***Лабораторная работа №15***

**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.

**Порядок выполнения работы:**

1. Выполните задания 1-3.

**Задание 1. Критерии выбора языка программирования.**

1. **Изучить критерии выбора языка программирования.**

На начальном этапе создания программы так или иначе становится вопрос выбора языка программирования. Кто-то выбирает язык только из личных предпочтений, кто-то только потому, что знает только этот язык.

На данный момент существует огромное количество языков на любой, вкус и цвет. Выбор языка программирования во время создания программы является очень важным моментом, от которого зависит очень и очень многое – скорость создания программы, скорость тестирования, возможность переноса на другие платформы, возможность быстрого внесения изменений, быстрота выполнения конечного продукта и так далее.

Пункты, которые я считаю важными, могут совпадать с несколькими вещами, которые уже были сказаны, но я полагаю, что большинство в конечном итоге будет подмножеством одного из следующих четырех:

* Простота : студенты обычно обучаются программированию, алгоритмам и структурам данных, а не особенностям и тонкостям какого-либо конкретного языка программирования. Язык программирования, как правило, просто инструмент, а не цель.
* Однозначность : близкий родственник простоты, для каждой концепции должна быть одна конструкция с минимальным перекрытием.
* Переносимость : пользователь может входить на сайт с компьютерами Linux, Windows и OSX, и среда программирования должна быть максимально похожа (идентична) для всех трех. Результаты любой программы также должны быть идентичными.

Так же при выборе ЯП можно учитывать такие критерии как:

1. **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.
2. **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.
3. **Скорость разработки программы.**  
   Данное требование возникает тогда, когда начальник говорит «программа нужна не позже, чем вчера!» или еще какая срочность. Тогда выбор падает на высокоуровневые языки с максимально человеколюбивым синтаксисом. Это, например, *Java, Flash* и подобные. На данных языках время разработки может существенно сокращаться из-за обилия сторонних библиотек, максимально «очеловеченного» синтаксиса, и подобных вещей. Скорость выполнения программ, написанных на данных языках страдает, причем порой весьма ощутимо.
4. **Ориентированность на компьютер или человека**

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. Однако, если ГИП не очень сложный, но красивый, возможно использование *Java* и *Flash*, на которых создание красивых интерфейс гораздо проще, нежели на С++ и, тем более, С. Если программа ориентирована в первую очередь на «скрытую работу» с минимумом взаимодействия с пользователем, тогда выбор должен ложиться в сторону быстрых языков (*ASM, C*)

1. **Кроссплатформенность.**  
   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.
2. **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.
3. **Обосновать выбор языка программирования для решения поставленной задачи в соответствии с  индивидуальным заданием.**

Для написания программы была выбрана среда программирования Visual Studio C#, основанная на языке программирования C#. Данная среда выгодно отличается эффективностью и надежностью.

Для решения поставленной задачи необходимо использовать функциональную, эффективную и удобную платформу для разработки, позволяющую применять принципы объектно-ориентированного программирования. В качестве такой платформы была выбрана среда .NET.

Среда разработки Visual Studio, поставляемая вместе с .NET, предоставляет необходимый инструментарий для эффективного и быстрого создания приложений с графическим интерфейсом.

Язык программирования C# претендует на подлинную объектную ориентированность.

Язык программирования C# призван реализовать компонентно-ориентированный подход к программированию, который способствует меньшей машинно-архитектурной зависимости результирующего программного кода, большей гибкости, переносимости и легкости повторного использования программ.

Принципиально важным отличием от предшественников является изначальная ориентация на безопасность кода.

Расширенная поддержка событийно-ориентированного программирования.

**Задание 2. Изучение парадигм программирования**

* 1. **Изучить различные парадигмы программирования.**
* Паради́гма программи́рования — 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[[1]](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#cite_note-_12a0a1f0bdcfbf26-1).
* Парадигма программирования не определяется однозначно языком программирования; практически все современные языки программирования в той или иной мере допускают использование различных парадигм ([мультипарадигмальное программирование](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0%D0%BB%D1%8C%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" \o "Мультипарадигмальное программирование)).
* Также существующие парадигмы зачастую пересекаются друг с другом в деталях, поэтому можно встретить ситуации, когда разные авторы употребляют названия из разных парадигм, говоря при этом, по сути, об одном и том же явлении.

