АЛГОРИТМЫ. СХЕМЫ АЛГОРИТМОВ.

Понятие алгоритма

- **ГОСТ 34.003-90:** Конечный набор предписаний для получения решения задачи посредством конечного количества операций.
- **ГОСТ Р 52292-2004:** Конечное упорядоченное множество точно определенных правил для решения конкретной задачи.
- **ГОСТ 30721-2000:** Последовательность действий для определенного вычисления.
- **Научно-технический энциклопедический словарь:** Разложенный поэтапно набор команд или процедур, которым необходимо следовать для получения определенного результата из исходного набора вводных данных. Термин используют в КОМПЬЮТЕРНОМ ПРОГРАММИРОВАНИИ для обозначения последовательности команд в формате, пригодном для считывания компьютером, которая содержит ряд шагов для получения решения задачи.

Понятие алгоритма

$$y_1 = F_1(x_1, x_2, \ldots x_N)$$
$$y_2 = F_2(x_1, x_2, \ldots x_N)$$
$$\ldots$$
$$y_K = F_K(x_1, x_2, \ldots x_N)$$

Каждый ***выход*** является результатом действия ***алгоритма преобразования входов.***

Свойства алгоритмов

Алгоритм, как точно или явно определённый инструмент, для конструирования реальных процедур должен обладать следующими свойствами.

1) **Конструктивность**: Любые входы и выходы (интуитивно понятные и содержательные множества) должны быть конструктивно описаны в виде структур или типов данных. Любые интуитивно понятные функции (предикаты) должны быть конструктивно (формально) описаны в терминах определения инструмента.
2) **Детерминированность:** (следует из 1), т.к. интуитивно алгоритм предполагает реализацию функций и только их (заданному набору входных данных отвечает только один результат).
3) **Результативность:** Определение и распознавание начального (правильные входные данные) и конечного события (правильные выходные данные), связанного с правильным или неправильным результатом работы алгоритма.

Свойства алгоритмов

4) **Дискретность:** алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов. При этом для выполнения каждого шага алгоритма требуется конечный отрезок времени, то есть преобразование исходных данных в результат осуществляется во времени дискретно.
5) **Завершаемость (конечность)**: при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов.
6) **Понятность:** алгоритм для исполнителя должен включать только те команды, которые ему (исполнителю) доступны, которые входят в его систему команд.
7) **Массовость:** алгоритм должен быть применим к разным наборам исходных данных.

Эффективность алгоритмов (программ)

Научно-технический энциклопедический словарь: КОМПЬЮТЕРНАЯ ПРОГРАММА (ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ), набор расположенных поэтапно команд, позволяющих КОМПЬЮТЕРУ выполнить поставленную задачу…

Эффективность программы (алгоритма): минимальное время выполнения или минимальные ресурсы (память), требуемые для выполнения

Эффективность работы программиста: количество полезных (для пользователя) функций программы, реализуемых в единицу времени.

Эффективность программы и эффективность программиста — противоречивые требования.

Способы описания алгоритмов

- Естественный язык (при необходимости — с использованием математических формул) — текстовое описание последовательности шагов
  - (+) не нужны специальные знания
  - (-) громоздкость и неоднозначность
- Графическое описание (схема)
- Описание на искусственном языке (программа).

Описание алгоритма на искусственном языке

- Программа всегда имеет особенности, связанные с синтаксисом языка
- Для записи алгоритмов специально был придуман Pascal (в развитии — Modula2)
- Остальные ЯП могут быть «непрозрачны» с позиций понимания алгоритма (специфичные конструкции)
- Если не хотим писать программы сначала на Pascal’е, а потом на другом ЯП, остается «псевдокод».

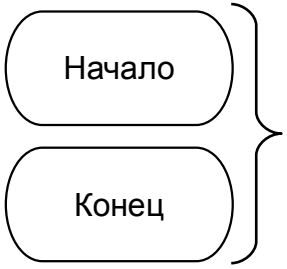
Схемы алгоритмов

Графическое изображение алгоритма — универсальный способ представления. Для процедурного (модульного, структурного) подхода алгоритм является описанием (моделью) какого-то процесса.

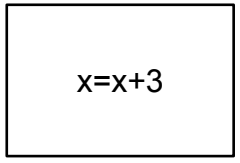
Стандарты и спецификации (правила) создания схем алгоритмов.

- Международный стандарт **ISO 5807-85** он же **ГОСТ 19.701-90** (ЕСПД. «Схемы алгоритмов, программ, данных и систем»).
- Диаграммы Насси-Шнейдермана (структурограммы)
- Спецификация IDEF3
- Диаграммы Гейна-Сарсона (data flow diagrams — DFD)
- Диаграммы состояний UML (**ISO/IEC 19505:2012**)
- Диаграммы деятельности UML
- Схемы BPMN
- …

Элементы схемы алгоритма по ГОСТ 19.701

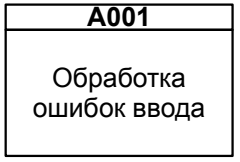


«Терминатор» (Знак завершения) – размещаются в начале и в конце схемы



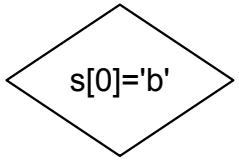
Действие («Процесс», операторный блок) – любые операции установки значений или модификации данных (вычислений)

«Данные» (ввод/вывод) – операции ввода (чтения) и вывода (отображения) данных

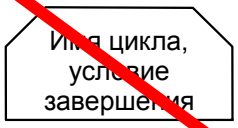


«Символ с полосой» – указывает на фрагмент, который детализирован далее. При детализации в блоках «начало» и «конец» указывается имя символа с полосой.

Элементы схемы алгоритма по ГОСТ 19.701

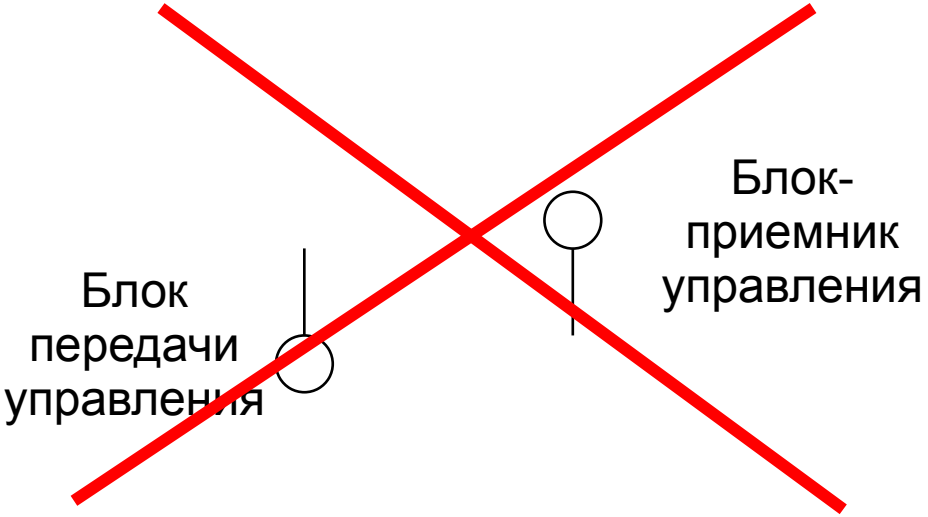


«Решение» (проверка условия) – определяет наличие различных вариантов дальнейшего выполнения алгоритма



