ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»
Кафедра «Математика и теоретическая механика»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Программирование и основы алгоритмизации

Направление подготовки **27.03.04 Управление в технических системах**

Профиль образовательной программы **Системы и средства автоматизации технологических процессов**

Форма обучения **очная**

СОДЕРЖАНИЕ

1. КОНСПЕКТ ЛЕКЦИЙ

1. 1 Лекция № 1 (2 часа).

Тема: «Основные понятия алгоритмизации»

1.1.1 Вопросы лекции:

1. Понятие алгоритма.
2. Свойства алгоритмов.
3. Формы записей алгоритмов. Блок-схемы.
4. Основные алгоритмические конструкции: линейные, разветвляющиеся, циклические.

1.1.2 Краткое содержание вопросов:

1. Понятие алгоритма.

Одним из фундаментальных понятий в информатике является понятие алгоритма. Происхождение самого термина «алгоритм» связано с математикой. Это слово происходит от Algorithmi – латинского написания имени Мухаммеда аль-Хорезми (787 – 850) выдающегося математика средневекового Востока. В своей книге "Об индийском счете" он сформулировал правила записи натуральных чисел с помощью арабских цифр и правила действий над ними столбиком. В дальнейшем алгоритмом стали называть точное предписание, определяющее последовательность действий, обеспечивающую получение требуемого результата из исходных данных.

Алгоритм может быть предназначен для выполнения его человеком или автоматическим устройством. Создание алгоритма, пусть даже самого простого, - процесс творческий. Он доступен исключительно живым существам, а долгое время считалось, что только человеку. В XII в. был выполнен латинский перевод его математического трактата, из которого европейцы узнали о десятичной позиционной системе счисления и правилах арифметики многозначных чисел. Именно эти правила в то время называли алгоритмами.

Данное выше определение алгоритма нельзя считать строгим – не вполне ясно, что такое «точное предписание» или «последовательность действий, обеспечивающая получение требуемого результата».

2. Свойства алгоритмов.

• ***Дискретность*** (прерывность, раздельность) – алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов. Каждое действие, предусмотренное алгоритмом, исполняется только после того, как закончилось исполнение предыдущего.

• ***Определенность*** – каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

• ***Результативность (конечность)*** – алгоритм должен приводить к решению задачи за конечное число шагов.

• ***Массовость*** – алгоритм решения задачи разрабатывается в общем виде, то есть, он должен быть применим для некоторого класса задач, различающихся только исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

На основании этих свойств иногда дается определение алгоритма, например: *“Алгоритм – это последовательность математических, логических или вместе взятых операций, отличающихся детерменированностью, массовостью, направленностью и приводящая к решению всех задач данного класса за конечное число шагов”*.

Такая трактовка понятия “алгоритм” является неполной и неточной.

Во-первых, неверно связывать алгоритм с решением какой-либо задачи. Алгоритм вообще может не решать никакой задачи.

Во-вторых, понятие “массовость” относится не к алгоритмам как к таковым, а к математическим методам в целом. Решение поставленных практикой задач математическими методами основано на абстрагировании – мы выделяем ряд существенных признаков, характерных для некоторого круга явлений, и строим на основании этих признаков математическую модель, отбрасывая несущественные признаки каждого конкретного явления. В этом смысле любая математическая модель обладает свойством массовости. Если в рамках построенной модели мы решаем задачу и решение представляем в виде алгоритма, то решение будет “массовым” благодаря природе математических методов, а не благодаря “массовости” алгоритма.

Требования, предъявляемые к алгоритмам:

Первое правило – при построении алгоритма прежде всего необходимо задать множество объектов, с которыми будет работать алгоритм. Формализованное (закодированное) представление этих объектов носит название данных. Алгоритм приступает к работе с некоторым набором данных, которые называются входными, и в результате своей работы выдает данные, которые называются выходными. Таким образом, алгоритм преобразует входные данные в выходные. Это правило позволяет сразу отделить алгоритмы от «методов» и «способов». Пока мы не имеем формализованных входных данных, мы не можем построить алгоритм.

Второе правило – для работы алгоритма требуется память. В памяти размещаются входные данные, с которыми алгоритм начинает работать, промежуточные данные и выходные данные, которые являются результатом работы алгоритма. Память является дискретной, т.е. состоящей из отдельных ячеек. Поименованная ячейка памяти носит название переменной. В теории алгоритмов размеры памяти не ограничиваются, т. е. считается, что мы можем предоставить алгоритму любой необходимый для работы объем памяти. В школьной «теории алгоритмов» эти два правила не рассматриваются. В то же время практическая работа с алгоритмами (программирование) начинается именно с реализации этих правил.

В языках программирования распределение памяти осуществляется декларативными операторами (операторами описания переменных). В языке Бейсик не все переменные описываются, обычно описываются только массивы. Но все равно при запуске программы транслятор языка анализирует все идентификаторы в тексте программы и отводит память под соответствующие переменные.

Третье правило – дискретность. Алгоритм строится из отдельных шагов (действий, операций, команд). Множество шагов, из которых составлен алгоритм, конечно.

Четвертое правило – детерменированность. После каждого шага необходимо указывать, какой шаг выполняется следующим, либо давать команду остановки. Пятое правило – сходимость (результативность). Алгоритм должен завершать работу после конечного числа шагов. При этом необходимо указать, что считать результатом работы алгоритма.

3. Формы записей алгоритмов. Блок-схемы.

Алгоритм может быть записан различными способами: на естественном языке в виде описания; в виде графических блок-схем; на специальном алгоритмическом языке. В школе на уроках информатики для записи алгоритмов используется, так называемый, "школьный алгоритмический язык". Этот язык по существу является "мёртвым" языком, так как на нём не работают компьютеры, и мы не будем им пользоваться. Запись алгоритмов на родном языке доступна и удобна. Примеров таких записей множество, хотя бы книга кулинарных рецептов есть не что иное, как сборник алгоритмов, написанных на родном языке.

Существенным недостатком такой записи является недостаточная наглядность, что особенно сказывается, когда алгоритм имеет много ветвлений. Поэтому, мы будем записывать наши алгоритмы в виде блок-схемы.

Блок-схема — распространенный тип схем (*графических моделей*), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков

различной формы, соединенных между собой линиями, указывающими направление последовательности.

Основные элементы схем алгоритма

Наименование	Обозначение	Функция
Блок начало-конец (пуск-остановка)		Элемент отображает выход во внешнюю среду и вход из внешней среды (наиболее частое применение − начало и конец программы). Внутри фигуры записывается соответствующее действие.
Блок действия		Выполнение одной или нескольких операций, обработка данных любого вида (изменение значения данных, формы представления, расположения). Внутри фигуры записывают непосредственно сами операции, например, операцию присваивания: `a = 10*b + c`.
Логический блок (блок условия)		Отображает решение или функцию переключательного типа с одним входом и двумя или более альтернативными выходами, из которых только один может быть выбран после вычисления условий, определенных внутри этого элемента. Вход в элемент обозначается линией, входящей обычно в верхнюю вершину элемента. Если выходов два или три, то обычно каждый выход обозначается линией, выходящей из оставшихся вершин (боковых и нижней). Если выходов больше трех, то их следует показывать одной линией, выходящей из вершины (чаще нижней) элемента, которая затем разветвляется. Соответствующие результаты вычислений могут записываться рядом с линиями, отображающими эти пути. Примеры решения: в программировании − условные операторы `if` (два выхода: `true`, `false`) и `case` (множество выходов).
Предопределённый процесс		Символ отображает выполнение процесса, состоящего из одной или нескольких операций, который определен в другом месте программы (в подпрограмме, модуле). Внутри символа записывается название процесса и передаваемые в него данные. Например, в программировании − вызов процедуры или функции.
Данные (ввод-вывод)		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). Данный символ не определяет носителя данных (для указания типа носителя данных используются специфические символы).

Граница цикла		Символ состоит из двух частей − соответственно, начало и конец цикла − операции, выполняемые внутри цикла, размещаются между ними. Условия цикла и приращения записываются внутри символа начала или конца цикла − в зависимости от типа организации цикла. Часто для изображения на блок-схеме цикла вместо данного символа используют символ условия, указывая в нём решение, а одну из линий выхода замыкают выше в блок-схеме (перед операциями цикла).

4. Основные алгоритмические конструкции: линейные, разветвляющиеся, циклические.

Все имеющиеся алгоритмы можно разделить на три вида:

- ***линейные алгоритмы;***
- ***алгоритмы ветвления;***
- ***циклические алгоритмы.***

Линейный алгоритм – набор команд (указаний), выполняемых последовательно во времени друг за другом.

• ***Разветвляющийся*** алгоритм – алгоритм, содержащий хотя бы одно условие, в результате проверки которого ЭВМ обеспечивает переход на один из двух возможных шагов.

• ***Циклический алгоритм*** – алгоритм, предусматривающий многократное повторение одного и того же действия (одних и тех же операций) над новыми исходными данными. К циклическим алгоритмам сводится большинство методов вычислений, перебора вариантов.

Цикл программы – последовательность команд (серия, тело цикла), которая может выполняться многократно (для новых исходных данных) до удовлетворения некоторого условия.

Теорема Дейкстра. Алгоритм любой сложности можно реализовать, используя только три конструкции: следования (линейные), выбора (ветвления) и повторения (циклические).

1. 2 Лекция № 2 (2 часа).

Тема: «Логические основы алгоритмизации»

1.2.1 Вопросы лекции:

1. Основы алгебры логики.
2. Логические операции с высказываниями.
3. Законы логических операций. Таблицы истинности

1.2.2 Краткое содержание вопросов:

1. Основы алгебры логики.

Информация (данные, машинные команды и т. д.) в компьютере представлена в двоичной системе счисления, в которой используется две цифры – 0 и 1. Электрический сигнал, проходящий по электронным схемам и соединительным проводникам (шинам) компьютера, может принимать значения 1 (высокий уровень электрического напряжения) и 0 (низкий уровень электрического напряжения) и рассматривается как импульсный сигнал, который математически может быть описан в виде двоичной переменной, принимающей также значения 0 или 1. Для решения различных логических задач, например, связанных с анализом и синтезом цифровых схем и электронных блоков компьютера, широко используются логические функции и логические операции с двоичными переменными, которые называются также логическими переменными.

Логические переменные изучаются в специальном разделе математики, который носит название алгебры логики (высказываний), или булевой алгебры. Булева алгебра названа по имени английского математика Джорджа Буля (1815–1864), внесшего значительный вклад в разработку алгебры логики. Предметом изучения алгебры логики являются высказывания, при этом анализу подвергается истинность или ложность высказываний, а не их смысловое содержание. Простые высказывания в алгебре логики обозначаются заглавными латинскими буквами: *A, B, C, D,…* и т. д. Составные высказывания на естественном языке образуются с помощью союзов. В алгебре логики эти союзы заменяются логическими операциями. В соответствии с алгеброй логики любое составное высказывание можно рассматривать как логическую функцию *F(A, B, C, …)*, аргументами которой являются логические переменные *A, B, C…* (простые высказывания). Логические функции и логические переменные (аргументы) принимают только два значения: «истина», которая обозначается логической единицей – 1 и «ложь», обозначаемая логическим нулем – 0. Логическую функцию называют также предикатом.

Логические выражения могут быть простыми и сложными.

Простое логическое выражение состоит из одного высказывания и не содержит логические операции. В простом логическом выражении возможно только два результата — либо «истина», либо «ложь».

Сложное логическое выражение содержит высказывания, объединенные логическими операциями. По аналогии с понятием функции в алгебре сложное логическое выражение содержит аргументы, которыми являются высказывания.

2. Логические операции с высказываниями.

В качестве основных **логических операций** в сложных логических выражениях используются следующие:

• НЕ (логическое отрицание, инверсия);

• ИЛИ (логическое сложение, дизъюнкция);

• И (логическое умножение, конъюнкция).

Логическое отрицание является одноместной операцией, так как в ней участвует одно высказывание. Логическое сложение и умножение — двуместные операции, в них участвует два высказывания. Существуют и другие операции, например операции следования и эквивалентности, правило работы которых можно вывести на основании основных операций.

Все операции алгебры логики определяются **таблицами истинности** значений. Таблица истинности определяет результат выполнения операции для **всех возможны**х логических значений исходных высказываний. Количество вариантов, отражающих результат применения операций, будет зависеть от количества высказываний в логическом выражении, например:

- таблица истинности одноместной логической операции состоит из двух строк: два различных значения аргумента — «истина» (1) и «ложь» (0) и два соответствующих им значения функции;
- в таблице истинности двуместной логической операции — четыре строки: 4 различных сочетания значений аргументов — 00, 01, 10 и 11 и 4 соответствующих им значения функции;
- если число высказываний в логическом выражении N, то таблица истинности будет содержать 2^N строк, так как существует 2^N различных комбинаций возможных значений аргументов.

Операция НЕ — логическое отрицание (инверсия)

Логическая операция НЕ применяется к одному аргументу, в качестве которого может быть и простое, и сложное логическое выражение. Результатом операции НЕ является следующее:

• если исходное выражение истинно, то результат его отрицания будет ложным;

• если исходное выражение ложно, то результат его отрицания будет истинным.

Для операции отрицания НЕ приняты следующие условные обозначения:
не А, Ā, not A, ¬A.
Результат операции отрицания НЕ определяется следующей таблицей истинности:

A	не A
0	1
1	0

Результат операции отрицания истинен, когда исходное высказывание ложно, и наоборот.
Принцип работы переключателя настольной лампы таков: если лампа горела, переключатель выключает ее, если лампа не горела — включает ее. Такой переключатель можно считать электрическим аналогом операции отрицания.

Операция ИЛИ — логическое сложение (дизъюнкция, объединение)

Логическая операция ИЛИ выполняет функцию объединения двух высказываний, в качестве которых может быть и простое, и сложное логическое выражение. Высказывания, являющиеся исходными для логической операции, называют аргументами. Результатом операции ИЛИ является выражение, которое бу¬дет истинным тогда и только тогда, когда истинно будет хотя бы одно из исходных выражений.
Применяемые обозначения: A или B, A V B, A or B.
Результат операции ИЛИ определяется следующей таблицей истинности:

A	B	A или B
0	0	0
0	1	1
1	0	1
1	1	1

Результат операции ИЛИ истинен, когда истинно А, либо истинно В, либо истинно и А и В одновременно, и ложен тогда, когда аргументы А и В — ложны.
Кто хоть однажды использовал елочную гирлянду с параллельным соединением лампочек, знает, что гирлянда будет светить до тех пор, пока цела хотя бы одна лампочка. Логическая операция ИЛИ чрезвычайно схожа с работой подобной гирлянды, ведь результат операции ложь только в одном случае — когда все аргументы ложны.

Операция И — логическое умножение (конъюнкция)

Логическая операция И выполняет функцию пересечения двух высказываний (аргументов), в качестве которых может быть и простое, и сложное логическое выражение. Результатом операции И является выражение, которое будет истинным тогда и только тогда, когда истинны оба исходных выражения.
Применяемые обозначения: A и B, A Λ B, A & B, A and B.
Результат операции И определяется следующей таблицей истинности:

A	B	A и B
0	0	0
0	1	0
1	0	0
1	1	1

Результат операции И истинен тогда и только тогда, когда истинны одновременно высказывания А и В, и ложен во всех остальных случаях.
Логическую операцию И можно сравнить с последовательным соединением лампочек в гирлянде. При наличии хотя бы одной неработающей лампочки электрическая цепь оказывается разомкнутой, то есть гирлянда не работает. Ток протекает только при одном условии — все составляющие цепи должны быть исправны.

Операция «ЕСЛИ-ТО» — логическое следование (импликация)

Эта операция связывает два простых логических выражения, из которых первое является условием, а второе — следствием из этого условия.

Применяемые обозначения:

если A, то B; A влечет B; if A then B; A→ B.

Таблица истинности:

A	B	A → B
0	0	1
0	1	1
1	0	0
1	1	1

Результат операции следования (импликации) ложен только тогда, когда предпосылка A истинна, а заключение B (следствие) ложно.

Приведем примеры операции следования.

2. Рассмотрим два высказывания: A {x делится на 9}, B {x делится на 3}. Операция A → B означает следующее: «Если число делится на 9, то оно делится и на 3». Рассмотрим возможные варианты:

■ A — ложно, B — ложно. Можно найти такие числа, для которых истиной является высказывание «если A — ложно, то и B — ложно». Например, x = 4, 17, 22.

■ A — ложно, B — истинно. Можно найти такие числа, для которых истиной является высказывание «если A — ложно, то B — истинно». Например, x = б, 12, 21.

■ A — истинно, B — ложно. Невозможно найти такие числа, которые делились бы на 9, но не делились на 3. Истинная предпосылка не может приводить к ложному результату импликации.

■ A — истинно, B — истинно. Можно найти такие числа, для которых истиной является высказывание «если A — истинно, то и B — истинно». Например, x = 9, 18, 27.

Операция «A тогда и только тогда, когда B» (эквивалентность, равнозначность)

Применяемое обозначение: A ↔ B, A ~ B.

Таблица истинности:

A	B	A↔B
0	0	1
0	1	0
1	0	0
1	1	1

Результат операции эквивалентность истинен только тогда, когда A и B одновременно истинны или одновременно ложны.

3. Законы логических операций. Таблицы истинности

В алгебре логики имеются законы, которые записываются в виде соотношений. Логические законы позволяют производить равносильные (эквивалентные) преобразования логических выражений. Преобразования называются равносильными, если истинные значения исходной и полученной после преобразования логической функции совпадают при любых значениях входящих в них логических переменных.

Для простоты записи приведем основные законы алгебры логики для двух логических переменных *A* и *B*. Эти законы распространяются и на другие логические переменные.

1. Закон противоречия: $A \wedge \overline{A} = 0$.

2. Закон исключенного третьего: $A \vee \overline{A} = 1$

3. Закон двойного отрицания: $\overline{\overline{A}} = A$.

4. Законы де Моргана: $\overline{A \vee B} = \overline{A} \wedge \overline{B}$

5. Законы повторения: $A \wedge A = A$, $A \vee A = A$.
6. Законы поглощения: $A \vee (A \wedge B) = A$, $A \wedge (A \vee B) = A$.

Для логических переменных справедливы и общематематические законы. Для простоты записи приведем общематематические законы для трех логических переменных *A, B и C:*

1. Коммутативный закон: *A & B = B & A; A ? B = B ? A.*
2. Ассоциативный закон: *A & (B & C) = (A & B) & C; A ? (B ? C) = (A ? B) ? C.*
3. Дистрибутивный закон: *A & (B ? C) = (A & B) ? (A & C).*

Как уже отмечалось, с помощью законов алгебры логики можно производить равносильные преобразования логических выражений с целью их упрощения. В алгебре логики на основе принятого соглашения установлены следующие правила (приоритеты) для выполнения логических операций: первыми выполняются операции в скобках, затем в следующем порядке: инверсия (отрицание), конъюнкция, дизъюнкция, импликация, эквиваленция.

Логические функции и таблицы истинности

Соотношения между логическими переменными и логическими функциями в алгебре логики можно отобразить также с помощью соответствующих таблиц, которые носят название таблиц истинности. Таблицы истинности находят широкое применение, поскольку наглядно показывают, какие значения принимает логическая функция при всех сочетаниях значений ее логических переменных. Таблица истинности состоит из двух частей. Первая (левая) часть относится к логическим переменным и содержит полный перечень возможных комбинаций логических переменных *A, B, C*... и т. д. Вторая (правая) часть этой таблицы определяет выходные состояния как логическую функцию от комбинаций входных величин.

Например, для логической функции $F = A \vee B \vee C$ (дизъюнкции) трех логических переменных *A, B,* и *C* таблица истинности будет иметь вид, показанный на рис. 1. Для записи значений логических переменных и логической функции данная таблица истинности содержит 8 строк и 4 столбца, т. е. число строк для записи значений аргументов и функции любой таблицы истинности будет равно 2^n, где n – число аргументов логической функции, а число столбцов равно $n + 1$.

A	B	C	$F = A \vee B \vee C$
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

Рис. 1. Таблица истинности для логической функции $F = A \vee B \vee C$

Логические функции, с помощью которых можно выразить другие логические функции методом суперпозиции, называются базовыми логическими функциями. Такой набор базовых логических функций называется функционально полным набором логических функций. На практике наиболее широко в качестве такого набора используют три логических функции: конъюнкцию, дизъюнкцию и отрицание. Если логическая функция представлена с помощью базовых функций, то такая форма представления называется нормальной. В предыдущем примере логическая функция Шеффера, выраженная через базовые функции, представлена в нормальной форме.

При помощи набора базовых функций и соответствующих им технических устройств, реализующих эти логические функции, можно разработать и создать любое логическое устройство или систему.

1. 3 Лекция № 3 (2 часа).

Тема: «Языки и системы программирования»

1.3.1 Вопросы лекции:

1. Эволюция языков программирования.
2. Классификация языков программирования.
3. Понятие системы программирования.

1.3.2 Краткое содержание вопросов:

1. Эволюция языков программирования.

Язык программирования – это способ записи программ решения задач на ЭВМ в понятной для компьютера форме.

Языки программирования – это искусственно созданные языки. От естественных они отличаются ограниченным числом «слов» и очень строгими правилами записи команд (операторов). Языки программирования – это формальные языки общения человека с ЭВМ, предназначенные для описания совокупности инструкций, выполнение которых обеспечивает правильное решение требуемой задачи.

Конкретный компьютер способен работать с программами, написанными на его родном машинном языке. Поэтому машинных языков большое количество. Процессор компьютера непосредственно понимает язык машинных команд (ЯМК). Для программирования на ЯМК программист должен знать числовые коды всех машинных команд, должен сам распределять память под команды программы и данные.

Машинный язык – это язык, с помощью которого программист задает команды, оперируя с ячейками памяти, полностью используя возможности машины.

Суть этого языка — набор кодов, обязательно понятных процессору, к кому обращаются. Части («слова») этого языка называются *инструкциями*, каждая из которых представляет собой одно элементарное действие для центрального процессора, как, например, считывание информации из ячейки памяти. Если знаешь, можно непосредственно управлять процессором. Например, для организации чтения блока данных с гибкого диска программист может использовать 16 различных команд, каждая из которых требует 13 параметров, таких как номер блока на диске, номер сектора на дорожке и т. п. Когда выполнение операции с диском завершается, контроллер возвращает 23 значения, отражающие наличие и типы ошибок, которые надо анализировать. Уже одно обращение к процессору громоздко, а уж анализ ошибок и вовсе представляется невообразимым, особенно, если не именно с этим процессором приходится работать. Вообще набор команд машинного языка сильно зависит от типа процессора.

В 1950-х гг. появляются первые средства автоматизации программирования – языки автокоды. Не очень заметный, казалось бы, шаг — переход к символическому кодированию машинных команд — имел на самом деле огромное значение. Программисту не надо было больше вникать в хитроумные способы кодирования команд на аппаратном уровне.

Автокод — язык программирования, предложения которого по своей структуре в основном подобны командам и обрабатываемым данным конкретного машинного языка.