***2.* Провести сравнительную характеристику различных парадигм программирования. Результаты  оформить в виде таблицы следующего вида:**

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. | [Fortran](https://ru.wikipedia.org/wiki/Fortran), [Algol](https://ru.wikipedia.org/wiki/Algol), [COBOL](https://ru.wikipedia.org/wiki/COBOL) и [Basic](https://ru.wikipedia.org/wiki/Basic) | использование именованных [переменных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%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));  использование оператора [присваивания](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D1%81%D0%B2%D0%B0%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5); использование составных выражений;  использование [подпрограмм](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0); | Императивный язык должен описывать не столько саму задачу сколько её решение. Некоторыми авторами считается, что данное определение скорее относится к «процедурной» парадигме, которая, помимо императивного, включает в себя функциональное программирование. |  |
| **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, что представляют собой проблема и ожидаемый результат. |  | К подвидам декларативного программирования также зачастую относят [функциональное](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%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) и [логическое программирование](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%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). Несмотря на то, что программы на таких языках нередко содержат алгоритмические составляющие, архитектура в императивном понимании (как нечто отдельное от кодирования) в них также отсутствует: схема программы является непосредственно частью исполняемого кода[[1]](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%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#cite_note-1). | декларативные программы не используют понятия состояния, в частности, не содержат [переменных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%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)) и операторов [присваивания](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D1%81%D0%B2%D0%B0%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), обеспечивается [ссылочная прозрачность](https://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B7%D1%80%D0%B0%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C). |  |
| **3.** | **Структурное**  **программирование** | В соответствии с парадигмой, любая программа, которая строится без использования оператора goto, состоит из трёх базовых управляющих конструкций: последовательность, [ветвление](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%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)), [цикл](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB_(%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)); кроме того, используются [подпрограммы](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). При этом разработка программы ведётся пошагово, методом «сверху вниз». | Ада,  Goto  delphi | Методология структурной разработки программного обеспечения была признана «самой сильной формализацией 70-х годов». | Программы становились слишком сложными, чтобы их можно было нормально сопровождать. Поэтому потребовалась систематизация процесса разработки и структуры программ. |  |
| **4.** | **Процедурное**  **программирование** | программирование на [императивном языке](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%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), при котором последовательно выполняемые [операторы](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)) можно собрать в [подпрограммы](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), то есть более крупные целостные единицы [кода](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), с помощью механизмов самого языка | 1С  [Ada](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_(%D1%8F%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))  [Алгол 60](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D0%BB_60)  [Basic](https://ru.wikipedia.org/wiki/Basic)  [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%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)) | Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов. | Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. |  |
| **5.** | **Модульное**  **программирование** | это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам | [Assembler](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0), [Кобол](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B1%D0%BE%D0%BB), [RPG](https://ru.wikipedia.org/wiki/RPG_(%D1%8F%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)), [ПЛ/1](https://ru.wikipedia.org/wiki/%D0%9F%D0%9B/1), [Ада](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_(%D1%8F%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)) | Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.  Модульное программирование может быть осуществлено, даже когда синтаксис языка программирования не поддерживает явное задание имён модулям. |  |  |
| **6.** | **Объектно**  **ориентированное**  **программирование** | [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 [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 наследованя. | , [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Java](https://ru.wikipedia.org/wiki/Java) , [Smalltalk](https://ru.wikipedia.org/wiki/Smalltalk), [Eiffel](https://ru.wikipedia.org/wiki/Eiffel), [OCaml](https://ru.wikipedia.org/wiki/OCaml) | Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную [нормализации](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0)) и их целостность, поэтому созданное удобно управляемым — будет и удобно пониматься. Таким образом, через тактическую задачу управляемости решается стратегическая задача — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму. | То есть фактически речь идёт о прогрессирующей организации информации согласно первичным семантическим критериям: «важное/неважное», «ключевое/подробности», «родительское/дочернее», «единое/множественное». |  |
| **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](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%81%D0%BF)   [APL](https://ru.wikipedia.org/wiki/APL_(%D1%8F%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)) —   [ML](https://ru.wikipedia.org/wiki/ML)   [F#](https://ru.wikipedia.org/wiki/F_Sharp)   [Miranda](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D1%80%D0%B0%D0%BD%D0%B4%D0%B0_(%D1%8F%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)) [Nemerle](https://ru.wikipedia.org/wiki/Nemerle) | Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния (в отличие от [императивного](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%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), где одной из базовых концепций является [переменная](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%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)), хранящая своё значение и позволяющая менять его по мере выполнения [алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)). | На практике отличие математической функции от понятия «функции» в императивном программировании заключается в том, что императивные функции могут опираться не только на аргументы, но и на состояние внешних по отношению к функции переменных, а также иметь [побочные эффекты](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B1%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%8D%D1%84%D1%84%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)) и менять состояние внешних переменных. |  |
| **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 автоматическом доказательстве теорем, а также раздел [дискретной математики](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), изучающий принципы логического вывода информации на основе заданных фактов и правил вывода. Логическое программирование основано на теории и аппарате [математической логики](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) с использованием математических принципов резолюций. | [Planner](https://ru.wikipedia.org/wiki/Planner), [QA-4](https://ru.wikipedia.org/w/index.php?title=QA-4&action=edit&redlink=1), [Popler](https://ru.wikipedia.org/w/index.php?title=Popler&action=edit&redlink=1), [Conniver](https://ru.wikipedia.org/w/index.php?title=Conniver&action=edit&redlink=1) и [QLISP](https://ru.wikipedia.org/w/index.php?title=QLISP_(%D1%8F%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)&action=edit&redlink=1). | Первым языком логического программирования был язык [Planner](https://ru.wikipedia.org/wiki/Planner)[[1]](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%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#cite_note-1), в котором была заложена возможность автоматического вывода результата из данных и заданных правил перебора вариантов (совокупность которых называлась планом). Planner использовался для того, чтобы понизить требования к вычислительным ресурсам (с помощью бэктрекинга — [поиска с возвратом](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D1%81_%D0%B2%D0%BE%D0%B7%D0%B2%D1%80%D0%B0%D1%82%D0%BE%D0%BC)) и обеспечить возможность вывода фактов, без активного использования [стека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA). Затем был разработан язык [Prolog](https://ru.wikipedia.org/wiki/Prolog), который не требовал плана перебора вариантов и был, в этом смысле, упрощением языка [Planner](https://ru.wikipedia.org/wiki/Planner). |  |  |

