Минообрнауки России
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
“Тверской государственный университет”
(ТвГТУ)

Кафедра “Информационные системы”

КУРСОВАЯ РАБОТА

Сравнение программ для, которая генерируется множество первых n случайных чисел с помощью решета Эратосфена

Выполнил: Калашников М.С
Группа: Б.ИСТ.РВС.21.35
Проверил: Зыков И.И.

Тверь 2022 г

Описание задания

Номер варианта: 14

Задание: Написать программу, которая генерируется множество первых n случайных чисел с помощью решета Эратосфена Описание задания:

1. Описать современные выбранные языки программирования, на которых возможно написать программу для генерации чисел с помощью решета Эратосфена
2. Обосновать выбор каждого языка программирования, на котором будет решиться задача по следующим критериям
3. Сравнить получившиеся коды программ

Решето Эйлера

Решето Эйлера это вариант решета Эратосфена, в котором каждое составное число удаляется из списка только один раз.
Составляется исходный список начиная с числа *2*. На каждом этапе алгоритма первый номер в списке берется как следующее простое число, и определяются его произведения на каждое число в списке которые, маркируются для последующего удаления. После этого из списка убирают первое число и все помеченные числа, и процесс повторяется вновь:

Оглавление

Введение

Язык программирования — формальный язык, предназначенный 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 эзотерические, визуальные и игрушечные). Каждый год их число увеличивается. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей. Профессиональные программисты могут владеть несколькими языками 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 управления ЭВМ, в то время как естественные языки используются, прежде всего, для общения людей между собой. Большинство языков программирования использует специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

Как правило, язык программирования определяется не только через спецификации стандарта языка, формально определяющие его синтаксис и семантику, но и через воплощения (реализации) стандарта — программные средства, обеспечивающие трансляцию или интерпретацию программ на этом языке; такие программные средства различаются по производителю, марке и варианту (версии), времени выпуска, полноте воплощения стандарта, дополнительным возможностям; могут иметь определённые ошибки или особенности воплощения, влияющие на практику использования языка или даже на его стандарт.

Аналитическая часть

Описание языков программирования

1) Python - язык программирования общего назначения:

Представленный Гвидо ван Россумом в 1991 году, Python - чрезвычайно популярный язык общего назначения, широко используемый в сообществе специалистов по науке о данных и аналитике. Он имеет широкий спектр специализированных модулей и может похвастаться поддержкой мирового сообщества с многочисленными онлайн-сервисами, которые предоставляют Python API (интерфейс прикладного программирования). Его легко выучить, а низкий входной барьер также делает его идеальным первым языком для тех, кто плохо знаком с областью науки о данных и аналитики. Python также является отличной перспективой для тех, кто ищет карьеру в области науки о данных, основанной на приложениях. Большая часть процесса науки о данных вращается вокруг процесса ETL (извлечение-преобразование-загрузка), который поддерживается универсальностью, которую предлагает Python. Python также предоставляет такие пакеты, как Tensorflow, pandas и scikit-learn, что делает его фантастическим вариантом для расширенных приложений машинного обучения.

2) Java:

В настоящее время Java поддерживается корпорацией Oracle. Это стандартный язык общего назначения, работающий на виртуальной машине Java (JVM). Он обладает мощной способностью интегрировать методы науки о данных и аналитики в существующую кодовую базу. В результате многие современные системы построены на серверной части Java. Это бесценный язык для критически важных приложений обработки данных, поскольку он обеспечивает серьезную безопасность типов.

Java - идеальная вычислительная система, которая обеспечивает легкую переносимость между различными платформами. Эти факторы делают его подходящим для написания конкретных производственных кодов ETL и алгоритмов машинного обучения с интенсивными вычислениями. Многословие Java делает его очевидным первым выбором для специального анализа и специализированных статистических приложений. Многие компании требуют от специалистов по обработке данных, чтобы они могли беспрепятственно интегрировать производственный код науки о данных в их существующую кодовую базу, что стало возможным

благодаря преимуществам, предлагаемым производительностью и безопасностью типов Java.

3) C++:

C++ — это язык объектно-ориентированного программирования общего назначения (ООП), разработанный Бьярном Страуструпом и являющийся расширением языка Си. Следовательно, можно писать на C++ в «стиле C» или «объектно-ориентированном стиле». В определенных сценариях на нем можно писать любым способом и, таким образом, он является эффективным примером гибридного языка.

C++ считается языком промежуточного уровня, так как он включает в себя функции языка высокого и низкого уровня. Первоначально язык назывался «C с классами», так как он имел все свойства языка C с дополнительным понятием «классы». Тем не менее, он был переименован в C++ в 1983 году.

C ++ - один из самых популярных языков, в основном используемый с системным/прикладным программным обеспечением, драйверами, клиент-серверными приложениями и встроенным программным обеспечением.

Основным преимуществом C++ является коллекция предопределенных классов, представляющие собой типы данных, которые могут быть созданы несколько раз. Язык также облегчает объявление пользовательских классов. Классы могут дополнительно приспосабливать функции-члены для реализации определенной функциональности. Несколько объектов определенного класса могут быть определены для реализации функций в классе. Объекты могут быть определены как экземпляры, созданные во время выполнения. Эти классы также могут наследоваться другими новыми классами, которые по умолчанию принимают общедоступные и защищенные функции.

C++ включает в себя несколько операторов, таких как сравнение, арифметика, битовые манипуляции и логические операторы. Одна из наиболее привлекательных особенностей C++ заключается в том, что он допускает перегрузку некоторых операторов, таких как сложение.

Некоторые из основных концепций языка программирования C++ включают полиморфизм, виртуальные и дружественные функции, шаблоны, пространства имен и указатели.

Обоснования выбора языков программирования

1. Философия

Разработчики языка Python придерживаются определённой философии программирования, называемой «Дзэном Питона»[5]. Её текст выдаётся интерпретатором Питона по команде import this (работает один раз за сессию). Автором этой философии считается Тим Пейтерс.

Текст философии:

Оригинальный текст (англ.)

- *Beautiful is better than ugly.*
- *Explicit is better than implicit.*
- *Simple is better than complex.*
- *Complex is better than complicated.*
- *Flat is better than nested.*
- *Sparse is better than dense.*
- *Readability counts.*
- *Special cases aren't special enough to break the rules.*
- *Although practicality beats purity.*
- *Errors should never pass silently.*
- *Unless explicitly silenced.*
- *In the face of ambiguity, refuse the temptation to guess.*
- *There should be one — and preferably only one — obvious way to do it.*
- *Although that way may not be obvious at first unless you're Dutch.*
- *Now is better than never.*
- *Although never is often better than 'right now'.*
- *If the implementation is hard to explain, it's a bad idea.*
- *If the implementation is easy to explain, it may be a good idea.*
- *Namespaces are one honking great idea — let's do more of those!*

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.

- Должен существовать один — и, желательно, *только* один — очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец[6].
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем *прямо* сейчас.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, *возможно*, хороша.
- Пространства имён — отличная штука! Будем делать их побольше!

2. История

Разработка языка Python была начата в конце 1980-х годов[7] сотрудником голландского института CWI Гвидо ван Россумом. Для распределённой ОС Amoeba требовался расширяемый скриптовый язык, и Гвидо начал писать Python на досуге, позаимствовав некоторые наработки для языка ABC (*англ.*) (Гвидо участвовал в разработке этого языка, ориентированного на обучение программированию). В феврале 1991 года Гвидо опубликовал исходный текст в ньюсгруппе alt.sources[8]. С самого начала Python проектировался как объектно-ориентированный язык.

.py

Название языка произошло вовсе не от вида пресмыкающихся. Автор назвал язык в честь популярного британского комедийного телешоу 1970-х «Летающий цирк Монти Пайтона». Впрочем, всё равно название языка чаще ассоциируют именно со змеёй, нежели с фильмом — пиктограммы файлов в KDE или в Microsoft Windows и даже эмблема на сайте python.org (до выхода версии 2.5) изображают змеиные головы.

Наличие дружелюбного, отзывчивого сообщества пользователей считается наряду с дизайнерской интуицией Гвидо одним из факторов успеха Python. Развитие языка происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации документов PEP (Python Enhancement Proposal) — предложений по развитию Python[9].

3 декабря 2008 года[10], после длительного тестирования, вышла первая версия Python 3000 (или Python 3.0, также используется сокращение Py3k). В Python 3000 устранены многие недостатки архитектуры с максимально возможным (но не полным) сохранением совместимости со старыми версиями Питона. На сегодня поддерживаются обе ветви развития (Python 3.x и 2.x).

2.1. Влияние других языков на Python

Появившись сравнительно поздно, Python создавался под влиянием множества языков программирования:

- ABC (*англ.*) — отступы для группировки операторов, высокоуровневые структуры данных (map)[11][12] (фактически, Python создавался как попытка исправить ошибки, допущенные при проектировании ABC);
- Modula-3 — пакеты, модули, использование else совместно с try и except, именованные аргументы функций (на это также повлиял Common Lisp);
- C, C++ — некоторые синтаксические конструкции (как пишет сам Гвидо ван Россум — он использовал наиболее непротиворечивые конструкции из C, чтобы не вызвать неприязнь у C-программистов к Python[11]);
- Smalltalk — объектно-ориентированное программирование;
- Lisp — отдельные черты функционального программирования (lambda, map, reduce, filter и другие);
- Fortran — срезы массивов, комплексная арифметика;
- Miranda — списочные выражения;
- Java — модули logging, unittest, threading (часть возможностей оригинального модуля не реализована), xml.sax стандартной библиотеки, совместное использование finally и except при обработке исключений, использование @ для декораторов;
- Icon — генераторы.

Большая часть других возможностей Python (например, байт-компиляция исходного кода) также была реализована ранее в других языках.

3. Портируемость

Python портирован и работает почти на всех известных платформах — от КПК до мейнфреймов. Существуют порты под Microsoft Windows, практически все варианты UNIX (включая FreeBSD и Linux), Plan 9, Mac OS и Mac OS X, iPhone OS 2.0 и выше, Palm OS, OS/2, Amiga, AS/400 и даже OS/390, Symbian и Android [13].

По мере устаревания платформы её поддержка в основной ветви языка прекращается. Например, с серии 2.6 прекращена поддержка Windows 95, Windows 98 и Windows ME[14]. Однако на этих платформах можно использовать предыдущие версии Python — на данный момент сообщество активно поддерживает версии Python начиная от 2.3 (для них выходят исправления).

