Лекция 2. Основы разработки алгоритмов

2.1. Основные понятия и определения

Существуют различные определения понятия алгоритма:

- **Алгоритм** – это описание последовательности действий для решения задачи или достижения поставленной цели.
- **Алгоритм** – это правила выполнения основных операций обработки данных.
- **Алгоритм** – это описание вычислений по математическим формулам.

Использование алгоритмов позволяет оценить эффективность предлагаемого способа решения поставленной задачи, результативность данного решения, исправить возможные ошибки, сравнить его еще до применения на практике с другими алгоритмами решения этой же задачи. А также, алгоритм является основой для составления программы, которую пишет программист на каком-либо языке программирования. Т.е. алгоритм не привязан к конкретному языку программирования и пользователь вправе сам выбирать с использованием каких программных средств будет решена задача.

Алгоритм должен обладать следующими свойствами:

- **Понятность для исполнителя**. Т.е. алгоритм должен быть достаточно подробным и доступным для его понимания. Необходимо, чтобы в нем отсутствовали символы и описания понятные только составителю данного алгоритма.
- **Дискретность**. Алгоритм должен представлять процесс решения задачи как последовательное выполнение простых шагов.
- **Определённость**. Каждый шаг в алгоритме должен быть четким и однозначным. Т.е. процесс выполнения алгоритма не должен требовать каких-либо дополнительных указаний или сведений о задаче.
- **Результативность**. Данное свойство состоит в том, что алгоритм должен приводить к решению задачи, либо остановки, в случае невозможности решения поставленной задачи, за конечное число шагов.
- **Массовость**. Т.е. алгоритм решения задачи разрабатывается в общем виде и не имеет привязки к средствам его реализации.

На территории РФ действует единая система программной документации (ЕСПД), частью которой является ГОСТ 19.701-90 “Схемы алгоритмов программ, данных и систем”. Данный ГОСТ практически полностью соответствует международному стандарту ISO 5807:1985.

2.2. Основные блоки для построения алгоритмов

В данном разделе представлены основные блоки, применяемые при составлении алгоритмов.

2.2.1. Блок начала и конца работы функций

Данным блоком начинается и заканчивается любая функция. Он имеет следующий вид:

Рис. 2.1. Блок начала и конца работы функций

Тип возвращаемого значения и аргументов функции обычно указывается в комментариях к данному блоку.

2.2.2. Блок объявления переменных

Блок объявления переменных имеет следующий вид:

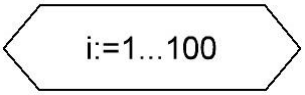


Рис. 2.2. Блок объявления переменных

В данном блоке указывается перечисление переменных, которые необходимы для выполнения поставленной задачи. В данном блоке можно как просто объявить переменные, так и указать их начальные значения.

2.2.3. Блок операций ввода и вывода данных

В ГОСТ определено много различных символов ввода/вывода данных, которые отличаются источником и приемником данных. Если источник или приемник данных не принципиален, обычно используется символ параллелограмма. В случае необходимости подробности ввода/вывода можно указать в комментариях.

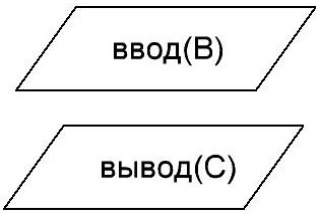


Рис. 2.3. Блок ввода/вывода данных

2.2.4. Блок выполнения операций над данными

Данный блок имеет следующий вид:

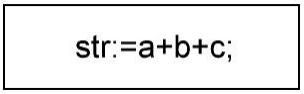


Рис. 2.4. Блок выполнения операций

В блоке операций обычно размещают одну или несколько операций, не требующих вызова внешних функций. Тут могут быть прописаны операции присвоения, копирования, арифметические и логические операции и т.д.

