

Создание нового языка программирования. Основные принципы.

План лекции:

- основные принципы создания нового языка программирования;
- Bosque – новый язык программирования от Microsoft;
- Carbon – новый язык программирования от Google.

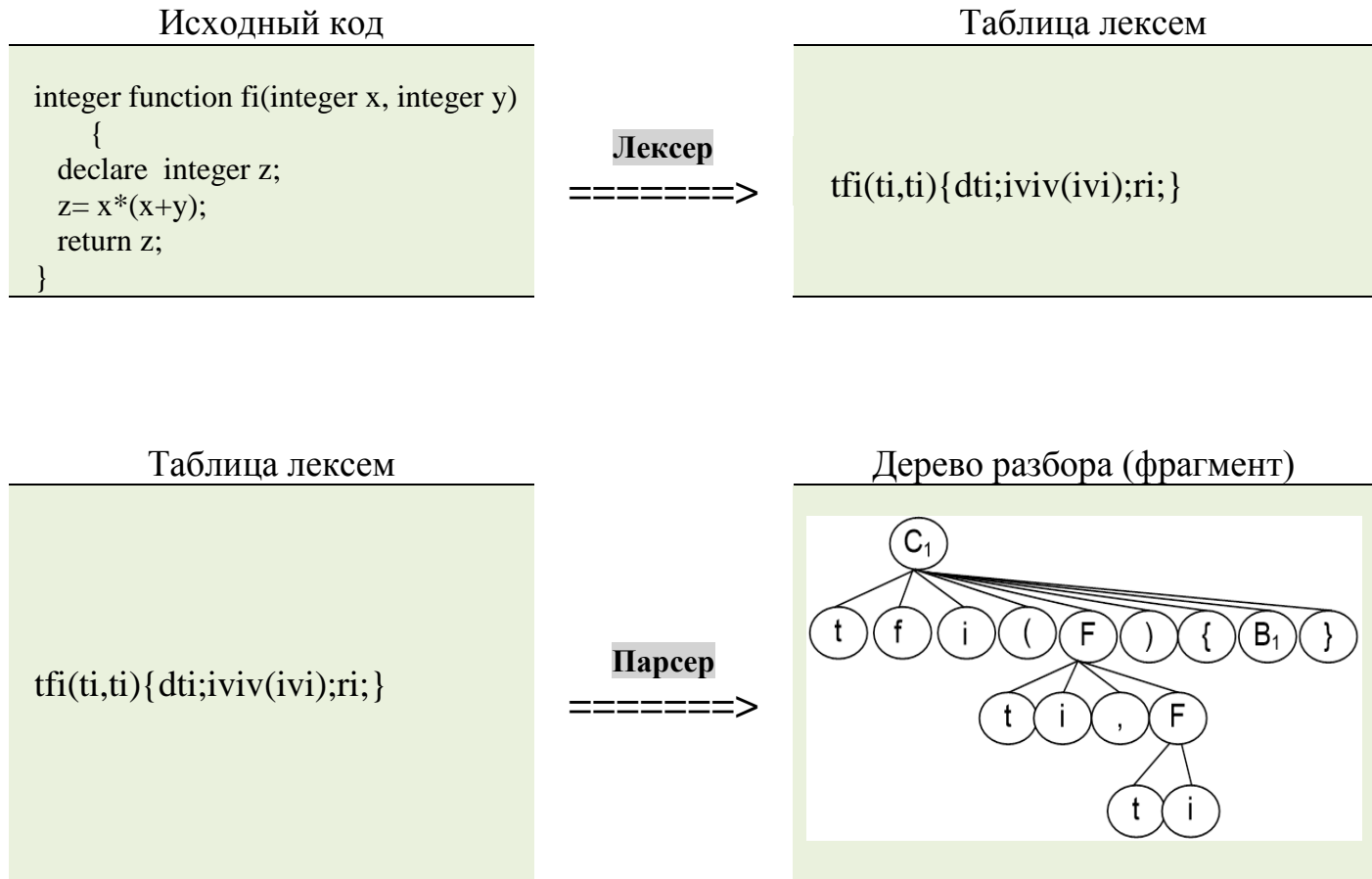
Что необходимо для успеха:

1. Изучить устройство компьютера.
2. Определиться с *назначением языка*.
3. Очертить основные концепции языка.
4. Поэкспериментируйте с синтаксисом.
5. Язык *разработки* для вашего нового языка C/C++.
6. Разработать лексический анализатор (лексер) и синтаксический анализатор (парсер).
7. Выполнить генерацию кода.
8. Подключить стандартную библиотеку.
9. Написать тесты.
10. **Финальная стадия:** успешно защитить курсовой проект.
11. **Дополнительно:** опубликуйте язык.

3. Очертить основные концепции языка:

- *способ реализации языка:* компиляция или интерпретация?
- *система типов:* статическая или динамическая типизация?
- будет ли язык содержать автоматический *сборщик мусора* или ручное управление памятью?
- какую вы планируете использовать модель (*парадигму*) программирования: ООП, логическое, функциональное, структурное, другую?
- содержит ли язык основные базовые функции или все возможности языка будут поддерживаться из внешних библиотек?
- как будет выглядеть архитектура программы?

6. Разработать лексический анализатор (лексер) и синтаксический анализатор (парсер)



Язык программирования

Язык программирования — формальная знаковая система, предназначенная для записи компьютерных программ.

Язык программирования определяет набор лексических, синтаксических и семантических правил, задающих вид программы и действия, которые будут выполняться под её управлением.

Язык программирования представляется в виде набора спецификаций, определяющих его синтаксис и семантику.

Типы данных

Система, по которой данные организуются в программе, – это **система типов** языка программирования.

Языки могут быть классифицированы как системы со статической типизацией и языки с динамической типизацией.

Статически-типизированные языки могут быть в дальнейшем подразделены на языки с **обязательной декларацией**, где каждая переменная и объявление функции имеет обязательное объявление типа, и языки с **выводимыми типами**.

Плюсы статической типизации:

- строгость программы;
- выявление ошибок на этапе компиляции;
- возможность для оптимизации и анализа кода.

Минусы:

- типы надо определять явно (автовывод типов снижает эту проблему).

Структуры данных

Системы типов в языках высокого уровня позволяют определять сложные, составные типы, так называемые структуры данных на основе базовых (атомарных) типов и составных типов.