При этом, в отличие от многих портируемых систем, для всех основных платформ Python имеет поддержку характерных для данной платформы технологий (например, Microsoft COM/DCOM). Более того, существует специальная версия Питона для виртуальной машины Java — Jython, что позволяет интерпретатору выполняться на любой системе, поддерживающей Java, при этом классы Java могут непосредственно использоваться из Питона и даже быть написанными на Питоне. Также несколько проектов обеспечивают интеграцию с платформой Microsoft .NET, основные из которых — IronPython и Python.Net.

4. Типы и структуры данных

Python поддерживает динамическую типизацию, то есть тип переменной определяется только во время исполнения. Поэтому вместо «присваивания значения переменной» лучше говорить о «связывании значения с некоторым именем». В Питоне имеются встроенные типы: булевые, строки, Unicode-строки, целые числа произвольной точности, числа с плавающей запятой, комплексные числа и некоторые другие. Из коллекций Python поддерживает кортежи (*tuples*), списки, словари (ассоциативные массивы) и, начиная с версии 2.4, множества. Все значения в Питоне являются объектами, в том числе функции, методы, модули, классы.

Добавить новый тип можно либо написав класс (class), либо определив новый тип в модуле расширения (например, написанном на языке C). Система классов поддерживает наследование (одиночное и

множественное) и метапрограммирование. Возможно наследование от большинства встроенных типов и типов расширений.

Все объекты делятся на ссылочные и атомарные. К атомарным относятся int, long, complex и некоторые другие. При присваивании атомарных объектов копируется их значение, в то время как для ссылочных копируется только указатель на объект, таким образом, обе переменные после присваивания используют одно и то же значение. Ссылочные объекты бывают изменяемые и неизменяемые. Например, строки и кортежи являются неизменяемыми, а списки, словари и многие другие объекты — изменяемыми. Кортеж в Питоне является, по сути, неизменяемым списком. Во многих случаях кортежи работают быстрее списков[15], поэтому если вы не планируете изменять последовательность, то лучше использовать именно их.

5. Синтаксис и семантика

Язык обладает чётким и последовательным синтаксисом, продуманной модульностью и масштабируемостью, благодаря чему исходный код написанных на Питоне программ легко читаем.

5.1. Операторы

Набор операторов достаточно традиционен. Вот некоторые из них:

- условный оператор if (если). Альтернативный блок после else (иначе). Если условий и альтернатив несколько, можно использовать elif (сокр. от else if).
- операторы цикла while (пока) и for (для). Внутри цикла возможно применение break и continue для прерывания цикла и перехода сразу к следующей итерации соответственно.
- оператор определения класса class.
- оператор определения функции, метода или генератора def. Внутри возможно применение return (возврат) для возврата из функции или метода, а в случае генератора — yield (давать).
- оператор обработки исключений try — except — else или try — finally (начиная с версии 2.5, можно использовать finally, except и else в одном блоке).
- оператор pass ничего не делает. Используется для пустых блоков кода.

Одной из интересных синтаксических особенностей языка является выделение блоков кода с помощью отступов (пробелов или табуляций), поэтому в Питоне отсутствуют операторные скобки begin/end, как в языке Паскаль, или фигурные скобки, как в Си. Такой «трюк» позволяет сократить количество строк и символов в программе и приучает к «хорошему» стилю программирования. С другой стороны, поведение и даже корректность программы может зависеть от начальных пробелов в тексте. Некоторые критики языка считают такое поведение неинтуитивным и неудобным.[]

5.2. Выражения

Выражение является полноправным оператором в Питоне. Состав, синтаксис, ассоциативность и приоритет операций достаточно привычны для языков программирования и призваны минимизировать употребление скобок.

Отдельно стоит упомянуть *операцию форматирования* для строк (работает по аналогии с printf() из Си), которая использует тот же символ, что и взятие остатка от деления:

```
>>> print ("Здравствуй, %s!" % "Мир")
Здравствуй, Мир!
```

Python имеет удобные *цепочечные сравнения*. Такие условия в программах — не редкость:

```
1 <= a < 10 and 1 <= b < 20
```

Кроме того, логические операции (or и and) являются ленивыми: если для вычисления значения операции достаточно первого операнда, этот операнд и является результатом, в противном случае вычисляется второй операнд логической операции. Это основывается на свойствах алгебры логики: например, если один аргумент операции «ИЛИ» (or) является истиной, то и результат этой операции всегда является истиной. В случае, если второй операнд является сложным выражением, это позволяет сократить издержки на его вычисление. Этот факт широко использовался до версии 2.5 вместо условной конструкции:

```
(a < b) and "меньше" or "больше или равно"
```

Встроенные типы данных, как правило, имеют особый синтаксис для своих литералов (записанных в исходном коде констант):

```
"строка" + 'строка'  """тоже строка"""  u"Юникод-строка"
True or False          # булевы литералы
3.14                   # число с плавающей запятой
012 + 0xA              # числа в восьмеричной и шестнадцатеричной системах счисления
1 + 2j                 # целое число и мнимое число
[1, 2, "a"]            # список
(1, 2, "a")            # кортеж
{'a': 1, 'b': 'B'}     # словарь
lambda x: x**2         # неименованная функция
```

Для списков (и других последовательностей) Python предлагает набор операций над срезами. Особенностью является индексация, которая может показаться новичку странной, но раскрывает свою согласованность по мере использования. Индексы элементов списка начинаются с нуля. Запись среза s[N:M] означает, что в срез попадают все элементы от N включительно до M не включая. В качестве иллюстрации можно посмотреть этот пример.

5.3. Имена

Имя (идентификатор) может начинаться с латинской буквы любого регистра или подчёркивания, после чего в имени можно использовать и цифры. В качестве имени нельзя использовать ключевые слова (их список можно узнать по import keyword; print keyword.kwlist) и нежелательно переопределять встроенные имена. Имена, начинающиеся на подчёркивание, имеют специальное значение[16].

В каждой точке программы интерпретатор имеет доступ к трём пространствам имён (то есть отображениям имён в объекты): локальному, глобальному и встроенному.

Области видимости имён могут быть вложенными друг в друга (внутри определяемой функции видны имена из окружающего блока кода). На практике с областями видимости и связыванием имён связано несколько правил «хорошего тона», о которых можно подробнее узнать из документации.

5.4. Строки документации

Python предлагает механизм документирования кода pydoc. В начало каждого модуля, класса, функции вставляется строка документации — *docstring (*англ.*)*. Строки документации остаются в коде на момент времени исполнения, и в язык встроен доступ к документации[17], что используется современными IDE (например, Eclipse).

В интерактивном режиме можно получить помощь, сгенерировать гипертекстовую документацию по целому модулю или даже применить *doctest (*англ.*)* для автоматического тестирования модуля.

5.5. Директивы

Начиная с Python 2.3, для использования в тексте программы символов, не входящих в ASCII, необходимо явно указывать кодировку исходного кода в начале модуля, например:

```
# -*- coding: utf-8 -*-
```

После этого можно, например, использовать кириллицу в Unicode-литералах.

6. Возможности

6.1. Интерактивный режим

Подобно Лиспу и Прологу в режиме отладки, интерпретатор Питона имеет интерактивный режим работы, при котором введённые с клавиатуры операторы сразу же выполняются, а результат выводится на экран. Этот режим интересен не только новичкам, но и опытным программистам, которые могут протестировать в интерактивном режиме любой участок кода, прежде чем использовать его в основной программе, или просто использовать как калькулятор с большим набором функций.

Так выглядит общение с Питоном в интерактивном режиме:

```
>>> 2 ** 100                    # возведение 2 в степень 100
```

```
126765060022822940149670320537 6L
>>> from math import *                # импорт математических функций
>>> sin(pi * 0.5)                     # вычисление синуса от половины пи
1.0
>>> help(sorted)                      # помощь по функции sorted
Help on built-in function sorted in module __builtin__:
sorted(...)
  sorted(iterable, cmp=None, key=None, reverse=False) --> new sorted list
```

В интерактивном режиме доступен отладчик pdb и система помощи (вызывается по help()). Система помощи работает для модулей, классов и функций, только если те были снабжены строками документации.

Кроме встроенной, существуют и улучшенные интерактивные оболочки IPython[18] и bpython[19].

6.2. Объектно-ориентированное программирование

Дизайн языка Python построен вокруг объектно-ориентированной модели программирования. Реализация ООП в Питоне является элегантной, мощной и хорошо продуманной, но вместе с тем достаточно специфической по сравнению с другими объектно-ориентированными языками.

Возможности и особенности:

1. Классы являются одновременно объектами со всеми ниже приведёнными возможностями.
2. Наследование, в том числе множественное.
3. Полиморфизм (все функции виртуальные).
4. Инкапсуляция (два уровня — общедоступные и скрытые методы и поля). Особенность — скрытые члены доступны для использования и помечены как скрытые лишь особыми именами.
5. Специальные методы, управляющие жизненным циклом объекта: конструкторы, деструкторы, распределители памяти.
6. Перегрузка операторов (всех, кроме is, '.', '=' и символьных логических).
7. Свойства (имитация поля с помощью функций).
8. Управление доступом к полям (эмуляция полей и методов, частичный доступ, и т. п.).

9. Методы для управления наиболее распространёнными операциями (истинностное значение, len(), глубокое копирование, сериализация, итерация по объекту, …)
10. Метапрограммирование (управление созданием классов, триггеры на создание классов, и др.)
11. Полная интроспекция.
12. Классовые и статические методы, классовые поля.
13. Классы, вложенные в функции и классы.

6.3. Функциональное программирование

Python поддерживает парадигму функционального программирования, в частности:

- функция является объектом
- функции высших порядков
- рекурсия
- развитая обработка списков (списковые выражения, операции над последовательностями, итераторы)
- аналог замыканий
- частичное применение функции
- возможность реализации других средств на самом языке (например, карринг)

6.4. Модули и пакеты

Программное обеспечение (приложение или библиотека) на Питоне оформляется в виде модулей, которые в свою очередь могут быть собраны в *пакеты*. Модули могут располагаться как в каталогах, так и в ZIP-архивах. Модули могут быть двух типов по своему происхождению: модули, написанные на «чистом» Питоне, и модули расширения (extension modules), написанные на других языках программирования. Например, в стандартной библиотеке есть «чистый» модуль pickle и его аналог на Си: cPickle. Модуль оформляется в виде отдельного файла, а пакет — в виде отдельного каталога. Подключение модуля к программе осуществляется оператором import. После импорта модуль представлен отдельным объектом, дающим доступ к пространству имён модуля. В ходе

выполнения программы модуль можно перезагрузить функцией reload().

