**Лабораторная работа No15**

Выбор и обоснование выбора среды разработки программы. Изучение различных стилей программирования, правил формирования листинга программы.

**Цель работы:**

1. Изучить критерии выбора языка программирования. Научиться обосновывать выбор среды разработки в

соответствии с критериями выбора языка программирования.

2. Изучить основные парадигмы программирования.

3. Изучить правила формирования листинга программы.

1. Дайте определение понятию: **язык программирования**. Что такое алфавит, **синтаксис, семантика и** **стандарт** языка программирования?

**Язык программирования** — [формальный язык](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA), предназначенный для записи [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). Язык программирования определяет набор [лексических](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D0%BA%D0%B0), [синтаксических](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [семантических](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) правил, определяющих внешний вид программы и действия, которые выполнит исполнитель (обычно — [ЭВМ](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%BE-%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0)) под её управлением.

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

**Синтаксис языка программирования** — набор правил, описывающий комбинации символов алфавита, считающиеся правильно структурированной программой или её фрагментом. Синтаксису языка противопоставляется его семантика.

**Семантика в программировании** — дисциплина, изучающая формализации значений конструкций языков программирования посредством построения их формальных математических моделей.

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

1. Приведите примеры известных вам **классификаций языков программирования.**

**Язык низкого уровня** – это язык программирования, предназначенный для определенного типа компьютера и отражающий его внутренний машинный код; языки низкого уровня часто называют машинно-ориентированными языками. Их сложно конвертировать для использования на компьютерах с разными центральными процессорами, а также довольно сложно изучать, поскольку для этого требуется хорошо знать внутренние принципы работы компьютера.

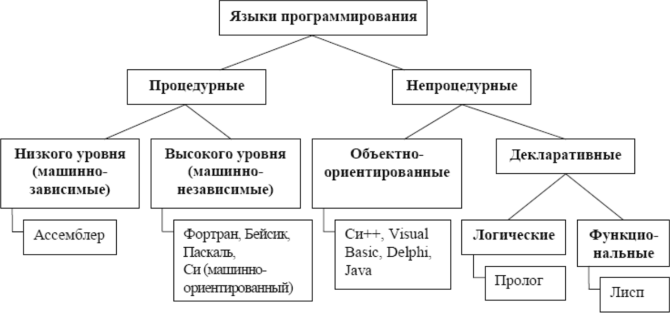
**Язык высокого уровня** – это язык программирования, предназначенный для удовлетворения требований программиста; он не зависит от внутренних машинных кодов компьютера любого типа. Языки высокого уровня используют для решения проблем, и поэтому их часто называют проблемно-ориентированными языками. Каждая команда языка высокого уровня эквивалентна нескольким командам в машинных кодах, поэтому программы, написанные на языках высокого уровня, более компактны, чем аналогичные программы в машинных кодах.

**В процедурных языках** программа явно описывает действия, которые необходимо выполнить, а результат задается только способом получения его при помощи некоторой процедуры, которая представляет собой определенную последовательность действий. В эту большую группу входят, например, ПАСКАЛЬ, С, АДА, ПЛ/1, ФОРТРАН и БЕЙСИК.

**В объектно-ориентированных языках** не описывают подробной последовательности действий для решения задачи, хотя они содержат элементы процедурного программирования. Программа пишется в терминах объектов, которые обладают свойствами и поведением. Объекты обмениваются сообщениями.

**В функциональных языках** программа описывает вычисление некоторой функции. Обычно эта функция задается как композиция других, более простых, те в свою очередь разлагаются на еще более простые и т.д. Один из основных элементов в функциональных языках – рекурсия, т.е. вычисление значения функции через значение этой же функции от других элементов. Присваивания и циклов в классических функциональных языках нет. Представителями этой группы являются ЛИСП, ML и Haskell.

**В логических языках** программа вообще не описывает действий. Она задает данные и соотношения между ними. После этого системе можно задавать вопросы. Машина перебирает известные и заданные в программе данные и находит ответ на вопрос. Порядок перебора не описывается в программе, а неявно задается самим языком. Классическим языком логического программирования считается ПРОЛОГ.



1. Перечислите **критерии выбора языка программирования**. Дайте краткую характеристику каждому критерию.

