1) Основные характеристики алгоритмов.

Алгори́тм — набор инструкций, описывающих порядок действий исполнителя для достижения некоторого результата.

Для решения одной и той же задачи, как правило, можно использовать различные алгоритмы. В связи с этим, возникает необходимость сравнивать их между собой, и для этого нужны определенные критерии качества алгоритмов.

Временные характеристики алгоритма определяют длительность решения или временную сложность .

Длительность решения часто выражается в единицах времени, но удобнее ее выражать через количество операций, так как количество операций не зависит от быстродействия конкретной машины.

Временной сложностью алгоритма называется зависимость времени счета, затрачиваемого на получение результатов от объема исходных данных.

Временная сложность позволяет определить наибольший размер задачи, которую можно решить с помощью данного алгоритма на ПК. Каждый алгоритм можно характеризовать функцией J[n), выражающей скорость роста объема вычислений при увеличении размерности задачи - п. Если эта зависимость имеет линейный или полиномиальный характер, то алгоритм считается «хорошим», если экспоненциальный - «плохим».

Объемные характеристики алгоритма определяют его информационную сложность. Информационная сложность связана со сложностью описания, накопления и хранения исходных, промежуточных и результирующих данных при решении определенной задачи.

Объем текста алгоритма (программы) определяется количеством операторов, использованных для записи алгоритма.

Объем внутренней и внешней памяти необходимой для хранения данных и программ при использовании данного алгоритма определяется на основании расчетов или опытным путем.

Сложность структуры алгоритма определяется количеством маршрутов, по которым может реализовываться процесс вычислений и сложностью каждого маршрута.

Алгоритмы обладают целым рядом свойств: дискретностью, точностью, результативностью, массовостью.

Свойства алгоритма - набор свойств, отличающих алгоритм от любых предписаний и обеспечивающих его автоматическое исполнение.

Детерминированность (определенность, точность, однозначность). Это свойство заключается в том, что при задании одних и тех же исходных данных несколько раз алгоритм будет выполняться абсолютно одинаково и всегда будет получен один и тот же результат. Свойство детерминированности проявляется также и в том, что на каждом шаге выполнения алгоритма всегда точно известно, что делать дальше, а каждое действие однозначно понятно исполнителю и не может быть истолковано неопределенно. Благодаря этому свойству выполнение алгоритма носит механический характер.

Массовость - выражается в том, что с помощью алгоритма можно решать не одну конкретную задачу, а любую задачу из некоторого класса однотипных задач при всех допустимых значениях исходных данных.

Результативность (направленность) - означает, что выполнение алгоритма обязательно должно привести к решению поставленной задачи, либо к сообщению о том, что при заданных исходных величинах задачу решить невозможно. Алгоритмический процесс не может обрываться безрезультатно.

Дискретность - означает, что алгоритм состоит из последовательности отдельных шагов - элементарных действий, выполнение которых не представляет сложности. Именно благодаря этому свойству алгоритм может быть реализован на ЭВМ.

Конечность (финитность) - заключается в том, что последовательность элементарных действий алгоритма не может быть бесконечной, неограниченной, хотя может быть очень большой (если требуется, например, большая точность вычислений).

Корректность - означает, что если алгоритм создан для решения определенной задачи, то для всех исходных данных он должен всегда давать правильный результат и ни для каких исходных данных не будет получен неправильный результат. Если хотя бы один из полученных результатов противоречит хотя бы одному из ранее установленных и получивших признание фактов, алгоритм нельзя признать корректным.

2) Понятие о процедурно-ориентированных языках и объектно-ориентированном программировании.

Процедурно-ориентированный язык программирования - язык программирования высокого уровня, в основу которого положен принцип описания (последовательности) действий, позволяющей решить поставленную задачу.

Обычно процедурно-ориентированные языки задают программы как совокупности процедур или подпрограмм.

Процедурно-ориентированные языки программирования относятся к машинно-независимым. Они являются основными языками описания алгоритмов и имеются в математическом обеспечении по существу всех современных вычислительных машин.

