Лекция 12

БГТУ, ФИТ, ПОИТ, 2 семестр,
Конструирование программного обеспечения

Принципы разработки языков программирования. Сравнение основных языков программирования

1. 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.

Существует более двух с половиной тысяч языков программирования.

Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

2. Типы данных

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

Тип данных определяет:

- внутреннее представление данных в памяти компьютера;
- диапазон значений, которые могут принимать величины этого типа;
- операции и функции, которые можно применять к величинам этого типа.

✓ Языки программирования по типизации разделяют на
- **типизированные** (например: C, Python, Scala, PHP);
- **нетипизированные** (например: язык ассемблера, Forth).

✓ Классификация типизированных языков программирования:
- системы со **статической типизацией** (переменная связывается с типом в момент присваивания значения)
- системы с **динамической типизацией** (переменная связывается с типом в момент объявления).

Примеры:
Статическая типизация: C, Java, C#;
Динамическая типизация: Python, JavaScript, Ruby.

Наиболее распространенными способами реализации ***динамической типизации*** являются:

- таблица указателей на объекты;
- хранение информации об объекте в памяти вместе с самим объектом.

Это позволяет свести операцию определения типа либо к поиску в таблице, либо к просмотру некоторой области памяти, хранимой вместе с объектом.

✓ Типизированные языки могут быть сильно или слабо типизированы (также говорят: строго или нестрого).

Примеры:
Сильная типизация: Java, Python, Haskell, Lisp, Ruby;
Слабая типизация: C, JavaScript, Visual Basic, PHP.

✓ Статически-типизированные языки могут быть подразделены на языки:

- с ***обязательной декларацией***, где каждая переменная и функция имеют обязательное объявление;
- языки с ***выводимыми типами***.

✓ Статически-типизированные языки могут быть с явной или неявной типизацией.

В явно-типизированных языках ***тип новых переменных*** или ***функций,*** или ***их аргументов*** нужно задавать явно.
В языках с неявной типизацией эту задачу выполняет компилятор или интерпретатор.

Примеры:
Явная типизация: C++, D, C#
Неявная типизация: PHP, JavaScript

Язык C имеет статическую слабую явную типизацию.
Язык Python — динамическую сильную неявную типизацию.

3. Примеры определения переменных

C/C++	Java	Pascal/Delphi
int A; char C = 'A'; const double PI = 3.1415926535;	int A; char C = 'A'; final double PI = 3.1415926535;	var A: Integer; var C: char = 'A'; const PI:Double = 3.1415926535;

Примеры явного объявления переменных:

```
var x = 0;            // изменяемая переменная
let y = 0;             // неизменяемая переменная
letprivate z = 0; // локальная переменная, не видна вне процедуры/функции
```

4. Область видимости

Различают два подхода к организации области видимости – ***динамический*** и ***лексический***.

> ***Динамическая область*** видимости означает, что в программе есть одно глобальное пространство, в котором определены все переменные.
>
> ***Лексическая область*** видимости означает, что программе определяются специальные лексические блоки (например, функции, блоки, конструкции языка). Переменные, объявленные внутри таких блоков, видны только в них.

Язык C является примером языка с лексической областью видимости.

Язык JavaScript заимствовал синтаксис из языка C, но лексическую область в нем задают только функции. Неявные объявления позволяют пользоваться и динамической областью видимости.

Пример JavaScript:

```
        function Foo()
{
  if (x) {
    var y = 0;   // переменная y видна за пределами if
  }
  z = y;         // а здесь мы неявно объявляем глобальную переменную z
}
```

5. Структуры данных

Системы типов в языках высокого уровня позволяют определять сложные, составные типы на основе базовых типов и составных типов.

6. Парадигма программирования

Язык программирования строится в соответствии с ***базовой моделью вычислений*** и ***парадигмой программирования*** (пример парадигмы: объектно-ориентированное программирование).

Парадигма программирования — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию).

Основные парадигмы программирования:

- Императивное программирование
- Декларативное программирование
- Структурное программирование
- Функциональное программирование
- Логическое программирование
- Объектно-ориентированное программирование
  - Компонентно-ориентированное программирование
  - Прототипно-ориентированное программирование
  - Агентно-ориентированное программирование

7. Способы реализации языков

Языки программирования могут быть реализованы как ***компилируемые*** и ***интерпретируемые***.

