МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)

Кафедра философии

РЕФЕРАТ ПО ИСТОРИИ НАУКИ

ИСТОРИЯ РАЗВИТИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Аспирант — Перов Максим Николаевич

Научный руководитель аспиранта _____ Дроздов А.Ю.

Преподаватель кафедры философии — Храмов О.С.

Москва 2016

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В связи с появлением первых вычислительных машин в середине двадцатого столетия, было положено начало бурному развитию языков программирования. Если окунуться в историю прошлого столетия и посмотреть на первые вычислительные машины, то мы сильно удивимся их отличиям от современных компьютеров, которые глубоко вошли в наш быт и без которых мы уже не представляем своей жизни. То же самое и с первыми языками программирования: программы, разработанные на них, современному программисту покажутся лишь не читаемым набором коротких фраз.

Таким образом, вместе с бурным развитием вычислительных машин происходила и не менее бурная эволюция языков программирования для них.

Первые компьютеры для современного человека выглядят чем-то очень странным, большим по размеру и непонятным в использовании, а и их применение значительно отличалось от сегодняшнего. Так и первые языки программирования для современного программиста могут показаться чем-то удивительным. Это связано в первую очередь с тем, что электронно-вычислительные машины устроены следующим образом: понятные для них команды являются машинным кодом — определенной последовательностью из нулей и единиц. Собственно, все программы для ЭВМ являлись не чем иным, как машинными кодами.

Как выяснилось впоследствии, такой способ программирования электронных машин оказался неудобным, так как было легко допустить ошибку, например, написать вместо единички ноль в команде. Таким образом, было очень трудно написать программу, не допустив ни одной ошибки, не говоря уже о каких-либо модификациях.

Такое сложное управление компьютером побудило программистов создать несколько упрощённое программирование ЭВМ. Этот процесс выглядел следующим образом: набор символических команд, понятных

человеку, при помощи специального транслятора переводился в набор машинных команд.

Собственно, это упрощение и можно назвать началом развития языков программирования.

Считается, что первым в мире языком программирования является Планкалкюль, созданный в середине 1940-х годов немецким физиком Конрадом Цузе. В переводе на русский язык данное название звучит как «планирующее исчисление». К сожалению, из-за начавшейся в это же время Второй мировой войны, проект не получил практической реализации, более того, его описание опубликовали лишь в 1972 году. Разработки Цузе могли значительно ускорить процесс развития языков программирования, если бы стали доступны другим учёным раньше.

Подводя итог всему вышесказанному, хотелось бы сказать, что сравнительно молодая история программирования так же интересна и богата, как и история любых научных разработок.

ГЛАВА I Низкоуровневые языки программирования

В начале развития компьютерной техники для управления ЭВМ приходилось использовать непосредственно команды или инструкции процессора, применяя так называемый машинный язык. Можно дать точное определение данного понятия: «Машинный код (машинный язык) — система команд конкретной вычислительной машины, которая интерпретируется непосредственно процессором этой вычислительной машины.» [1, стр. 56] В его основе лежит набор элементарных команд, понятных процессору: запись данных в нужную ячейку памяти, сложение данных из разных регистров процессора и т.д. В свою очередь, каждая инструкция представляет из себя двоичное число, но для удобства команды могут быть записаны и в шестнадцатеричной системе счисления, а уже потом переведены в двоичный код.

Программы в виде машинного кода выглядят очень громоздкими, например, программа для одного из современных процессоров, выводящая на экран сообщение «Hello, world!» выглядит в шестнадцатеричном представлении так: BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD 10 E2 F9 CD 20 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21. Невозможно не согласиться с тем фактом, что это не только громоздко, но и требует специальных знаний для восприятия.

Стоит подчеркнуть, что у каждого типа процессора может быть свой набор команд, никак не пересекающийся с другим типом. Поэтому программа на машинном коде для процессора A выглядит одним образом, а программа, делающая абсолютно одно и то же, для процессора B может выглядеть совершенно по-другому. Из этого следует, что код программ практически не переносим с одной машины на другой, что приводит к написанию «с нуля».