Парадигма программирования

Язык программирования строится в соответствии с той или иной базовой моделью вычислений и парадигмой программирования.

Способы реализации языков

Языки программирования могут быть реализованы как компилируемые и интерпретируемые.

Разделение на компилируемые и интерпретируемые языки является условным.

Для любого компилируемого языка, можно написать интерпретатор. Большинство современных «чистых» интерпретаторов компилируют исходный код в некоторое высокоуровневое промежуточное представление.

Используемые символы (алфавит языка)

Базовый алфавит для большинства языков программирования использует символы ASCII для записи конструкций языка (*переносимый набор символов*). Управляющие символы ASCII используются ограниченно: допускаются только некоторые из них, например, возврат каретки CR, перевод строки LF, горизонтальная табуляция HT, переход к следующей странице FF.

Для работы с текстовыми данными языки программирования поддерживают Unicode.

Классы языков программирования

- Функциональные
- Процедурные (императивные)
- Стековые
- Аспектно-ориентированные
- Декларативные
- Динамические
- Учебные
- Описания интерфейсов
- Прототипные
- Объектно-ориентированные
- Рефлексивные – поддерживающие отражение
- Логические
- Скриптовые (сценарные)
- Эзотерические

Основная идея КП

Разработать простой язык программирования с некоторой функциональностью.

Влияние синтаксиса на стиль кода:

«Писать на фортране можно на любом языке»

В питоне длинное, тяжеловесное объявление анонимных функций.

Два примера на **Python** с объявлением анонимных функций:

1	<code>filtered_lst = [elem for elem in lst if elem.y > 2]</code>	
2	<code>filtered_lst = list(filter(lambda elem: elem.y > 2, lst))</code>	

Тоже в **Scala**:

```
val filteredLst = lst.filter(_.y > 2)
```

Ничего лишнего.

Грамматика языка

Грамматика языка содержит правила вывода, определяющие структуру цепочек порождаемого языка.

Для задания правил используются различные формы описания языка программирования:

- символическая;
- форма Бэкуса-Наура;
- итерационная форма;
- синтаксические диаграммы.

Удобнее и нагляднее использование расширенных БНФ для описания элементов языка.

Язык программирования – это его идея, спецификация описывающая синтаксис, семантику и стандартную библиотеку.

Реализация языка программирования – программа, которая транслирует код, написанный на исходном языке в код какого-либо другого выходного языка. При этом выходной язык может быть: *ассемблер, байт-код виртуальной машины* или любой *другой язык*.

Сам язык программирования придумать можно, не написав ни строчки кода, например: программа, которая переводит текст вида:

Исходный текст:

```
print('test')
```

транслируется в код:

```
<?php  
echo("test");  
?>
```

Такая программа может считаться примитивным вариантом транслятора.

Пример. Спецификация языка L1

- строка кода начинается с символа «:» (двоеточие);
- строка с комментарием начинается с символа «-» (минус);
- одна инструкция языка размещается на одной строке кода;
- переменные не декларируются заранее;
- тип переменных определяется при первом использовании;
- Типы данных:
 - \$ – строка символов;
 - % – число с плавающей запятой (целая часть отделяется от дробной запятой);
 - @ – целое число со знаком.
- преобразование типов:
 - переменные могут быть преобразованы *временно* в другой тип, если при их последующем использовании они обозначены как другой тип данных;
 - любой тип данных можно преобразовать в другой тип (например, число в строку, или целое число в число с плавающей запятой);
- Оператор > может означать:
 - ввод с клавиатуры, если она является первой после символа «:»;
 - вывод на экран, если она является последней на строке;
 - копирование данных из переменной или константы слева в переменную справа.
- константы пишутся в одинарных кавычках (вне зависимости от типа данных).
- оператор & – это:
 - переписывает строку в обратном порядке символов в ней;
 - для численных переменных меняет их знак;
- оператор # вычисляет длину строки;
- последняя строка программы начинается с символа «^».

Пример программы на языке L1:

	Спецификация
<p>- Это пример программы</p> <p>- вводим строку с клавиатуры :>\$a</p> <p>- выполняем реверс строки :&a</p> <p>- выводим строку на экран :a></p> <p>- записываем строку из двух пробелов в переменную s :' '>\$s</p> <p>- выводим пробелы на экран :s></p> <p>- записываем длину строки a в целочисленную переменную b :#a>@b</p> <p>- выводим значение b на экран :b></p> <p>- записываем значение 1,8 в переменную с плавающей запятой :'1,8'>%c</p> <p>- выводим переменную с на экран :c></p> <p>- выводим на экран переменную, преобразованную в целое число :@c></p> <p>- конец программы ^</p>	<p>комментарий начинается с «-»</p> <p>код начинается с «:»</p> <p>оператор &: переписывает строку в обратном порядке; для численных переменных меняет их знак;</p> <p>оператор > выводит строку на экран</p> <p>оператор > копирует данные из переменной или константы слева в переменную справа</p> <p>оператор > выводит 2 пробела на экран</p> <p>оператор # вычисляет длину строки; оператор > копирует длину строки в b (@целое)</p> <p>оператор > копирует; % – число с плавающей запятой</p> <p>преобразование типов</p> <p>^ последняя строка кода.</p>

Курсовой проект

Минимальный синтаксис языка:

- два типа данных;
- наличие нескольких связанных программных блоков (функций, процедур и пр.);
- наличие не менее 2-х функций стандартной библиотеки;
- поддержка выражений;
- использование функций в выражениях;
- наличие оператора вывода;
- использование одного управляющего оператора.

Объявление переменной.

Что делает данный код?

x = 2;

- 1) Изменяет значение переменной, объявленной где-то раньше?
- 2) Вводит новую переменную?
- 3) Если эта переменная уже объявлена раньше, и мы просто изменяем ее значение, то тогда, где именно она объявлена?

Возможные варианты:

Просматриваем код выше, находим оператор:

x = 0;

Что именно делает данный код:

- 1) Он действительно вводит новую переменную?
- 2) А может, просто изменяет переменную, объявленную ранее?

Как это определить?

Когда мы объявляем переменную явно, мы можем указать дополнительную информацию об этой переменной. Например, тип переменной, что у переменной должна быть некоторая область видимости – т.е., что к ней нельзя обратиться из другого модуля или программного блока.