- ✓ ***Компилятор*** анализирует программу целиком, транслирует ее в машинный код и сохраняет для последующего выполнения.
- ✓ ***Интерпретатор*** разбирает и выполняет программу построчно в режиме реального времени.

Разделение на компилируемые и интерпретируемые языки является условным. Для любого компилируемого языка можно написать интерпретатор.

Современные интерпретаторы компилируют исходный код в некоторое промежуточное представление.

Программы на таких языках, как, например Java и C#, компилируется в машинно-независимый код низкого уровня (***байт-код***). Затем байт-код выполняется ***виртуальной машиной***.
Для выполнения байт-кода обычно используется ***интерпретация***.
Отдельные части байт-кода для ускорения работы программы могут быть транслированы в машинный код непосредственно во время выполнения программы по технологии компиляции «***на лету***» (***Just-in-time compilation***, JIT).
Для Java байт-код исполняется виртуальной машиной Java (Java Virtual Machine, JVM).
Для C# — средой выполнения является Common Language Runtime (CLR).

8. Алфавит языка программирования

Алфавит — разрешенный к использованию набор символов, с помощью которого могут быть образованы лексемы данного языка*:*

- идентификаторы;
- ключевые слова;
- знаки операций;
- разделители;
- литералы.

Алфавит большинства современных языков программирования использует символы ASCII для записи конструкций языка.

Для работы с ***текстовыми данными*** языки программирования также поддерживают Unicode.

9. Директивы препроцессора и комментарии

Директивы препроцессора предназначены для предварительной обработки исходного текста программы перед компиляцией.

10. Директивы препроцессора и комментарии

Комментарии (однострочные, многострочные) предназначены для записи пояснений к программе.

11. Типы данных

C++	Pascal	Java*	C#
		byte (1)	sbyte
int (4)	integer	int (4)	int
		long (8)	long
bool (1)	boolean	boolean (1|8)	bool
float (4)	real	float (4)	
double (8)		double (8)	double
char (1)	char		char
wchar_t (2)		char (2)	

* В языке Java – длины и диапазоны значений примитивных типов определяются ***стандартом***, а не реализацией. Такая жёсткая стандартизация была необходима, чтобы сделать язык платформенно-независимым, что является одним из идеологических требований к Java. Все переменные требуют или явного определения, или автоматически заполняются нулями.

12. Инструкции

Инструкция — это некое элементарное действие.

Несколько идущих подряд инструкций образуют последовательность инструкций.

13.Блок вычислений

Инструкции могут быть сгруппированы в специальные вычислительные блоки следующего вида:

Псевдокод	***C/C++, C#, Java***	***Pascal/Delphi***

Начало
  список операторов
Конец
	{
	  список операторов
	}
		begin
		  список операторов
		end

Блоки ограничены двумя разделителями, обозначающими начало и конец вычислительного блока. Вычислительный блок называют составным оператором.

14.Выражения

Выражения не являются самостоятельными конструкциями языка программирования. Чаще всего, они являются операндами оператора присваивания, условного оператора или цикла.

15.Логическое выражение

Логическое выражение – выражение, результатом которого является ***истина*** (true) или ***ложь*** (false). В состав логического выражения также могут входить константы, переменные, функции и скобки. Могут использоваться операции сравнения: “больше”, “меньше”, “равно”; и логические операции: “И” (AND), “ИЛИ” (OR) и “НЕ” (NOT).

Пример логического выражения (*условие существования решения квадратного уравнения):*

Псевдокод	***C/C++,C#, Java***	***Pascal/Delphi, Visual Basic/VB.NET***

(A=0) ИЛИ (C=0) ИЛИ
  ((B^2-4*A*C)>=0)
	(A==0) || (C==0) ||
	  ((B*B-4*A*C)>=0)
		(A=0) or (C=0) or
		  ((B*B-4*A*C)>=0)

16.

17.Основные операторы

Операторы: арифметические, операторы сравнения, присваивания, битовые, логические и другие.