Таким образом, для программирования компьютера было необходимо знать машинный язык и понимать устройство ЭВМ, для которого писалась программа. В сумме, это задавало высокие требования к программистам той поры, поэтому и самой профессии не существовало, а программистами

являлись математики и физики, использовавшие компьютеры для необходимых расчётов.

В начале второй половины двадцатого века увеличился спрос на разработку программного обеспечения, вместе с этим усложнялись программы, соответственно, и их исходные коды становились все больше и сложнее. Это побудило программистов к использованию специальных программ, ассемблеров, позволявшим транслировать код программ на языках ассемблера в машинный код. Переход к использованию ассемблеров может считаться следующим этапом развития языков программирования.

Язык ассемблера представляет из себя некий формат записи машинных команд, удобный для человеческого восприятия. То есть теперь разработчик может не знать, как выглядят команды процессора в числовом виде, ему достаточно лишь знать запись команды на языке ассемблера. Причём данная запись легче запоминается, а значит, появились следующие преимущества: программы стало намного проще писать; они выглядят более читабельными и понятными человеку, нежели в машинных кодах. Например, команда переноса с одного регистра в другой - mov, то есть сокращение от слова move, что в переводе с английского - перемещать, а команда сложения - add (добавить). Использование мнемонических команд, безусловно, упростило жизнь программистам, но не снизило требования к ним: по-прежнему легко допустить ошибку, программы не менее сложно контролировать, да и проблему зависимости от платформ не удалось решить с их помощью.

К первым ассемблерам можно отнести Autocode, IPL, FLOW-MATIC, предложенные в середине 1950-х годов. Хотя язык ассемблера придуман давно, он используется в виде ассемблерных вставок и по сей день в тех программах, где существуют высокие требования к производительности. Актуальность его использования заключается в том, что современные языки зачастую не позволяют писать так же оптимально.

Итак, машинный язык и язык ассемблера относятся к классу низкоуровневых языков программирования.

Низкоуровневые языки программирования были первыми в индустрии разработки программного обеспечения. Однако в связи с усложнением необходимых программ становилось все труднее писать требуемый код качественно и в нужные сроки, поэтому разработчикам понадобились языки с более высоким уровнем абстракции, которым посвящена следующая глава.

ГЛАВА II Высокоуровневые языки программирования

Следующий этап в эволюции программирования — использование высокоуровневых языков программирования, в которых имитируются естественные языки, с использованием их разговорных слов и математических символов. Такой подход ещё больше упрощает разработку, чем язык ассемблера. Однако не стоит полагать, что теперь вместо «mov» в высокоуровневых языках нужно писать «move», так как программист пишет определенные смысловые конструкции, которые описывают данные и операции над ними, но их описания на машинном коде длинны и сложны для понимания.

Например, в некоторых высокоуровневых языках инкрементирование переменной выглядит очень просто: a += 1, а на ассемблере это займёт несколько операций:

```
mov ax, a     // перенести значение переменной a в регистр
add ax, 1     // инкрементировать значение регистра ax
mov a, ax     // перенести полученное значение в переменную a
```

Перенос значения из переменной в регистр необходим, так как во многих архитектурах «add» работает лишь с регистрами. Данный простой пример уже показывает, насколько проще писать на языках высокого уровня, благодаря их расширенным возможностям, а так же абстрагированию. Кроме этого, код на высокоуровневых языках еще меньше зависит от платформы. Единственное условие — наличие компилятора под нужную архитектуру.

Компилятор — компьютерная программа, транслирующая код, составленный на языке высокого уровня, в эквивалентный низкоуровневый код, близкий к машинному, другими словами, программа выполняющая компиляцию. Отсюда мы можем сделать вывод, что вместе с появлением высокоуровневого программирования возникли такие понятия, как компилятор и компиляция.

