**Семинар 1. Схемы алгоритмов программ**

**Цель изучения дисциплины Алгоритмизация и программирование**– освоение теоретических сведений и практических приемов, знание которых необходимо для разработки простейшего программного обеспечения с использованием структурного и объектного подходов на универсальных языках программирования.

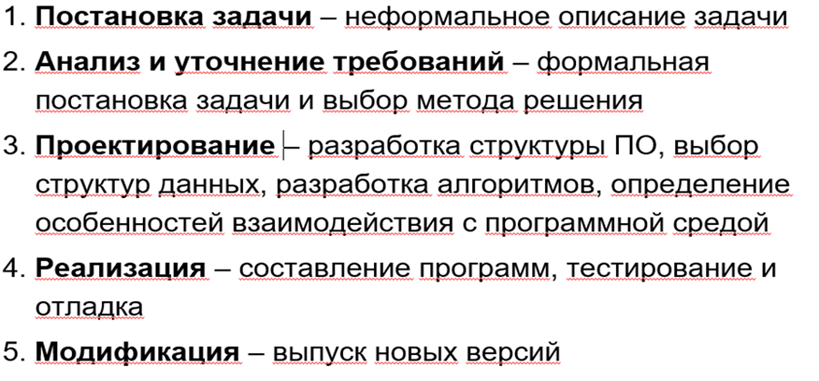
**Программирование** – важнейшая часть профессиональной деятельности специалиста в области информатики и вычислительной техники. Без знания основ программирования невозможно работать в указанной области.

Обучение основам программирования – **многоаспектная задача**, при решении которой невозможно отделить разработку последовательности выполняемых для решения задачи операций от изучения средств, предоставленных в распоряжении программиста тем или иным языком программирования и имеющимися библиотеками. Язык программирования и среда разработки непосредственно определяют набор проектных операций, в которых записываются разрабатываемые алгоритмы, а **реализованные в языке подходы диктуют набор технологий**, которые может использовать программист.

При разработке программы программисту одновременно приходится мыслить на макро- и микроуровнях: на макроуровне – определять последовательности выполняемых действий, а на микроуровне – оценивать и выбирать используемые технологии, механизмы и средства реализации этих действий.

**Основы алгоритмизации и процедурное программирование.**

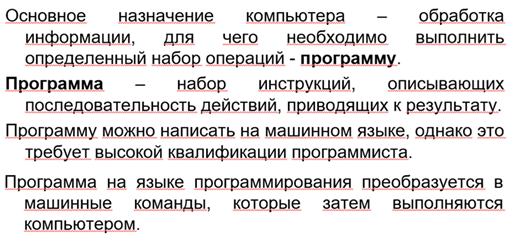
**Основные этапы создания ПО**



На первый план выходит проектирование решения, поскольку именно от разработанного проекта зависит, насколько правильно и эффективно с точки зрения используемого на конкретном этапе обучения набора средств будет решена задача.

Именно поэтому, основная цель семинарских занятий – обучение основам алгоритмизации, принципам структурной и объектной декомпозиции, т.е. разработке проектов программ.

**Программы и алгоритмы**

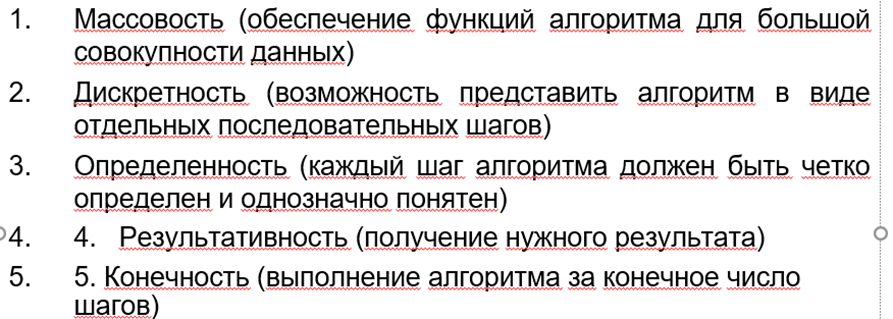


**Алгоритм** – это явно заданная, однозначная, механически исполняемая последовательность элементарных инструкций, обычно предназначенных для достижения конкретной цели.

