Объектно-ориентированное программирование. Лекция 1. Общее введение

Макаров П. А.

5 сентября 2024 г.

Содержание

1 Литература

1.1 Основная литература

1. *Вайсфельд М.* Объектно-ориентированное мышление.

2. *Мейер Б.* Объектно-ориентированное конструирование программных систем.

3. *Пышкин Е. В.* Основные концепции и механизмы объектно-ориентированного программирования.

4. *Пол А.* Объектно-ориентированное програмирование на C++.

5. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++.

6. *Андрианова А.А., Исмагилов Л.Н., Мухтарова Т.М.* Объектно-ориентированное програмирование на C++.

7. *Страуструп Б.* Язык программирования C++.

8. *Липпман С., Лажойе Ж.* C++ для начинающих.

9. *Столяров А. В.* Введение в язык Си++.

10. *Подбельский В. В.* Язык Си++.

11. http://www.cplusplus.com.

12. *Пилгрим М.* Погружение в Python 3.

13. https://www.python.org/doc/.

14. https://docs.python.org.

15. https://en.wikibooks.org/wiki/Python_Programming.

16. https://en.wikibooks.org/wiki/Python_Programming/Classes

17. https://docs.python.org/3/tutorial/classes.html

18. https://proglib.io/p/python-oop

19. https://younglinux.info/oopython/oop

1.2 Дополнительная литература

1. *Кнут Д.* Искусство программирования.

2. *Вирт Н.* Алгоритмы и структуры данных.

3. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы. Построение и анализ.

4. *Реймонд Э. С.* Искусство программирования для Unix.

5. *Брайант Р., О'Халларон Д.* Компьютерные системы: архитектура и программирование.

6. *Столяров А. В.* Программирование: введение в профессию.

7. *Керниган Б., Ритчи Д.* Язык программирования С.

8. *Подбельский В. В., Фомин С. С.* Программирование на языке Си.

9. *Хэзфилд Р., Кирби Л.* Искусство программирования на С: Фундаментальные алгоритмы, структуры данных и примеры приложений.

10. *Stallman R.* GNU C Language Intro and Reference Manual.

2 Основные понятия и термины

Определение 1. ***Программа*** *(в широком смысле) — это текст, представляющий собой исчерпывающее описание действий исполнителя и написанный на понятном ему языке.*

В рамках нашего курса мы будем иметь в виду исключительно *компьютерные программы,* то есть тексты исполняемые именно *компьютерами.*

Определение 2. *Под* ***компьютерами*** *будем подразумевать любые электронные устройства, способные исполнять программы, независимо от их конкретной архитектуры. Часто в качестве синонима используются понятия* ***вычислительная машина/система****.*

Очень часто программы пишут не на языке исполнительного устройства, а на других языках (по тем или иным причинам более удобным для программиста), которые при этом называют *языками программирования.*

Определение 3. ***Язык программирования*** *— это формальный язык, предназначенный для записи компьютерных программ. Язык программирования (как и любой естественный язык) представляет собой совокупность лексических, синтаксических и семантических правил.*

В настоящее время существует тысячи языков программирования и множество вариантов их классификации. Приведём несколько основных вариантов классификаций.

- По уровню применяемых абстракций:
  - низкого;
  - среднего;
  - высокого уровня.
- По режиму исполнения программ:
  - транслируемые;
  - интерпретируемые.
- По используемой типизации:
  - со статической;
  - либо динамической типизацией.
- По поддержке различных компьютерных архитектур:
  - непереносимые;
  - переносимые;
  - кроссплатформенные.
- По назначению:
  - унивесальные (общего назначения);
  - специализированные.
- По возможности следования парадигме программирования:
  - императивные;
  - процедурные;
  - функциональные;
  - логические;
  - объектно-ориентированные.

Действительно ли являются компьютерными программами тексты, приведённые в листингах 1 и 2?