2.2.5. Блок ветвления алгоритма

Блок ветвления выглядит следующим образом:

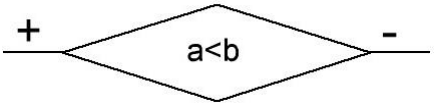


Рис. 2.5. Блок ветвления

Данный блок имеет один вход и несколько выходов. Внутри блока прописывается условие, по которому происходит переход по соответствующему выходу. В случае, если блок имеет 2 выхода на них подписывается результат сравнения – «да/нет» или «+/-». Если из блока выходит большее число линий (оператор выбора), внутри него записывается имя переменной, а на выходах значения этой переменной.

2.2.6. Блок вызова внешней процедуры/функции

Блок вызова процедуры или функции имеет вид:

Рис. 2.6. Блоки вызова функций/процедур.

Данный блок означает вызов функции, требующей перехода к подпрограмме. Если для решения поставленной задачи необходимо использовать дополнительные функции или процедуры, то необходимо приводить отдельный алгоритм данных процедур/функций.

2.2.7. Соединитель

Соединитель используется в случае, если блок-схема не умещается на лист.

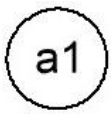


Рис. 2.7. Соединитель

Данный блок отражает переход управления между листами. Символ может использоваться и на одном листе, если по каким-либо причинам необходимо переместить часть алгоритма.

2.2.8. Блоки начала и конца цикла

Символы начала и конца цикла содержат имя и условие (рис. 2.8).

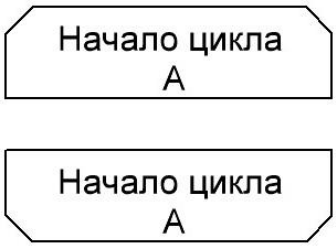


Рис. 2.8. Блоки начала и конца цикла

В данных блоках условие может отсутствовать, но только в одном из блоков. Наличие условия в блоке начала или блоке конца цикла будет определять тип условного оператора цикла, соответствующего символам на языке высокого уровня, например, ***while*** или ***do-while*** (см. ниже).

2.2.9. Блок комментариев

Комментарии при составлении алгоритмов имеют следующий вид:

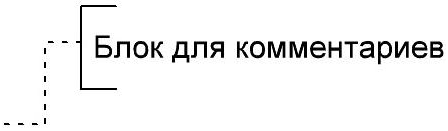


Рис. 2.9. Блок комментариев

Комментарий может быть соединен как с одним блоком, так и группой блоков. Во втором случае группа блоков выделяется на схеме пунктирной линией. При составлении алгоритмов необходимо следить за тем, чтобы комментариев было достаточно для правильного прочтения, но при этом они должны быть краткими и информативными.

2.3. Алгоритмическое описание действий, условий и циклов

2.3.1. Описание последовательности действий

Пример описания последовательности действий посредствам алгоритмических блоков представлен на рис. 2.10.

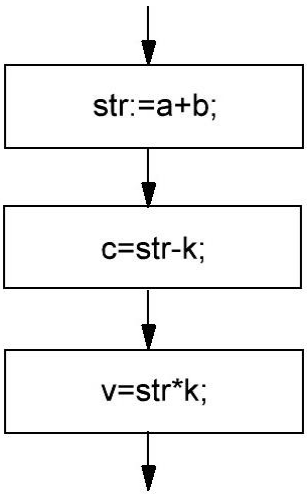


Рис. 2.10. Последовательность действий

Последовательность действий в алгоритме представляется соединением блоков с помощью стрелок. Переход от одного блока к другому определяется направлением стрелок.

2.3.2. Условие «*if*»

Блок «if» представляется с помощью блока ветвления (рис. 2.5). При переходе к данному блоку происходит проверка какого-либо условия и если результат проверки «истина», то выполняются какие-либо действия. В ином случае ничего не происходит. Пример алгоритма условия «***if***» представлен на рис 2.11.

