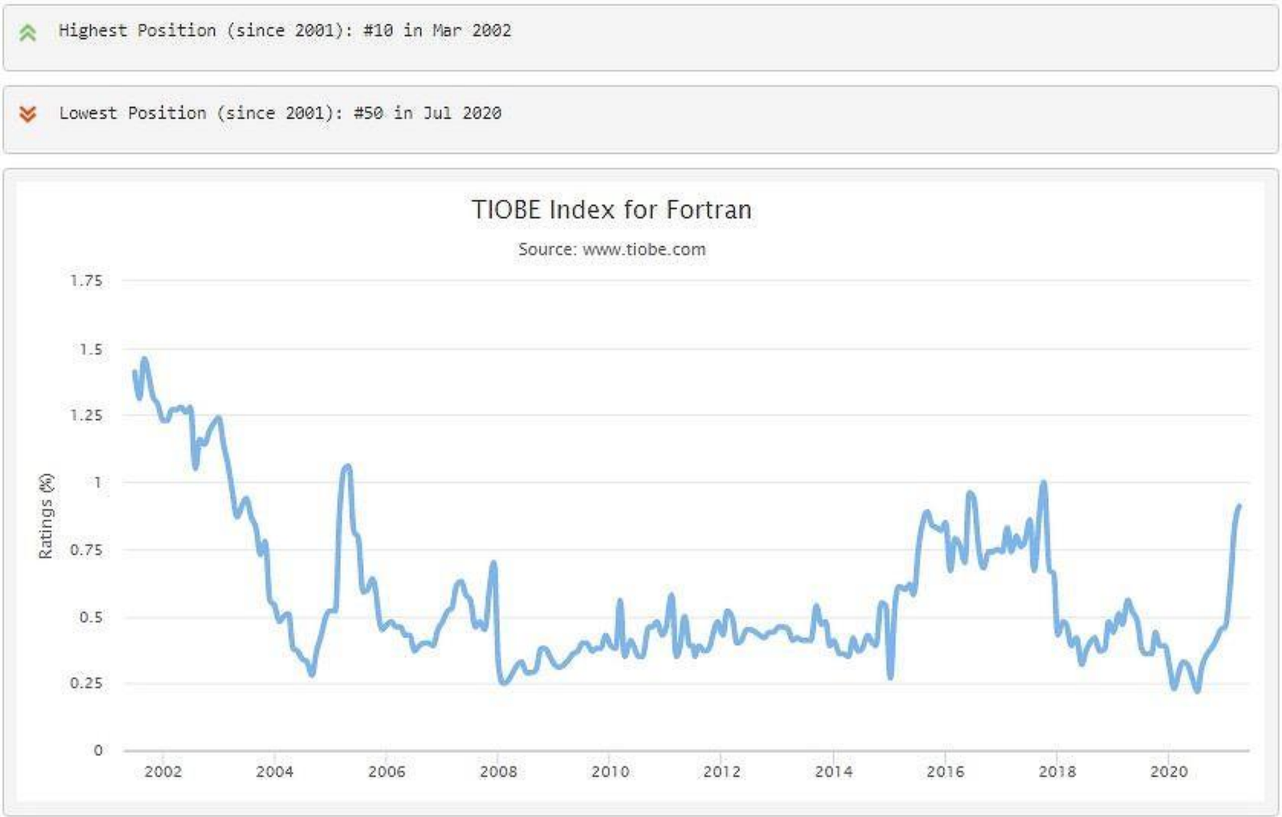
Fortran демонстрирует взрывной рост популярности, пришедший на смену постепенному забвению. Так, если в апреле 2020 года он занимал 34 место, то к июню 2020 года почти выбыл из рейтинга, скатившись на 50 строчку. В итоге менее чем за год Fortran отыграл 30 позиций, но составители рейтинга TIOBE пока не прогнозируют его дальнейшие успехи или, наоборот, неудачи.

За 18 лет существования рейтинга Fortran поднимался в нем максимум до 10 места. Этот личный рекорд он поставил в марте 2002 года, 19 лет назад, после чего его популярность стала падать. В 2005 году был зафиксирован кратковременный рост интереса к нему, но затем вплоть до 2015 года он держался примерно на одном уровне. С 2015 по 2017 гг. популярность Fortran вновь подскочила и затем снова рухнула. Не исключено, что новый рывок этого языка окажется началом следующего длительного периода роста его востребованности.