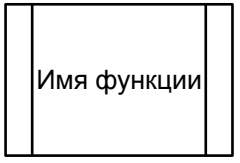
Начало цикла

Конец цикла



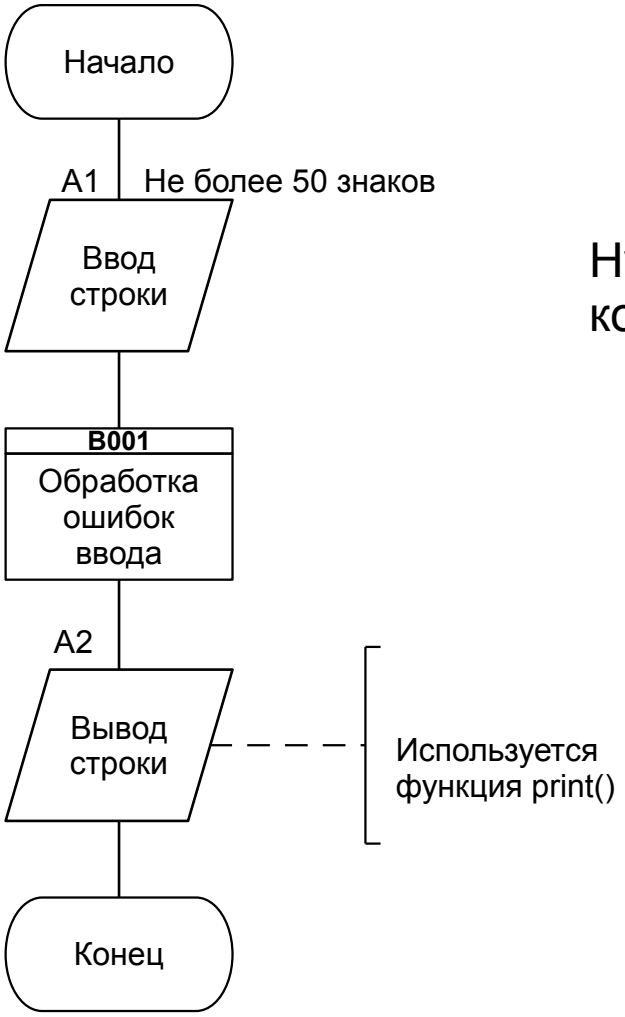
Блок передачи управления

Блок-приемник управления



Предопределенный процесс (функция)

Элементы схемы алгоритма по ГОСТ 19.701



Нумерация символов, пояснения и комментарии.

Правила создания схем алгоритмов (ГОСТ+)

- У каждого алгоритма и подалгоритма («вставки») — одно начало и один конец.
- Начало, конец, и «магистральный путь» выполнения алгоритма должны размещаться на одной вертикали.
- Элементы (символы) должны образовывать вертикальные и горизонтальные ряды.
- Линии не должны пересекаться.
- Размеры элементов: 3x1.5 см, 3x2 см или 4x2 см. Желательно придерживаться одних и тех же размеров.
- Алгоритм должен помещаться на одной странице. Для подробностей используются «вставки» – символ с полосой или предопределенный процесс.
- Допускается не более 20 элементов на странице А4.
- Размер шрифта — не более 12 пт (лучше 10 пт).
- Для аннотирования используются комментарии и описания символов (справа над символом). Идентификаторы символов — слева над символом. Описания и идентификаторы — не обязательны.

См. также: Паронджанов В.Д. Как улучшить работу ума без лишних хлопот. - М., ДМК-Пресс, 2010.

Структурный подход и схемы алгоритмов

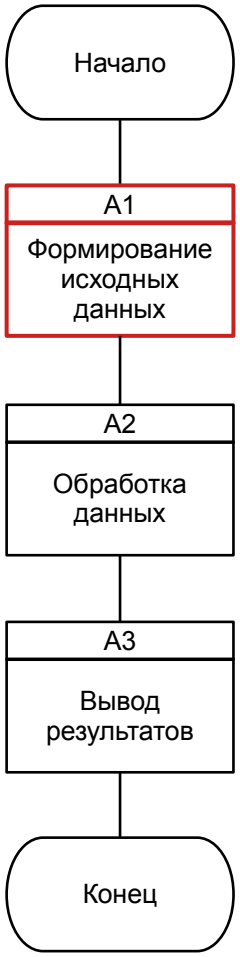
I. Любая программа строится из нескольких основных конструкций
– Конструкции для последовательного выполнения
– Конструкции, организующие ветвления
– Конструкции, реализующие циклические вычисления (циклы)

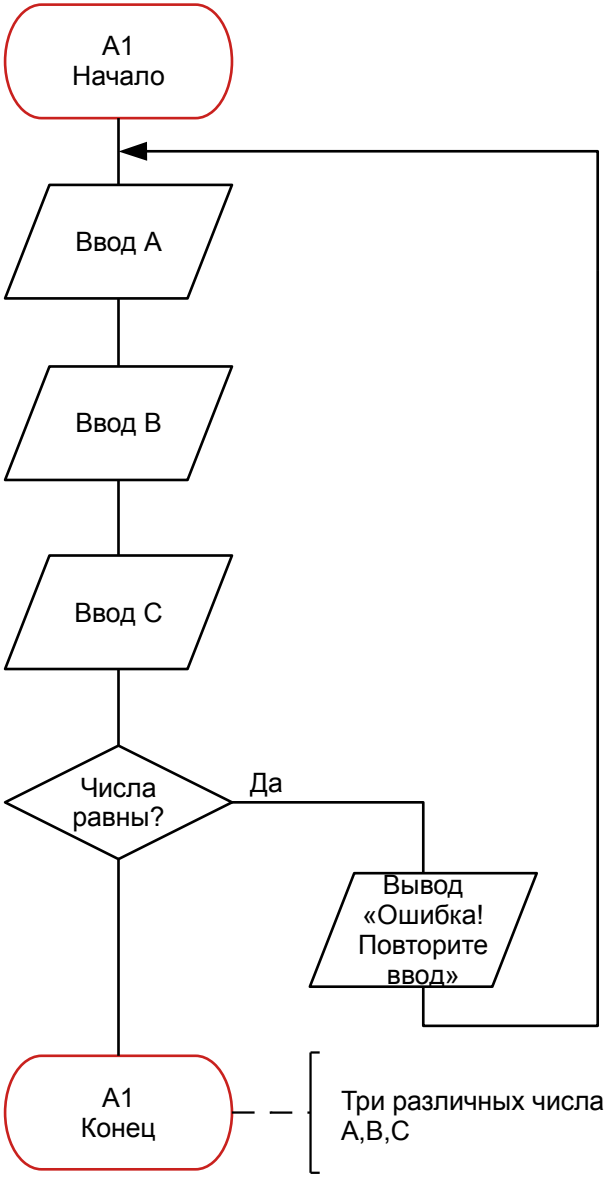
II. Повторяющиеся фрагменты программ оформляются как подпрограммы (процедуры или функции). В основной программе – вызовы подпрограмм.