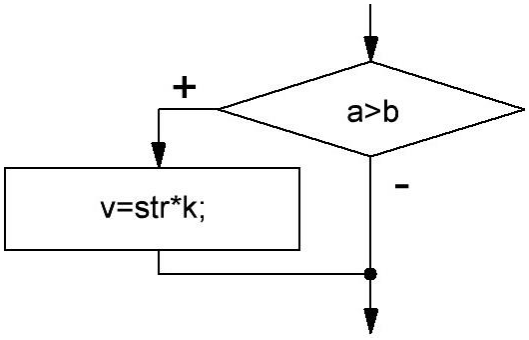


Рис. 2.11. Условие «if»

В данном примере происходит проверка условия ***a***>***b***. Если переменная ***a*** больше, чем переменная ***b***, то будет выполняться операция ***v***=***str*********k***. Если же переменная ***a*** меньше либо равна переменной ***b***, то никакой операции выполняться не будет и алгоритм переходит к следующим блокам.

2.3.3. Условие «*if-else*»

Блок «***if-else***» также представляется с помощью блока ветвления (рис. 2.5). При переходе к данному блоку происходит проверка какого-либо условия и действие, которое будет выполняться в данном случае определяется тем является ли условие «истиной» или «ложью». Пример алгоритма условия «***if-else***» представлен на рис 2.12.

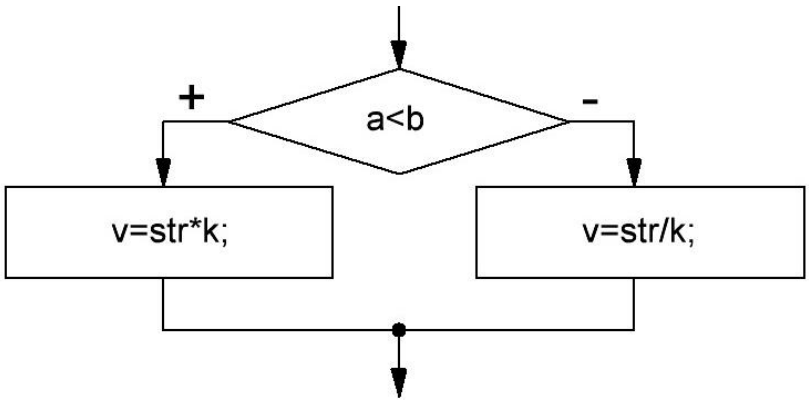


*Рис. 2.12. Условие «**if-else**»*

В данном примере происходит проверка условия ***a***<***b***. Если переменная ***a*** меньше, чем переменная ***b***, то будет выполняться операция ***v***=***str****k***. Если же переменная ***a*** больше либо равна переменной ***b***, то будет выполняться операция ***v***=***str/k***. Далее независимо от условия алгоритм переходит к следующим блокам.

2.3.4. Условие «*switch-case*»

Условие «***switch-case***» представляет собой операцию множественного выбора, при которой проверка условия может иметь более двух возможных вариантов. Действие, которое будет выполняться зависит от того какое условие имеет значение «истина». При выполнении какого-либо условия осуществляется переход к операции и далее условия больше не проверяется и алгоритм переходит к следующему блоку. Пример данного условия представлен на рис. 2.13.