Как было сказано ранее, Планкалкюль (с немецкого — «планирующее исчисление») — первый высокоуровневый язык программирования,

предложенный немецким физиком Конрадом Цузе в середине 40-х годов двадцатого столетия. Он предназначался для вычислительной машины Z4, созданной Цузе в годы Второй мировой войны. Война была тем самым внешним фактором, который не позволил научному миру того времени ознакомиться с новоявленным языком программирования. В итоге, о нем узнали лишь 1972 году, а компилятор данного языка был разработан лишь в 2000 году. Возможно, при другом развитии событий, Планкалкюль имел бы возможность не просто ускорить развитие программирования, но даже изменить его эволюцию в целом. Это предположение имеет место быть, поскольку данный язык является довольно богатым для 40-х годов: в нём имеются циклы «for» и «while», условные конструкции, массивы и даже кортежи! К тому же, в этом языке есть возможность описывать и вызывать подпрограммы, однако до рекурсии Конрад Цузе не добрался.

В связи с тем, что Планкалкюль долго оставался недоступным, а потребность в высокоуровневых языках уже возникла, компания IBM разработала другой язык, принадлежащий к этому классу, — «FORTRAN (FORmula TRANslator, с английского — «преобразование формул»), компилятор для которого впервые появился в апреле 1957 году.» [2, стр. 65] Язык возник в связи с необходимостью выполнения расчётов в физике, математике, астрономии, инженерии и экономике, поэтому он лучше всего работает с числами, в том числе позволяет выполнять операции с плавающей точкой. Фортран и в настоящее время используется учёными для расчётов, так как на нем реализовано очень много программ, а также имеется большое количество полезных библиотек.

Вторым старейшим языком можно назвать LISP (List Information Symbol Processing), который Дж. Маккарти предложил для удобной работы со строками в 1962 году. LISP, как и FORTRAN, используется по сей день, но в аналитических и экспертных системах.

Кроме того, очень много языков высокого уровня появилось для решения каких-либо задач, например, COBOL (Common business Orientated Language) —

для решения бизнес-задач; всем известный со школьных времен BASIC (Beginners All-purpose Symbolic Instruction Code), позволяющий с лёгкостью дать основы программирования и многие другие. Однако стоит больше внимания уделить тем языкам, которые в большей степени влияли на эволюцию программирования, в частности, язык Никлауса Вирта — Pascal, разработанный 1968-1969 годах, в основе которого лежит структурное программирование.

Структурное программирование — методология разработки программного обеспечения, основанная на том, что код любой программы должен быть представлен в виде иерархической структуры. В рамках данной методологии разработка программы ведётся пошагово, «сверху вниз». Помимо структуризации кода в структурном программировании уделяется внимание данным, например, возникло такое понятие, как локальная переменная. Но самым большим достижением при работе с данными является объединение разнородных типов данных в единое целое — в структуру. Структура — составной тип данных, построенный с использованием других типов данных.

Появление такого программирования обусловлено возрастанием сложности решаемых задач с помощью компьютеров, и соответственно, усложнения программного обеспечения.

Основоположником структурного программирования является нидерландский учёный Эдсгер Вибе Дейкстра. Всё началось в 1968 году с его письма «Оператор GOTO считается вредным» («GOTO considered harmful»). Из названия очевидно, что Дейкстра был противником GOTO и считал, что качество исходного кода обратно пропорционально количеству операторов GOTO в нём. Более того, он сформулировал, что можно обойтись без GOTO, используя три базовые конструкции: последовательность, цикл и ветвление. Его вклад на этом далеко не заканчивается. Кроме структурного программирования он сформулировал такие значимые понятия, которыми, к сожалению, пренебрегают. Программирование, по своей сути, — сложная инженерная и научная деятельность, а попытки превратить программирование

в доступное занятие для каждого обречены на провал, существует лишь возможность освободить разработчиков от части рутиной работы.