Процедурно-ориентированные языки программирования могут лишь косвенно отражать содержание будущей реализации, используя специфические синтаксические конструкции для описания порядка преобразований.

Процедурные языки программирования

* Ada (язык общего назначения)
* Алгол 60
* Basic (до появления Visual Basic)
* Си
* C++
* КОБОЛ
* Фортран
* Модула-2
* HAL/S
* Pascal
* PureBasic
* ПЛ/1
* Рапира
* REXX
* Go
* Nim

Процедурно ориентированные языки применятся в концепции процедурно-ориентированного программирования.

*Слово*процедура*является здесь ключевым элементом. Оно означает*набор процедур*, который представляет собой*набор подпрограмм*или*набор функций*. Мы все знаем о*функциях*в языке C. C это процедурно-ориентированный язык. В ПОП основное внимание уделяется функциям или подпрограммам. Функции содержат набор команд, выполняющих определенную задачу.**Функции в программе вызываются повторно**для выполнения определенных задач.*

Объе́ктно-ориенти́рованное программи́рование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

При процедурно ориентированном подходе все программы, большие и маленькие, писались в одном файле. С течением времени программы становились всё сложнее и больше, что доставляло проблемы разработчикам при поддержке таких программ и внесении изменений. Эту проблему решает объектно-ориентированное программирование. ООП позволяет объединить данные и методы, относящиеся к одной сущности, и работать с ними, как с одним целым.

**Класс** - это шаблон, на основе которого может быть создан конкретный программный объект, он описывает свойства и методы, определяющие поведение объектов этого класса. Каждый конкретный объект, имеющий структуру этого класса, называется экземпляром класса.

Следующими важнейшими принципами ООП являются наследование, инкапсуляция и полиморфизм.

**Наследование** - предусматривает создание новых классов на базе существующих и позволяет классу потомку иметь (наследовать) все свойства класса - родителя.

**Инкапсуляция** – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали

реализации от пользователя.

**Полиморфизм** - означает, что рожденные объекты обладают информацией о том, какие методы они должны использовать в зависимости от того, в каком месте цепочки они находятся.

Современными языками объектно-ориентированного программирования являются С++ и Java.

3) Способы описания алгоритмов, правила оформления схем алгоритмов.

Основные способы записи алгоритмов:

• словесный ;

• с помощью схем;

• языком псевдокоде;

• языком программирования.

**Словесное** описание представляет структуру алгоритма на естественном языке. Никаких правил составления словесного описания не существует. Запись алгоритма осуществляется в произвольной форме на естественном, например, русском языке. Этот способ описания не имеет широкого распространения, так как строго не формализуем допускает неоднозначность толкования при описании некоторых действий; страдает многословностью.

**Псевдокод** – описание структуры алгоритма на естественном, частично формализованном языке, позволяющее выявить основные этапы решения задачи, перед точной его записью на языке программирования. В псевдокоде используются некоторые формальные конструкции и общепринятая математическая символика. Строгих синтаксических правил для записи псевдокода не существует. Это облегчает запись алгоритма при проектировании и позволяет описать алгоритм, используя любой набор команд. Однако в псевдокоде обычно используются некоторые конструкции, присущие формальным языкам, что облегчает переход от псевдокода к записи алгоритма на языке программирования.