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
```

```
6     cout << "Hello, world!" << endl;
7     return 0;
8 }
```

Листинг 1: Пример текста, написанного на языке C++

```
1 #!/usr/bin/python3.8
2
3 print("Hello, world!")
```

Листинг 2: Пример текста, написанного на языке Python

Определение 4. ***Исходный код*** *— это текст компьютерной программы на каком-либо языке программирования или языке разметки, который может быть прочтён человеком. Исходный код транслируется в исполняемый код целиком до запуска программы при помощи транслятора либо исполняется непосредственно при помощи интерпретатора. В широком смысле, исходный код — это любые входные данные для транслятора.*

Определение 5. ***Транслятор*** *— это программа, выполняющая* ***трансляцию****, то есть преобразование программы, представленной на одном из языков программирования, в программу на другом языке.*

Процесс трансляции программ, написанных на языке C++ включает следующие этапы.

1. Обработка исходного текста *препроцессором.*
2. *Компиляция.*
3. *Ассемблирование.*
4. *Компоновка.*

Определение 6. ***Интерпретатор*** *— программа, выполняющая* ***интерпретацию*** *— построчный анализ, обработку и выполнение исходного кода программы.*

Существует две основные разновидности интерпретаторов.

1. Простые.
2. Компилирующего типа.

Алгоритм работы простого интерпретатора:

1. прочитать инструкцию;

2. проанализировать инструкцию и определить соответствующие действия;

3. выполнить соответствующие действия;

4. если не достигнуто условие завершения программы, прочитать следующую инструкцию и перейти к пункту 2.

Некоторые интерпретаторы (например, для языка Python и многих других) могут работать в режиме диалога или так называемого *цикла чтения-вычисления-печати* (REPL — read-eval-print loop). В таком режиме интерпретатор считывает законченную конструкцию языка, выполняет её, печатает результаты, после чего переходит к ожиданию ввода пользователем следующей конструкции.

Пример 1. *Фреймворк ROOT, используемый для анализа данных в области физики высоких энергий, предоставляет пользователю возможность работать в режиме диалога с помощью интерпретаторов языков C++, Python либо R.*

3 Стандартизация языков

3.1 Стандарты языка C++

По приведённым ссылкам находятся черновики, наиболее близкие к финальным вариантам соответствующих стандартов.

- ISO/IEC 14882:1998 (C++98).
- ISO/IEC 14882:2003 (C++03) https://cs.nyu.edu/courses/fall11/CSCI-GA.2110-003/documents/c++2003std.pdf.
- ISO/IEC 14882:2011 (C++11).
- ISO/IEC 14882:2014 (C++14).
- ISO/IEC 14882:2017 (C++17).
- ISO/IEC 14882:2020 (C++20).

3.2 Стандартизация Python

Python, как и многие другие современные языки с открытым исходным кодом, формально не стандартизирован никакими “внешними” организациями, но де-факто имеет стандарт, подробно описанный в официальной документации. Кратко перечислим самые основные документы, являющиеся наиболее близким к тому, что можно считать стандартом языка Python.

- The Python Language Reference.
- The Python Standard Library.
- PEP 0 — Index of Python Enhancement Proposals.
- PEP 8 — Style Guide for Python Code.
- PEP 257 — Docstring Conventions.

4 Основы архитектуры

Определение 7. ***Архитектура компьютера*** *— это концептуальная модель компьютерной системы, определяемая её компонентами и их взаимодействиями между собой и окружением. Кроме того, в понятие архитекуры включаются принципы её проектирования и развития.*

Выделяют несколько уровней организации компьютерной архитектуры. Приведём схему, иллюстрирующую многоуровневую структуру компьютера.

-1. Уровень физических устройств.
0. Цифровой логический уровень.
1. Микроархитектурный уровень.
2. Уровень архитектуры системы команд.
3. Уровень операционной системы.
4. Уровень языка ассемблера.
5. Язык высокого уровня.

5 Основы операционных систем

Определение 8. ***Операционная система*** *— это комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.*

Основные этапы развития компьютеров и операционных систем.

- Пакетный режим. Необходимость оптимального использования дорогостоящих вычислительных ресурсов привела к появлению концепции пакетного режима исполнения программ. Пакетный режим предполагает наличие очереди программ на исполнение, причём система может обеспечивать загрузку программы с внешних носителей данных в оперативную память, не дожидаясь завершения исполнения предыдущей программы, что позволяет избежать простоя процессора.

