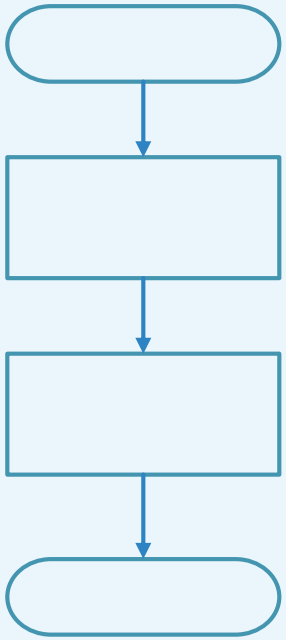
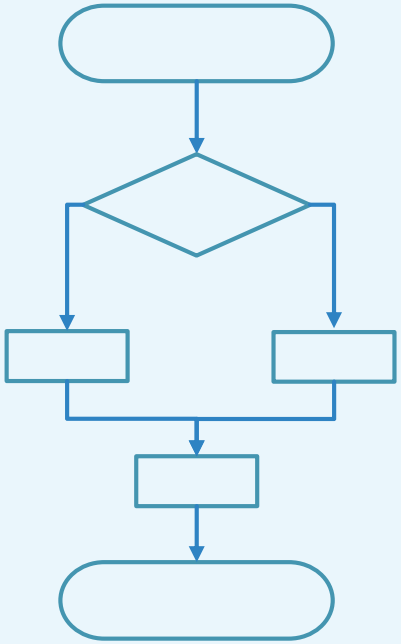
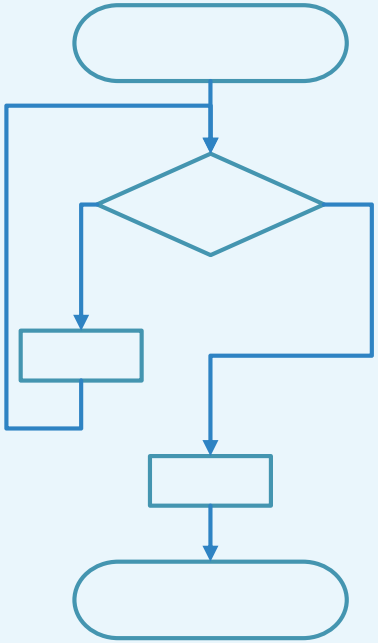
Операторы и выражения

Лекция 3

Оператор

- Оператор – действие в программе.
- Оператор заканчивается «;»
- Пустой оператор – «;»
- Оператор-выражение – оператор, имеющий значение
- Оператор управления – оператор, управляющий выполнением программы
- Оператор условия – позволяют выполнять разные действия при выполнении разных условий
- Оператор множественного выбора – похож на оператор условия
- Оператор цикла – позволяет повторять некоторые действия в программе много раз
- Составной оператор – блок операторов, заключенный в фигурные скобки

Последовательность действий в программе

Линейная	Ветвление	Повторы
		

Оператор условия `if / else`

- Позволяет выполнять определенные действия в зависимости от истинности условия
- Существует в 2 вариантах:
  - первый

```
if (x > y)
        std::cout << 1 << std::endl;
```

  - второй

```
if ( x > y)
        std::cout << 1 << std::endl;
else
        std::cout << 0 << std::endl;
```

- Может быть заменен условной операцией:

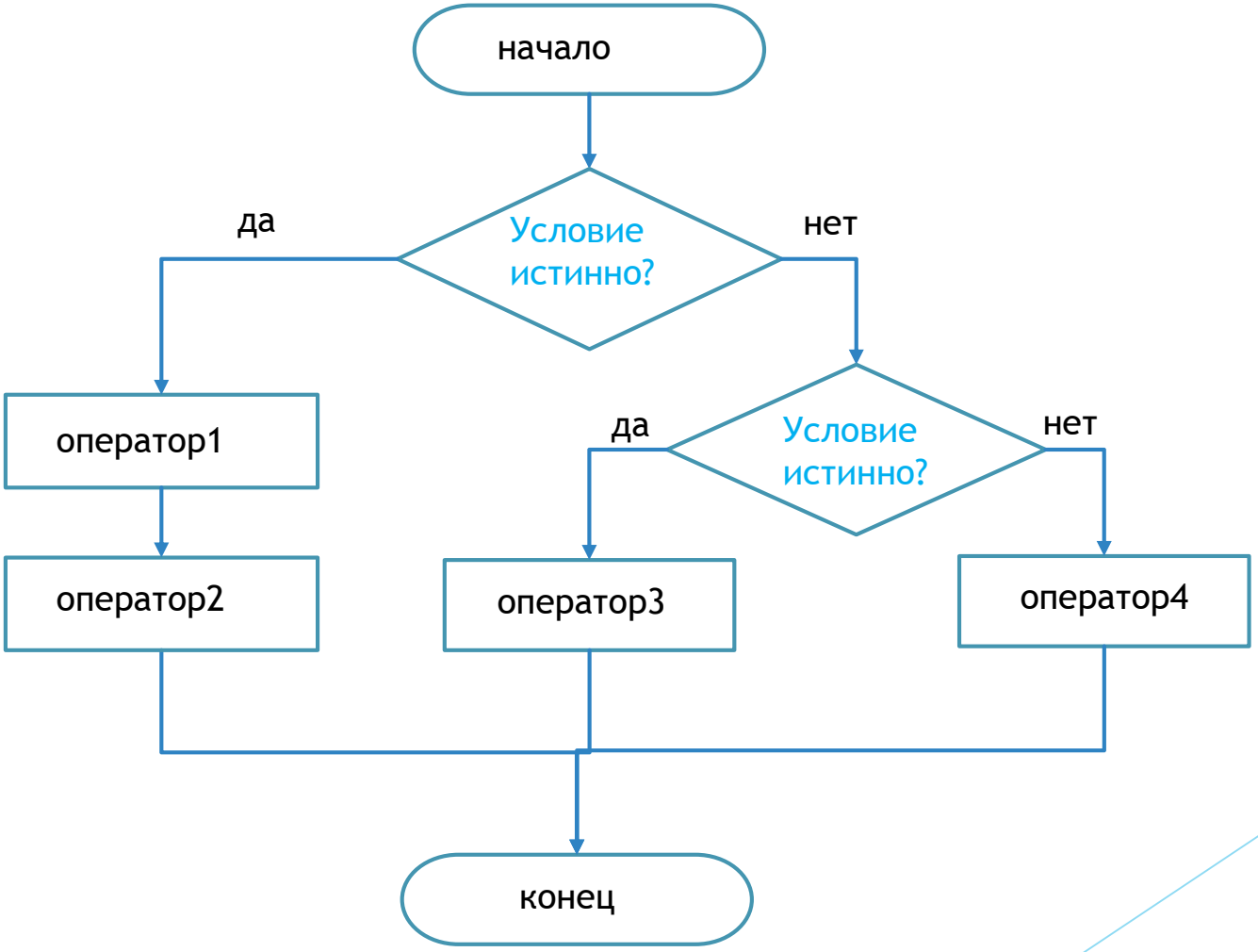
```
std::cout << (x > y ? 1 : 0 ) << std::endl;
```

Оператор условия `if / else`

С помощью этого оператора можно проверять несколько условий:

```
if (x > y)
    std::cout << 1 << std::endl;
else if (x > z)
    std::cout << 2 << std::endl;
else if (y > z)
    std::cout << 3 << std::endl;
else
    std::cout << 4 << std::endl;
```

Оператор условия `if / else`



Составной оператор `if / else`

Если при выполнении условия требуется выполнить сразу несколько действий, то их можно объединить с помощью фигурных скобок `{}`

```
if (x > y)
{
        std::cout << 1 << std::endl;
        ++x;
}
```

Хорошо!!!
Всегда заключайте блок `if / else`
в фигурные скобки `{}`

Вложенный оператор `if / else`

- Операторы **`if / else`** можно вкладывать друг в друга в любом порядке
- **`else`** всегда относится в ближайшему **`if`**

```
int x = 5, y = 6, z = 7;
if (x > y)
{
        if (x > z)
                std::cout << x << std::endl;
        else if (y > z)
                std::cout << y << std::endl;
        else
                std::cout << z << std::endl;
}
```

Ничего не будет напечатано, так как **`x`** не больше **`y`**

Вложенный оператор

Вложенный оператор `if / else`

- Изменить порядок принадлежности блока **`else`** можно с помощью фигурных скобок **`{ }`**

```
int x = 5, y = 6, z = 7;
if (x > y)
{
    if (x > z)
        std::cout << x << std::endl;
}
else if (y > z) {
        std::cout << y << std::endl;
}
else {
        std::cout << z << std::endl;
}
```

Вложенный блок

Результат:

```
7
```

Оператор `if /else`

Ожидаемую часть следует располагать в части **`if`**, исключение — в части **`else`**. Это позволяет убедиться, что исключения не вносят неясности в нормальный ход выполнения. Важно для читаемости и производительности

```
bool isOk = readFile (fileName);
if (isOk) {
    std::cout << "OK"    << std::endl;
}
else {
    std::cout << "Error" << std::endl;
}
```

Оператор if /else

Старайтесь избегать сложных условных выражений. Лучше вводить логические переменные. Это приведет к самодокументированию программы. Ее будет легче читать, отлаживать, поддерживать.