**Скорость работы конечного продукта.**

Требовательным к скорости выполнения могут быть программы с большим объемом математических вычислений, например моделирование физических систем, расчеты большого объема экономических данных, выведение трехмерной графики и прочее. Для данных целей хорошо подойдут компилируемые языки: ассемблер, С/С++, фортран и т.д.

**Объем занимаемой оперативной памяти.**

Данное требование появляется, когда программа разрабатывается для встраиваемых систем, мобильных платформ, микроконтроллеров и так далее. В данных случаях, чем меньше памяти расходует программа на данном языке – тем лучше. К таким языкам, опять же, относятся ассемблер, С/С++, Objective-C и другие.

**Скорость разработки программы**

Выбор падает на высокоуровневые языки Java, Flash и подобные.

**Ориентированность на компьютер или человека**

С кем будет работать программа в первую очередь? С человеком, или с компьютером? В пером случае программа должна обладать мощной графической частью, отвечающей требованиям дизайна и юзабилити. Разработка графической части зачастую требует достаточно много времени, т.к. отличается немалой сложностью. Здесь сложность возникает в том, что вывод графики – это немало математики, а значит присутствует требовательность к скорости исполнения, а из-за сложности разработки присутствует необходимость в высокоуровневом языке. В данном случае, на мой взгляд, очень хорошо подходит С++/C# с их одновременной и высокоуровневостью, и скоростью выполнения программ на них.

**Кроссплатформенность** – возможность работы программы на различных платформах, в различных ОС с минимальными изменениями. В этой сфере можно выделить такие языки: Java, C#,Flash,C++ с различными библиотеками и другие, менее используемые, языки. Java создавался с тем условием, что программы на данном языке должны работать на любой платформе, где есть JVM – Java Virtual Machine.

**Скорость внесения изменений, скорость тестирования**

Проект стремительно развивается, в него постоянно вносятся изменения, порой немало? Тогда выбор должен падать на высокоуровневые языки, где любой функциональный блок можно быстро переписать. Для подтверждения – я думаю, гораздо проще дебажить тот же С++, чем ассемблер. А еще проще Java.

1. Дайте определение понятию: **парадигма программирования (стиль программирования)**. Перечислите известные вам стили программирования.

**Стили программирования** - набор приемов или методов программирования, которые используют программисты, чтобы получить правильные, эффективные, удобные для применения и легкочитаемые программы.

**Императивное программирование** — это парадигма, основанная на составлении алгоритма действий (инструкций/команд), которые изменяют состояние (информацию/данные/память) программы.

**Декларативное программирование** — парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается ожидаемый результат, а не способ его получения.

**Структурное программирование** — [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в основе которой лежит представление программы в виде иерархической структуры [блоков](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)).

**Процедурное программирование** — парадигма, основанная на использовании процедур. Процедура (иногда также называемая подпрограммой или методом) — это последовательность команд, которые следует выполнить. Любая процедура может быть вызвана из любой точки программы, включая другие процедуры или ее же саму (рекурсивный вызов).

**Модульное**[**программирование**](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок.

**Объектно-ориентированное**[**программирование**](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) ([сокр.](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D0%B1%D1%80%D0%B5%D0%B2%D0%B8%D0%B0%D1%82%D1%83%D1%80%D0%B0) ООП) — [методология программирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), основанная на представлении программы в виде совокупности взаимодействующих [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), каждый из которых является экземпляром определённого [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), а классы образуют иерархию наследования.

**Модульное программирование** — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок.

**Функциональное программирование** — [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в которой процесс [вычисления](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) трактуется как вычисление значений [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) в математическом понимании последних (в отличие от [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) как подпрограмм в [процедурном программировании](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)).

**Логическое программирование** — [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), основанная на [математической логике](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0) — программы в ней задаются в форме логических утверждений и правил вывода.