Позднее для языков этого уровня стало применяться названия ассемблера. Появление языков типа Автокод – Ассемблер облегчило участь программистов. Переменные величины стали изображаться символическими именами. Числовые коды операций заменились на мнемонические (словесные) обозначения, которые легче запомнить. Язык программирования стал понятнее для человека, но при этом удалился от языка машинных команд. Чтобы компьютер мог исполнять программы на автокоде, потребовался специальный переводчик – транслятор. Здесь впервые в истории развития программирования появились два представления программы: в исходных текстах и в откомпилированном виде.

Транслятор – это системная программа, переводящая текс программы на Автокоде в текст эквивалентной программы на ЯМК. Перевод программы на языке автокода в исполнимый машинный код (вычисление выражений, раскрытие макрокоманд, замена мнемоник собственно машинными кодами и символьных адресов на абсолютные или относительные адреса) производится *ассемблером* — программой-транслятором, которая и дала языку ассемблера его название.

Команды языка ассемблера один к одному соответствуют командам процессора. Фактически, они и представляют собой более удобную для человека символьную форму записи — *мнемокоды* — команд и их аргументов. При этом одной команде языка ассемблера может соответствовать несколько вариантов команд процессора.

Исторически, если первым поколением языков программирования считать машинные коды, то язык ассемблера можно рассматривать как второе поколение языков программирования. Однако, языки ассемблера сохраняют свою нишу, обусловленную их уникальными преимуществами в части эффективности и возможности полного использования специфических средств конкретной платформы.

На языке ассемблера пишут программы или их фрагменты в тех случаях, когда критически важны:

- быстродействие (драйверы, игры);
- объём используемой памяти (загрузочные секторы, встраиваемое программное обеспечение, вирусы, антивирусы, программные защиты).

С использованием программирования на языке ассемблера производятся:

- Оптимизация критичных к скорости участков программ в программах на языках высокого уровня, таких как C++ или Pascal. Это особенно актуально для игровых приставок, имеющих фиксированную производительность.
- Создание операционных систем (ОС) или их компонентов. В настоящее время подавляющее большинство ОС пишут на более высокоуровневых языках. Аппаратно зависимые участки кода, такие как загрузчик ОС, уровень абстрагирования от аппаратного обеспечения и ядро, часто пишутся на языке ассемблера.
- Создание драйверов. Хотя, в настоящее время, драйверы также стремятся писать на языках высокого уровня (на высокоуровневом языке много проще написать надёжный драйвер), в связи с повышенными требованиями к надёжности, и достаточной производительностью современных процессоров и достаточным совершенством компиляторов с языков высокого уровня, подавляющая часть современных драйверов пишется на языке ассемблера.
- Создание антивирусов и других защитных программ.
- Написание трансляторов языков программирования.

Языки типа Автокод – Ассемблер являются машинно-зависимыми языками, т.е. они ориентированы на конкретный тип процессора и учитывают его особенности.

Машинно-зависимые языки – это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.).

Эти языки называют языками программирования низкого уровня.

Языки высокого уровня (машинно-независимые языки)

Недостатки языка ассемблера, сложность разработки на нём больших программных комплексов привели к появлению языков третьего поколения — языков программирования высокого уровня (таких как Фортран, Лисп, Кобол, Алгол, Бейсик, Паскаль, Си и др.). Именно языки программирования высокого уровня и их наследники в основном используются в настоящее время в индустрии информационных технологий. Языки высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны

для человека, с помощью них, можно писать программы до нескольких тысяч строк длиной. Конечно, это достижение было очень ценно.

Машинно-независимые языки – это средство описания алгоритмов решения задач, не требующих от пользователя знания особенностей организации функционирования ЭВМ и вычислительной системы.

Операторы языка описывают действия, которые должна выполнять система после трансляции программы на машинный язык.

2. Классификация языков программирования.

Первыми языками высокого уровня были Фортран, Кобол (в США) и Алгол (в Европе). Для первых ЯПВУ предметная ориентация языков была характерной чертой. За всю историю было создано более тысячи языков, но распространились и выдержали испытание временем только несколько.

Все языки, о которых пойдет речь, имеют одно общее свойство: они императивны. Это означает, что программы на них, в конечном итоге, представляют собой пошаговое описание решения той или иной задачи. Можно попытаться описывать лишь постановку проблемы, а решать задачу поручить компилятору.

1) Язык **Фортран** был ориентирован на научно-технические расчеты математического характера, для научных и инженерных вычислений. **Фортра́н** (*Fortran*) — первый язык программирования высокого уровня, имеющий транслятор. Создан в период с 1954 по 1957 год группой программистов под руководством Джона Бэкуса. Название Fortran является сокращением от **FOR**mula **TRAN**slator (переводчик формул). Одно из преимуществ современного Фортрана — большое количество написанных на нём программ и библиотек подпрограмм. Среди учёных, например, ходит такая присказка, что любая математическая задача уже имеет решение на Фортране, и, действительно, можно найти среди тысяч фортрановских пакетов и пакет для перемножения матриц, и пакет для решения сложных интегральных уравнений, и многие, многие другие. Современный Фортран (Fortran 95 и Fortran 2003) приобрёл черты, необходимые для эффективного программирования для новых вычислительных архитектур, позволяет применять современные технологии программирования.

2) В 1960 году командой во главе с Петером Науром (Peter Naur) был создан язык программирования **Algol.** Этот язык дал начало целому семейству Алгол-подобных языков (важнейший представитель — Pascal). В 1968 году появилась новая версия языка. Она не нашла столь широкого практического применения, как первая версия, но была весьма популярна в кругах теоретиков. Язык был достаточно интересен, так как обладал многими уникальными на так момент характеристиками.

3) В 1960 году был создан язык программирования **Cobol.** Он задумывался как язык для программирования экономических задач, и он стал таковым. На Коболе написаны тысячи прикладных коммерческих систем. Отличительной особенностью языка является возможность эффективной работы с большими массивами данных, что характерно именно коммерческих приложений. Популярность Кобола столь высока, что даже сейчас, при всех его недостатках (по структуре и замыслу Кобол во многом напоминает Фортран) появляются новые его диалекты и реализации.

4) В 1963 - 64 годах профессорами Дартмутского колледжа Томасом Курцем и Джоном Кемени был создан язык программирования **BASIC** (Beginners' All-Purpose Symbolic Instruction Code - универсальный код символических инструкций для начинающих). Язык задумывался в первую очередь как средство обучения и как первый изучаемый язык программирования. Он предназначался для более «простых» пользователей, не столько заинтересованных в скорости исполнения программ, сколько просто в возможности использовать компьютер для решения своих задач, не имея специальной подготовки. Надо сказать, что BASIC действительно стал языком, на котором учатся программировать (по крайней мере, так было еще несколько лет назад; сейчас эта роль отходит к Pascal). Было создано несколько мощных реализаций BASIC, поддерживающих самые современные

концепции программирования. Язык был основан частично на Фортране II и частично на Алголе 60, с добавлениями, делающими его удобным для работы в режиме разделения времени и, позднее, обработки текста и матричной арифметики.

5) В 1969 году был создан язык **SETL** — язык для описания операций над множествами. Основной структурой данных в языке является множество, а операции аналогичны математическим операциям над множествами. Полезен при написании программ, имеющих дело со сложными абстрактными объектами.

6) Пролог (фр. ***Pro****grammation en* ***Log****ique*) — язык и система логического программирования, основанные на языке предикатов математической логики, представляющей собой подмножество логики предикатов первого порядка (1971).

Основными понятиями в языке Пролог являются факты, правила логического вывода и запросы, позволяющие описывать базы знаний, процедуры логического вывода и принятия решений. Факты в языке Пролог описываются логическими предикатами с конкретными значениями. Правила в Прологе записываются в форме правил логического вывода с логическими заключениями и списком логических условий.

Особую роль в интерпретаторе Пролога играют конкретные запросы к базам знаний, на которые система логического программирования генерирует ответы «истина» и «ложь». Для обобщённых запросов с переменными в качестве аргументов созданная система Пролог выводит конкретные данные в подтверждение истинности обобщённых сведений и правил вывода.

7) Значительным событием в истории языков программирования стало создание в 1971 году языка **Паскаль** швейцарским профессором Никлаусом Виртом. Один из наиболее известных языков программирования, используется для обучения программированию в старших классах и на первых курсах вузов, является базой для ряда других языков. Далее разрабатывается система программирования Турбо Паскаль, включающая язык, транслятор и операционную оболочку, обеспечивающую пользователю удобство работы.

8) В 1972 году Керниганом и Ритчи был создан язык программирования **Си**. Он создавался как язык для разработки операционных систем. Си часто называют «переносимым ассемблером», имея в виду то, что он позволяет работать с данными практически так же эффективно, как на ассемблере, предоставляя при этом структурированные управляющие конструкции и абстракции высокого уровня (структуры и массивы). Именно с этим связана его огромная популярность и поныне. И именно это является его ахиллесовой пятой. Компилятор С очень слабо контролирует типы, поэтому очень легко написать внешне совершенно правильную, но логически ошибочную программу.

В 1986 году Бьярн Страуструп создал первую версию языка C++, добавив в язык С объектно-ориентированные черты, взятые из Simula (см. ниже), и исправив некоторые ошибки и неудачные решения языка. C++ продолжает совершенствоваться и в настоящее время, так в 1998 году вышла новая (третья) версия стандарта, содержащая в себе некоторые довольно существенные изменения. Язык стал основой для разработки современных больших и сложных проектов. У него имеются, однако же, и слабые стороны, вытекающие из требований эффективности.

3. Понятие системы программирования.

Системой программирования называется комплекс программ, предназначенный для автоматизации программирования задач на ЭВМ. Система программирования освобождает проблемного пользователя или прикладного программиста от необходимости написания программ решения своих задач на неудобном для него языке машинных команд и предоставляют им возможность использовать специальные языки более высокого уровня. Для каждого из таких языков, называемых входными или исходными, система программирования имеет программу, осуществляющую автоматический перевод (трансляцию) текстов программы с входного языка на язык машины. Обычно система программирования содержит описания применяемых языков программирования,

программы-трансляторы с этих языков, а также развитую библиотеку стандартных подпрограмм. Важно различать язык программирования и реализацию языка.

Язык – это набор правил, определяющих систему записей, составляющих программу, синтаксис и семантику используемых грамматических конструкций.

Реализация языка – это системная программа, которая переводит (преобразует) записи на языке высокого уровня в последовательность машинных команд.

Имеется два основных вида средств реализации языка: компиляторы и интерпретаторы.

Компилятор транслирует весь текст программы, написанной на языке высокого уровня, в ходе непрерывного процесса. При этом создается полная программа в машинных кодах, которую затем ЭВМ выполняет без участия компилятора.

Интерпретатор последовательно анализирует по одному оператору программы, превращая при этом каждую синтаксическую конструкцию, записанную на языке высокого уровня, в машинные коды и выполняя их одна за другой. Интерпретатор должен постоянно присутствовать в зоне основной памяти вместе с интерпретируемой программой, что требует значительных объемов памяти.

Следует заметить, что любой язык программирования может быть как интерпретируемым, так и компилируемым, но в большинстве случаев у каждого языка есть свой предпочтительный способ реализации. Языки Фортран, Паскаль в основном компилируют; язык Ассемблер почти всегда интерпретирует; языки Бейсик и Лисп широко используют оба способа.

Основным преимуществом компиляции является скорость выполнения готовой программы. Интерпретируемая программа неизбежно выполняется медленнее, чем компилируемая, поскольку интерпретатор должен строить соответствующую последовательность команд в момент, когда инструкция предписывает выполнение.

В то же время интерпретируемый язык часто более удобен для программиста, особенно начинающего. Он позволяет проконтролировать результат каждой операции. Особенно хорошо такой язык подходит для диалогового стиля разработки программ, когда отдельные части программы можно написать, проверить и выполнить в ходе создания программы, не отключая интерпретатора.

По набору входных языков различают системы программирования одно- и многоязыковые. Отличительная черта многоязыковых систем состоит в том, что отдельные части программы можно составлять на разных языках и помощью специальных обрабатывающих программ объединять их в готовую для исполнения на ЭВМ программу.

Для построения языков программирования используется совокупность общепринятых символов и правил, позволяющих описывать алгоритмы решаемых задач и однозначно истолковывать смысл созданного написания. Основной тенденцией в развитии языков программирования является повышение их семантического уровня с целью облегчения процесса разработки программ и увеличения производительности труда их составителей.

По структуре, уровню формализации входного языка и целевому назначению различают системы программирования машинно-ориентированные и машинно-независимые.

Машинно-ориентированные системы программирования имеют входной язык, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.). Машинно-ориентированные системы позволяют использовать все возможности и особенности машинно-зависимых языков:

- высокое качество создаваемых программ;
- возможность использования конкретных аппаратных ресурсов;
- предсказуемость объектного кода и заказов памяти;
- для составления эффективных программ необходимо знать систему команд и особенности функционирования данной ЭВМ;

- трудоемкость процесса составления программ (особенно на машинных языках и ЯСК), плохо защищенного от появления ошибок;
- низкая скорость программирования;
- невозможность непосредственного использования программ, составленных на этих языках, на ЭВМ других типов.

Машинно-ориентированные системы по степени автоматического программирования подразделяются на классы:

1.Машинный язык. В таких системах программирования отдельный компьютер имеет свой определенный Машинный Язык (далее МЯ), ему предписывают выполнение указываемых операций над определяемыми ими операндами, поэтому МЯ является командным. Однако, некоторые семейства ЭВМ (например, ЕС ЭВМ, IBM/370/ и др.) имеют единый МЯ для ЭВМ разной мощности. В команде любого из них сообщается информация о местонахождении операндов и типе выполняемой операции. В новых моделях ЭВМ намечается тенденция к повышению внутренних языков машинно-аппаратным путем реализовывать более сложные команды, приближающиеся по своим функциональным действиям к операторам алгоритмических языков программирования.

2.Система Символического Кодирования. В данных системах используются Языки Символического Кодирования (далее ЯСК), которые так же, как и МЯ, являются командными. Однако коды операций и адреса в машинных командах, представляющие собой последовательность двоичных (во внутреннем коде) или восьмеричных (часто используемых при написании программ) цифр, в ЯСК заменены символами (идентификаторами), форма написания которых помогает программисту легче запоминать смысловое содержание операции. Это обеспечивает существенное уменьшение числа ошибок при составлении программ. Использование символических адресов – первый шаг к созданию ЯСК. Команды ЭВМ вместо истинных (физических) адресов содержат символические адреса. По результатам составленной программы определяется требуемое количество ячеек для хранения исходных промежуточных и результирующих значений. Назначение адресов, выполняемое отдельно от составления программы в символических адресах, может проводиться менее квалифицированным программистом или специальной программой, что в значительной степени облегчает труд программиста.

3.Автокоды. Существуют системы программирования, использующие языки, которые включают в себя все возможности ЯСК, посредством расширенного введения макрокоманд – они называются Автокоды. В различных программах встречаются некоторые достаточно часто использующиеся командные последовательности, которые соответствуют определенным процедурам преобразования информации. Эффективная реализация таких процедур обеспечивается оформлением их в виде специальных макрокоманд и включением последних в язык программирования, доступный программисту. Макрокоманды переводятся в машинные команды двумя путями – расстановкой и генерированием. В постановочной системе содержатся «остовы» – серии команд, реализующие требуемую функцию, обозначенную макрокомандой. Макрокоманды обеспечивают передачу фактических параметров, которые в процессе трансляции вставляются в «остов» программы, превращая её в реальную машинную программу. В системе с генерацией имеются специальные программы, анализирующие макрокоманду, которые определяют, какую функцию необходимо выполнить и формируют необходимую последовательность команд, реализующих данную функцию. Обе указанных системы используют трансляторы с ЯСК и набор макрокоманд, которые также являются операторами автокода. Развитые автокоды получили название Ассемблеры. Сервисные программы и пр., как правило, составлены на языках типа Ассемблер.

4.Макрос. В таких системах язык, являющийся средством для замены последовательности символов описывающих выполнение требуемых действий ЭВМ на более сжатую форму – называется Макрос (средство замены). В основном, Макрос

предназначен для того, чтобы сократить запись исходной программы. Компонент программного обеспечения, обеспечивающий функционирование макросов, называется макропроцессором. На макропроцессор поступает макросопределяющий и исходный текст. Реакция макропроцессора на вызов – выдача выходного текста. Макрос одинаково может работать, как с программами, так и с данными.

Машинно-независимые системы программирования – это средство описания алгоритмов решения задач и информации, подлежащей обработке. Они удобны в использовании для широкого круга пользователей и не требуют от них знания особенностей организации функционирования ЭВМ. В таких системах программы, составляемые языках, имеющих название высокоуровневых языков программирования, представляют собой последовательности операторов, структурированные согласно правилам рассматривания языка (задачи, сегменты, блоки и т.д.). Операторы языка описывают действия, которые должна выполнять система после трансляции программы на МЯ. Таким образом, командные последовательности (процедуры, подпрограммы), часто используемые в машинных программах, представлены в высокоуровневых языках отдельными операторами. Программист получил возможность не расписывать в деталях вычислительный процесс на уровне машинных команд, а сосредоточиться на основных особенностях алгоритма.

Среди машинно-независимых систем программирования следует выделить:

1.Процедурно-ориентированные системы. Входные языки программирования в таких системах служат для записи алгоритмов (процедур) обработки информации, характерных для решения задач определенного класса. Эти языки, должны обеспечить программиста средствами, позволяющими коротко и четко формулировать задачу и получать результаты в требуемой форме. Процедурных языков очень много, например: Фортран, Алгол – языки, созданные для решения математических задач; Simula, Слэнг - для моделирования; Лисп, Снобол – для работы со списочными структурами.

2.Проблемно-ориентированные системы в качестве входного языка используют язык программирования с проблемной ориентацией. С расширением областей применения вычислительной техники возникла необходимость формализовать представление постановки и решение новых классов задач. Необходимо было создать такие языки программирования, которые, используя в данной области обозначения и терминологию, позволили бы описывать требуемые алгоритмы решения для поставленных задач. Эти языки, ориентированные на решение определенных проблем, должны обеспечить программиста средствами, позволяющими коротко и четко формулировать задачу и получать результаты в требуемой форме. Программы, составленные на основе этих языков программирования, записаны в терминах решаемой задачи и реализуются выполнением соответствующих процедур.

3.Диалоговые языки. Появление новых технических возможностей поставило задачу перед системными программистами – создать программные средства, обеспечивающие оперативное взаимодействие человека с ЭВМ их назвали диалоговыми языками. Создавались специальные управляющие языки для обеспечения оперативного воздействия на прохождение задач, которые составлялись на любых раннее неразработанных (не диалоговых) языках. Разрабатывались также языки, которые кроме целей управления обеспечивали бы описание алгоритмов решения задач. Необходимость обеспечения оперативного взаимодействия с пользователем потребовала сохранения в памяти ЭВМ копии исходной программы даже после получения объектной программы в машинных кодах. При внесении изменений в программу система программирования с помощью специальных таблиц устанавливает взаимосвязь структур исходной и объектной программ. Это позволяет осуществить требуемые редакционные изменения в объектной программе.

4.Непроцедурные языки. Непроцедурные языки составляют группу языков, описывающих организацию данных, обрабатываемых по фиксированным алгоритмам

(табличные языки и генераторы отчетов), и языков связи с операционными системами. Позволяя четко описывать как задачу, так и необходимые для её решения действия, таблицы решений дают возможность в наглядной форме определить, какие условия должны выполнятся, прежде чем переходить к какому-либо действию. Одна таблица решений, описывающая некоторую ситуацию, содержит все возможные блок-схемы реализаций алгоритмов решения. Табличные методы легко осваиваются специалистами любых профессий. Программы, составленные на табличном языке, удобно описывают сложные ситуации, возникающие при системном анализе.

В самом общем случае для создания программы на выбранном языке программирования нужно иметь следующие компоненты.

1.*Текстовый редактор*. Специализированные текстовые редакторы, ориентированные на конкретный язык программирования, необходимы для получения файла с *исходным текстом* программы, который содержит набор стандартных символов для записи алгоритма.

2.Исходный текст с помощью *программы-компилятора* переводится в машинный код. Исходный текст программы состоит, как правило, из нескольких модулей (файлов с исходными текстами). Каждый модуль компилируется в отдельный файл с *объектным кодом*, которые затем требуется объединить в одно целое. Кроме того, системы программирования, как правило, включают в себя библиотеки стандартных подпрограмм. Стандартные подпрограммы имеют единую форму обращения, что создает возможности автоматического включения таких подпрограмм в вызывающую программу и настройки их параметров.

3.Объектный код модулей и подключенные к нему стандартные функции обрабатывает специальная программа – *редактор связей*. Данная программа объединяет объектные коды с учетом требований операционной системы и формирует на выходе работоспособное приложение – *исполнимый код* для конкретной платформы. Исполнимый код это законченная программа, которую можно запустить на любом компьютер, где установлена операционная система, для которой эта программа создавалась.

4. В современных системах программирования имеется еще один компонент – *отладчик*, который позволяет анализировать работу программы во время ее исполнения. С его помощью можно последовательно выполнять отдельные операторы исходного текста последовательно, наблюдая при этом, как меняются значения различных переменных.

1. 4 Лекция № 4 (2 часа).

Тема: «Основные элементы языка. **Операторы языка»**

1.4.1 Вопросы лекции:

1. Общие свойства языка высокого уровня (ЯВУ).
2. Синтаксис и семантика алгоритмического языка программирования.
3. Структурная схема программы на алгоритмическом языке.
4. Синтаксис операторов: присваивания, ввода-вывода, условные операторы.

1.4.2 Краткое содержание вопросов:

1. Общие свойства языка высокого уровня (ЯВУ).

Основными элементами любого языка программирования являются его алфавит, синтаксис и семантика.

Алфавит – совокупность символов, отображаемых на устройствах печати и экранах и/или вводимых с клавиатуры терминала. Обычно это набор символов Latin-1 с исключением управляющих символов. Иногда в это множество включаются неотображаемые символы с указанием правил их записи (комбинирование в лексемы).

Лексика – совокупность правил образования цепочек символов (лексем), образующих идентификаторы (переменные и метки), операторы, операции и другие лексические компоненты языка. Сюда же включаются зарезервированные (запрещенные,

ключевые) слова языка программирования, предназначенные для обозначения операторов, встроенных функций и пр. Иногда эквивалентные лексемы, в зависимости от языка программирования, могут обозначаться как одним символом алфавита, так и несколькими. Например, операция присваивания значения в языке Си обозначается как «=», а в языке Паскаль – «:=». Операторные скобки в языке Си задаются символами «{» и «}», а в языке Паскаль – begin и end. Граница между лексикой и алфавитом, таким образом, является весьма условной, тем более что компилятор обычно на фазе лексического анализа заменяет распознанные ключевые слова внутренним кодом (например, begin – 512, end – 513) и в дальнейшем рассматривает их как отдельные символы.

2. Синтаксис и семантика алгоритмического языка программирования.

Синтаксис – совокупность правил образования языковых конструкций, или предложений языка программирования – блоков, процедур, составных операторов, условных операторов, операторов цикла и пр. Особенностью синтаксиса является принцип вложенности (рекурсивность) правил построения конструкций. Это значит, что элемент синтаксиса языка в своем определении прямо или косвенно в одной из его частей содержит сам себя. Например, в определении оператора цикла телом цикла является оператор, частным случаем которого является все тот же оператор цикла.