Сравните:

```
bool isFinished = (elementNo < 0) ||
                  (elementNo > maxElement);
bool isRepeatedEntry = (elementNo == lastElement);

if (isFinished || isRepeatedEntry) {
     std::cout << "Done" << std::endl;
}
```

```
if ((elementNo < 0) || (elementNo > maxElement) ||
(elementNo == lastElement)) {
     std::cout << "Done" << std::endl;
}
```

Оператор if /else

Старайтесь избегать вызова функций в условии. Они усложняют читаемость и отладку.

Сравните:

```
File* fileHandle = open(fileName, "w");
if (!fileHandle) {
    std::cout << "Can't open file" << std::endl;
}
```

```
if (!open(fileName, "w")) {
    std::cout << "Can't open file" << std::endl;
}
```

Оператор if /else

Правильно подбирайте условия

```
if (grade > 40) {
    std::cout << "You got 5!" << std::endl;
}
if (grade > 30 && grade <= 40) {
    std::cout << "You got a 4!" << std::endl;
}
if (grade > 20 && grade <= 30) {
    std::cout << "You got a 3!" << std::endl;
}
```

Плохо!!!

```
if (grade > 40) {
    std::cout << "You got 5!" << std::endl;
}
else if (grade > 30) {
    std::cout << "You got 4!" << std::endl;
}
else if (grade > 20) {
    std::cout << "You got 3!" << std::endl;
}
```

Хорошо!!!

Оператор множественного выбора `switch`

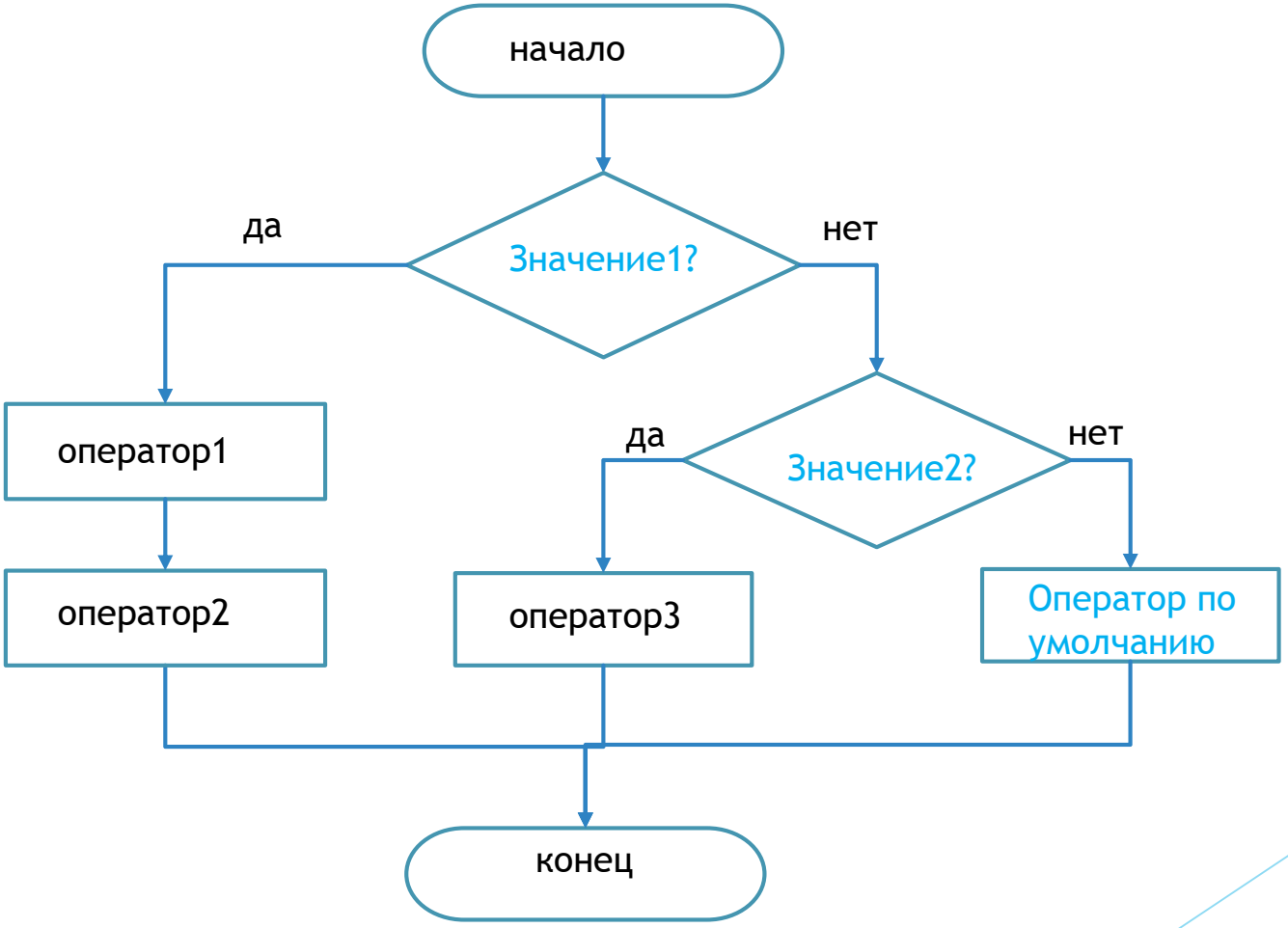
Позволяет выбрать один из многих вариантов хода выполнения программы в зависимости от значения выражения