**Примеры:**

**Императивный стиль = последовательность команд + переходы**

print "Hello";

print ", world!";

**Процедурный стиль = императивный + вызов других последовательностей команд (процедур)**

function Greeting {

print "Hello, world!";

}

Greeting;

**Функциональный стиль = императивный + процедуры принимают параметры и возвращают значения**

function Greeting(s) {

return "Hello, {s}!";

}

print Greeting(world);

**Объектно-ориентированный стиль = группировка кода в классы**

class World {

function Greeting() {

print "Hello, world!";

}

}

World.Greeting();

**Декларативный стиль = Программист не отвечает за реализацию.**

Greeting;

1. Дайте характеристику следующим **парадигмам программирования**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | **Название парадигмы** | **Сущность**  **парадигмы,**  **основные**  **идеи,**  **принципы,**  **объекты.** | **Языки**  **поддерживаю**  **щие данную**  **парадигму.**  **(2-3 языка)** | **Достоинства** | **Недостатки** | **Примеры**  **программ** |
| 1 | Императивное программирование | В исходном коде программы записываются [инструкции](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (команды);  инструкции должны выполняться последовательно;  данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями;  данные, полученные при выполнении инструкции, могут записываться в память. | C,  C++,  Java. | **Оригинальная форма программирования**  **Относительно прост в освоении**  **Легкая читаемость** | Код быстро становится очень обширным и, следовательно, запутанным  Повышенный риск ошибок при редактировании  Системно-ориентированное программирование означает, что обслуживание блокирует разработку приложений | function double (arr) {  let results = [];  for (let i = 0; i < arr.length; i++){  results.push(arr[i] \* 2);  }  return results;  } |
| 2 | Декларативное  программирование | [Парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в которой задаётся [спецификация](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) решения задачи, то есть описывается ожидаемый результат, а не способ его получения. | SQL, HTML | Более безопасный и поддерживаемый код. Компиляторы имеют больше возможность при оптимизации программ | Более сложный в понимании | <article>  <header>  <h1>Declarative Programming</h1>  <p>Sprinkle Declarative in your verbiage to sound smart</p>  </header>  </article> |
| 3 | Структурное программирование | В основе парадигмы лежит представление программы в виде иерархической структуры [блоков](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). | Basic  Cg  JOVIAL  Pascal. | Программы любой сложности и размера можно получать на основе соответствующего сочетания расширенного базиса управляющих структур  программа может быть проанализирована проверкой ее структуры, позволяя обнаруживать в ней ошибки уже на стадии проектирования | Локальные модификации могли нарушить работоспособность всей системы.  Далеко не все задачи поддаются алгоритмическому описанию и тем более алгоритмической декомпозиции, как того требует структурное программирование.  Невозможно полностью отказаться от меток и операторов безусловного перехода при создании сложных ПП | if (p1)  {  f1;  goto L3;  }  L1:  if (p2)  {  L2:  f2;  L3:  f3;  goto L1;  }  else if (p3)  {  f4;  goto L2;  }  f5; |
| 4 | Процедурное программирование | Тип программирования, в котором инструкции для решения задачи выполняются одна за другой, сверху вниз, иногда возникают изменения в их последовательности | C, Fortran | Исходный код переносим, ​​поэтому его можно использовать и для другого процессора  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, когда используется процедурное программирование  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 | int sum = 0;  for (int v : vec)  sum += v;  std::cout << sum << std::endl; |
| 5 | Модульное программирование | Организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам | IBM S/360 Assembler, Кобол, RPG | Модульные программы легко составлять и отлаживать. Функциональные компоненты такой программы могут быть написаны и отлажены порознь.  Модульную программу легче сопровождать и модифицировать. Функциональные компоненты могут быть изменены, переписаны или заменены без изменений в остальных частях. | Модульность требует большей дополнительной работы. Чтобы писать модульные программы, программист должен быть значительно более аккуратным на этапе проектирования программной разработки. Он должен проектировать свои программы по нисходящей схеме, начиная с верхних уровней всей программы и затем продвигаясь вниз к более детальному проектированию отдельных подпрограмм. | **//main.c** #include <stdlib.h> #include "hello.h" int main() {     hello\_world();     return EXIT\_SUCCESS; } |
| 6 | Объектно-  ориентированное  программирование | Основанная на представлении программы в виде совокупности взаимодействующих [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), каждый из которых является экземпляром определённого [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), а классы образуют иерархию наследования | * Java. * Python. * C++. * Ruby. | Классы позволяют проводить конструирование из полезных компонент, обладающих простыми инструментами, что дает возможность абстрагироваться от деталей реализации.  Данные и операции вместе образуют определенную сущность и они не «размазываются» по всей программе, как это нередко бывает в случае процедурного программирования.  Локализация кода и данных улучшает наглядность и удобство сопровождения программного обеспечения. | Необходимо понимать базовые концепции, такие как классы, наследование и динамическое связывание. Для программистов, уже знакомых с понятием модуля и с абстрактными типами данных, это потребует минимальных усилий. Для тех же, кто никогда не использовал инкапсуляцию данных, это может означать изменения мировоззрения и может отнять на изучение значительное количество времени. | class Accumulator  {  public:  void feed(int value) { fSum += value; }  int sum() const { return fSum; }  private:  int fSum = 0;  }  Accumulator acc;  for (int v : vec)  acc.feed(vec);  std::cout << acc.sum() << std::endl; |
| 7 | Функциональное программирование | [Парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в которой процесс [вычисления](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) трактуется как вычисление значений [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) в математическом понимании последних (в отличие от [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) как подпрограмм в [процедурном программировании](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). | Лисп,  Erlang | Высокоуровневые абстракции, которые скрывают большое количество подробностей таких рутинных операций, как, например, итерирование. За счет этого код получается короче, и, как следствие, гарантирует меньшее количество ошибок, которые могут быть допущены. | На чистых функциональных языках не существует эффективного неупорядоченного словаря и множества  Не существует чисто функциональных слабых хэш-таблиц (weak hash table) ...  Не существует чисто функциональных потокобезопасных коллекций | int sum = std::accumulate(vec.begin(), vec.end(), 0);  std::cout << sum << std::endl; |
| 8 | Логическое программирование | [Парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), основанная на [математической логике](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0) — программы в ней задаются в форме логических утверждений и правил вывода. | Datalog ,Mercury , Oz | Операции, совершаемые в логическом программировании всегда понятны;  Результат практически всегда не зависит от выбранного пути реализации;  Может быть использован в качестве невычислительного языка используя только выражения и факты. | 1. Если брать за пример логического языка программирования Prolog, то на лицо невозможность создания комплексных задач. То есть в реальности логический язык может идти дополнением к процедурному, но самостоятельно используется крайне редко;   Из-за недостатка в инвестициях и простом внимании, логические языки слабо развиваются; | ?- factorial(1,F).  F = 1 .  ?- factorial(2,F).  F = 2 .  ?- factorial(3,F).  F = 6 .  ?- factorial(4,F).  F = 24 .  ?- factorial(5,F).  F = 120 .  ?- factorial(10,F).  F = 3628800 . |

