ГОСУДАРСТВЕННЫЙ КОМИТЕТ СВЯЗИ, ИНФОРМАТИЗАЦИИ И ТЕЛЕКОММУНИКАЦИОННЫХ ТЕХНОЛОГИИ РЕСПУБЛИКИ УЗБЕКИСТАН

НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Факультет _____*«Компьютер инжиниринга»*_____

Направление_____*Телекоммуникационные технологии*_____

КУРСОВАЯ РАБОТА

По предмету _____*« Программирования на языке С++»*_____

На тему: _____*«Языки программирования и их особенности»*_____

Выполнил(а): _*студент 2[В] курса Лазарев А.*_

Принял(а): ____*Ядгаров Ш.*___

Нукус 2014 г.

Языки программирования и их особенности

План:

Введение

Язык программирования — формальная знаковая система, предназначенная для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель (обычно — ЭВМ) под её управлением.

Со времени создания первых программируемых машин человечество придумало более восьми тысяч языков программирования (включая нестандартные, визуальные и эзотерические языки). Каждый год их число увеличивается. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей. Профессиональные программисты могут владеть десятком и более разных языков программирования.

Язык программирования предназначен для написания компьютерных программ, которые представляют собой набор правил, позволяющих компьютеру выполнить тот или иной вычислительный процесс, организовать управление различными объектами, и т. п. Язык программирования отличается от естественных языков тем, что предназначен для взаимодействия человека с ЭВМ, в то время как естественные языки используются для общения людей между собой. Большинство языков программирования использует специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

Историческая справка

Ранние этапы развития

Можно сказать, что первые языки программирования возникали еще до появления современных электронных вычислительных машин: уже в XIX веке были изобретены устройства, которые можно с долей условности назвать программируемыми — к примеру, механические пианино и ткацкие станки. Для управления ими использовались наборы инструкций, которые в рамках современной классификации можно считать прототипами предметно-ориентированных языков программирования. Значимым можно считать «язык», на котором леди Ада Августа графиня Лавлейс написала программу для вычисления чисел Бернулли для Аналитической машины Чарльза Бэббиджа, ставшей бы, в случае реализации, первым компьютером — хотя и механическим, с паровым двигателем — в мире.

В 1930—1940 годах, А. Чёрч, А. Тьюринг, А. Марков в СССР разработали математические абстракции (лямбда-исчисление, машину Тьюринга, нормальные алгорифмы) соответственно — для формализации алгоритмов.

В это же время, в 1940-е годы, появились электрические цифровые компьютеры и был разработан язык, который можно считать первым высокоуровневым языком программирования для ЭВМ — «Plankalkül», созданный немецким инженером К. Цузе в период с 1943 по 1945 годы.

Программисты ЭВМ начала 1950-х годов, в особенности таких, как UNIVAC и IBM 701, при создании программ пользовались непосредственно машинным кодом, запись программы на котором состояла из единиц и нулей и который принято считать языком программирования первого поколения (при этом разные машины разных производителей использовали различные коды, что требовало переписывать программу при переходе на другую ЭВМ). Вскоре на смену такому методу программирования пришло применение языков второго поколения, также ограниченных спецификациями конкретных машин, но более простых для использования человеком за счет использования мнемоник (символьных обозначений машинных команд) и возможности сопоставления имен адресам в машинной памяти. Они традиционно известны под наименованием языков ассемблера и автокодов. Однако, при использовании ассемблера становился необходимым процесс перевода программы на язык машинных кодов перед ее выполнением, для чего были разработаны специальные программы, также получившие название ассемблеров. Сохранялись и проблемы с переносимостью программы с ЭВМ одной архитектуры на другую, и

необходимость для программиста при решении задачи мыслить терминами «низкого уровня» — ячейка, адрес, команда. Позднее языки второго поколения были усовершенствованы: в них появилась поддержка макрокоманд.

С середины 1950-х начали появляться языки третьего поколения, такие как Фортран, Лисп и Кобол. Языки программирования этого типа более абстрактны (их еще называют «языками высокого уровня») и универсальны, не имеют жесткой зависимости от конкретной аппаратной платформы и используемых на ней машинных команд. Программа на языке высокого уровня может исполняться (по крайней мере, в теории, на практике обычно имеются ряд специфических версий или диалектов реализации языка) на любой ЭВМ, на которой для этого языка имеется транслятор (инструмент, переводящий программу на язык машины, после чего она может быть выполнена процессором).

Обновленные версии перечисленных языков до сих пор имеют хождение в разработке программного обеспечения, и каждый из них оказал определенное влияние на последующее развитие языков программирования. Тогда же, в конце 1950-х годов, появился Алгол, также послуживший основой для ряда дальнейших разработок в этой сфере. Необходимо заметить, что на формат и применение ранних языков программирования в значительной степени влияли интерфейсные ограничения.

Совершенствование

В период 1960-х — 1970-х годов были разработаны основные парадигмы языков программирования, используемые в настоящее время, хотя во многих аспектах этот процесс представлял собой лишь улучшение идей и концепций, заложенных еще в первых языках третьего поколения.

Язык APL оказал влияние на функциональное программирование и стал первым языком, поддерживавшим обработку массивов.

Язык ПЛ/1 (NPL) был разработан в 1960-х годах как объединение лучших черт Фортрана и Кобола.

Язык Симула, появившийся примерно в это же время, впервые включал поддержку объектно-ориентированного программирования. В середине 1970-х группа специалистов представила язык Smalltalk, который был уже всецело объектно-ориентированным.

В период с 1969 по 1973 годы велась разработка языка Си, популярного и по сей день и ставшего основой для множества последующих языков, например, столь популярных, как C++ и Java.

В 1972 году был создан Пролог — наиболее известный (хотя и не первый, и далеко не единственный) язык логического программирования.

В 1973 году в языке ML была реализована расширенная система полиморфной типизации, положившая начало типизированным языкам функционального программирования.

Каждый из этих языков породил по семейству потомков, и большинство современных языков программирования в конечном счете основано на одном из них.

Кроме того, в 1960—1970-х годах активно велись споры о необходимости поддержки структурного программирования в тех или иных языках. В частности, голландский специалист Э. Дейкстра выступал в печати с предложениями о полном отказе от использования инструкций GOTO во всех высокоуровневых языках. Развивались также приемы, направленные на сокращение объема программ и повышение продуктивности работы программиста и пользователя.

Объединение и развитие

В 1980-е годы наступил период, который можно условно назвать временем консолидации. Язык C++ объединил в себе черты объектно-ориентированного и системного программирования, правительство США стандартизировало язык Ада, производный от Паскаля и предназначенный для использования в бортовых системах управления военными объектами, в Японии и других странах мира осуществлялись значительные инвестиции в изучение перспектив так называемых языков пятого поколения, которые включали бы в себя конструкции логического программирования. Сообщество функциональных языков приняло в качестве стандарта ML и Лисп. В целом этот период характеризовался скорее опорой на заложенный в предыдущем десятилетии фундамент, нежели разработкой новых парадигм.