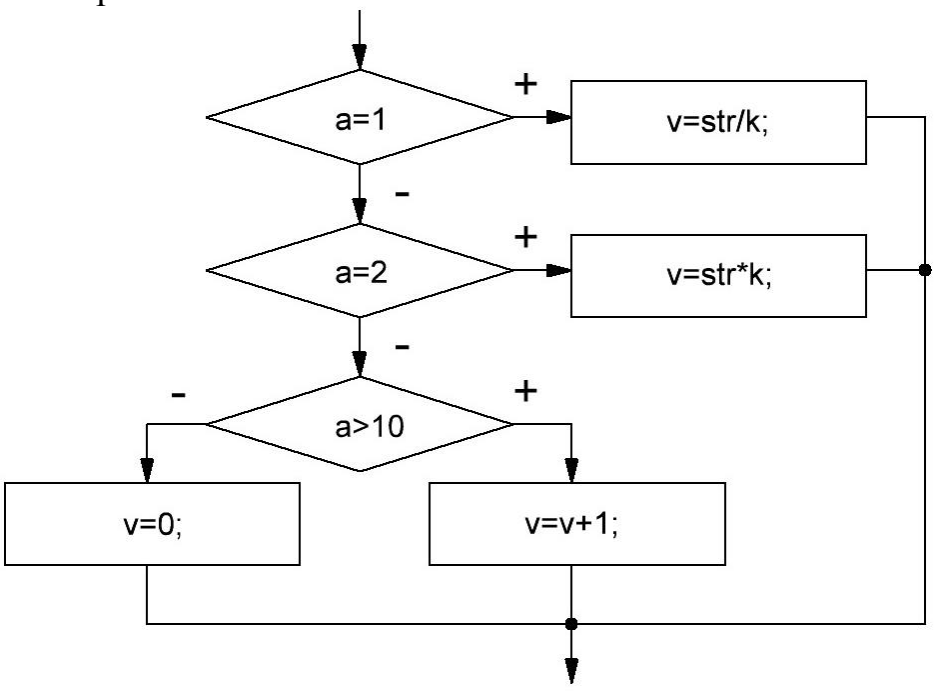


Рис. 2.13. Условие «switch…case»

При выполнении данного блока сначала осуществляется проверка ***a***=1, если условие «истина», то выполняется операция ***v***=***str/k*** и алгоритм переходит далее без проверки остальных условий. В случае если переменная ***a*** не равна

1, то проверяется условие ***a***=2. В случае выполнения данного условия выполняется действие ***v***=***str*******k*** и алгоритм переходит далее без проверки остальных условий. В случае если переменная ***a*** не равна 2, то проверятся условие ***a***>10 и в зависимости от результата проверки будет выполняться либо операция ***v***=0, либо операция ***v***=***v***+1, и алгоритм переходит далее.

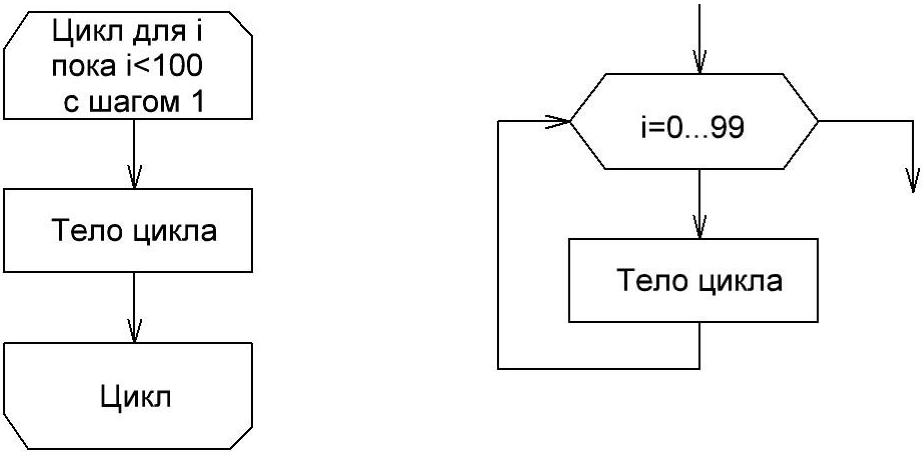
2.3.5. Цикл «*for*»

Цикл «***for***» - это цикл с заданным числом повторение. Синтаксис данного цикла в языках программирования подразумевает задание начального значения какой-либо переменной, действия которое необходимо выполнять с данной переменной и условие для данной переменной при котором цикл заканчивается. Приведем пример:

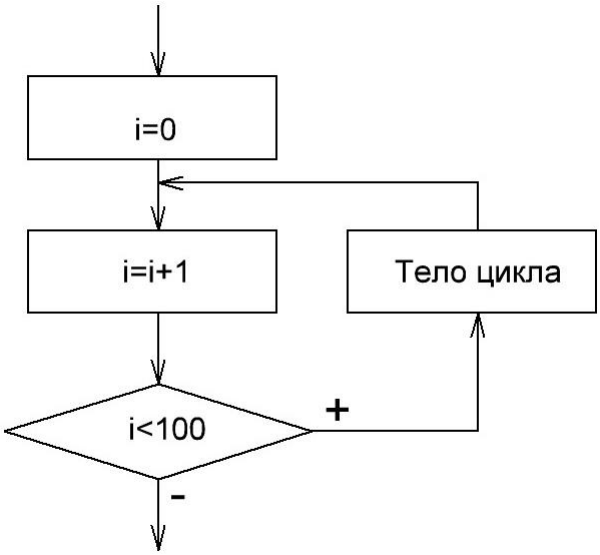
```
for(i=0;i<100;i=i+1)
{
тело цикла (операции)
}
```