1. Что такое **стандарт оформления программного кода.** Обоснуйте необходимость **качественного формирования листинга программы**. Перечислите правила формирования листинга программы.

**Стандарт оформления кода** (стандарт кодирования, стиль программирования) (англ. coding standards, coding convention или programming style) — набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования.

Несмотря на то, что программа исполняется машиной, программный код пишется людьми и для людей — неслучайно высокоуровневые языки программирования имеют человекопонятные синтаксис и команды. Современные программные проекты разрабатываются группами программистов, порой разделённых не только офисным пространством, но и материками и океанами. Благо, уровень развития технологий позволяет использовать навыки лучших разработчиков, вне зависимости от места нахождения их работодателей. Такой подход к разработке предъявляет серьёзные требования к качеству кода, в частности, к его читабельности и понятности.

**Программный код должен быть сопровожден комментариями**. Рекомендуется использовать возможности **самодокументирования** кода. В основной части работы для иллюстрации излагаемого теоретического материала **должны приводиться листинги** фрагментов программ, которые следует **располагать** непосредственно **после текста**, в котором они впервые упоминаются. Название листинга печатается тем же шрифтом, что и основной текст, и размещается над листингом слева, без абзацного отступа через тире после номера листинга.

При оформлении листингов следует использовать **шрифт** **Courier New, размер – 12 пт., межстрочный интервал – одинарный.** Рекомендуется **отделять** смысловые блоки **пустыми строками**, а также визуально **обозначать** вложенные конструкции с помощью **отступов**. Листинги должны **иметь порядковую нумерацию** в пределах каждого раздела. Номер листинга должен состоять из **номера раздела и порядкового номера листинга, разделенных точкой**

1. Дайте объяснение следующим **правилам оформления кода программы**:

* **Венгерская нотация**

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

* **Верблюжья нотация**

**Верблюжья нотация —** стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово внутри фразы пишется с прописной буквы. Стиль получил название CamelCase, поскольку прописные буквы внутри слова напоминают горбы верблюда.