История Fortran

Fortran входит в число старейших языков программирования – он был создан в 1957 г. специалистами компании IBM. Название 64-летнего языка расшифровывается как FORmula TRANslator (переводчик формул).

Несмотря на свой почтенный возраст, Fortran по-прежнему востребован в ряде сфер, включая инженерные вычисления. Его разработка продолжается, и на момент публикации материала самой актуальной его версией была Fortran 2018 (ISO/IEC 1539-1:2018), вышедшая в конце ноября 2018 г.



Highest Position (since 2001): #10 in Mar 2002

Lowest Position (since 2001): #50 in Jul 2020

TIOBE Index for Fortran

Source: www.tiobe.com

Рис. 2. График изменения популярности Fortran

Широкое распространение Fortran в странах СССР началось спустя 10 лет с момента его создания. Толчком этому послужила разработка первого советского компилятора этого языка для ЭВМ «Минск-2», плюс через год появился компилятор «Фортран-Дубна» для ЭВМ БЭСМ-6.

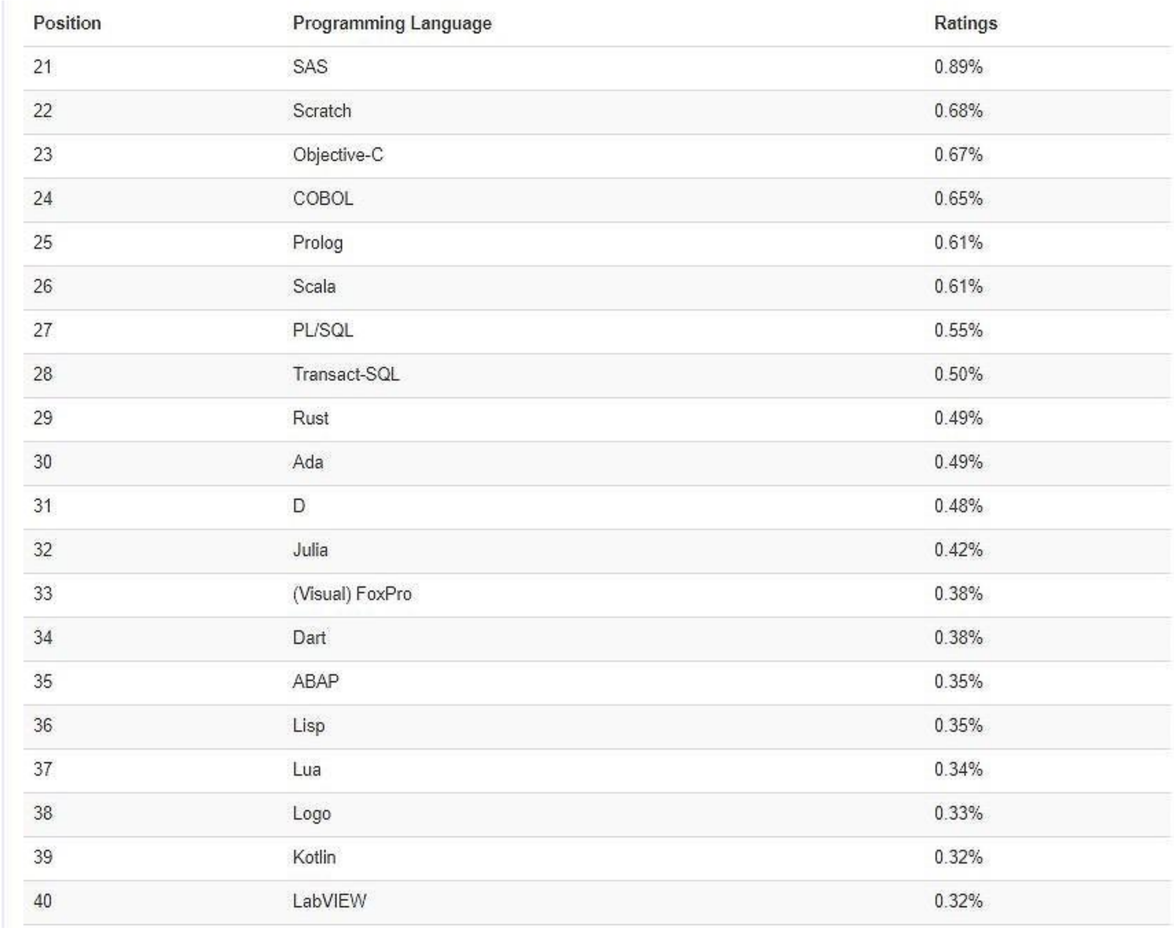
Комментируя новый всплеск популярности Fortran, составители индекса TIOBE заявили: «Этот динозавр вернулся в топ-20 спустя более 10 лет. Fortran был первым коммерческим языком программирования и набирает популярность благодаря огромной потребности в научных вычислениях». «С возвращением Fortran», – добавили они.