III. Разработка программы ведется пошагово («сверху вниз»).

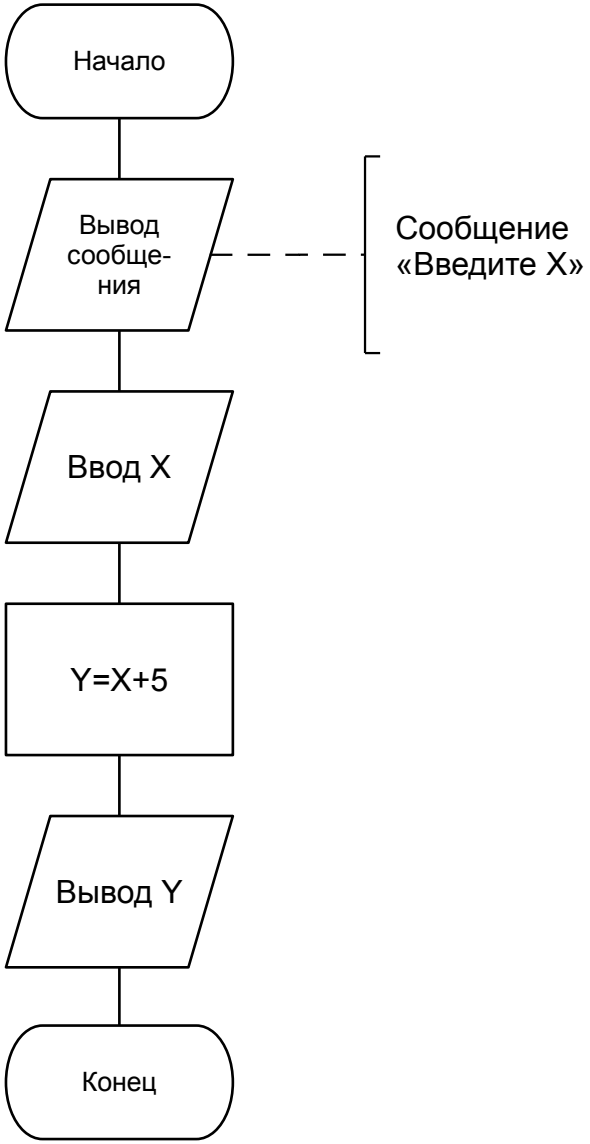
Результат: сокращается количество вариантов программы по одной и той же спецификации, упрощается отладка и понимание программы другими специалистами (сопровождение).

Последовательная декомпозиция схемы алгоритма

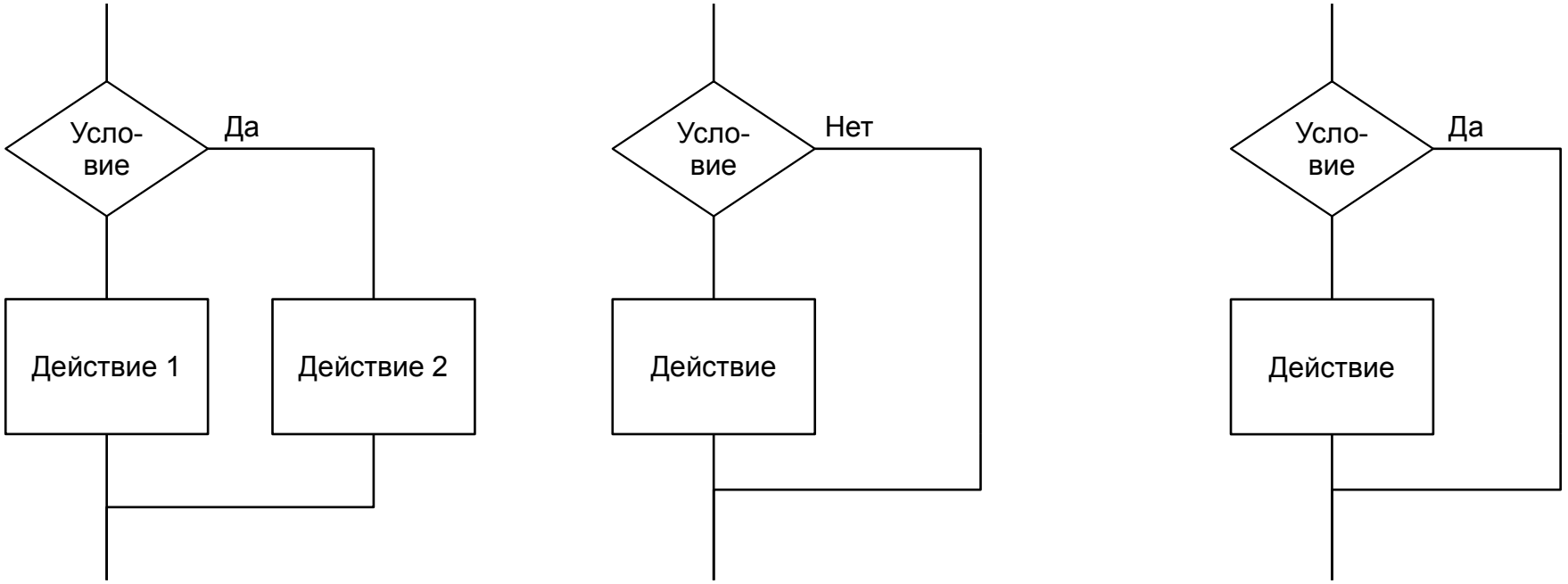




Простая последовательность операций (схема)



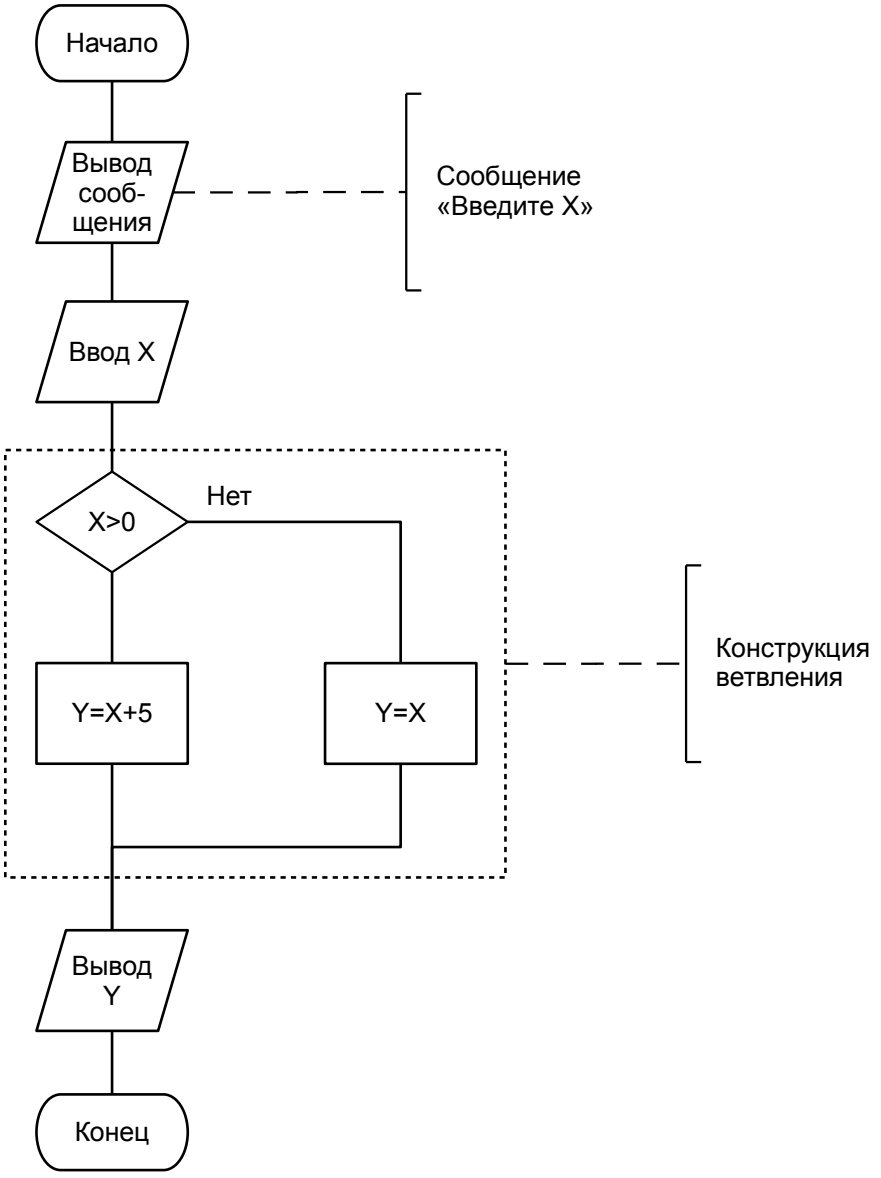
Ветвление выполнения операций (схема)