* **Грамотный выбор идентификаторов** (правила именования переменных, констант, структур в программе, соглашения о длине имен идентификаторов).
* **Отступы в программном коде** (горизонтальные, вертикальные отступы, отступы с помощью пробелов)

**Стиль отступов (индентация)** — правила форматирования исходного кода, в соответствии с которыми отступы программных блоков проставляются в удобочитаемой манере. Используемый стиль отступов обычно особо оговаривается в стандарте оформления кода. Редакторы текста, входящие в состав большинства популярных сред разработки, часто предоставляют средства для поддержки используемого стиля отступов, например, автоматическую вставку пробелов/табуляции при вводе скобок, обозначающих начало/конец логического блока. Большинство специалистов сходятся во мнении, что комментарии должны объяснять намерения программиста, а не код; то, что можно выразить на языке программирования, не должно выноситься в комментарии — в частности, надо использовать говорящие названия переменных, функций, классов, методов и других сущностей (см. Соглашения об именах), разбивать программу на лёгкие для понимания части, стремиться к тому, чтобы структура классов и структура баз данных были максимально понятными и прозрачными и т. д. Есть даже мнение (его придерживаются в экстремальном программировании и некоторых других гибких методологиях программирования), что если для понимания программы требуются комментарии — значит, она плохо написана.

* **Использование комментариев**

Использование комментариев пояснения к [исходному тексту программы](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), находящиеся непосредственно внутри комментируемого кода. [Синтаксис](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) комментариев определяется [языком программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). С точки зрения [компилятора](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) или [интерпретатора](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80), комментарии — часть текста программы, не влияющая на её семантику.

* **Разбиение задачи на подзадачи**

**Разбиение задачи на подзадачи - декомпозицией в тайм-менеджменте называют разделение целей или задач на отдельные небольшие шаги — подзадачи**. Например, задача «навести порядок на кухне» делится на следующие этапы: убрать со стола, помыть посуду, протереть газовую плиту, подмести. А чтобы достичь цели «улучшить физическую форму» нужно заняться физкультурой, сесть на диету и отказаться от вредных привычек.

* **Правила оформления процедур, функций, модулей, классов.**

**Описание**[**процедур и функций**](https://its.1c.ru/db/v8std/content/647/hdoc) рекомендуется выполнять в виде комментария к ним. Необходимость комментирования отдельных участков кода процедур и функций должна определяться разработчиком исходя из сложности и нестандартности конкретного участка кода.

**Описание модуля** (общие модули, модули объектов, модули менеджеров объектов, модули форм, команд и т.п.) в общем случае могут присутствовать следующие разделы в приведенной ниже последовательности:

* заголовок модуля
* раздел описания переменных
* экспортные процедуры и функции модуля, составляющие его программный интерфейс
* обработчики событий объекта (формы)
* служебные процедуры и функции модуля
* раздел инициализации

**Названия классов пишите латиницей и по-английски.** Если с английским языком туго, помогут онлайн-переводчики — они вполне справляются с задачей. Это правило помогает повысить поддерживаемость кода и избежать проблем с кодировками. **Используйте нижний регистр.** Также в названиях классов принято использовать только строчные буквы. А вместо пробелов, как правило, — дефис (символ “-”). **Пользуйтесь популярными тегами.**Среди разработчиков уже сложился [список общепринятых имён для классов](https://github.com/yoksel/common-words). Например, для картинок используют img (сокращение от англ. image – картинка), а для кнопок — button (в переводе с английского «кнопка»).

Во время вёрстки теги могут меняться, а одинаковые стили могут не подходить по назначенным тегам. Например, на одной странице параметры для абзаца будут совпадать с параметрами для подзаголовка на другой. **Старайтесь избегать нумерации**, потому что потом будет сложно сориентироваться в коде и понять разницу. Имена можно давать исходя из отличительных параметров или из блоков, где используется кнопка.