Необходимо строгое соблюдение правил правописания (синтаксиса) программы. В частности, в Паскале однозначно определено назначение знаков пунктуации. Точка с запятой (;) ставится в конце заголовка программы, в конце раздела описания переменных, после каждого оператора. Перед словом End точку с запятой можно не ставить. Запятая (,) является разделителем элементов во всевозможных списках: списке переменных в разделе описания, списке вводимых и выводимых величин.

Строгий синтаксис в языке программирования необходим, прежде всего, для транслятора. Транслятор – это программа, которая исполняется формально. Если, допустим, разделителем в списке переменных должна быть запятая, то любой другой знак будет восприниматься как ошибка. Если точка с запятой является разделителем операторов, то транслятор в качестве оператора воспринимает всю часть текста программы от одной точки с запятой до другой. Если вы забыли поставить этот знак между какими-то двумя операторами, то транслятор будет принимать их за один, что неизбежно приведет к ошибке.

Основное назначение синтаксических правил – придать однозначный смысл языковым конструкциям. Если какая-то конструкция может трактоваться двусмысленно, значит, в ней обязательно содержится ошибка. Лучше не полагаться на интуицию, а выучить правила языка.

Синтаксис языка описывается путем последовательного усложнения понятий: сначала определяются простейшие (базовые), затем все более сложные, включающие в себя предыдущие понятия в качестве составляющих. В такой последовательности, очевидно, конечным определяемым понятием должно быть понятие программы.

Семантика – смысловое содержание конструкций, предложений языка, семантический анализ – это проверка смысловой правильности конструкции. Например, если мы в выражении используем переменную, то она должна быть определена ранее по тексту программы, а из этого определения может быть получен ее тип. Исходя из типа переменной, можно говорить о допустимости операции с данной переменной. Семантические ошибки возникают при недопустимом использовании операций, массивов, функций, операторов и пр.

Существуют языки с ленивой и с энергичной семантикой. Различие заключается, грубо говоря, в том, что в языках с энергичной семантикой вычисления производятся в том же месте, где они описаны, а в случае ленивой семантики вычисление производится только тогда, когда оно действительно необходимо. Первые языки имеют более эффективную реализацию, в то время как вторые — лучшую семантику.

Для описания синтаксиса языка программирования тоже нужен какой-то язык. В этом случае речь идет о метаязыке («надъязыке»), предназначенном для описания других языков. Наиболее распространенными метаязыками в литературе по программированию являются металингвистические формулы Бекуса – Наура (язык БНФ) и синтаксические диаграммы. Язык синтаксических диаграмм более нагляден, легче воспринимается.

В БНФ всякое синтаксическое понятие описывается в виде формулы, состоящей из правой и левой части, соединенных знаком ::=, смысл которого эквивалентен словам «по определению есть». Слева от знака ::= записывается имя определяемого понятия (метапеременная), которое заключается в угловые скобки < >, а в правой части записывается формула или диаграмма, определяющая все множество значений, которые может принимать метапеременная.

Синтаксис языка описывается путем последовательного усложнения понятий: сначала определяются простейшие (базовые), затем все более сложные, включающие в себя предыдущие понятия в качестве составляющих.

В такой последовательности, очевидно, конечным определяемым понятием должно быть понятие программы.

В записях метаформул приняты определенные соглашения. Например, формула БНФ, определяющая понятие «двоичная цифра», выглядит следующим образом:

<двоичная цифра>::=0|1

Значок «|» эквивалентен слову «или».

В диаграммах стрелки указывают на последовательность расположения элементов синтаксической конструкции; кружками обводятся символы, присутствующие в конструкции.

Понятие «двоичный код» как непустую последовательность двоичных цифр БНФ описывает так:

<двоичный код>::=<двоичная цифра>|<двоичный код><двоичная цифра>

Определение, в котором некоторое понятие определяется само через себя, называется рекурсивным. Рекурсивные определения характерны для БНФ.

Возвратная стрелка обозначает возможность многократного повторения. Очевидно, что диаграмма более наглядна, чем БНФ.

3. Структурная схема программы на алгоритмическом языке.

Каждая программа на языке программирования высокого уровня должна быть оформлена в соответствии с правилами этого языка.

Алгоритмический язык

В алгоритмическом языке структура программы имеет следующий вид:

алг <имя программы> (<список переменных>)
    <список аргументов>
    <список результатов>
нач
    <операторы>
кон

- <имя программы> — идентификатор, однозначно определяющий программу;
- <список переменных> — список величин, которые обрабатываются программой;
- <список аргументов> — список величин, которые передаются в программу для обработки;
- <список результатов> — список величин, которые вычисляются программой;
- <операторы> — конечная последовательность операторов, реализующих исходный алгоритм и составляющих тело программы.

Бейсик

<метка> оператор
…

<метка> оператор
<метка> END
<метка> — уникальный числовой идентификатор каждой строки, позволяющий операторам перехода изменять ход выполнения операторов.

Паскаль

Program <имя программы>;
    <описания>
begin
    <операторы>
end

<описания> — раздел, в котором описываются модули, используемые программой, константы, переменные, которые используются в программе, описываются пользовательские типы данных, используемые подпрограммы.

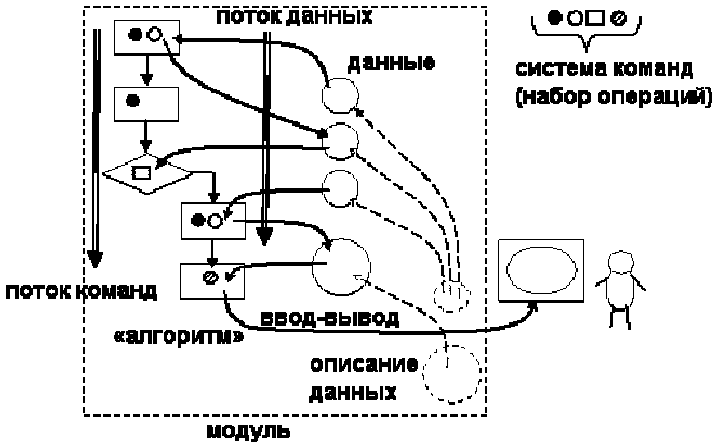


рис.1.1. Структурная схема компьютерной программы

Компьютерная программа, в отличие от абстрактного алгоритма, имеет собственные элементы, над которыми она совершает действия, и которые являются ее составной частью. Это – данные. Таким образом, она представляет собой замкнутую систему, отделенную от внешней среды. Посмотрим, из каких еще частей состоит компьютерная программа. Все они должны выражаться в соответствующих компонентах языка программирования:

прежде всего, программа работает не с пользователем, а с **данными**. Эта первая и основная компонента программы – предметы (объекты), над которыми реализуется алгоритм. Данные состоят из отдельных **переменных,** связанных как между собой непосредственно (через указатели), так и косвенно (как входные данные – результат);

в языке программирования имеются средства **описания данных,** которые позволяют программисту конструировать различные формы их представления – **типы данных;**

программа базируется на **наборе операций (системе команд),** которые можно выполнять над данными. В этот набор входят арифметические операции, присваивание (сохранение результата в переменной), ввод-вывод, проверка значения переменной и т.п..;

вторая основная компонента программы – описание порядка, последовательности выполняемых действий, также называется **алгоритмом** «в узком смысле», или алгоритмической компонентой. Она обычно состоит из двух частей. Первая часть – **выражения**, представляет собой описание линейной последовательности выполнения простейших действий из набора операций (арифметические операции, присваивание, условные выражения). Они включаются во вторую компоненту – **операторы,** которые задают ту или иную последовательность действий;

как уже отмечалось, программа работает исключительно с данными, что и определяет сущность алгоритма. В наборе операций имеются команды ввода-вывода,

осуществляющие обмен данными между переменными и внешней средой (посредством устройств ввода-вывода). С «программно-эгоцентрической» точки зрения это выглядит чистой формальностью и не является существенной частью программы;

Любая программа выполняется в компьютере. Посмотрим, как соотносятся между собой компоненты программы и компьютерной архитектуры:

компоненты программы находятся **в памяти.** В принципе, память является общей для них всех, но логически она разделяется на области, именуемые **сегментами.** Прежде всего, это **сегмент данных,** содержащий, естественно, данные программы. Алгоритмическая компонента (выражения, операторы) также находится в памяти в собственном **сегменте команд;**

одновременное нахождение в памяти «алгоритма» и данных соответствует принципу **хранимой программы.** Перед загрузкой в память эти же компоненты находятся в **программной файле,** который представляет собой точную копию представления программы в памяти – «**образ памяти**». Это позволяет рассматривать всю программу (в том числе и алгоритм) как данные для работы других программ, например, трансляторов;

набор операций, выполняемый в программе, соответствует **системе команд** процессора, на котором она выполняется. Сюда же входят команды, которые обеспечивают заданный в программе порядок действий (операторов).

И, наконец, язык программирования также содержит в себе компоненты, предназначенные для описания соответствующих частей программы:

средства описания данных: определение типов данных (форма представления) и переменных;

набор операций над основными типами данных (включая ввод-вывод), а также средства записи выражений;

набор операторов, определяющих различные варианты порядка выполнения выражений в программе (последовательность, условие, повторение, блок);

средства разбиения программы на независимые части – модули (функции, процедуры), взаимодействующие между собой через программные интерфейсы.

Определение программы уже давно дано в простой формуле: **«Программа = алгоритм + данные»**. Но в ней алгоритм и данные не просто «складываются» в одно целое как независимые части, но являются двумя взаимозависимыми элементами. Это своего рода «Янь и Инь» программы, олицетворяющие единство и борьбу двух противоположных начал (в философии этот принцип положен в основу диалектики – учения о развитии). Попробуем привести несколько аналогий, поясняющих сущность взаимодействий в этой «парочке»:

если данные можно в какой-то мере обладают свойствами пространства (объем, протяженность), то алгоритм – свойствами времени (эффективность, быстродействие). Тезис «проигрывая в пространстве, выигрываем во времени» здесь также уместен: эффективность программ может быть принципиально повышена за счет использования дополнительных структур данных в памяти;

синтаксически данные являются аналогом существительных (объектов, над которыми производятся действия), набор операций – аналогом глаголов (выполняемых действий). Программа в целом аналогично предложению, описывающему процесс – последовательность действий над заданными предметами с целью получения результата.

Взаимосвязь алгоритма и данных в программе не является простой и линейной. Процесс выполнения любой программы можно рассматривать с двух точек зрения: как последовательность выполнения операций (команд), в которых содержится информация об операндах (данных), которые они обрабатывают – **поток команд (поток управления).** С другой стороны – любой элемент данных можно рассматривать как

результат выполнения действий над исходными данными и как источник данных (операнд) для последующих результатов. Т.е. в программе также присутствует логическая последовательность вычислений (преобразований данных), называемая**потоком данных.** Исторически сложилось, что в традиционной (фон Неймановской) архитектуре в программе в явном виде задается последовательность команд, т.е. программа выглядит как **поток управления,** в котором алгоритмическая компонента является первичной (ведущей), а данные – вторичной (ведомой).

4. Синтаксис операторов: присваивания, ввода-вывода, условные операторы.

Оператор присваивания состоит из трех частей: в левой части пишется имя переменной, которой присваивается значение; в средней части пишется операция присваивания **:=** (двоеточее равно); в правой части записывается выражение, значение которого вы хотите присвоить переменной. Оператор присваивания выполняется следующим образом: сначала вычисляется значение выражения, стоящего в правой части оператора, а затем оно заносится в переменную, указанную в левой части оператора.

Примеры:

x :=5; y := x -10; x := x +2; y := y /2

В первом операторе переменной **x** присваивается значение **5**. Во втором- переменная y получает значение **5-10=-5**. Затем значение переменной **x** увеличивается на два и становится равным **7**. Наконец, в последнем операторе значение **y** уменьшается вдвое и становится равным **-2.5**. Тип выражения и тип переменной в операторе присваивания должны быть согласованы. Переменной типа **real** можно присваивать значения типа**real** или **integer**. Переменным остальных типов можно присаивать только значения своих типов, т.е. целой переменной- целое, символьной- символьное, логической- логическое. В соответствии с этими правилами в приведенных выше примерах присваиваний переменная y должна иметь тип **real**, а **x** может иметь как тип**real**, так и **integer**.

Операторы ввода/вывода

Ввод/вывод информации в программе выполняется с помощью специальных процедур ввода/вывода. Любая процедура делает некоторые действия и общается с программой через список параметров. Так процедура вывода выводит на экран компьютера те параметры, которые передаются ей через список, а процедура ввода вводит информацию с клавиатуры компьютера и помещают ее в переменные, указанные в списке параметров.

Обращение к любой процедуре состоит из двух частей: имени процедуры и списка параметров, заключенного в круглые скобки.

Параметры в списке разделяются запятыми. В процедурах ввода/вывода число параметров может быть любым. При обращении к процедуре без параметров круглые скобки не пишутся.

В Паскале имеется две стандартные процедуры вывода: write и **writeln**, выводящие значения своих параметров в стандартный файл вывода **output** (обычно это экран компьютера). Вторая процедура отличается от первой тем, что после вывода значений своих параметров переводит курсор на экране в начало следующей строки. Поясним подробнее, как это делается, на следующем примере (здесь все переменные имеют тип **integer**):

```
x := 5;
y := sqr ( x ) - 1;
n := -15;
write('Печатаем x, y и их сумму:', x, y, x+y);
write(' теперь n', n);
writeln;
writeln('С новой стро', 'ки пе', 'чатаем т','екст, текс', 'т, текст', ', текст!')
```

Результат на экране будет выглядеть так:

Печатаем x, y и их сумму:52429 теперь n-15 С новой строки печатаем текст, текст, текст, текст!

Как видно из этого примера, процедуры вывода не вставляют пробелов между целыми числами при их печати. Из-за этого три числа **5**, **24** и **29** слились на экране вместе в **52429**. Во второй строке вывода мы воспользовались тем, что пробелы между параметрами при выводе не вставляются и слили вместе разрозненные куски текста в осмысленный текст. Обратите внимание на то, что запятая внутри строкового значения выводится на печать, а вне строки она используется как разделитель параметров в списке.

При выводе между параметрами можно вставить пробелы с помощью строки из пробелов или управляя шириной поля вывода. Ширина поля вывода задается после выводимого параметра: ставится двоеточее, а затем целочисленное выражение, задающее минимальное число символьных позиций, отводимых под значение параметра. Если количество символов в значении параметра при выводе оказалось меньше этой ширины, то перед ним вставляются недостающие пробелы. Переделаем первых два оператора вывода в предыдущем примере, продемонстрировав обе возможности вставки пробелов:

```
write('Печатаем x, y и их сумму: ', x, ' ',y, ' ', x+y, ' теперь n ', n);
write('Печатаем x, y и их сумму:', x:4, y:4, x+y:4,'теперь n':10, n:3);
```

Результат будет таким:

```
Печатаем x, y и их сумму: 5 24 29 теперь n -15
Печатаем x, y и их сумму: 5 24 29 теперь n -15
```

Для выражений типа **real** спецификация вывода может иметь вид

: n : m , где n - ширина поля вывода, а m - количество цифр числа после точки, выводимых на печать:

```
write(A1:10:3, A2:10:3);
```

В Паскале имеется две процедуры ввода: **read** и **readln**. Вторая обычно используется для ввода строковых значений и нам пока не требуется. В списке параметров этих процедур должны быть только переменные. Ввод данных осуществляется в соответствии с типами этих переменных: если это числовая переменная (целая или вещественная), то из стандартного файла ввода **input** считывается число; если же это символьная переменная, то считывается один символ. Логические переменные в списке ввода не используются. При чтении чисел пробелы перед ними в файле ввода **input** пропускаются. Обычно файл стандартного ввода связан с клавиатурой. При этом все, что вводится с клавиатуры, отображается на экране компьютера. Пусть, например, требуется ввести целое число в переменную **n**, означающую число элементов суммы ряда. Это можно сделать следующим образом:

```
write('Введите число элементов ряда ');
read ( n );
```

Условный оператор

Условный оператор позволяет выполнять разветвления в программе. Его синтаксис следующий:

if *логическое выражение* **then** *оператор*$_1$ **else** *оператор*$_2$

Если *логическое выражение* истинно, то выполняется **then**-часть (т.е. *оператор*$_1$). В противном случае выполняется **else**-часть (т.е.*оператор*$_2$).

В условном операторе (else)-часть может отсутствовать. Часто в (else) и (then)-частях оператора находятся другие условные операторы. Например, математической формуле

```
        x   при  x  <0
y=   {  x 2 при 0 x 1
        x 3 при x >1
```

соответствует следующая программа:

```
if x<0 then y := x
else if x<=0 then y := sqr(x)
else y := x*sqr(x)
```

Чтобы определить в сложном условном операторе к какому **if** относится какой **else** используют следующее правило: **else**-часть относится к ближайшему **if**, еще не имеющему **else**-части.

1. 5 Лекция № 5 (2 часа).

Тема: «Операторы для организации цикла»

1.5.1 Вопросы лекции:

1. Циклические конструкции.
2. Циклы с параметром.
3. Циклы с предусловием и постусловием.
4. Вложенные циклы.

1.5.2 Краткое содержание вопросов:

1. Циклические конструкции.

При решении подавляющего большинства задач (в том числе и весьма несложных) в программе практически невозможно задать в явном виде все операции, которые необходимо выполнить. В самом деле, пусть необходимо вычислить сумму первых n членов гармонического ряда:

Y= 1+ 1/2+ 1/3 + …+ 1/ n

Очевидно, что с использованием только рассмотренных выше типов операторов можно составить программу лишь для фиксированного значения n. Например, при n=5 требуемые вычисления можно задать с помощью оператора присваивания вида:

Y:= 1+1/2+1/3+1/4+1/5

Если же значение n не фиксируется, а является исходным данным, вводимым в процессе выполнения программы (и даже константой, описанной в программе), то аналогичный оператор присваивания записать невозможно. Ибо запись вида Y:= 1+1/2+1/3+…+1/ n в языках программирования недопустима.

Для устранения возникающих трудностей служат **операторы цикла**. Они позволяют повторять выполнение отдельных частей программы. Можно выделить четыре о**ператора цикла**, присутствующих в том или ином виде во всех языках программирования: **простой арифметический оператор цикла** (цикл с параметром с шагом 1), **сложный арифметический оператор цикла** (цикл с параметром произвольного шага), **итерационный оператор цикла с предусловием**, **итерационный оператор цикла с постусловием**.

2. Циклы с параметром.

Простой арифметический оператор цикла Паскаля (цикл с параметром)

Вернемся к рассмотренной выше задаче вычисления суммы первых n членов гармонического ряда, правила которой невозможно задать в виде арифметического выражения, если значение n заранее не фиксировано.

На самом деле вычисление этой суммы можно осуществить по очень простому и компактному алгоритму: предварительно положим y=0 (с помощью оператора присваивания y:=0), а затем выполним оператор присваивания y:= y+1/ i для последовательных значений i= 1,2,…, n. При каждом очередном выполнении этого оператора к текущему значению y будет прибавляться очередное слагаемое. Как видно, в этом случае процесс вычислений будет носить циклический характер: оператор **y:= y+1/i** должен выполняться многократно, т.е. циклически, при различных значениях i.

Этот пример циклического вычислительного процесса является весьма типичным; его характерные особенности состоят в том, что

- число повторений цикла известно к началу его выполнения (в данном случае оно равно значению n, которое предполагается заданным к этому времени);
- управление циклом осуществляется с помощью переменной порядкового типа, которая в этом циклическом процессе принимает последовательные значения от

заданного начального до заданного конечного значений (в нашем случае – это целочисленная переменная i, принимающая последовательные значения от 1 до n).

Для компактного задания подобного рода вычислительных процессов и служит **оператор цикла с параметром**. Чаще всего используется следующий вид этого оператора В Паскале:

For V:= E1 to E2 do S,

где **for** (для), **to** (увеличиваясь к) и **do** (выполнять, делать) – служебные слова, **V** – переменная порядкового типа, называемая параметром цикла, **E1** и **E2** – выражения того же типа, что и параметр цикла, **S** – оператор, который и выполняется многократно в цикле, называемый телом цикла.

Заметим, что в Паскале после **do** должен стоять один оператор, если необходимо выполнить несколько действий, то они должны быть объединены в один составной оператор путем заключения в операторные скобки.

Этот оператор цикла Паскаля предусматривает присваивание параметру цикла V последовательных значений от начального значения, равного значению выражения E1, до конечного значения, равного значению выражения E2, т.е. при каждом повторении выполняется оператор присваивания **V:= succ(V)**, и выполнение оператора S при каждом значении параметра цикла V. При этом значения выражений E1 и E2 вычисляются один раз, при входе в оператор цикла, а значение параметра цикла V не должно изменяться в результате выполнения оператора S. Если заданное конечное значение меньше начального значения (что допустимо), то оператор S не выполняется ни разу.

В Паскале считается, что при нормальном завершении выполнения оператора цикла значение параметра цикла не определено.

С использованием **оператора цикла с параметром** алгоритм вычисления суммы первых n членов гармонического ряда может быть задан следующим образом:

Пример кода программы для суммирования первых n членов гармонического ряда

```
Readln(n);
Y:=                                                    0;
For i:= 1 to n do y:= y+1/i;
```

В некоторых случаях бывает удобно, чтобы параметр цикла Паскаля принимал последовательные, но не возрастающие, а убывающие значения. Для таких случаев в Паскале предусмотрен оператор цикла с параметром следующего вида:

For V:= E1 downto E2 do S,

где **downto** (уменьшаясь к) – служебное слово, а все остальные слова и выражения имеют прежний смысл. Изменение параметра цикла от большего значения к меньшему происходит при выполнении присваивания V:=pred(V). Заметим, что начальное значение может быть меньше конечного значения. В этом случае оператор S не выполнится ни разу. Значение параметра цикла по завершении выполнения такого цикла так же считается неопределенным.

Следует запомнить и то, что для обоих вариантов записи **цикла с параметром** справедливо: если начальное и конечное значения равны, то тело цикла (оператор S) выполнится один раз.

Заметим так же, что **параметр цикла** может и не использоваться в теле цикла, так что основное его назначение – это управление числом повторений цикла. Например, значение y= x n, где n>=0 – целое, можно вычислить по следующему алгоритму: предварительно положить y=1, а затем n раз домножить это значение на x:

Пример кода программы цикла Паскаля

```
Readln(n);
Readln(x);
Y:=                                                    1;
For i:= 1 to n do y:= y*x;
```

Как видно, здесь параметр цикла i служит лишь для того, чтобы тело цикла (оператор y:= y* x) выполнилось нужное число раз.

Арифметический оператор цикла Паскаля с произвольным шагом

Естественным усложнением простого арифметического цикла Паскаля, является цикл, в котором параметр цикла изменяется не на 1, а на произвольную величину – **шаг приращения**. При этом в процессе выполнения цикла шаг изменяется по заданному закону. Стандартные операторы для реализации такого цикла есть в Форте, в других языках их приходится организовывать из простейшего арифметического цикла.

3. Циклы с предусловием и постусловием.

Итерационные циклы отличаются от циклов с параметром тем, что в них заранее неизвестно число повторений.