В данном случае при переходе к данному циклу переменной ***i*** задается начальное значение равное 0. Тело цикла будет выполняться пока выполняется условие ***i***<100, при этом при каждом проходе цикла с переменной ***i*** будет осуществляться действие ***i***=***i***+1.

Алгоритмические представления цикла «***for***» могут быть различными. Приведем примеры алгоритмов для цикла «for», представленного выше (рис. 2.14).



а *б*



в

Рис. 2.14. Цикл «for»

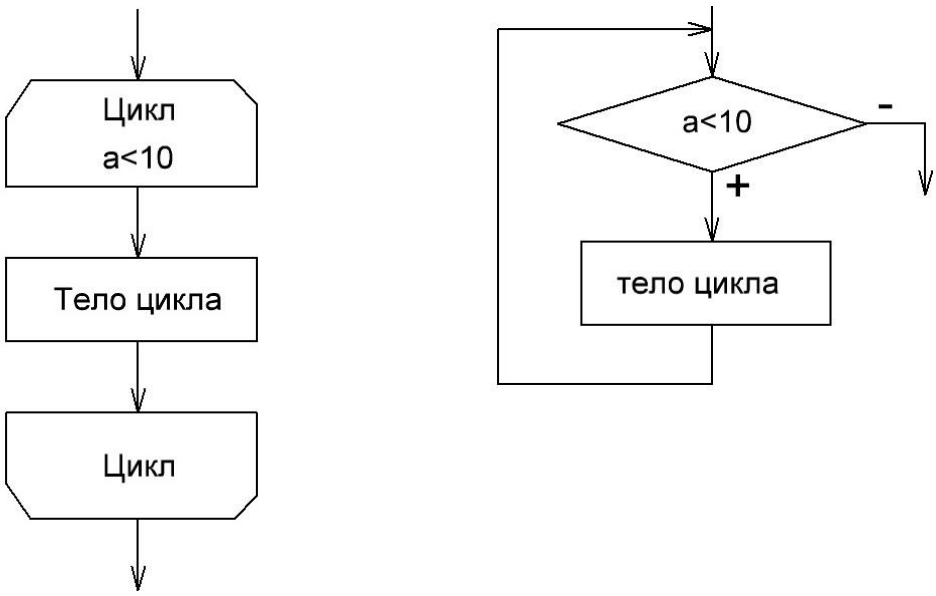
При разработке алгоритма можно использовать любой из блоков, представленных на рис. 2.14. При этом разработчик программы сам определяет каким оператором цикла он будет его реализовывать.

2.3.6. Цикл «*while*»

Цикл «*while*» – это цикл с предварительным условием. При переходе к данному циклу перед началом выполнения осуществляется проверка условия. В случае если условие не выполняется, происходит выход из цикла. Цикл с предусловием может не выполниться ни одного раза. Приведем пример цикла:

***while*(*a*<10)**
{
тело цикла (операции)
}

В данном примере будет выполняться тело цикла до тех пор пока выполняется условие ***a***<10. Как и в предыдущем случаем могут быть разные варианты представления данного цикла в алгоритме (рис. 2.15).