6.5. Интроспекция

Python поддерживает полную интроспекцию времени исполнения. Это означает, что для любого объекта можно получить всю информацию о его внутренней структуре.

Применение интроспекции является важной частью того, что называют pythonic style, и широко применяется в библиотеках и фреймворках Python, таких как PyRO, PLY, Cherry, Django и др., значительно экономя время использующего их программиста.

6.6. Обработка исключений

Обработка исключений поддерживается в Python посредством операторов try, except, else, finally, raise, образующих блок обработки исключения. В общем случае блок выглядит следующим образом:

```
try:
    # Здесь код, который может вызвать исключение
    raise Exception("message")  # Exception, это один из стандартных типов исключения (всего лишь класс),
                                # может использоваться любой другой, в том числе свой
except (Тип исключения1, Тип исключения2, …), Переменная:
    # Код в блоке выполняется, если тип исключения совпадает с одним из типов
    # (Тип исключения1, Тип исключения2, …) или является наследником одного
    # из этих типов.
    # Полученное исключение доступно в необязательной Переменной.
except (Тип исключения3, Тип исключения4, …), Переменная:
    # Количество блоков except не ограниченно
    raise  # Сгенерировать исключение "поверх" полученного; без параметров - повторно сгенерировать полученное
except:
    # Будет выполнено при любом исключении, не обработанном типизированными блоками except
else:
    # Код блока выполняется, если не было поймано исключений.
```

```
finally:
    # Будет исполнено в любом случае, возможно после
соответствующего
    # блока except или else
```

Совместное использование else, except и finally стало возможно только начиная с Python 2.5. Информация о текущем исключении всегда доступна через sys.exc_info(). Кроме значения исключения, Python также сохраняет состояние стека вплоть до точки возбуждения исключения — так называемый traceback.

В отличие от компилируемых языков программирования, в Python использование исключения не приводит к значительным накладным расходам (а зачастую даже позволяет ускорить исполнение программ) и очень широко используется. Исключения согласуются с философией Python (10-й пункт «дзена Python» — «Ошибки никогда не должны умалчиваться») и являются одним из средств поддержки «утиной типизации».

Иногда вместо явной обработки исключений удобнее использовать блок with (доступен, начиная с Python 2.5).

6.7. Итераторы

В программах на Питоне широко используются итераторы.
Цикл for может работать как с последовательностью, так и с итератором. Все коллекции, как правило, предоставляют итератор. Объекты определённого пользователем класса тоже могут быть итераторами. Подробнее об итераторах можно узнать в разделе о функциональном программировании. Модуль itertools стандартной библиотеки содержит много полезных функций для работы с итераторами.

6.8. Генераторы

Одной из интересных возможностей языка являются **генераторы** — функции, сохраняющие внутреннее состояние: значения локальных переменных и текущую инструкцию). Генераторы могут

использоваться как итераторы для структур данных и для ленивых вычислений. См. пример: генератор чисел Фибоначчи.

При вызове генератора функция немедленно возвращает объект-итератор, который хранит текущую точку исполнения и состояние локальных переменных функции. При запросе следующего значения (посредством метода next(), неявно вызываемого в for цикле) генератор продолжает исполнение функции от предыдущей точки останова до следующего оператора yield или return.

В Python 2.4 появились **генераторные выражения** — выражения, дающие в результате генератор. Генераторные выражения позволяют сэкономить память там, где иначе требовалось бы использовать список с промежуточными результатами:

```
>>> sum(i for i in xrange(1, 100) if i % 2 != 0)
2500
```

В этом примере суммируются все нечётные числа от 1 до 99.

Начиная с версии 2.5, Python поддерживает полноценные сопроцедуры: теперь в генератор можно передавать значения с помощью метода send() и возбуждать в его контексте исключения с помощью метода throw().

6.9. Управление контекстом выполнения

В Python 2.5 появились средства для управления контекстом выполнения блока кода — оператор with и модуль contextlib. См.: пример.

Оператор может применяться в тех случаях, когда до и после некоторых действий должны обязательно выполняться некоторые другие действия, независимо от возбуждённых в блоке исключений или операторов return: файлы должны быть закрыты, ресурсы освобождены, перенаправление стандартного ввода вывода закончено и т. п. Оператор улучшает читаемость кода, а значит, помогает предотвращать ошибки.

6.10. Декораторы

Начиная с версии 2.4, Python позволяет использовать т. н. *декораторы*[20] (не следует путать с одноимённым шаблоном проектирования) для поддержки существующей практики преобразования функций и методов в месте определения (декораторов может быть несколько). После долгих дебатов для декораторов стал использоваться символ @ в строках, предшествующих определению функции или метода. Следующий пример содержит описание статического метода без применения декоратора:

```
def myWonderfulMethod():
    return "Некоторый метод"
myWonderfulMethod = staticmethod(myWonderfulMethod)
```

и с помощью декоратора:

```
@staticmethod
def myWonderfulMethod():
    return "Некоторый метод"
```

Декоратор является ничем иным, как функцией, получающей в качестве первого аргумента декорируемую функцию или метод. Декораторы можно считать элементом аспектно-ориентированного программирования.

С версии 2.6 декораторы можно использовать с классами, аналогично функциям.

6.11. Другие возможности

В Python есть ещё несколько возможностей, отличающих его от многих других языков высокой гибкостью и динамичностью.

Например, класс является объектом, а в операторе определения класса можно использовать выражения в списке родительских классов.

```
def getClass():
    return dict
class D(getClass()):
```

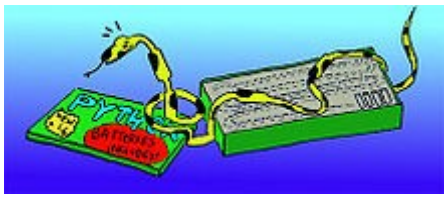
```
    pass
d = D()
```

Можно модифицировать многие объекты во время исполнения, например классы:

```
>>> class X(object): pass
…
>>> y = X()
>>> y.wrongMethod() # такого метода пока нет
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'X' object has no attribute 'wrongMethod'
>>> X.wrongMethod = lambda self : 'im here' # добавим его
>>> y.wrongMethod() # так как доступ к методу приводит к поиску по
__dict__ класса,
'im here' # то wrongMethod становится доступным всем экземплярам
```

7. Библиотеки

7.1. Стандартная библиотека



Python поставляется «с батарейками в комплекте».

Богатая стандартная библиотека является одной из привлекательных сторон Питона. Здесь имеются средства для работы со многими сетевыми протоколами и форматами Интернета, например, модули для написания HTTP-серверов и клиентов, для разбора и создания почтовых сообщений, для работы с XML и т. п. Набор модулей для работы с операционной системой позволяет писать кросс-платформенные приложения. Существуют модули для работы с регулярными выражениями, текстовыми кодировками, мультимедийными форматами, криптографическими протоколами, архивами, сериализации данных, поддержка юнит-тестирования и др.

7.2. Модули расширения и программные интерфейсы

Помимо стандартной библиотеки существует множество библиотек, предоставляющих интерфейс ко всем системным вызовам на разных платформах; в частности, на платформе Win32 поддерживаются все вызовы Win32 API, а также COM в объёме не меньшем, чем у Visual Basic или Delphi. Количество прикладных библиотек для Python в самых разных областях без преувеличения огромно (веб, базы данных, обработка изображений, обработка текста, численные методы, приложения операционной системы и т. д.).

Для Python принята спецификация программного интерфейса к базам данным DB-API 2 и разработаны соответствующие этой спецификации пакеты для доступа к различным СУБД: PostgreSQL, Oracle, Sybase, Firebird (Interbase), Informix, Microsoft SQL Server, MySQL и sqlite. На платформе Microsoft Windows доступ к БД возможен через ADO (ADOdb). Коммерческий пакет mxODBC для доступа к СУБД через ODBC для платформ Windows и UNIX разработан eGenix[21]. Для Питона написано много ORM (SQLObject, SQLAlchemy, Dejavu, Django), выполнены программные каркасы для разработки веб-приложений (Django, Pylons).

Библиотека NumPy для работы с многомерными массивами позволяет достичь производительности научных расчётов, сравнимой со специализированными пакетами. SciPy использует NumPy и предоставляет доступ к обширному спектру математических алгоритмов (матричная алгебра — BLAS, level 1-3 и LAPACK; БПФ). Numarray[22] специально разработан для операций с большими объёмами научных данных.

На стадии разработки[23] находится WSGI — интерфейс шлюза с веб-сервером (Python Web Server Gateway Interface).

Python предоставляет простой и удобный программный интерфейс C API для написания собственных модулей на языках Си и Си++. Такой инструмент как SWIG позволяет почти автоматически получать привязки для использования C/C++ библиотек в коде на Питоне. Возможности этого и других инструментов варьируются от автоматической генерации (C/C++/Fortran)-Python интерфейсов по специальным файлам (SWIG, pyste[24], SIP[25], pyfort[26]), до предоставления более удобных API (boost::python[27], CXX[28] и др.). Инструмент стандартной библиотеки ctypes позволяет программам Питона напрямую обращаться к динамическим библиотекам/DLL, написанным на C. Существуют модули, позволяющие встраивать код

на C/C++ прямо в исходные файлы Python, создавая расширения «на лету» (pyinline[29], weave[30]).

Другой подход состоит во встраивании интерпретатора Python в приложения. Python легко встраивается в программы на Java, C/C++, Ocaml. Взаимодействие Python-приложений с другими системами возможно также с помощью CORBA, XML-RPC, SOAP, COM.

С помощью Pyrex[31] возможна компиляция Python-подобного языка (добавлена возможность типизации) в эквивалентный Си-код и связывание с внешними модулями.

Экспериментальный проект shed skin[32][33] предполагает создание компилятора для трансформации неявно типизированных Python программ в оптимизированный C++ код. Начиная с версии 0.22 shed skin позволяет компилировать отдельные функции в модули расширений. Полная компиляция (по состоянию на 1 июля 2007 года) далека от завершения.