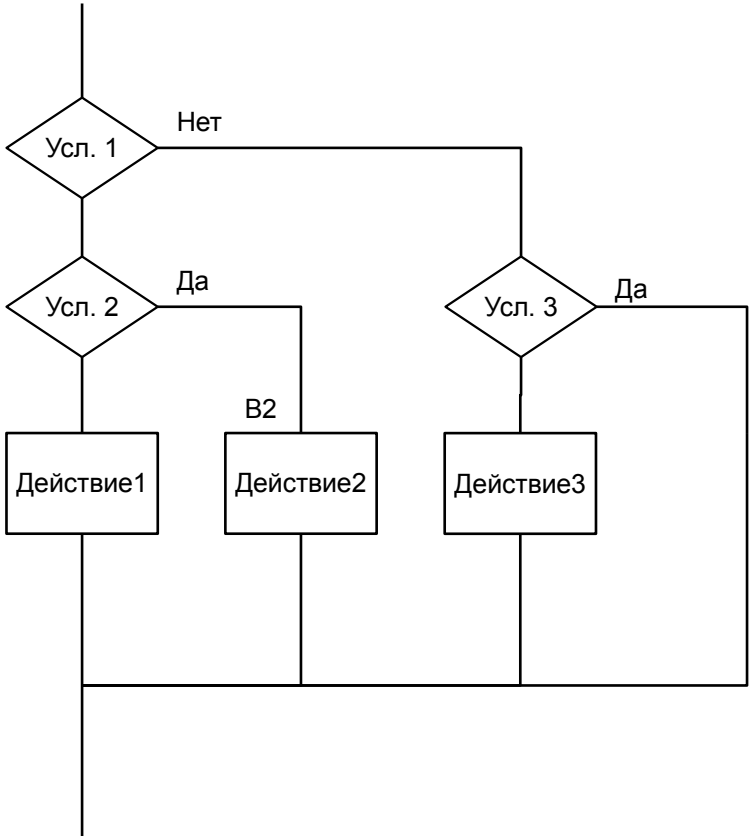
Одна из ветвей подписывается («Да» или «Нет»), в зависимости от условия.

Пример ветвления (схема)

$$y = \begin{cases} x+5, при\ x>0 \\ x, в\ других\ случаях \end{cases}$$



Вложенные ветвления (схема)



Наиболее «благоприятный» путь выполнения алгоритма – вертикаль от блока «Начало» до блока «Конец» (вертикальный шампур).

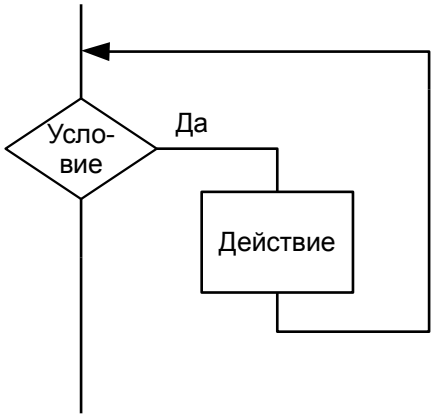
Все отклонения — только вправо.

Должен быть единый принцип отклонений («чем хуже — тем правее», «чем реже — тем правее» и т.п.)

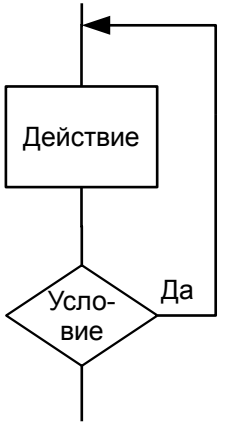
Идентификаторы блоков позволяют ссылаться на них в тексте (в документации), например «*в блоке B2 реализовано вычисление интеграла по поверхности*».

Циклические вычисления (схемы)

Цикл — действие или последовательность действий, выполняемых неоднократно (повторяющиеся действия)

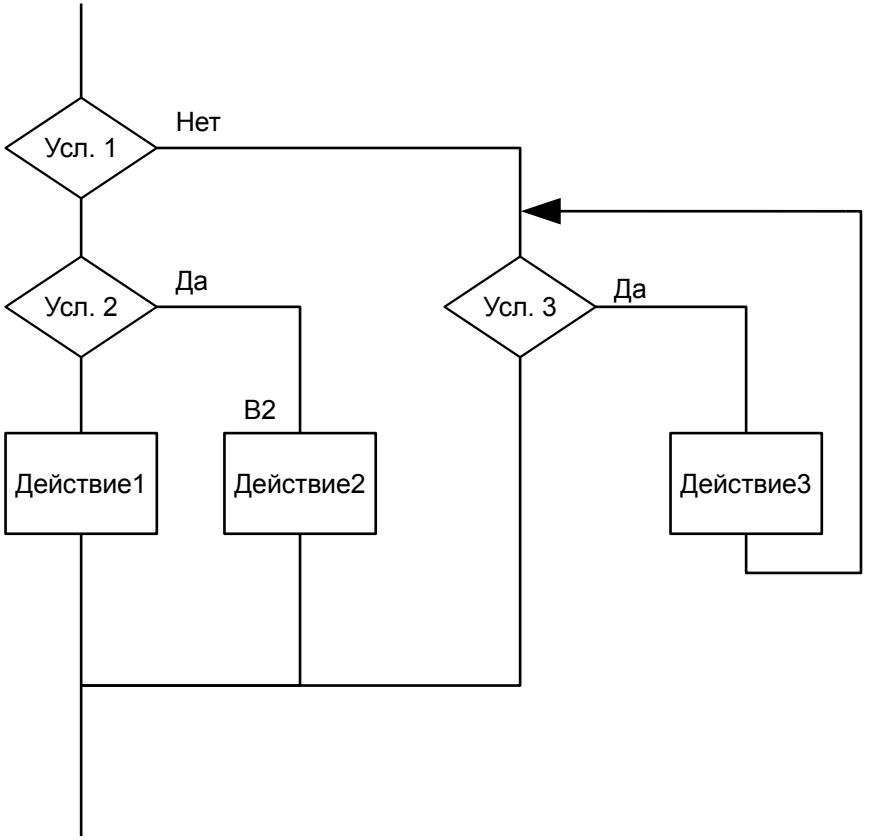


Цикл с пред-условием
(«Цикл ***while***»)