Пусть мы отправляемся за грибами и возвращаемся домой, когда корзина наполнится. Все грибники делятся на 2 категории:

- Смотрят, есть ли место в корзине, а уже потом срывают грибы, если их можно поместить в корзину. (Правда, в жизни таких грибников встречать не приходилось)
- Сначала срывают грибы, а уже потом думают, как их положить в корзину.

Отсюда получаются два варианта реализации итерационных циклов: с предусловием и с постусловием.

В цикле с предусловием сначала проверяется условие, а потом делается шаг. Грибник придет с полной или почти полной корзиной. В цикле с постусловием – сначала шаг, а потом проверка. Как всякий нормальный грибник, этот принесет полную или слегка переполненную корзину.

Какой алгоритм выбрать? Это зависит от конкретной задачи.

Если, сделав шаг без проверки, можно свалиться в яму, то лучше проверка вначале (как слепой с палочкой). Ну, а если шаг без проверки вас не пугает, то можно отложить ее до завершения шага.

Нужно также проанализировать событие, которого мы ожидаем. Если оно может случиться до первого шага, то нужен цикл с предусловием. А если событие не может случиться до первого шага, то нужен цикл с постусловием.

Оператор цикла Паскаля с постусловием

Рассмотрим теперь математическую задачу. Пусть нам необходимо вычислить сумму первых членов гармонического ряда, удовлетворяющих условию 1/i>= e, где 0< e<1, а i=1,2,3…...Эту задачу можно решить по следующему алгоритму: положить предварительно y=0 и i=0, а затем в цикле увеличивать i на 1, к значению y добавлять очередное слагаемое 1/ i до тех пор, пока текущее значение 1/ i впервые окажется больше заданного значения 0< e<1.

Очевидно, что число повторений этого цикла заранее не известно. В подобного рода случаях мы можем лишь сформулировать условие, при выполнении которого процесс добавления к сумме очередного слагаемого должен завершиться.

Для задания таких вычислительных процессов и служит оператор цикла Паскаля с постусловием. Этот оператор имеет вид:

Repeat S1; S2;…; Si until B,

где **repeat** (повторять) и **until** (до) – служебные слова, через **Si** обозначен любой оператор Паскаля, а через **B** – логическое выражение.

При выполнении этого оператора цикла последовательность операторов, находящихся между словами repeat и until, выполнится один или более раз. Этот процесс завершается, когда после очередного выполнения заданной последовательности операторов логическое выражение B примет (впервые) значение true. Таким образом, с помощью логического выражения B задается условие завершения выполнения оператора цикла. Поскольку в данном случае проверка условия производится после выполнения последовательности операторов (тела цикла), этот оператор цикла и называется оператором цикла с постусловием.

С использованием этого вида оператора цикла Паскаля задача о суммировании первых членов гармонического ряда, удовлетворяющих заданному условию, может быть реализована следующим образом:

Пример кода оператора цикла Паскаля с постусловием

```
readln(e);
i:=0;
y:=0;
Repeat
   i:=i+1;
   y:=y+1/i;
Until 1/i<e;
```

Заметим, что оператор цикла с постусловием является более общим, чем оператор цикла с параметром — любой циклический процесс, задаваемый с помощью цикла с параметром можно представить в виде цикла с постусловием. Обратное утверждение неверно. Например, задача о суммировании первых n членов гармонического ряда, рассмотренная ранее, с оператором цикла с постусловием будет выглядеть так:

Пример кода оператора цикла Паскаля с постусловием

```
Readln(n);
i:=0;
y:=0;
Repeat
   i:=i+1;
   y:=y+1/i;
Until i>n;
```

Оператор цикла Паскаля с предусловием

В случае оператора цикла Паскаля с постусловием входящая в него последовательность операторов заведомо будет выполняться хотя бы один раз. Между тем довольно часто встречаются такие циклические процессы, когда число повторений цикла тоже неизвестно заранее, но при некоторых значениях исходных данных предусмотренные в цикле действия вообще не должны выполняться, и даже однократное выполнение этих действий может привести к неверным или неопределенным результатам.

Пусть, например, дано вещественное число M. Требуется найти наименьшее целое неотрицательное число k, при котором 3 k> M. Эту задачу можно решить по следующему алгоритму: предварительно положить y=1 и k=0; затем в цикле домножать значение y на 3 и увеличивать значение k на 1 до тех пор, пока текущее значение y впервые окажется больше значения M. На первый взгляд, здесь можно воспользоваться оператором цикла с постусловием:

Пример кода оператора цикла Паскаля с постусловием

```
y:=1; k:=0;
Repeat
  y:=y*3;
  k:=k+1;
Until y> M;
```

Однако нетрудно убедиться в том, что при M<1 будет получен неправильный результат k=1, тогда как должно быть получено k=0: в этом случае предварительно сформированное значение k=0 является окончательным результатом и действия, предусмотренные в цикле, выполняться не должны.

Для задания подобного рода вычислительных процессов, когда число повторений цикла заранее неизвестно и действия, предусмотренные в цикле, могут вообще не выполняться, и служит оператор цикла с предусловием. Этот оператор цикла имеет в Паскале следующий вид:

While B do S,

где **while** (пока), **do** (делать, выполнять) – служебные слова, **B** – логическое выражение, **S** – оператор. Здесь оператор S выполняется ноль или более раз, но перед каждым очередным его выполнением вычисляется значение выражения B, и оператор S выполняется только в том случае, когда значение выражения B true. Выполнение оператора цикла завершается, когда выражение B впервые принимает значение false. Если это значение выражение B принимает при первом же его вычислении, то оператор S не выполнится ни разу.

В рассматриваемой нами задаче правильное значение k при любом значении M может быть получено следующим образом:

Пример кода оператора цикла Паскаля с предусловием

```
y:=1; k:=0;
While y<=M do
Begin
  y:=y*3;
  k:=k+1;
End;
```

Оператор цикла Паскаля с предусловием можно считать наиболее универсальным – с использованием таких операторов можно задать и циклические процессы, определяемые операторами цикла с параметром и постусловием.

Отметим **отличия и особенности хорошего стиля работы** с рассмотренными циклическими операторами.

Цикл с предусловием While (пока условие истинно)	**Цикл с постусловием Repeat (до истинности условия)**
1. До начала цикла должны быть сделаны начальные установки переменных, управляющих условием цикла, для корректного входа в цикл	
2. В теле цикла должны присутствовать операторы, изменяющие переменные условия так, чтобы цикл через некоторое число итераций завершился	
3. Цикл работает пока условие истинно (пока True)	3. Цикл работает пока условие ложно (пока False)
4. Цикл завершается, когда условие становится ложным (до False)	4. Цикл завершается, когда условие становится истинным (до True)
5. Цикл может не выполниться ни разу, если исходное значение условия при входе в цикл False	5. Цикл обязательно выполнится как минимум один раз
6. Если в теле цикла требуется выполнить более одного оператора, то необходимо использовать составной оператор	6. Независимо от количества операторов в теле цикла, использование составного оператора не требуется
Цикл со счетчиком (с параметром) For	
• Начальная установка переменной счетчика цикла до заголовка не требуется	
• Изменение в теле цикла значений переменных, стоящих в заголовке не допускается	
• Количество итераций цикла неизменно и точно определяется значениями нижней и верхней границ и шага приращения	
• Нормальный ход работы цикла может быть нарушен оператором goto или процедурами Break и Continue	
• Цикл может не выполниться ни разу, если шаг цикла будет изменять значение счетчика от нижней границы в направлении, противоположном верхней границе	

Оператор, который выполняется в цикле, сам может быть циклом. Это относится ко всем видам циклов. В результате мы получаем **вложенные циклы**. Механизм работы вложенных циклов удобнее всего рассмотреть на примере вложенных циклов с параметром. Пусть нам нужно описать работу электронных часов, начиная с момента

времени 0 часов, 0 минут, 0 секунд. Значение минут станет равным 1 только после того, как секунды «пробегут» все последовательные значения от 0 до 59. Часы изменят свое значение на 1 только после того, как минуты «пробегут» все последовательные значения от 0 до 59. Таким образом, вывод всех значений времени от начала суток до конца суток может быть представлен следующим фрагментом программы:

```
For h:=0 to 23 do
For m:=0 to 59 do
For s:=0 to 59 do
Writeln(h,':',m,':',s);
```

Для удобства реализации циклических структур на Паскале в последних версиях языка введены операторы **break** и **continue**, применяемые внутри циклов. Они расширяют возможности использования циклов и улучшают структуру программы.

В процессе выполнения тела цикла до полного завершения цикла могут возникнуть дополнительные условия, требующие завершения цикла. В этом случае цикл может быть прекращен оператором break.

В ходе выполнения цикла может возникнуть условие, при котором необходимо пропустить все или некоторые действия, предусмотренные в цикле, не прекращая работу цикла совсем. Для этого используется оператор continue, который передает управление в ту точку программы, где проверяется условие продолжения или прекращения цикла.

4. Вложенные циклы.

В теле любого оператора цикла могут находиться другие операторы цикла. При этом цикл, содержащий в себе другой, называют *внешним*, а цикл, находящийся в теле первого — *внутренним (вложенным)*. Правила организации внешнего и внутреннего циклов такие же, как и для простого цикла.

Обратите внимание — при программировании вложенных циклов необходимо соблюдать следующее дополнительное условие: *все операторы внутреннего цикла должны полностью располагаться в теле внешнего цикла*.

Рассмотрим задачу вывода на экран таблицы умножения, решение которой предполагает применение вложенных циклов. С использованием цикла forсоответствующий фрагмент программы имеет вид:

```
    var i, j :byte;
  begin
    for i:= 1 to 10 do
      for j:=1 to 10 do
        writeln (i,'*',j,'=',i*j);
  end;
```

Проанализируем действие данной программы. В разделе описания переменных описываются переменные I, J целого типа byte, выполняющие функции управляющих переменных циклов for.

Выполнение программы начинается с внешнего цикла. При первом обращении к оператору внешнего цикла for вычисляются значения начального (1) и конечного (10) параметров цикла и управляющей переменной I присваивается начальное значение 1.

Затем циклически выполняется следующее:

1. Проверяется условие I<=10.

2. Если оно соблюдается, то выполняется оператор в теле цикла, т. е. выполняется внутренний цикл.

• При первом обращении к оператору внутреннего цикла for вычисляются значения начального (1) и конечного (10) параметров цикла и управляющей переменнойJ присваивается начальное значение 1.

Затем циклически выполняется следующее:

• Проверяется условие J<=10.

• Если оно удовлетворяется, то выполняется оператор в теле цикла, т. е. операторWriteln(I,' * ',J,' = ',I*J), выводящий на экран строку таблицы умножения в соответствии с текущими значениями переменных I и J.

• Затем значение управляющей внутреннего цикла J увеличивается на единицу и оператор внутреннего цикла for проверяет условие J<=10. Если условие соблюдается, то выполняется тело внутреннего цикла при неизменном значении управляющей переменной внешнего цикла до тех пор, пока выполняется условиеJ<=10.

Если условие J<=10 не удовлетворяется, т. е. как только J станет больше 10, оператор тела цикла не выполняется, внутренний цикл завершается и управление в программе передается за пределы оператора for внутреннего цикла, т. е. на операторfor внешнего цикла.

3. Значение параметра цикла I увеличивается на единицу, и проверяется условиеI<=10. Если условие I<=10 не соблюдается, т. е. как только I станет больше 10, оператор тела цикла не выполняется, внешний цикл завершается и управление в программе передается за пределы оператора for внешнего цикла, т. е. на операторend, и программа завершает работу.

Таким образом, на примере печати таблицы умножения и на блок-схеме наглядно показано, что при вложении циклов внутренний цикл выполняется *полностью* от начального до конечного значения параметра, при *неизменном* значении параметра внешнего цикла. Затем значение параметра *внешнего* цикла изменяется на *единицу*, и опять от начала и до конца выполняется *вложенный* цикл. И так до тех пор, пока значение параметра внешнего цикла не станет больше конечного значения, определенного в операторе for внешнего цикла.

1. 6 Лекция № 1 (2 часа).

Тема: «Подпрограммы»

1.6.1 Вопросы лекции:

1. Понятие подпрограммы. Процедуры и функции, их сущность, назначение, различие.
2. Организация процедур, стандартные процедуры.
3. Процедуры, определенные пользователем: синтаксис, передача аргументов. Формальные и фактические параметры.

1.6.2 Краткое содержание вопросов:

1. Понятие подпрограммы. Процедуры и функции, их сущность, назначение, различие.

Подпрограммы нужны для того, чтобы упростить структуру программы и облегчить ее отладку. В виде подпрограмм оформляются логические законченные части программы.

Подпрограмма - это фрагмент кода, к которому можно обратиться по имени. Она описывается один раз, а вызываться может столько раз, сколько необходимо. Одна и та же подпрограмма может обрабатывать различные данные, переданные ей в качестве аргументов.

В Паскале два вида подпрограмм: процедуры и функции. Они имеют незначительные отличия в синтаксисе и правилах вызова. Процедуры и функции описываются в соответствующих разделах описания, до начала блока исполняемых операторов.

Само по себе описание не приводит к выполнению подпрограммы. Для того чтобы подпрограмма выполнилась, ее надо вызвать. Вызов записывается в том месте программы, где требуется получить результаты работы подпрограммы. Подпрограмма вызывается по имени, за которым следует список аргументов в круглых скобках. Если аргументов нет,

скобки не нужны. Список аргументов при вызове как бы накладывается на список параметров, поэтому они должны попарно соответствовать друг другу.
Процедура вызывается с помощью отдельного оператора, а функция - в правой части оператора присваивания, например:

```
inc(i); writeln(a, b, c);   { вызовы процедур }
y := sin(x) + 1;            { вызов функции }
```

Внутри подпрограмм можно описывать другие подпрограммы. Они доступны только из той подпрограммы, в которой они описаны. Рассмотрим правила описания подпрограмм.
Процедуры
Структура процедуры аналогична структуре основной программы:

```
procedure имя [(список параметров)];      { заголовок }
разделы описаний
begin
раздел операторов
end;
```

2. Организация процедур, стандартные процедуры.

Часто в задаче требуется повторить определенную последовательность операторов в разных частях программы. Для того, чтобы описывать эту последовательность один раз, а применять многократно, в языках программирования применяются подпрограммы. Подпрограмма - автономная часть программы, выполняющая определенный алгоритм и допускающая обращение к ней из различных частей общей программы. Использование подпрограмм позволяет реализовать один из самых современных методов программирования - структурное программирование.
В языке Паскаль существует два вида подпрограмм: процедура (PROCEDURE) и функция (FUNCTION). Процедуры и функции в Паскале объявляются в разделе описания за разделом переменных. В данном уроке приведены примеры и задачи использования процедуры и функций, а также использование рекурсии в языке Паскаль.

Процедуры

Процедуры используются в случаях, когда в подпрограмме необходимо получить несколько результатов. В языке Паскаль существует два вида процедур: процедуры с параметрами и без параметров. Обращение к процедуре осуществляется по имени процедуры, за которым могут быть указаны фактические параметры. Все формальные параметры являются локальными для данной процедуры и глобальными для каждой процедуры в ней. При вызове процедуры устанавливается взаимно однозначное соответствие между фактическими и формальными параметрами, затем управление передается процедуре. После выполнения процедуры управление передается следующему, после вызова процедуры, оператору вызывающей программы.
Пример. Процедура без параметров, которая печатает строку из 60 звездочек.

```
procedure pr;
  var i : integer ;
begin
  for i :=1 to 60 do write (‘ * '); writeln;
end.
```

Пример 2. Процедура с параметрами. Даны 3 различных массива целых чисел (размер каждого не превышает 15). В каждом массиве найти сумму элементов и среднеарифметическое значение.

```
program proc;
var i , n , sum: integer;
   sr : real;
   procedure work (r:integer; var s:integer; var s1:real); {процедура work}
var mas : array [1..15] of integer ; { объявление массива mas}
   j : integer;
```

```
begin
  s:=0;
  for j:=1 to r do begin {ввод элементов массива mas}
  write(' Vvedite element - ', j,': ') ;
  read (mas[j]);
  s:=s+mas [j];
end;
 s1:=s/r;
end;
begin { главная программа}
  for i:=1 to 3 do begin
  write ('Vvedite razmer ',i, ' masiva: ');
  readln(n);
  work (n, sum, sr); {вызов процедуры work}
  writeln ('Summa elementov = ',sum);
  writeln ('Srednearifmeticheskoe = ',sr:4:1);
end;
  readln;
end.
```

Результат работы программы: В программе трижды вызывается процедура work, в которой формальные переменные r, s, s1 заменяются фактическими n, sum, sr. Процедура выполняет ввод элементов массива, вычисляет сумму и среднее значение. Переменные s и s1 возвращаются в главную программу, поэтому перед их описанием ставится служебное слово var. Локальные параметры mas, j действуют только в процедуре. Глобальные - i, n, sum, sr доступны во всей программе.

3. Процедуры, определенные пользователем: синтаксис, передача аргументов. Формальные и фактические параметры.

Набор встроенных функций в языке Паскаль достаточно широк (ABS, SQR, TRUNC и т.д.). Если в программу включается новая, нестандартная функция, то ее необходимо описать в тексте программы, после чего можно обращаться к ней из программы. Обращение к функции осуществляется в правой части оператора присваивания, с указанием имени функции и фактических параметров. Функция может иметь собственные локальные константы, типы, переменные, процедуры и функции. Описание функций в Паскале аналогично описанию процедур. Отличительные особенности функций: - результат выполнения - одно значение, которое присваивается имени функции и передается в основную программу;

- имя функции может входить в выражение как операнд.

Пример. Написать подпрограмму-функцию степени ax, где a, x – любые числа. Воспользуемся формулой: ax = ex ln a

```
program p2;
var f, b, s, t, c, d : real; { глобальные параметры}
    function stp (a, x : real) : real;
var y : real; { локальные параметры}
begin
   y := exp (x * ln ( a)) ;
   stp:= y;{присвоение имени функции результата вычислений подпрограммы}
end; { описание функции закончено }
begin {начало основной программы }
   d:= stp (2.4, 5); {вычисление степеней разных чисел и переменных }
   writeln (d, stp (5,3.5));
   read (f, b, s, t); c := stp (f, s)+stp (b, t);
```

```
  writeln (c);
end.
```

Процедуры и функции в Паскале могут вызывать сами себя, т.е. обладать свойством рекурсивности. Рекурсивная функция обязательно должна содержать в себе условие окончания рекурсивности, чтобы не вызвать зацикливания программы. При каждом рекурсивном вызове создается новое множество локальных переменных. То есть переменные, расположенные вне вызываемой функции, не изменяются.

Пример. Составить рекурсивную функцию, вычисляющую факториал числа n следующим образом: n! = 1 , если n= 1 и n!= (n -1)! · n , если n > 1

```
function f ( n : integer): integer;
begin
if n = 1 then f := 1 else f := n * f ( n -1 ); { функция f вызывает саму себя}
end;
```

Правила корректной работы с процедурными типами.

- Подпрограмма, присваиваемая процедурной переменной должна быть транслирована в режиме “дальнего вызова”.
- Подпрограмма, присваиваемая процедурной переменной, не должна быть стандартной процедурой или функцией.

- Подпрограмма, присваиваемая процедурной переменной, не может быть вложенной в другие подпрограммы.
- Подпрограмма, присваиваемая процедурной переменной, не может быть подпрограммой специального вида (interrupt или inline).

Процедурная переменная занимает в памяти 4 байта (2 слова). В первом хранится смещение, во втором - сегмент (т.е. указатель на код подпрограммы).

Параметры, записываемые в обращении к подпрограммам, называются фактическими; параметры, указанные в описании подпрограмм - формальными. Фактические параметры должны соответствовать формальным по количеству, порядку следования и типу. Параметры, объявленные в основной (главной) программе, действуют в любой подпрограмме и называются глобальными. Параметры, объявленные в подпрограмме, действуют только в этой подпрограмме и называются локальными.

1. 7 Лекция № 7 (2 часа).

Тема: «Структурированные типы данных»

1.7.1 Вопросы лекции:

1. Строковый тип.
2. Массивы.
3. Множественный тип данных.
4. Статические и динамические типы данных.

1.7.2 Краткое содержание вопросов:

1. Строковый тип.

В математике принято классифицировать переменные в соответствии с некоторыми важными характеристиками. Производится строгое разграничение между вещественными, комплексными и логическими переменными, между переменными, представляющими отдельные значения и множество значений, и так далее.

При обработке данных на ЭВМ такая классификация еще более важна. В любом алгоритмическом языке каждая константа, переменная, выражение или функция бывают *определенного типа*.

Любой тип данных определяет множество значений, которые может принимать переменная или выражение, а также возвращать операция или функция. Каждая операция

или функция требует аргументов также фиксированного типа и выдает результат фиксированного типа.

Тип определяет:

— возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;

— внутреннюю форму представления данных в ЭВМ;

— операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

Алгоритмический язык

Тип	Описание	Диапазон значений	Использование
Integer	целые числа	от-32 768 до 32 767	Var%
Long	целые числа	от-2 147 483 648 до 2 147 483 647	Var&
Single	действительные числа	от $-3.4*10^{38}$ до $-1.4*10^{-45}$ 0 и от $1.4*10^{-45}$ до $3.4*10^{38}$	Var!
Double	действительные числа	от $-1.8*10^{308}$ до $-4.9*10^{-324}$ 0 и от $4.9*10^{-324}$ до $1.8*10^{308}$	Var #
String	набор символов	от 0 до приблизительно 2 миллиардов символов	Var$

Бейсик

Тип	Описание	Диапазон значений
byte	короткое целое без знака	от 0 до 255
shortint	короткое целое со знаком	от-127 до 127
word	целое без знака	от 0 до 65536
integer	целое со знаком	от -32 768 до 32 767
longint	длинное целое	от -2 147 483 648 до 2 147 483 647
real	действительное	от $-3.4*10^{38}$ до $-1.4*10^{-45}$ 0 и от $1.4*10^{-45}$ до $3.4*10^{38}$
double	двойное действительное	от $-1.8*10^{308}$ до $-4.9*10^{-324}$ 0 и от $4.9*10^{-324}$ до $1.8*10^{308}$
char	символьное	один символ
string	строковое	до 255 символов
boolean	логическое	true (истина) и false (ложь)

Паскаль

2. Массивы.

Массивы **–** переменные с индексами описывают структуры, состоящие из ограниченного множества компонент, упорядоченных в соответствии со значениями индексов. Число индексов определяет размерность (одномерные, двумерные и т.д.).Индекс обеспечивает прямой доступ к любому элементу массива. Элементами массива м.б. как простые так и структурированные данные. Например, м.б. массив массивов.

Строки – упорядоченные, ограниченные последовательности символов некоторого алфавита.

Записи – структура данных, состоящая из фиксированного числа компонент, называемых полями, каждая из которых может иметь свой тип. Записи позволяют в удобной форме представлять ведомости, таблицы, картотеки, каталоги и лр. данные.

Списки – цепочки записей. Основные операции со списками: просмотр записей, включить новую запись и исключить запись из списка. Списки позволяют создавать объекты со сложной меняющейся структурой.