Кроме этого, ещё давно Эдсгер Дейкстра понял, что схема разработки: «написать программу — протестировать — найти ошибки — исправить ошибки — протестировать — т.д.» является неправильной, однако она до сих пор используется большинством современных программистов. Гарантировать корректность программ на основе проведения тестирования нельзя, и этот факт является доказательством того, что данная схема имеет существенный недостаток.

Вклад Дейкстра неоценим, подтверждение этому мы видим в словах Бертрана Мейера: «Революция во взглядах на программирование, начатая Дейкстрой, привела к движению, известному как структурное программирование, которое предложило систематический, рациональный подход к конструированию программ. Структурное программирование стало основой всего, что сделано в методологии программирования, включая и объектное программирование.» [3, стр. 208]

Возвращаясь к языкам программирования, стоит упомянуть, что в Pascal впервые были предложены процедуры и функции, столь необходимые при многоразовом выполнении задач: при таком подходе повторяющегося кода становиться гораздо меньше. Помимо этого в Pascal впервые была реализована проверка типов, что позволило выявлять большое количество ошибок на этапе компиляции. Таким образом, Pascal породил новый вид языков программирования — сильно типизированных.

После Pascal появлялось много других высокоуровневых языков программирования, но больше всего выделяется язык C, так как его синтаксис стал основой для многих языков программирования, таких как C++, C#, Java и D. Появление C датируется 1972 годом, он разрабатывался в лабораториях Bell Labs двумя сотрудниками Деннисом Ритчи и Кеном Томпсоном и изначально предназначался для написания операционной системы UNIX. Впоследствии он стал языком общего назначения, а его популярность высока до сих пор.

Основным преимуществом C является оптимальность написанных на нём программ. Это связано с тем, что конструкции языка эффективно сопоставляются с машинными инструкциями.

Безусловно, структурное программирование дало много возможностей и преимуществ разработчикам. Однако при увеличении кода программ имеющихся возможностей стало не хватать, поэтому потребовался новый виток в развитии языков программирования, речь о котором пойдёт в следующей главе.

ГЛАВА III ООП и другие современные подходы

Следующим этапом в развитии языков программирования является появление объектно-ориентированного программирования (ООП). «Объектно-ориентированное программирование — это метод программирования, основанный на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определенного класса, а классы являются членами определенной иерархии наследования.» [4, стр. 69] Для большего понимания хотелось бы представить следующие важные аспекты приведенного ранее определения: «1) объектно-ориентированное программирование использует в качестве основных логических конструктивных элементов объекты, а не алгоритмы; 2) каждый объект является экземпляром (instance) определенного класса (class); 3) классы образуют иерархии. Программа считается объектно-ориентированной, только если выполнены все три указанных требования. В частности, программирование, не использующее наследование, называется не объектно-ориентированным, а программированием с помощью абстрактных типов данных.» [4, стр. 69]

С появлением ООП возникли следующие важные понятия, без которых не возможно понять объектно-ориентированное программирование на необходимом уровне:

1) Класс — абстрактный тип данных, содержащий в себе не только переменные, но и функции, работающие с этими переменными.
2) Объект — экземпляр класса.
3) Наследование — механизм, позволяющий на основе уже существующего класса создать новый класс.
4) Термин инкапсуляция носит двоякий смысл, поскольку в разных языках программирования имеет разные трактовки:

- языковая конструкция, позволяющая связывать данные с методами, которые предназначены для обработки этих же данных.
- механизм, позволяющий ограничить доступ компонентов программы к другим компонентам.

5) «Полиморфизм — способность функции обрабатывать данные разных типов.» [5, стр. 3, «перевод с англ. автора»]
6) «Абстракция — популярная и в общем неверно определяемая техника программирования. Фундаментальная идея состоит в разделении несущественных деталей реализации подпрограммы и характеристик, существенных для корректного ее использования. Такое разделение может быть выражено через специальный «интерфейс», сосредотачивающий описание всех возможных применений программы.» [6, стр. 3]

Первым языком программирования, в котором были предложены идеи ООП, является Simula, разработанный в 1967 году норвежскими учёными Кристеном Нюгордом и Оле-Йоханом Далем для моделирования сложных систем. Для 67-ого года в языке были применены революционные идеи, такие как классы, объекты и виртуальные методы. Благодаря этим идеям, язык значительно опередил своё время, но как часто случается с революционными прорывами, современники недооценили предлагаемые им возможности. К тому же его реализация не являлась самой эффективной, именно поэтому он не выдержал конкуренции и уступил место языкам того времени, прежде всего Фортрану.