Важной тенденцией, которая наблюдалась в разработке языков программирования для крупномасштабных систем, было сосредоточение на применении модулей — объемных единиц организации кода. Хотя некоторые языки, такие, как ПЛ/1, уже поддерживали соответствующую функциональность, модульная система нашла свое отражение и применение также и в языках Модула-2, Оберон,Ада и ML. Часто модульные системы объединялись с конструкциями обобщенного программирования.

Важным направлением работ становятся визуальные (графические) языки программирования, в которых процесс «написания» программы как текста заменяется на процесс «рисования» (конструирования программы в виде диаграммы) на экране ЭВМ. Визуальные языки обеспечивают наглядность и лучшее восприятие логики программы человеком.

В 1990-х годах в связи с активным развитием Интернета распространение получили языки, позволяющие создавать сценарии для веб-страниц — главным образом Perl, развившийся из скриптового инструмента для Unix-систем, и Java. Возрастала также и популярность технологий виртуализации. Эти изменения, однако, также не представляли собой фундаментальных новаций, являясь скорее совершенствованием уже существовавших парадигм и языков (в последнем случае — главным образом семейства Си).

В настоящее время развитие языков программирования идет в направлении повышения безопасности и надежности, создания новых форм модульной организации кода и интеграции с базами данных.

Стандартизация языков программирования

Язык программирования может быть представлен в виде набора спецификаций, определяющих его синтаксис и семантику.

Для многих широко распространённых языков программирования созданы международные стандарты. Специальные организации проводят регулярное обновление и публикацию спецификаций и формальных определений соответствующего языка. В рамках таких комитетов продолжается разработка и модернизация языков программирования и решаются вопросы о расширении или поддержке уже существующих и новых языковых конструкций.

Типы данных

Современные цифровые компьютеры являются двоичными и данные хранят в двоичном (бинарном) коде (хотя возможны реализации и в других системах счисления). Эти данные как правило отражают информацию из реального мира (имена, банковские счета, измерения и др.), представляющую высокоуровневые концепции.

Особая система, по которой данные организуются в программе, — это система типов языка программирования; разработка и изучение систем типов известна под названием теория типов. Языки можно поделить на имеющие статическую типизацию и динамическую типизацию, а также бестиповые языки (например, Forth).

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. Иногда динамически типизированные языки называют латентно типизированными.

Структуры данных

Системы типов в языках высокого уровня позволяют определять сложные, составные типы, так называемые структуры данных. Как правило, структурные типы данных образуются как декартово произведение базовых (атомарных) типов и ранее определённых составных типов.

Основные структуры данных (списки, очереди, хеш-таблицы, двоичные деревья и пары) часто представлены особыми синтаксическими конструкциями в языках высокого уровня. Такие данные структурируются автоматически.

Семантика языков программирования

Существует несколько подходов к определению семантики языков программирования.

Наиболее широко распространены разновидности следующих трёх: операционного, деривационного (аксиоматического) и денотационного (математического).

При описании семантики в рамках операционного подхода обычно исполнение конструкций языка программирования интерпретируется с помощью некоторой воображаемой (абстрактной) ЭВМ.

Аксиоматическая (Деривационная) семантика описывает последствия выполнения конструкций языка с помощью языка логики и задания пред- и постусловий.

Денотационная семантика оперирует понятиями, типичными для математики — множества, соответствия, а также суждения, утверждения и др.

Парадигма программирования

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

Несмотря на то, что большинство языков ориентировано на императивную модель вычислений, задаваемую фон-неймановской архитектурой ЭВМ, существуют и другие подходы. Можно упомянуть языки со стековой вычислительной моделью (Форт, Factor, PostScript и др.), а также функциональное (Лисп, Haskell, ML, F#, РЕФАЛ, основанный на модели вычислений, введённой советским математиком А. А. Марковым-младшим и др.) и логическое программирование (Пролог).

В настоящее время также активно развиваются декларативные и визуальные языки программирования, а также методы и средства разработки проблемно-специфичных языков (см. Языково-ориентированное программирование).

Способы реализации языков

Языки программирования могут быть реализованы как компилируемые и интерпретируемые.

Программа на компилируемом языке при помощи компилятора (особой программы) преобразуется (компилируется) в машинный код (набор инструкций) для данного типа процессора и далее собирается в исполнимый

модуль, который может быть запущен на исполнение как отдельная программа. Другими словами, компилятор переводит исходный текст программы с языка программирования высокого уровня в двоичные коды инструкций процессора.

Если программа написана на интерпретируемом языке, то интерпретатор непосредственно выполняет (интерпретирует) исходный текст без предварительного перевода. При этом программа остаётся на исходном языке и не может быть запущена без интерпретатора. Процессор компьютера, в этой связи, можно назвать интерпретатором для машинного кода.

Разделение на компилируемые и интерпретируемые языки является условным. Так, для любого традиционно компилируемого языка, как, например, Паскаль, можно написать интерпретатор. Кроме того, большинство современных «чистых» интерпретаторов не исполняют конструкции языка непосредственно, а компилируют их в некоторое высокоуровневое промежуточное представление (например, с разыменованием переменных и раскрытием макросов).

Для любого интерпретируемого языка можно создать компилятор — например, язык Лисп, изначально интерпретируемый, может компилироваться без каких бы то ни было ограничений. Создаваемый во время исполнения программы код может так же динамически компилироваться во время исполнения.

Как правило, скомпилированные программы выполняются быстрее и не требуют для выполнения дополнительных программ, так как уже переведены на машинный язык. Вместе с тем, при каждом изменении текста программы требуется её перекомпиляция, что замедляет процесс разработки. Кроме того, скомпилированная программа может выполняться только на том же типе компьютеров и, как правило, под той же операционной системой, на которую был рассчитан компилятор. Чтобы создать исполняемый файл для машины другого типа, требуется новая компиляция.

Интерпретируемые языки обладают некоторыми специфическими дополнительными возможностями (см. выше), кроме того, программы на них можно запускать сразу же после изменения, что облегчает разработку. Программа на интерпретируемом языке может быть зачастую запущена на разных типах машин и операционных систем без дополнительных усилий.

Однако интерпретируемые программы выполняются заметно медленнее, чем компилируемые, кроме того, они не могут выполняться без программы-интерпретатора.

Некоторые языки, например, Java и C#, находятся между компилируемыми и интерпретируемыми. А именно, программа компилируется не в машинный язык, а в машинно-независимый код низкого уровня, байт-код. Далее байт-код выполняется виртуальной машиной. Для выполнения байт-кода обычно используется интерпретация, хотя отдельные его части 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# — Common Language Runtime.