Область видимости имен

Классическим языком с точки зрения определения области видимости является Си. В Си область видимости задают не только функции, но и другие конструкции языка.

Выражения

Правила записи выражений определяют:

- какие операции допустимы для типов операндов, входящих в выражение;
- допустимость и правило преобразования типов операндов;
- тип результата после выполнения каждого подвыражения;
- порядок выполнения операций;
- явное изменение порядка выполнения операций;
- порядок выполнения операций одного приоритета.

Операторы

Операторы предназначены для осуществления действий и для управления ходом выполнения программы.

В C/C++:

- условные операторы (if, swich)
- операторы цикла (while, for, do while)
- операторы перехода
- метки (case, default)
- операторы-выражения (состоят из выражений)
- блоки (фрагмент кода, заключенный в {}).

Как правило в языках программирования имеются следующие соглашения по операторам:

- о разделителях операторов;
- о терминаторах операторов;
- о продолжении строки

Разделитель операторов используется для определения границы между двумя отдельными операторами.

Терминатор операторов используется для маркировки конца отдельного оператора.

Языки, которые интерпретируют конец строки, как конец оператора называются однострочными языками программирования.

Операторы	C#	Python	Swift
Условные	if (условие) { инструкции } [else{ инструкции }]	if логическое_выражение: инструкции [elif логическое выражение: инструкции] [else: инструкции]	if условие { инструкции } [else{ инструкции }]
	switch (условие) { case метка: инструкции break; ... default: инструкции break; }		[первый_операнд - условие] ? [второй_операнд] : [третий_операнд]
	[первый_операнд - условие] ? [второй_операнд] : [третий_операнд];		switch условие { case метка: инструкции ... default: инструкции }

Операторы	C#	Python	Swift
Циклы	for ([инициализация_счетчика] ; [условие]; [изменение_счетчика]) { инструкции } <hr/> do { инструкции } while (условие); <hr/> while (условие) { инструкции } <hr/> foreach (тип имя_переменной in коллекция) { инструкции }	while условие_выражение: инструкции <hr/> for int_var in функция_range: инструкции	for объект_последовательности in последовательность { инструкции } <hr/> while условие { инструкции } <hr/> repeat { инструкции } while условие
Выход из цикла	break continue	break continue	break continue
Безусловный переход	goto	нет	нет

Операторы	C#	Python	Swift
Метки	Идентификатор:	нет	Идентификатор:
Выражения	Обрабатываются слева направо, состав, синтаксис, учитывается приоритет и ассоциативность.	Обрабатываются слева направо, состав, синтаксис, учитывается приоритет и ассоциативность.	<p>Виды выражений: префиксные, бинарные, первичные и постфиксные выражения.</p> <p>Синтаксис бинарного выражения: левый_аргумент оператор правый_аргумент</p> <p>(во время парсинга, выражение из бинарных операторов выглядит как простой список. Затем этот список превращается в дерево в соответствии с порядком выполнения операций.)</p>
Блоки	{ }	Основная_инструкция: Вложенный_блок_инструкций	{ инструкции }

Инструкции

Инструкция — это некоторое элементарное действие, например:

выражение

Синтаксис конструкций проектируемого языка программирования

Оператор цикла.

Можно выбрать синтаксис, подобный `for`, или `while` в C/C++.

Можно выбрать синтаксис, подобный `foreach` в C#.

Можно выбрать другое ключевое имя, например `repeat`, и предложить свой синтаксис.

Важный вопрос: компилируемый или интерпретируемый?

Компилятор анализирует программу целиком, превращает её в машинный код и сохраняет для последующего выполнения.

Интерпретатор разбирает и выполняет программу построчно в режиме реального времени.

Вычислительная система, для которой выполняется компиляция, называется *целевой вычислительной системой*. В это понятие входит архитектура аппаратных средств компьютера, операционная система, набор динамически подключаемых библиотек.

Общая схема работы компилятора

Вход: исходная программа в виде последовательности символов входного языка (“цепочку”).

Компилятор проверяет, принадлежит ли она входному языку;
определяет набор правил, по которым эта последовательность строится.

Выход: объектный код программы, сформированной компилятором.

Задача

Для целых чисел от 1 до 100 (включительно) вывести:

- строку **Fizz**, если число кратно трем;
- строку **Buzz**, если число кратно пяти;
- строку **FizzBuzz**, если число кратно и трем и пяти;
- в остальных случаях вывести само число.

Примеры реализации на различных языках программирования:

АВАР (v. 7.4 SP05) - язык используется для бизнес-приложений и промежуточного ПО в среде **SAP** (компания **SAP**), 1983г. Синтаксис близок к синтаксису языка COBOL.

Система типов строгая, статическая, безопасная. Поддерживает ООП, имеет сборщик мусора, компилируется в исполняемый АВАР байт-код. Исполняется на виртуальной машине.

Язык реализует работу с внутренними структурами данных, интерфейсом пользователя, транзакциями, отчётами, интерфейсами загрузки и выгрузки данных. Имеет свою собственную среду разработки.

```
cl_demo_output=>display( VALUE stringtab( FOR i = 1 WHILE  
i <= 100 ( COND #( LET m3 = i MOD 3 m5 = i MOD 5 IN  
    WHEN m3 = 0 AND m5 = 0 THEN |FIZZBUZZ|  
    WHEN m3 = 0          THEN |FIZZ|  
    WHEN m5 = 0          THEN |BUZZ|  
    ELSE i ) ) ) ).
```

Ключевые слова и переменные регистронезависимы
е
разделитель операторов
– '.'
слова выделяются пробелами с обеих сторон
Управление потоком:
COND – учитывать несколько условий

АДА – язык для систем управления автоматизированными комплексами, функционирующими в реальном времени, 1980г. Синтаксис близок к синтаксису языка Алгол.

Система типов строгая, статическая, безопасная. Поддерживает ООП, средства параллельного программирования, имеет структурную обработку ошибок, компилируемый.