Таблицы – набор записей, с каждой из которых связано имя, называемое ключом. Поиск нужной записи в таблице производится по ее ключу. Основные операции с таблицами: найти запись, включить новую запись и исключить запись из таблицы.

Очереди – структуры данных организованные по принципу «первым пришел – первым ушел». Это динамические структуры, число элементов которых может меняться в процессе обработки. Обработка элементов очереди ведется последовательно один за другим. Добавление новых элементов производится в конец очереди. Основные операции с элементами очереди: чтение, обработка, запись в очередь, удаление из очереди

Стеки – структуры данных организованные по принципу «последним пришел – первым ушел». Примеры: стопка книг, пистолетная обойма, магазин автомата, очередь в магазине. Поэтому эта память называется магазинной. Ссылки – адреса поля памяти, содержимым которого являются другого поля памяти.

Графы **–** математические модели системы связей между объектами. Граф состоит из вершин (узлов) и ребер (ветвей) соединяющих узлы расположенные на различных уровнях.

Деревья – связной граф, в котором нет циклов. При решении многих прикладных задач бывает удобно представлять наборы объектов в виде деревьев. Например, представление двоичных кодов.

3. Множественный тип данных

Одним из фундаментальных разделов математики является теория множеств. Некоторые моменты математического аппарата этой теории реализованы в Паскале через множественный тип данных (множества).

Множеством называется совокупность однотипных элементов, рассматриваемых как единое целое. В Паскале могут быть только конечные множества. В Турбо Паскале множество может содержать от 0 до 255 элементов.

В отличие от элементов массива элементы множества не пронумерованы, не упорядочены. Каждый отдельный элемент множества не идентифицируется, и с ним нельзя выполнить какие-либо действия. Действия могут выполняться только над множеством в целом.

Тип элементов множества называется базовым типом. Базовый тип может быть любым скалярным, за исключением типа Real.

Конструктор множества. Конкретные значения множества задаются с помощью конструктора множества, представляющего собой список элементов, заключенный в квадратные скобки. Сами элементы могут быть либо константами, либо выражениями базового типа. Вот несколько примеров задания множеств с помощью конструктора:

[3,4,7,9,12] — множество из пяти целых чисел;

[1.. 100] — множество целых чисел от 1 до 100;

['a','b','c'] — множество, содержащее три литеры a, Ь, c;

Символы [] обозначают пустое множество, т.е. множество, не содержащее никаких элементов.

Не имеет значения порядок записи элементов множества внутри конструктора. Например, [1,2,3] и [3,2,1] эквивалентные множества.

Каждый элемент в множестве учитывается только один раз. Поэтому множество [1,2,3,4,2,3,4,5] эквивалентно [1.. 5].

Переменные множественного типа описываются так:

Var <идентификатор>: Set Of <базовый тип>

Например:

Var A,D: Set Of Byte; B: Set Of ' a' . . ' z; C: Set Of Boolean;

Нельзя вводить значения во множественную переменную оператором ввода и выводить оператором вывода. Множественная переменная может получить конкретное значение только в результате выполнения оператора присваивания следующего формата:

<множественная переменная>: <множественное выражение>

Например:

A:=[50,100,150,200]; B:=['m', 'n','k']; C:=[True,False] ; D:=A;

Кроме того, выражения могут включать в себя операции над множествами.

Операции над множествами. В Паскале реализованы основные операции теории множеств. Это объединение, пересечение, разность множеств. Во всех таких операциях операнды и результаты есть множественные величины одинакового базового типа.

Объединение множеств. Объединением двух множеств A и B называется множество, состоящее из всех элементов, принадлежащих хотя бы одному из множеств A или B. Знак операции объединения в Паскале +.

Пересечение множеств. Пересечением двух множеств A и B называется множество, состоящее из всех элементов принадлежащих, одновременно множеству A и множеству B

Разность множеств. Разностью двух множеств A и B называется множество, состоящее из элементов множества A, не принадлежащих множеству B.

Очевидно, что операции объединения и пересечения — перестановочны, а разность множеств — не перестановочная операция.

Операции отношения. Множества можно сравнивать между собой, т.е. для них определены операции отношения. Результатом отношения, как известно, является логическая величина true или false. Для множеств применимы все операции отношения, за исключением > и <. В таблице описаны операции отношения над множествами. Предполагается, что множества A и B содержат элементы одного типа.

Отношение	Результат	
	True	*False*
A = *B*	Множества *A* и *B* совпадают	В противном случае
A <> *B*	Множества *A* и *B* не совпадают	В противном случае
A <= *B*	Все элементы *A* принадлежат *B*	В противном случае
A >= *B*	Все элементы *B* принадлежат *A*	В противном случае

Операция вхождения. Это операция, устанавливающая связь между множеством и скалярной величиной, тип которой совпадает с базовым типом множества. Если x — такая скалярная величина, а M — множество, то операция вхождения записывается так:

X In M

Результат — логическая величина true, если значение x входит в множество M, и false — в противном случае. Для описанного выше множества

4. Статические и динамические типы данных.

В любой вычислительной системе память относится к таким ресурсам, которых всегда не хватает. Управление памятью - одна из главных забот программиста, так как для него очень важно создавать программы, эффективно использующие память, ведь во время

выполнения программы память необходима для следующих элементов программ и данных:

- сама программа пользователя;
- системные программы времени выполнения, которые осуществляют вспомогательные действия при работе программы пользователя;
- определяемые пользователем структуры данных и константы;
- точки возврата для программ;
- временная память для хранения промежуточных результатов при вычислении выражений;
- временная память при передаче параметров;
- буферы ввода-вывода, используемые как временные области памяти, в которых хранятся данные между моментом их реальной физической передачи с внешнего устройства или на него и моментом инициализации в программе операции ввода или вывода;
- различные системные данные (информация о статусе устройств ввода-вывода и др.).

Из этого перечня видно, что управление памятью касается широкого класса объектов.

До сих пор мы использовали простейший способ распределения памяти - статическое распределение, т. е. распределение памяти при трансляции программы. То есть когда Вы объявляете переменную `Var  A : Array[1..100] of integer;` вы даете указание компилятору выделить память размера, соответствующего заданному типу, т.е. 2*100=200 байт. Если в программе на нужный программный объект мы ссылаемся по имени A[3], то машинный код содержит ссылку на номер ячейки памяти (адрес байта), начиная с которой размещается этот объект.

Адреса задаются двумя 16-тиразрядными словами (тип word) - *сегментом* и *смещением*. Каждое из них способно адресовать 216=65536 байт (64 Кбайт). Для адресации пространства размером в 1 Мбайт требуется 20 разрядов. Сегменты адресуют память с точностью до *параграфа* - фрагмента памяти в 16 байт. Смещение адресует память с точностью до байта, но в пределах сегмента. Реальный (абсолютный) адрес складывается из значения сегмента, сдвинутого на 4 разряда влево (умноженного на 16), и смещения.

Программа на Паскале получает один сегмент данных, поэтому область памяти, в которой могут быть размещены статические переменные Вашей программы, ограничена 64 Кбайтами. При попытке исполнить программу, требующую большего размера памяти, будет выдана ошибка:

Error 49: Data Segment too large{Слишком большой сегмент данных}

При динамическом распределении памяти Вы можете запросить блоки размером до одного сегмента (64 Кбайт) каждый, причем их можно требовать в пределах основной памяти (640 Кбайт) в реальном режиме и без программных ограничений в защищенном.

Бывают такие данные, размер которых выясняется только при выполнении программ. Кроме того, иногда мы не знаем, будет существовать некоторый объект или нет.

Например, в программе для обработки текстов (такие программы называются "текстовыми процессорами") требуется организовать поиск слов, определенных пользователем. Естественно, что определить заранее длину слова, которое будет задано, невозможно. Часто программный объект, причем значительного размера, бывает нужен на непродолжительное время. Использование статических программных объектов в таких случаях очень неэффективно, поскольку программа должна быть рассчитана на максимальные размеры объектов. Область памяти, в которой могут быть размещены статические переменные, ограничена, и, рассчитывая на максимальный размер переменных, мы ограничиваем их количество. В Паскале, кроме статических, предусмотрены динамические объекты. Память под них отводится во время исполнения программы, а когда программный объект можно удалить, память освобождается.

И статические, и динамические переменные вызываются по их адресам. Без адреса не получить доступ к нужной ячейке памяти, но, используя статические переменные, непосредственно адрес Вы не указываете, а обращаетесь к переменной по имени. Компилятор размещает переменные в памяти и подставляет нужные адреса в коды команд.
Адресация динамических переменных происходит через указатели. В Паскале можно определить переменные, которые имеют тип указатель, их значения определяют адрес объекта. Для работы с динамическими переменными в программе должны быть предусмотрены:

- выделение памяти под динамическую переменную;
- присвоение указателю на динамическую переменную адреса выделенной памяти (инициализация указателя);
- освобождение памяти после использования динамической переменной.

Программист сам должен резервировать место под переменную, определять значения указателей, освобождать память - удалять динамические переменные. Для использования динамической переменной где-то в статике должен быть указатель на нее. Компилятор предусматривает место под указатель, об инициализации указателя должен заботиться программист.
Вместо любой статической переменной можно использовать динамическую, но без реальной необходимости этого делать не стоит. Переменные простых типов нет смысла размещать в динамической области, поскольку они занимают меньше места, чем указатель на них. Например, указатель на целое занимает 4 байта, само целое - 2 байта. Кроме того, при динамическом распределении памяти удлиняется текст программы, снижаются наглядность и быстродействие. Это объясняется тем, что, во-первых, нужно во время исполнения программы определять значения указателей, а во-вторых, усложняется доступ к значению переменной.
Указатели могут ссылаться на любой тип данных, кроме файлового.
Обратите внимание, что указатель является обычной статической переменной, а переменная, на которую он указывает - динамической.
Использование имени указателя в программе означает обращение к адресу ячейки памяти, на которую он указывает. Чтобы обратиться к содержимому ячейки, на которую указывает указатель, требуется после его идентификатора поставить символ ^. Эта операция еще называется разыменованием.

1. 8 Лекция № 8 (2 часа).

Тема: «Объектно-ориентированное программирование»

1.8.1 Вопросы лекции:

1.Базовые понятия ООП: объект, его свойства и методы, класс, интерфейс.
2.Основные принципы ООП: инкапсуляция, наследование, полиморфизм.

1.8.2 Краткое содержание вопросов:

1. Базовые понятия ООП: объект, его свойства и методы, класс, интерфейс. Концепция объектно-ориентированного программирования (ООП) появилась более сорока лет назад, как развитие идей процедурного программирования. Идеология процедурного программирования, на мой взгляд, ничего особенного собой не представляет: все программы представлены набором процедур и функций, в то время как сами процедуры и функции – это последовательности операторов, выполняя которые компьютер модифицирует значения переменных в памяти. Основная программа в процедурном программировании также является процедурой (функцией), в теле которой могут быть вызовы других процедур и функций – подпрограмм. Суть процедурного программирования проста: данные отдельно, поведение отдельно. То из чего состоит процедурный язык программирования (какие конструкции в него входят), я постарался

собрать в отдельном разделе. Разделение кода на подпрограммы, во-первых, позволяет выделить повторно используемые фрагменты кода, а во-вторых, делает код программы структурированным.

Идеология объектно-ориентированного программирования, как следует из самого названия, строится вокруг понятия объект. Объект объединяет в себе и данные и поведение. Объект – это любая сущность, с которой имеет дело программа, а именно: объекты предметной области, моделируемые программой; ресурсы операционной системы; сетевые протоколы и многое другое. По сути, объект – это та же структура (составной тип), но дополненная процедурами и функциями, управляющими элементами этой структуры. К примеру, для работы с файлом в процедурном языке программирования отдельно была бы создана переменная для хранения имени файла и отдельно – для хранения его дескриптора (уникальный идентификатор ресурса в операционной системе), а также ряд процедур работы с файлом: открыть файл, прочитать данные из файла и закрыть файл. Все бы эти процедуры, помимо обычных параметров и переменных для хранения результата, обязаны были бы принимать тот самый дескриптор, чтобы понять, о каком именно файле идет речь. В объектно-ориентированном языке для этих же целей был бы описан объект-файл, который также бы хранил внутри себя имя и дескриптор и предоставлял бы пользователю процедуры для открытия, чтения и закрытия себя самого (файла, ассоциированного с конкретным объектом). Разница была бы в том, что дескриптор был бы скрыт от остальной части программы, создавался бы в коде процедуры открытия файла и использовался бы неявно только самим объектом. Таким образом, пользователю объекта (программному коду внешней по отношению к объекту программы) не нужно было бы передавать дескриптор каждый раз в параметрах процедур. Объект – это комплект данных и методов работы с этими данными, часть из которых может быть скрыта от окружающего его мира, к которой и относятся детали реализации. Более подробно о терминологии объектно-ориентированного программирования будет рассказано далее.

Объектом в объектно-ориентированном языке программирования является практически все, за исключением операторов: и элементарные типы являются объектами, и описание ошибки является объектом и, наконец, основная программа также является объектом. Осталось понять, что такое объект с точки зрения самой программы, как он создается и используется. Вторым основополагающим понятием ООП является класс. Класс – это тот самый новый в сравнении с процедурным программированием тип данных, экземпляры которого и называются объектами. Класс, как уже было сказано, похож на составной тип данных или структуру, но дополненный процедурами и функциями (методами) для работы со своими данными. Теперь самое время описать основные термины объектно-ориентированного программирования.

Терминология объектно-ориентированного программирования

Перед тем, как перейти к описанию преимуществ, которые дает ООП разработчикам программного обеспечения в процессе проектирования, кодирования и тестирования программных продуктов необходимо познакомиться с наиболее часто встречающимися терминами в этом области.

Класс – тип данных, описывающий структуру и поведение объектов.

Объект – экземпляр класса.

Поле – элемент данных класса: переменная элементарного типа, структура или другой класс, являющийся частью класса.

Состояние объекта – набор текущих значений полей объекта.

Метод – процедура или функция, выполняющаяся в контексте объекта, для которого она вызывается. Методы могут изменять состояние текущего объекта или состояния объектов, передаваемых им в качества параметров.

Свойство – специальный вид методов, предназначенный для модификации отдельных полей объекта. Имена свойств обычно совпадают с именами соответствующих полей.

Внешне работа со свойствами выглядит точно так же, как работа с полями структуры или класса, но на самом деле перед тем, как вернуть или присвоить новое значение полю может быть выполнен программный код, осуществляющий разного рода проверки, к примеру, проверку на допустимость нового значения.

Член класса – поля, методы и свойства класса.

Модификатор доступа – дополнительная характеристика членов класса, определяющая, имеется ли к ним доступ из внешней программы, или же они используются исключительно в границах класса и скрыты от окружающего мира. Модификаторы доступа разделяют все элементы класса на детали реализации и открытый или частично открытый интерфейс.

Конструктор – специальный метод, выполняемый сразу же после создания экземпляра класса. Конструктор инициализирует поля объекта – приводит объект в начальное состояние. Конструкторы могут быть как с параметрами, так и без. Конструктор без параметров называют конструктором по умолчанию, который может быть только один. Имя метода конструктора, чаще всего, совпадает с именем самого класса.

Деструктор – специальный метод, вызываемый средой исполнения программы в момент, когда объект удаляется из оперативной памяти. Деструктор используется в тех случаях, когда в состав класса входят ресурсы, требующие явного освобождения (файлы, соединения с базами данных, сетевые соединения и т.п.)

Интерфейс – набор методов и свойств объекта, находящихся в открытом доступе и призванных решать определенный круг задач, к примеру, интерфейс формирования графического представления объекта на экране или интерфейс сохранения состояния объекта в файле или базе данных.

Статический член – любой элемент класса, который может быть использован без создания соответствующего объекта. К примеру, если метод класса не использует ни одного поля, а работает исключительно с переданными ему параметрами, то ничто не мешает его использовать в контексте всего класса, не создавая отдельных его экземпляров. Константы в контексте класса обычно всегда являются статическими его членами.

2. Основные принципы ООП: инкапсуляция, наследование, полиморфизм.

Инкапсуляция обозначает сокрытие деталей реализации классов средствами награждения отдельных его членов соответствующими модификаторами доступа. Таким образом, вся функциональность объекта, нацеленная на взаимодействие с другими объектами программы группируется в открытый интерфейс, а детали тщательно скрываются внутри, что избавляет основной код бизнес-логики информационной системы от ненужных ему вещей. Инкапсуляция повышает надежность работы программного кода, поскольку гарантирует, что определенные данные не могут быть изменены за пределами содержащего их класса.

Наследование. Краеугольный камень ООП. В объектно-ориентированном программировании есть возможность наследовать структуру и поведение класса от другого класса. Класс, от которого наследуют, называется базовым или суперклассом, а класс, который получается вследствие наследования – производным или просто потомком. Любой класс может выступать как в роли суперкласса, так и в роли потомка. Связи наследования классов образуют иерархию классов. Множественным наследованием называют определение производного класса сразу от нескольких суперклассов. Не все объектно-ориентированные языки программирования поддерживают множественное наследование. Наследование – это эффективный способ выделения многократно используемых фрагментов кода, но у него есть и минусы, о которых будет рассказано далее.

Абстрагирование. Возможность объединять классы в отдельные группы, выделяя общие, значимые для них всех характеристики (общие поля и общее поведение). Собственно, абстрагирование и есть следствие наследования: базовые классы не всегда

имеют свою проекцию на объекты реального мира, а создаются исключительно с целью выделить общие черты целой группы объектов. К примеру, объект мебель – это базовое понятие для стола, стула и дивана, всех их объединяет то, что это движимое имущество, часть интерьера помещений, и они могут быть выполнены для дома или офиса, а также относиться к “эконом” или “премиум” классу. В ООП есть для этого отдельное понятие абстрактный класс – класс, объекты которого создавать запрещено, но можно использовать в качестве базового класса. Наследование и абстрагирование позволяют описывать структуры данных программы и связи между ними точно так же, как выглядят соответствующие им объекты в рассматриваемой модели предметной области.

Пример диаграммы классов, построенной путем абстрагирования, в ходе анализа видов существующих транспортных средств приведен на следующем рисунке. На верхних уровнях *иерархии наследования* находятся абстрактные классы, объединяющие транспортные средства по наиболее значимым характеристикам.

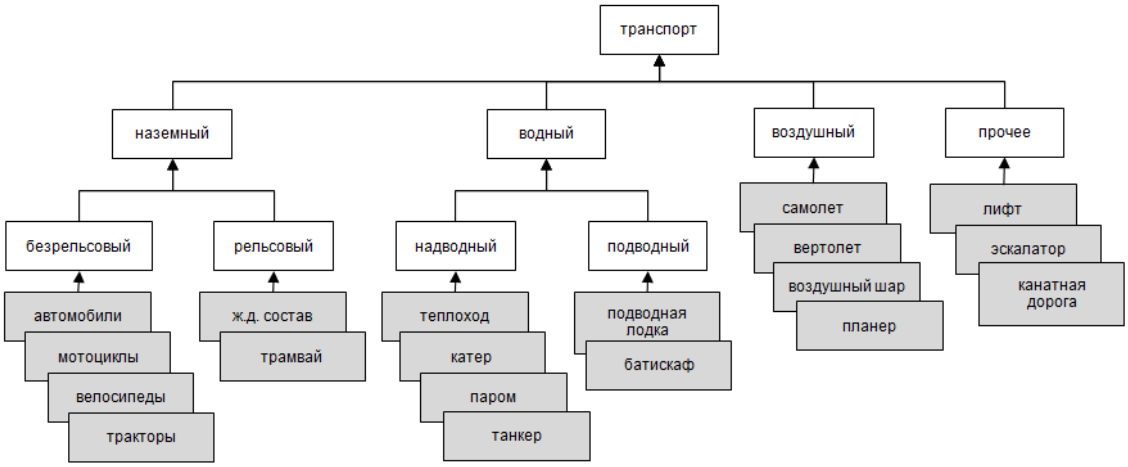


Диаграмма классов или иерархия наследования "Транспортные средства". Белые квадраты обозначают абстрактные классы.

Полиморфизм. Еще одно свойство, которое является следствием наследования. Дело в том, что объектно-ориентированные языки программирования позволяют работать с набором объектов из одной иерархии точно так же, как если бы все они были объектами их базового класса. Если вернуться к примеру про мебель, то можно предположить, что в контексте создания информационной системы для мебельного магазина в базовый класс для всех видов мебели разумно добавить общий для всех метод “показать характеристики”. При распечатке характеристик всех видов товара программа бы без разбору для всех объектов вызывала бы этот метод, а каждый конкретный объект уже сам бы решал, какую информацию ему предоставлять. Как это реализуется: Во-первых, в базовом классе определяют общий для всех метод с общим для всех поведением. В случае с нашим примером это будет метод, печатающий общие для любых типов мебели параметры. Во-вторых, в каждом производном классе, где это необходимо, переопределяют базовый метод (добавляют метод с тем же именем), где расширяют базовое поведение своим, например, выводят характеристики, свойственные только конкретному виду мебельной продукции. Метод в базовом классе иногда вообще не обязан содержать какой-либо код, а необходим только для того, чтобы определить имя и набор параметров – сигнатуру метода. Такие методы называют абстрактными методами, а классы, их содержащие, автоматически становятся абстрактными классами. Итак, полиморфизм – это возможность единообразного общения с объектами разных классов через определенный интерфейс. Идеология полиморфизма гласит, что для общения с объектом вам не нужно знать его тип, а нужно знать, какой интерфейс он поддерживает.

Интерфейс. В некоторых языках программирования (C#, Java) понятие интерфейса выделено явно - это не только открытые методы и свойства самого класса. Такие языки,

как правило, не поддерживают множественного наследования и компенсируют это тем, что любой объект может иметь один базовый объект и реализовывать любое количество интерфейсов. Интерфейс в их интерпретации – это подобие абстрактного класса, содержащего только описание (сигнатуру) открытых методов и свойств. Реализация интерфейса ложится на плечи каждого класса, который собирается его поддерживать. Один и тот же интерфейс могут реализовывать классы абсолютно разных иерархий, что расширяет возможности полиморфизма. К примеру, интерфейс “сохранение/восстановление информации в базе данных” могли бы реализовывать как классы иерархии “мебель”, так и классы, связанные с оформлением заказов на изготовление мебели, а при нажатии на кнопку “сохранить” программа бы прошлась по всем объектами, запросила бы у них этот интерфейс и вызвала бы соответствующий метод.

1. 9 Лекция № 9 (2 часа).

Тема: «Событийно-управляемая модель программирования»

1.9.1 Вопросы лекции:

1. Развитие идей ООП.

2 Парадигмы программирования.

1.9.2 Краткое содержание вопросов:

1. Развитие идей ООП.

Новые термины и понятия, связанные с ООП.

События. Специальный вид объектов, создаваемый для оповещения одних объектов о событиях, происходящих с другими объектами. В разных языках программирования механизм событий реализуется по-разному: где-то с помощью специальных синтаксических конструкции, а где-то силами базовых средств ООП.

