**Конспект по теме «Этап реализации»**

1. **Язык программирования** — [формальная](https://dic.academic.ru/dic.nsf/ruwiki/90924) [знаковая система](https://dic.academic.ru/dic.nsf/ruwiki/929891), предназначенная для записи [компьютерных программ](https://dic.academic.ru/dic.nsf/ruwiki/6823). Язык программирования определяет набор [лексических](https://dic.academic.ru/dic.nsf/ruwiki/29259), [синтаксических](https://dic.academic.ru/dic.nsf/ruwiki/357868) и [семантических](https://dic.academic.ru/dic.nsf/ruwiki/681558) правил, задающих внешний вид программы и действия, которые выполнит исполнитель (компьютер) под её управлением.

**Алфавит языка программирования** — это все символы или комбинации символов, которые используются при программировании на этом **языке**.

**Синтаксис языка программирования** - это правила составления предложений языка из отдельных предложений. Такие предложения - это операции, операторы, определения функций, переменные, разделы описания и т.к., в том числе и программа в целом.

**Семантика в** [**программировании**](https://dic.academic.ru/dic.nsf/ruwiki/8183) — дисциплина, изучающая формализации значений конструкций [языков программирования](https://dic.academic.ru/dic.nsf/ruwiki/1465) посредством построения их формальных математических моделей. В качестве инструментов построения таких моделей могут использоваться различные средства, например, [математическая логика](https://dic.academic.ru/dic.nsf/ruwiki/8918), [λ-исчисление](https://dic.academic.ru/dic.nsf/ruwiki/756513), [теория множеств](https://dic.academic.ru/dic.nsf/ruwiki/1358), [теория категорий](https://dic.academic.ru/dic.nsf/ruwiki/19391), [теория моделей](https://dic.academic.ru/dic.nsf/ruwiki/307233), [универсальная алгебра](https://dic.academic.ru/dic.nsf/ruwiki/96852). Формализация семантики языка программирования может использоваться как для описания языка, определения свойств языка, так и для целей [формальной верификации](https://dic.academic.ru/dic.nsf/ruwiki/576384) программ на этом языке программирования.

**Стандарт языка программирования** — это предмет документации, который определяет язык **программирования**, чтобы пользователи и разработчики **языка** могли согласовывать, что означают программы на данном **языке**.

1. Классификация языков программирования:
   1. Процедурные языки
   2. Языки программирования низкого уровня
   3. Языки программирования высокого уровня
   4. Объектно-ориентированные языки
   5. Декларативные языки программирования
   6. Функциональные языки программирования
   7. Логические языки программирования
   8. Языки сценариев (скрипты)
   9. Языки, ориентированные на данные
2. Критерии выбора языка программирования:

* Скорость работы конечного продукта.
* Объем занимаемой оперативной памяти.
* Скорость разработки программы.
* Ориентированность на компьютер или человека
* Кроссплатформенность.
* Скорость внесения изменений, скорость тестирования

1. **Парадигма программирования** — это совокупность идей и понятий, определяющих стиль написания [компьютерных программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) (подход к программированию). Это способ [концептуализации](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%86%D0%B5%D0%BF%D1%82%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BB%D0%B8%D0%BD%D0%B3%D0%B2%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0)), определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

**Процедурное программирование** – это программирование, при котором программа представляет собой последовательность операторов. Используется в языках высокого уровня Basic, Fortran и др.

**Функциональное программирование** – это программирование, при котором программа представляет собой последовательность вызовов функций. Используется в языках Lisp и др.

**Логическое программирование** – это программирование, при котором программа представляет собой совокупность определения соотношений между объектами. Используется в языках Prolog и др.

**Объектно-ориентированное программирование** – это программирование, при котором основой программы является объект, представляющий собой совокупность данных и правил их преобразования. Используется в языках Turbo-Pascal, C++ и др.

5)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Название парадигмы** | **Сущность**  **парадигмы,**  **основные**  **идеи,**  **принципы,**  **объекты.** | **Языки**  **поддерживающие данную**  **парадигму.**  **(2-3 языка)** | **Достоинства** | **Недостатки** |
| **1** | Императивное  программирование | В исходном коде записываются «приказы» команды, а не классы  Все инструкции должны выполняться последовательно, один за другим.  После выполнения инструкций данные могут записываться в память и считываться памяти. | Ассамблер, fortran, algo | эффективная реализация | Нет возможности перескакивать с одного участка кода на другой |
| **2** | Декларативное  программирование | Декларативная программа состоит из ограничений и правил, из которых компьютер генерирует способ получения результата | SQL, MySQL | позволяет писать более безопасный и поддерживаемый код, который легко параллелится. А компиляторы декларативных языков имеют больше возможностей при оптимизации программ. | Сложно в понимании |
| **3** | Структурное  программирование | представление программы в виде иерархической структуры блоков. | C, Pascal, Basic | позволяет сократить число вариантов построения программы по одной и той же спецификации  логически связанные операторы находятся визуально ближе, а слабо связанные — дальше  Сильно упрощается процесс тестирования и отладки структурированных программ. | работа со структурными данными практически не возможна без использования скриптов и знания как эти скрипты функционируют |
| **4** | Процедурное  программирование | процедура может иметь несколько точек выхода (return в C-подобных языках), несколько точек входа, иметь аргументы, возвращать значение как результат своего выполнения, быть перегруженной по количеству или типу параметров и много чего еще. | C, C++, Pascal | отлично подходит для программирования общего назначения  Закодированная простота наряду с простотой реализации компиляторов и интерпретаторов  Исходный код переносим  Код может быть повторно использован в разных частях программы, без необходимости копировать его  Благодаря методике процедурного программирования требования к памяти также сокращаются  Ход программы можно легко отследить | Код программы труднее писать  Процедурный код часто не может быть использован повторно, что может привести к необходимости воссоздания кода, если это необходимо для использования в другом приложении.  Сложно общаться с реальными объектами  Данные открыты для всей программы, что делает их не очень безопасными |
| **5** | Модульное  программирование | Программа описанная в стиле модульного программирования — это набор модулей. Что внутри, классы, императивный код или чистые функции — не важно. | Haskell, Pascal | Повышение удобства сопровождения кода.  Совместная работа нескольких человек не мешает друг другу. | невозможно загрузить модули по запросу. |
| **6** | Объектно-  ориентированное  программирование | основанно на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования | Java, Python, C# | **Параллельная разработка, Модульность, Безопасность. Возможность повторного использования** | Низкая **производительность**  **Требуется больше планирования** |
| **7** | Функциональное  программирование | Программы создаются путем последовательного применения функций, а не инструкций | Haskell, F# | Легкая отладка, Параллельное программирование, Модульность, Отложенное вычисление | плохо подходит для алгоритмов, основанных на графах из-за сравнительно более медленной работы программы |
| **8** | Логическое  программирование | Подход к программированию, при  котором программа задаѐтся совокупностью правил без явного указания  последовательности их применения. | Prolog, Delta Prolog | Компактность кода, Легкость понимания, Эффективный метод вычислений - рекурсия | Невозможность создания комплексных задач |