Если Fortran сможет приблизиться к топ-10 языков программирования, ему будет непросто пробиться в него. В рейтинге TIOBE первые 10 мест почти не меняются на протяжении нескольких лет, и на апрель 2021 г их занимали, в порядке снижения популярности, следующие языки: C, Java, Python, C++, C#, Visual Basic, JavaScript, Assembly, PHP и SQL.

Конкуренты из прошлого

Fortran – не единственный язык программирования середины XX века, набирающий популярность в третьем десятилетии XXI века. В апреле 2020 г. в мире и, в особенности, в США резко взлетел спрос на разработчиков, пишущих на COBOL (COmmon Business Oriented Language).

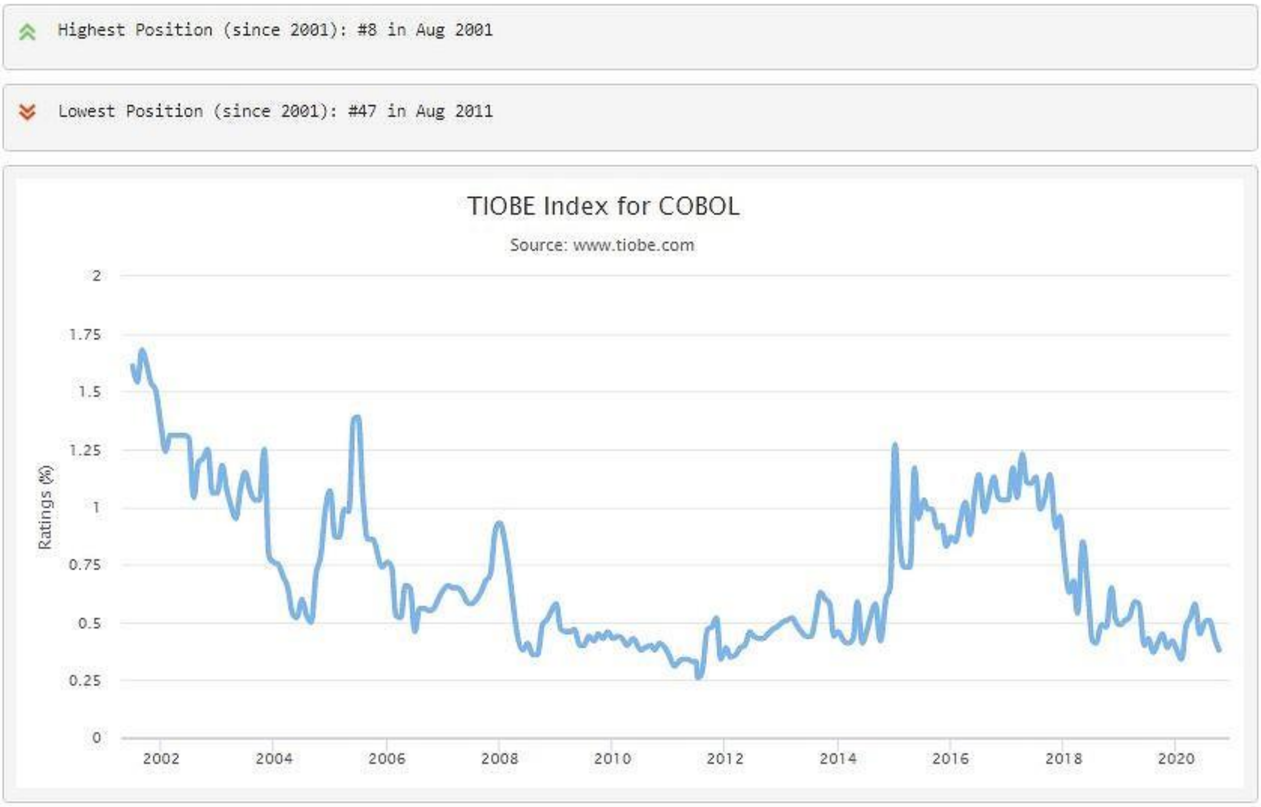
Интерес к COBOL подхлестнула пандемия коронавируса, повлекшая за собой рост числа безработных. ПО, написанное на этом языке, используется в США в системе занятости, и в настоящее время оно требует обновления и оптимизации. COBOL младше Fortran на два года – он появился в 1959 г.