C++	Kotlin	Python
= *int a = 4;*	= *val c = 5*	= *>>> a = 20*
/ *int c = a / b*	/ *val c = a / b*	/ *>>> c = a / b*
% *int c = a % b*	% *val c = a % b*	% *>>> c = a % b*
++ *a++ / ++a*	++ *a++ / ++a*	-
&& *a && b*	&& *i > 0 && i != 2*	and *a and b*
<< *8 << 1 // 16*	shl *8 shl 1 // 16*	<< *>>> 8 << 1 // 16*
& *7 & 3 // 3*	and *7 and 3 // 3*	& *7 & 3 // 3*
?: *Y = x>9 ? 100 : 200*	-	-

18.Строковое выражение

Строковое выражение по виду напоминает математическое выражение, но результатом его является новая строка. Операция + (сложение) в контексте строкового выражения означает объединение (конкатенацию) двух и более строк.

19.Оператор (инструкция) присваивания

Назначение: изменение значения переменной.

Аргументом оператора присваивания может быть математическое, логическое, строковое выражение или функция.

Во многих языках программирования оператор присваивания можно совместить с объявлением переменной.

20.Следование, ветвление, цикл

Следованием называется конструкция, представляющая собой последовательное выполнение двух или более операторов (простых или составных).

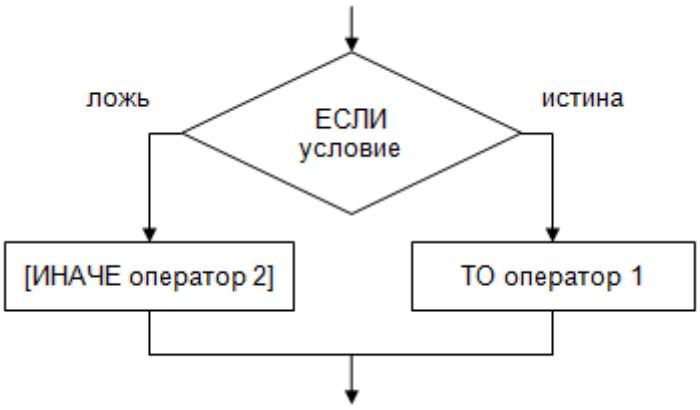
Ветвление задает выполнение либо одного, либо другого оператора в зависимости от выполнения условия.

Цикл задает многократное выполнение оператора|операторов.

21.Условный оператор

Назначение: организация ветвления хода вычислений в зависимости от условия.

ЕСЛИ условие истинно, выполняется основная часть условного оператора **ТО**. Так же можно определить альтернативный ход вычислений **ИНАЧЕ**, наличие которого не является обязательным.



Блок-схема условного оператора.

C/C++, C#, Java	**Pascal/Delphi**	**Visual Basic/VB.NET**

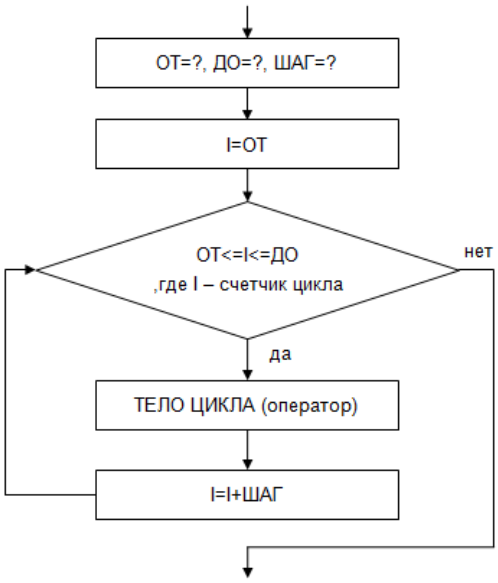
if(a > b){
  max = a;
}
else{
  max = b;
}
	if m > n
	  then
	  begin
	  max :=m;
	  end
	  else max :=n;
		If (m > n) Then
		  max = m
		Else
		  max = n

Python	**Kotlin**	

if a > b:
  max = a
else:
  max = b
	var max: Int
	if (a > b)
	  max = a
	else
	  max = b	

22.Цикл с определенным количеством итераций или цикл FOR

Организация числа итераций, определенного заголовком цикла.



Блок-схема цикла FOR.