Python и подавляющее большинство библиотек к нему бесплатны и поставляются в исходных кодах. Более того, в отличие от многих открытых систем, лицензия никак не ограничивает использование Python в коммерческих разработках и не налагает никаких обязательств кроме указания авторских прав.

7.3. Графические библиотеки

С Питоном поставляется библиотека tkinter на основе Tcl/Tk для создания кроссплатформенных программ с графическим интерфейсом.

Существуют расширения, позволяющие использовать все основные GUI библиотеки — wxPython[34], основанное на библиотеке wxWidgets, PyGTK для Gtk, PyQt и PySide для Qt и другие. Некоторые из них также предоставляют широкие возможности по работе с базами данных, графикой и сетями, используя все возможности библиотеки, на которой основаны.

Для создания игр и приложений, требующих нестандартного интерфейса, можно использовать библиотеку Pygame. Она также предоставляет обширные средства работы с мультимедиа: с её помощью можно управлять звуком и изображениями, воспроизводить

видео. Предоставляемое pygame аппаратное ускорение графики OpenGL имеет более высокоуровневый интерфейс по сравнению с PyOpenGL[35], копирующей семантику C-библиотеки для OpenGL. Есть также PyOgre[36], обеспечивающая привязку к Ogre — высокоуровневой объектно-ориентированной библиотеке 3D-графики. Кроме того, существует библиотека pythonOCC[37], обеспечивающая привязку к среде 3D-моделирования и симуляции OpenCascade[38].

Для работы с растровой графикой используется библиотека Python Imaging Library.

8. Примеры программ

В статье «Примеры программ на языке Python» собраны примеры небольших программ, демонстрирующих некоторые возможности языка Python и его стандартной библиотеки.

9. Профилирование и оптимизация кода

В стандартной библиотеке Python имеется профайлер (модуль **profile**), который можно использовать для сбора статистики о времени работы отдельных функций. Для решения вопроса о том, какой вариант кода работает быстрее, можно использовать модуль timeit. Производимые в следующей программе измерения позволяют выяснить, какой из вариантов конкатенации строк более эффективен:

```
from timeit import Timer
def case1():  # А. инкрементальные конкатенации в цикле
    s = ""
    for i in range(10000):
        s += str(i)

def case2():  # Б. через промежуточный список и метод join
    s = []
    for i in range(10000):
        s.append(str(i))
    s = "".join(s)

def case3():  # В. списковое выражение и метод join
    return "".join([str(i) for i in range(10000)])

def case4():  # Г. генераторное выражение и метод join
```

```
    return "".join(str(i) for i in range(10000))

for v in range(1,5):
    print (Timer("func()","from __main__ import case%s as func" %
v).timeit(200))
```

Как и в любом языке программирования, в Питоне имеются свои приемы оптимизации кода. Оптимизировать код можно исходя из различных (часто конкурирующих друг с другом) критериев (увеличение быстродействия, уменьшение объёма требуемой оперативной памяти, компактность исходного кода и т. д.). Чаще всего программы оптимизируют по времени исполнения.

Здесь есть несколько очевидных правил:

- Не нужно оптимизировать программу, если скорость её выполнения достаточна.
- Используемый алгоритм имеет определённую временную сложность, поэтому перед оптимизацией кода программы стоит сначала пересмотреть алгоритм.
- Стоит использовать готовые и отлаженные функции и модули, даже если для этого нужно немного обработать данные. Например, в Питоне есть встроенная функция **sort()**.
- Профилирование поможет выяснить узкие места. Оптимизацию нужно начинать с них.

Python имеет следующие особенности и связанные с ними правила оптимизации:

- Вызов функций является достаточно дорогостоящей операцией, поэтому внутри вложенных циклов нужно стараться избегать вызова функций или, например, переносить цикл в функции. Функция, обрабатывающая последовательность, эффективнее, чем обработка той же последовательности в цикле вызовом функции.
- Старайтесь вынести из глубоко вложенного цикла всё, что можно вычислить во внешних циклах. Доступ к локальным переменным более быстрый, чем к глобальным, или чем доступ к полям.
- Оптимизатор **psyco** может помочь ускорить работу модуля программы при условии, что модуль не использует динамических свойств языка Питон.
- В случае, если модуль проводит массированную обработку данных и оптимизация алгоритма и кода не помогает, можно переписать критические участки, скажем, на языке Си или Pyrex.

Инструмент под названием Pychecker[39] поможет проанализировать исходный код на Питоне и выдать рекомендации по найденным проблемам (например, неиспользуемые имена, изменение сигнатуры метода при его перегрузке и т. п.). В ходе такого статического анализа исходного кода могут быть выявлены и ошибки. Pylint[40] призван решать близкие задачи, но имеет уклон в сторону проверки стиля кода.

10. Сравнение с другими языками

Наиболее часто Python сравнивают с Perl и Ruby. Эти языки также являются интерпретируемыми и обладают примерно одинаковой скоростью выполнения программ. Как и Perl, Python может успешно применяться для написания скриптов (сценариев). Как и Ruby, Python является хорошо продуманной системой для ООП.

Средства функционального программирования частично позаимствованы из Scheme и Icon.

В среде коммерческих приложений скорость выполнения программ на Python часто сравнивают с Java-приложениями[41].

Несмотря на то, что Python обладает достаточно самобытным синтаксисом, одним из принципов дизайна этого языка является принцип наименьшего удивления.

2.Java:

1. История создания языка JAVA

Язык Java зародился как часть проекта создания передового программного обеспечения для различных бытовых приборов. Реализация проекта была начата на языке C++, но вскоре возник ряд проблем, наилучшим средством борьбы с которыми было изменение самого инструмента - языка программирования. Стало очевидным, что необходим платформенно-независимый язык программирования, позволяющий создавать программы, которые не приходилось бы компилировать отдельно для каждой архитектуры и можно было бы использовать на различных процессорах под различными операционными системами.

Рождению языка Java предшествовала довольно интересная история. В 1990 году разработчик ПО компании Sun Microsystems Патрик Нотон понял, что ему надоело поддерживать сотни различных интерфейсов программ, используемых в компании, и сообщил исполнительному директору Sun Microsystems и своему другу Скотту МакНили о своем намерении перейти работать в компанию NeXT. МакНили, в свою очередь, попросил Нотона составить список причин своего недовольства и выдвинуть такое решение проблем, как если бы он был Богом и мог исполнить все, что угодно.

Нотон, хотя и не рассчитывал на то, что кто-то обратит внимание на его письмо, все же изложил свои претензии, беспощадно раскритиковав недостатки Sun Microsystems, в частности, разрабатываемую в тот момент архитектуру ПО NeWS. К удивлению Нотона, его письмо возымело успех: оно было разослано всем ведущим инженерам Sun Microsystems, которые не замедлили откликнуться и высказать горячую поддержку своему коллеге и одобрение его взглядов на ситуацию в Sun Microsystems. Обращение вызвало одобрение и у высшего руководства компании, а именно, у Билла Джоя, основателя Sun Microsystems, и Джеймса Гослинга (James Gosling), начальника Нотона.

В тот день, когда Нотон должен был уйти из компании, было принято решение о создании команды ведущих разработчиков с тем, чтобы они делали что угодно, но создали нечто необыкновенное. Команда из шести человек приступила к разработке нового объектно-ориентированного языка программирования, который был назван Oak (дуб), в честь дерева, росшего под окном Гослинга.

Вскоре компания Sun Microsystems преобразовала команду Green в компанию First Person. Новая компания обладала интереснейшей концепцией, но не могла найти ей подходящего применения. После ряда неудач неожиданно ситуация для компании резко изменилась: был анонсирован броузер Mosaic. Так родился World Wide Web, с которого началось бурное развитие Internet. Нотон предложил использовать Oak в создании Internet-приложений. Так Oak стал самостоятельным продуктом, вскоре был написан Oak-компилятор и Oak-браузер "WebRunner". В 1995 году компания Sun Microsystems приняла решение объявить о новом продукте, переименовав его в Java (единственное разумное объяснение названию - любовь программистов к кофе). Когда Java оказалась в руках Internet, стало необходимым запускать Java-аплеты - небольшие программы, загружаемые через Internet. WebRunner был переименован в HotJava и компания Netscape встала на поддержку Java-продуктов.

2. Достоинства языка

Язык должен был воплощать следующие качества: простоту и мощь, безопасность, объектную ориентированность, надежность, интерактивность, архитектурную независимость, возможность интерпретации, высокую производительность и легкость в изучении. Даже если вы никогда не напишете ни одной строки на языке Java, знать о его возможностях весьма полезно, поскольку именно перечисленные выше свойства языка придают динамику страницам Всемирной паутины.

2.1 Безопасность

Поскольку язык Java предназначен для использования в сетевой или распределенной среде, то вопросам его безопасности было уделено большое внимание. На данный момент язык Java позволяет создавать системы, надежно защищенные от вирусов и несанкционированного доступа.

Но 100% безопасности не может обеспечить ни один язык программирования, ведь все предусмотреть невозможно. Первые дыры в Java были найдены экспертами по вопросам безопасности из Пристонского университета еще в версии Java 1.0. Можно сказать, что все новые ошибки находятся до сих пор, но от этого не застрахован ни один язык программирования.

Тем не менее, специалисты компании Sun делают все, чтобы своевременно устранить бреши в безопасности JDK. Так, компания опубликовала внутренние спецификации интерпретатора языка Java и привлекла к поиску ошибок независимых специалистов в области безопасного ПО.

Вот лишь небольшой список ситуаций, возникновение которых предотвращает система безопасности языка Java:

Переполнение стека выполняемой программы, к которому приводили некоторые черви;

Повреждение участков памяти, которые находятся за пределами пространства, выделенного процессу;

Считывание и запись локальных файлов при использовании безопасного загрузчика классов, например веб-браузера, который запрещает такой доступ к файлам.

Все меры безопасности вполне уместны и обычно работают без проблем, но осмотрительность никогда не повредит. Со временем в язык были добавлены новые средства защиты. Начиная с версии 1.1, в языке Java появилось понятие классов с цифровой подписью. Пользуясь классом с цифровой подписью, вы можете быть уверенными в его авторе. Если вы ему доверяете, то можете предоставить этому классу все привилегии, доступные на вашей машине.