Универсальный тип. Концепция универсальных типов не связана непосредственно с концепцией ООП, но она является причиной появление таких элементов, как универсальный класс, универсальный метод, универсальное событие и т.д. Универсальный тип – это тип, параметризованный другим типом (набором типов). Кем является этот тип-параметр в контексте проектирования универсального типа неизвестно, хотя есть возможность ограничить значения типов-параметров, заставив их быть производными от конкретного класса или реализовывать определенные интерфейсы. В качестве примера можно привести универсальный класс сортировки последовательности элементов, где тип элемента в последовательности заранее неизвестен. При проектировании такого класса важно указать, что тип-параметр должен поддерживать операцию сравнения. При создании объектов универсальных типов параметр указывается явно, например целочисленный или строковый тип, а сам объект начинает себя вести так, как если бы это был экземпляр класса, созданный специально для сортировки целых чисел или строк.

Исключения. Еще один специальный вид объектов, поддерживаемый встроенным в конкретный язык программирования механизмом обработки ошибок и исключительных ситуаций. Исключения, помимо кода ошибки, содержат ее описание, возможные причины возникновения и стек вызовов методов, имевший место до момента возникновения исключения в программе.

Недостатки объектно-ориентированного программирования

Про то, что популярность объектно-ориентированного подхода к созданию программных продуктов огромна я уже сказал. Про то, что тех, кто стремится расширить эту парадигму довольно много, я тоже уже отметил. Но есть еще один способ выделиться среди огромного сообщества специалистов в информационных технологиях – это заявить, что ООП себя не оправдало, что это не панацея, а, скорее, плацебо. Есть среди этих людей действительно специалисты очень высокого класса, такие как Кристофер Дэйт, Александр

Степанов, Эдсгер Дейкстра и другие, и их мнение заслуживает внимания, но есть и те, про которых говорят, что “плохому танцору всегда что-то мешает”. Вот они, наиболее очевидные недостатки ООП, на которые указывают специалисты:

1. ООП порождает огромные иерархии классов, что приводит к тому, что функциональность расползается или, как говорят, размывается по базовым и производным членам класса, и отследить логику работы того или иного метода становится сложно.
2. В некоторых языках все данные являются объектами, в том числе и элементарные типы, а это не может не приводить к дополнительным расходам памяти и процессорного времени.
3. Также, на скорости выполнения программ может неблагоприятно сказаться реализация полиморфизма, которая основана на механизмах позднего связывания вызова метода с конкретной его реализацией в одном из производных классов.
4. Психологический аспект. Многие считают, что ООП это круто и начинают использовать его подходы всегда и везде и без разбору. Все это приводит к снижению производительности программ в частности и дискредитации ООП в целом.

Процедура метода работает с экземплярами объекта. Они очень напоминают библиотечные подпрограммы, поскольку должны иметь хорошо определенный вызов и возвращать в виде интерфейса значение, но знание внутренней организации процедуры метода не требуется.

Процедуры метода для объекта должны обеспечивать исчерпывающее обслуживание объектов, то есть, они должны быть единственными процедурами, для который разрешен непосредственный доступ к объектам. Кроме того, при построении методов следует использовать принципы абстрактных данных: вы должны иметь возможность вызова процедур метода без необходимости знать о том, как они работают и какую имеют внутреннюю организацию.

Что касается всех других аспектов, то вы можете писать процедуры методов на любом известном вам языке или интерфейсе, хотя обычно используются соглашения по вызову C++ или Паскаля. Аргументы процедур также выбираются по вашему усмотрению. Обычно необходимым является один аргумент - указатель на экземпляр объекта. Некоторые процедуры методов могут потребовать дополнительных параметров. Например, инициализация метода для объекта списка требует просто указатель на объект списка, в то время как метод включения в список требует указатель на список, указатель на новый узел для вставки и указатель на узел, включаемый после него.

В использовании статических и виртуальных методов есть свои достоинства и недостатки. Статические методы определяются на этапе компиляции, а результатом будет непосредственный вызов процедуры метода. Это позволяет выполнить вызов быстрее и не требует использования промежуточных регистров (как при использовании виртуальных методов). Однако, поскольку эти вызовы определяются на этапе компиляции, вызовы статических методов не обладают гибкостью вызовов виртуальных методов.

Вызовы виртуальных методов выполняются непосредственно через реализацию таблицы виртуальных методов объекта. Тот факт, что вызов является косвенным, приводит к тому недостатку, что при выполнении вызова требуется использование промежуточных регистров (что может усложнить код программы). Однако большим преимуществом является то, что вызовы виртуальных методов определяются на этапе выполнения. Таким образом, вы можете выполнять вызовы виртуальных методов для порожденного объекта с помощью вызова метода общего объекта-"предка". При этом не требуется точно знать, с каким видом объекта-потомка вы имеете дело.

Описание процедур статических и виртуальных методов в точности совпадает с описанием любой другой процедуры, но имеется следующее исключение: если для

виртуального метода вы опускаете имя процедуры, то в таблице виртуальных методов создается пустая неинициализированная ячейка, и Турбо Ассемблер не выводит вам никаких предупреждений. Если метод не является виртуальным, то пропуск имени процедуры является ошибкой, поскольку не виртуальные методы не включаются в таблицу.

2 Парадигмы программирования.

Парадигма (метод) программирования — это совокупность идей и понятий, определяющих стиль написания компьютерных программ.

Парадигма программирования представляет (и определяет) то, как программист видит выполнение программы. Например, в объектно-ориентированном программировании программист рассматривает программу как набор взаимодействующих объектов, тогда как в функциональном программировании программа представляется в виде цепочки вычисления функций.

Следует отметить, что парадигма программирования не определяется однозначно языком программирования; практически все современные языки программирования в той или иной мере допускают использование различных парадигм.

Можно выделить две большие группы языков программирования, отражающих разные парадигмы программирования:

- императивными (imperative), называемые также процедурными (procedural),
- декларативные (declarative) языки.

Главное различие между этими группами языков заключается в следующем: декларативная программа заявляет (декларирует), что должно быть достигнуто в качестве цели, а директивная предписывает, как ее достичь.

Пример. Предположим, вам надо пройти в городе из пункта А в пункт Б. Декларативная программа - это план города, в котором указаны оба пункта, плюс правила уличного движения. Руководствуясь этими правилами и планом города, курьер сам найдет путь от пункта А к пункту Б. Императивная программа - это список команд примерно такого рода: от пункта А по ул. Садовой на север до площади Славы, оттуда по ул. Пушкина два квартала, потом повернуть направо и идти до Театрального переулка, по этому переулку налево по правой стороне до дома 20, который и есть пункт Б.

Императивное программирование — это парадигма программирования, которая описывает процесс вычисления в виде инструкций, изменяющих состояние программы. Императивная программа очень похожа на приказы, выражаемые повелительным наклонением в естественных языках, то есть это последовательность команд, которые должен выполнить компьютер.

ПРИМЕРЫ: Фортран (1957), Бейсик (1963) , Паскаль (1970), Си++ (1983).

Декларативное программирование — это парадигма программирования, которая описывает конечное состояние программы**.** Программа «декларативна», если она написана на исключительно функциональном, логическом или языке программирования с ограничениями. Выражение «декларативный язык» иногда употребляется для описания всех таких языков программирования как группы, чтобы подчеркнуть их отличие от императивных языков.

ПРИМЕРЫ: Лисп (1967), Пролог (1974),

Все парадигмы - всего лишь различные инструменты, которые можно использовать при программировании. Каждый из этих инструментов по-своему хорош. Все описанные в статье модели вычислений эквивалентны. Но это не значит, что они "эффективно универсальны". То есть, на самом деле, различные методики программирования дают разный выигрыш для решения задач разных классов. Этот выигрыш можно мерить по двум параметрам:

- эффективность программного обеспечения на современных ЭВМ

- общие затраты на разработку программного обеспечения

Так как современные компьютеры (персональные, как наиболее широко распространенные и, как ни странно, наиболее востребованные широким пользователем) практически все построены по принципам, заложенным еще Фон Нейманом в середине нашего века. То есть, есть процессор, есть память, есть внешние устройства, и все это работает под управлением последовательной выборки команд из памяти.

Читатель, сведущий в архитектурах ЭВМ, знает, что компьютеры, на самом деле, не далеко ушли по своему внутреннему устройству от гипотетических последовательных вычислительных моделей, использовавшихся в начале века пионерами исследования алгоритмов для доказательств базовых утверждений современной теории вычислимости. В самом деле, процессор - это система (да, очень большая) логических элементов. То есть, процессор - схема булевых элементов, реализующая функцию переходов некоторого автомата (почему некоторого? сам компьютер и есть этот автомат) на множестве состояний (вся память какая есть и будет этим самым состоянием). Поэтому современные компьютеры практически все ориентированы на последовательные вычисления. Со всеми замечаниями относительно разных уровней параллелизма согласен, но об этом будет отдельный разговор дальше. Следовательно, парадигмой, имеющей наиболее "естественную" реализацию семантики на нынешних компьютерах, является императивное программирование. Оно заведомо выигрывает любой другой методологии в эффективности реализации. Хорошие трансляторы, например, с чистых объектно-ориентированных языков - вещь достаточно редкая. По изложенным выше причинам не стоит осуждать отдельные парадигмы исключительно руководствуясь аргументом: "мала эффективность готовой программы", забывая о том, что практически все сложные программы работают не так быстро и требуют больших объемов памяти.

В следующих частях будут подробно рассмотрены отдельные методологии программирования. Это потребует описания традиционного синтаксиса некоторого языка программирования, поддерживающего эту методологию (да, между Паскалем и Си очень много разницы; в одном надо писать begin...end, а в другом {...}, и так далее). Будут описаны традиционные средства структурирования программ, принятые в этих языках. Естественно, будет рассказано о типичной семантике таких языков. И, что наиболее важно, будет рассмотрен класс задач, для которого эти методологии дают достаточно эффективные решения.

Практически любой язык программирования в наши дни - это язык определений. Программы представляют из себя множество определений программных объектов (типов данных, функций, ...), которые как-то взаимосвязаны. Методологии программирования, как правило, фокусируются на описании алгоритмической части определений, входящих в программу. Методология для работы с описательной частью всего одна: программа должна быть максимально структурирована. Это помогает переиспользовать единожды написанный код, да и просто облегчает понимание текста программы. Это знали и в середине XX века, это знают и сейчас. Структурированность программы позволяет повысить уровень декларативности (то есть, еще сильнее оторваться от деталей архитектуры конкретного компьютера и программировать практически в терминах предметной области задачи) даже для программирования на языке ассемблера процессора Intel. В алгоритмической же части многие современные языки на самом деле поддерживают в явном виде несколько парадигм программирования.

Для описания синтаксиса будут использованы грамматики в расширенной форме Бекуса-Науэра. В правой части таких грамматик допускается использование следующих "регулярных операций":

A B - последовательно A, за тем B.

A | B - альтернатива. Читается: "A или B".

A* - произвольное количество повторений (в том числе - 0 раз) A. Читается:

"последовательность A".
A # B - эквивалентно A (B A)*. Читается: "последовательность A через B".
Терминальные символы выделяются подчеркиванием.

Императивное программирование

Про императивное программирование мы уже практически все сказали. Автомат, последовательно изменяющий свои состояния под управлением некоторой схемы, наиболее просто реализуется технически. Поэтому первые компьютеры были императивными. И остались такими и в наши дни, несмотря на все эксперименты с оригинальными вычислительными устройствами. Даже типичное определение алгоритма, вдалбливаемое еще со школы (описание последовательности действий для решения какой-либо задачи), несет на себе сильнейший отпечаток императивного подхода. Стоит ли говорить о том, почему императивное программирование - практически наиболее "популярное"?

Одна из характерных черт императивного программирования - наличие переменных с операцией "разрушающего присвоения". То есть, была переменная A, было у нее значение X. Алгоритм предписывает на очередном шаге присвоить переменной A значение Y. То значение, которое было у A, будет "навсегда забыто". Вот что на практике означает "переход между состояниями под управлением функции переходов".

Синтаксис описания алгоритмов в простейшем языке, поддерживающем императивную модель программирования, мог бы быть таким:

Оператор ::= Простой оператор | Структурный оператор
Простой оператор ::= Оператор присваивания | Оператор вызова | Оператор возврата
Структурный оператор ::= Оператор последовательного исполнения | Оператор ветвления | Оператор цикла
Оператор присваивания ::= Переменная := Выражение ;
Оператор вызова ::= Имя подпрограммы (Список параметров) ;
Оператор возврата ::= return [Выражение] ;
Оператор последовательного исполнения ::= begin Оператор* end
Оператор ветвления ::= if Выражение then Оператор* (elseif Выражение then Оператор*)* [else Оператор*] end
Оператор цикла ::= while Выражение do Оператор* end

Семантика такого языка описывается достаточно легко. Состоянием вычислительного устройства будут указатель текущей инструкции, значения всех используемых программой ячеек памяти, и состояние стека возвратов из подпрограмм. Исполнение каждого оператора тривиальным образом записывается как изменение этого "состояния вычислителя" (если считать, что алгоритм представлен в виде дерева вывода в указанной грамматике, то с описанием переходов не должно возникнуть никаких проблем).

Про наш мир можно сказать, что он локально императивен. То есть, если взять достаточно узкую задачу, то ее можно вполне легко описать методами последовательного программирования. Практика показывает, что более сложные императивные программы (компиляторы, например) пишутся и отлаживаются долго (годами). Переиспользование кода и создание предметно-ориентированных библиотек упрощает программирование, но ошибки в реализации сложных алгоритмов проявляются очень часто.

Императивное программирование наиболее пригодно для реализации небольших подзадач, где очень важна скорость исполнения на современных компьютерах. Кроме этого, работа с внешними устройствами, как правило, описывается в терминах последовательного исполнения операций ("открыть кран, набрать воды"), что делает такие задачи идеальными кандидатами на императивную реализацию.

Параллелизм.
Параллельное и событийно-управляемое программирование

Исторически сложилась такая ситуация, что компьютеры изначально использовались для решения больших вычислительных задач, которые были просто "неподъемны" для человека. Для таких задач характерно, что приходится проделывать практически те же вычисления для каждого элемента очень большого массива данных. Постепенно многие крупные организации осознали пользу компьютеров как средств централизованного хранения и обработки своих "картотек" (из которых и выросли современные базы данных). Оказалось, что и в задачах, связанных с систематизацией такой информации, оказывается много общего с численными методами математической физики, в терминах алгоритмической реализации.

Достаточно быстро были разработаны супер-ЭВМ, имеющие "векторную" или "матричную" архитектуру. Такие ЭВМ представляют, на самом деле, несколько процессоров, достаточно автономных, но способных обмениваться между собой информацией о результатах своих вычислений. Достаточно хорошим приближением к таким архитектурам являются сети компьютеров с возможностью распределенных вычислений. Кроме этого, практически все современные микропроцессоры максимально используют в своей архитектуре возможности параллельного исполнения отдельных операций. Таким образом, параллелизм можно мысленно разбить на два уровня: параллелизм уровня микроопераций и параллелизм уровня **процессов**.

Процессы - это абстракция достаточно высокого уровня. В какой вычислительной модели работает каждый отдельный процесс - не принципиально. Важно, что они могут работать параллельно, и могут обмениваться между собой результатами своих вычислений через "каналы связи". Примерно такую модель взаимодействия процессов реализует язык параллельного программирования Occam. Вот некоторая грамматика описания процессов, достаточно близкая к принятой в Occam-е:

Процесс ::= Простой процесс | Структурный процесс

Простой процесс ::= Послать значение | Принять значение | Процесс вычислительной модели

Структурный процесс ::= Последовательный процесс | Параллельный процесс

Послать значение ::= Канал связи << Выражение ;

Принять значение ::= Канал связи >> Выражение ;

Процесс вычислительной модели ::= *нечто, определяемое конкретным вычислителем*

Последовательный процесс ::= seq Процесс* end

Параллельный поцесс ::= par Процесс* end

Семантически взаимодействие параллельных процессов лучше всего представлять как работу сети неких устройств, соединенных "проводками", по которым текут данные. Спроектировав в таких терминах, например, систему параллельных процессов для быстрого суммирования большого количества чисел, можно легко описать эту систему в приведенном синтаксисе.

Каждый вычислитель производит типичные для его вычислительной модели операции (например, императивный вычислитель будет переходить из состояния в состояние). Когда процесс встречает инструкцию "Принять значение (из канала)", он входит в состояние ожидания, пока канал пуст. Как только в канале появляется значение, процесс его считывает и продолжает работу.

Достаточно распространенной является так же и следующее понимание параллелизма: в системе параллельных процессов каждый отдельный процесс обрабатывает события. События могут быть как общими для всей системы, так и индивидуальными для одного или нескольких процессов. В таких терминах достаточно удобно описывать, например, элементы графического интерфейса пользователя, или моделирование каких-либо реальных процессов (например, управление уличным движением) - так как понятие события является для таких задач естественным. Такое программирование принято

называть **событийно-управляемым**. В событийно-управляемом программировании отдельные процессы максимально автономны, единственное средство общения между ними - посылка сообщений (порождение новых событий). Событийно-управляемое программирование очень близко к объектно-ориентированному прграммированию, которое будет подробно разобрано в дальнейшем.

Параллелизм является не только надстройкой над другими вычислительными моделями; некоторые методологии программирования имеют естественную реализацию на платформах, поддерживающих параллелизм. Об этом будет упомянуто отдельно в каждом таком случае.

Функциональное программирование

До сих пор рассматриваемые нами парадигмы программирования воспринимались нами как некоторые "полезные надстройки над императивным программированием". Уже отмечалось, например, что параллельное программирование - это программирование в терминах взаимодействия некоторых одновременно работающих абстрактных вычислителей, и почти ничего не говорили о вычислительной модели, на которой основаны отдельные элементы этой системы. Мы ничего не сказали о том, на каком языке описаны обработчики сообщений у объектов (кроме того, что в этих языках основной операцией является посылка сообщения). Функциональное программирование представляет из себя одну из альтернатив императивному подходу.

В императивном программировании алгоритмы - это описания последовательно исполняемых операций. Здесь существует понятие "текущего шага исполнения" (то есть, времени), и "текущего состояния", которое меняется с течением этого времени.

В функциональном программировании понятие времени отсутствует. Программы являются выражениями, исполнение программ заключается в вычислении этих выражений. Практически все математики, сами того не замечая, занимаются функциональным программированием, описывая, например, чему равно абсолютное значение произвольного вещественного числа.

Императивное программирование основано на машине Тьюринга-Поста - абстрактном вычислительном устройстве, предложенном на заре алгоритмической эры для описания алгоритмов. Функциональное программирование основано на более естественном с математической точки зрения формализме - лямбда-исчислении Черча.

Как правило, рассматривают так называемое "расширенное лямбда-исчисление". Его грамматику можно описать следующим образом:

Выражение ::= Простое выражение | Составное выражение

Простое выражение ::= Константа | Имя

Составное выражение ::= Лямбда-абстракция | Применение | Квалифицирванное выражение | Ветвление

Лямбда-абстракция ::= lambda Имя -> Выражение end

Применение ::= (Выражение Выражение)

Квалифицированное выражение ::= let (Имя = Выражение ;)* in Выражение end

Ветвление ::= if Выражение then Выражение

(elseif Выражение then Выражение)* else Выражение end

Константами в расширенном лямбда-исчислении могут быть числа, кортежи, списки, имена предопределенных функций, и так далее.

Результатом вычисление применения предопределенной функции к аргументам будет значение предопределенной функции в этой "точке". Результатом применения лямбда-абстракции к аргументу будет подстановка аргумента в выражение - "тело" лямбда-абстракции. Сами лямбда-абстракции так же являются выражениями, и, следовательно, могут быть аргументами.

Вы уже заметили, что лямбда-абстракции имеют всего один аргумент. В то же время, функции в традиционном понимании не обязаны быть одноместными.

Чистое лямбда-исчисление Черча позволяет обходится исключительно именами, лямбда-абстракциями от одного аргумента и применениями выражений к выражениям. Оказывается, в этих терминах можно описать и "предопределенные" константы (числа и т.п.), структуры данных (списки, кортежи...), логические значения и ветвление. Более того, в чистом лямбда-исчислении можно обойтись без квалифицированных выражений, и, следовательно, выразить рекурсию, не используя для этого употребления имени функции в теле функции. Некоторые экспериментальные модели функционального программирования позволяют обходится без каких-либо имен вообще. Подробнее об этом можно почитать в специальной литературе, например, в книге Филда и Харрисона "Функциональное программирование".

Функциональное программирование обладает следующими двумя примечательными свойствами:

1.) **Аппликативность**: программа есть выражение, составленное из применения функций к аргументам.

2.) **Настраиваемость**: так как не только программа, но и любой программный объект (в идеале) является выражением, можно легко порождать новые программные объекты по образцу, как значения соответствующих выражений (применение порождающей функции к параметрам образца).

Настраиваемость активно используется в таком направлении программирования, как generic programming. Основная задача, решаемая в рамках это направления - создание максимально универсальных библиотек, ориентированных на решение часто встречающихся подзадач (обработка агрегатных данных; потоковый ввод-вывод; взаимодействие между программами, написанными на разных языках и различающихся в деталях семантики; универсальные оконные библиотеки). Эти направления наиболее ярко представлены в STL - стандартной библиотеке шаблонов (контейнеров) языка Си++, а так же - в реализации платформы .NET фирмы MicroSoft. Нередко в разговорах о пользе функционального программирования можно услышать следующее утверждение: "самые крупные специалисты по функциональному языку Haskell в настоящее время находятся в MicroSoft Research".

Для обеспечения видовой корректности программ в функциональные языки вводят специальные системы типов, ориентированные на поддержку настраиваемости. Как правило, трансляторы функциональных языков могут самостоятельно определять типы выражений, без каких-либо описаний типов вообще. Так, функция `add = lambda x -> lambda y -> x+y end end` будет иметь тип `number -> number -> number`, а уже рассматриваемая нами функция apply - тип `any(X).any(Y).(X->Y)->X->Y`, где any обозначает "квантор всеобщности" для типов, а X и Y являются переменными.

Можно заметить, что так как порядок вычисления подвыражений не имеет значения (благо "состояния" у функциональной программы нет), функциональное программирование может быть естественным образом реализовано на платформах, поддерживающих параллелизм. "Потоковая модель" функционального программирования, о которой так же можно почитать у Филда и Харрисона, является естественным представлением функциональных программ в терминах систем взаимодействующих процессов.

Функциональное программирование, как и другие модели "неимперативного" программирования, обычно применяется для решения задач, которые трудно сформулировать в терминах последовательных операций. Практически все задачи, связанные с искусственным интеллектом, попадают в эту категорию. Среди них следует отметить задачи распознавания образов, общение с пользователем на естественном языке, реализацию экспертных систем, автоматизированное доказательство теорем, символьные вычисления. Эти задачи далеки от традиционного прикладного программирования, поэтому им уделяется не так много внимания в учебных программах по информатике.

Логическое программирование

В функциональном программировании программы - это выражения, и их исполнение заключается в вычислении их значения. В логическом программировании программа представляет из себя некоторую теорию (описанную на достаточно ограниченном языке), и утверждение, которое нужно доказать. В доказательстве этого утверждения и будет заключаться исполнение программы.