а *б*

Рис. 2.15. Цикл «while»

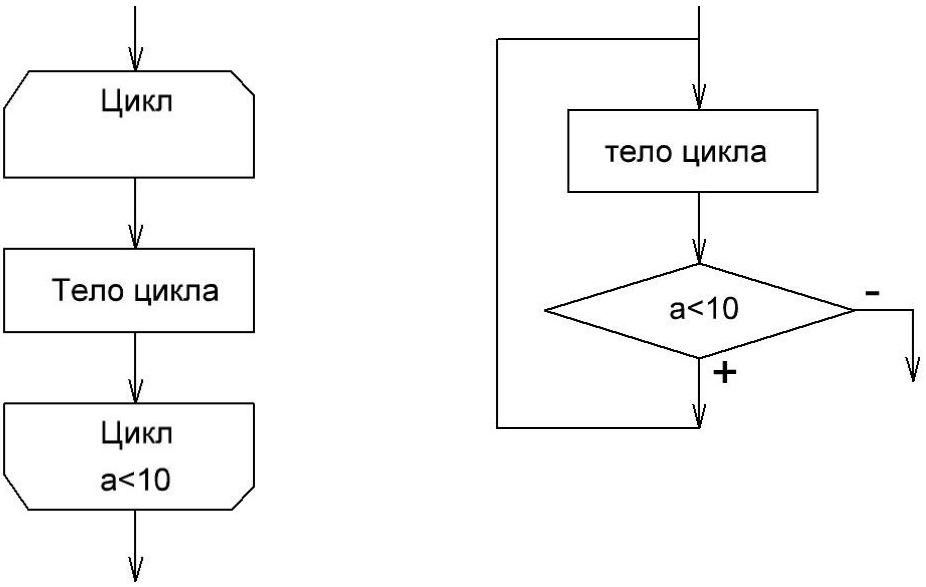
При разработке алгоритма можно использовать любой из блоков, представленных на рис. 2.15.

2.3.7. Цикл «*do-while*»

Цикл «***do-while***» – это цикл с постусловием. При переходе к данному циклу сначала выполняется операция (тело цикла) и только потом проверяется условие. В случае если условие не выполняется, происходит выход из цикла. Таким образом, данный цикл выполняется как минимум один раз, в отличие от цикла «***while***». Приведем пример цикла:

```
do
{
тело цикла (операции)
}
while(a<10)
```

В данном примере будет выполняться тело цикла и после проверяется условие ***a***<10. На рис. 2.16 показаны варианты представления данного цикла в алгоритме.



а *б*

Рис. 2.16. Цикл «do-while»

При разработке алгоритма можно использовать любой из блоков, представленных на рис. 2.16.

2.4. Примеры алгоритмов

Пример 2.1. Алгоритм подсчета суммы двух чисел (***A*** и ***B***), введенных с клавиатуры, с выводом на экран результата (***C***). Алгоритм представлена на рис. 2.17.

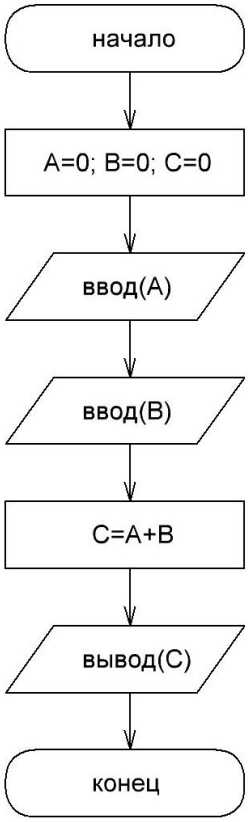


Рис. 2.17. Алгоритм подсчета суммы двух чисел

Пример 2.2. Алгоритм сравнения двух чисел (***A*** и ***B***), введенных с клавиатуры, с выводом на экран большего числа. В случае равенства выводить знак “=” (рис. 2.18).