Альтернативный механизм доставки кода, предложенный компанией Microsoft, опирается на технологию ActiveX и для безопасности использует только цифровые подписи. Но этого недостаточно, ведь любой пользователь ПО фирмы Microsoft может подтвердить, что программы даже известных производителей часто завершаются аварийно, создавая тем самым опасность повреждения данных.

В тоже время система безопасности в языке Java намного надежнее технологии ActiveX, поскольку она самостоятельно контролирует приложение с момента его запуска и не позволяет ему причинить ущерб.

2.2 Объектная ориентированность

Язык Java разрабатывался как чисто объектно-ориентированный язык, в отличие от C++, объектная парадигма которого «ослабляется» возможностями, оставшимися от языка C. В Java отсутствуют такие структуры C++, как struct, union и procedure; они заменены на методы, интерфейсы и более развитые классы.

Классы в Java создаются иначе, чем в C++; это касается того, как Java обрабатывает операции наследования. Когда от родительского класса (или суперкласса) порождается подкласс, в Java используется ключевое слово extends:

```
public class MyString extends String{}
```

После этого оператора класс MyString наследует все методы и переменные своего суперкласса. В C++ для этого используется объявление типа class:mode superclass{.

Функции и процедуры в Java заменены на конструкции, называемые методами. Методы очень напоминают процедуры языка C++ за исключением того, что методы не могут быть независимыми от класса (кроме методов из интерфейсов).

В Java, как и в C++, возможны множественные конструкторы, дающие программисту возможность инициализировать объект различными способами. При объявлении конструкторов существуют два основных правила: имя конструктора и название класса должны совпадать; при объявлении конструктора не указывается возвращаемый тип. Как и другие ссылочные переменные, классы создаются динамически при помощи ключевого слова new. Ниже приведен пример объявления класса с несколькими конструкторами:

```
public Class MyString extends String{

public String x;

public MyString(){

x=new String(“Строка по умолчанию”);

//вызов конструктора класса String

}

public MyString(String x){

this.x=new String(x);

}
```

В приведенном примере ключевое слово this используется так же, как и в C++, для того чтобы различать обращения к переменным класса и переменным методов.

Другой способ создания класса - использовать конструктор суперкласса и ключевое слово super. Вот простой пример:

```
public class ParentClass{

int x,y;

public ParentClass(x,y){

this.x=0;

this.y=0;

}
```

```
}

public class ChildClass extends ParentClass{

public ChildClass(x,y){

super(x,y);

//вызов конструктора суперкласса

}

}
```

Методы класса во многом похожи на конструкторы, однако они могут возвращать любой тип.

```
public int ClassMethod(int j){

x+=j;

y+=j;

return (x+y);

}
```

2.3 Надежность

Java ограничивает вас в нескольких ключевых областях и таким образом способствует обнаружению ошибок на ранних стадиях разработки программы. В то же время в ней отсутствуют многие источники ошибок, свойственных другим языкам программирования (строгая типизация, например). Большинство используемых сегодня программ “отказывают” в одной из двух ситуаций: при выделении памяти, либо при возникновении исключительных ситуаций. В традиционных средах программирования распределение памяти является довольно нудным занятием -- программисту приходится самому следить за всей используемой в программе памятью, не забывая освобождать ее по мере того, как потребность в ней отпадает. Зачастую программисты забывают освобождать захваченную ими память или, что еще хуже, освобождают ту память, которая все еще используется какой-либо частью программы. Исключительные ситуации в традиционных средах программирования часто возникают в таких, например, случаях, как деление на нуль или попытка открыть несуществующий файл, и их приходится

обрабатывать с помощью неуклюжих и нечитабельных конструкций (кроме Delphi). Java фактически снимает обе эти проблемы, используя сборщик мусора для освобождения незанятой памяти и встроенные объектно-ориентированные средства для обработки исключительных ситуаций.

Специальный процесс сборки мусора - это одна из интереснейших особенностей языка программирования Java и среды выполнения приложений Java, предназначенная для удаления ненужных объектов из памяти. Эта система избавляет программиста от необходимости внимательно следить за использованием памяти, освобождая ненужные более области явным образом.

Создавая объекты в Java, вы можете руководствоваться принципом "создай и забудь", так как система сборки мусора позаботится об удалении ваших объектов. Объект будет удален из памяти, как только на него не останется ни одной ссылки из других объектов.

Приоритет процесса сборки мусора очень низкий, поэтому "уборка" среды выполнения приложений Java не отнимает ресурсы у самих приложений.

Указатели или адреса в памяти -- наиболее мощная и наиболее опасная черта C++. Причиной большинства ошибок в сегодняшнем коде является именно неправильная работа с указателями. Например, одна из типичных ошибок -- просчитаться на единицу в размере массива и испортить содержимое ячейки памяти, расположенной вслед за ним.

Хотя в Java дескрипторы объектов и реализованы в виде указателей, в ней отсутствуют возможности работать непосредственно с указателями. Вы не можете преобразовать целое число в указатель, а также обратиться к произвольному адресу памяти.

2.4 Интерактивность

Java создавалась как средство, которое должно удовлетворить насущную потребность в создании интерактивных сетевых программ. В Java реализовано несколько интересных решений, позволяющих писать код, который выполняет одновременно массу различных функций и не забывает при этом следить за тем, что и когда должно произойти. В языке Java для решения проблемы синхронизации процессов применен наиболее элегантный из всех когда-либо изобретенных методов, который позволяет конструировать прекрасные интерактивные системы. Простые в обращении изящные

под процессы Java дают возможность реализации в программе конкретного поведения, не отвлекаясь при этом на встраивание глобальной циклической обработки событий.

2.5 Независимость от архитектуры ЭВМ

Компилятор генерирует объектный файл, формат которого не зависит от архитектуры компьютера. В данном случае скомпилированная программа может выполняться на любых процессорах под управлением системы выполнения программ языка Java.

Для этого компилятор языка Java генерирует команды байт-кода, не зависящие от конкретной архитектуры компьютера. Байт-код разработан таким образом, чтобы на любой машине его можно было легко интерпретировать либо на лету перевести в машинозависимый код.

Но эту идею нельзя назвать революционной. Еще в 70-е годы в системе реализации языка Pascal, разработанной Никлаусом Виртом (Niclaus Wirth) и в системе UCSD Pascal применялась та же самая технология.

Использование байт-кодов дает большой выигрыш при выполнении программы (хотя в некоторых случаях синхронная компиляция его компенсирует). Разработчики языка Java прекрасно справились с разработкой набора команд байт-кода, которые отлично работают на большинстве современных компьютеров, легко транслируясь в реальные машинные команды.

2.6 Интерпретация и высокая производительность

Необычайная способность Java исполнять свой код на любой из поддерживаемых платформ достигается тем, что ее программы транслируются в некое промежуточное представление, называемое байт-кодом (bytecode). Байт-код, в свою очередь, может интерпретироваться в любой системе, в которой есть среда времени выполнения Java. Большинство ранних систем, в которых пытались обеспечить независимость от платформы, обладало огромным недостатком -- потерей производительности (Basic, Perl). Несмотря на то, что в Java используется интерпретатор, байт-код легко переводится непосредственно в “родные” машинные коды (Just In Time compilers) “на лету”. При этом достигается очень высокая производительность (Symantec JIT встроен в Netscape Navigator).

2.7 Простота изучения

Язык Java, хотя и более сложный чем языки командных интерпретаторов, все же неизмеримо проще для изучения, чем другие другие языки программирования, например C++. Java отличен от C++ несколькими основными изменениями, облегчающими восприятие синтаксиса Java: удалены препроцессор, заголовочные файлы, операторы typeded и директивы #define. Благодаря этому языку Java легче изучать. К примеру, рассмотрим следующий фрагмент программы:

Как можно видеть, в Java удалены все директивы препроцессора, такие как #define, что облегчает восприятие текста программы. Вместо директивы C++ #include в языке Java используется оператор import, позволяющий импортировать другие объектные классы в создаваемый код.

3. Аплеты JAVA

Программы, составленные на языке программирования Java, можно разделить по своему назначению на две большие группы.

К первой группе относятся приложения Java, предназначенные для автономной работы под управлением специальной интерпретирующей машины Java. Реализации этой машины созданы для всех основных компьютерных платформ.

Вторая группа - это так называемые апплеты (applets). Каждый апплет -- это небольшая программа, динамически загружаемая по сети -- точно так же, как картинка, звуковой файл или элемент мультипликации. Главная особенность апплетов заключается в том, что они являются настоящими программами, а не очередным форматом файлов для хранения мультфильмов или какой-либо другой информации. Апплет не просто проигрывает один и тот же сценарий, а реагирует на действия пользователя и может динамически менять свое поведение.

Приложения, относящиеся к первой, - это обычные автономные программы. Так как они не содержат машинного кода и работают под управлением специального интерпретатора, их производительность заметно ниже, чем у обычных программ, составленных, например, на языке программирования C++. Однако не следует забывать, что программы Java без перетрансляции способны работать на любой платформе, что само по себе имеет большое значение в плане разработок для Internet.

Апплеты Java встраиваются в документы HTML, хранящиеся на сервере Web. С помощью апплетов вы можете сделать страницы сервера Web динамичными и интерактивными. Апплеты позволяют выполнять сложную локальную обработку данных, полученных от сервера Web или введенных пользователем с клавиатуры. Из соображений безопасности апплеты (в отличие от обычных приложений Java) не имеют никакого доступа к файловой системе локального компьютера. Все данные для обработки они могут получить только от сервера Web. Более сложную обработку данных можно выполнять, организовав взаимодействие между апплетами и расширениями сервера Web - приложениями CGI и ISAPI.

Для повышения производительности приложений Java в современных браузерах используется компиляция "на лету"- Just-In-Time compilation (JIT). При первой загрузке апплета его код транслируется в обычную исполнимую программу, которая сохраняется на диске и запускается. В результате общая скорость выполнения апплета Java увеличивается в несколько раз.

4. Виртуальная машина JAVA и байт-коды

Программа, написанная на одном из языков высокого уровня, к которым относится и язык Java, так называемый исходный модуль, не может быть сразу же выполнена. Ее сначала надо откомпилировать, т. е. перевести в последовательность машинных команд -- объектный модуль. Но и он, как правило, не может быть сразу же выполнен: объектный модуль надо еще скомпоновать с библиотеками использованных в модуле функций и разрешить перекрестные ссылки между секциями объектного модуля, получив в результате загрузочный модуль -- полностью готовую к выполнению программу.