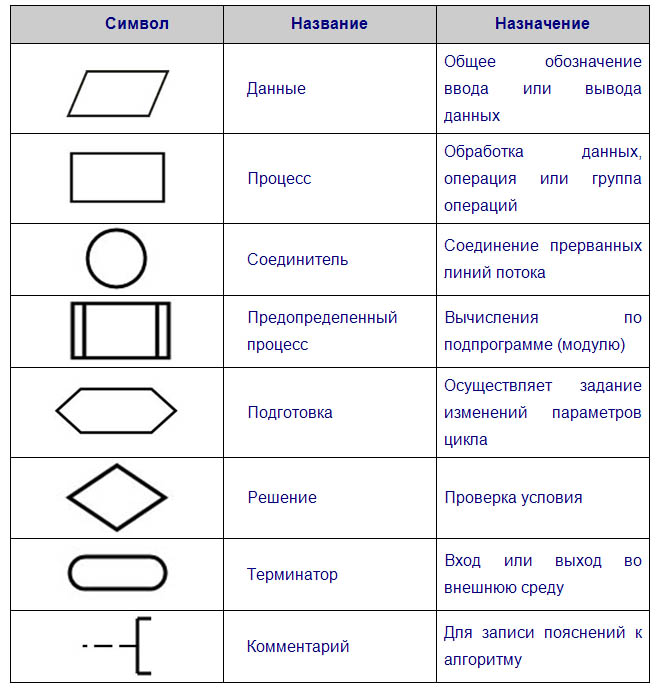
Алгоритм, записанный **на языке программирования** называется программой. Словесная и графическая форма записи алгоритма предназначены для человека. Алгоритм, предназначенный для исполнителя на компьютере записывается на языке программирования.

**Графическое описание** алгоритма в виде блок-схемы – это описание структуры алгоритма с помощью геометрических фигур с линиями связи.

Уловные обозначения и правила выполнения схем алгоритмов регламентируются требованиями Единой системы программной документации в соответствии с ГОСТ 19.701-90 [6].

Правила изображения блок- схем алгоритма:

1. В блок-схеме можно использовать строго определённые типы блоков.



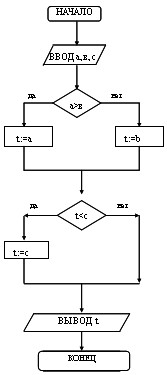
2. Стрелки на линиях связи можно не ставить при направлении сверху вниз и слева направо; противоположные направления обязательно указывают стрелкой на линии.

3. Для удобства блоки могут помечаться метками(буквами или цифрами).

4. Внутри блока ввода/вывода пишется ВВОД или ВЫВОД и перечисляются имена данных, подлежащих вводу/выводу.

5. Внутри блока действия для присваивания переменных значений используется знак присваивания.

Пример.



4) Разновидности структур алгоритмов.

По структуре алгоритмы разделяют на линейные, разветвляющиеся и циклические.

**Линейными** называют алгоритмы, в которых операции выполняются последовательно одна за другой, в естественном и единственном порядке следования. В таких алгоритмах все блоки имеют последовательное соединение логической связью передачи информационных потоков. В них могут использоваться все блоки, за исключением блоков проверки условия и модификации. Линейные алгоритмы, как правило, являются составной частью любого алгоритмического процесса.

При составлении схем алгоритмов часто возникает необходимость проведения анализа исходных данных или промежуточных результатов вычислений и определения дальнейшего порядка выполнения вычислительного процесса в зависимости от результатов этого анализа.

Алгоритмы, в которых в зависимости от выполнения некоторого логического условия происходит разветвление вычислений по одному из нескольких возможных направлений, называют **разветвляющимися.**Подобные алгоритмы предусматривают выбор одного из альтернативных путей продолжения вычислений. Каждое возможное направление вычислений называется *ветвью.*Логическое условие называют *простым,*если разветвляющийся процесс имеет две ветви, и *сложным,*-если процесс разветвляется на три и более ветви.

Любое сложное логическое условие может быть представлено в виде простых.

**Алгоритм циклической структуры** предусматривает многократное повторение действий в одной и той же последовательности по одним и тем же математическим зависимостям, но при разных значениях некоторой специально изменяемой величины. Циклические алгоритмы позволяют существенно сократить объем программы за счет многократного выполнения группы повторяющихся вычислений, так называемого тела цикла. Специально изменяемый по заданному закону параметр, входящий в тело цикла, называется переменной цикла. Переменная цикла используется для подготовки очередного повторения цикла и отслеживания условий его окончания. В качестве переменной цикла используют любые переменные, индексы массивов, аргументы вычисляемых функций и тому подобные величины. Во время выполнения тела цикла параметры переменной цикла изменяются в интервале от начального до конечного значения с заданным шагом. Следовательно, при организации циклических вычислений необходимо предусмотреть задание начального значения переменной цикла, закон ее изменения перед каждым новым повторением и ее конечное значение, при достижении которого произойдет завершение цикла.

