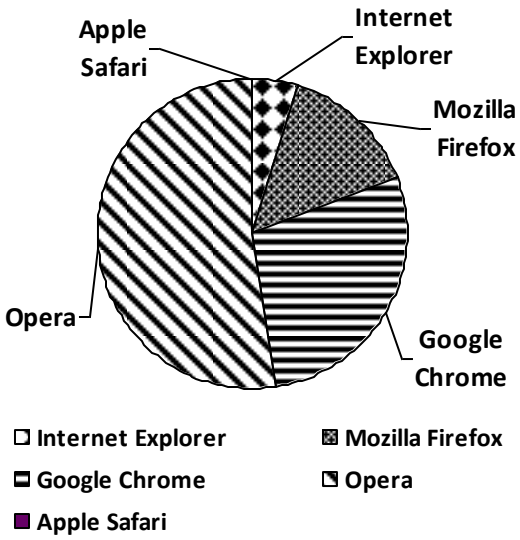

Рис. 18. Диаграмма опроса пользователей социальной сети «ВКонтакте»

Тема браузеров очень широка и практически неисчерпаема, ведь сегодня существует достаточно большое количество браузеров, как сходных друг с другом, так и нет. И проанализировав, исследовав и сравнив только часть из них, нельзя с большой уверенностью сказать, какой из них лучший, несмотря на то, что сравнивались достаточно популярные браузеры. Каждый браузер имеет свои достоинства, недостатки и индивидуальные возможности, следовательно, каждый пользователь сможет найти себе браузер по душе. Поэтому следует попробовать попользоваться несколькими браузерами, а уже потом сделать окончательный выбор.

А.А. Царегородцев, РГППУ
студент группы КТ-306
Руководитель: ст. преп. кафедры СИС
Н.В. Меньшикова

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Вокруг нас множество электронных устройств, внутри которых содержится маленький компьютер, который нужно программировать, чтобы устройства хорошо работали. Изначально компьютеры программировались в машинных кодах в виде чисел, которые записывались в память и потом выполнялись центральным процессором. Однако это было неудобно и по-

этому достаточно быстро придумали называть команды с помощью некоторых англоязычных аббревиатур, которые было легче запомнить. Такой язык назывался языком Ассемблера, так как он позволял не запоминать адреса конкретных областей данных, а называть их именами и затем эти адреса в процессе преобразования в машинные коды автоматически подставлять. Однако такой способ программирования был доступен только продвинутым программистам. Поэтому в 50-х годах Дж. Бэкус придумал язык FORTRAN (Formula Translator), в котором формулы могли записываться на языке, похожем на математический язык. Такой язык получил название языка высокого уровня. Языки высокого уровня также называются императивными, поскольку они состоят из набора операторов, каждый из которых «что-то делает», то есть он изменят состояние памяти определенным образом. Память в таких языках представлена переменными, которые мы можем описывать, задавая им определенный тип.

С тех пор появилось множество языков программирования, и у большинства начинающих программистов появилась некоторая сложность в выборе языка и среды программирования. Чтобы прийти к какому-либо решению, нужно прочитать «море» литературы, рассмотреть и сравнить все возможные языки программирования. И чтобы не потеряться в таком объеме информации и все же выбрать для себя наиболее полезный и интересный язык необходимо уяснить ряд моментов.

Во-первых, следует отличать язык программирования от среды программирования – набора средств для редактирования исходных текстов, генерации исполняемого кода, отладки, управления проектами и т.д. Каждая среда программирования предоставляет свой интерпретатор или компилятор с этого языка.

Во-вторых, очень существенно, для какой цели выбирается язык - для обучения программированию либо для решения конкретной прикладной задачи. В первом случае язык должен быть простым для понимания, строгим и по возможности лишенным «подводных камней». Во втором - пусть сложным, но эффективным в решении поставленной программистом задачи.

Конечно, не стоит забывать о так называемых учебных задачах, которые чаще всего страдают излишней абстрактностью и неприменимостью в жизни. Эффективное освоение языка возможно только на реальных примерах. С другой стороны, программирование решения полноценной проблемы «из жизни» на начальном этапе освоения языка может оказаться непосильной ношей, которая отпугнет, а не заинтересует.

В-третьих, необходимо учитывать возможности языка программирования:

- какими базовыми типами данных, характеризуется язык;
- способы определения процедур и функций;
- объектно-ориентированность языка;
- переносимость, т.е. независимость от аппаратуры, реализуемая при помощи семантики, не зависящей от конкретной машины.

