**Дисциплина «Информатика» для студентов 1 курса**

**Преподаватель Шитова Анастасия Александровна.**

**Лекция 1 «Алгоритмы, величины, структура алгоритмов»**

Задание: прочесть лекцию, посмотреть презентацию, сделать краткий конспект лекции в тетради, на паре мы будем выполнять тест по данной теме. Тестирование на время. Знать содержание лекции, уметь ответить на поставленные вопросы.

  *На этом занятии закрепляется ранее изученный материал. Речь пойдет о том, что такое алгоритм, какие бывают данные и величины; рассматриваются разновидности типов данных и их основные отличия; изучается структура алгоритмов, а также базовые способы их записи, такие как следование, ветвление и цикл.*

На этом занятии мы с вами закрепим ранее изученный вами материал. Вспомним, что такое алгоритм, какие бывают величины, а также разберёмся со структурой алгоритмов.

Начнём мы с вами с этапов решения задач. Всего их шесть:

1.     Постановка задачи.

2.     Формализация задачи.

3.     Построение алгоритма.

4.     Составление программы на языке программирования.

5.     Отладка и тестирование программы.

6.     Проведение расчётов и анализ полученных результатов.

Но к программированию из этого списка относятся только пункты 3-5.

Давайте разберёмся, что происходит на каждом этапе решения задачи.

Итак, на первом этапе – *постановка задачи*, нужно понять условие задачи, выделить исходные данные и данные, которые требуется получить в результате, и понять отношения между ними.

Второй этап – *формализация задачи*. На этом этапе задача чаще всего переводится на математический язык. То есть строится математическая модель.



Далее идёт этап *построения алгоритма*. На данном этапе нужно записать решение задачи в виде, например, блок-схемы.



На четвёртом этапе идёт уже работа с компьютером. То есть здесь мы будем *записывать задачу на языке программирования*, в нашем случае на языке Паскаль.



На этапе *тестирования и отладки проверяется*, работает ли программа. И если работает, то правильно ли. Также выявляется наличие либо отсутствие ошибок в программе.



И на последнем этапе проводятся *расчёты и анализ полученных результатов*. То есть используется разработанная программа в практических целях.

Но, при решении задач по программированию на занятиях, учащиеся обычно заканчивают работу на 5 этапе.

Исходя из этапов решения задач, можно сделать *вывод*, что программист должен уметь строить алгоритмы, знать языки программирования, а также уметь работать в соответствующей системе программирования. В основе всей работы лежит развитое алгоритмическое мышление.

А сейчас обратимся немного к истории появления такого понятия, как «алгоритм».

Происхождение термина «алгоритм» связано с математикой. История его возникновения такова.

В IX веке в Багдаде жил учёный Абу Аль-Хорезми, математик, астроном, географ.



В одном из своих трудов он описал десятичную систему счисления и впервые сформулировал правила выполнения арифметических действий над целыми числами и обыкновенными дробями.

Арабский оригинал этой книги был утерян, но остался латинский перевод XII века, по которому Западная Европа ознакомилась с десятичной системой счисления и правилами выполнения арифметических действий.

Первыми алгоритмами в математике являются сложение, вычитание, умножение «столбиком», деление «уголком» многозначных чисел.

Многие столетия разрабатывались алгоритмы для решения всё новых и новых классов задач, но само понятие алгоритма не имело точного математического определения.

В наше же время, **алгоритм** – это последовательность команд управления каким-либо исполнителем. С понятием алгоритма и методами его построения учащиеся знакомятся на примере таких исполнителей, как Робот, Черепашки, Чертёжник и другими.



Также существует графический исполнитель, с помощью которого можно создавать рисунки на экране, осуществлять перемещение в лабиринтах, перемещать предметы с места на место. Таких исполнителей принято называть **исполнителями, работающими в обстановке**.

При выполнении какой-либо программы исполнитель работает с **величинами** – различными информационными объектами. К таким объектам относятся числа, символы, коды и так далее. Таким образом, **алгоритмы работы с величинами –** это алгоритмы, предназначенные для управления компьютером.

А сейчас рассмотрим более подробно данные и величины.

**Данные** – это совокупность величин, с которыми работает компьютер.

Данные делятся на *исходные*, *результаты* и *промежуточные* данные. **Исходные данные** – это данные, которые заданы изначально, по условию. **Промежуточные данные** – это данные, которые получаются в процессе вычислений. **Результаты** – это данные, которые являются окончательным ответом на поставленную задачу. То есть мы вводим в программу исходные данные. Затем исполнитель (компьютер) выполняет действия, описанные в программе, в результате чего получаются промежуточные данные. А после выполнения всех команд программы, компьютер получает и выдаёт нам результаты.



Например, при нахождении площади треугольника через полупериметр, *исходными* *данными* будут являться стороны a, b, c. К *промежуточным* *данным* будет относится нахождение полупериметра. А *результатом* – площадь треугольника.

Необходимо помнить следующее **правило**: всякая величина занимает своё определённое место в памяти компьютера. Мы с вами знаем, что память компьютера можно представить в виде таблицы, которая состоит из ячеек. То есть всякая величина занимает определённую ячейку в памяти компьютера.



Любая величина имеет три основных *свойства*: имя, значение и тип. Процессор же различает величины по адресу ячеек памяти, в которых они хранятся.

Величины делятся на *константы* и *переменные*. **Константа** – это неизменная величина, которая представляется в алгоритме собственным значением. Её значение в течение всей программы не изменяется, то есть оно постоянно.

**Переменная** же может изменять своё значение. Например, в начале программы она была равна 3, а в конце – 10.

Любая константа или переменная занимают ячейку памяти, а значение этих величин определяется двоичным кодом в этой ячейке.



А сейчас перейдём к типам данных.