Основное требование – надёжность, максимально лёгкая читаемость текстов программ.

```
with Ada.Text_IO; use Ada.Text_IO;  
  
procedure Fizzbuzz is  
begin  
    for I in 1..100 loop  
        if I mod 15 = 0 then  
            Put_Line("FizzBuzz");  
        elsif I mod 5 = 0 then  
            Put_Line("Buzz");  
        elsif I mod 3 = 0 then  
            Put_Line("Fizz");  
        else  
            Put_Line(Integer'Image(I));  
        end if;  
    end loop;  
end Fizzbuzz;
```

Программы – модульные, механизм контроля импорта-экспорта описаний между модулями включает две директивы: одну для подключения другого модуля (with) и другую для импорта его описаний (use).
Использует операторные скобки begin-end.
Завершающие ключевые слова для разных управляющих конструкций различны: условный оператор if заканчивается комбинацией end if, циклы – комбинацией end loop, оператор множественного выбора – end case и т. д.
В процедурах и функциях поддерживаются входные и выходные параметры, передача параметров по имени, параметры со значениями по умолчанию.
Поддерживается переопределение процедур, функций и операторов.

ALGOL 68 – универсальный язык для описания произвольных алгоритмов обработки данных высокой сложности, 1968г. Синтаксис унаследован от языков ALGOL Y и ALGOL 60.

Алгол 68 является процедурным языком, ориентированным на описание последовательности команд, имеет развитые средства описания типов и операций, может быть использован для написания программ практически в любом стиле, например, в функциональном стиле. В языке есть возможность использования различных таблиц трансляции, что позволяет для каждого естественного языка определить свой набор ключевых слов Алгола-68, может быть расширен путем переопределения синтаксиса и операторов, что позволяет программистам создавать собственные операции.

```
main:(  
  FOR i TO 100 DO  
    printf(($gl$,  
      IF i %* 15 = 0 THEN  
        "FizzBuzz"  
      ELIF i %* 3 = 0 THEN  
        "Fizz"  
      ELIF i %* 5 = 0 THEN  
        "Buzz"  
      ELSE  
        i  
      FI  
    ))  
  OD  
)
```

Программы – процедуры

Babel – компилятор (транспайлер) с открытым исходным кодом, конфигурируемый transpiler в JavaScript используется для веб-разработки, Babel 6 выпущен в 2015 году. Популярный инструмент для использования новейших возможностей языка программирования JavaScript (ES6).

```
main:  
  { { iter 1 + dup  
    15 %  
    { "FizzBuzz" <<  
      zap }  
    { dup  
    3 %  
    { "Fizz" <<  
      zap }  
    { dup  
    5 %  
    { "Buzz" <<  
      zap}  
    { %d << }  
    if }  
    if }  
    if  
    "\n" << }  
  100 times }
```

Bash – популярная командная оболочка UNIX. В среде Linux используется в качестве предустановленной командной оболочки.

Это командный процессор, работающий в интерактивном режиме в текстовом окне.

Bash может читать команды из файла, который называется скриптом (или сценарием).

```
for n in {1..100}; do
    ((( n % 15 == 0 )) && echo 'FizzBuzz') ||
    ((( n % 5 == 0 )) && echo 'Buzz') ||
    ((( n % 3 == 0 )) && echo 'Fizz') ||
    echo $n;
done
```

C – компилируемый процедурный язык общего назначения, 1972г. Система типов статическая слабая. Разрабатывался как язык системного программирования.

```
#include <stdio.h>

main() {
    int i = 1;
    while(i <= 100) {
        if(i % 15 == 0)
            puts("FizzBuzz");
        else if(i % 3 == 0)
            puts("Fizz");
        else if(i % 5 == 0)
            puts("Buzz");
        else
            printf ("%d\n", i);
        i++;
    }
}
```

Неявный int main
возвращает 0 (C99 +)
Функция puts выводит строку, на которую указывает параметр

C++ – компилируемый, статически типизированный язык общего назначения, 1983г. Поддерживает парадигмы: процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, функциональное программирование. Область применения: создание операционных систем, прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, игр. Синтаксис C-подобный. Система типов статическая.

```
#include <iostream>

using namespace std;
int main ()
{
    for (int i = 1; i <= 100; i++)
    {
        if ((i % 15) == 0)
            cout << "FizzBuzz\n";
        else if ((i % 3) == 0)
            cout << "Fizz\n";
        else if ((i % 5) == 0)
            cout << "Buzz\n";
        else
            cout << i << "\n";
    }
    return 0;
}
```

<pre>// C++11 #include <iostream> #include <algorithm> #include <vector> int main() { std::vector<int> range(100); std::iota(range.begin(), range.end(), 1); std::vector<std::string> values; values.resize(range.size()); auto fizzbuzz = [](int i) -> std::string { if ((i%15) == 0) return "FizzBuzz"; if ((i%5) == 0) return "Buzz"; if ((i%3) == 0) return "Fizz"; return std::to_string(i); }; std::transform(range.begin(), range.end(), values.begin(), fizzbuzz); for (auto& str: values) std::cout << str << std::endl; return 0; }</pre>	
---	--

<p>C# – компилируется в машинно-независимый код низкого уровня (байт-код), исполняется средой Common Language Runtime, 2000г. Поддерживает парадигмы: объектно-ориентированный, обобщённый, процедурный, функциональный, событийный, рефлексивный. Синтаксис C-подобный. Система типов статическая, динамическая, строгая, есть вывод типов.</p>	
<p>Написать самостоятельно ☺</p> <p>+ вариант с использованием делегатов</p>	

COBOL – компилируемый язык для разработки бизнес-приложений, 1959г. Поддерживает парадигмы: объектно-ориентированную, процедурную. Система типов статическая, строгая.

<pre> * FIZZBUZZ.COB * cobc -x -g FIZZBUZZ.COB * IDENTIFICATION DIVISION. PROGRAM-ID. fizzbuzz. DATA DIVISION. WORKING-STORAGE SECTION. 01 CNT PIC 9(03) VALUE 1. 01 REM PIC 9(03) VALUE 0. 01 QUOTIENT PIC 9(03) VALUE 0. PROCEDURE DIVISION. * PERFORM UNTIL CNT > 100 DIVIDE 15 INTO CNT GIVING QUOTIENT REMAINDER REM IF REM = 0 THEN DISPLAY "FizzBuzz " WITH NO ADVANCING ELSE DIVIDE 3 INTO CNT GIVING QUOTIENT REMAINDER REM IF REM = 0 THEN DISPLAY "Fizz " WITH NO ADVANCING ELSE DIVIDE 5 INTO CNT GIVING QUOTIENT REMAINDER REM IF REM = 0 THEN DISPLAY "Buzz " WITH NO ADVANCING ELSE DISPLAY CNT " " WITH NO ADVANCING END-IF END-IF END-IF ADD 1 TO CNT END-PERFORM DISPLAY "" STOP RUN. </pre>	<p>* комментарий * описание программы</p> <p>описания переменных</p> <p>начало процедуры</p> <p>цикл</p> <p>Конец цикла</p>
---	---

По горизонтали:

Строка COBOL программы состоит из 80 символов.