Циклы, в теле которых нет разветвлений и других встроенных в них циклов, называют **простыми**. В противном случае их относят к **сложным**. Циклические алгоритмы разделяют на **детерминированные** и **итерационные**.

Циклы, в которых число повторений заранее известно из исходных данных или определено в ходе решения задачи, называют **детерминированными.** Для организации детерминированных циклов наиболее целесообразно использовать блок модификации, внутри которого указывается переменная цикла, ее начальное и конечное значения, а также шаг ее изменения (если шаг изменения равен 1, то его допускается не указывать).

Циклы, в которых число повторений неизвестно из исходных данных и не определено по ходу решения задачи, называют **итерационными**. В итерационных циклах для организации выхода из тела цикла предусматривается проверка некоторого заранее заданного условия, для чего используют блок проверки условия. В итерационных циклах невозможно использовать блоки модификации, так как при организации таких циклов заранее неизвестно количество изменений переменной цикла и ее конечное значение.

В зависимости от местонахождения блока проверки условия итерационные циклы могут быть организованы как циклы с предусловием (блок проверки условия размещен перед телом цикла) или с постусловием (блок проверки условия размещен после тела цикла). Если в цикле с предусловием входящие в тело цикла команды могут не выполняться ни разу (если начальное значение параметра цикла удовлетворяет условию выхода из цикла), то в цикле с постусловием - выполняются как минимум один раз (даже если начальное значение параметра цикла удовлетворяет условию выхода из него).

5) Полное построение алгоритма решения задачи

Алгоритм представляет собой строгую систему правил, определенную последовательность действий над некоторыми объектами. Следуя такой системе правил, как инструкции, различные исполнители будут действовать одинаково и получать одинаковые результаты.

Полное построение алгоритма предусматривает выполнение следующих последовательно друг за другом этапов:

1) постановка задачи;

2) построение модели;

3) разработка алгоритма;

4) проверка правильности алгоритма;

5) реализация, т. е. программирование алгоритма;

6) анализ алгоритма и его сложности;

7) проверка (отладка) программы;

8) составление документации

**1) Постановка задачи**

Прежде чем понять задачу, необходимо ее точно сформулировать. Обычно процесс точной формулировки задачи сводится к постановке правильных вопросов. Перечислим некоторые полезные вопросы:

Понятна ли терминология?

Что дано?

Что следует найти?

Как определить решение?

Каких данных не хватает и все ли они необходимы?

Являются ли какие-то имеющиеся данные бесполезными?

Какие сделать допущения?

Если задачу необходимо решить математически, то ее постановка должна быть формализована, т. е. выражена в терминах понятий, обладающих точно определенными свойствами и находящихся между собой в строго определенных отношениях.

Если сказано, что задача поставлена, то предполагается, что известны не только характеристики существующей ситуации, но и требования к решению задачи. Формализация постановки задачи позволяет наметить путь, который ведет от постановки решения задачи к ее решению, т. е. провести логический анализ формальной постановки задачи в результате выявления свойств и отношений между понятиями.

**2) Построение модели**

Задача четко поставлена, теперь нужно сформулировать для нее математическую модель. Выбор модели существенно влияет на остальные этапы в процессе решения.

Выбор модели — в большей степени искусство, чем наука. Правда, если вы будете встречаться с типовой задачей, то опыт, приобретенный ранее, не потребует от вас особого творческого подхода, и в этом случае как раз будет удобным и целесообраз­ным воспользоваться ранее наработанными правилами. Поэтому изучение удачных моделей — это наилучший способ приобрести опыт в моделировании.

Приступая к разработке модели, следует задать, по крайней мере, два основных вопроса:

1. Какие математические структуры больше всего подходят

для задачи?

2. Существуют ли решенные аналогичные задачи?