Логическое программирование и язык Пролог появились в результате исследования группы французских ученых под руководством Колмерье в области анализа естественных языков. В последствии было обнаружено, что логическое программирование столь же эффективно в реализации других задач искусственного интеллекта, для чего оно в настоящий момент, главным образом, и используется. Но логическое программирование оказывается удобным и для реализации других сложных задач; например, диспетчерская система лондонского аэропорта Хитроу в настоящий момент переписывается на Прологе. Оказывается, логическое программирование является достаточно выразительным средством для описания сложных систем.

В логике теории задаются при помощи аксиом и правил вывода. То же самое мы имеем и в Прологе. Аксиомы здесь принято называть фактами, а правила вывода ограничить по форме до так называемых "дизъюнктов Хорна" - утверждений вида $A <= B_1 \& ... \& B_n$. В Прологе такие утверждения принято записывать так: `a :- b1, ..., bn.`

а факты, они же аксиомы, представлять как правила с пустой "посылкой": `a.`

Переменные в утверждениях Пролога принято обозначать идентификаторами, начинающимися с заглавной буквы.

Логическое программирование допускает естественную параллельную реализацию. В примере a :- b, c. порядок согласования целей b и c не имеет значения, поэтому их можно доказывать параллельно. Говорят, что процессы доказательства b и c образуют **И-систему** процессов: И-система успешно доказывается, если каждый процесс, входящий в систему, успешен. В примере с предикатом member два правила для него могли применяться параллельно, образуя **ИЛИ-систему** процессов. ИЛИ-система успешно доказывается, если хотя бы 1 процесс в системе успешен. Переменные, общие для системы процессов (например, в случае a(X) :- b(X), c(X).) преобразуются в каналы связи между процессами в системе. Связывание переменной (присвоение ей значения) аналогично посылке значения в канал.

В настоящее время существует несколько "промышленных" реализаций языка Пролог (наряду с большим количеством "исследовательских" версий). "Промышленный" транслятор Пролога, как правило, порождает исполняемый код, сопоставимый по эффективности с кодом аналогичной программы на императивных языках; компилируемое им подмножество "чистого Пролога" наделено строгой системой типов и возможностью вызывать процедуры, написанные на других языках (Си, Паскаль, Ассемблер...).

Среди экспериментальных расширений Пролога следует упомянуть такие языки, как лямбда-Пролог (Пролог с элементами функционального программирования), Goedel (язык, в котором семантический анализ может быть описан алгоритмически средствами самого языка), Mercury (версия чистого Пролога, предназначенная для промышленного использования и снабженная системой полиморфных типов, аналогичной используемой в современных функциональных языках).

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

2.1 Лабораторная работа № 1 (2 часа).

Тема: «Арифметические операции и выражения»

2.1.1 Цель работы: изучить стандартный набор арифметических операций и выражений, используемый в языке программирования, рассмотреть особенности выполнения действий.

2.1.2 Задачи работы:

1. Знакомство с языком программирования высокого уровня
2. Изучение стандартных арифметических операций
3. Запись стандартных математических функций

2.1.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Язык программирования высокого уровня, компьютер

2.1.4 Описание (ход) работы:

Ознакомиться с теоретическим материалом.

Алфавит языка Паскаль состоит из множества символов, включающих в себя буквы, цифры и специальные символы:

1) Латинские буквы: от *A* до *Z* (прописные) и от *a* до *z* (строчные). В служебных словах строчные и прописные буквы не различаются.
2) Цифры: от 0 до 9.
3) Специальные символы:

:= знак присваивания
>= больше или равно
<= меньше или равно
<> не равно

Идентификатор – символическое имя определенного программного объекта. Такими объектами являются имена констант, переменных, типов данных, процедур и функций, программ. Идентификатор – это любая последовательность букв и цифр, начинающаяся с буквы.

В Паскале типы данных: а) *целые* (идентификатор – Integer, диапазон значений: от -32768 до 32767); б) *вещественные* (идентификатор – Real, диапазон значений: от $2{,}9 \cdot 10^{-39}$ до $1{,}7 \cdot 10^{38}$)

В таблице 1 даны арифметические операции, в ней *I* обозначает целые типы, *R* – вещественные типы.

Таблица 1

Знак	Выражение	Типы операндов	Тип результатов	Операция
+	A+B	*R, R* *I, I* *I, R; R, I*	*R* *I* *R*	Сложение
-	A-B	*R, R* *I, I* *I, R; R, I*	*R* *I* *R*	Вычитание
*	A*B	*R, R* *I, I* *I, R; R, I*	*R* *I* *R*	Умножение
/	A/B	*R, R* *I, I* *I, R; R, I*	*R* *R* *R*	Вещественное деление
div	A div B	*I, I*	*I*	Целое деление
mod	A mod B	*I, I*	*I*	Остаток от целого деления

Таблица 2 содержит описания математических стандартных функций, аргументы записываются в круглых скобках.

Таблица 2

Обозначение	Типы аргумента	Тип результата	Функция
Abs(x)	*I, R*	*I, R*	Модуль аргумента x
Sqr(x)	*I, R*	*R*	Квадрат аргумента x
Sqrt(x)	*I, R*	*I, R*	Корень квадратный из аргумента x
Int(x)	*I, R*	*R*	Целая часть аргумента x
Frac(x)	*I, R*	*R*	Дробная часть аргумента x
Trunc(x)	*R*	*I*	Ближайшее целое, не превышающее x по модулю
Round(x)	*R*	*I*	Округление до ближайшего целого
Exp(x)	*I, R*	*R*	Экспонента e^x
Ln(x)	*I, R*	*R*	Натуральный логарифм x
Cos(x)	*I, R*	*R*	Косинус x (x в радианах)
Sin(x)	*I, R*	*R*	Синус x (x в радианах)
Arctan(x)	*I, R*	*R*	Арктангенс x (радианы)

Замечание: в Паскале для вычисления x^y при вещественном y используется формула $x^y = e^{y \cdot Ln(x)}$.

Задание 1. Записать программу извлечения корня пятой степени из суммы двух чисел. Проверить выполнимость программы на Паскале.

Задание 2. Записать программу на Паскале, запрашивающую поочередно 2 числа и вычисляющую значение выражения e^{2x+y}. Вычислить значение при $x = 3,\ y = -4$.

Задание 3. Записать программу деления двух натуральных чисел с определением частного и остатка.

Задание 4. Записать программу вычисления среднего арифметического и среднего геометрического трех чисел.

Задание 5. Для следующих формул записать соответствующие арифметические выражения на Паскале:

а) $\sqrt[3]{5+4x^2}$; б) $\cos^2 x^3$; в) $\log_3 \frac{x}{2}$.

Вычислить значение каждого выражения при $x = 7$, составив программу на Паскале. Записать программу в тетрадь.

2.2 Лабораторная работа № 2 (2 часа).

Тема: «Ввод и вывод данных»

2.2.1 Цель работы: изучить операторы ввода и вывода данных, используемые в языке программирования, рассмотреть особенности данных операторов.

2.2.2 Задачи работы:

1. Изучение форматов ввода и вывода чисел
2. Запись алгоритма решения простейших математических задач
3. Составление программы на языке программирования высокого уровня

2.2.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Язык программирования высокого уровня, компьютер

2.2.4 Описание (ход) работы:

Записать программу решения задач и проверить ее правильность на языке высокого уровня.

Задача 1. Составить программу в Паскале, вычисляющую определитель третьего порядка.

Задание 2. Дано действительное число x. Определить результат выполнения оператора:
WriteLn(x); WriteLn(x:6); WriteLn(x:12); WriteLn(x:18);
WriteLn(x:12:3); WriteLn(x:12:6); WriteLn(x:18:6)
для чисел: а) $x = 86{,}57$; б) $x = 0{,}00421$, в) $x = 15469$, г) $x = 7.29E + 002$; д) $x = 285.6E - 04$.
Сделать вывод о форматах вывода чисел в Паскале.
Задание 3. Записать программу вывода первых четырех степеней числа e (основание натурального логарифма) с точностью до десятитысячных.
Задание 4. Составить программу, вычисляющую с точностью до сотых в градусной мере острые углы прямоугольного треугольника по катету и гипотенузе, и ведущую диалог с пользователем:
ВВЕДИТЕ ДЛИНУ КАТЕТА
$a =$ …….
ВВЕДИТЕ ДЛИНУ ГИПОТЕНУЗЫ
$c =$ …….
ОСТРЫЕ УГЛЫ РАВНЫ
……
Задание 5. Дан радиус окружности. Составить одну программу на Паскале, вычисляющую: а) длину окружности; б) диаметр окружности; в) площадь круга, ограниченного этой окружностью; г) сторону равностороннего треугольника, вписанного в данную окружность; д) сторону квадрата, описанного около этой окружности. Использовать комментарии при выводе каждой вычисляемой величины.

2.3 Лабораторная работа № 3 (2 часа).

Тема: «Составление программ простейших задач»

2.3.1 Цель работы: научиться составлять программы простейших задач

2.3.2 Задачи работы:

1. Изучение особенностей записи программ математических задач
2. Составление алгоритмов и программ разветвляющейся структуры

2.3.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Язык программирования высокого уровня, компьютер

2.3.4 Описание (ход) работы:

Задание 1. Записать в Паскале программу решения квадратного уравнения:

```
Var a,b,c,d,x1,x2: Real;
Begin  WriteLn ('Введите числа a,b,c');
Write('a='); ReadLn(a);   Write('b='); ReadLn(b);   Write('c='); ReadLn(c);
 if a<>0
  Then Begin
 d:=b*b-4*a*c;    if d>=0
  Then Begin
   x1:=(-b+sqrt(d))/(2*a);   x2:=(-b-sqrt(d))/(2*a);
   WriteLn('Корни квадратного уравнения');
   WriteLn('x1=',x1);  WriteLn('x2=',x2);
   End
   Else WriteLn('Корней квадратного уравнения нет'); End
   Else WriteLn('Это не квадратное уравнение');
End.
```

Ввести значения: 1) 1.5 (Enter), -4.9 (Enter), 2.6 (Enter); 2) 4, 7, 5; 3) 0, 1, 2.
Записать в тетрадь программу и результаты вычислений в каждом случае.
Задание 2. Составить и записать программу, вычисляющую с точностью до сотых в градусной мере острые углы прямоугольного треугольника по катету и гипотенузе. При неверных исходных данных выводить на экран сообщение об этом.

Задание 3. Заданы два вектора в пространстве своими координатами. Написать программу, которая определяет угол между векторами в радианной мере.
Задание 4. Услуги телефонной сети оплачиваются по следующему правилу: за разговоры до А минут в месяц – В рублей, а за разговоры сверх установленной нормы оплачиваются из расчета С рублей за минуту. Написать программу, вычисляющую плату за пользование телефоном для введенного времени разговоров за месяц.
Задание 5. Написать программу, вычисляющую площадь круга, вписанного в треугольник с заданными сторонами.

2.4 Лабораторная работа № 4 (2 часа).
Тема: «Условный оператор»

2.4.1 Цель работы: изучить условный оператор, используемый в языке программирования, рассмотреть особенности данного оператора, его различные формы записи.

2.4.2 Задачи работы:

1. Составление алгоритмов и программ разветвляющейся структуры
2. Использование комментариев при записи программы

2.4.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Язык программирования высокого уровня, компьютер

2.4.4 Описание (ход) работы:

Записать программу решения задач и проверить ее правильность на языке высокого уровня.

1. Написать программу, которая проверяет, лежат ли три данные точки на одной прямой или нет, и выдает сообщение об этом.

2. В восточном календаре принят двенадцатилетний цикл. Годы внутри цикла носили название животных: крысы, коровы, тигра, кролика, дракона, змеи, лошади, овцы, обезьяны, петуха, собаки и свиньи. Написать программу, которая по введенному году печатает его название по восточному календарю (2008 год – год Крысы).

3. Составить программу, вычисляющую с точностью до десятых значение функции

$$f(x)=\begin{cases}3+x^2, & x\le 0\\ \cos x, & 0<x\le\pi\\ 1-x, & x>\pi\end{cases}.$$

4. Даны числа a,b,c. Найти площадь треугольника с данными сторонами. Рассмотреть все варианты неверных исходных данных.

5. Составить программу в Паскале, решающую биквадратное уравнение.

6. Записать в Паскале программу выбора наибольшего значения из трех различных действительных чисел.

2.5 Лабораторная работа № 5 (2 часа).
Тема: «Циклические конструкции»

2.5.1 Цель работы: изучить циклические конструкции, используемые в языке программирования, рассмотреть их особенности, условия использования.

2.5.2 Задачи работы:

1. Виды циклических конструкций
2. Условия использования цикла с параметром, циклов с постусловием и предусловием
3. Составление программ с использованием циклических конструкцию.

2.5.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Язык программирования высокого уровня, компьютер

2.5.4 Описание (ход) работы:

Записать программу решения задач и проверить ее правильность на языке высокого уровня.

1. Начав тренировки, спортсмен в первый день пробежал заданное 10 км. Каждый день он увеличивал дневную норму на 10% нормы предыдущего дня. Чему равна дневная норма десятого дня тренировок? Определить суммарный путь, который пробежит спортсмен за неделю.

2. Ежемесячная стипендия студента составляет А рублей, а расходы на проживание превышают стипендию и составляют В рублей в месяц. Рост цен ежемесячно увеличивает расходы на 3%. Какую сумму денег необходимо единовременно попросить у родителей, чтобы можно было прожить учебный год (10 месяцев), используя только эти деньги и стипендию?

3. Дана арифметическая прогрессия с первым членом A=3, разностью d=1,2. Написать первые N=10 членов последовательности.

4. Дана геометрическая прогрессия с первым членом A=5, знаменателем q=1/4. Написать первые N=6 членов последовательности.

5. Даны последовательность с общим членом $a_n = \frac{n+5}{2^n}$ и некоторое $\varepsilon < 1$. Найти сумму тех членов последовательности, которые больше или равны заданному ε=0,08.

6. Вычислить: а) sin3, б) cos0,8, разложив функцию в степенной ряд.

7. Задана рекуррентная последовательность $a_i = 0{,}5a_{i-2} + 2a_{i-1}$. Записать восьмой член последовательности. Найти количество членов последовательности, меньших заданному числу $K = 100$ при значениях $a_1 = 1$ и $a_2 = 1{,}5$. Найти сумму семи первых членов.

8. Вычислить число размещений из n элементов по κ элементов: а) $n = 9$, $\kappa = 2$; б) $n = 12$, $\kappa = 4$; в) $n = 15$, $\kappa = 7$.

2.6 Лабораторная работа № 6 (2 часа).

Тема: «Процедуры и функции»

2.6.1 Цель работы: изучить процедуры и функции, используемые в языке программирования, рассмотреть их особенности, условия использования.

2.6.2 Задачи работы:

1. Изучение подпрограмм.
2. Особенности задания процедур и функций.
3. Решение прикладных задач с использованием подпрограмм.

2.6.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Язык программирования высокого уровня, компьютер

2.6.4 Описание (ход) работы:

Записать программу решения задач и проверить ее правильность на языке высокого уровня.

1. Ввести новую функцию tg(x).

2. Ввести новую функцию fconocr, которая вычисляет площадь между концентрическими окружностями радиусами R_1 и R_2.

3. Ввести новую функцию geron, которая вычисляет площадь треугольника по трем сторонам. Рассмотреть варианты введения неверных ответов.

4. Используя подпрограмму-процедуру, подсчитать количество цифр в целом числе.

5. Используя процедуру алгоритма Евклида, составить программу умножения двух обыкновенных дробей, результат – несократимая дробь. Например, при умножении 7/9 на 15/28 программа должна выдать число 5/12. Дополнить программу, чтобы она находила и деление двух обыкновенных дробей.

2.7 Лабораторная работа № 7 (2 часа).

Тема: «Структурированные типы данных»

2.7.1 Цель работы: изучить структурированные типы данных, используемые в языке программирования, рассмотреть их особенности, условия использования.

2.7.2 Задачи работы:

1. Рассмотреть строковый и множественный типы данных
2. Изучение массивов.
3. Решение задач с использованием структурных типов данных

2.7.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Язык программирования высокого уровня, компьютер

2.7.4 Описание (ход) работы:

Записать программу решения задач и проверить ее правильность на языке высокого уровня.

1. С помощью процедур и функций получить из строки «величина» строку «наличие». Рассмотреть несколько вариантов.
2. Составить программу, которая печатает вводимую строку в обратном порядке.
3. Записать программу, печатающую простые числа от 1 до 100.
4. Даны два множества натуральных чисел. Найти их объединение, пересечение и разность.
5. Дана строка текста на русском языке. Подсчитать в ней количество гласных букв. Определить каких букв больше: гласных или согласных.
6. В данной строке подсчитать количество цифр.

Составить программу в Паскале и записать ее в тетрадь:

1. В массиве A[*n*] записать оценки абитуриентов, полученные на первом экзамене. При поступлении в вуз абитуриенты, получившие двойку на первом экзамене, ко второму не допускаются. Подсчитать количество человек, не допущенных ко второму экзамену.
2. Дан массив *A* размерности *N* . Вывести его элементы с четными номерами.

Замечание: Результатом функции odd(x) целого типа значений является **True,** если x нечетное, **False**, если x – четное.

2.8 Лабораторная работа № 8 (2 часа).

Тема: «Использование модулей при разработке программ»

2.8.1 Цель работы: изучение графического модуля, рассмотреть его особенности, условия использования.

2.8.2 Задачи работы:

1. Рассмотрение возможностей графического модуля.
2. Составление программ с использованием модуля.
3. Построение фигур с использованием графических примитивов.

2.8.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Язык программирования высокого уровня, компьютер

2.8.4 Описание (ход) работы:

Записать программу решения задач и проверить ее правильность на языке высокого уровня.

1. Построить голову робота.
2. Записав программу в Паскале:

```
uses GraphABC;
begin
SetBrushColor(clGray);
Rectangle(100,100,140,160);
end.
```

определить следующие цвета:

1. clGray;

2. clBlack;
3. clRed;
4. clGreen;
5. clBlue;
6. clYellow;
7. clFuchsia;
8. clPurple;
9. clSkyBlue

3. Изобразить на экране рисунок (дается индивидуально).

2.9 Лабораторная работа № 9 (2 часа).

Тема: «Создание собственного модуля»

2.9.1 Цель работы: проверка умений решения практических задач.

2.9.2 Задачи работы:

1. Составление программы решения задачи.
2. Разработка и использование модуля.

2.9.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Язык программирования высокого уровня, компьютер

2.9.4 Описание (ход) работы:

Реализовать в виде модуля набор подпрограмм для выполнения основных арифметических операций над обыкновенными дробями.

При разработке модуля рекомендуется последовательность действий:

1) Спроектировать модуль, т.е. определить основные и вспомогательные подпрограммы и другие ресурсы.

2) Описать компоненты модуля.

3) Каждую подпрограмму целесообразно отладить отдельно, после чего добавить в модуль.

3. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

3.1 Практическое занятие № 1 (2 часа).

Тема: «Составление блок-схем»

3.1.1 Задание для работы:

1. Входной контроль.
2. Составление блок-схем

3.1.2 Краткое описание проводимого занятия:

Вариант входного контроля

1. Найти сторону правильного треугольника, вписанного в окружность длиной 6π.

2. Начав тренировки, спортсмен в первый день пробежал 8 км. Каждый день он увеличивал дневную норму на 10% нормы предыдущего дня (значения округлять до тысячных). Определить суммарный путь, который пробежит спортсмен за неделю.

3. Для острого угла выразить арксинус через арктангенс.

4. Вычислить определитель матрицы $\begin{pmatrix} 3 & 2 & -3 \\ -2 & 0 & 1 \\ 2 & -1 & 1 \end{pmatrix}$.

5. Заданы координаты трех вершин треугольника $A(-5;0)$, $B(7;9)$, $C(5;-5)$. Найти его площадь.

Составить блок-схемы следующих задач:

1) нахождение длины окружности и площади круга;

2) решения приведенного квадратного уравнения.

3.1.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.2 Практическое занятие № 2 (2 часа).

Тема: «Составление алгоритмов»

3.2.1 Задание для работы:

1. Составление линейных алгоритмов.
2. Составление разветвляющихся алгоритмов

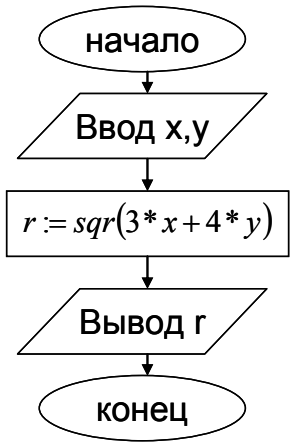
3.2.2 Краткое описание проводимого занятия:

1. Определить значение выражения $(3x+4y)^2$.

а) Запись решения задачи на алгоритмическом языке

алг значение
вещ x, y, r
нач
    ввод x, y
    $r := sqr(3x+4y)$
    Вывод r
кон

б) Запись алгоритма в виде блок-схемы



2. Определить частное от деления b на a, если a – положительное и среднее арифметическое в противном случае. Записать решение задачи на алгоритмическом языке и виде блок-схемы.

3. Записать решение задачи на алгоритмическом языке и виде блок-схемы: деление двух натуральных чисел с определением частного и остатка.

4. Записать решение задачи на алгоритмическом языке: нахождение значения функции $f(x)=\begin{cases}4x+5, & x\le 1\\ \ln x, & x>1\end{cases}$.

3.2.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.3 Практическое занятие № 3 (2 часа).

Тема: «Составление алгоритмов»

3.3.1 Задание для работы:

1. Составление линейных алгоритмов.

2. Составление разветвляющихся алгоритмов

3.3.2 Краткое описание проводимого занятия:

1. Записать алгоритм решения уравнения $ax^3 + bx = 0$ в виде блок-схемы и на алгоритмическом языке.

2. Дан радиус окружности. Составить алгоритм решения задачи на алгоритмическом языке, вычисляющую: а) длину окружности; б) диаметр окружности; в) площадь круга, ограниченного этой окружностью; г) сторону равностороннего треугольника, вписанного в данную окружность. Рассмотреть введение неверных данных и использовать комментарии при выводе каждой вычисляемой величины.

3. Составить блок-схему вычисления площади равнобокой трапеции по двум основаниям и углу при нижнем основании. Составить алгоритм решения на универсальном алгоритмическом языке.

3.3.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.4 Практическое занятие № 4 (2 часа).

Тема: «Логические основы алгоритмизации»

3.4.1 Задание для работы:

1. Логические операции с высказываниями.

2. Составление таблиц истинности.

3.4.2 Краткое описание проводимого занятия:

1. Указать ложные и истинные высказывания:

1) $2 \in \{x \mid 2x^3 - 3x^2 + 1,\ x \in R\}$;

2) 1 – простое число;

3) Число 1278 делится на 6;

4) Во множестве комплексных чисел существует приведенное квадратное уравнение, сумма корней которого равна свободному члену.

Составить всевозможные высказывания с помощью логических операций. Указать истинные из них.

2. Придумать имя, для которого истинно высказывание:

Если вторая буква имени согласная, *то четвертая буква имени гласная*

3. Идет чемпионат школы по гимнастике. Болельщики горячо обсуждают ход борьбы и высказывают немало предположений о будущих победителях.

- Первой будет Наташа, а Майя будет второй, - Сказал Сережа.

- Нет, Лида займет второе место, а Рита будет четвертой, - возразил Вова.

- Второй будет Наташа, а Рита будет третьей, - авторитетно заявил Толя.