- Разделение времени и многозадачность. Уже пакетный режим в своём развитом варианте требует разделения процессорного времени между выполнением нескольких программ. Необходимость в разделении времени (многозадачности, мультипрограммировании) проявилась ещё сильнее при распространении в качестве устройств ввода-вывода телетайпов (а позднее, терминалов с электронно-лучевыми дисплеями) (1960-е годы). Поскольку скорость клавиатурного ввода (и даже чтения с экрана) данных оператором намного ниже, чем скорость обработки этих данных компьютером, использование компьютера в "монопольном" режиме (с одним оператором) могло привести к простою дорогостоящих вычислительных ресурсов. Разделение времени позволило создать "многопользовательские" системы, в которых один (как правило) центральный процессор и блок оперативной памяти соединялся с многочисленными терминалами. При этом часть задач (таких как ввод или редактирование данных оператором) могла исполняться в режиме диалога, а другие задачи (такие как массивные вычисления) — в пакетном режиме.

- Разделение полномочий. Распространение многопользовательских систем потребовало решения задачи разделения полномочий, позволяющей избежать возможности изменения исполняемой программы или данных одной программы в памяти компьютера другой программой (намеренно или по ошибке), а также изменения самой системы прикладной программой. Реализация разделения полномочий в операционных системах была поддержана разработчиками

процессоров, предложивших архитектуры с двумя режимами работы процессора — "реальным" (в котором исполняемой программе доступно всё адресное пространство компьютера) и "защищённым" (в котором доступность адресного пространства ограничена диапазоном, выделенным при запуске программы на исполнение).

- Системы реального времени.
- Гибридные системы.

Есть приложения вычислительной техники, для которых операционные системы излишни. Например, встроенные микрокомпьютеры, содержащиеся во многих бытовых приборах, автомобилях, простейших сотовых телефонах и т. д. Операционные системы используются:

- если нужен универсальный механизм хранения данных;
- для предоставления программам системных библиотек с часто используемыми подпрограммами;
- для распределения полномочий;
- необходима возможность имитации "одновременного" исполнения нескольких программ на одном компьютере;
- для управления процессами выполнения отдельных программ.

Таким образом, современные универсальные операционные системы можно охарактеризовать, прежде всего, как:

- использующие файловые системы (с универсальным механизмом доступа к данным);
- многопользовательские (с разделением полномочий);
- многозадачные (с разделением времени).

Определение 9. ***Файловая система*** *— это способ хранения данных на внешних запоминающих устройствах.*

Функции операционных систем.

- Основные функции:
  - загрузка программ в оперативную память и их выполнение.

  - исполнение запросов программ (ввод и вывод данных, запуск и остановка других программ, выделение и освобождение дополнительной памяти и др.).
  - стандартизованный доступ к периферийным устройствам.
  - управление оперативной памятью (распределение между процессами, организация виртуальной памяти).
  - управление доступом к данным на энергонезависимых носителях (жёсткие, оптические диски и др.), организованным в той или иной файловой системе.
  - обеспечение пользовательского интерфейса.
  - сохранение информации об ошибках системы.

- Дополнительные функции:
  - параллельное или псевдопараллельное выполнение задач (многозадачность).
  - эффективное распределение ресурсов вычислительной системы между процессами.
  - разграничение доступа различных процессов к ресурсам.
  - организация надёжных вычислений (невозможности одного вычислительного процесса повлиять на вычисления в другом процессе), основана на разграничении доступа к ресурсам.
  - взаимодействие между процессами: обмен данными, взаимная синхронизация.
  - защита самой системы, а также пользовательских данных и программ от действий пользователей или приложений.
  - многопользовательский режим работы и разграничение прав доступа.

Многозадачность и распределение полномочий требуют определённой иерархии привилегий компонентов в самой операционной системе. В составе операционной системы различают три группы компонентов.

- Ядро и планировщик. Драйверы устройств, непосредственно управляющие оборудованием. Сетевая и файловая системы.
- Системные библиотеки.
- Оболочка с утилитами.

Большинство программ, как системных (входящих в операционную систему), так и прикладных, исполняются в непривилегированном ("пользовательском") режиме работы процессора и получают доступ к оборудованию (и, при необходимости, к другим ресурсам ядра, а также ресурсам иных программ) только посредством системных вызовов. Ядро исполняется в привилегированном режиме: именно в этом смысле говорят, что система (точнее, её ядро) управляет оборудованием.