Второй вопрос, возможно, самый полезный во всей матема­тической модели. В контексте моделирования он часто дает от­вет на первый вопрос. Действительно, большинство решаемых в математике задач, как правило, являются модификациями ранее решенных.

Сначала нужно рассмотреть первый вопрос. Мы должны описать математически, что мы знаем и что хотим найти.

**3) Разработка алгоритма**

Выбор метода разработки алгоритма зачастую сильно зависит от выбора модели и может в значитель­ной степени повлиять на эффективность алгоритма решения. Два различных алгоритма могут быть правильными, но очень сильно отличаться по эффективности.

**4) Проверка правильности алгоритма**

Доказательство правильности алго­ритма — это один из наиболее трудных, а иногда и особенно утомительных этапов создания алгоритма. Вероятно, наиболее распространенный прием доказательства правильности програм­мы — это прогон ее на разных тестах. Если выданные програм­мой ответы могут быть подтверждены известными или вычис­ленными вручную данными, возникает искушение сделать вы­вод, что программа «работает» правильно. Однако этот метод редко исключает все сомнения; может существовать случай, в котором программа не работает.

Следует подчеркнуть и тот факт, что правильность алгоритма еще ничего не говорит об его эффективности. В этом смысле ис­черпывающие алгоритмы, или, как их еще называют — алгорит­мы полного перебора — редко бывают хорошими во всех отно­шениях.

**5) Реализация**

Как только алгоритм выражен, допус­тим, в виде последовательности шагов, и мы убедились в его правильности, настает черед реализации алгоритма, т. е. написа­ния программы для компьютера.

При этом возникают следующие проблемы:

• очень часто отдельно взятый шаг алгоритма может быть выражен в форме, которую трудно перевести непосредст­венно в конструкции языка программирования. Например, один из шагов алгоритма может быть записан в виде, тре­бующем целой подпрограммы для своей реализации;

• реализация может оказаться трудным процессом потому, что перед тем, как написать программу, необходимо по­строить целую систему структур данных для представления важных аспектов используемой модели.

Чтобы сделать это, необходимо ответить, например, на такие вопросы:

• Каковы основные переменные?

• Каких они типов?

• Сколько нужно массивов и какой размерности?

• Какие нужны подпрограммы (возможно, уже записанные в памяти)?

• Каким языком программирования пользоваться?

Конкретная реализация может существенно влиять на требования к памяти и на скорость алгоритма.

6) Основные характеристики изучаемого алгоритмического языка. Алфавит языка.

Основными характеристиками в алгоритмических языках обычно являются следующие.

1.  Имена (идентификаторы) — употpебляются для обозначения объектов пpогpаммы (пеpеменных, массивов, функций и дp.).

2.  Опеpации. Типы операций:

* аpифметические опеpации   +   ,   —   ,   \*   ,   /   и дp. ;
* логические опеpации   и   ,   или   ,   не  ;
* опеpации отношения   <   ,   >   ,   <=   ,   >=   ,   =   ,   <>   ;
* опеpация сцепки (иначе,   "присоединения",   "конкатенации"  ) символьных значений дpуг с другом с образованием одной длинной строки; изображается знаком "+".

3.  Данные — величины, обpабатываемые пpогpаммой. Имеется тpи основных вида данных: константы, пеpеменные и массивы.

* Константы — это данные, которые зафиксированы в тексте программы и не изменяются в процессе ее выполнения.

Пpимеpы констант:

* + числовые   7.5   ,   12   ;
  + логические   да (истина),   нет   (ложь);
  + символьные (содержат ровно один символ)   "А"   ,   "+"   ;
  + литеpные (содержат произвольное количество символов) "a0",  "Мир",  ""  (пустая строка).
* Пеpеменные обозначаются именами и могут изменять свои значения в ходе выполнения пpогpаммы. Пеpеменные бывают целые, вещественные, логические, символьные и литерные.
* Массивы — последовательности однотипных элементов, число которых фиксировано и которым присвоено одно имя. Положение элемента в массиве однозначно определяется его индексами (одним, в случае одномерного массива, или несколькими, если массив многомерный). Иногда массивы называют таблицами.

