1 курс

ПЛАН – КОНСПЕКТ
проведения занятия по дисциплине «Информатика»

Раздел 4. «Основы алгоритмизации и программирования.»

Тема 4.1: «Общие принципы построения базовых алгоритмических структур в среде программирования.»

часть 1

Подготовил: преподаватель
В.Н. Борисов

Рязань 2023

Вопросы занятия:

1. Понятие и свойства алгоритмов.
2. Способы описания алгоритма.
3. Таблица блочных символов.
4. Базовые алгоритмические структуры.
5. Расчет результатов выполнения алгоритма.

Время проведения занятия – 2 часа.

Первый вопрос: Понятие и свойства алгоритмов.

На ЭВМ могут решаться задачи различного характера, например: научно-технические; управления производственными процессами; разработки системного, программного обеспечения; обучениями др. Значительную долю в указанном перечне составляют научно-технические задачи. В процессе подготовки и решения их на ЭВМ можно выделить следующие этапы:

- постановка задачи;
- математическое описание задачи;
- выбор и обоснование метода решения;
- алгоритмизация вычислительного процесса;
- составление программы;
- отладка программы;
- решение задачи на ЭВМ и анализ результатов.

Алгоритм – *заранее заданное понятное и точное предписание возможному исполнителю совершить определенную последовательность действий для получения решения задачи за конечное число шагов.*

Это – не определение в математическом смысле слова, а, скорее, описание интуитивного понятия алгоритма, раскрывающее его сущность.

Понятие алгоритма является не только одним из главных понятий математики, но одним из главных понятий современной науки. Основные свойства алгоритмов следующие:

1. **Понятность** для исполнителя – исполнитель алгоритма должен понимать, как его выполнять. Иными словами, имея алгоритм и произвольный вариант исходных данных, исполнитель должен знать, как надо действовать для выполнения этого алгоритма.

2. **Дискретность** (прерывность, раздельность) – алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов (этапов).

3. **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.

4. **Результативность** (или конечность) состоит в том, что за конечное число шагов алгоритм либо должен приводить к решению задачи, либо после конечного числа шагов останавливаться из–за невозможности получить решение с выдачей соответствующего сообщения, либо неограниченно продолжаться в течение времени, отведенного для исполнения алгоритма, с выдачей промежуточных результатов.

5. **Массовость** означает, что алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

Второй вопрос: Способы описания алгоритма.

На практике наиболее распространены следующие формы представления алгоритмов:

· *словесная* (запись на естественном языке);

· *графическая* (изображения из графических символов);

· *псевдокоды* (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);

· *программная* (тексты на языках программирования).

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

Словесный способ не имеет широкого распространения, так как такие описания:

· строго не формализуемы;

· страдают многословностью записей;

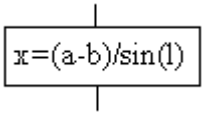
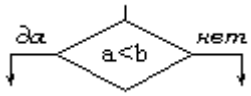
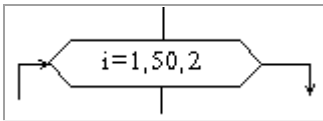
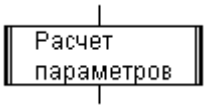
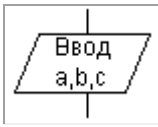
· допускают неоднозначность толкования отдельных предписаний.

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным. *При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.*

Такое графическое представление называется схемой алгоритма или ***блок–схемой***. В блок–схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде *блочного символа*. Блочные символы соединяются *линиями переходов*, определяющими очередность выполнения действий.

Третий вопрос: <u>Таблица блочных символов.</u>

В таблице приведены наиболее часто употребляемые символы

Название символа	Обозначение и пример заполнения	Пояснение
Процесс		Вычислительное действие или последовательность действий
Решение		Проверка условий
Модификация		Начало цикла
Предопределенный процесс		Вычисления по подпрограмме, стандартной подпрограмме
Ввод–вывод		Ввод–вывод в общем виде
Пуск–останов		Начало, конец алгоритма, вход и выход в подпрограмму
Документ		Вывод результатов на печать

Блок **"процесс"** применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для

улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.

Блок **"решение"** используется для обозначения переходов управления по условию. В каждом блоке "решение" должны быть указаны вопрос, условие или сравнение, которые он определяет.

Блок **"модификация"** используется для организации циклических конструкций. (Слово модификация означает видоизменение, преобразование). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.