Подобный подход в некотором смысле позволяет использовать плюсы как интерпретаторов, так и компиляторов. Следует упомянуть, что есть языки, имеющие и интерпретатор, и компилятор (Форт).

Языки программирования низкого уровня

Первые компьютеры приходилось программировать двоичными машинными кодами. Однако программировать таким образом — довольно трудоемкая и тяжелая задача. Для упрощения этой задачи начали появляться языки программирования низкого уровня, которые позволяли задавать машинные команды в понятном для человека виде. Для преобразования их в двоичный код были созданы специальные программы — трансляторы.

Трансляторы делятся на:

Компиляторы — превращают текст программы в машинный код, который можно сохранить и после этого использовать уже без компилятора (примером является исполняемые файлы с расширением *.exe) .

Интерпретаторы — превращают часть программы в машинный код, выполняют его и после этого переходят к следующей части. При этом каждый раз при выполнении программы используется интерпретатор .

Примером языка низкого уровня является ассемблер. Языки низкого уровня ориентированы на конкретный тип процессора и учитывают его особенности, поэтому для переноса программы на ассемблере на другую аппаратную платформу ее нужно почти полностью переписать. Определенные различия есть и в синтаксисе программ под разные компиляторы. Правда, центральные процессоры для компьютеров фирм AMD и Intel практически совместимы и отличаются лишь некоторыми специфическими командами. А вот специализированные процессоры для других устройств, например, видеокарт и телефонов содержат существенные различия.

Языки низкого уровня, как правило, используют для написания небольших системных программ, драйверов устройств, модулей стыков с нестандартным оборудованием, программирование специализированных микропроцессоров, когда важнейшими требованиями являются компактность, быстродействие и возможность прямого доступа к аппаратным ресурсам.

Ассемблер — язык низкого уровня, широко применяется до сих пор.

Языки программирования высокого уровня

Особенности конкретных компьютерных архитектур в них не учитываются, поэтому созданные приложения легко переносятся с компьютера на компьютер. В большинстве случаев достаточно просто перекомпилировать программу под определенную компьютерную архитектурную и операционную систему. Разрабатывать программы на таких языках значительно проще и ошибок допускается меньше. Значительно сокращается время разработки программы, что особенно важно при работе над большими программными проектами .

Сейчас в среде разработчиков считается, что языки программирования, которые имеют прямой доступ к памяти и регистров или имеют ассемблерные вставки, нужно считать языками программирования с низким уровнем абстракции. Поэтому большинство языков, считавшихся языками высокого уровня до 2000 года сейчас уже таковыми не считаются.

Адресный язык программирования
Фортран
Кобол
Алгол
Pascal
Pascal ABC
Java
C
Basic
C++
Objective-C
Smalltalk
C#
Delphi

Недостатком некоторых языков высокого уровня является большой размер программ в сравнении с программами на языках низкого уровня. С другой стороны, для алгоритмически и структурно сложных программ при использовании суперкомпиляции преимущество может быть на стороне языков высокого уровня. Сам текст программ на языке высокого уровня меньше, однако, если взять в байтах, то код, изначально написанный на ассемблере, будет более компактным. Поэтому в основном языки высокого уровня используются для разработки программного обеспечения компьютеров и устройств, которые имеют большой объем памяти. А разные подвиды ассемблера применяются для программирования других устройств, где критичным является размер программы.

Используемые символы

Современные языки программирования рассчитаны на использование ASCII, то есть доступность всех графических символов ASCII является необходимым и достаточным условием для записи любых конструкций языка. Управляющие символы ASCII используются ограниченно: допускаются только возврат каретки CR, перевод строки LF и горизонтальная табуляция HT (иногда также вертикальная табуляция VT и переход к следующей странице FF).

Подробнее по этой теме см.: Переносимый набор символов.

Ранние языки, возникшие в эпоху 6-битных символов, использовали более ограниченный набор. Например, алфавит Фортрана включает 49 символов (включая пробел): A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 = + - * / () . , $ ' :

Заметным исключением является язык APL, в котором используется очень много специальных символов.

Использование символов за пределами ASCII (например, символов KOI8-R или символов Юникода) зависит от реализации: иногда они разрешаются только в комментариях и символьных/строковых константах, а иногда и в идентификаторах. В СССР существовали языки, где все ключевые слова писались русскими буквами, но большу́ю популярность подобные языки не завоевали (исключение составляет Встроенный язык программирования 1С:Предприятие).

Подробнее по этой теме см.: Языки программирования с ключевыми словами не на английском.

Расширение набора используемых символов сдерживается тем, что многие проекты по разработке программного обеспечения являются международными. Очень сложно было бы работать с кодом, где имена одних переменных записаны русскими буквами, других — арабскими, а третьих — китайскими иероглифами. Вместе с тем, для работы с текстовыми данными языки программирования нового поколения (Delphi 2006, C#, Java) поддерживают Unicode.

Категории языков программирования

Функциональные
Процедурные (императивные)
Стековые
Аспектно-ориентированные
Декларативные
Динамические
Учебные
Описания интерфейсов

Прототипные
Объектно-ориентированные
Рефлексивные (то есть поддерживающие отражение)
Логические
Скриптовые (сценарные)
Эзотерические

Функциональное программирование

Функциональное программирование объединяет разные подходы к определению процессов вычисления на основе достаточно строгих абстрактных понятий и методов символьной обработки данных. Сформулированная Джоном Мак-Карти (1958) концепция символьной обработки информации компьютером восходит к идеям Черча и других математиков, известным как лямбда-исчисление с конца 20-х годов XX века. Выбирая лямбда-исчисление как теоретическую модель, Мак-Карти предложил рассматривать функции как общее базовое понятие, к которому достаточно естественно могут быть сведены все другие понятия, возникающие при программировании. Существуют различия в понимании функции в математике и функции в программировании, вследствие чего нельзя отнести Си-подобные языки к функциональным, использующим менее строгое понятие. Функция в математике не может изменить вызывающее её окружение и запомнить результаты своей работы, а только предоставляет результат вычисления функции.

Программирование с использованием математического понятия функции вызывает некоторые трудности, поэтому функциональные языки, в той или иной степени предоставляют и императивные возможности, что ухудшает дизайн программы (например возможность безболезненных дальнейших изменений). Дополнительное отличие от императивных языков программирования заключается в декларативности описаний функций. Тексты программ на функциональных языках программирования описывают «как решить задачу», но не предписывают последовательность действий для решения. Первым спроектированным функциональным языком стал Лисп. Он был предложен Джоном Мак-Карти в качестве средства исследования границ применимости компьютеров, в частности, методом решения задач искусственного интеллекта. Лисп послужил эффективным инструментом экспериментальной поддержки теории программирования и развития сферы его применения. Вариант данного языка широко используется в системе автоматизированного проектирования AutoCAD и называется AutoLISP

В качестве основных свойств функциональных языков программирования обычно рассматриваются следующие:

краткость и простота;

Программы на функциональных языках обычно намного короче и проще, чем те же самые программы на императивных языках.