К *основным типам данных* относятся целый, вещественный, логический и символьный типы. Типы данных различаются между собой по трём *критериям*: множество допустимых значение, множество допустимых операций, форма внутреннего представления. Все эти свойства предоставлены в таблице на экране.



Типы констант определяются по форме записи в тексте.



А типы переменных задаются в разделе описания переменных.



Данные также можно классифицировать *по структуре*. Они делятся на простые и структурированные. Простые величины имеют одно значение, а вот структурированные – множество. К структурированным величинам относятся массивы, строки и так далее.

В нашем конкретном случае при написании программы исполнителем является *компьютер*. А если разбираться более подробно, то исполнителем является комплекс: компьютер плюс система программирования.



Таким образом программист будет составлять программу на том языке, на который ориентирована система программирования. В нашем случае, мы будем составлять программы на языке Pascal.

Тем не менее, независимо от того, на каком языке программирования будет написана программа, алгоритм решения любой задачи на компьютере может быть составлен из следующих команд:

·        присваивание;

·        ввод;

·        вывод;

·        обращение к вспомогательному алгоритму (подпрограмме);

·        цикл;

·        ветвление.

Для описания алгоритмов лучше всего использовать блок-схемы.

В 1969 году известный голландский учёный-программист Эдсгер Вибе Дейкстра *доказал следующее*: алгоритм для решения любой логической задачи можно составить только из структур «следование», «ветвление», «цикл». Такие структуры называются базовыми алгоритмическими структурами.



Методика программирования, основанная на этой теореме, называется **структурным программированием**.

Для описания структур алгоритмов можно использовать два способа*: блок-схемы* и *учебный Алгоритмический язык*.

Давайте ещё раз рассмотрим базовые алгоритмические структуры и способы их записи.

*Следование* – это линейная последовательность действий.



В данном случае, *серия* – это простой или составной оператор. **Составной оператор** – это серия, заключённая в операторные скобки. В языке Паскаль: begin…end.

Далее рассмотрим *ветвление*. **Ветвление** – это алгоритмическая альтернатива. Рассмотрим блок-схему полного ветвления.



Если заданное условие выполняется, то управление передаётся серии 1, а если нет – серии 2. Полное ветвление, записанное на алгоритмическом языке:



**Условие** – это утверждение, которое может быть истинным или ложным. Такое утверждение также называют **логическим выражением**.

Также для записи можно использовать неполную форму ветвления.



В ней, если условие выполняется, управление передаётся серии, а если нет, то программа переходит к выполнению остальных команд, следующих за ветвлением.

**Цикл** – это повторение некоторой группы действий по условию. Существуют два типа цикла: цикл с предусловием (цикл-пока), цикл с постусловием (цикл-до).



При *цикле с предусловием*, серия будет выполняться до тех пор, пока условие истинно. То есть сначала идёт проверка условия, а затем решается, будет выполняться серия или нет. При использовании данного цикла серия может ни разу не выполниться, так как условие может быть ложно.

А при *цикле с постусловием* сначала идёт серия, а затем условие. Серия будет выполняться до тех пор, пока условие ложно. В данном цикле серия выполнится хотя бы один раз, прежде чем программа приступит к проверке условия.

До этого мы с вами рассматривали базовые структуры. Сложный же алгоритм состоит из соединённых между собой базовых структур. Между собой эти структуры могут соединяться двумя способами: *последовательным* и *вложенным*.

Если блок, который составляет тело цикла сам по себе является циклической структурой, то это будет являться вложенным циклом.



Внутренний цикл также может иметь в себе ещё один цикл и так далее. Таким образом появляется представление о глубине вложенности циклов.



Таким же образом и ветвления могут быть вложенными друг в друга.



При изображении блок-схем вложенных циклов или ветвлений нужно соблюдать некоторые правила.

Давайте рассмотрим их на примере. На данной схеме «у» – условие, «с» – серия, «+» – да, «–» – нет.



Каждая базовая структура должна иметь один вход и один выход. Чертить их нужно при помощи таких же элементов, как и ранее, в обратном случае нестандартно изображённая блок-схема будет плохо читаться и будет теряться наглядность алгоритма.

Наша же блок-схема легко читается, а структура хорошо воспринимается визуально. У нас изображён цикл с вложенным ветвлением.

Рассмотрим ещё одну блок-схему.



На ней изображены вложенные циклы-пока. Глубина вложенности – 1.



На третьей схеме изображены вложенные ветвления. Глубина вложенности – 1.

При описании алгоритма на алгоритмическом языке нужно структурировать внешний вид текста. Для этого также существуют свои правила:

·  конструкции одного уровня вложенности записываются на одном вертикальном уровне (начинаются с одной позиции в строке);

· вложенная конструкция записывается смещённой по строке на несколько позиций вправо относительно внешней для неё конструкции.

Всё это вы можете увидеть при записи приведённых выше блок-схем на алгоритмическом языке.



Такой же способ используется и при структуризации текста в программе языка Паскаль.

**Структурное программирование –** это не только форма описания алгоритма и программы, но это ещё и способ мышления программиста. При размышлении над составлением алгоритма, необходимо представлять его при помощи стандартных структур. То есть складывать сложную программу из простых частей.

А сейчас подведём итоги занятия.

Существует шесть этапов решения задачи на компьютере.

*Исполнителем* *алгоритмов обработки данных* являются компьютер и система программирования.

Данные можно классифицировать по нескольким *критериям*: по отношению к алгоритму, по значениям, по типам и по структуре.

К *базовым алгоритмическим структурам* относятся следование, ветвление, цикл.

Соединять базовые структуры можно двумя способами: последовательным и вложенным.

При составлении блок-схем или записи алгоритма на алгоритмическом языке нужно соблюдать некоторые правила.