**Алгоритм** – это абстрактные автоматически выполняемые процедуры, которые можно реализовать на любом языке программирования, поддерживающем элементарные низкоуровневые операции

**Программа** – точное представление алгоритма, но алгоритм – это не программа.

**Свойства алгоритмов**



**Способы представления алгоритма:**

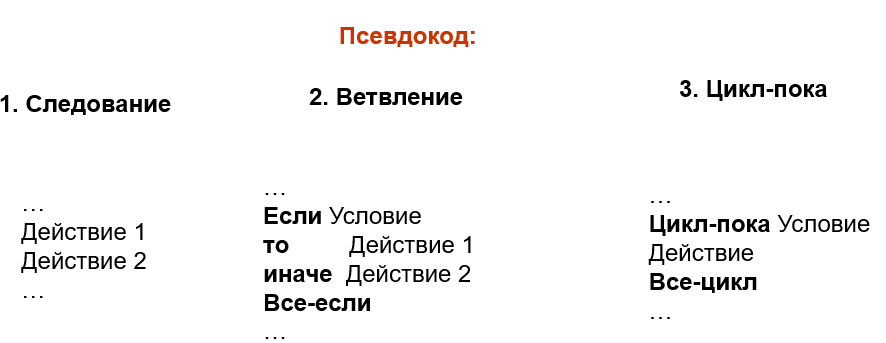
1. Описательная форма (на естественном языке)

2. Псевдокод (описательная форма с ограниченным числом элементов)

3. Графическая форма ( схема алгоритма)

4. Табличная форма (таблицы решений)

**Основные конструкции псевдокода**



**Обозначения схем алгоритма**

|  |  |  |
| --- | --- | --- |
| Название блока | Обозначение | Назначение блока |
| 1. Терминатор |  | Начало, завершение программы или подпрограммы |
| 2. Процесс |  | Обработка данных (вычисления, пересылки и т.п.) |
| 3. Данные |  | Операции ввода-вывода |
| 4. Решение |  | Ветвления, выбор, итерационные и поисковые циклы |
| 5. Подготовка |  | Заголовки счетных циклов |
| 6. Границы цикла |  | Начало и конец любого цикла |
| 7. Предопределенный процесс |  | Вызов процедур |
| 8. Соединитель |  | Маркировка разрывов линий |
| 9. Комментарий |  | Пояснения к операциям |

**Таблицы решений**

В столбик выписываются все условия, от которых зависят дальнейшие вычисления, а по горизонтали - все случаи для вычислений.

На пересечении каждого столбца и строки ставят букву **Y**, если для данного решения данное условие должно выполняться, букву **N**, если данное условие обязательно должно не выполняться, и **прочерк**, если исход сравнения не важен.

**Основные алгоритмические структуры**. Показывают, как происходит процесс вычислений.

-линейная

-разветвлённая ( ветвление)

-циклическая

В теории программирования доказано, что для записи любого, сколь угодно сложного алгоритма достаточно трех **базовых управляющих конструкций:**

• следование – обозначает последовательное выполнение действий;

• ветвление – соответствует выбору одного из двух вариантов действий;

• цикл-пока – определяет повторение действий, пока не будет нарушено некоторое условие, выполнение которого проверяется в начале цикла.