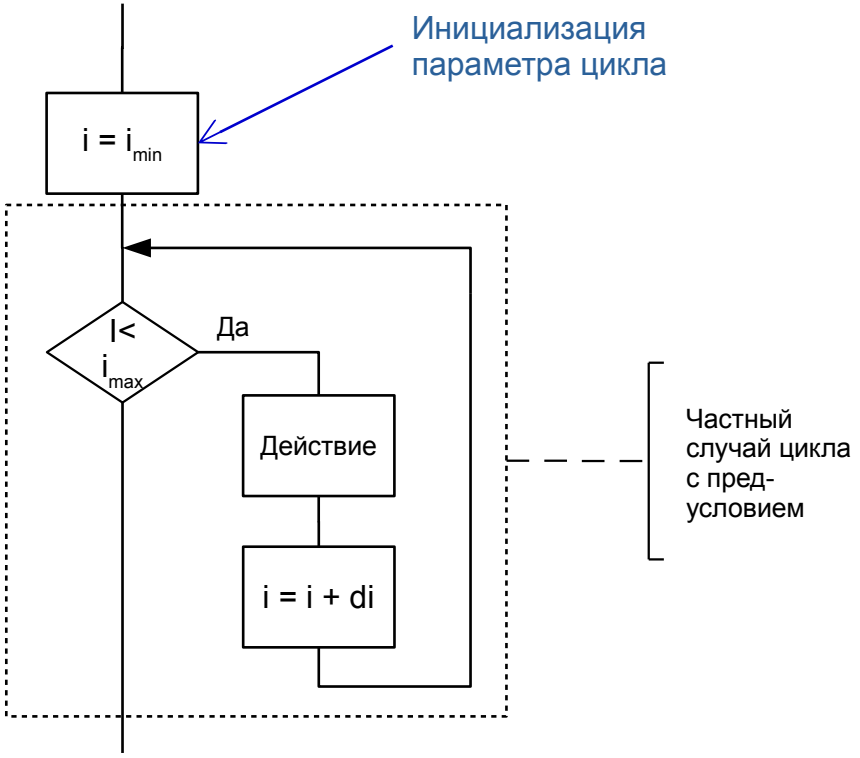
Цикл с пост-условием
(«Цикл ***do … while***»)

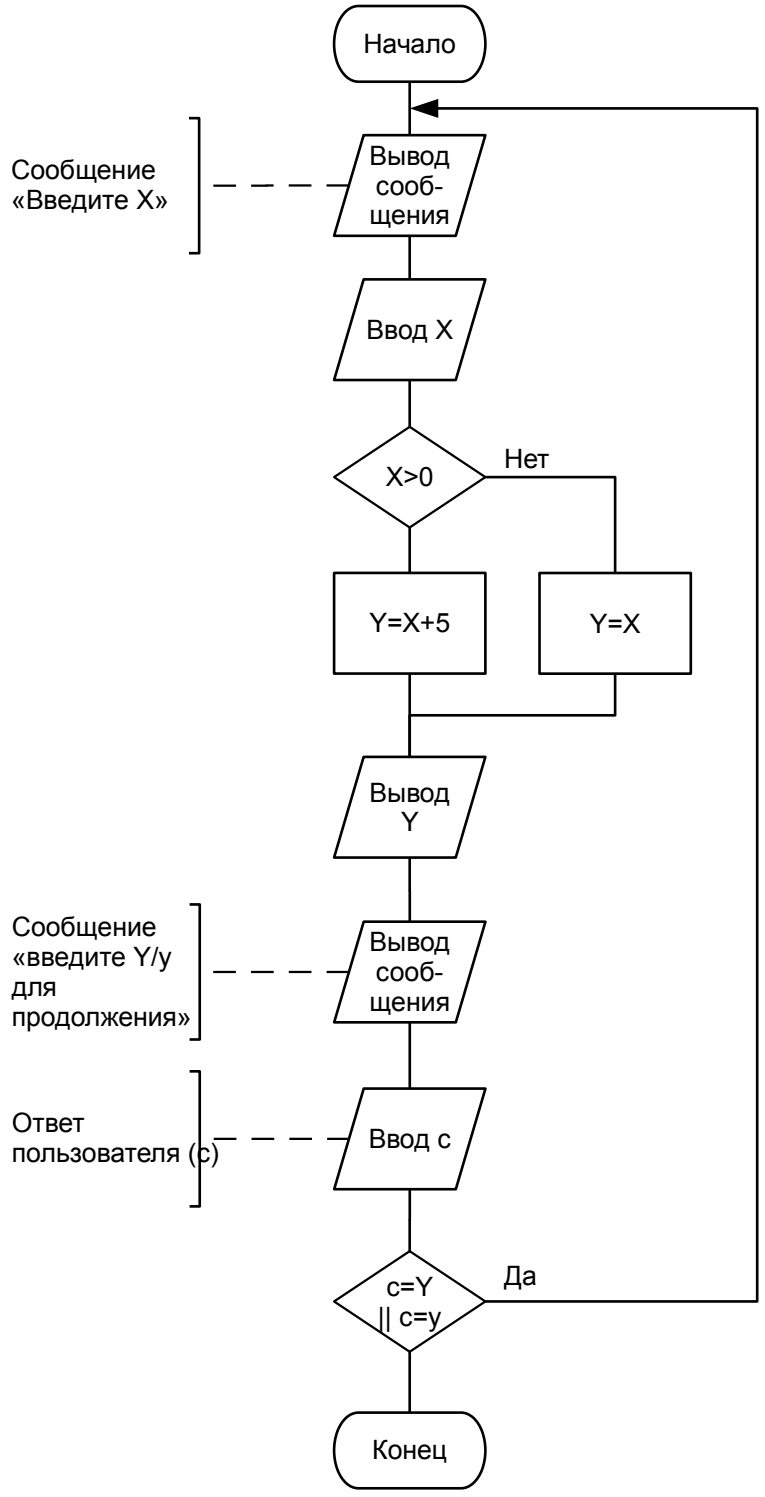
Циклические вычисления (схемы)



Циклические вычисления (схемы)

Цикл с параметром:
параметр цикла ***i*** изменяется от i_{min} до i_{max} с некоторым шагом ***di***



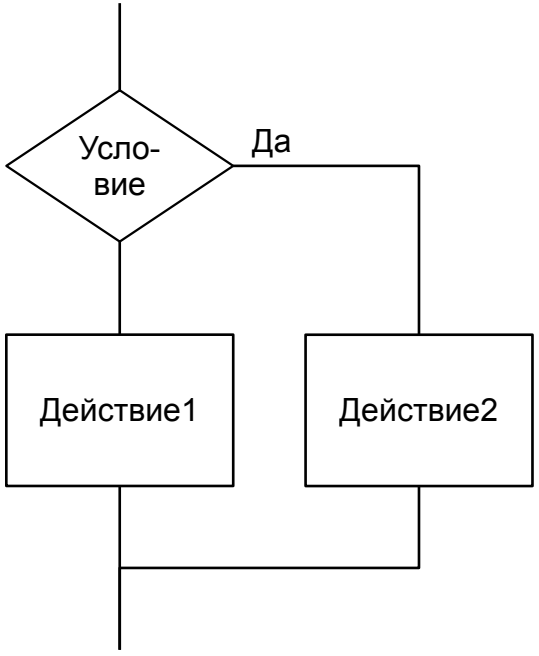


$$y = \begin{cases} x+5, при\ x>0 \\ x, в\ других\ случаях \end{cases}$$

Вычисляем многократно, для следующей попытки нужно нажать «Y» или «y»

Условные операторы в Си

Используется для реализации ветвлений (различные наборы операторов при выполнении или отсутствии выполнения условия).




```
if(<Условие>) <Действие2>;
else <Действие 1>;
```

или

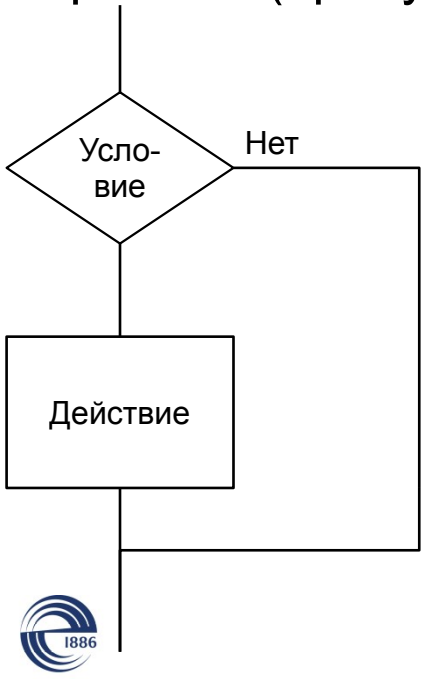
```
if(<Условие>)
{
   <Действие2-1>;
   <Действие2-2>;
   …
   <Действие2-n>;
}
else
{
    <Действие 1-1>;
   …
   <Действие 1-m>;
}
```

Условные операторы в Си

Если условие выполнено (результат выражения <Условие> не равен 0), выполняется Действие2 (если это одиночный оператор) или составной оператор после if() («ветка Да»).