4.  Выpажения — пpедназначаются для выполнения необходимых вычислений, состоят из констант, пеpеменных, указателей функций (напpимеp, exp(x)), объединенных знаками опеpаций.

Выражения записываются в виде линейных последовательностей символов (без подстрочных и надстрочных символов, "многоэтажных" дробей и т.д.), что позволяет вводить их в компьютер, последовательно нажимая на соответствующие клавиши клавиатуры.

Различают выражения арифметические, логические и строковые.

* Арифметические выражения служат для определения одного числового значения.  Например,  (1+sin(x))/2. Значение этого выражения при x=0 равно 0.5,  а при x=p/2 — единице.
* Логические выражения описывают некоторые условия, которые могут удовлетворяться или не удовлетворяться. Таким образом, логическое выражение может принимать только два значения —  "истина"  или  "ложь"  (да  или  нет). Рассмотрим в качестве примера логическое выражение  x\*x + y\*y < r\*r ,  определяющее принадлежность точки с координатами  (x, y)  внутренней области круга радиусом  r  c центром в начале координат.  При  x=1,  y=1,  r=2 значение этого выражения —  "истина", а при  x=2,  y=2,  r=1  — "ложь".
* Cтроковые (литерные) выражения, значениями которых являются текcты. В строковые выражения могут входить литерные и строковые константы, литерные и строковые переменные, литерные функции, разделенные знаками операции сцепки. Например, А + В означает присоединение строки  В  к концу строки  А . Если   А = *"куст "*,   а   В = *"зеленый"*,   то значение выражения   А + В   есть   *"куст зеленый".*

5.  Операторы (команды). Оператор — это наиболее крупное и содержательное понятие языка: каждый оператор представляет собой законченную фразу языка и определяет некоторый вполне законченный этап обработки данных. В состав опеpатоpов входят:

* ключевые слова;
* данные;
* выpажения и т.д.

Операторы подpазделяются на исполняемые и неисполняемые. **Неисполняемые** опеpатоpы пpедназначены для описания данных и стpуктуpы пpогpаммы, а **исполняемые** — для выполнения pазличных действий (напpимеp, опеpатоp пpисваивания, опеpатоpы ввода и вывода, условный оператор, операторы цикла, оператор процедуры и дp.).

Алфавит языка, или его символы — это основные неделимые знаки, с помощью которых пишутся все тексты на языке.

Обычно язык программирования включает следующий набор основных символов:

1. строчные и прописные буквы латинского и национального алфавитов
2. цифры
3. знаки операций: + — \* / = < : @ & |
4. символ подчеркивания и пробела
5. ограничители и разделители: . , ' ( ) [ ] { }
6. специальные символы: ^ # $ и др.

С помощью символов алфавита записываются **служебные слова**, которые составляют словарь языка.

7) Правила записи основных объектов языка

У каждого языка программирования есть свой стандарт оформления кода, который говорит, как надо делать отступы, где ставить пробелы и скобки, как называть объекты, как комментировать код и т.д.

Необходимо использовать длинные наглядные имена создаваемых объектов языка. Имена классов и типов должны быть существительными. Название метода должно содержать глагол. Если метод определяет, является ли какая-то информация об объекте истинной или ложной, его имя должно начинаться с «is». Методы, которые возвращают свойства объектов, должны начинаться с «get», а устанавливающие значения свойств — «set».

Далее по конвеншину языка

8) Типы данных. Константы. Переменные. Метки. Выражения.

**Тип данных** — класс данных, характеризуемый членами класса и операциями, которые могут быть к ним применены

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.

Величина, имеющая неизменное значение в процессе исполнения программы, называется **константой.** Константа может быть непоименованной, тогда она записывается в тексте программы как непосредственное значение, например 4, 23.1, -6.41Е23, «Строка». Поименованая константа имеет имя, которое нужно указывать при необходимости использовать ее в программе.