Оператор множественного выбора **switch**

Синтаксис:

```
switch (<переменная_или_выражение>)
 {
 case <константное_выражение1>:
      <группа_операторов>;
      break;
 case <константное_выражение2>:
      <группа операторов>;
      break;
 //.     .     .
 default :
      <группа операторов>;
 }
```

Оператор `switch`

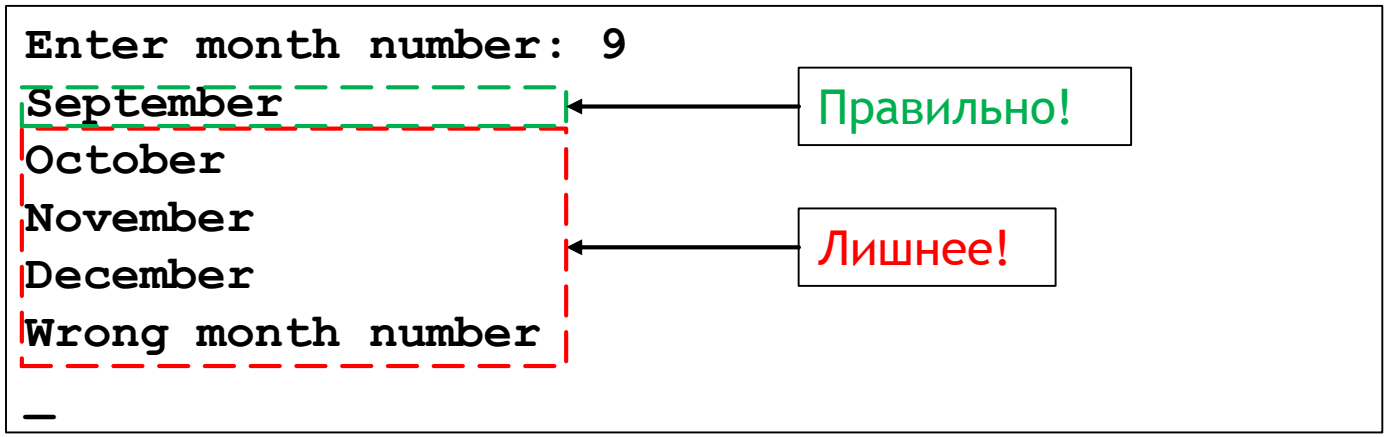


Напечатать название месяца по его номеру

```
int monthNo=0;
std::cout << "Enter month number: ";
std::cin >> monthNo;
switch (monthNo)
{
case 1:
     std::cout << "January" << std::endl;
case 2:
     std::cout << "February" << std::endl;
…
case 12:
     std::cout << "December" << std::endl;
default:
     std::cout << "Wrong month number" << std::endl;
}
```

Напечатать название месяца по его номеру

Пример работы программы:



Напечатать название месяца по его номеру - исправления

```
switch (monthNo)
{
case 1:
     std::cout << "January" << std::endl;
     break;
case 2:
     std::cout << "February" << std::endl;
     break;
…
case 12:
     std::cout << "December" << std::endl;
     break;
default:
     std::cout << "Wrong month number" << std::endl;
}
```

Теперь правильно:

```
Enter month number: 9
September
_
```

Напечатать название сезона по номеру месяца

```
switch (monthNo)
{
case 1:
     std::cout << "Winter" << std::endl;
     break;
case 2:
     std::cout << "Winter" << std::endl;
     break;
…
case 12:
     std::cout << "Winter" << std::endl;
     break;
default:
     std::cout << "Wrong month number" << std::endl;
}
```

Программа работает правильно, но в каждом сезоне по три месяца!

```
Enter month number: 9
Autumn

_
```