В-четвертых, с точки зрения эффективности, важно как исполняется программа – последовательной интерпретацией исходного текста (интерпретатор) либо непосредственным исполнением готового кода (компилятор). Интерпретатор целесообразно использовать лишь в случае, когда скорость интерпретации не сильно сказывается на эффективности программы.

Итак, рассмотрим и попытаемся сравнить языки программирования с точки зрения приспособленности под различные классы задач, однако, в виду того, что языков программирования существует достаточно много, мы рассмотрим лишь самые популярные из них: Паскаль и Си, Delphi, C# и Java.

Начнем с процедурных языков программирования – Pascal (Паскаль) и C (Си), которые отличаются по синтаксису, но при этом имеют много общего между собой.

Типы данных в Паскале и Си делятся на простые и порядковые. К простым типам данных относятся числа с плавающей запятой (real), целые (integer), символьный (char), логический (boolean) и перечисления (конструктор нового типа, введённый в Pascal), а к порядковым - целые типы (знаковые и беззнаковые), логический (boolean), символьный (char), перечислимые типы. [2, 5]

Синтаксис этих языков программирования рассмотрим на примере программы «Hello, World!» (табл. 2).

Таблица 2

Листинги кода Pascal и C

Pascal	C

Program HelloWorld(output);
begin
  writeln('Hello, World!');
end.
	#include <stdio.h>
	int main()
	{
	  printf("Hello, World!\n");
	  return 0;
	}

На примере видно, что количество символов в программе и на Си, и на Паскале почти одинаковое, но обозначение логических скобок (Pascal: begin end; C: { }) в Паскале гораздо массивнее, нежели в Си.

Как уже было сказано ранее, Паскаль и Си являются процедурными языками, то есть в нем подпрограммы делятся на процедуры и функции. Синтаксически процедуры и функции, как в Паскале, так и в языке Си имеют одинаковую структуру, разница заключается лишь в том, что в этих языках используются собственные ключевые слова.

Тело процедуры в Паскале, как и программы, в свою очередь, может содержать описания процедур и функций. Таким образом, процедуры и функции могут быть вложены друг в друга, при этом тело программы — самое верхнее в цепочке. В Си же программа начинается с входной процедуры Main и в отличие от Паскаля её тело, как и тело других процедур не может содержать описания процедур и функций.

Паскаль, как и Си, не является объектно-ориентированным языком программирования, что делает его не универсальным в сравнении с другими языками, поэтому в мире большей популярностью пользуются их потомки Object Pascal, C++ и C#.

Языки Object Pascal, C# и Java, которые будут рассматриваться и сравниваться в дальнейшем, относятся к классу объектно-

ориентированных языков. Базовые типы данных в этих языках абсолютно те же, что в Паскаль и в Си.

Синтаксис языков Object Pascal, C# и Java так же рассмотрим на примере программы «Hello, Wolrd!» (табл. 3):

Таблица 3

Листинги кода Object Pascal, C#, Java

Object Pascal	`program Helloworld;`
	`begin`
	`  writeln('Hello, world!');`
	`end.`

C#	`using System;`
	`class Program`
	`{`
	`    static void Main()`
	`    {`
	`        Console.WriteLine("Hello World!");`
	`    }`
	`}`

Java	`public class HelloWorld {`
	`    public static void main(String[] args) {`
	`        System.out.println("Hello, world!");`
	`    }`
	`}`

На этом примере можно увидеть, что синтаксис Object Pascal точь-в-точь похож на синтаксис языка Паскаль. Это неудивительно, потому что Object Pascal является потомком Паскаля. Синтаксис Java один в один похож на синтаксис C#, так как язык Java имеет си-подобный синтаксис. Это обусловлено тем, что Си, прародитель языка C#, в свои времена был очень популярен и его синтаксис стал основой для многих других языков. Но если абстрагироваться от примера «Hello, World!», то основанный на использовании пар begin end для разделения блоков кода, синтаксис Delphi является более громоздким и прямолинейным, чем синтаксис C#. Что касается синтаксиса Java, то синтаксис C# немного мощнее, чем Java, так как C#

поддерживает перегрузку операторов и безопасные по типу перечисления.[1, 3, 4]

Описание подпрограмм в Object Pascal и C# с Java в точности совпадают с описанием подпрограмм в Паскале и Си соответственно.