Символ 1-6: номер строки (необязателен)

Символ 7: “индикатор”:

- * - строка комментариев,
- — - строка “продолжение”,
- D - строка debug.

Символ 8 – 11: Зона для DIVISION'ы, SECTION'ы, имен и заголовков параграфов, а также индикаторов и номеров “уровней”

Символ 12-72: Зона кода.

Символ 73-80: Зона комментария. Не обрабатывается компилятором и полностью предоставлена программисту.

По вертикали:

Каждая COBOL программа содержит 4 раздела – DIVISION, они идут в строгом порядке и содержат определенные элементы.

CoffeeScript – язык программирования, транслируемый в JavaScript, 2009г. Парадигмы: объектно-ориентированный, императивный, функциональный, аспектно-ориентированный, прототипный. CoffeeScript добавляет синтаксический сахар для улучшения читаемости кода и уменьшения его размера.

```
for i in [1..100]
  if i % 15 is 0
    console.log "FizzBuzz"
  else if i % 3 is 0
    console.log "Fizz"
  else if i % 5 is 0
    console.log "Buzz"
  else
    console.log i

for i in [1..100]
  console.log \
    if i % 15 is 0
      "FizzBuzz"
    else if i % 3 is 0
      "Fizz"
    else if i % 5 is 0
      "Buzz"
    else
      i

for i in [1..100]
  console.log(['Fizz' if i % 3 is 0] + ['Buzz' if i % 5 is 0] or i)
```

Delphi – императивный, структурированный, объектно-ориентированный, высокоуровневый язык программирования со строгой статической типизацией переменных для написания прикладного программного обеспечения, 1995г.

```
{$APPTYPE CONSOLE}

uses SysUtils;

var
  i: Integer;
begin
  for i := 1 to 100 do
  begin
    if i mod 15 = 0 then
      Writeln('FizzBuzz')
    else if i mod 3 = 0 then
      Writeln('Fizz')
    else if i mod 5 = 0 then
      Writeln('Buzz')
    else
      Writeln(i);
  end;
end.
```

Ela – интерпретируемый функциональный язык, 1995г., с С-подобным синтаксисом. Операционная система: DOS

<pre>open list prt x x % 15 == 0 = "FizzBuzz" x % 3 == 0 = "Fizz" x % 5 == 0 = "Buzz" else = x [1..100] > map prt</pre>	<pre>// открываем и исполняем модуль List (в файле List.ela) // - битовое «Или» // оператор forward pipe « >»</pre>
--	---

F# – компилируемый функциональный язык, в качестве промежуточного языка используется Common Intermediate Language (CIL), 2005г. F#-интерпретатор (fsi) исполняет F#-код интерактивно. Синтаксис похож на синтаксис Haskell, построен на математической нотации.

Парадигмы: функциональное, объектно-ориентированное, обобщённое, императивное программирование. Типизация строгая, неявные преобразования типов полностью отсутствуют, что исключает ошибки, связанные с приведением типов. По умолчанию все значения являются константами. F# допускает переменные, для чего требуется специально пометить значения как изменяемые при помощи слова mutable.

<pre>let fizzbuzz n = match n%3 = 0, n%5 = 0 with true, false -> "fizz" false, true -> "buzz" true, true -> "fizzbuzz" _ -> string n let printFizzbuzz() = [1..100] > List.iter (fizzbuzz >> printfn "%s")</pre>	<pre>// Разбор ведётся с помощью сопоставления с образцом: оператор match // образец «true, true» означает "fizzbuzz" // оператор forward pipe « >»</pre>
--	---

```
[1..100]
|> List.map (fun x ->
    match x with
    | _ when x % 15 = 0 -> "fizzbuzz"
    | _ when x % 5 = 0 -> "buzz"
    | _ when x % 3 = 0 -> "fizz"
    | _ -> x.ToString())
|> List.iter (fun x -> printfn "%s" x)
```

Fortran – компилируемый императивный язык, 1957г. Библиотеки, написанные на Фортране, являются достоянием человечества: они доступны в исходных кодах, хорошо документированы, отлажены, эффективны.

Парадигмы: процедурный, с элементами объектно-ориентированного программирования. Система типов строгая, статическая.

```
program fizzbuzz_if
  integer :: i

  do i = 1, 100
    if (mod(i,15) == 0) then; print *, 'FizzBuzz'
    else if (mod(i,3) == 0) then; print *, 'Fizz'
    else if (mod(i,5) == 0) then; print *, 'Buzz'
    else;
      print *, i
    end if
  end do
end program fizzbuzz_if
```

Haskell– стандартизированный чистый функциональный язык программирования общего назначения, 1990г.

Парадигмы: модульный, функциональный. Система типов сильная, статическая, с автоматическим выводом типов.

```
fizzbuzz :: Int -> String
fizzbuzz x
| f 15 = "FizzBuzz"
| f 3  = "Fizz"
| f 5  = "Buzz"
| otherwise = show x
where
  f = (0 ==) . rem x

main :: IO ()
main = mapM_ (putStrLn. fizzbuzz) [1 .. 100]
```

Функция `fizzbuzz` имеет тип из целого в строковый
Тело: определение, описывает процесс вычисления на основе образца

«.» – оператор композиции функций

`map` – функция высшего порядка применяет определенную функцию к каждому элементу списка, возвращая список результатов.

```
fizzbuzz :: Int -> String
fizzbuzz n =
  '\n' :
  if null (fizz ++ buzz)
  then show n
  else fizz ++ buzz
where
  fizz =
    if mod n 3 == 0
    then "Fizz"
    else ""
  buzz =
    if mod n 5 == 0
    then "Buzz"
    else ""

main :: IO ()
main = putStr $ concatMap fizzbuzz [1 .. 100]
```

Java – транслируются в байт-код, работает на любой компьютерной архитектуре с помощью виртуальной Java-машины, 1995г.