Напечатать название сезона по номеру месяца - оптимизация

```
switch (monthNo)
{
case 1: case 2: case 12:
     std::cout << "Winter" << std::endl;
     break;
case 3: case 4: case 5:
     std::cout << "Spring" << std::endl;
     break;
…
case 9: case 10: case 11:
     std::cout << "Autumn" << std::endl;
     break;
default :
     std::cout << "Wrong month number" << std::endl;
}
```

Объединение нескольких констант

```
Enter month number: 9
Autumn
_
```

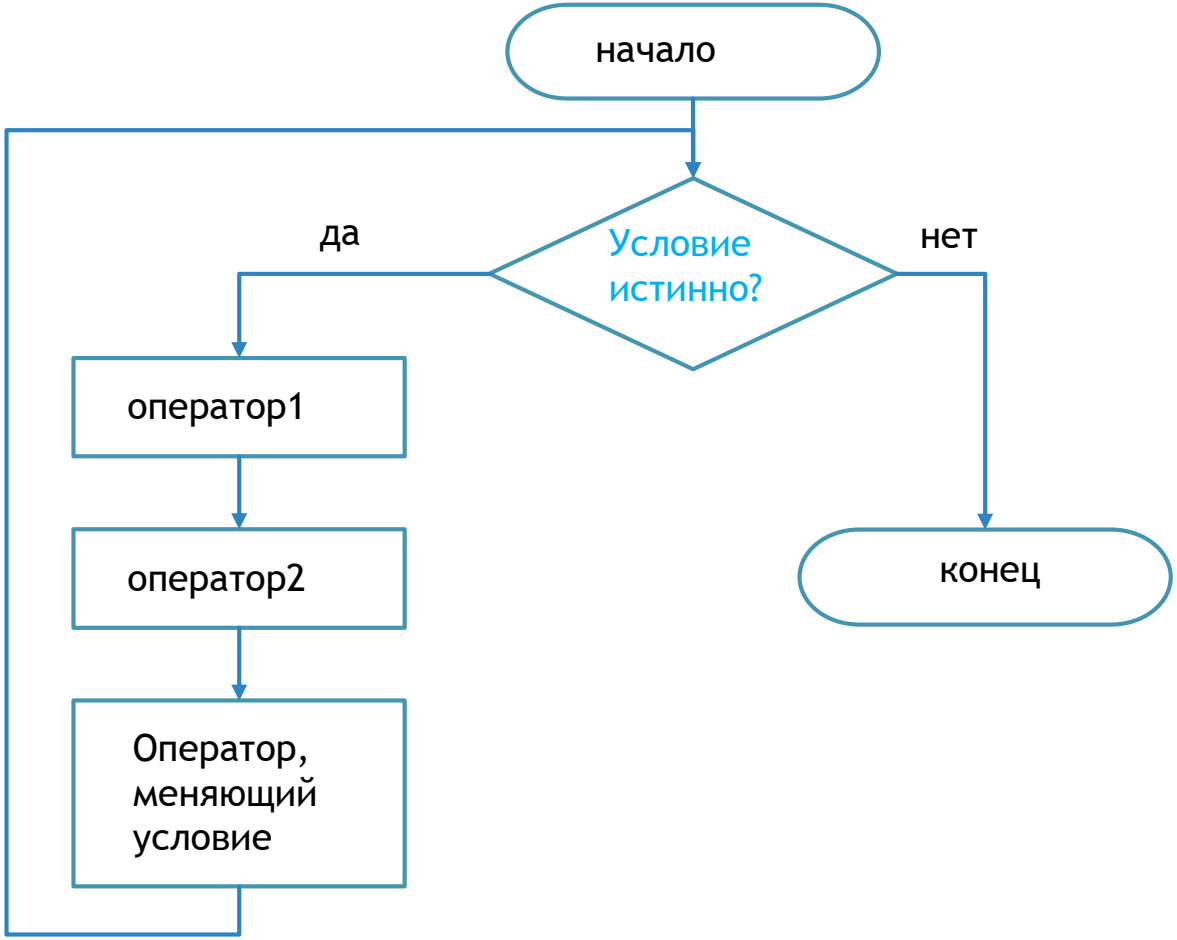
Операторы цикла

- Циклы позволяют повторять один и тот же кусок кода любое количество раз
- Виды циклов:
  - С предусловием
    - `for`
    - `while`
  - С постусловием
    - `do/while`

Оператор цикла `while`

- Алгоритм цикла **`while`**:
  1. Сначала проверяется условие цикла **`while`**
  2. Если условие истинное, выполняется тело цикла **`while`**. Возврат к первому пункту
  3. Если условие ложное – выход из цикла
- Цикл **`while`** может не выполниться ни разу, если при первой проверке условие будет ложным
- Тело цикла **`while`** должно содержать инструкцию, изменяющую истинность условия цикла
- Цикл **`while`** обычно используется, когда заранее не известно число повторов