Пример (быстрая сортировка Хоара на абстрактном функциональном языке):

quickSort ([]) = []

quickSort ([h : t]) = quickSort (n | n t, n <= h) + [h] + quickSort (n | n t, n > h)

строгая типизация;

В функциональных языках большая часть ошибок может быть исправлена на стадии компиляции, поэтому стадия отладки и общее время разработки программ сокращаются. Вдобавок к этому строгая типизация позволяет компилятору генерировать более эффективный код и тем самым ускорять выполнение программ.

модульность;

Механизм модульности позволяет разделять программы на несколько сравнительно независимых частей (модулей) с чётко определёнными связями между ними. Тем самым облегчается процесс проектирования и последующей поддержки больших программных систем. Поддержка модульности не является свойством именно функциональных языков программирования, однако поддерживается большинством таких языков.

функции — объекты вычисления;

В функциональных языках (равно как и вообще в языках программирования и математике) функции могут быть переданы другим функциям в качестве аргумента или возвращены в качестве результата. Функции, принимающие функциональные аргументы, называются функциями высших порядков или функционалами.

чистота (отсутствие побочных эффектов);

В чистом функциональном программировании оператор присваивания отсутствует, объекты нельзя изменять и уничтожать, можно только создавать новые путем декомпозиции и синтеза существующих. О ненужных объектах позаботится встроенный в язык сборщик мусора. Благодаря этому в чистых функциональных языках все функции свободны от побочных эффектов.

отложенные (ленивые) вычисления.

В традиционных языках программирования (например, C++) вызов функции приводит к вычислению всех аргументов. Этот метод вызова функции называется вызов-по-значению. Если какой-либо аргумент не использовался в функции, то результат вычислений пропадает, следовательно, вычисления были произведены впустую. В каком-то смысле противоположностью вызова-по-значению является вызов-по-необходимости (ленивые вычисления). В этом случае аргумент вычисляется, только если он нужен для вычисления результата.

Некоторые языки функционального программирования
Лисп
Common Lisp
Scheme
Clojure
FP
FL
Hope
Miranda
Haskell
Curry
Clean
Gofel
ML
Standard ML
Objective CAML
Harlequin's MLWorks
F#
Scala
Nemerle
Erlang
Пифагор
Рефал

Процедурное программирование

Процедурное программирование — программирование на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка.

Процедурное программирование является отражением архитектуры традиционных ЭВМ, которая была предложена фон Нейманом в 1940-х годах. Теоретической моделью процедурного программирования служит абстрактная вычислительная система под названием машина Тьюринга.

Процедурный язык программирования предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Особенность таких языков программирования состоит в том, что задачи разбиваются на шаги и решаются шаг за шагом. Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов.

Процедурные языки программирования:
Ada (язык общего назначения)
Алгол 60
Алгол 68
Basic (до появления Visual Basic)
Си

КОБОЛ
Фортран
Модула-2
HAL/S
Pascal
PureBasic
ПЛ/1
Рапира
REXX

Стековый язык программирования

Стековый язык программирования (англ. stack-oriented programming language) — это язык программирования, в котором для передачи параметров используется машинная модель стека. Этому описанию соответствует несколько языков, в первую очередь Forth и PostScript, а также многие ассемблерные языки (использующие эту модель на низком уровне — Java, C#). При использовании стека, в качестве основного канала передачи параметров между словами, элементы языка, естественным образом, образуют фразы (последовательное сцепление). Это свойство сближает данные языки с естественными языками.

Выполнение программы в стековом языке программирования представляет собой операции на одном или нескольких стеках, которые могут иметь различное предназначение. Вследствие этого программные конструкции других языков программирования должны быть изменены, прежде чем они могут быть использованы в стековом языке. Стековые языки программирования используют так называемую «обратную польскую» нотацию (англ. RPN, reverse polish notation), или постфиксную нотацию, в которой аргументы или параметры команды должны быть записаны перед самой командой. Например, в обратной польской нотации операция сложения записывается как «2 3 +», а не «+ 2 3» (префиксная или «польская» нотация) или «2 + 3» (инфиксная нотация). Это позволяет использовать, в полной мере, стековые языки при ограниченных аппаратных ресурсах памяти в контроллерах встроенных систем.

Аспектно-ориентированное программирование

Аспектно - ориентированное программирование (АОП) — парадигма программирования, основанная на идее разделения функциональности для улучшения разбиения программы на модули.

Методология АОП была предложена группой инженеров исследовательского центра Xerox PARC под руководством Грегора Кичалеса (Gregor Kiczales). Ими же было разработано аспектно-ориентированное расширение для языка Java, получившее название AspectJ — (2001 год).

Обоснование

Существующие парадигмы программирования — процедурное, модульное, объектно-ориентированное программирование (ООП) и предметно-ориентированное проектирование — предоставляют определённые способы для разделения и выделения функциональности: функции, модули, классы, но некоторую функциональность с помощью предложенных методов невозможно выделить в отдельные сущности. Такую функциональность называют сквозной (от англ. scattered — разбросанный или англ. tangled — переплетённый), так как её реализация распределена по различным модулям программы. Сквозная функциональность приводит к рассредоточенному и запутанному коду, сложному для понимания и сопровождения.

Ведение лога и обработка исключений — типичные примеры сквозной функциональности. Другие примеры: трассировка; аутентификация и проверка прав доступа; контрактное программирование (в частности, проверка пред- и постусловий). Для программы, написанной в парадигме ООП, любая функциональность, по которой не была проведена декомпозиция, является сквозной.

Основные концепции

Все языки АОП предоставляют средства для выделения сквозной функциональности в отдельную сущность. Так как AspectJ является родоначальником этого направления, используемые в этом расширении концепции распространились на большинство языков АОП.

Основные понятия АОП:

Аспект (англ. aspect) — модуль или класс, реализующий сквозную функциональность. Аспект изменяет поведение остального кода, применяя совет в точках соединения, определённых некоторым срезом.

Совет (англ. advice) — средство оформления кода, которое должно быть вызвано из точки соединения. Совет может быть выполнен до, после или вместо точки соединения.

Точка соединения (англ. join point) — точка в выполняемой программе, где следует применить совет. Многие реализации АОП позволяют использовать вызовы методов и обращения к полям объекта в качестве точек соединения.