**Переменная** — именованный участок памяти для хранения данных определенного типа. Значение переменной (информация в соответствующих ячейках памяти) в ходе выполнения программы может быть изменено. Каждая переменная имеет имя. Некоторые языки программирования (например Паскаль) требуют явного описания типа переменной, другие (например Бейсик) определяют тип переменной по ее имени.

**Метка** (англ. label) — символьное имя, идентификатор для более удобного указания данных и кода в языках программирования. Позволяет программисту обходиться без вычисления и пересчёта адресов и смещений внутри программы (эти действия за него выполняет компилятор). Несмотря на то, что большинство языков программирования высокого уровня поддерживают операции с метками, их использование крайне нежелательно, так как программный код становится плохочитаемым, и отладка такой программы занимает большее время.

При компиляции каждой метке сопоставляется определённый адрес (число), который заносится в исполняемый модуль. Таким образом, в случае необходимости компилятор может вычислить смещение между метками (разность адресов).

В большинстве языков программирования формат записи метки такой: название\_метки:

Н-р:

for (…)

for (…) {

…

if (disaster) /\* если бедствие \*/

goto error; /\* уйти на ошибку \*/

error: /\* обработка ошибки \*/

ликвидировать беспорядок

Над переменными и константами в языках программирования можно производить различные операции. ***Выражением*** называется совокупность переменных, констант, знаков операций, имен функций, скобок, которая может быть вычислена в соответствии с синтаксисом языка программирования. Результатом вычисления выражения является величина определенного типа. Если эта величина имеет числовой тип, то такое выражение называется *арифметическим*.

В состав арифметического выражения могут входить:

— числовые константы;

— имена переменных;

— знаки математических операций;

— математические функции и функции, возвращающие число;

— открывающиеся и закрывающиеся круглые скобки.

Величины, над которыми выполняются операции, называются *операндами*. В зависимости от количества операндов операции могут быть *унарными* (один операнд) и *бинарными* (два операнда).

9) Арифметические и логические выражения

Арифметические выражения. В состав арифметических выражений могут входить кроме переменных числового типа также и числа, над переменными и числами могут произво­диться различные арифметические операции, а также мате­матические операции, выраженные с помощью функций.

Порядок вычисления арифметических выражений соот­ветствует общеизвестному порядку выполнения арифметиче­ских операций (возведение в степень, умножение или деле­ние, сложение или вычитание), который может изменяться с помощью скобок.

Логические выражения. В состав логических выражений кроме логических переменных могут входить также числа, числовые или строковые переменные или выражения, кото­рые сравниваются между собой с использованием операций сравнения (>, <, =, >=, <= и пр.).

Логическое выражение может принимать лишь два зна­чения: «истина» или «ложь». Например:

5 > 3 — истинно;

2\*2 = 5 — ложно.

Над элементами логических выражений могут произво­диться логические операции, которые на языке Visual Basic обозначаются следующим образом: логическое умноже­ние — And,логическое сложение — Or и логическое отрица­ние Not.При записи сложных логических выражений час­то используются скобки. Например:

(5 > 3) And(2\*2 = 5)— ложно;

(5 > 3) Or (2\*2 = 5)— истинно.

10) Классификация операторов алгоритмического языка

**Оператор** (Statement) – это основная единица программы, которая выполняет определенные действия над данными.

Между собой операторы разделяются точкой с запятой.

По действиям операторы разделяются на следующие типы:

1) ***обрабатывающие***, которые обрабатывают данные:

- ввода;

- вывода;

- присваивания (вычисления новых значений).

2) ***управляющие***, которые управляют выполнением обрабатывающих операторов:

- изменяющие естественный порядок выполнения других операторов в зависимости от некоторого условия или безусловно;

- обеспечивающие повторение некоторых групп операторов.

По строению или структуре операторы разделяются на следующие типы:

1) ***простые***, которые не содержат в себе других операторов:

- оператор присваивания =

- оператор безусловного перехода GOTO, BREAK, CONTINUE,RETURN;

- оператор обращения к процедуре (функции) (содержит имя процедуры и перечисленные в скобках параметры процедуры);

- пустой оператор (не выполняет никаких действий и не отображается в программе. Может потребоваться для осуществления перехода на него).