Оператор цикла **while**



Рассчитать среднюю оценку

```
int counter = 0, grade = 0, total = 0;
float average = 0.0;
std::cout << "Enter the grade or -1 for stopping: ";
std::cin >> grade;
while (grade != -1)
{
    total += grade;
    ++counter;
    std::cout << "Enter the grade or -1 for stopping: ";
    std::cin >> grade;
}
average = (float)total / counter;
std::cout << "The average grade is " << average << std::endl;
```

Рассчитать среднюю оценку

Пример работы программы:

```
Enter the grade or -1 for stopping: 5
Enter the grade or -1 for stopping: 4
Enter the grade or -1 for stopping: 5
Enter the grade or -1 for stopping: 4
Enter the grade or -1 for stopping: 5
Enter the grade or -1 for stopping: 5
Enter the grade or -1 for stopping: 5
Enter the grade or -1 for stopping: 4
Enter the grade or -1 for stopping: 3
Enter the grade or -1 for stopping: 4
Enter the grade or -1 for stopping: -1
The average grade is 4.4
```

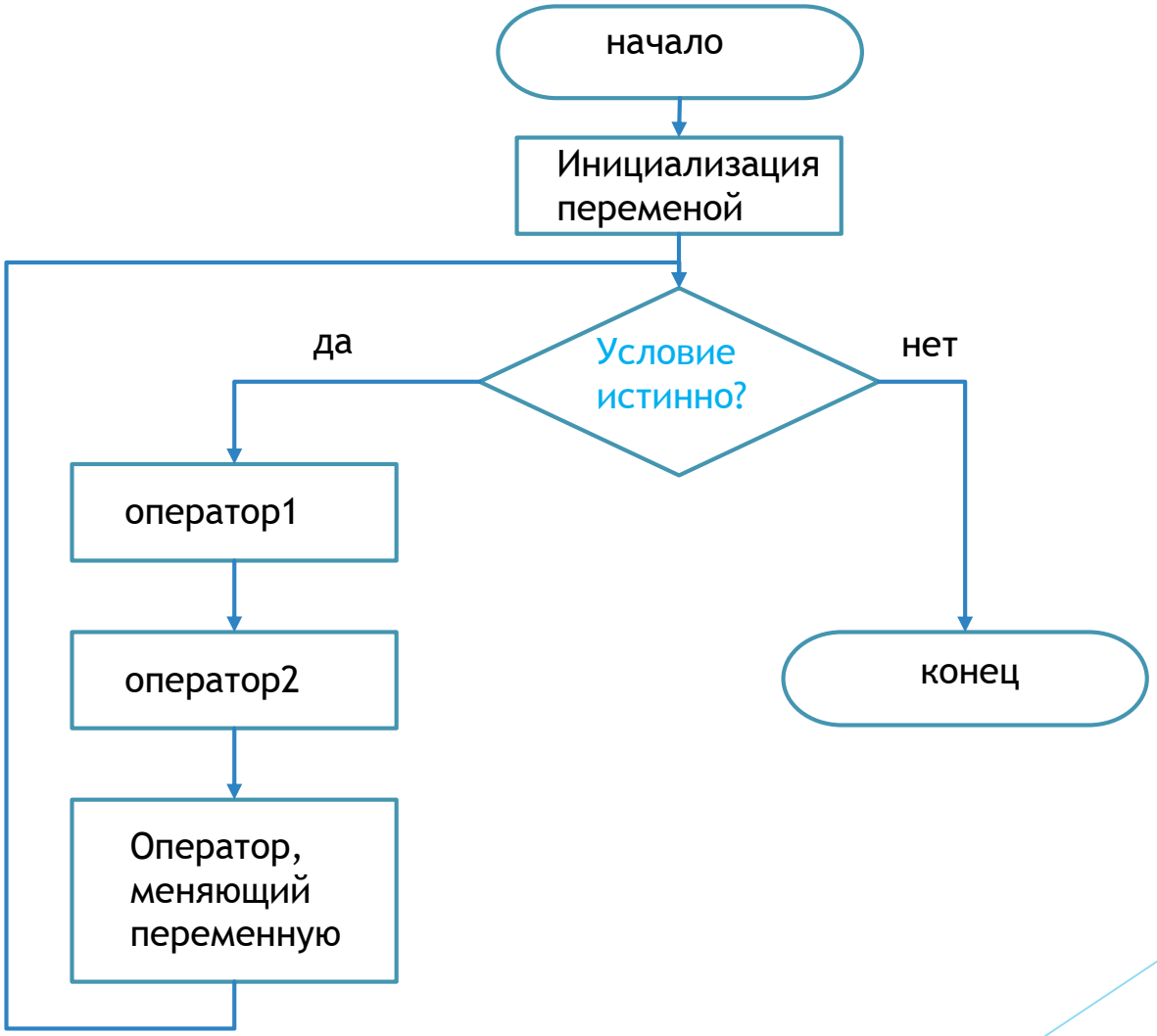
Бесконечный цикл `while`

```
while(true)
{
    std::cout << something << std::endl;
    …
}
```