В противном случае выполняется Действие1 или составной оператор после else.

Варианты (пропуск действий)




```
if(<Условие>) <Действие>;
```

или

```
if(<Условие>)
{
    <Действие-1>;
    <Действие-2>;
    …
    <Действие-n>;
}
```

<Условие> можно написать по-разному.

Условные операторы в Си

```
if(x>5) y=2; /* y получит значение 2, если x>5 */

if(x=5) y=2; /* y всегда получает значение 2, т.к. x получил значение 5 */
             /* (true) */

if(x==5) y=2; /* y получит значение 2, если x имеет значение 5 */


if(x==5)
{
   y=2;    /* при выполнении условия выполняются все операторы в { } */
   q=0;    /* (составной оператор) */
   z=3;
}

if(x==5)
   y=2;    /* при выполнении условия выполняется только первый оператор */
   q=0;    /* («;» – конец оператора if) */
   z=3;    /* остальные операторы выполняются всегда. */
```

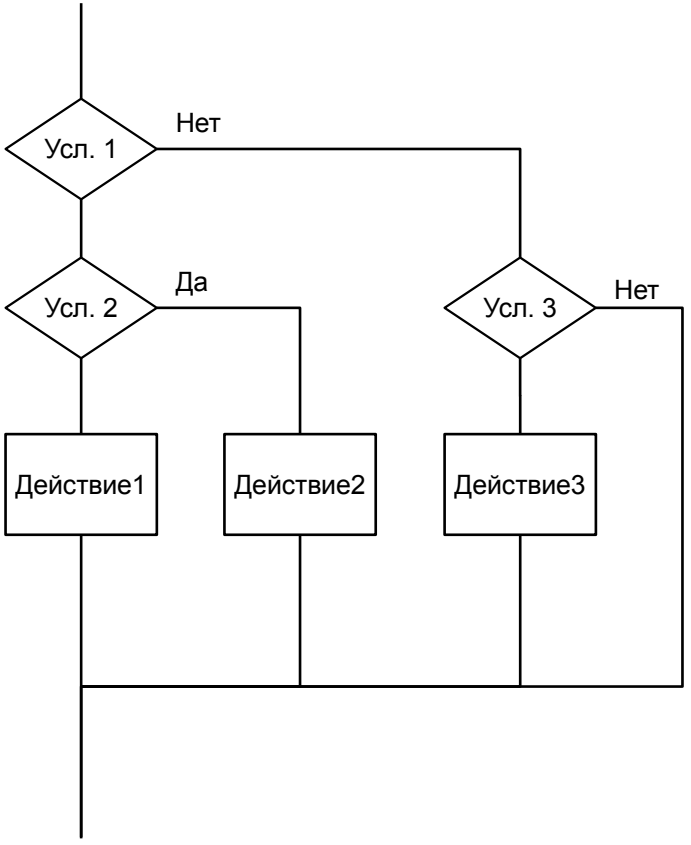
Условные операторы в Си

```
if(x>5) y=2;     /* y получит значение 2, если x>5 */
else y=7;        /* в противном случае y получит значение 7 */

if(x>5) y=2;     /* y получит значение 2, если x>5 */
else
{
   y=7;          /* в противном случае y получит значение 7, */
   q=2.5;        /* q получит значение 2.5 */
}

if(x<=0) y=2;
else if((x>0)&&(x<=3)) y=0;
else y=1;
```

Количество **`else if()`** не ограничивается.

В каждой ветви (**`if, else, else if`**) может быть как одно выражение, так и составной оператор.

Условные операторы в Си




```
if(<Усл. 1>)
{
   if(<Усл. 2>) <Действие2>;
   else <Действие1>;
}
else
{
    if(<Усл. 3>) <Действие3>;
}
```

В практике программирования вложенные условные операторы используются довольно часто.

Условные операторы в Си

Резюмируя:

Условные операторы позволяют организовать ветвления в программе.
Структура ***if() – else*** позволяет выбрать один оператор или группу операторов из двух.
Одиночный ***if()*** позволяет выполнить оператор (или группу операторов) или пропустить действия.

else *связывается с ближайшим предыдущим* ***if()****, не содержащим* ***else****.*

Оператор ***if*** является вложенным, если он находится внутри другого оператора ***if*** или ***else***. Во вложенном условном операторе «ветка» ***else*** всегда ассоциирована с ближайшим ***if*** в том же блоке, если этот ***if*** не ассоциирован с другой «веткой» ***else***.

Операторные скобки { … } и отступы помогают разобраться в структуре вложенного ***if***.

Выбор из нескольких вариантов

Задача:

В зависимости от полученного символа выполнить различные действия с исходным значением X.

- Если получен символ «*», X нужно умножить на 2
- Если получен символ «/», X нужно разделить на 2
- Если получен символ «-», от X нужно отнять 2
- Если получен символ «+», к X нужно прибавить 2
- Любые другие символы проводят к сообщению «Ничего не делаем».

Вывести итоговое значение X.

Можно использовать ***if() - else if() - else***
(пример lect-02-01.c)

Выбор из нескольких вариантов

Оператор выбора
(пример lect-02-02.c)

После ***case:*** может быть как простой, так и составной оператор.

break; обеспечивает завершение оператора выбора (иначе будут просматриваться все варианты)