Следует заметить, что большинство промышленных языков программирования, такие как C++, Java, C#, Delphi, основаны на объектной модели Simula, поэтому не стоит недооценивать вклад языка Simula в эволюцию программирования.

Что касается первого широко распространенного объектно-ориентированного языка программирования,то им является Smalltalk, который

разрабатывался в 70-х годах прошлого столетия группой программистов в компании Xerox PARC под руководством Алана Кэйема.

Smalltalk внёс большой вклад в развитие программирования, например, в сообществе данного языка появились такие вещи, как рефакторинг кода, шаблоны проектирования и экстремальное программирование, которые детально будут описаны ниже. Кроме этого, Smalltalk оказал большое влияние как на известные всем Objective-C, Ruby, Python, так и на менее известные Groovy, Actor, Erlang,

Стоит обратить внимание на такое интересное явление, как шаблон проектирования. Это конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. В наше время при разработке программного обеспечения всё чаще применяют данную технологию, так как она позволяет снизить сложность разработки ПО за счёт готовых абстракций. Зачастую хорошо спроектированные шаблоны могут применяться при разработке совершенно иного программного обеспечения. К сожалению, без минусов в данной ситуации тоже не обойтись: неопытные программисты, изучив шаблоны, начинают повсеместно их применять, что заметно усложняет исходный код.

Для повышения продуктивности разработчиков применяют методы экстремального программирования, например, используют автоматическое тестирование, парное программирование: когда один программист пишет код, а другой вслед за ним исправляет ошибки. Появление данной методологии подтверждает, что вместе с развитием языков программирования менялся и способ разработки в целом, и, как следствие, возникала некоторая культура программирования.

Возвращаясь к языкам программирования, стоит упомянуть, что Smalltalk имел следующие существенные недостатки: большие требования к памяти и низкая производительность получаемых на нём программ, из-за которых и проигрывал появляющимся в 1980-х годах C++ и Ada, в которых ООП было более эффективно реализовано.

C++ удостаивается большого внимания, так как он является одним из самых распространенных языков программирования. «C++ возник в начале 1980-х годов, когда сотрудник фирмы Bell Labs Бьёрн Страуструп придумал ряд усовершенствований к языку C под собственные нужды.» [7, стр. 46] Отсюда и название языка, ведь «++» является оператором увеличения переменной на единицу в C, то есть C++ — это расширение C. Изначально C++ отличался от C лишь возможностью использования классов, но к 1985 году состоялся коммерческий выпуск языка, в котором поддерживались такие возможности, как виртуальные функции, ссылки, перегрузка функций, константы и многое другое. C++ развивается до сих, в него добавляется много новых функций и возможностей. Самой последней модификацией на 2015 год является C++14.

Кроме ранее упомянутых языков, существует множество других, которых на данный момент насчитывается порядка нескольких тысяч. Это обусловлено прежде всего тем, что для решения задачи создают наиболее удобный язык. Но основная тенденция в развитии связана с повышением уровня абстракции. В последнее время добавляется очень много новых функций, начиная от добавления специальных типов данных, и заканчивая поддержкой системных технологий, например, работа с сетью.

К сожалению, если мы будем подробно останавливаться на истории развития каждого известного науке языка программирования, это займет гораздо больше объема, чем предполагает реферат.