Парадигмы: объектно-ориентированное программирование. Система типов сильная, статическая, с автоматическим выводом типов.

```
class FizzBuzz {

    public static void main(String[] args) {

        for (int i = 1; i < 101; i++) {
            if ((i % 3 == 0) && (i % 5 == 0)) {
                System.out.print("'fizz buzz', ");
            } else if (i % 3 == 0) {
                System.out.print("'fizz', ");
            } else if (i % 5 == 0) {
                System.out.print("'buzz', ");
            } else {
                System.out.printf("%d, ", i);
            }
        }
    }
}
```

JavaScript– интерпретируемый, 1995г.

Парадигмы: объектно-ориентированное (прототипное), событийно-ориентированное, аспектно-ориентированное. Система типов динамическая.

```
var fizzBuzz = function () {
    var i, output;
    for (i = 1; i < 101; i += 1) {
        output = '';
        if (!(i % 3)) { output += 'Fizz'; }
        if (!(i % 5)) { output += 'Buzz'; }
        console.log(output || i); //empty string is false, so we short-circuit
    }
};
```

Kotlin – интерпретируемый, работающий поверх JVM, 2016г.

Парадигмы: объектно-ориентированное программирование. Система типов статическая.

```
fun fizzBuzz() {
    for (i in 1..100) {
        when {
            i % 15 == 0 -> println("FizzBuzz")
            i % 3 == 0 -> println("Fizz")
            i % 5 == 0 -> println("Buzz")
            else -> println(i)
        }
    }
}
```

Императивное
решение

```
fun fizzBuzz1() {
    fun fizzBuzz(x: Int) = if (x % 15 == 0) "FizzBuzz" else x.toString()
    fun fizz(x: Any) = if (x is Int && x % 3 == 0) "Buzz" else x
    fun buzz(x: Any) = if (x is Int && x.toInt() % 5 == 0) "Fizz" else x

    (1..100).map { fizzBuzz(it) }.map { fizz(it) }.map { buzz(it) }
}.forEach { println(it) }
```

Функциональное
решение

Objective-C– компилируемый, основан на C и Smalltalk, 1983г.

Парадигмы: объектно-ориентированное, рефлексивно-ориентированное программирование. Система типов слабая, статическая, динамическая.

```
// FizzBuzz in Objective-C
#import <stdio.h>

main() {
    for (int i=1; i<=100; i++) {
        if (i % 15 == 0) {
            printf("FizzBuzz\n");
        } else if (i % 3 == 0) {
            printf("Fizz\n");
        } else if (i % 5 == 0) {
            printf("Buzz\n");
        } else {
            printf("%i\n", i);
        }
    }
}
```

Pascal– компилируемый, основан на C и Smalltalk, 1970г.

Парадигмы: процедурная. Система типов строгая.

```
program fizzbuzz(output);
var
    i: integer;
begin
    for i := 1 to 100 do
        if i mod 15 = 0 then
            writeln('FizzBuzz')
        else if i mod 3 = 0 then
            writeln('Fizz')
        else if i mod 5 = 0 then
            writeln('Buzz')
        else
            writeln(i)
    end.
end.
```

Perl– интерпретируемый, унаследовал многое от C и AWK, 1987г.

Парадигмы: императивная, объектно-ориентированная, функциональная. Система типов слабая динамическая.

```
use strict;
use warnings;
use feature qw(say);

for my $i (1..100) {
    say $i % 15 == 0 ? "FizzBuzz"
      : $i % 3 == 0 ? "Fizz"
      : $i % 5 == 0 ? "Buzz"
      : $i;
}
```

Или коротко:

```
print 'Fizz'x!($_ % 3) . 'Buzz'x!($_ % 5) || $_, "\n" for 1 .. 100;
```

PHP– интерпретируемый скриптовый язык общего назначения для разработки веб-приложений, 1995г.

Парадигмы: императивный, объектно-ориентированный, сценарный. Система слабая динамическая.

```
<?php
for ($i = 1; $i <= 100; $i++)
{
    if (!($i % 15))
        echo "FizzBuzz\n";
    else if (!($i % 3))
        echo "Fizz\n";
    else if (!($i % 5))
        echo "Buzz\n";
    else
        echo "$i\n";
}
?>
```

PL/I

```
do i = 1 to 100;
    select;
        when (mod(i,15) = 0) put skip list ('FizzBuzz');
        when (mod(i,3) = 0) put skip list ('Fizz');
        when (mod(i,5) = 0) put skip list ('Buzz');
        otherwise put skip list (i);
    end;
end;
```

PL/SQL

```
BEGIN
  FOR i IN 1 .. 100
  LOOP
    CASE
      WHEN MOD(i, 15) = 0 THEN
        DBMS_OUTPUT.put_line('FizzBuzz');
      WHEN MOD(i, 5) = 0 THEN
        DBMS_OUTPUT.put_line('Buzz');
      WHEN MOD(i, 3) = 0 THEN
        DBMS_OUTPUT.put_line('Fizz');
      ELSE
        DBMS_OUTPUT.put_line(i);
      END CASE;
    END LOOP;
  END;
```

Prolog– язык и система логического программирования, основанные на языке предикатов математической логики дизъюнктов Хорна, представляющей собой подмножество логики предикатов первого порядка., 1972г.

```
fizzbuzz(X) :- 0 is X mod 15, write('FizzBuzz').
fizzbuzz(X) :- 0 is X mod 3, write('Fizz').
fizzbuzz(X) :- 0 is X mod 5, write('Buzz').
fizzbuzz(X) :- write(X).

dofizzbuzz :- foreach(between(1, 100, X), (fizzbuzz(X),nl)).
```

Python – компилируемый в байт-код, интерпретируемый, 1991г.

Парадигмы: объектно-ориентированный, рефлексивный, императивный, функциональный, аспектно-ориентированный, динамический. 1972г. Система типов сильная динамическая.

```
for i in range(1, 101):
    if i % 15 == 0:
        print ("FizzBuzz")
    elif i % 3 == 0:
        print ("Fizz")
    elif i % 5 == 0:
        print ("Buzz")
    else:
        print (i)
```

Ruby – интерпретируемый, 1995г.