Блок **"предопределенный процесс"** используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

Четвертый вопрос: <u>Базовые алгоритмические структуры.</u>

Алгоритмы можно представлять как некоторые структуры, состоящие из отдельных **базовых** (т.е. основных) **элементов**. Естественно, что при таком подходе к алгоритмам изучение основных принципов их конструирования должно начинаться с изучения этих базовых элементов. Для их описания будем использовать язык схем алгоритмов и простейший псевдокод.

Логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур: следование, ветвление, цикл. Характерной особенностью базовых структур является наличие в них **одного входа и одного выхода.**

1.***Базовая структура "следование".*** Образуется последовательностью действий, следующих одно за другим:

Псевдокод	Язык блок–схем
действие 1 действие 2 . . . . . действие n	

2.***Базовая структура "ветвление"***. Обеспечивает в зависимости от результата проверки условия (**да** или **нет**) выбор одного из альтернативных путей работы алгоритма. Каждый из путей ведет к *общему выходу*, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран.

Структура *ветвление* существует в четырех основных вариантах:

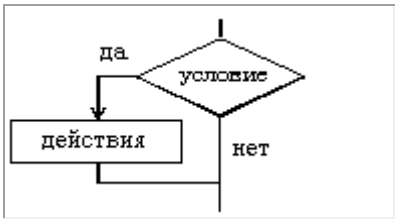
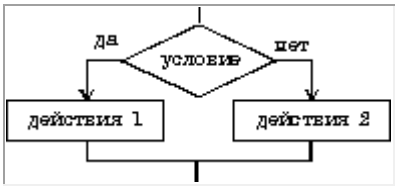
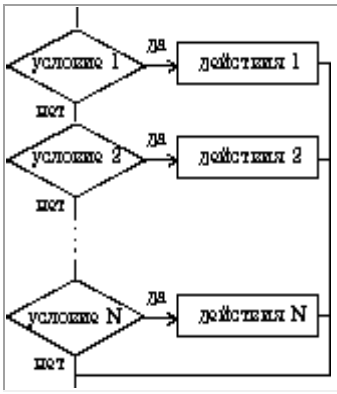
· если–то;

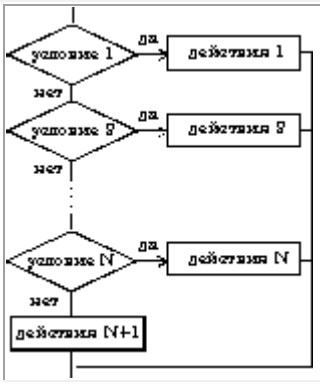
· если–то–иначе;

· выбор;

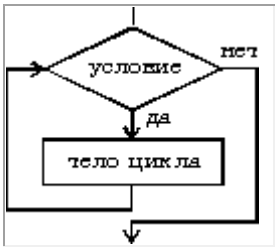
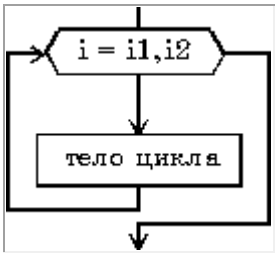
· выбор–иначе.

.

Псевдокод	Язык блок–схем
1. если–то	
если условие то действия все	
2. если–то–иначе	
если условие то действия 1 иначе действия 2 все	
3. выбор	
выбор при условие 1: действия 1 при условие 2: действия 2 . . . . . при условие N: действия N все	
4. выбор–иначе	

выбор при условие 1: действия 1 при условие 2: действия 2 . . . . . при условие N: действия N иначе действия N+1 все	

3. ***Базовая структура "цикл" (повторение).*** Обеспечивает *многократное выполнение некоторой совокупности действий***,** которая называется *телом цикла*. Основные разновидности циклов представлены в таблице:

Псевдокод	Язык блок–схем
Цикл типа пока. Предписывает выполнять тело цикла до тех пор, пока выполняется условие, записанное после слова пока.	
нц пока условие тело цикла (последовательность действий) кц	
Цикл типа для. Предписывает выполнять тело цикла для всех значений некоторой переменной (параметра цикла) в заданном диапазоне.	
нц для i от i1 до i2 тело цикла (последовательность действий) кц	

Алфавит учебного алгоритмического языка является открытым. В него могут быть введены любые понятные всем символы: русские и латинские буквы, знаки математических операций, знаки отношений, специальные знаки и т. д. Кроме алфавита, в алгоритмической нотации определяются служебные слова, которые являются неделимыми. Служебные слова обычно выделяются жирным шрифтом или подчеркиванием. К служебным словам относятся:

алг — заголовок алгоритма	нц — начало цикла	знач
нач — начало алгоритма	кц — конец цикла	и

кон — конец алгоритма	дано	или
арг — аргумент	надо	не
рез — результат	если	да
цел — целый	то	нет
сим — символьный	иначе	при
лит — литерный	всё	выбор
лог — логический	пока	утв
вещ — вещественный	для	ввод
таб — таблица	от	вывод
длин — длина	до	

Общий вид записи алгоритма на псевдокоде:
алг — название алгоритма (аргументы и результаты)
дано — условие применимости алгоритма
надо — цель выполнения алгоритма
нач — описание промежуточных величин
последовательность команд (тело алгоритма)

Пятый вопрос: Расчет результатов выполнения алгоритма.