2) ***структурные или структурированные***, которые состоят из других операторов:

- составной оператор (последовательность операторов программы, заключенная в операторные скобки );

- условные операторы;

- операторы цикла;

Операторы, в которых используется один операнд, например оператор инкремента (++) или new, называются ***унарными***. Операторы, в которых используются два операнда, например арифметические операторы (+,-,\*,/), называются ***бинарными*** . Для одного оператора — условного (?:) — используются три операнда, и такой оператор является единственным **тернарным** оператором в C#.

11) Оператор присваивания. Операторы управления.

Операторы присваивания сохраняют значение в объекте, обозначенном левым операндом. Существует два типа операторов присваивания: **простое присваивание**, при котором значение второго операнда сохраняется в объекте, заданном первым операндом, и **составное присваивание**, при котором сначала выполняется арифметическая или побитовая операция или операция сдвига, а затем сохраняется результат.

Простой оператор присваивания (=) сохраняет значение второго операнда в объекте, указанном первым операндом. Если оба объекта имеют арифметические типы, перед сохранением значения правый операнд преобразуется к типу левого.

операторs управления программой это механизмы, с помощью которых можно изменять порядок выполнения программы.

С предоставляет три категории операторов управления программой:

* итерационные операторы;
* операторы выбора;
* операторы переходов.

Итерационные операторы - это while, for и do/while. Они чаще всего называются циклами. Операторы выбора или условные операторы - это if и switch. Операторы перехода - это break, continue и goto. (Оператор return, в принципе, также является оператором перехода, поскольку он воздействует на программу.)

12) Организация ввода-вывода данных. Структура программы.

Взаимодействие программы с внешней средой (пользователем, другими программами, хранимыми данными) является совершенно необходимым. За такой интерфейс в языках программирования отвечают операторы ввода-вывода информации. Эти инструкции позволяют ввести в программу данные во время выполнения программы (а не на этапе ее написания) и осуществить вывод рассчитанных данных в понятном человеку виде.

Функции ввода-вывода производят чтение и запись для файлов и устройств. Операции файлового ввода-вывода происходят в текстовом или двоичном режиме. Библиотека времени выполнения Microsoft содержит три типа функций ввода-вывода:

* Функции [Stream I/O](https://msdn.microsoft.com/ru-ru/library/c565h7xx.aspx) обрабатывают данные как поток отдельных символов.
* Функции [Low-level I/O](https://msdn.microsoft.com/ru-ru/library/40bbyw78.aspx) вызывают непосредственно операционную систему для выполнения низкоуровневых операций, которые затем выполняются с помощью потокового ввода-вывода.
* Функции [Console and port I/O](https://msdn.microsoft.com/ru-ru/library/7x2hy4cx.aspx) производят чтение или запись непосредственно на консоль (клавиатуру, экран) или в порт ввода-вывода (например, порт принтера).

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

В C# основными понятиями организационной структуры являются ***программы***, ***пространства имен***, ***типы***, ***члены*** и ***сборки***. Программа на языке C# состоит из одного или нескольких файлов. В программе объявляются типы, которые содержат члены. Эти типы можно организовать в пространства имен. Примерами типов являются классы и интерфейсы. К членам относятся поля, методы, свойства и события. При компиляции программы на C# упаковываются в сборки. Сборка — это файл, обычно с расширением .exe или .dll, если она реализует ***приложение*** или ***библиотеку***, соответственно.

Проект представляет собой совокупность файлов, которые компилятор использует для создания выполняемого файла. Структура проекта отображается в окне Обозреватель решений. Основными элементами проекта являются:-главный модуль приложения (файл program.cs);-модули форм. В главном модуле находится функция Main, с которой начинается выполнение программы. Чтобы увидеть текст главного модуля, сделайте двойной щелчок по файлу program.cs. Функция Mainсоздает стартовую форму (имя класса стартовой формы указывается в качестве параметра метода Run), в результате чего на экране появляется окно программы.

13) Переход от схемы алгоритма к схеме программы.