Оператор цикла `for`

- Алгоритм цикла `for`:
  1. Инициализируется переменная цикла `for` – переменная, от значения которой зависит истинность условия цикла `for`
  2. Затем проверяется условие цикла `for`
  3. Если условие истинное, выполняется тело цикла `for`. Иначе – переход к пункту 6.
  4. Изменяется значение переменной цикла
  5. Переход к пункту 2.
  6. Если условие ложное – выход из цикла
- Цикл `for` может не выполниться ни разу, если при первой проверке условие будет ложным
- Цикл `for` обычно используется, когда заранее известно число повторов

Оператор цикла `for`



Оператор цикла `for`

1. Инициализация переменной
2. Проверка условия цикла
3. Изменение переменной цикла
4. Выполнение тела цикла

```
for(int i = 0; i<=16; i++)
{
    std::cout << i << "\t" << step << std::endl;
    step*=2;
}
```

Порядок выполнения шагов:

```
1 => 2 => 4 => 3 => … => 4 => 3 => 2 => выход
```

Бесконечный цикл `for`

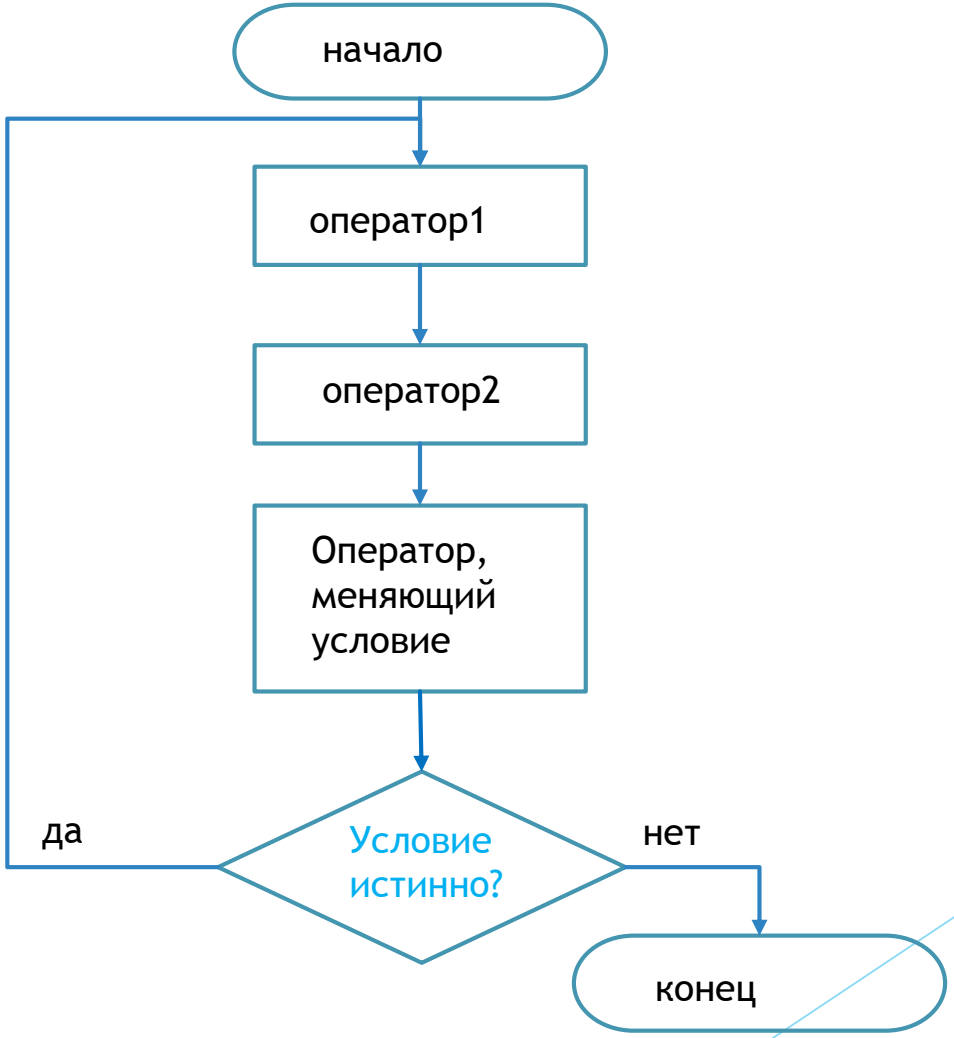
```
for(;;)
{
    std::cout << something << std::endl;
    …
}
```