Определение 10. ***Ядро*** — *центральная часть операционной системы, управляющая ресурсами компьютера, выполнением процессов и обеспечивающая координированный доступ процессов к ресурсам.*

Основными ресурсами являются *процессорное время, память* и *устройства ввода-вывода.* Доступ к файловой системе и сетевое взаимодействие также могут быть реализованы на уровне ядра. Перечислим основные объекты ядра операционной системы.

- Процессы.
- Файлы.
- События.
- Потоки.
- Семафоры.
- Мьютексы.
- Каналы.

Как основополагающий элемент операционной системы, ядро представляет собой наиболее низкий уровень абстракции для доступа приложений к ресурсам вычислительной системы, необходимым для их работы. Как правило, ядро предоставляет такой доступ исполняемым процессам соответствующих приложений за счёт использования механизмов межпроцессного взаимодействия и обращения приложений к системным вызовам операционной системы.

Определение 11. ***Системный вызов*** — *это элемент пользовательского интерфейса ядра, представляющий собой обращение прикладной программы к ядру операционной системы для выполнения той или иной операции.*

С целью обеспечения совместимости системные вызовы стандартизируются, а доступ к ним в пользовательском режиме предоставляется с помощью интерфейса в виде стандартной библиотеки. При этом с точки зрения прикладного программиста, системные вызовы реализуются в виде функций, прототипы которых следуют соглашениям API.

Определение 12. ***Программный интерфейс приложения (API — Application Programming Interface)*** *— это описание способов взаимодействия одной компьютерной программы с другими. Обычно входит в описание интернет-протоколов, программных фреймворков или стандарта вызовов функций операционной системы.*

Системный программист, работающий с системными вызовами на уровне ядра, кроме API часто сталкивается с понятием ABI.

Определение 13. ***Двоичный интерфейс приложений (ABI — Application Binary Interface)*** *— это набор соглашений для доступа приложения к операционной системе и другим низкоуровневым сервисам, спроектированный для переносимости исполняемого кода между машинами, имеющими совместимые ABI.*

В отличие от API, который регламентирует совместимость на уровне исходного кода, ABI можно рассматривать как набор правил, позволяющих компоновщику объединять откомпилированные модули компонента без перекомпиляции всего кода. Двоичный интерфейс приложений регламентирует:

- конкретный набор инструкций процессора;
- использование регистров процессора;
- соглашение о вызове функций;
- состав и формат системных вызовов и вызовов одного модуля другим;
- форматы исполняемых файлов.

UNIX и UNIX-like ОС, стандартизация и POSIX

К концу 1960-х годов отраслью и научно-образовательным сообществом был создан целый ряд операционных систем, реализующих все или часть очерченных выше функций. К ним относятся Atlas (Манчестерский университет), CTTS и ITS (Массачусетский технологический институт,

MIT), THE (Эйндховенский технологический университет), RS4000 (Университет Орхуса) и другие. Всего эксплуатировалось более сотни различных ОС.

Наиболее развитые операционные системы, такие как OS/360 (IBM), SCOPE (CDC) и завершённый уже в 1970-х годах Multics (MIT и Bell Labs), предусматривали возможность исполнения на многопроцессорных компьютерах.

Эклектичный характер разработки операционных систем привёл к нарастанию кризисных явлений, прежде всего, связанных с чрезмерными сложностью и размерами создаваемых систем. Системы были плохо масштабируемыми (более простые не могли использовать все возможности крупных вычислительных систем; более развитые неоптимально исполнялись на малых или не могли исполняться на них вовсе) и полностью несовместимыми между собой, их разработка и совершенствование затягивались.

Задуманная и реализованная в 1969 году Кеном Томпсоном при участии нескольких коллег (включая Денниса Ритчи и Брайана Кернигана), операционная система UNIX (первоначально UNICS, что обыгрывало название Multics) вобрала в себя многие черты более ранних систем, но обладала целым рядом свойств, отличающих её от большинства предшественниц:

- простая метафорика (два ключевых понятия: вычислительный процесс и файл);
- компонентная архитектура: принцип “одна программа — одна функция” плюс мощные средства связывания различных программ для решения возникающих задач (“оболочка”);
- минимизация ядра (кода, выполняющегося в “реальном” (привилегированном) режиме процессора) и количества системных вызовов;
- независимость от аппаратной архитектуры и реализация на машинонезависимом языке программирования (язык программирования Си стал побочным продуктом разработки UNIX);