Срез (англ. pointcut) — набор точек соединения. Срез определяет, подходит ли данная точка соединения к данному совету. Самые удобные реализации АОП используют для определения срезов синтаксис основного языка (например, в AspectJ применяются Java-сигнатуры) и позволяют их повторное использование с помощью переименования и комбинирования.

Внедрение (англ. introduction, введение) — изменение структуры класса и/или изменение иерархии наследования для добавления функциональности аспекта в инородный код. Обычно реализуется с помощью некоторого метаобъектного протокола (англ. metaobject protocol, MOP).

Декларативный язык программирования

Декларативный языки программирования — это язык программирования высокого уровня, в котором вместо пошагового алгоритма решения задачи («как» решить задачу) описывается «что» требуется получить в качестве результата. Механизм обработки сопоставления с образцом декларативных утверждений уже реализован в устройстве языка. Типичным примером таких языков являются языки логического программирования (языки, основанные на системе правил).

В программах на языках логического программирования соответствующие действия выполняются только при наличии необходимого разрешающего условия.

Характерной особенностью декларативных языков является их декларативная семантика. Основная концепция декларативной семантики заключается в том, что смысл каждого оператора не зависит от того, как этот оператор используется в программе. Декларативная семантика намного проще семантики императивных языков, что может рассматриваться как преимущество декларативных языков перед императивными.

Динамический язык

Динамический язык — язык программирования, который позволяет определять типы данных и осуществлять синтаксический анализ и компиляцию «на лету», на этапе выполнения программы. Динамические языки удобны для быстрой разработки приложений.

Динамическая типизация является основным, но не единственным критерием динамического языка программирования.

К динамическим языкам относятся: Perl, Tcl, Python, PHP, Ruby, Smalltalk, JavaScript. Некоторыми динамическими чертами обладает также Visual Basic.

Учебный язык программирования

Учебный язык программирования — язык программирования, предназначенный для обучения. В качестве таковых разрабатывались такие языки как BASIC и Паскаль. Из разработанного для обучения языка ABC вырос Python. Популярным языком, разработанным специально для образования является LOGO. Специально для российских школ разработана языковая среда КуМир. Набирает популярность созданный в Массачусетском технологическом институте язык визуального программирования Scratch и тому подобные среды программирования.

Требования к учебному языку программирования

Учебный язык должен обеспечивать простоту, ясность и удобочитаемость конструкций. Излишняя гибкость, «вседозволенность» синтаксиса может затруднить понимание программ. Не слишком хорошо подходят для обучения языки, поощряющие к использованию различных «программистских трюков». С этим связаны преимущества использования в

образовательном процессе языков семейства Pascal перед Си-подобными языками.

При выборе языка программирования не играют роль такие факторы, как его новизна, эффективность реализации (в виде компилятора или интерпретатора). Фактор распространённости имеет как психологическое значение (влияя на мотивацию учащихся), так и практическое (востребованность получаемых знаний без необходимости переучивания).

Учебный язык программирования должен обеспечивать плавный переход от псевдокода к собственно программированию. Полезным в обучении может быть возможность использования национальной лексики для ключевых слов и идентификаторов.

Альтернативой относительно трудоёмким для изучения комплексным языкам программирования общего назначения могут составить простые миниязыки, в которых, для наглядности, имеется графический исполнитель, вроде черепашки в Лого — первом и одном из самых известных таких языков.

Язык описания интерфейсов

Язык описания интерфейсов (англ. Interface Description Language или Interface Definition Language) — язык спецификаций для описания интерфейсов, синтаксически похожий на описание классов в языке C++.

Объектно-ориентированный язык программирования

Объектно-ориентированный язык программирования (ОО-язык) — язык, построенный на принципах объектно-ориентированного программирования.

В основе концепции объектно-ориентированного программирования лежит понятие объекта — некой сущности, которая объединяет в себе поля (данные) и методы (выполняемые объектом действия).

Например, объект человек может иметь поля имя, фамилия и методы есть и спать. Соответственно, в программе можем использовать операторы Человек.Имя:="Иван" и Человек.Есть(пища).

Логическое программирование

Логическое программирование — парадигма программирования, основанная на автоматическом доказательстве теорем, а также раздел дискретной математики, изучающий принципы логического вывода информации на основе заданных фактов и правил вывода. Логическое программирование основано на теории и аппарате математической логики с использованием математических принципов резолюций.

Сценарный язык

Сценарный язык (язык сценариев, жарг. скриптóвый язык, от англ. scripting language) — высокоуровневый язык программирования для написания сценариев (англ. script) — кратких описаний действий,

выполняемых системой. Разница между программами и сценариями довольно размыта. Сценарий — это программа, имеющая дело с готовыми программными компонентами.

Эзотерический язык программирования — язык программирования, разработанный для исследования границ возможностей разработки языков программирования, для доказательства потенциально возможной реализации некой идеи (так называемое «доказательство концепции», англ. proof of concept), в качестве произведения программного искусства, или в качестве шутки (компьютерного юмора).

Эзотерические язык

Многие эзотерические языки придумываются для развлечения, часто они пародируют «настоящие» или являются абсурдным воплощением «серьёзных» концепций программирования. Некоторые эзотерические языки нарочно ограничены, (как, например, язык HQ9+), другие являются тьюринг-полными, то есть языками общего назначения. Общее свойство, присущее любому эзотерическому языку — текст программы на нём понятен лишь «посвящённому», либо непонятен вообще, потому что для составления программы нужно написать программу на обычном языке. В то время, как разработчики «реальных» языков программирования стараются сделать синтаксис максимально понятным, а программирование — удобным, создатели эзотерических языков обычно ставят перед собой противоположные задачи.

На практике такие языки, как правило, бесполезны, однако программирование на некоторых из них является неплохой тренировкой. Эзотерические языки нередко включают в список разрешённых языков на конкурсах по программированию.

Классификация функциональных языков

В качестве примера чистого функционального языка можно привести Haskell. Однако большинство функциональных языков являются гибридными и содержат свойства как функциональных, так и императивных языков. Яркие примеры — языки Scala и Nemerle. В них органично сочетаются характеристики как объектно-ориентированных языков, так и функциональных. Реализована хвостовая рекурсия и её оптимизация, функция является полноправным объектом, то есть может быть сохранена в переменной, передана в качестве аргумента в другую функцию или возвращена из функции.

Также функциональные языки делят на строгие и нестрогие. К нестрогим языкам относят те, которые поддерживают отложенные вычисления (F#), то есть аргументы функции вычисляются только тогда, когда они действительно понадобятся при вычислении функции. Ярким примером нестрогого языка является Haskell. В качестве примера строгого языка можно привести Standard ML.