Если необходим бесконечный цикл, то лучше использовать цикл **`while`**

Оператор цикла `do / while`

- Алгоритм цикла **`do / while`**:
  1. Выполняется тело цикла **`do / while.`**
  2. Проверяется условие цикла **`do / while`**
  3. Если условие истинное - возврат к первому пункту
  4. Если условие ложное – выход из цикла
- Цикл **`do / while`** выполнится минимум один раз, даже если при первой проверке условие будет ложным
- Тело цикла **`do / while`** должно содержать инструкцию, изменяющую истинность условия цикла
- Цикл **`do / while`** предпочтительнее не использовать

Оператор цикла `do / while`



Определить количество цифр в целом числе

```
int number =0, step = 0, del=1;
std::cout << "Enter number: ";
std::cin >> number;
do
{
    del*=10;
    step++;
} while(number/del);
std::cout << "Digits quantity: " << step << std::endl;
```

Результат (несколько запусков):

```
Enter number: -145
Digits quantity: 3
```

```
Enter number: 32000765
Digits quantity: 8
```

Вложенные операторы

- Циклы можно вкладывать:
  - Друг в друга
  - В оператор условия
  - В оператор множественного выбора
- Оператор условия можно вкладывать:
  - В другой оператор условия
  - В оператор множественного выбора
  - В цикл
- Оператор множественного выбора можно вкладывать:
  - В другой оператор множественного выбора
  - В оператор условия
  - В оператор цикла
- Глубина вложенности определяется решаемой задачей

Распечатать таблицу умножения

```
for (int i = 1; i <= 10; i++)
{
    for (int j = 1; j <= 10; j++)
    {
        std::cout << i * j << "\t";
    }
    std::cout << std::endl;
}
```

Переменные цикла `j, k, l` используются только во внутренних циклах после использования переменной `i`

Распечатать таблицу умножения - результат

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Какой цикл выбрать?

- Если известно число повторений - отдайте предпочтение циклу **`for`**
- Если число повторений неизвестно - выбирайте цикл **`while`**
- Если требуется организовать бесконечный цикл - выбирайте цикл **`while`**
- Цикл **`for`** можно записать через цикл **`while`** и наоборот (операторы взаимозаменяемы)
- По возможности избегайте использования цикла **`do / while`**

Операторы управления

- **return** – возврат управления
- **break** – немедленный выход из цикла или switch
- **continue** – переход на следующую итерацию цикла
- **goto** – безусловный переход на метку

Оператор `return`

- Возвращает управление из функции в точку вызова
- Если вызывается из функции **`main`** – завершает программу
- Существует в двух формах:
  - возвращает управление
  - возвращает управление и значение
- Если оператор **`return`** возвращает и значение, то тип возвращаемого значения должен совпадать с типом функции
- Если функция не имеет типа возвращаемого значения, то оператор **`return`** можно не писать

Оператор `break`

- Изменяет поток управления
- Вызывается из операторов циклов и оператора множественного выбора `switch`
- При вызове передает управление на следующий после цикла или `switch` оператор
- По возможности следует избегать использования этого оператора в циклах

Оператор `continue`

- Изменяет поток управления
- Вызывается из операторов циклов
- При вызове передает управление на следующую итерацию цикла
- По возможности следует избегать использования этого оператора

Различия в циклах `while` и `for`

```
for (int i = 0; i < 10; i++)
{
    if (5 == i)
        continue;
    std::cout << i << std::endl;
}
```

Результат:

```
0
1
2
3
4
6
7
8
9
_
```

Различия в циклах `while` и `for`

```
int i = 0;
while (i < 10)
{
     if (5 == i)
          continue;
     std::cout << i++ << std::endl;
}
```

Результат:

```
0
1
2
3
4
_
```

Различия в циклах `while` и `for` - исправления

```
int i = 0;
while (i < 10) {
    if (5 == i)  {
        ++i;
        continue;
    }
    std::cout << i++ << std::endl;
}
```

Результат:

```
0
1
2
3
4
6
7
8
9
_
```

Оператор `goto`

- Изменяет поток управления
- Вызывается в любом месте программы
- Передает управление в любое место функции (выше, ниже) на указанную метку
- Использование приводит к спагетти-коду
- Обоснованно использовать при выходе из глубоко вложенных циклов на следующий после всех циклов оператор

В УЧЕБНЫХ ПРОГРАММАХ ОПЕРАТОРА `goto` БЫТЬ НЕ ДОЛЖНО

Конец