****

Помимо базовых структур используют еще три **дополнительные конструкции**, производные от базовых:

* ***выбор*** – обозначающий выбор одного варианта из нескольких в зависимости от значения некоторой величины;
* ***цикл-до*** – отображающий повторение некоторых действий до выполнения заданного условия, проверка которого осуществляется после выполнения действий в цикле;
* ***цикл с заданным числом повторений*** (***счетный цикл***) – указывающий на повторение некоторых действий указанное количество раз



Перечисленные конструкции были положены в основу ***структурного* программирования**

**Управляющие операторы языка** реализуют управляющие конструкции.

Управляющими называют операторы, способные изменять линейность процесса вычислений. К таким операторам относятся оператор условной передачи управления **if,** оператор выбора **switch**, операторы циклов **while, do, for** и операторы безусловной передачи управления **goto, break, continue, exit**

**Задания семинара**

**Задача 1**. Определить площадь треугольника по трем сторонам.

**Задача 2**. Усложнить задачу 1, задав требование проверки существования треугольника и определение его вида треугольника, если он существует (равносторонний, равнобедренный, прямоугольный).

**Задача 3**. Рассмотреть, как построить цикл до ввода длин сторон, удовлетворяющих условию существования треугольника.

**Задача 4.** Треугольник и точка заданы координатами на плоскости. Определить, находится ли точка внутри треугольника.

**Решение задачи 1**



**Решение задачи 2**



**Решение задачи 3**

Попытка построить цикл с учетом необходимости запроса на повторный ввод, если данные были введены неверно, приводит к запутанному алгоритму, поэтому целесообразно выделить подпрограмму проверки правильности ввода с результатом булевского типа («Верно», «Неверно»). Этот результат и будем использовать в цикле ввода в случае неправильного ввода значений. Запрос на повторный ввод можно совместить с запросом на повторное решение задачи в конце алгоритма, тогда алгоритм получится структурным.

Схема алгоритма к задаче 3

Данная схема не учитывает вероятность зацикливания при повторном вводе данных.

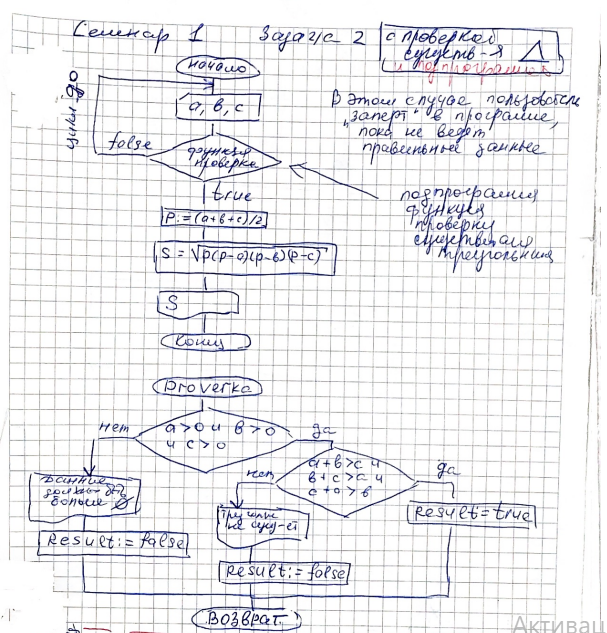
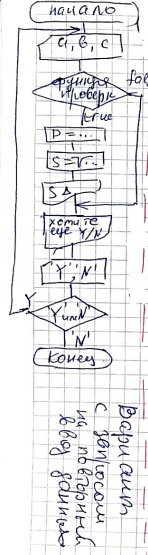


Схема алгоритма к задаче 3 с учётом исправления ошибки зацикливания при повторном вводе данных



**Решение задачи 4**

Одни из вариантов решения.

Идея решения: сравнить суммы площадей треугольников, построенных на заданной точке D и сторонах имеющегося треугольника: если точка внутри (D′), то сумма трех площадей равна площади треугольника, иначе (D′′) – больше, чем последняя.



Детальной проработки схемы не требуется, поэтому, для уточнения последовательности действий решения задачи вполне достаточно обобщенной схемы, в которой текстом описывают необходимые укрупненные операции. При этом используется блок Предопределенный процесс, который применяют для описания вызова подпрограмм.

В случае, когда данные могут быть заданы вещественными числами, необходимо учитывать погрешность вычисления и правила сравнения вещественных числе в которые в памяти ЭВМ представляются неточно.

Обобщённая схема алгоритма к решению задачи 4