C/C++, C#, Java	*Pascal/Delphi*	*Visual Basic/VB.NET*

`int N=10;`
`int F=1;`
`for (int i = 2; i <= N;`
`i++)`
`{`
`F = F * i;`
`}`
	`program example;`
	`var N,F,I:Integer;`
	`begin`
	`N:=10;`
	`for i:=2 to N do`
	`begin`
	`F:=F*I;`
	`end;`
	`end;`
		`Dim N As Integer = 10`
		`Dim F As Integer = 1`
		`For I As Integer = 2 To N`
		`F = F * I`
		`Next`

23.Цикл перебора элементов в коллекции или цикл FOREACH

Организовать ***перебор элементов*** в массиве или коллекции. Количество итераций соответствует числу элементов в заданном множестве.

24.Выход из подпрограммы или оператор RETURN / EXIT

В языках C/C++, C#, Java и VB.NET оператор ***return*** в контексте процедуры завершает ее работу, а в контексте функции завершает и определяет возвращаемое значение: ***return*** result_value;.

Пример. В языках Pascal/Delphi оператор ***exit*** завершает выполнение процедуры или функции, а значение функции присваивается по ее имени: function_name := result_value;.

25.Метка и переход к метке или операторы LABEL и GOTO

Выделение определенного места в программе (установка метки) с возможностью безусловного перехода к этой метке из другой части программы. Переход чаще всего возможен только в границах одной процедуры или функции.
В большинстве языков программирования работа с метками будет выглядеть примерно так:

Псевдокод	***C/C++,C#***
`[label] Имя_метки:`	`goto Имя_метки;`

Оператор goto в C++ передает управление на помеченный оператор.
Метка – это идентификатор, областью видимости которого является функция, в теле которой он встречается.
В C++ есть пять операторов, изменяющих естественный порядок выполнения вычислений:

- оператор выхода из цикла и переключателя ***break***;
- оператор перехода к следующей итерации цикла ***continue***;
- оператор возврата из функции ***return***;
- оператор безусловного перехода ***goto***;
- оператор генерации исключения ***throw***.

26.**Обработчик** исключительной ситуации

Выделение части программного кода, результат выполнения которого потенциально может привести к ошибке или исключительной ситуации с целью контроля над процессом обработки этой ситуации.

Пример: неудачная попытка соединения с базой данных.

27.**Декомпозиция** программного кода

Процедуры и ***функции*** являются ключевыми конструкциями в ***процедурном программировании***.

Назначение: позволяют разбить весь программный код на фрагменты для решения локальных задач, пригодные для многократного использования.

a. **Процедура**

Выделяет часть программы, пригодную для повторного использования.
Имеют ***имена***, отражающие суть того, что они выполняют.
Могут иметь набор входных параметров – ***аргументов процедуры***.
Вызов: в исходном коде указать имя и список значений ее аргументов.

b. **Функция**

Функция возвращает результат определенного для нее типа. Вызов функции может быть частью составного оператора, аргументом оператора присваивания, частью математического, логического или строкового выражения.

28.Элементарные (базовые) типы данных

Описание типа	Размер (биты)	Диапазон	.NET	C#	VB.NET	C/C++	Java	Pascal/Delphi
Байт со знаком	8	от -128 до 127	SByte	sbyte	SByte	signed char	byte	shortint
Байт без знака	8	от 0 до 255	Byte	byte	Byte	unsigned char	-	byte
Короткое целое число со знаком	16	от -32 768 до 32 767	Int16	short	Short	short	short	integer
Короткое целое число без знака	16	от 0 до 65 535	UInt16	ushort	UShort	unsigned short	-	word
Среднее целое число со знаком	32	от -2147483648 до 2147483647	Int32	int	Integer	int	int	longint
Среднее целое число без знака	32	от 0 до 4294967295	UInt32	uint	UInteger	unsigned int	-	-
Длинное целое число со знаком	64	от -9223372036854775808 до 9223372036854775807	Int64	long	Long	long	long	-
Длинное целое число без знака	64	от 0 до 18446744073709551615	UInt64	ulong	Ulong	unsigned long	-	-
Вещественное число одинарной точности с плавающей запятой	32	$\pm 1{,}5*10^{-45}$ до $\pm 3{,}4*10^{33}$	Single	float	Single	float	float	single
Вещественное число двойной точности с плавающей запятой	64	$\pm 5*10^{-324}$ до $\pm 1{,}7*10^{306}$	Double	double	Double	double	double	double
Символ (8 бит)	8	ASCII	-	-	-	char	-	char
Символ (16 бит)	16	UNICODE	Char	char	Char	wchar_t	char	-
Логический тип	8	{true, false}	Boolean	bool	Boolean	bool	boolean	boolean
Строка	-	-	String	string	String	char*/wchar_t*	String	string
Пустой тип	-	-	-	void	-	void	void	-

29.Kotlin (Ко́тлин)

Kotlin, разрабатываемый компанией JetBrains, — объектно-ориентированный, статически типизированный язык программирования, работающий поверх JVM. Компилируется также в JavaScript и на другие платформы.