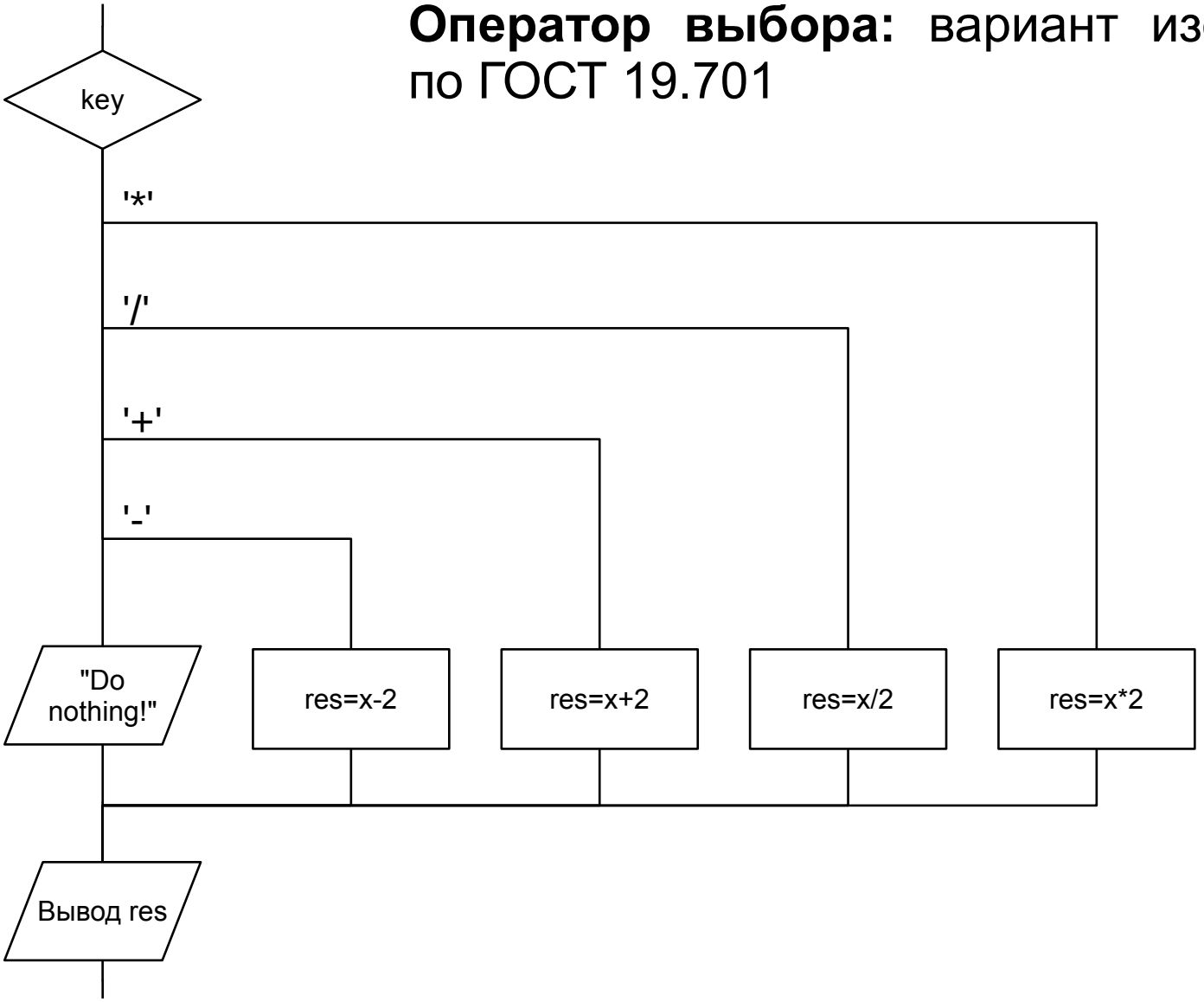
default: если не реализован ни один из вариантов.

Возможен вариант

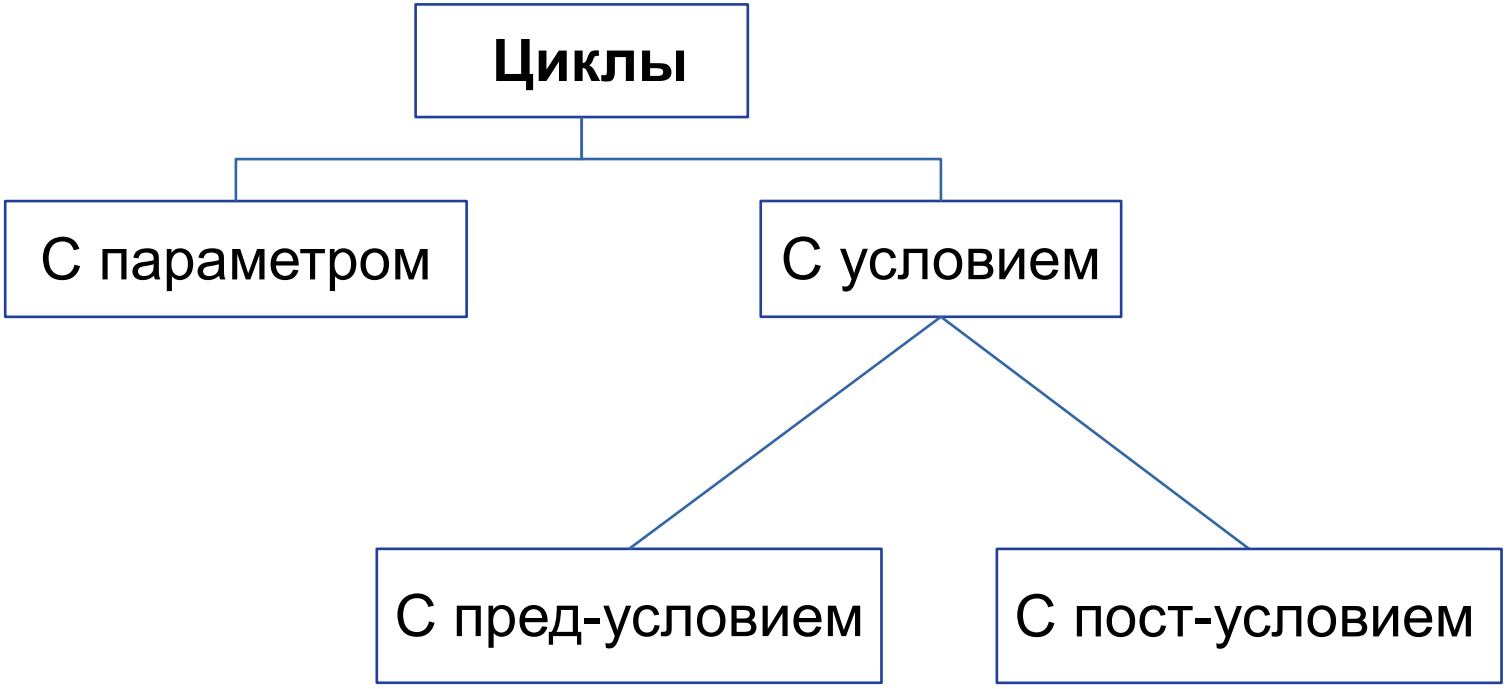
```
case 'a':
case 'A': puts("Pressed key A");
   break;
```

Выбор из нескольких вариантов

Оператор выбора: вариант изображения по ГОСТ 19.701



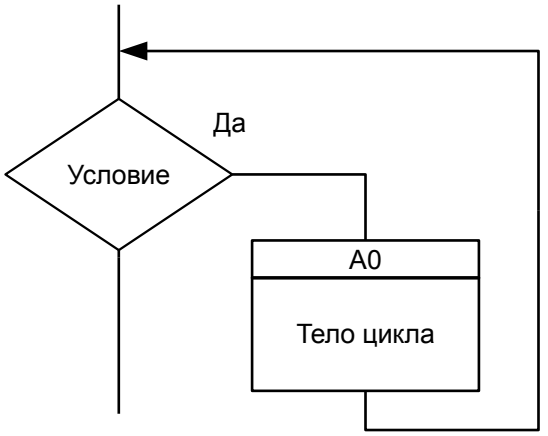
Циклические вычисления в Си



Цикл с предусловием

Цикл while()

```
while(<условие>)
{
   <тело цикла>
};
```



В теле цикла может быть один оператор, тогда { } не нужны:

```
while(<условие>) <действие>;
```

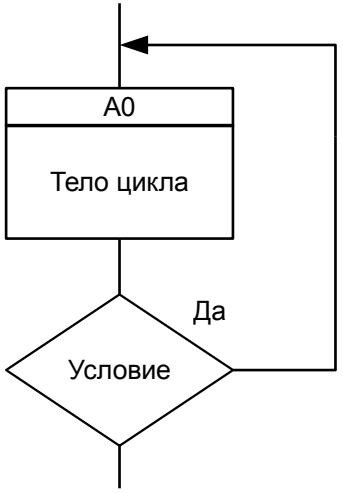
Действие или тело цикла выполняется, пока выполняется условие (выражение <условие> должно иметь значение «истина»).
Цикл завершается, когда <условие> принимает значение «ложь».