Некоторые функциональные языки реализованы поверх платформообразующих виртуальных машин (JVM, .NET), то есть приложения на этих языках могут работать в среде времени исполнения (JRE, CLR) и использовать встроенные классы. К ним относятся Scala, Clojure (JVM), F#, Nemerle, SML.NET (.NET).

Математически обоснованные языки программирования

Ряд известных авторов выделяют в особую категорию «языки, наследованные от математики» (англ. mathematically-derived languages). Алан Кэй также отделяет языки, являющиеся «стилем во плоти» (crystalization of style) от прочих языков, являющихся «склеиванием возможностей» (agglutination of features).

Это языки, семантика которых является непосредственным воплощением некой математической модели, незначительно адаптированной (без нарушения целостности) для того, чтобы быть более практичным языком для разработки реальных программ. Лишь некоторые языки попадают под эту категорию, большинство языков проектируются приоритетно исходя из возможности эффективной трансляции в машину Тьюринга, и имеют лишь некое подмножество в своём составе, воплощающее ту или иную математическую модель — от арифметики до средств параллелизма (например, Occam - π — это Occam, дополненный набором конструкций, воплощающих -исчисление)..

Примеры математически обоснованных языков и воплощаемых ими математических моделей:

Agda — Интуиционистская теория типов Мартин-Лёфа.

APL и его потомки (J, K) — оригинальная семантика, не имеющая названия, воплощающая нотацию Айверсона для исчисления массивов (часто встречается термин «array languages»).

Coq — исчисление индуктивных конструкций.

Erlang — исчисление процессов (первоначально в форме модели акторов, позже также построено обоснование на -исчислении).

Forth — стековая машина и конкатенативный язык программирования.

Haskell — теория категорий (включая «декартово замкнутую категорию», воплощающую лямбда-исчисление; категорию монад для моделирования побочных эффектов; расширение системы типов Хиндли — Милнера; систему родóв; и др.)

Joy — композиция функций и гомоморфизм (иначе говоря, чистый конкатенативный язык программирования, и, как следствие, чистый функциональный).

Lisp — лямбда-исчисление Чёрча (в том числе язык S-выражений, воплощающий нотацию пар чёрча).

Scheme — «облагороженный» диалект Лиспа (сильнее типизированный, в большей степени гомознаковый, ограничивающийся

гигиеническими макроопределениями и соблюдающий числовую башню), дополненный нотацией продолжений.

ML — типизированное лямбда-исчисление, то есть лямбда-исчисление, дополненное системой типов Хиндли — Милнера.

Prolog — исчисление предикатов.

Mercury — исчисление предикатов, дополненное системой типов Хиндли — Милнера.

Smalltalk — теория множеств (с соблюдением числовой башни).

SQL — исчисление кортежей (вариант реляционного исчисления, в свою очередь основанного на исчислении предикатов первого порядка)

SGML и его потомки (HTML, XML) — нотация деревьев (важный случай графов).

Unlambda — комбинаторная логика.

Регулярные выражения.

Рефал — оригинальная семантика Турчина, носящая название «Рефал-машины» или «Рефал-автомата», созданная на основе нормального алгоритма Маркова, воплощающая композицию теории автоматов, сопоставления с образцом и переписывания термов.

Наличие математического обоснования для языка может гарантировать (или, как минимум, обещать с очень высокой вероятностью) некоторые или все из следующих положительных свойств:

Существенное повышение стабильности программ. В одних случаях — за счёт упрощения формальной верикации программ, вплоть до построения доказательства корректности для самого языка (см. типо безопасность — примерами служат Standard ML и Haskell) и даже получения языка, который сам является системой доказательства (Coq, Agda). В других случаях — за счёт быстрого обнаружения ошибок на первых же пробных запусках программ (Forth и регулярные выражения).

Обеспечение потенциально более высокой эффективности программ. Даже если семантика языка далека от архитектуры целевой платформы компиляции, к нему могут быть примененимы формальные методики глобального анализа программ (хотя трудоёмкость написания даже тривиального транслятора может оказаться выше). Например, для языков Scheme и Standard ML существуют развитые глобально-оптимизирующие компиляторы (вплоть до суперкомпиляторов), результат работы которых может уверенно конкурировать по скорости с языком низкого уровня Си и даже опережать последний (хотя ресурсоёмкость работы самих компиляторов оказывается значительно выше). Одна из самых быстрых СУБД — KDB — написана на языке K. Язык Scala (унаследовавший математику от ML) обеспечивает на платформе JVM более высокую скорость, чем «родной» для неё язык Java. С другой стороны, Forth имеет репутацию одного из самых нетребовательных к ресурсам языков (менее требователен, чем Си) и используется для разработки приложений реального

времени под самые маломощные ЭВМ; кроме того, транслятор Форта является одним из наименее трудоёмких в реализации на ассемблере.

Заранее известный (неограниченный или, наоборот, чётко очерченный) предел роста сложности программных компонентов, систем и комплексов, которые можно выразить средствами этого языка с сохранением показателей качества. Языки, не имеющие математического обоснования (а именно такие наиболее часто применяются в мейнстриме: C++, Java, C#, Delphi и др.), на практике ограничивают реализуемую функциональность и/или снижают качество по мере усложнения системы, так как им присущи экспоненциальные кривые роста сложности как касательно работы одного отдельно взятого человека, так и касательно сложности управления проектом в целом. Прогнозируемая сложность системы приводит либо к поэтапной декомпозиции проекта на множество более мелких задач, каждая из которых решается соответствующим языком; либо к языково-ориентированному программированию для случая, когда адресуемой языком задачей является как раз описание семантик и/или символьные вычисления (Lisp, ML, Haskell, Рефал, Регулярные выражения). Языки с неограниченным пределом роста сложности программ нередко относят к метаязыкам (что в непосредственном толковании термина не верно, но практике сводимо, так как всякий мини-язык, выбранный для решения некоторой подзадачи в составе общей задачи, может быть представлен в виде синтаксического и семантического подмножества данного языка, не требуя трансляции).

Удобство для человека при решении задач, на которые этот язык ориентирован по своей природе (см. предметно-специфичный язык), что в некоторой степени также способно (косвенно) повлиять на повышение стабильности результирующих программ за счёт повышения вероятности обнаружения ошибок в исходном коде и снижения дублирования кода.

Следует иметь ввиду, что языки, наследованные от «наследованных от математики» уже не обязательно будут обладать этими свойствами. Например, язык Python соединяет в себе несколько упомянутых моделей, но для их совмещения не существует обоснования, поэтому он не может считаться «наследованным от математики», и, как следствие, ему присуще лишь последнее из указанных свойств.