**Стандарт** **оформления** **кода**  — набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования. Наличие общего стиля программирования облегчает понимание и поддержание исходного кода, написанного более чем одним программистом, а также упрощает взаимодействие нескольких человек при разработке программного обеспечения.

**Венгерская** **нотация** в программировании — соглашение об именовании переменных, констант и прочих идентификаторов в коде программ.

**Верблюжья** нотация — практика написание текстов, которая придерживается принципа игнорирования пробелов и знаков препинания.

**Грамотный выбор идентификаторов:**

* Поднимайтесь на один уровень абстракции выше тела элемента (за исключением имён классов).
* Имя класса должно описывать его обязанность.
* Уважайте принцип единственной обязанности (одно из SOLID-правил).
* Разбивайте задачу на более мелкие подзадачи.

Для форматирования кода должны использоваться символы табуляции и пробела. Символ табуляции должен использоваться для оформления отступов, а символ пробела - для разделения лексем в коде.

**Отступы**

Отступы (в виде одного символа табуляции) должны использоваться в следующих случаях:

* В блоке описания переменных. Строки, непосредственно описывающие переменные, должны располагаться под строкой с оператором **Var** и иметь отступ:
* В многострочных выражениях, в описаниях параметров процедур и функций, не умещающихся в одну строку:
* В описаниях процедур и функций. Код, расположенный между операторами **Begin** и **End**, должен иметь отступ:
* В описаниях классов и перечислимых типов:
* В операторе выбора. Во-первых, должен быть отступ перед каждым вариантом выбора. Во-вторых, код, относящийся к каждому варианту, должен начинаться в следующей строке, и иметь отступ по отношению к строке с оператором Case/Else:
* В условном операторе:
* В операторах циклов:
* В операторе обработки исключительных ситуаций:

Во всех остальных случаях использование отступов не допустимо.

**Пробелы**

Пробелы должны использоваться для разделения лексем в исходном коде. Несколько последовательно идущих пробелов между лексемами должны заменяться одним.

Пробелы должны использоваться для разделения лексем в следующих случаях:

* до и после ключевых слов языка;
* до и после знака присваивания в операторе присваивания;
* до и после знаков операций в выражениях;
* после запятой в параметрах при описании и вызове функций и в других перечислениях элементов;
* после знака вопроса, а также до и после двоеточия в операции выбора;
* после двоеточия в описании переменных, свойств и параметров функций.

Пробелы не должны использоваться для разделения лексем в следующих случаях:

* перед точкой с запятой;
* после открывающей и перед закрывающей круглыми скобками;
* перед запятой в параметрах при описании и вызове функций и в других перечислениях;
* перед двоеточием в описаниях переменных, свойств и параметров функций.

**Пустые строки**

Пустая строка должна использоваться в следующих случаях:

* для отделения деклараций классов и типов друг от друга;
* для отделения реализаций методов и функций друг от друга.

**Прочее**

Операторы, относящиеся к одной конструкции языка (**Begin** – **End**, **Repeat** – **Until**, **Try** – **Except** – **Finally** – **End** **Try**, **Select** – **End Select**, и др.) должны находиться в одной позиции в строке (в одной колонке).

**6 правил по оформлению комментариев в коде:**

## Комментарии не должны дублировать код.

## Хорошие комментарии не оправдывают непонятный код.

## Добавляйте ссылки на исходный код, который вы скопировали.

## Добавляйте ссылки на внешние примеры в тех случаях, когда это полезнее всего.

## Исправляя багги, добавляйте комментарии.

## Помечайте комментариями незаконченные реализации.

## 6 принципов именования переменных:

1. **Следуйте S-I-D**Имя должно быть коротким (**S**hort), интуитивно понятным (**I**ntuitive) и описательным (**D**escriptive).
2. **Избегайте сокращений**  
   Не используйте сокращения. Обычно они только ухудшают читаемость кода. Найти короткое, описательное имя может быть сложно, но сокращения не могут быть оправданием для того, чтобы этого не делать
3. **Избегайте дублирования контекста**  
   Всегда удаляйте контекст из имени, если это не снижает его читабельность.
4. **Отражайте в имени ожидаемый результат**
5. **Учитывайте единственное/множественное число**  
   Как и префикс, имена переменных могут быть единственного или множественного числа в зависимости от того, имеют ли они одно значение или несколько.
6. **Используйте осмысленные и произносимые имена**