Примеры программ:

1. **Императивная**

function double (arr) {

let results = []

for (let i = 0; i < arr.length; i++){

results.push(arr[i] \* 2)

}

return results

}

1. **Декларативное**

class Button extends React.Component{this.state = { color: 'red' }handleChange = () => {  
 const color = this.state.color === 'red' ? 'blue' : 'red';  
 this.setState({ color });  
 }render() {  
 return (<div>  
 <button   
 className=`btn ${this.state.color}`  
 onClick={this.handleChange}>  
 </button>  
 </div>);  
 }  
}

**3) Структурное**

if p1:

f1

f3

l = 3

while l > 0:

if p2:

f2

f3

elif p3:

f4

f2

f3

else:

f5

l = 0;

**4) Процедурное**

int sum = 0;

for (int v : vec)

sum += v;

std::cout << sum << std::endl;

**5) модульное**

**//main.c**  
#include <stdlib.h>  
#include "hello.h"  
int main()  
{  
    hello\_world();  
    return EXIT\_SUCCESS;  
}

**6) Функциональное**

int sum = std::accumulate(vec.begin(), vec.end(), 0);

std::cout << sum << std::endl;

**7) Объектно-ориентированние**

**class** Data:

**def** \_\_init\_\_(self, \*info):

self.info = list(info)

**def** \_\_getitem\_\_(self, i):

**return** self.info[i]

1. Логические

?- study(mark, X).