Алгоритмизация вычислительного процесса.

На данном этапе составляется алгоритм решения задачи согласно действиям, задаваемым выбранным методом решения. Процесс обработки данных разбивается на отдельные относительно самостоятельные блоки и устанавливается последовательность выполнения блоков. Разрабатывается блок-схема алгоритма.

Составление программы.

При составлении программы алгоритм решения задачи переводится на конкретный язык программирования. Для программирования обычно используются языки высокого уровня, поэтомусоставленная программа требует перевода ее на машинный язык ЭВМ. После такого перевода выполняется уже соответствующая машинная программа.

Отладка программы.

Отладка заключается в поискеи устранении синтаксических и логических (семантических, алгоритмических) ошибок в программе.

В ходе синтаксического контроля программы транслятором (подразд. 2.5) выявляются конструкции и сочетаниясимволов, недопустимые с точки зрения правил их построения или написания, принятых в данном языке. Сообщения об ошибках ЭВМ выдает программисту, при этом вид и форма выдачи подобных сообщений зависят от типа языка и версии используемого транслятора.

После устранения синтаксических ошибок проверяется логика работы программы в процессе ее выполнения с конкретными исходными данными. Для этого используются специальные методы, например, в программе выбираются контрольные точки, для которых вручную рассчитываются промежуточные ре зультаты. Эти результаты св еряются со значениями, получаемыми ЭВМ в данных точках при выполнении отлаживаемой программы. Кроме того, для поиска ошибок могут быть использованы отладчики, выполняющие специальныедействия на этапе отладки, например, удаление, замена или вставка отдельных операторов или целых фрагментов программы, вывод и ли изменение значений заданных переменных.

Решение задачи на ЭВМ и анализ результатов.

После отладки программу можно использовать для решения прикладной задачи. При этом обычно выполняется многократное решение задачи на ЭВМ для различных н аборов исходных данных. Получаемые результаты интерпретируются и анализируются специалистом или пользователем, поставившим задачу.

Разработанная программа длительного использования устанавливается на ЭВМ, как правило, в виде готовой к выполнению машинной программы. К программе прилагается документация, включая инструкцию для пользователя.

Часто при установке программы на диск для ее последующего использования устанавливаются, помимо файлов с исполняемым кодом, различные вспомогательные программы (утилиты, справочники, настройщики и т. д.), а также необходимые для работы программ разного рода файлы с текстовой, графической, звуковой и другой информацией.

Расчет результатов выполнения алгоритма.

Эффективность. Рассматривая различные алгоритмы решения одной и той же задачи, полезно проанализировать, сколько вычислительных ресурсов они требуют (время работы, память), и выбрать наиболее эффективный.

Под временем работы (running time) алгоритма будем подразумевать число элементарных шагов, которые он выполняет. Положим, что одна строка псевдокода требует не более чем фиксированного числа операций (если только это не словесное

описание каких-то сложных действий – типа «отсортировать все точки по ***x***-координате»). Следует также различать *вызов* (*call*) процедуры (на который уходит фиксированное число операций) и её исполнение (*execution*), которое может быть долгим.

Сложность алгоритма – это величина, отражающая порядок величины требуемого ресурса (времени или дополнительной памяти) в зависимости от размерности задачи.

Анализ трудоёмкости алгоритмов.

В качестве критерия оптимальности алгоритма выбирается трудоемкость алгоритма, понимаемая как количество элементарных операций, которые необходимо выполнить для решения задачи с помощью данного алгоритма.

Функцией трудоемкости называется отношение, связывающие входные данные алгоритма с количеством элементарных операций.

Одним из упрощенных видов анализа, используемых на практике, является асимптотический анализ алгоритмов. Целью асимптотического анализа является сравнение затрат времени и других ресурсов различными алгоритмами, предназначенными для решения одной и той же задачи, при больших объемах входных данных.

Используемая в асимптотическом анализе оценка функции трудоёмкости, называемая сложностью алгоритма, позволяет определить, как быстро растет трудоёмкость алгоритма с увеличением объема данных. В асимптотическом анализе алгоритмов используются обозначения, принятые в математическом асимптотическом анализе