Парадигмы: объектно-ориентированный. Система типов строгая динамическая.

```
1.upto(100) do |n|
  print "Fizz" if a = (n % 3).zero?
  print "Buzz" if b = (n % 5).zero?
  print n unless (a || b)
  puts
end
```

Scala– компилируемый в байт-код 2004г.

Парадигмы: объектно-ориентированный, функциональный. Система типов статическая с автовыведением.

```
object FizzBuzz extends App {
  1 to 100 foreach { n =>
    println((n % 3, n % 5) match {
      case (0, 0) => "FizzBuzz"
      case (0, _) => "Fizz"
      case (_, 0) => "Buzz"
      case _ => n
    })
  }
}
```

Go (часто также golang) – компилируемый многопоточный язык программирования, разработанный компанией Google, 10 ноября 2009.

Назначение: для создания высокоэффективных программ, работающих на современных распределённых системах и многоядерных процессорах. Поддерживает функциональную парадигму.

Go – язык со строгой статической типизацией, доступен автоматический вывод типов, для пользовательских типов – «утиная типизация», поддерживается полноценная поддержка указателей, но без возможности применять к ним арифметические операции, в отличие от C/C++/D.

Была попытка создать замену языкам Си и C++

```
func StartTimer (name string) func(){
    t := time.Now()
    log.Println(name, "started")
    return func() {
        d := time.Now().Sub(t)
        log.Println(name, "took", d)
    }
}

func RunTimer() {
    stop := StartTimer("My timer")
    defer stop()
    time.Sleep(1 * time.Second)
}
```

Rockstar

Midnight takes your heart and your
soul
While your heart is as high as your
soul
Put your heart without your soul into
your heart

Give back your heart

Desire is a lovestruck ladykiller
My world is nothing
Fire is ice
Hate is water
Until my world is Desire,
Build my world up
If Midnight taking my world, Fire is
nothing and Midnight taking my world,
Hate is nothing
Shout "FizzBuzz!"
Take it to the top

If Midnight taking my world, Fire is
nothing
Shout "Fizz!"
Take it to the top

If Midnight taking my world, Hate is
nothing
Say "Buzz!"
Take it to the top

Whisper my world

Rockstar

Полночь захватывает ваше сердце и вашу душу
Пока твое сердце так же высоко, как и твоя
душа
Положи свое сердце без души в свое сердце

Верни свое сердце

Желание - страдающая от любви божья коровка
Мой мир ничто
Огонь - это лед
Ненависть это вода
Пока мой мир не будет Желанием,
Построй мой мир
Если полночь захватывает мой мир, Огонь -
ничто, а полночь - мой мир, Ненависть - ничто
Крик "FizzBuzz!"
Возьми это наверх

Если полночь захватывает мой мир, Огонь -
ничто
Крик "Fizz!"
Возьми это наверх

Если Полночь захватывает мой мир, Ненависть -
ничто
Скажи "Buzz!"
Возьми это наверх

Шепот мой мир

Язык Bosque — новый язык программирования от Microsoft

В середине апреля 2019 года Microsoft представила новый язык программирования, который получил название Bosque (разработчик Марк Баррон (Mark Barron)). Он распространяется с открытым исходным кодом и предназначен для того, чтобы написанный код был простым и понятным как для человека, так и для компьютера.

Новый язык, чьё название с испанского переводится как «лес», призван быть как можно более простым для понимания и помочь избежать сложностей при разработке и написании кода. Однако отмечается, что язык экспериментальный и пока не готов к широкому использованию.

В основу Bosque легли типы и синтаксис TypeScript, а семантика позаимствована из ML и Node/JavaScript. Также в нём отсутствуют циклы и ограничена рекурсия. Разработал Bosque информатик Марк Маррон.

Главная миссия дизайна языка — чтобы он был прост и понятен как для человека, так и для компьютера. Подробно об особенностях нового языка программирования можно прочитать в [документации](#) Microsoft.

1) Все значения в Bosque являются неизменяемыми (immutable).

Но при этом можно объявить изменяемую переменную ключевым словом **var**!

2) В языке нет циклов `for`, `while` и т.д. Вместо этого есть коллекции и конвейеры (пайплайны). Другими словами, вместо циклов нужно использовать `map`, `filter` и т.д. Используются функциональные объекты (Functors), которые выполняют роль циклов и могут повысить качество работы ПО.

3) Строки можно делать разных типов. Т.е., например, можно сделать строку-имя или строку-zipcode, и для type-чекера это будут две разные строки. Если вы в аргументе функции ожидаете `zipcode`, а вам по ошибке туда положат имя, то компилятор это не проглотит. Синтаксис такой: `String[Zipcode]`.

4) Вызов функций можно делать с указанием названия аргументов из сигнатуры функции, например: **`myfunc (x=1, y=2)`**

5) В стандартной библиотеке есть различные коллекции, и с коллекциями можно работать по-разному. Можно просто по цепочке вызывать `map`, потом `filter` и т.д., а можно работать через конвейеры.

Пример кода:

```
var v: List[Int?] = List@{1, 2, none, 4};

//Chained - List@{1, 4, 16}
v->filter(fn(x) => x != none)->map[Int](fn(x) => x*x)

//Piped none filter - List@{1, 4, 16}
v |> filter(fn(x) => x != none) |> map[Int](fn(x) => x*x)

//Piped with noneable filter - List@{1, 4, 16}
v |??> map[Int](fn(x) => x*x)

//Piped with none to result - List@{1, 4, none, 16}
v |?> map[Int](fn(x) => x*x)
```

6) рекурсия считается злом, которое может усложнить программу, поэтому рекурсивные функции надо помечать словом **rec**

7) Программы на Bosque являются детерминированными: в языке нет неопределенного поведения. Например, нельзя использовать переменные, пока они не были определены; алгоритмы сортировки только стабильные и т.д. Если программа выдала какой-то результат, то такой же результат будет и после повторного вызова.

8) Вместо классов и интерфейсов в языке есть понятия **entity** и **concept**.

```
concept Bar {
    field f: Int;
}

entity Baz provides Bar {
    field g: Int;
    field h: Bool = true;
}

//Create a Baz entity with the given field values
var y = Baz@{f=1, g=2, h=false};
//Create a Baz entity with default value for h
var x = Baz@{f=1, g=2};
```

Язык Carbon — новый язык программирования от Google

В июле 2022 года инженер Google Чендлер Каррут впервые представил язык **Carbon** — экспериментальный язык программирования общего назначения, созданный компанией Google, как «*преемник C++*». Презентация прошла на конференции Cpp North в Торонто (Канада). Чендлер Каррут называет Carbon не заменой, но преемником C++ (разработанный в 1982 году и выпущенный в 1985 году).