X = book ;

X = studentbook ;

X = docs.

1. **Сделать выводы о том, какие из парадигм программирования используются наиболее часто.  Самым распространённым парадигмами программирования являются:**

## Процедурное программирование

Этот подход — разновидность императивной парадигмы программирования. Процедурами здесь называют команды, которые применяются в определённом порядке и последовательно меняют состояние памяти. После применения всех команд программа выдаёт результат.

**Метапрограммирование**

Здесь программа, которую вы создаёте, сама генерирует код новой программы или модифицирует свой. С помощью этого подхода часть задач разработчика можно автоматизировать. А ещё он даёт возможность людям, не владеющим языками программирования, создавать программы с помощью графических интерфейсов или словесных команд на естественном языке — программа преобразует их в обычный код.

## Обобщённое программирование

В этой парадигме программист создаёт обобщённые представления для классов и функций. Изначально у них отсутствуют требования типа данных для входных параметров, поэтому шаблоны можно сделать более универсальными.

Преимущество этой парадигмы в том, что можно создавать алгоритмы, которые будут работать с разными типами, и для этого не придётся добавлять реализации для каждого типа отдельно. Такой подход можно совместить как с ООП, так и с другими современными парадигмами программирования.

## Логическое программирование

Логическое программирование — это подвид декларативного. Основан на выводе информации из заданных фактов и логических правил, которые к ним можно применить. При выполнении программ используются правила формальной логики.

1. **Для своего индивидуального задания привести обоснование того какие парадигмы  программирования вы будете использовать.**

Для индивидуального задания я буду использовать Объектно-ориентированную парадигму. [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 [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 наследованя.

**Задание 3. Изучение правил формирования листинга программ.**

**2. Оформить листинги программ в соответствии с приведенными рекомендациями по оформлению**

**программ.**

* 1. Структурное программирование (в программе должны использоваться процедуры, функции). (Pascal)

**var** chislo: integer;

**procedure** calcevensum(chislo: integer);

**begin**

**var** summa := 0;

**while** chislo > 0 **do begin**

**if** chislo **mod** 10 **mod** 2 = 0 **then**

summa := summa + chislo **mod** 10;

chislo := chislo **div** 10;

] **end**;

writeln ('Суммма чётных цифр =', summa );

**end**;

**begin**

writeln ('Ведите 10 любых чисел');

**for var** i:=1 **to** 10 **do begin**

readln (chislo);

calcevensum(chislo);

**end**;

**end**.

2)

**const** a = 1.2 ;b = -3.2;

**var** z, x, y:real;

**function** f(u, t:real):real;

**begin**

**if** u>0 **then** f:=(u+2\*t);

**if** (u>-1) **and** (u>0) **then** f:=(sqr (u)-2\*t+1);

**if** u<=-1 **then** f:=(u+t);

**end**;

**begin**

x:= - 4.21;y :=31.2;

writeln ('x=',x,'y=',y);

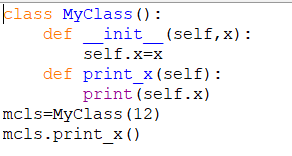
z:= f (sqrt (x), y)

+ f ( a , b ) + f ( x + 1, - y ) + f ( x - y , x ) + f ( x + y , a + b );

writeln ('z:=',z:0:3);

**end**.

* 1. Пример модульного программирования (Python)? (привести пример оформленного модуля).



* 1. Объектно-ориентированное программирование (привести пример оформленного класса). (С#)

/Класс для чтения/записи INI-файлов

public class INIManager

{

//Конструктор, принимающий путь к INI-файлу//

public INIManager(string aPath)

{

path = aPath;

}

//Конструктор без аргументов (путь к INI-файлу нужно будет задать отдельно)

public INIManager() : this("") { }

//Возвращает значение из INI-файла (по указанным секции и ключу)

public string GetPrivateString(string aSection, string aKey)

{