Цель — создать язык более лаконичный и типобезопасный, чем Java, и более простой, чем Scala.

Синтаксис языка похож на Pascal, TypeScript, Haxe, PL/SQL, F#, Go и Scala, C++, Java, C# и D.

При объявлении переменных и параметров типы данных указываются после названия (разделитель двоеточие).

Точка с запятой, как разделитель операторов, является необязательной (как в Scala и Groovy) в большинстве случаев перевода строки достаточно, чтобы компилятор понял, что выражение закончилось.

Кроме объектно-ориентированного подхода, Kotlin также поддерживает процедурный стиль с использованием функций.

Точкой входа в программу является функция "main", которая принимает массив параметров командной строки.

Kotlin также поддерживает вывод типов.

Объявления в Kotlin объединяются в *пространства имен* (namespace), и функция может быть объявлена непосредственно внутри пространства имен:

```
namespace example {
  fun max(a : Int, b : Int) :
Int {
    if(a < b)
      returna
    returnb
  }
}
```

Переменные в Kotlin объявляются с помощью ключевых слов **val** (неизменяемые переменные) и **var** (изменяемые):

```
var sinSum : Double = 0.0
for (x in xs) {
  val y = sin(x)
  sinSum += y
}
```

Управляющие конструкции. When

Kotlin поддерживает традиционные для императивных языков управляющие конструкции **if**, **for** и **while**, на которых мы не будем останавливаться подробно, а также конструкцию **when** — операцию ветвления, которую можно

рассматривать как расширенную версию традиционного оператора **switch,** позволяет проверять аргумент на принадлежность коллекции, с помощью операции **in:**

```
when (x) {
  in set => print(“in set”)
  in 1..10 => print(“1..10”)
  else => print(“Default
branch”)
}
```

Система типов

Нулевые ссылки. Система типов языка Kotlin позволяет гарантировать отсутствие некоторых видов ошибок, например, разадресации нулевой ссылки. Типы в Kotlin делятся на содержащие **null** и не содержащие **null**. Типы, содержащие **null**, аннотируются ***знаком вопроса***:

```
fun isEmpty(s: String?): Boolean
{...}
```

Знак вопроса после имени типа (String) означает, что ссылка **s** указывает на объект класса String *или имеет значение* ***null***. Результат функции isEmpty, в свою очередь, должен быть булевым значением и не может иметь значения **null**, поскольку соответствующий тип не проаннотирован знаком вопроса.

Компилятор не позволяет разадресовать ссылку типа String? без предварительной проверки.

Автоматическое приведение типа работает для всех условных конструкций Kotlin: **if, when, while**, ||, && и т. д.

30.TypeScript

TypeScript — язык программирования (Microsoft в 2012 г.) – средство разработки веб-приложений, расширяющее возможности JavaScript.

TypeScript является языком обратно совместимым с JavaScript и компилируется в JavaScript. После компиляции программу на TypeScript можно выполнять в любом современном браузере.

TypeScript является ***надстройкой*** над JavaScript. Следовательно, программа JavaScript также является правильной программой TypeScript.

TypeScript отличается от JavaScript возможностью явного ***статического*** назначения типов, поддержкой использования полноценных классов (как в традиционных объектно-ориентированных языках), а также поддержкой подключения модулей, что повышает скорость разработки, облегчает читаемость, рефакторинг и повторное использование кода, помогает осуществлять поиск ошибок на этапе разработки и компиляции.

Объявление типов

TypeScript обеспечивает объявления типов для статической проверки их согласования. Это не является обязательным и может быть проигнорировано, в этом случае используется обычная динамическая типизация JavaScript.