Исходный модуль, написанный на Java, не может избежать этих процедур, но здесь проявляется главная особенность технологии Java -- программа компилируется сразу в машинные команды, но не команды какого-то конкретного процессора, а в команды так называемой виртуальной машины Java (JVM, Java Virtual Machine). Виртуальная машина Java -- это совокупность команд вместе с системой их выполнения. Виртуальная машина Java полностью стековая, так что не требуется сложная адресация ячеек памяти и большое количество регистров. Поэтому команды JVM короткие, большинство из них имеет длину 1 байт, от чего команды JVM называют байт-кодами (bytecodes), хотя имеются команды длиной 2 и 3 байта. Согласно статистическим исследованиям средняя длина команды составляет 1,8 байта. Полное описание команд и всей

архитектуры JVM содержится в спецификации виртуальной машины Java (VMS, Virtual Machine Specification).

Другая особенность Java -- все стандартные функции, вызываемые в программе, подключаются к ней только на этапе выполнения, а не включаются в байт-коды. Как говорят специалисты, происходит динамическая компоновка (dynamic binding). Это тоже сильно уменьшает объем откомпилированной программы.

Итак, на первом этапе программа, написанная на языке Java, переводится компилятором в байт-коды. Эта компиляция не зависит от типа какого-либо конкретного процессора и архитектуры некоего конкретного компьютера. Она может быть выполнена один раз сразу же после написания программы. Байт-коды записываются в одном или нескольких файлах, могут храниться во внешней памяти или передаваться по сети. Это особенно удобно благодаря небольшому размеру файлов с байт-кодами. Затем полученные в результате компиляции байт-коды можно выполнять на любом компьютере, имеющем систему, реализующую JVM. При этом не важен ни тип процессора, ни архитектура компьютера. Так реализуется принцип Java "Write once, run anywhere" -- "Написано однажды, выполняется где угодно".

Интерпретация байт-кодов и динамическая компоновка значительно замедляют выполнение программ. Это не имеет значения в тех ситуациях, когда байт-коды передаются по сети, сеть все равно медленнее любой интерпретации, но в других ситуациях требуется мощный и быстрый компьютер. Поэтому постоянно идет усовершенствование интерпретаторов в сторону увеличения скорости интерпретации. Разработаны JIT-компиляторы (Just-In-Time), запоминающие уже интерпретированные участки кода в машинных командах процессора и просто выполняющие эти участки при повторном обращении, например, в циклах. Это значительно увеличивает скорость повторяющихся вычислений. Фирма SUN разработала целую технологию Hot-Spot и включает ее в свою виртуальную машину Java. Но, конечно, наибольшую скорость может дать только специализированный процессор.

Фирма SUN Microsystems выпустила микропроцессоры PicoJava, работающие на системе команд JVM, и собирается выпускать целую линейку все более мощных Java-процессоров. Есть уже и Java-процессоры других фирм. Эти процессоры непосредственно выполняют байт-коды. Но при выполнении программ Java на других процессорах требуется еще интерпретация команд JVM в команды конкретного процессора, а значит, нужна программа-интерпретатор,

причем для каждого типа процессоров, и для каждой архитектуры компьютера следует написать свой интерпретатор.

Эта задача уже решена практически для всех компьютерных платформ. На них реализованы виртуальные машины Java, а для наиболее распространенных платформ имеется несколько реализаций JVM разных фирм. Все больше операционных систем и систем управления базами данных включают реализацию JVM в свое ядро. Создана и специальная операционная система JavaOS, применяемая в электронных устройствах. В большинство браузеров встроена виртуальная машина Java для выполнения апплетов.

Кроме реализации JVM для выполнения байт-кодов на компьютере еще нужно иметь набор функций, вызываемых из байт-кодов и динамически компонующихся с байт-кодами. Этот набор оформляется в виде библиотеки классов Java, состоящей из одного или нескольких пакетов. Каждая функция может быть записана байт-кодами, но, поскольку она будет храниться на конкретном компьютере, ее можно записать прямо в системе команд этого компьютера, избегнув тем самым интерпретации байт-кодов. Такие функции называют "родными" методами (native methods). Применение "родных" методов ускоряет выполнение программы.

Фирма SUN Microsystems -- создатель технологии Java -- бесплатно распространяет набор необходимых программных инструментов для полного цикла работы с этим языком программирования: компиляции, интерпретации, отладки, включающий и богатую библиотеку классов, под названием JDK (Java Development Kit).

Набор программ и классов JDK содержит:

1. Компилятор javac из исходного текста в байт-коды; интерпретатор java, содержащий реализацию JVM;

2. Облегченный интерпретатор jre (в последних версиях отсутствует);

3. Программу просмотра апплетов appietviewer, заменяющую браузер;

4. Отладчик jdb;

5. Дизассемблер javap;

6. Программу архивации и сжатия jar;

7. Программу сбора документации javadoc;

8. Программу javah генерации заголовочных файлов языка C;

9. Программу javakey добавления электронной подписи;

10. Программу native2ascii, преобразующую бинарные файлы в текстовые;

11. Программы rmic и rmiregistry для работы с удаленными объектами;

12. Программу serialver, определяющую номер версии класса;

13. Библиотеки и заголовочные файлы "родных" методов;

14. Библиотеку классов Java API (Application Programming Interface).

Кроме JDK, компания SUN отдельно распространяет еще и набор JRE (Java Runtime Environment).

Набор программ и пакетов классов JRE содержит все необходимое для выполнения байт-кодов, в том числе интерпретатор java (в прежних версиях облегченный интерпретатор jre) и библиотеку классов. Это часть JDK, не содержащая компиляторы, отладчики и другие средства разработки. Именно JRE или его аналог других фирм содержится в браузерах, умеющих выполнять программы на Java, операционных системах и системах управления базами данных.

5. Мобильность JAVA

Создание приложений, действительно работающих на разных платформах - непростая задача. К сожалению, дело не ограничивается необходимостью перекомпиляции исходного текста программы для работы в другой среде. Много проблем возникает с несовместимостью программных интерфейсов различных операционных систем и графических оболочек, реализующих пользовательский интерфейс.

Вспомните хотя бы проблемы, связанные с переносом 16-разрядных приложений Windows в 32-разрядную среду Windows 95 и Windows NT. Даже если вы тщательно следовали всем рекомендациям, разрабатывая приложения так, чтобы они могли работать в будущих версиях Windows, едва ли вам удастся просто перекомпилировать исходные тексты, не изменив в них ни строчки. Ситуация еще больше ухудшается, если вам нужно, например, перенести исходные тексты приложения Windows в среду операционной системы OS/2 или в

оболочку X-Windows операционной системы UNIX. А ведь есть еще другие компьютеры и рабочие станции!

Как нетрудно заметить, даже если стандартизовать язык программирования для всех платформ, проблемы совместимости с программным интерфейсом операционной системы значительно усложняют перенос программ на различные платформы. И, конечно, вы не можете мечтать о том, чтобы загрузочный модуль одной и той же программы мог работать без изменений в среде различных операционных систем и на различных платформах. Если программа подготовлена для процессора Intel, она ни за что не согласится работать на процессоре Alpha или каком-либо другом.

В результате создавая приложение, способное работать на различных платформах, вы вынуждены фактически делать несколько различных приложений и сопровождать их по отдельности.

Вначале программист готовит исходные тексты приложения для платформы Windows NT и отлаживает их там. Для получения загрузочного модуля исходные тексты компилируются и редактируются. Полученный в результате загрузочный модуль может работать на процессоре фирмы Intel в среде операционной системы Windows NT.

Для того чтобы перенести приложение в среду операционной системы компьютера Macintosh, программист вносит необходимые изменения в исходные тексты приложения. Эти изменения необходимы из-за различий в программном интерфейсе операционной системы Windows NT и операционной системы, установленной в Macintosh. Далее эти исходные тексты транслируются и редактируются, в результате чего получается загрузочный модуль, способный работать в среде Macintosh, но не способный работать в среде Windows NT.

Программа на языке Java компилируется в двоичный модуль, состоящий из команд виртуального процессора Java. Такой модуль содержит байт-код, предназначенный для выполнения Java-интерпретатором. На настоящий момент уже созданы первые модели физического процессора, способного выполнять этот байт-код, однако интерпретаторы Java имеются на всех основных компьютерных платформах. Разумеется, на каждой платформе используется свой интерпретатор, или, точнее говоря, свой виртуальный процессор Java.

Если ваше приложение Java (или апплет) должно работать на нескольких платформах, нет необходимости компилировать его исходные тексты несколько раз. Вы можете откомпилировать и

отладить приложение Java на одной, наиболее удобной для вас платформе. В результате вы получите байт-код, пригодный для любой платформы, где есть виртуальный процессор Java.

Таким образом, приложение Java компилируется и отлаживается только один раз, что уже значительно лучше. Остается, правда, вопрос - как быть с программным интерфейсом операционной системы, который отличается для разных платформ?

Здесь, на наш взгляд, разработчиками Java предлагается достаточно неплохое решение. Приложение Java не обращается напрямую к интерфейсу операционной системы. Вместо этого оно пользуется готовыми стандартными библиотеками классов, содержащими все необходимое для организации пользовательского интерфейса, обращения к файлам, для работы в сети и так далее.

Внутренняя реализация библиотек классов, разумеется, зависит от платформы. Однако все загрузочные модули, реализующие возможности этих библиотек, поставляются в готовом виде вместе с виртуальной машиной Java, поэтому программисту не нужно об этом заботиться. Для операционной системы Windows, например, поставляются библиотеки динамической загрузки DLL, внутри которых запрятана вся функциональность стандартных классов Java.

Абстрагируясь от аппаратуры на уровне библиотек классов, программисты могут больше не заботиться о различиях в реализации программного интерфейса конкретных операционных систем. Это позволяет создавать по-настоящему мобильные приложения, не требующие при переносе на различные платформы перетрансляции и изменения исходного текста.

Еще одна проблема, возникающая при переносе программ, составленных на языке программирования C, заключается в том, что размер области памяти, занимаемой переменными стандартных типов, различный на разных платформах. Например, в среде операционной системы Windows версии 3.1 переменная типа int в программе, составленной на C, занимает 16 бит. В среде Windows NT этот размер составляет 32 бита.