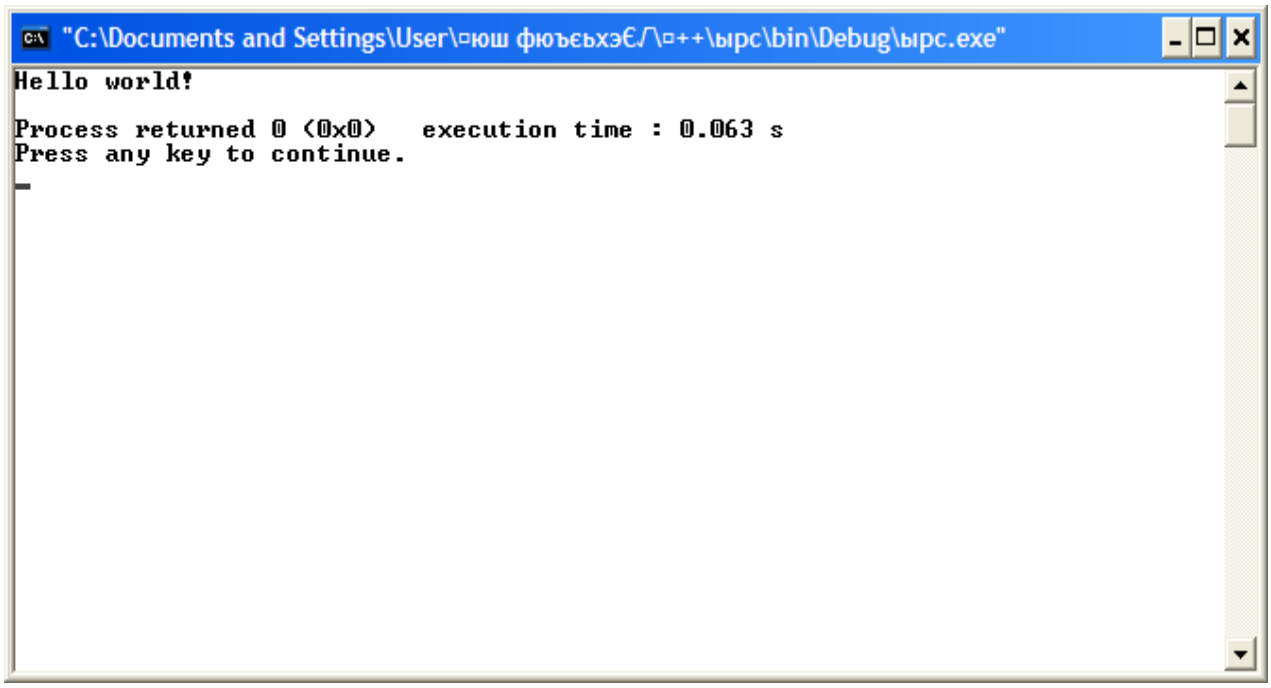
Практическая часть
Примеры на языке программирования С++

Пример 1.
Язык программирования С++ простой пример для вывода текста:

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello world!" << endl;
   return 0;
}
```



Пример 2.

Математическая задача на языке программирования C++:

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
    {
        float a,b,x;
        cin>>a>>b;
        x=sqrt(a*b) ;
        cout << x << endl;
        return 0;
    }
```

Пример 3.

Использование атрибута STYLE на HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
<title>Использование стилей</title>
</head>
<body>
<p style="color:#CC0000; background:#9999CC; word-spacing:20px;">
style="color:#CC0000; background:#9999CC;
word-spacing:20px;"
</p>
<p style="color:#66FFFF; background:#990000; word-spacing:10px;">
style="color:#66FFFF; background:#990000;
word-spacing:10px;"
</p>
<table border="1" >
<tr>
<td style="color:#66FFFF; background:#990000; word-spacing:10px;">
style="color:#66FFFF; background:#990000;
word-spacing:10px;"
</td>
</tr>
</table>
</body>
</html>
```

Здесь применяются такие же стили, как и в предыдущем примере, однако параметры задаются с помощью атрибута STYLE.

На рис. 1.1 показан результат обработки кода из листинга 1.

Результаты обработки кодов на рис1.1 абсолютно одинаковые, однако задавать стили в начале документа удобнее, так как при необходимости их легко найти и исправить.

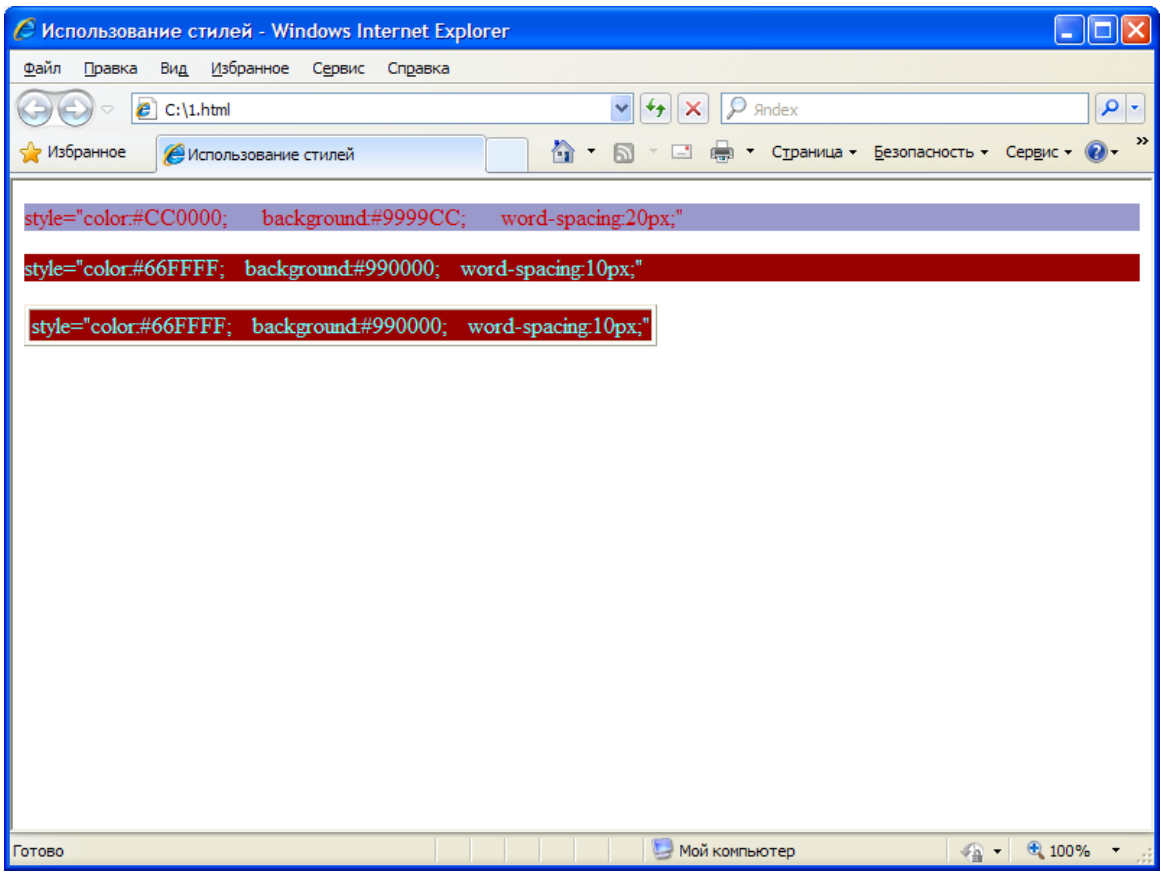


Рис. 1.1 Установка стилей через атрибут STYLE

Пример 4.

Программирования на Delphi

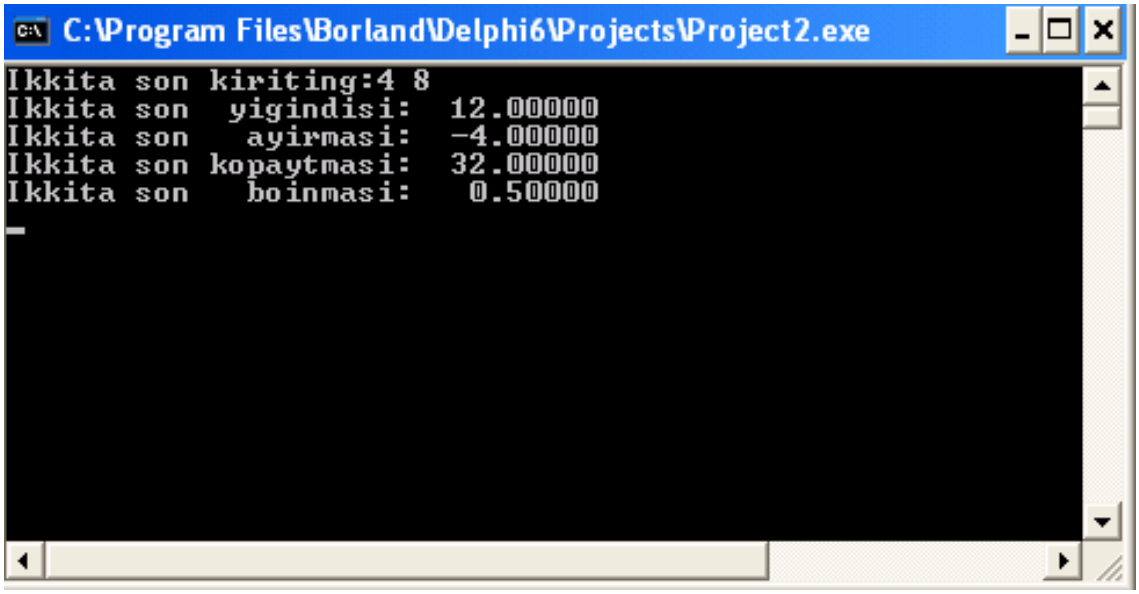
Составление программы для * / + -.