Существует несколько примитивных типов: number, boolean и string. Слабо или динамически введенные структуры имеют тип any.

```
function add(left: number, right: number):
number {
      return left + right;
}
```

Компилятор TypeScript пытается вывести типы, если они не указаны явно. Метод **add** в приведенном выше коде будет выводить как возврат типа number, даже если бы не было предусмотрено никакого возврата типа в определении метода. Правило основано на статических типах left number и right number. Компилятор знает о том, что результат сложения двух значений типа number всегда number. Явное указание возвращаемого типа позволяет компилятору проверить правильность.

Язык TypeScript:

- язык со строгой типизацией;
- код, написанный на TypeScript, компилируется в JavaScript;
- язык обратно совместим с JavaScript;
- язык разработан компанией Microsoft.

TypeScript используется для устранения недостатков JavaScript при применении его в масштабных приложениях.

Код экспериментального компилятора, транслирующего TypeScript в JavaScript, распространяется под лицензией Apache. Его разработка ведётся в публичном репозитории через сервис GitHub.

31.Ruby

Ruby — динамический, рефлективный, интерпретируемый высокоуровневый язык программирования.

Язык предоставляет возможности независимой от операционной системы реализации многопоточности, строгой динамической типизации, сборки мусора и др. По особенностям синтаксиса он близок к языкам Perl и Eiffel, по объектно-ориентированному подходу — к Smalltalk. Также некоторые черты языка взяты из Python, Lisp, Dylan и Клу.

Кроссплатформенная реализация интерпретатора языка является полностью свободной.

Мацумото, фанат объектно-ориентированного программирования, мечтал о языке, более мощном, чем Perl, и более объектно-ориентированном, чем Python. Основное назначение Ruby — создание простых и в то же время понятных программ, где важна не скорость работы программы, а малое время разработки, понятность и простота синтаксиса.

Язык следует принципу «наименьшей неожиданности»: программа должна вести себя так, как ожидает программист. Однако в контексте Ruby это означает наименьшее удивление не при знакомстве с языком, а при его основательном изучении ☺.

Семантика

Ruby — полностью объектно-ориентированный язык. Все данные являются объектами (нет примитивных типов). Каждая функция — метод.

Ruby использует вызов по соиспользованию (call-by-sharing), хотя в сообществе Ruby часто говорят, что он использует вызов по ссылке. Некоторые неожиданные эффекты такого решения:

```
a = "abcdefg"
b = a
b                          #=> "abcdefg"
a[3] = 'R'
b                          #=> "abcRefg"
```

При изменении значения переменной ***a*** неявно изменилось и значение ***b***, так как они содержат ссылку на один объект. То есть механизм присваивания действует одинаково для всех объектов, в отличие от многих других языков, где присваивание может означать как копирование значения, так и копирование ссылки на значение.

Целые переменные в Ruby автоматически конвертируются между типами Fixnum (32-разрядные) и Bignum (больше 32 разрядов) в зависимости от их значения. Это позволяет производить целочисленные математические расчёты со сколь угодно большой точностью.

32.Java

Не требует предварительного объявления переменных, но для интерпретатора желательно, чтобы переменным присваивалось пустое значение null (тогда интерпретатор знает, что идентификатор обозначает переменную, а не имя метода).

Язык использует простые соглашения для обозначения области видимости:

- var — локальная переменная,
- @var — переменная экземпляра (член или поле объекта класса),
- @@var — переменная класса,
- $var — глобальная переменная.

В языке есть 2 эквивалентных способа записи блоков кода:

```
{ puts "Hello, World!" }
```

```
do puts "Hello, World!" end
```

Синтаксис Java основан на языке программирования C. В частности, от него унаследованы без изменений:

- обозначения начала/конца блока кода фигурными скобками;
- обозначения, ассоциативность и приоритет большинства встроенных операций (присвоение, арифметические, логические, побитовые операции, операции инкремента/декремента, тернарная условная операция ?:);
- синтаксис описания и использования переменных и функций (порядок «тип, имя», использование модификаторов, обязательность скобок для функций, описание формальных параметров);
- синтаксис всех основных конструкций: условного оператора, циклов, оператора множественного выбора.

Заключение

Каждый **256**-й день года отмечается неофициальный праздник – ***День программиста***. Это число (2^8 – два в восьмой степени) выбрано неслучайно, так как это количество чисел, которые можно выразить с помощью **одного байта**!

В 2022 году отмечается 13 сентября (256-й день 2022 года) и в этом году выпадает на вторник.