Position	Programming Language	Ratings
21	SAS	0.89%
22	Scratch	0.68%
23	Objective-C	0.67%
24	COBOL	0.65%
25	Prolog	0.61%
26	Scala	0.61%
27	PL/SQL	0.55%
28	Transact-SQL	0.50%
29	Rust	0.49%
30	Ada	0.49%
31	D	0.48%
32	Julia	0.42%
33	(Visual) FoxPro	0.38%
34	Dart	0.38%
35	ABAP	0.35%
36	Lisp	0.35%
37	Lua	0.34%
38	Logo	0.33%
39	Kotlin	0.32%
40	LabVIEW	0.32%

Рис. 3

В середине апреля 2020 г. фирма IBM, создатель Fortran, даже объявила о готовности провести обучающие курсы по языку COBOL, лишь бы увеличить суммарное число программистов во всем мире, знающих этот язык. В конце июля 2020 г. специалисты Института инженеров электротехники и электроники (Institute of Electrical and Electronics Engineers, IEEE) опубликовали рейтинг языков программирования в своем ежемесячном журнале IEEE Spectrum. Он включает 55 позиций, и COBOL занял в нем 43 строчку.



Highest Position (since 2001): #8 in Aug 2001

Lowest Position (since 2001): #47 in Aug 2011

Рис. 4. График изменения популярности COBOL

В рейтинге TIOBE COBOL тоже присутствует – по состоянию на апрель 2021 г. он находился в нем на 24 месте. В этом списке COBOL достигал максимум 8 места – сделать это ему удалось в августе 2011 г. Спустя ровно 11 лет, в августе 2011 г., COBOL оказался на 47 месте, и пока это наихудший его результат за все 18 лет существования рейтинга.

Конечно, может возникнуть вопрос, зачем нужны эти древние языки, когда есть мощные современные системы и языки. Дело в том, что в Fortran’е, например, еще давно были предусмотрены многие средства для решения задач численными методами (типа решения системы дифференциальных уравнений методом Рунге-Кута), и смельчаков, желающих начинать с нуля, найдется мало…

Статья подготовлена по материалам сайта https://www.cnews.ru/news/top/2021-04-07_drevnejshij_yazyk_programmirovaniya

Книжная полка

Книга для подготовки к ЕГЭ по информатике

Издательство ДМК Пресс предлагает книгу Д.М. Златопольского «Подготовка к ЕГЭ по информатике в компьютерной форме».

Книга предназначена для самостоятельной подготовки учащихся к единому государственному экзамену по информатике и ИКТ, который начиная с 2021 года будет проходить в компьютерной форме. Согласно демонстрационному варианту ЕГЭ 2021 года, в содержании экзамена будет существенно увеличено количество заданий, связанных с алгоритмизацией и программированием. Таким задачам в книге уделяется особое внимание. Обсуждаются и методики выполнения ряда других заданий, представленных в демонстрационном варианте.

В приложениях приведены вспомогательные материалы, связанные с заданиями ЕГЭ.

Подробно — см. https://dmkpress.com/catalog/computer/handbooks/978-5-97060-896-8/