```
Program project 2;
Uses SysUtils;
  Var a,b,Rez,Real;

      Begin
         Write (‘ikkita son kiriting:);
         Readln (a,b);
         Rez=a+b;
         Writeln (‘ikkita son yigindisi:’Rez:10:5);
         Rez=a-b;
         Writeln (‘ikkita son ayirmasi:’Rez:10:5);
         Rez=a*b;
         Writeln (‘ikkita son ko’paytmasi:’Rez:10:5);
         Rez=a/b;
         Writeln (‘ikkita son buinmasi:’Rez:10:5);
         Readln;

    End.
```

После компиляции на экран выйдет окно зададим 2 любые числа после выйдет этот окно:



Заключение

Каждый 256-й день года отмечается неофициальный праздник - День программиста. Это число (два в восьмой степени) выбрано неслучайно, так как это количество чисел, которые можно выразить с помощью одного байта. Сегодня программист - одна из самых востребованных профессий, ведь значение деятельности этих специалистов невозможно переоценить: ежедневно миллионы людей сталкиваются с результатами их труда, когда работают за компьютером, пользуются телефоном и смотрят телевизор.

Литература

1. Список языков программирования (англ.). Проверено . Архивировано из первоисточника 22 августа 2011.

2. Rojas, Raúl, et al. (2000). «Plankalkül: The First High-Level Programming Language and its Implementation». Institut für Informatik, Freie Universität Berlin, Technical Report B-3/2000. (full text)

3. Linda Null, Julia Lobur, The essentials of computer organization and architecture, Edition 2, Jones & Bartlett Publishers, 2006, ISBN 0-7637-3769-0, p. 435

4. O'Reilly Media. History of programming languages (PDF). Проверено 5 октября 2006. Архивировано из первоисточника 10 мая 2013.

5. Frank da Cruz. IBM Punch Cards Columbia University Computing History.

6. Richard L. Wexelblat: History of Programming Languages, Academic Press, 1981, chapter XIV.

7. François Labelle. Programming Language Usage Graph. SourceForge. Проверено 21 июня 2006. Архивировано из первоисточника 10 мая 2013.

8. (2006) «The Semicolon Wars». American Scientist 94 (4): 299–303.

9. Tetsuro Fujise, Takashi Chikayama, Kazuaki Rokusawa, Akihiko Nakase (December 1994). «KLIC: A Portable Implementation of KL1» Proc. of FGCS '94, ICOT Tokyo, December 1994. http://www.icot.or.jp/ARCHIVE/HomePage-E.html KLIC is a portable implementation of a concurrent logic programming language KL1.

10. Jim Bender. Mini-Bibliography on Modules for Functional Programming Languages. ReadScheme.org (15 марта 2004). Проверено 27 сентября 2006. Архивировано из первоисточника 10 мая 2013.

11. 1 2 Andrew W. Appel A Critique of Standard ML. — Princeton University, revised version of CS-TR-364-92, 1992.

12. Greg Nelson Systems Programming with Modula-3. — NJ: Prentice Hall, Englewood Cliffs, 1991. — 288 с. — ISBN 978-0135904640.

13. Алан Кэй The Early History of Smalltalk. — Apple Computer, ACM SIGPLAN Notices, vol.28, №3, March 1993.

14. Thomas Noll, Chanchal Kumar Roy Modeling Erlang in the Pi–Calculus. — ACM 1-59593-066-3/05/0009, 2005.

15. Design Principles Behind Smalltalk

16. kx : Calibrated performance

Использованные интернет адресы для сбора информации :
http://ru.wikipedia.org
http://the-programmer.ru/

Использованные программные обеспечения для компиляции и т.д.:
Microsoft Office Word
Borland Delphi 7
Paint
CodeBlocks
Internet Explorer