Программисты на C++, желающие полностью перейти на Carbon, получают в свое распоряжение инструментарий для автоматической транслитерации библиотек C++ в код

на новом языке Google. Обратная миграция тоже возможна – в дальнейшем эти библиотеки могут использоваться в существующем проекте на C++.

Все необходимые разработчику инструменты Carbon размещены на принадлежащем Microsoft портале GitHub и распространяются по лицензии Apache 2.0. Компилятор кода Carbon написан при помощи LLVM (Low Level Virtual Machine) – специальной программной инфраструктуры для создания компиляторов. Также в нем использовались наработки из Clang – компилятора для C, C++, Objective-C и Objective-C++.

Программа «Hello, World!», написанная на языке Carbon:

```
package Sample api;

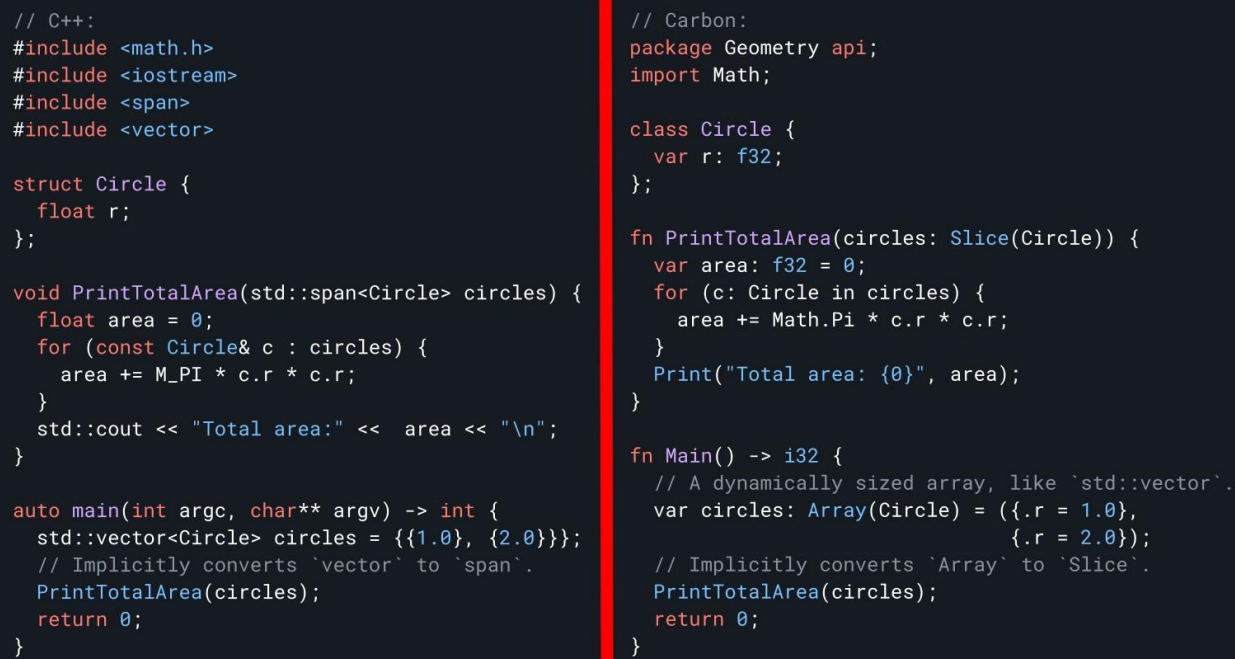
fn Main() -> i32 {
    Print("Hello, World!");
    return 0;
}
```

В настоящее время Carbon находится на экспериментальной стадии. Дорожная карта:

Выпуск основной рабочей версии 0.1 к концу 2022 г.

версии 0.2 в 2023 г.

Релиз 1.0 в 2024–2025 гг.

A side-by-side comparison of C++ and Carbon code. The C++ code on the left uses standard library types like `std::vector` and `std::cout`. The Carbon code on the right uses its own types like `Array` and `Slice`. Both programs define a `Circle` struct/class with a radius `r` and a function `PrintTotalArea` that iterates over a collection of circles and calculates their total area. The Carbon code is more concise, using `Print` instead of `cout` and `return` instead of `return 0`.

```
// C++:
#include <math.h>
#include <iostream>
#include <span>
#include <vector>

struct Circle {
    float r;
};

void PrintTotalArea(std::span<Circle> circles) {
    float area = 0;
    for (const Circle& c : circles) {
        area += M_PI * c.r * c.r;
    }
    std::cout << "Total area:" << area << "\n";
}

auto main(int argc, char** argv) -> int {
    std::vector<Circle> circles = {{1.0}, {2.0}};
    // Implicitly converts `vector` to `span`.
    PrintTotalArea(circles);
    return 0;
}

// Carbon:
package Geometry api;
import Math;

class Circle {
    var r: f32;
};

fn PrintTotalArea(circles: Slice(Circle)) {
    var area: f32 = 0;
    for (c: Circle in circles) {
        area += Math.Pi * c.r * c.r;
    }
    Print("Total area: {0}", area);
}

fn Main() -> i32 {
    // A dynamically sized array, like `std::vector`.
    var circles: Array(Circle) = ({.r = 1.0},
                                   {.r = 2.0});
    // Implicitly converts `Array` to `Slice`.
    PrintTotalArea(circles);
    return 0;
}
```

«Конечно, синтаксис Carbon будет проще, чем синтаксис C++. Но я не считаю это важным. Для меня любой синтаксис нормальный, потому что при изучении языка программирования синтаксис я осваиваю в последнюю очередь. Моя позиция такова: если вы решили изучить язык программирования, то синтаксис – последнее, что вы должны учить. Сначала освоите философию. Потому что синтаксис – это как писать на русском или на английском. Вы уже умеете буквы писать? Значит, научитесь. Здесь то же самое. Клавиши нажимать все умеют, разберитесь лучше сначала с системой типов в языке», – мнение эксперта