- унификация файлов.

UNIX, благодаря своему удобству прежде всего в качестве инструментальной среды (среды разработки), обрела популярность сначала в университетах, а затем и в отрасли, получившей прототип единой операционной системы, которая могла использоваться на самых разных вычислительных системах и, более того, могла быть быстро и с минимальными

усилиями перенесена на любую вновь разработанную аппаратную архитектуру.

В конце 1970-х годов сотрудники Калифорнийского университета в Беркли внесли ряд усовершенствований в исходные коды UNIX, включая работу с протоколами TCP/IP. Их разработка стала известна под именем BSD (Berkeley Software Distribution).

Задачу разработать независимую (от авторских прав Bell Labs) реализацию той же архитектуры поставил и Ричард Столлман, основатель проекта GNU.

Благодаря конкурентности реализаций архитектура UNIX стала вначале фактическим отраслевым стандартом, а затем обрела статус и стандарта юридического — ISO/IEC 9945 (POSIX).

Только системы, отвечающие спецификации Single UNIX Specification, имеют право носить имя UNIX. К таким системам относятся AIX, HP-UX, IRIX, Mac OS X, SCO OpenServer, Solaris, Tru64 и z/OS.

Операционные системы, следующие стандарту POSIX или опирающиеся на него, называют "POSIX-совместимыми" (чаще встречается словоупотребление "UNIX-подобные" или "семейство UNIX", но оно противоречит статусу торгового знака "UNIX", принадлежащего консорциуму The Open Group и зарезервированному для обозначения только операционных систем, строго следующих стандарту). Сертификация на совместимость со стандартом платная, из-за чего некоторые системы не проходили этот процесс, однако считаются POSIX-совместимыми по существу.

К UNIX-подобным относятся операционные системы, основанные на последней версии UNIX, выпущенной Bell Labs (System V), на разработках университета Беркли (FreeBSD, OpenBSD, NetBSD), на основе Solaris (OpenSolaris, BeleniX, Nexenta OS), а также Linux, разработанная в части утилит и библиотек проектом GNU и в части ядра — сообществом, возглавляемым Линусом Торвальдсом.

Стандартизация операционных систем преследует цель упрощения замены самой системы или оборудования при развитии вычислительной системы или сети и упрощении переноса прикладного программного обеспечения (строгое следование стандарту предполагает полную совместимость программ на уровне исходного текста; из-за профилирования стандарта и его развития некоторые изменения бывают всё же необходимы, но перенос программы между POSIX-совместимыми системами обходится на порядки дешевле, чем между альтернативными), а также преемственность опыта пользователей.

Самым заметным эффектом существования этого стандарта стало эффективное разворачивание Интернета в 1990-х годах.

6 Основы компьютерных сетей

- Клиент-серверная архитектура. https://habr.com/ru/post/495698/
- Сетевая модель OSI.
- Стек протоколов TCP/IP.
- Интерфейс.
- Порт.
- Сокет.

7 Инструменты разработчика

- Текстовый редактор.
- Набор транслятора либо интерпретатор.
- Библиотека необходимых функций и компонентов.
- Система сборки.
- Отладчик.
- Профилировщик кода.
- Система контроля версий.
- Генератор документации.

Определение 14. ***Интегрированная среда разработки*** *— это комплекс программных средств, используемый программистами для разработки программного обеспечения (IDE — Integrated Development Environment).*

8 Контрольные вопросы и задания

1. Найдите и изучите информацию о HTML, CSS, Perl, PHP, Java, JavaScript, SQL, Bash, TEX, Asymptote, Gnuplot. Является ли всё перечисленное языками программирования? Почему? Классифицируйте из данного списка всё, что является языками программирования.

2. Что такое язык разметки? Язык сценариев? Компьютерный язык? Как соотносятся все эти понятия с термином язык программирования?

3. Что такое полнота по Тьюрингу и какое это имеет отношение к программированию?

4. Найдите информацию о наиболее популярных трансляторах языка C++ и изучите их основные особенности. Существуют ли интерпретаторы языка C++?

5. Какие интерпретаторы Python вам известны? К какому типу они относятся и каков принцип их работы?

6. С какими архитектурами компьютеров вы работали? Каковы их основные особенности? Наидите и изучите информацию о неизвестных вам архитектурах.

7. Назовите существующие интерфейсы взаимодействия пользователя с операционной системой. Каковы их преимущества и недостатки?

8. Определите значения понятий, перечисленных в разделе 6.

9. Каковы, на ваш взгляд, преимущества и недостатки использования IDE для разработки ПО? С какими инструментами разработчика у вас есть опыт работы?