Очевидно, что трудно составлять программу, не зная точно, сколько имеется бит в слове или в байте. При переносе программ на платформы с иной разрядностью могут возникать ошибки, которые трудно обнаружить.

В языке Java все базовые типы данных имеют фиксированную разрядность, которая не зависит от платформы. Поэтому программисты всегда знают размеры переменных в своей программе.

Заключение

Язык программирования Java - это полностью объектно-ориентированный язык, который в отношении синтаксиса многое унаследовал от C++. Конечно, преимущества Java далеко не исчерпываются межплатформенностью. Язык Java в синтаксическом отношении проще и логичнее, чем C++. Java как платформа предоставляет в распоряжение программистов большое количество библиотек (пакетов), в которых содержится большое количество описаний классов и интерфейсов на все случаи жизни. С их помощью можно создавать стопроцентные приложения Java с возможностью обращения к базам данных, поддержкой передачи почтовых сообщений, с клиентской частью, которой необходим web-браузер, или, наоборот, с клиентской частью, обладающей изощренным интерфейсом.

Java - это очень элегантный и красивый язык. Однако при его использовании проблем также избежать не удастся. Одна из серьезных проблем заключается в том, что при создании сложного приложения на Java вам придется использовать только этот язык для создания всех частей этого приложения. В Java предусмотрено не так уж много средств для межъязыкового взаимодействия (что понятно ввиду предназначения Java быть единым многоцелевым языком программирования). В реальном мире существуют миллионы строк готового кода, которые хотелось бы интегрировать с новыми приложениями на Java. Однако это сделать очень трудно.

3.C++:

Безусловно, Си++ восходит, главным образом, к Си. Си сохранен как подмножество, поэтому сделанного в Си акцента на средствах низкого уровня достаточно, чтобы справляться с самыми насущными задачами системного программирования. Си, в свою очередь, многим обязан своему предшественнику BCPL.

Название Си++ - изобретение лета 1983-его. Более ранние версии языка использовались начиная с 1980-ого и были известны как "Си с Классами". Первоначально язык был придуман потому, что автор хотел написать событийно управляемые модели для чего был бы идеален Simula67, если не принимать во внимание эффективность.

"Си с Классами" использовался для крупных проектов моделирования, в которых строго тестировались возможности написания программ, требующих (только) минимального пространства памяти и времени на выполнение. В "Си с Классами" не хватало перегрузки операций, ссылок, виртуальных функций и многих деталей. Си++ был впервые введен за пределами исследовательской группы автора в июле 1983-го. Однако тогда многие особенности Си++ были еще не придуманы.

Название Си++ выдумал Рик Масситти. Название указывает на эволюционную природу перехода к нему от Си. "++" - это операция приращения в Си. Чуть более короткое имя Си+ является синтаксической ошибкой, кроме того, оно уже было использовано как имя совсем другого языка. Знатоки семантики Си находят, что Си+ + хуже, чем Си ++. Названия D язык не получил, поскольку он является расширением Си и в нем не делается попыток исцелиться от проблем путем выбрасывания различных особенностей.

Си++ - это универсальный язык программирования, задуманный так, чтобы сделать программирование более приятным для серьезного программиста. За исключением второстепенных деталей Си+ + является надмножеством языка программирования Си. Помимо возможностей, которые дает Си, Си++ предоставляет гибкие и эффективные средства определения новых типов. Используя определения новых типов, точно отвечающих концепциям приложения, программист может разделять разрабатываемую программу на легко поддающиеся контролю части. Такой метод построения программ часто называют абстракцией данных. Информация о типах содержится в некоторых объектах типов, определенных пользователем. Такие объекты просты и надежны в использовании в тех ситуациях, когда их тип нельзя установить на стадии компиляции. Программирование с применением таких объектов часто называют объектно-ориентированным. При правильном использовании этот метод дает более короткие, проще понимаемые и легче контролируемые программы.

Изначально Си++ был разработан, чтобы автору и его друзьям не приходилось программировать на ассемблере, Си или других современных языках высокого уровня. Основным его предназначением было сделать написание хороших программ более простым и приятным для отдельного программиста. Плана разработки Си++ на бумаге никогда не было. Проект, документация и реализация двигались одновременно. Разумеется, внешний интерфейс Си++ был написан на Си++. Никогда не существовало "Проекта Си++" и "Комитета по разработке Си++". Поэтому Си+ + развивался и продолжает развиваться во всех направлениях, чтобы

справляться со сложностями, с которыми сталкиваются пользователи, а также в процессе дискуссий автора с его друзьями и коллегами.

В качестве базового языка для Си++ был выбран Си, потому что он:

· многоцелевой, лаконичный и относительно низкого уровня:

· отвечает большинству задач системного программирования:

· идет везде и на всем:

· пригоден в среде программирования UNIX.

В Си есть свои сложности, но в наспех спроектированном языке тоже были бы свои, а сложности Си нам известны. Самое главное, работа с Си позволила "Си с Классами" быть полезным (правда, неудобным) инструментом в ходе первых месяцев раздумий о добавлении к Си Simula-подобных классов.

Си++ стал использоваться шире, и по мере того, как возможности, предоставляемые им помимо возможностей Си, становились все более существенными, вновь и вновь поднимался вопрос о том, сохранять ли совместимость с Си. Ясно, что отказавшись от определенной части наследия Си можно было бы избежать ряда проблем. Это не было сделано, потому что:

· есть миллионы строк на Си, которые могли бы принести пользу в Си++ при условии, что их не нужно было бы полностью переписывать с Си на Си++;

· есть сотни тысяч строк библиотечных функций и сервисных программ, написанных на Си которые можно было бы использовать из или на Си++ при условии, что Си++ полностью совместим с Си по загрузке и синтаксически очень похож на Си;

· есть десятки тысяч программистов, которые знают Си, и которым, поэтому, нужно только научиться использовать новые особенности Си++, а не заново изучать его основы;

· поскольку Си++ и Си будут использоваться на одних и тех же системах одними и теми же людьми, отличия должны быть либо очень большими, либо очень маленькими, чтобы свести к минимуму ошибки и недоразумения.

Позднее была проведена проверка определения Си++, чтобы удостовериться в том, что любая конструкция, допустимая и в Си, и в Си++, действительно означает в обоих языках одно и то же.

Си++ был развит из языка программирования Си и за очень немногими исключениями сохраняет Си как подмножество. Базовый язык, Си подмножество Си++, спроектирован так, что имеется очень близкое соответствие между его типами, операциями и операторами и компьютерными объектами, с которыми непосредственно приходится иметь дело: числами, символами и адресами. За исключением операций свободной памяти new и delete, отдельные выражения и операторы Си++обычно не нуждаются в скрытой поддержке во время выполнения или подпрограммах.

Одним из первоначальных предназначений Си было применение его вместо программирования на ассемблере в самых насущных задачах системного программирования. Когда проектировался Си++, были приняты меры, чтобы не ставить под угрозу успехи в этой области. Различие между Си и Си++ состоит в первую очередь в степени внимания, уделяемого типам и структурам. Си выразителен и снисходителен. Си++ еще более выразителен, но чтобы достичь этой выразительности, программист должен уделить больше внимания типам объектов. Когда известны типы объектов, компилятор может правильно обрабатывать выражения, тогда как в противном случае программисту пришлось бы задавать действия с мучительными подробностями. Знание типов объектов также позволяет компилятору обнаруживать ошибки, которые в противном случае остались бы до тестирования. Заметьте, что использование системы типов для того, чтобы получить проверку параметров функций, защитить данные от случайного искажения, задать новые операции и т.д., само по себе не увеличивает расходов по времени выполнения и памяти.

Особое внимание, уделенное при разработке Си++ структуре, отразилось на возрастании масштаба программ, написанных со времени разработки Си. Маленькую программу (меньше 1000 строк) вы можете заставить работать с помощью грубой силы, даже нарушая все правила хорошего стиля. Для программ больших размеров это не совсем так. Если программа в 10 000 строк имеет плохую структуру, то вы обнаружите, что новые ошибки появляются так же быстро, как удаляются старые. Си++был разработан так, чтобы дать возможность разумным образом структурировать большие программы таким образом, чтобы для одного человека не было непомерным справляться с программами в 25 000 строк. Существуют программы гораздо больших размеров, однако те, которые работают, в целом, как

оказывается, состоят из большого числа почти независимых частей, размер каждой из которых намного ниже указанных пределов.

Естественно, сложность написания и поддержки программы зависит от сложности разработки, а не просто от числа строк текста программы, так что точные цифры, с помощью которых были выражены предыдущие соображения, не следует воспринимать слишком серьезно.

Существенным критерием при разработке языка была простота. Там, где возникал выбор между упрощением руководства по языку и другой документации и упрощением компилятора, выбиралось первое. Огромное значение также предавалось совместимости с Си, это помешало удалить синтаксис Си.