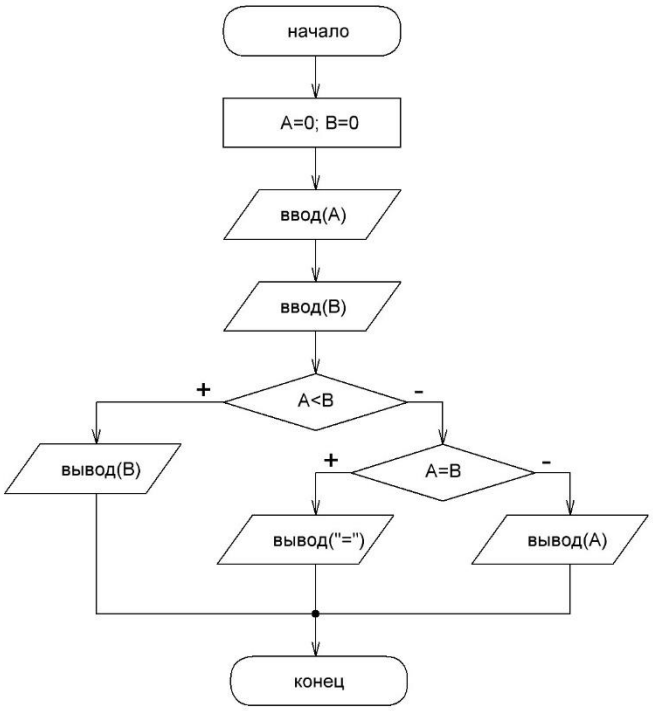


Рис. 2.18. Алгоритм сравнения двух чисел

Пример 2.3. Алгоритм подсчета суммы элементов массива ***A***[10] с размерностью 10 с выводом на экран результата ***B***. Три варианта алгоритма представлена на рис. 2.19-2.21.

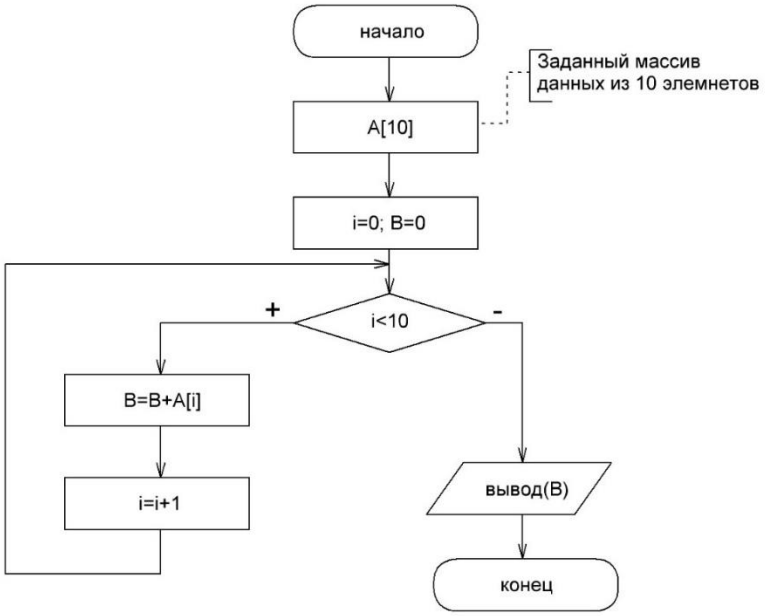


Рис. 2.19. Алгоритм подсчета суммы элементов массива с помощью функции «if-else»

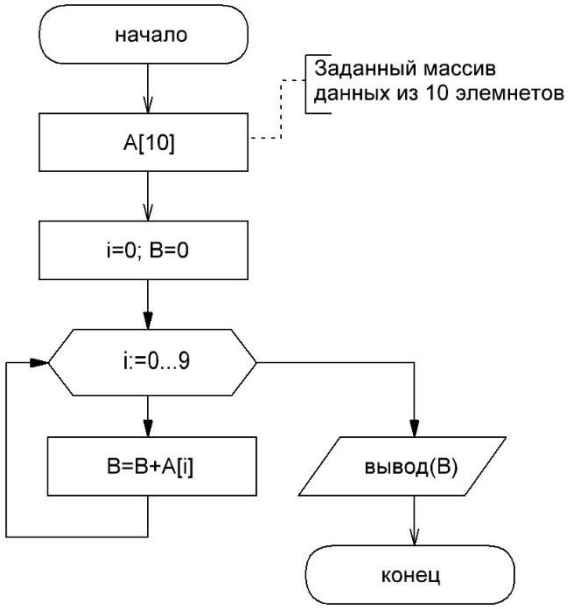


Рис. 2.20. Алгоритм подсчета суммы элементов массива с помощью функции «for»