Подводя промежуточный итог, можно сказать, что ООП положило начало новым возможностям в программировании: стало возможным разбивать программы на классы и тестировать отдельно каждый из них, а не всю программу в целом. Также ООП позволяет при написании кода значительно абстрагироваться, что заметно снижает количество ошибок. Разумеется, при высоком уровне абстракции понижается эффективность, однако это не столь критично, поскольку в настоящее время человечество имеет довольно мощные и постоянно развивающиеся компьютеры.

Во время распространения ООП, в 90-х гг. XX века впервые возникло такое понятие, как сверхвысокоуровневое программирование. Оно развивало идею, что для разработки программ достаточно описание того, что нужно сделать, в то время, как в высокоуровневых языках программирования существует принцип, как это нужно сделать. К сверхвысокоуровневым языкам относятся языки с очень высоким уровнем абстракции, такие как Icon, Perl, Ruby и Haskell.

Абсолютно новой и самой последней тенденцией в развитии языков программирования является появление ультра-высокоуровневых языков программирования. Языки такого типа имеют различные дополнительные объекты, ориентированные на прикладное использование. Такие объекты требуют минимальной настройки и практически сразу готовы к использованию. Подобный подход из-за уменьшения разрабатываемых исходных кодов способствует снижению как временных затрат при разработке программного обеспечения, так и количества возможных ошибок.

ЗАКЛЮЧЕНИЕ

В качестве заключения, стоит еще раз отметить, что языки программирования имели бурное развитие за достаточно короткий промежуток времени. Можно выделить 3 основных этапа в развитии языков программирования:

1) Появление низкоуровневых языков:
    a) Assembler;
2) Появление высокоуровневых языков программирования:
    a) FORTRAN;
    b) C;
    c) ALGOL;
    d) COBOL и др.
3) Возникновение объектно-ориентированного программирования:
    a) Simula;
    b) Smalltalk;
    c) C++;
    d) Python и др.

Пик развития программирования пришёлся на 80-90-е гг. XX столетия. В последнее время языки программирования меняются в основном в вопросе увеличения своей функциональности и добавления вспомогательных возможностей. Однако существуют и кардинальные изменения, связанные в основном с увеличением уровня абстракции, из-за чего снижается и количество кода при написании программного обеспечения, и количество допускаемых программистом ошибок. Безусловно, это понижает эффективность получаемых программ, однако эволюция компьютеров не стоит на месте, поэтому способна компенсировать новыми доступными мощностями подобное понижение эффективности.

С моей точки зрения как программиста, надеюсь, что настоящее и будущее развитие программирования не приведёт к снижению требований к разработчикам программного обеспечения настолько, что работодателям выгоднее будет нанять нескольких низкоквалифицированных специалистов, нежели одного профессионала. Как следствие, не только упадёт уровень программистов в целом, но и понизится качество создаваемых программных продуктов. Поэтому хотелось бы, чтобы люди, влияющие на развитие языков программирования, всегда помнили и не пренебрегали мнением Эдсгера Дейкстра, который предостерегал от попыток превратить разработку ПО в некий тривиальный процесс.

СПИСОК ЛИТЕРАТУРЫ

1. Иллингуорт В. «Толковый словарь по вычислительным системам» — Пер. с англ. А. К. Белоцкого и др. — М.: Машиностроение, 1990.
2. Роберт У. Себеста. «Основные концепции языков программирования» — 5-е изд.. — М.: Вильямс, 2001.
3. Мейер Б. «Почувствуй класс. Учимся программировать хорошо с объектами и контрактами.» — Пер. с англ. — М.: Национальный открытый университет ИНТУИТ: БИНОМ. Лаборатория знаний, 2011.
4. Гради Буч. «Объектно-ориентированный анализ и проектирование с примерами приложений на C++» — 2-е изд. — Пер. с англ. — М.: Бином, Невский Диалект, 1998.
5. Cardelli L. «Typeful Programming» — 1991.
6. Страуструп Б. «Абстракция данных в языке C++».
7. Страуструп Б. «Язык программирования C++» — Пер. с англ. — М.: Бином, Невский Диалект, 1999.