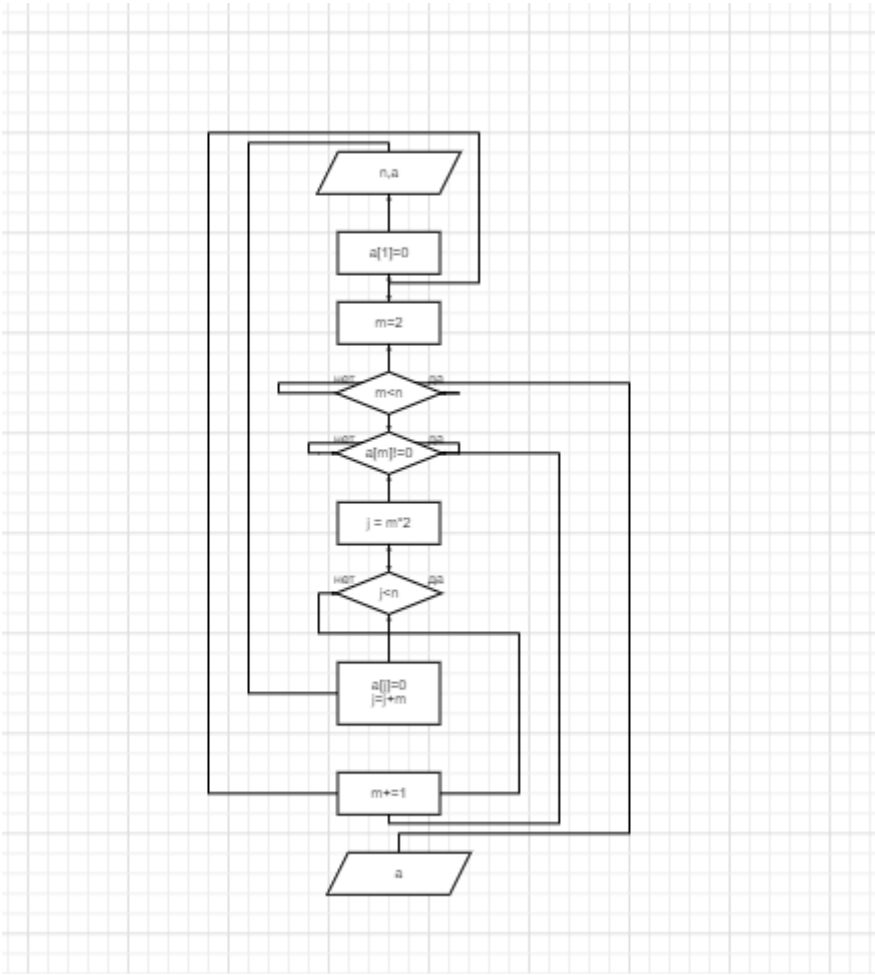
В Си++ нет типов данных высокого уровня и нет первичных операций высокого уровня. В нем нет, например, матричного типа с операцией обращения или типа строка с операцией конкатенации. Если пользователю понадобятся подобные типы, их можно определить в самом языке. По сути дела, основное, чем занимается программирование на Си++ - это определение универсальных и специально-прикладных типов. Хорошо разработанный тип, определяемый пользователем, отличается от встроенного типа только способом определения, но не способом использования.

Исключались те черты, которые могли бы повлечь дополнительные расходы памяти или времени выполнения. Например, мысли о том, чтобы сделать необходимым хранение в каждом объекте ”хозяйственной” информации, были отвергнуты. Если пользователь описывает структуру, состоящую из двух 16-битовых величин, то структура поместится в 32-битовый регистр.

Си++ проектировался для использования в довольно традиционной среде компиляции и выполнения, среде программирования на Си в системе UNIX. Средства обработки особых ситуаций и параллельного программирования, требующие нетривиальной загрузки и поддержки в процессе выполнения, не были включены в Си++. Вследствие этого реализация Си++ очень легко переносима. Однако есть полные основания использовать Си++ в среде, где имеется гораздо более существенная поддержка. Такие средства, как динамическая загрузка, пошаговая трансляция и база данных определений типов могут с пользой применяться без воздействия на язык.

си++ программирования масситти

Алгоритм выполнения задачи (блок-схема)



Описание задачи

Данная программа должна вывести все простые числа в заданном диапазоне (от 0 до n) при помощи алгоритма «Решето Эратосфена».

Сравнение кода программ по метрике Холстеда

Python:
NUOprtr (Number of Unique Operators) — число уникальных операторов программы на Python: 4
NUOprnd (Number of Unique Operands) — число уникальных операндов программы кода программы на Python: 4.
Noprtr (Number of Operators) — общее число операторов в программе на Python: 64.
Noprnd (Number of Operands) — общее число операндов в программе на Python: 4.

(Halstead Program Vocabulary, HPVoc): HPVoc = NUOprtr + NUOprnd = 5 + 5 = 8.
(Halstead Program Length, HPLen): HPLen = Noprtr + Noprnd = 64 + 4 = 64.

(Halstead Program Volume, HPVol): HPVol = HPLen log2 HPVoc = 87 log2 10 = 221.
(Halstead Difficulty, HDiff): HDiff = (NUOprtr/2) × (NOprnd / NUOprnd) = (64/2) × (4/4) = 32.
HEff = HDiff × HPVol = 32 × 221 = 7072.

Java:
NUOprtr (Number of Unique Operators) — число уникальных операторов программы на Java: 5.
NUOprnd (Number of Unique Operands) — число уникальных операндов программы кода программы на Java: 3
Noprtr (Number of Operators) — общее число операторов в программе на Java: 11.
Noprnd (Number of Operands) — общее число операндов в программе на Java: 39.

(Halstead Program Vocabulary, HPVoc): HPVoc = NUOprtr + NUOprnd = 9.
(Halstead Program Length, HPLen): HPLen = Noprtr + Noprnd = 50.
(Halstead Program Volume, HPVol): HPVol = HPLen log2 HPVoc = 54.
(Halstead Difficulty, HDiff): HDiff = (NUOprtr/2) × (NOprnd / NUOprnd) = (6/2) × (39 / 3) = 39.
HEff = HDiff × HPVol = 2106.

C++:
NUOprtr (Number of Unique Operators) — число уникальных операторов программы на C++: 7.
NUOprnd (Number of Unique Operands) — число уникальных операндов программы кода программы на C++: 3.
Noprtr (Number of Operators) — общее число операторов в программе на C++: 48.
Noprnd (Number of Operands) — общее число операндов в программе на C++: 13.

(Halstead Program Vocabulary, HPVoc): HPVoc = NUOprtr + NUOprnd = 10.
(Halstead Program Length, HPLen): HPLen = Noprtr + Noprnd = 61.
(Halstead Program Volume, HPVol): HPVol = HPLen log2 HPVoc = 60.
(Halstead Difficulty, HDiff): HDiff = (NUOprtr/2) × (NOprnd / NUOprnd) = (7/2) × (13/3) = 14.

HEff = HDiff × HPVol = 840.

Решение задачи

1. Все четные числа, кроме двойки, - составные, т. е. не являются простыми, так как делятся не только на себя и единицу, а также еще на 2.
2. Все числа кратные трем, кроме самой тройки, - составные, так как делятся не только на самих себя и единицу, а также еще на 3.
3. Число 4 уже выбыло из игры, так как делится на 2.
4. Число 5 простое, так как его не делит ни один простой делитель, стоящий до него.
5. Если число не делится ни на одно простое число, стоящее до него, значит оно не будет делиться ни на одно сложное число, стоящее до него.

Последний пункт вытекает из того, что сложные числа всегда можно представить как произведение простых. Поэтому если одно сложное число делится на другое сложное, то первое должно делиться на делители второго. Например, 12 делится на 6, делителями которого являются 2 и 3. Число 12 делится и на 2, и на 3.

Код выполнения программы

1) Python:

```
2)  список заполняется значениями от 0 до n
3)  a = []
4)  for i in range(n + 1):
5)      a.append(i)
6)
7)  # Вторым элементом является единица,
8)  # которую не считают простым числом
9)  # забиваем ее нулем.
10)  a[1] = 0
11)
12)  # начинаем с 3-го элемента
13)  i = 2
14)  while i <= n:
15)      # Если значение ячейки до этого
16)      # не было обнулено,
17)      # в этой ячейке содержится
18)      # простое число.
19)      if a[i] != 0:
20)          # первое кратное ему
21)          # будет в два раза больше
22)          j = i + i
23)          while j <= n:
24)              # это число составное,
25)              # поэтому заменяем его нулем
26)              a[j] = 0
27)              # переходим к следующему числу,
28)              # которое кратно i
29)              # (оно на i больше)
30)              j = j + i
31)      i += 1
32)
33)  # Превращая список во множество,
34)  # избавляемся от всех нулей кроме одного.
35)  a = set(a)
36)  # удаляем ноль
```

```
37) a.remove(0)
38) print(a)
```
39) java

```
mport java.util.ArrayList;
import java.util.List;

public class SieveEratosthenes {
    static class PrimePair {
        Integer prime;
        Integer lastCrossed;

        PrimePair(Integer prime, Integer lastCrossed) {
            this.prime = prime;
            this.lastCrossed = lastCrossed;
        }
    }

    private List<PrimePair> primes;

    private SieveEratosthenes() {
        primes = new ArrayList<>();
        primes.add(new PrimePair(2, 2));
        primes.add(new PrimePair(3, 3));
    }

    private void fillNPrimes(int n) {
        while (primes.size()<n) {
            addNextPrime();
        }
    }

    private void addNextPrime() {
        int candidate = primes.get(primes.size()-1).prime + 2;
        for (int i = 1; i < primes.size(); i++) {
            PrimePair p = primes.get(i);
            while (p.lastCrossed < candidate) {
                p.lastCrossed += p.prime;
            }
            if (p.lastCrossed == candidate) {
                //restart
                candidate+=2;
                i=-1;
            }
        }
        System.out.println(candidate);
        primes.add(new PrimePair(candidate, candidate));
    }
```

```
    public static void main(String[] args) {
        SieveEratosthenes test = new SieveEratosthenes();
        test.fillNPrimes(1000);
    }
}
```

40) C++

```
41)  int n;
42)  vector<char> prime (n+1, true);
43)  prime[0] = prime[1] = false;
44)  for (int i=2; i<=n; ++i)
45)          if (prime[i])
46)                  if (i * 1ll * i <= n) //1ll для перевода в long long
47)                          for (int j=i*i; j<=n; j+=i)
48)                                  prime[j] = false;
```

Выводы

В данной курсовой работе мною была разработана программа для которая генерируется множество первых n случайных чисел с помощью решета Эратосфена, используя 3 различных языка программирования, а именно Python, Java и C++. Был обоснован выбор каждого из этих языков по нескольким критериям и было проведено сравнение написанных программ по метрике Холстеда.

Сравнение данных программ подтвердило критерии по котором были выбраны данные языки для написания данной задачи и выявило положительные и отрицательные стороны каждого из этих языков программирования.

Список используемой литературы

Учебники, монография, брошюры

1. Программирование. Принципы и практика с использованием C++ — Бьёрн Страуструп.
2. Современный Java. Рецепты программирования – Кен Коузен.
3. Программирование на Python – Марк Лутц.

Электронные источники

1. Интернет-сайт Википедия- https://ru.wikipedia.org
2. Интернет-сайт Habr- https://habr.com/ru
3. Интернет-сайт tproger - https://tproger.ru
4. Интернет-сайт Oracle -https://www.oracle.com/java/technologies/javacommunity.html
5. Что пишут на Python: для чего нужен этот язык программирования / Skillbox Media
6. Моё обучение · Моё обучение · Stepik
7. Язык Java: зачем нужен, плюсы и минусы, области применения - сравнение с другими языками программирования, с чего начать (yandex.ru)