Когда соревнования закончились, оказалось, что каждый их мальчиков ошибся один раз. Какие места в соревнованиях заняли Наташа, Майя, Лида и Рита?

4. Перед началом забегов зрители обсуждали скаковые возможности трех лучших лошадей с кличками «Абрек», «Ветер» и «Стрелок».

- Победит или «Абрек», или «Стрелок», - сказал один болельщик.

- Если «Абрек» будет вторым, то победу принесет «Ветер», - сказал другой болельщик.

Много вы понимаете в лошадях, - возмутился третий болельщик. – Вторым придет или «Ветер», или «Абрек».

- А я вам скажу, - вмешался четвертый болельщик, - что если «Абрек» придет третьим, то «Стрелок» не победит.

После забега выяснилось, что три лошади - «Абрек», «Ветер» и «Стрелок» - заняли три первых места, не деля между собой ни одного из мест, и что все четыре предсказания болельщиков были правильными. Как закончился забег?

3.4.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.5 Практическое занятие № 5 (2 часа).

Тема: «Языки и системы программирования»

3.5.1 Задание для работы:

1. Классификация языков программирования.
2. Аттестация 5 недель.

3.5.2 Краткое описание проводимого занятия:

1. Рассмотреть различные классификации языков программирования. Привести примеры, рассказать о создании каждого языка.

2. Машинно-зависимые языки – это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.).

Задание: Охарактеризовать язык программирования низкого уровня: дать определение, указать особенности и область применимости: машинный язык, автокод, ассемблер.

Аттестация 5 недель.

1. Алгоритм может быть записан …

1) в словесной форме; 2) графическим способом;

3) на алгоритмическом языке; 4) в семантической форме.

2. Вывод данных на экран обеспечивают процедуры …

1) Read; 2) Print; 3) WriteLn; 4) ReadLn; 5) Write.

3. Символ-разделитель переменных в Паскале …

1) точка; 2) двоеточие; 3) точка с запятой; 4) пробел; 5) запятая.

4. Запись 5.7E-04 означает …

1) $(5{,}7)^{-4}$; 2) $5{,}7-0{,}4$; 3) $5{,}7\cdot 10^{-4}$; 4) $5{,}7\cdot 10^{-40}$.

5. Запись выражения $\frac{1}{\sqrt[4]{\sin x}}$ на Паскале будет иметь вид …

1) $\exp((-1/4)*Ln(\sin(x)))$; 2) $\exp((-4)*Ln(\sin(x)))$;

3) $\exp(\sin(x)*Ln(-0.25))$; 4) $\exp((-1/4)*Ln(\sin x))$.

6. При $x=10$ значение выражения $9*x\ div\ 8 \bmod 6$, вычисленного в Паскале, равно …

ОТВЕТ: ___

7. Для округления действительного числа x до сотых нужно использовать оператор …

1) WriteLn(x:12:2); 2) WriteLn(x:2); 3) WriteLn(x:2:8); 4) WriteLn(x:12:100).

8. Оператор WriteLn(x:8) при $x=7945.26$ выведет на экран …

1) 7945.26; 2) _7945.3; 3) _ _7945.3; 4) _7945.26

9.
```
Var x;r: Real
Begin ReadLn(x);
s=180/pi*arctg(sqrt(1-x^2)/x);
Write(s);
```

End.

В записи программы допущены … ошибки (ошибок).

Ответ: ___

3.5.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.6 Практическое занятие № 6 (2 часа).

Тема: «Языки и системы программирования»

3.6.1 Задание для работы:

1. Классификация языков программирования.
2. Системы программирования

3.6.2 Краткое описание проводимого занятия:

1. Охарактеризовать языки программирования высокого уровня: дать определение, указать особенности и область применимости.

1) Язык **Фортран** был ориентирован на научно-технические расчеты математического характера, для научных и инженерных вычислений. **Фортра́н** (*Fortran*) — первый язык программирования высокого уровня, имеющий транслятор. Создан в период с 1954 по 1957 год группой программистов под руководством Джона Бэкуса. Название Fortran является сокращением от **FOR**mula **TRAN**slator (переводчик формул). Одно из преимуществ современного Фортрана — большое количество написанных на нём программ и библиотек подпрограмм. Среди учёных, например, ходит такая присказка, что любая математическая задача уже имеет решение на Фортране, и, действительно, можно найти среди тысяч фортрановских пакетов и пакет для перемножения матриц, и пакет для решения сложных интегральных уравнений, и многие, многие другие. Современный Фортран (Fortran 95 и Fortran 2003) приобрёл черты, необходимые для эффективного программирования для новых вычислительных архитектур, позволяет применять современные технологии программирования.

2) В 1960 году командой во главе с Петером Науром (Peter Naur) был создан язык программирования **Algol.** Этот язык дал начало целому семейству Алгол-подобных языков (важнейший представитель — Pascal). В 1968 году появилась новая версия языка. Она не нашла столь широкого практического применения, как первая версия, но была весьма популярна в кругах теоретиков. Язык был достаточно интересен, так как обладал многими уникальными на так момент характеристиками.

3) В 1960 году был создан язык программирования **Cobol.** Он задумывался как язык для программирования экономических задач, и он стал таковым. На Коболе написаны тысячи прикладных коммерческих систем. Отличительной особенностью языка является возможность эффективной работы с большими массивами данных, что характерно именно коммерческих приложений. Популярность Кобола столь высока, что даже сейчас, при всех его недостатках (по структуре и замыслу Кобол во многом напоминает Фортран) появляются новые его диалекты и реализации.

4) В 1963 - 64 годах профессорами Дартмутского колледжа Томасом Курцем и Джоном Кемени был создан язык программирования **BASIC** (Beginners' All-Purpose Symbolic Instruction Code - универсальный код символических инструкций для начинающих). Язык задумывался в первую очередь как средство обучения и как первый изучаемый язык программирования. Он предназначался для более «простых» пользователей, не столько заинтересованных в скорости исполнения программ, сколько просто в возможности использовать компьютер для решения своих задач, не имея специальной подготовки. Надо сказать, что BASIC действительно стал языком, на котором учатся программировать (по

крайней мере, так было еще несколько лет назад; сейчас эта роль отходит к Pascal). Было создано несколько мощных реализаций BASIC, поддерживающих самые современные концепции программирования. Язык был основан частично на Фортране II и частично на Алголе 60, с добавлениями, делающими его удобным для работы в режиме разделения времени и, позднее, обработки текста и матричной арифметики.

5) В 1969 году был создан язык **SETL** — язык для описания операций над множествами. Основной структурой данных в языке является множество, а операции аналогичны математическим операциям над множествами. Полезен при написании программ, имеющих дело со сложными абстрактными объектами.

6) Пролог (фр. ***Pro****grammation en* ***Log****ique*) — язык и система логического программирования, основанные на языке предикатов математической логики, представляющей собой подмножество логики предикатов первого порядка (1971).

Основными понятиями в языке Пролог являются факты, правила логического вывода и запросы, позволяющие описывать базы знаний, процедуры логического вывода и принятия решений. Факты в языке Пролог описываются логическими предикатами с конкретными значениями. Правила в Прологе записываются в форме правил логического вывода с логическими заключениями и списком логических условий.

Особую роль в интерпретаторе Пролога играют конкретные запросы к базам знаний, на которые система логического программирования генерирует ответы «истина» и «ложь».

7) Значительным событием в истории языков программирования стало создание в 1971 году языка **Паскаль** швейцарским профессором Никлаусом Виртом. Один из наиболее известных языков программирования, используется для обучения программированию в старших классах и на первых курсах вузов, является базой для ряда других языков. Далее разрабатывается система программирования Турбо Паскаль, включающая язык, транслятор и операционную оболочку, обеспечивающую пользователю удобство работы.

8) В 1972 году Керниганом и Ритчи был создан язык программирования **Си**. Он создавался как язык для разработки операционных систем. Си часто называют «переносимым ассемблером», имея в виду то, что он позволяет работать с данными практически так же эффективно, как на ассемблере, предоставляя при этом структурированные управляющие конструкции и абстракции высокого уровня (структуры и массивы). Именно с этим связана его огромная популярность. Однако компилятор С очень слабо контролирует типы, поэтому очень легко написать внешне совершенно правильную, но логически ошибочную программу. В 1986 году Бьярн Страуструп создал первую версию языка С++, добавив в язык С объектно-ориентированные черты, взятые из Simula, и исправив некоторые ошибки и неудачные решения языка. С++ продолжает совершенствоваться и в настоящее время, так в 1998 году вышла новая (третья) версия стандарта, содержащая в себе некоторые довольно существенные изменения. Язык стал основой для разработки современных больших и сложных проектов.

3.6.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.7 Практическое занятие № 7 (2 часа).

Тема: «Составление программ линейной структуры»

3.7.1 Задание для работы:

1. Составление программ линейной структуры

3.7.2 Краткое описание проводимого занятия:

1. Составить программу на языке программирования высокого уровня: вычисление определителя четвертого порядка.

2. Даны длины a катета и c гипотенузы прямоугольного треугольника. Составить программу, вычисляющую периметр и площадь этого треугольника. Найти в радианной мере острые углы треугольника.

3. Записать решение задачи на алгоритмическом языке и виде блок-схемы: деление двух натуральных чисел с определением частного и остатка.

3.7.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.8 Практическое занятие № 8 (2 часа).

Тема: «Составление программ разветвляющейся структуры»

3.8.1 Задание для работы:

1. Составление программ разветвляющейся структуры
2. Логические выражения в управляющих операторах.

3.8.2 Краткое описание проводимого занятия:

1. Записать программу определения количества целых чисел среди трех введенных.
2. Записать программу решения уравнения $ax^3 + bx = 0$.
3. Составить линейную программу, печатающую значение true, если указанное высказывание является истинным, и false – в противном случае: график функции $y = ax^2 + bx + c$ проходит через заданную точку с координатами $(m;n)$.

3.8.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.9 Практическое занятие № 9 (2 часа).

Тема: «Составление программ циклической структуры»

3.9.1 Задание для работы:

1. Использование циклов с параметром при составлении программ.
2. Аттестация 9 недель.

3.9.2 Краткое описание проводимого занятия:

1. Цикл с параметром: описание, особенности, область применения.
2. Составить алгоритм решения задачи: найти сумму членов арифметической прогрессии, если известны ее первый член, знаменатель и число членов прогрессии.
3. Составить программу нахождения суммы первых п нечетных чисел.

Аттестация 9 недель

1. Конъюнкция в Паскале записывается словом …

1) Not; 2) Or; 3) And; 4) Xor.

2. Математическому неравенству $1 \le x \le 7$ соответствует в Паскале правильная запись…

1) (x>=1) Or (x<=7); 2) (1<=x) Or (x<=7); 3) (x<=1) And (x<=7); 4) (1<=x) And (x<=7).

3. Результатом выполнения фрагмента программы

```
a:=11;
b:=9;
k:=3+a+sqrt(b);
if (b>k) Then k:=k+6
 Else k:=k-4;
WriteLn(k);
```

будет значение …
ОТВЕТ: ___
4. При $a = 7$ и $b = 4$ логическая переменная d получит значение *True* в выражениях …
1) d:=(a>b); 2) d:=(b mod a=0); 3) d:=(sqr(b)-a=9); 4) d:=(a*b>0) or (b>6).

3.9.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.10 Практическое занятие № 10 (2 часа).

Тема: «Составление программ циклической структуры»

3.10.1 Задание для работы:

1. Использование циклов с предусловием при составлении программ
2.Составление программ циклической структуры с постусловием

3.10.2 Краткое описание проводимого занятия:

1. Записать рекуррентную формулу для последовательности:

а) 2, 6, 10, 14, 18 … б) 3, 3/5, 3/25, 3/125, …; в) 1, 2, 6, 24, … г) 1, $\frac{x}{1!}, \frac{x^2}{2!}, \frac{x^3}{3!}, \ldots$

2. Вычислить: а) sin 3, б) cos 0,8, разложив функцию в степенной ряд.

3. Задана рекуррентная последовательность $a_i = 0{,}5a_{i-2} + 2a_{i-1}$. Записать восьмой член последовательности. Найти количество членов последовательности, меньших заданному числу $K = 100$ при значениях $a_1 = 1$ и $a_2 = 1{,}5$. Найти сумму семи первых членов.

4. Вычислить число размещений из n элементов по κ элементов: а) $n = 9$, $\kappa = 2$; б) $n = 12$, $\kappa = 4$; в) $n = 15$, $\kappa = 7$.

3.10.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.11 Практическое занятие № 11 (2 часа).

Тема: «Процедуры»

3.11.1 Задание для работы:

1.Понятие подпрограммы. Процедуры и функции, их сущность, назначение, различие.
2.Организация процедур, стандартные процедуры.

3.11.2 Краткое описание проводимого занятия:

1. Дать сравнительную характеристику двум видам подпрограмм: процедуры и функции.

2. С помощью процедуры записать программу нахождения наибольшего общего делителя четырех чисел.

3.11.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.12 Практическое занятие № 12 (2 часа).

Тема: «Функции»

3.12.1 Задание для работы:

1. Функции, определенные пользователем: синтаксис, передача аргументов. Формальные и фактические параметры.

3.12.2 Краткое описание проводимого занятия:

1. Организация функций, ее особенности. Стандартные функции.

2. Записать алгоритм Евклида, используя функцию.

3. Составить программу умножения и деления двух дробей. Результатом должна быть несократимая дробь.

3.12.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.13 Практическое занятие № 13 (2 часа).

Тема: «Структурированные типы данных»

3.13.1 Задание для работы:

1. Строковый тип данных. Основные операции над строками.

2. Множественный тип данных. Операции отношения и вхождения.

3. Аттестация 13 недель.

3.13.2 Краткое описание проводимого занятия:

1. Даны две строки: S1:= 'нос', S2:= 'самостоятельность'. Определить результат выполнения функций: а) Pos (S1, S2); 2) Copy(S1,3,1)+Copy(S2,5,4); 3) Length(S2). С помощью процедур получить строки: 4) 'стол'; 5) 'кость'. Записать строку в обратной последовательности символов.

2. Какие задачи решают следующие программы?

а)
```
Var A: Set of Byte;
  P: Integer;   C,F: Boolean;
  Begin  A:=[3,4,10,12,15];
  C:=A<=[1..15];   Writeln(C);
  readln(P);  F:=(P in A);
        Writeln(F);        End.
```

б)
```
Var S:String; I,K:Byte;
 Begin  ReadLn(S); K:=0;
 For I:=1 To Length(S) Do
 If S[I] in ['.','!',':'] Then K:=K+1;
 WriteLn(K);
       End.
```

3. Даны два множества натуральных чисел. Найти их объединение, пересечение и разность. Запись программы на языке высокого уровня.

Аттестация 13 недель

1. Даны два множества: A:= [32, 35, 39, 34, 35, 32, 39, 38]; B:= [32, 39, 35, 32, 39, 38]. Результатом будет логическая величина «False» для отношений …

1) A=B; 2) A<>B; 3) A<=B; 4) A>=B.

2. Даны два множества A:=[10, 30, 60, 80, 110] и B:=[10, 80, 120, 50, 30]. Тогда результатом операции $B - A$ в Паскале является …

1) [10, 30, 50, 60, 80, 110, 120]; 2) [10, 30, 80]; 3) [60, 110]; +4) [50, 120] .

3. Знак операции объединения в Паскале …

1) * 2) + 3) $\cap$ 4) –

4. Правильный вариант задания процедуры:
1) Var A,B: Integer; C: Real;
Procedure Col(n:Integer; Var k,m:Integer);
2) Var A,B: Integer; C: Real;
Procedure Col(n:Integer; Var k,m:Real);
3) Var A,B: Integer; C: Real;
Procedure Col(n,k:Integer; Var m:Real);
4) Var A,B: Integer; C: Real;
Procedure Col(n,k:Integer; Var m,s:Real).
5. Результатом программы

```
Var C: Boolean;
begin
C:=[3,4,7,1,2,10]<=[1..10];
Writeln(C);
End.
```

является …
1) True; 2) False; 3) Ошибка; 4) [5,6,8,9].
6. Результатом выполнения программы

```
Var x,y,C: Real;
Function fpol(A,B:Real):Real;
Begin fpol:=(exp(A*ln(B)));
End;
Begin ReadLn(x); ReadLn(y);
 C:=fpol(x,y); WriteLn(C);
End.
```

при x=3 и y=4 будет …
ОТВЕТ: ___
7. Программа:

```
 Var S:String; I,K:Byte;
Begin
ReadLn(S); K:=1;
For I:=1 To Length(S) Do
If S[I] in [' '] Then K:=K+1;
WriteLn(K);
End.
```

решает задачу …
ОТВЕТ: _____

3.13.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.14 Практическое занятие № 14 (2 часа).

Тема: «Массивы»

3.14.1 Задание для работы:

1. Одномерные и двумерные массивы.
2. Действия над массивом. Обработка массива.

3.14.2 Краткое описание проводимого занятия:

1. Понятие массива, его способы записи. Типы элементов массива.
2. Описание массива. Ввод и вывод элементов массива.

3. Задан массив чисел.

1) Определить среднюю численность населения за указанный период.

2) Определить среднюю численность возрастных групп за указанный период.

3) Составить массив отклонения численности населения от среднего значения и определить год с максимальным приростом населения.

4) Найти долю основных возрастных групп в общей численности населения в процентах.

РАСПРЕДЕЛЕНИЕ НАСЕЛЕНИЯ ОРЕНБУРГСКОЙ ОБЛАСТИ ПО ОСНОВНЫМ ВОЗРАСТНЫМ ГРУППАМ (ЧЕЛОВЕК)

Годы	в том числе в возрасте		
	моложе трудоспо-собного	трудоспо-собном	старше трудоспо-собного
1994	559742	1233327	419969
1995	550207	1241434	426411
1996	536319	1245775	433842
1997	522626	1258518	436938
1998	507061	1274892	435605
1999	487070	1292039	432095
2000	469982	1306591	427043
2001	451779	1312745	425352
2002	430170	1324637	421193
2003	409814	1337453	415278

3.14.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.15 Практическое занятие № 15 (2 часа).

Тема: «Файлы и файловые переменные»

3.15.1 Задание для работы:

1.Типы файлов. Организация доступа к файлам.

2.Файлы последовательного доступа. Открытие и закрытие файла последовательного доступа.

3.Запись в файл и чтение из файла последовательного доступа.

3.15.2 Краткое описание проводимого занятия:

1. Понятие файла. Файловый тип переменной.

2. Запись в файл и чтение из файла. Внешние файлы.

3. Создать файл, содержащий среднесуточные температуры за некоторое количество дней. Найти самую высокую и самую низкую температуру. Определить среднюю температуру за данное количество дней.

4. Дан файл, содержащий некоторый текст. Подсчитать количество строк и число слов в этом тексте.

3.15.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы

организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.16 Практическое занятие № 16 (2 часа).

Тема: «Стандартные модули языка программирования»

3.16.1 Задание для работы:

1. Работа с графическим модулем.
2. Графические примитивы.
3. Построение графика функции.

3.16.2 Краткое описание проводимого занятия:

1. Подключение к графическому модулю, его возможности.
2. Графические режимы экрана. Графические координаты.
3. Перечислить графические примитивы с указанием процедур их рисования.
4. Составить программу рисования шахматного поля.
5. Точечный и кусочно-линейный метод построения графика функции. Их особенности и применение.
6. Составить программу универсального построения графика функции.

3.16.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.17 Практическое занятие № 17 (2 часа).

Тема: «Программирование модуля»

3.17.1 Задание для работы:

1.Внешние подпрограммы и модули.
2.Создание и использование модуля.

3.17.2 Краткое описание проводимого занятия:

1. Организация внешних подпрограмм. Введение дополнительных функций.
2. Записать программу, вычисляющую сумму первой и последней цифр натурального числа. Использовать функцию возведения натурального числа в натуральную степень.
3. Структура, разработка, компиляция и использование модуля.
4. Создать дополнительные функции для модуля «Обыкновенные дроби»: знак числа, правильная дробь, конечная десятичная дробь, период дроби.

3.17.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.

3.18 Практическое занятие № 18 (2 часа).

Тема: «Методы построения алгоритмов»

3.18.1 Задание для работы:

1. Рекурсивные методы.
2. Метод перебора в задачах поиска.
3. Методы сортировки данных.
4. Аттестация 18 недель.

3.18.2 Краткое описание проводимого занятия:

1. Составить алгоритм решения задачи «Ханойская башня». На площадке А находится пирамида, составленная из дисков уменьшающегося от основания к вершине размера. Эту пирамиду в том же виде требуется переместить на площадку В, используя вспомогательную площадку С, причем перекладывать можно только по одному диску, взятому сверху пирамиды, и класть диск можно либо на основание площадки, либо на диск большего размера.

2. Составить алгоритм прохождения лабиринта: Дан лабиринт, оказавшись внутри которого нужно найти выход наружу. Перемещаться можно только в горизонтальном и вертикальном направлениях.

3. Составить алгоритм быстрой сортировки.

Аттестация 18 недель

1. Задан массив Var P: Array [1..6, 3..5, 4..7] Of Real. Тогда объем памяти, зарезервированного для данного массива, составляет … байтов.

ОТВЕТ: ___

2. Окружность радиусом в 25 пикселей можно изобразить с помощью процедуры …

1) Circle (50,25,70); 2) Circle (25,50,70);
3) Circle (50,70,25); 4) Circle (80,100,50).

3. Правильным заданием процедуры «поставить точку» является …

1) PutPixel (20, 50);
2) PutPixel (20, 50, clAqua);
3) PutPixel (clAqua, 20, 50);
4) PutPixel (20, 50, 1).

4. Отрезок задан процедурой Line(200,100,350,400). Тогда параллельный ему отрезок задает фрагмент программы …

1) Line(300,200,450,400); 2) Line(200,100,450,400);
3) Line(200,100,450,500); 4) Line(300,100,450,400).

5. В Паскале две концентрические окружности задает фрагмент программы …

1) Circle (150,100,50); Circle (140,90,70);
2) Circle (150,100,70); Circle (150,100,50);
3) Circle(150,100,50); Circle(130,100,50);
4) Circle(150,100,50); Circle(150,70,50).

6. Дан массив T.

месяц	1	2	3	4	5	6
доход	10500	12300	11600	13800	12000	11400

В Паскале для введения значений используется следующий вариант …

1) ReadLn(T[1] .. T[6]);
2) ReadLn(10500 .. 11400);
3) For I:=1 To 6 Do ReadLn(10500 .. 11400);
4) For I:=1 To 6 Do ReadLn(T[I]).

7. Язык программирования Pascal создал …

1) Б. Паскаль; 2) М. Фортран; 3) Н. Вирт; 4) Д.Бэкус.

8. Основной принцип модульного программирования – это …

1) «скрой информацию от других»; 2) «разделяй и властвуй»;
3) «упрощай программу»; 4) «не используй алгоритмические структуры».

3.18.3 Результаты и выводы:

В результате проведения практического занятия студент освоит основные понятия алгоритмизации и программирования, основные алгоритмические конструкции, способы организации данных, синтаксис и семантику языка программирования высокого уровня; научится решать исследовательские и проектные задачи с использованием компьютеров; овладеет методами построения алгоритмов.