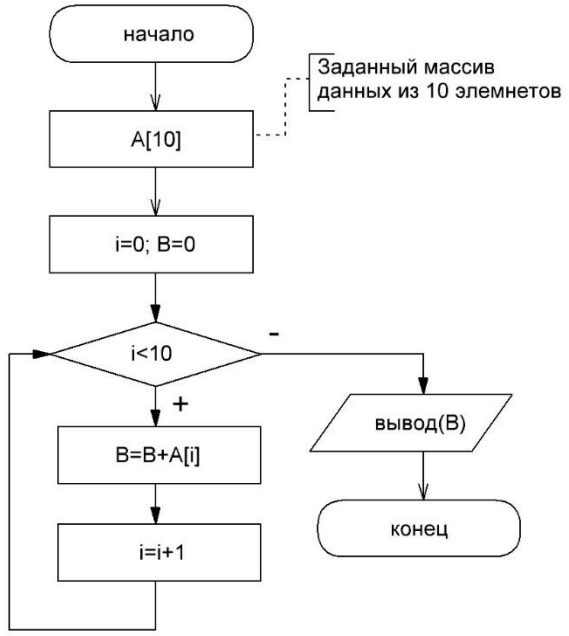


Рис. 2.21. Алгоритм подсчета суммы элементов массива с помощью функции «while»

Пример 2.4. Алгоритм вычисления площади квадрата (если нажата кнопка "***s***") или объема куба (если нажата кнопка "***v***") с использованием внешних функций и выводом результата на экран (рис. 2.22-2.24). Ввод величины грани квадрата/куба осуществляется с клавиатуры.

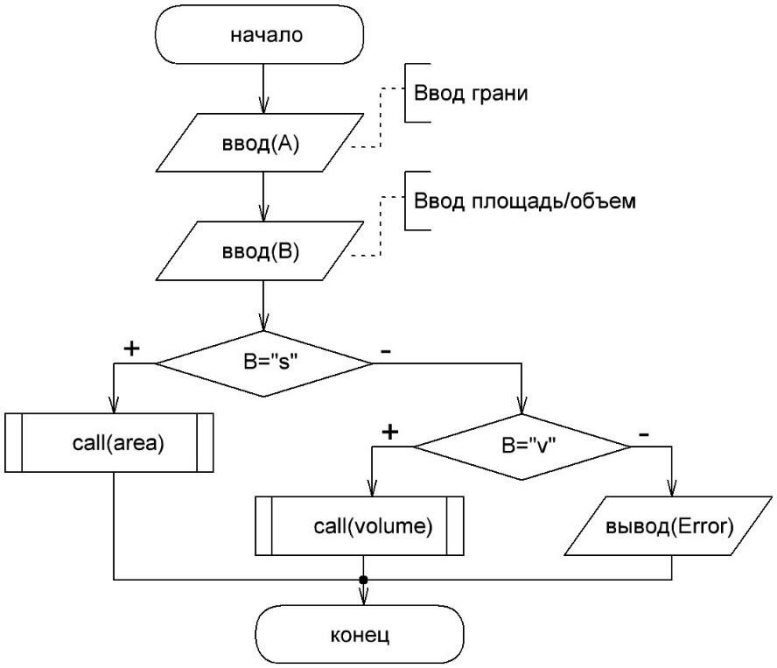


Рис. 2.22. Алгоритм вычисления площади квадрата

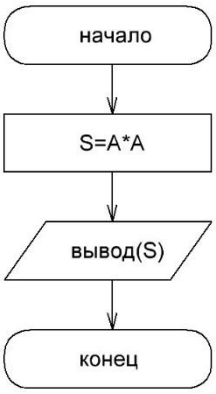


Рис. 2.23. Подпрограмма «area»

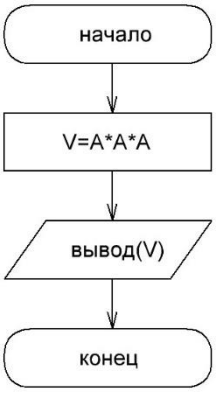


Рис. 2.24. Подпрограмма «volume»