Как уже говорилось ранее Java, C# и Object Pascal являются объектно-ориентированными языками, поэтому на объектно-ориентированности этих языков программирования следует остановиться поподробней. И первое, что стоит рассмотреть это описание классов и объектов, так как объектно-ориентированное программирование строится на идее класса (объекта). Например, если мы имеем класс MyClass с методом MyMethod, то мы можем написать следующее (табл. 4):

Таблица 4

Примеры описания классов

Object Pascal	`var obj: MyClass;`
	`begin`
	`obj := MyClass.Create;`
	`obj.MyMethod;`

C#	`MyClass obj;` `obj.MyMethod();`

Java	`MyClass obj;`
	`obj = new MyClass();`
	`obj.MyMethod();`

В C# память для этого объекта обычно выделяется в стеке, и вы можете сразу начать использовать объект, как это сделано во второй строке [0]. В Java прежде чем использовать объект, нужно вызвать «new» для выделения под него памяти [4]. Object Pascal использует подобный подход, но требует отдельных предложений для объявления и инициализации [3].

Если вы создали и использовали объект, вам нужно уничтожить его, чтобы не занимать неиспользуемую память. В C# уничтожить объект, расположенный в стеке, довольно просто. С другой стороны, уничтожение объектов, созданных динамически, зачастую является сложной проблемой. Есть много решений, включая подсчет ссылок и «интеллектуальные» указатели, но ни один из них не даёт простого решения. Первое впечатление

для C# программистов, что использование ссылочно-объектной модели сделает ситуацию только хуже [0]. Уничтожение объектов не касается Java, так как виртуальная машина запускает алгоритм сборки мусора. Сбор мусора предоставляется программистам бесплатно, но он может неблагоприятно влиять на эффективность выполнения приложения. Отсутствие явно записываемых деструкторов может приводить к ошибкам в завершающем коде [4]. В Delphi, наоборот, нет механизма сбора мусора. Однако компоненты Delphi поддерживают идею владельца (Owner) объекта: владелец становится ответственным за уничтожение всех объектов, которыми он владеет. Это делает управление уничтожением объекта очень простым и прямым. Delphi также использует механизм подсчёта ссылок для строк, динамических массивов и интерфейсов, освобождая объект в памяти [3].

Общим элементом всех трех языков является присутствие трех модификаторов доступа, указывающих на различные уровни инкапсуляции класса: public, protected и private. Public означает: видимый любым другим классом. Protected означает: видимый производными классами. Private означает: отсутствие видимости извне. Но в деталях есть различия. В C# можно использовать ключевое слово friend для обхода инкапсуляции. Видимость по умолчанию для класса - Private, для структур – Public [0]. В Delphi Private и Protected относятся только к классам других юнитов [3]. В терминах C#, класс является дружественным для любого другого класса, определенного в том же юните [0]. В Delphi есть еще один модификатор доступа – Published, который генерирует информацию времени выполнения об элементах [3]. В Java отличие синтаксиса от синтаксиса C# в том, что модификатор доступа повторяется для каждого элемента класса. А конкретнее, по умолчанию в Java используется friendly, это значит, что элемент видим для других классов этого же пакета [4]. Подобным образом, Protected означает видимость для подклассов, тогда как комбинация Private Protected соответствует Protected в C# [0].

Наследование у классов – одно из оснований объектно-ориентированного программирования. Оно может быть использовано для выражения генерализации или специализации. Основная идея в том, что можно определить новый тип, расширяя или модифицируя существующий,

другими словами, производный класс обладает всеми данными и методами базового класса, новыми данными и методами и модифицирует некоторые из существующих методов [0]. Различные объектно-ориентированные языки используют различные жаргоны для описания этого механизма, для класса, от которого наследуется и для нового класса. Вот пример синтаксиса наследования (табл. 5):

Таблица 5

Пример синтаксиса наследования

Delphi	`type`
	`dog = class (animal)`
	`    …`
	`end;`

C#	`class dog: public animal`
	`{`
	`    …`
	`};`

Java	`class dog extends animal`
	`{`
	`    …`
	`}`

C# использует слова Public, Protected и Private для определения типа наследования и чтобы спрятать наследуемые методы или данные, делая их приватными или защищёнными. Хотя публичное наследование наиболее часто используется, по умолчанию берётся приватное. C# - единственный из этих трех языков, поддерживающий множественное наследование [0]. Delphi при наследовании использует не ключевые слова, а специальный синтаксис, добавляя в скобках имя базового класса. Этот язык поддерживает только один тип наследования, который в C# называется публичным [3]. Java использует слово extends для выражения единственного типа наследования, соответствующего публичному наследованию в C#. Java не поддерживает множественное наследование. Классы Java тоже происходят от общего базового класса [4].