Примеры:
lect-02-03.c
lect-02-04.c

Цикл с постусловием

Цикл do … while()

```
do
{
    <тело цикла>
} while(<условие>);
```



Тело цикла выполняется, пока выполняется условие (выражение <условие> должно иметь значение «истина»), но в любом случае выполнится один раз.
Цикл завершается, когда <условие> принимает значение «ложь».

Пример lect-02-05.c

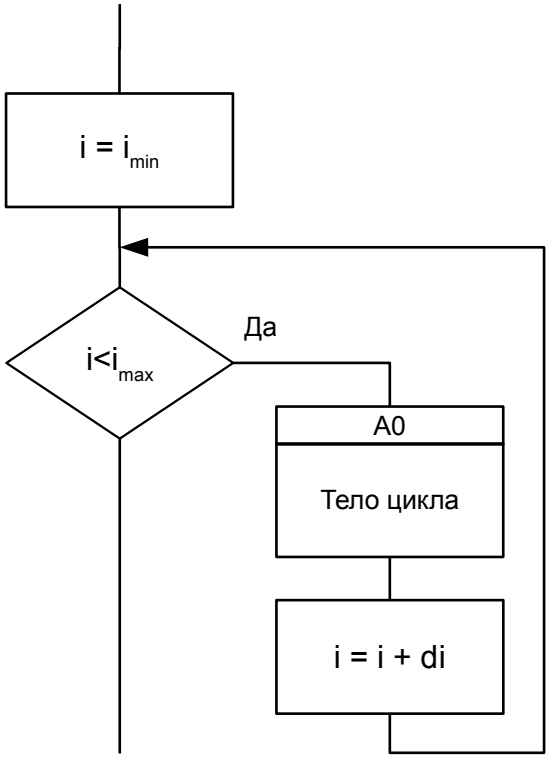
Цикл с параметром

Цикл for():

```
for(<инициализация параметров>;<условие>;<изменение параметров>)
{
    <тело цикла>
}
```

Примеры:
lect-02-06.c
lect-02-07.c

Д/З:
Построить схемы алгоритмов для этих примеров.



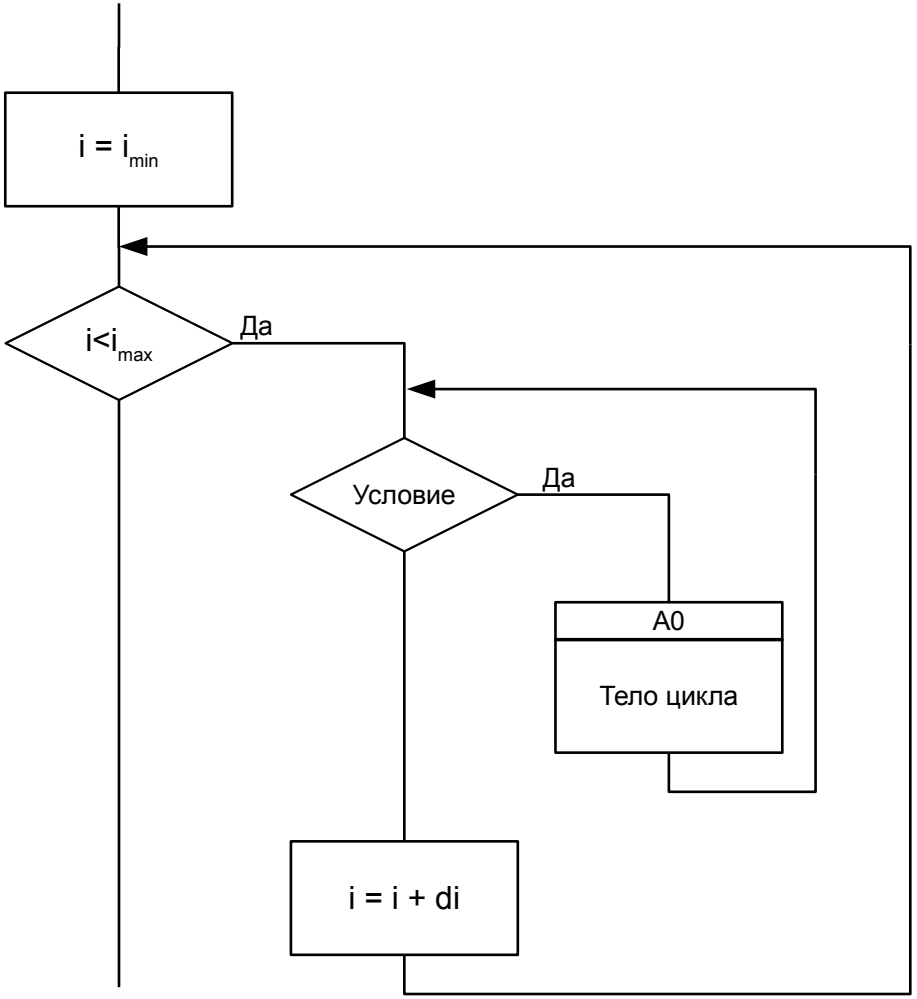
Цикл с параметром

Цикл for():

Используется тогда, когда количество повторений цикла заранее известно или может быть вычислено.

- Цикл **for** состоит из заголовка и тела цикла.
- В заголовке после слова **for** в круглых скобках записываются через точку с запятой три выражения:
  - **начальные значения**: операторы присваивания, которые выполняются один раз перед выполнением цикла;
  - **условие, при котором выполняется следующий шаг цикла**; если условие неверно, работа цикла заканчивается; если оно неверно в самом начале, цикл не выполняется ни одного раза (*цикл с предусловием*, то есть условие проверяется перед выполнением цикла);
  - **действия в конце каждого шага** цикла (в большинстве случаев это операторы присваивания).

Вложенные циклы



В зависимости от задачи (алгоритма) возможны различные варианты вложенных циклов.

Вложенный цикл по отношению к циклу в тело которого он вложен будет именоваться ***внутренним циклом***, и наоборот цикл в теле которого существует вложенный цикл будет именоваться ***внешним*** по отношению к вложенному.

Внутри вложенного цикла в свою очередь может быть вложен еще один цикл, образуя следующий уровень вложенности и так далее.

Количество уровней вложенности, как правило, не ограничивается.

Бесконечный цикл и досрочный выход из цикла

```
while(true)
{
    <тело цикла>
}
```

или

```
for(;;)
{
    <тело цикла>
}
```

Ключевое слово ***break;*** при выполнении некоторого условия внутри цикла обеспечивает завершение цикла (выход из цикла).
Структурный подход к программированию запрещает использование ***break;*** во всех случаях кроме ***switch … case***.

Для выхода из цикла по условию можно изменить условие выполнения цикла.

Эквивалентность for() и while()

```
#include <stdio.h>
int main(){
int x, s;
x=s=0;
while(x!=9999)
{
    printf("Enter value: ");
    scanf("%d", &x);
    if(x!=9999) s = s+x;
}
printf("Cumulative sum: %d\n", s);
return 0;
}
```

```
#include <stdio.h>
int main(){
int x, s;
for(s=0, x=0; x!=9999; )
{
    printf("Enter value: ");
    scanf("%d", &x);
    if(x!=9999) s = s+x;
}
printf("Cumulative sum: %d\n", s);
return 0;
}
```

Организация простого меню

Сочетание цикла (**`while()`** или **`do… while()`**) с оператором выбора (**`switch-case`**) позволяет организовать простое меню.

Пример lect-02-08.c

Применена условная компиляция: очистка экрана реализуется разными командами в зависимости от операционной системы с использованием вызова функции **`system()`** (обеспечивает вызов команды операционной системы, определена в **`stdlib.h`**).