В некоторых объектно-ориентированных языках каждый класс происходит от одного базового класса по умолчанию. Этот класс, часто называемый Object, обладает некоторыми основными способностями, доступными всем классам. Фактически, все другие классы в обязательном порядке ему наследуют. Этот подход является общим ещё и потому, что так первоначально делалось в SmallTalk (объектно-ориентированный язык программирования). Хотя язык C# и не поддерживает такое свойство, многие структуры приложений базируются на нём, вводя идею общего базового класса [0]. В Delphi каждый класс автоматически наследует классу TObject. Так как язык не поддерживает множественное наследование, все классы формируют гигантское иерархическое дерево. Общей практикой является использование этого класса, когда нужно передать объект неизвестного типа [3]. В Java, как и в Delphi, все классы безоговорочно наследуют классу Object [4].

Как и множество других языков программирования, все языки программирования, рассмотренные в статье, с легкостью переносятся, другими словами, они не зависят от аппаратуры компьютера.

Наконец можно подвести итог рассмотрения сравнения языков программирования. Все эти языки подходят под различные классы задач. Например:

Паскаль предназначен, прежде всего, для обучения программированию. Построенный по принципу "необходимо и достаточно", он располагает строгим контролем типов, конструкциями для описания произвольных структур данных, небольшим, но достаточным набором операторов структурного программирования. К сожалению, обратной стороной простоты и строгости является громоздкость описаний конструкций языка. Наиболее известная реализация - Turbo/Borland Pascal - несмотря на отличия от стандарта Паскаля, представляет собой среду и набор библиотек, сделавшие из учебного языка промышленную систему для разработки программ в среде MS-DOS.

В основе языка Си лежат полный и эффективный доступ ко всем ресурсам компьютера, средства программирования высокого уровня, переносимость программ между различными платформами и операционными системами. Благодаря перечисленным качествам, Си занял позицию универсального языка для

любых задач. Но его применение может стать неэффективным там, где требуется получить готовый к употреблению результат в кратчайшие сроки, либо там, где невыгодным становится сам процедурный подход.

Главное предназначение Delphi - быстрое создание приложений (Rapid Application Developing, RAD). Подобные средства позволяют в кратчайшие сроки создать рабочую программу из готовых компонентов, не растрачивая массу усилий на мелочи. Особое место в таких системах занимают возможности работы с базами данных.

Язык Java появился в ответ на потребность в идеально переносимом языке, программы на котором эффективно исполняются на стороне клиента всемирной паутины. В виду специфики окружения, Java может быть хорошим выбором для системы, построенной на Internet/Intranet технологии.

C#, сохраняя совместимость с Си, вносит возможности объектно-ориентированного программирования, выражая идею класса как определяемого пользователем типа. Благодаря перечисленным качествам, C# занял позицию универсального языка для любых задач.

Мы надеемся, что на основе проведенного сравнительного анализа языков программирования, многие начинающие программисты, смогут определить для себя какой язык всё-таки лучше и для каких задач его лучше всего использовать.

Библиографический список

1. Дейтел, Х. C# в подлиннике. Наиболее полное руководство [Текст] / Х., Дейтел - СПб.: БХВ-Петербург, 2006. - 1057 с.

2. Павловская, Т.А. Паскаль. Программирование на языке высокого уровня: Учебник для вузов [Текст] / Т.А., Павловская - СПб.: Питер, 2006. - 393 с.

3. Рубенкинг, Н. Язык программирования Delphi для «чайников» [Текст] / Н. Рубенкинг - М.: Диалектика, 2007. - 336 с.

4. Эккель, Б. Философия Java [Текст] / Б., Эккель - 4-е изд. - СПб.: Питер, 2003. - 976 с.

5. Яновский, А.В. Язык Си - первый шаг к серьезному программированию: Учеб. пособие [Текст] / А.В., Яновский - Томск, 2007. - 132 с.