# Операторы и выражения в C

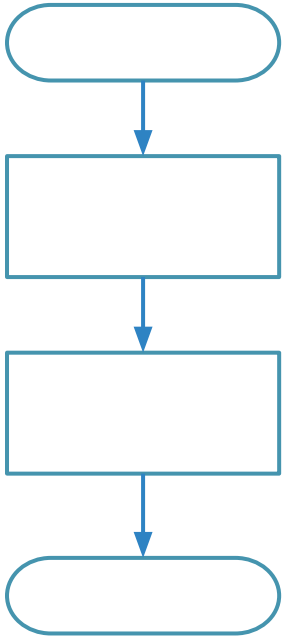
Лекция 3

# Оператор — действие в программе

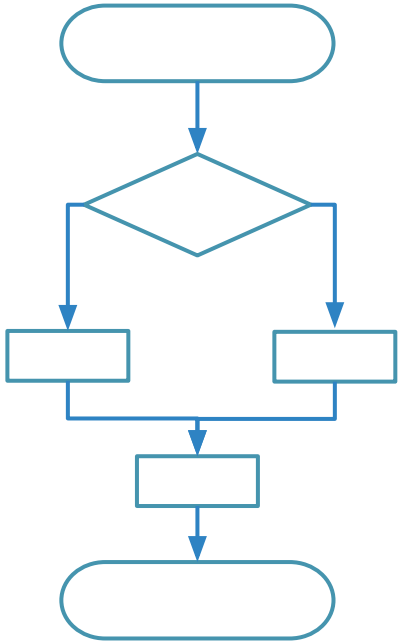
- Оператор заканчивается ;
- Пустой оператор — ;
- Оператор-выражение — оператор, имеющий значение
- Оператор управления — оператор, управляющий выполнением программы
- Оператор условия — позволяют выполнять разные действия при выполнении разных условий
- Оператор множественного выбора — похож на оператор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

# Последовательность действий в программе

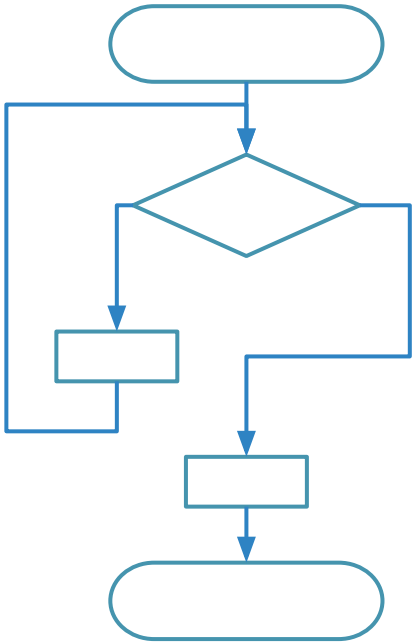
Линейная



Ветвление



Повторы



# Оператор условия `if / else`

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

```c
if (x > y)
    printf("%d\n", 1);
```

```c
if ( x > y)
   printf("%d\n", 1);
else
   printf("%d\n", 0);
```

Может быть заменен условной операцией:

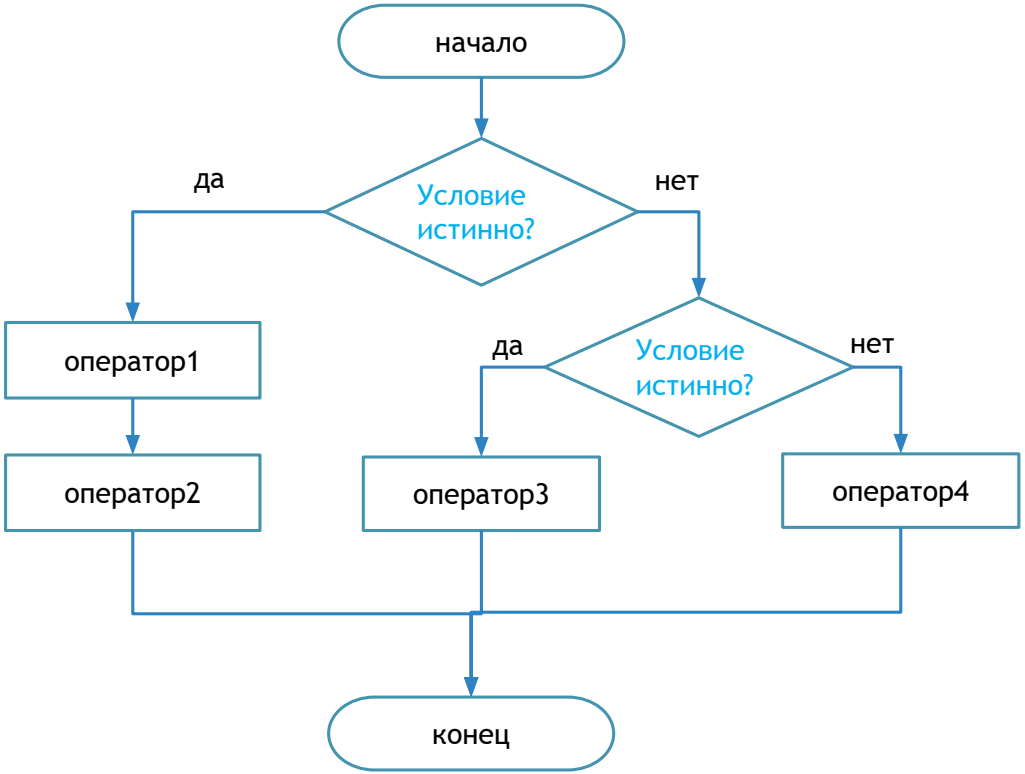
```c
printf("%d\n", (x > y ? 1 : 0 ));
```

# Оператор условия `if / else`

Можно проверять несколько условий

```
if (x > y)
    printf("%d\n", 1);
else if (x > z)
    printf("%d\n", 2);
else if (y > z)
    printf("%d\n", 3);
else
    printf("%d\n", 4);
```

# 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) {
    printf("%d\n", 1);
    ++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)
        printf("%d\n", x);
    else if (y > z)
        printf("%d\n", y);
    else
        printf("%d\n", z);
}
```

Ничего не будет напечатано, так как x не больше y

# 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)
    printf("%d\n", x);
}
else if (y > z) {
    printf("%d\n", y);
}
else {
    printf("%d\n", z);
}
```

# 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

```
int isOk = readFile (fileName);
if (isOk) {
    puts("OK");
}
else {
    puts("Error");
}
```

# Оператор if /else

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

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

if (isFinished || isRepeatedEntry) {
     puts("Done");
}
```

```
if ((elementNo < 0) || (elementNo > maxElement) || (elementNo == lastElement)) {
     puts("Done");
}
```

# Оператор `if /else`

## Правильно подбирайте условия

```
if (grade > 40) {
    puts("You got 5!");
}
else if (grade > 30) {
    puts("You got 4!");
}
else if (grade > 20) {
    puts("You got 3!");
}
```

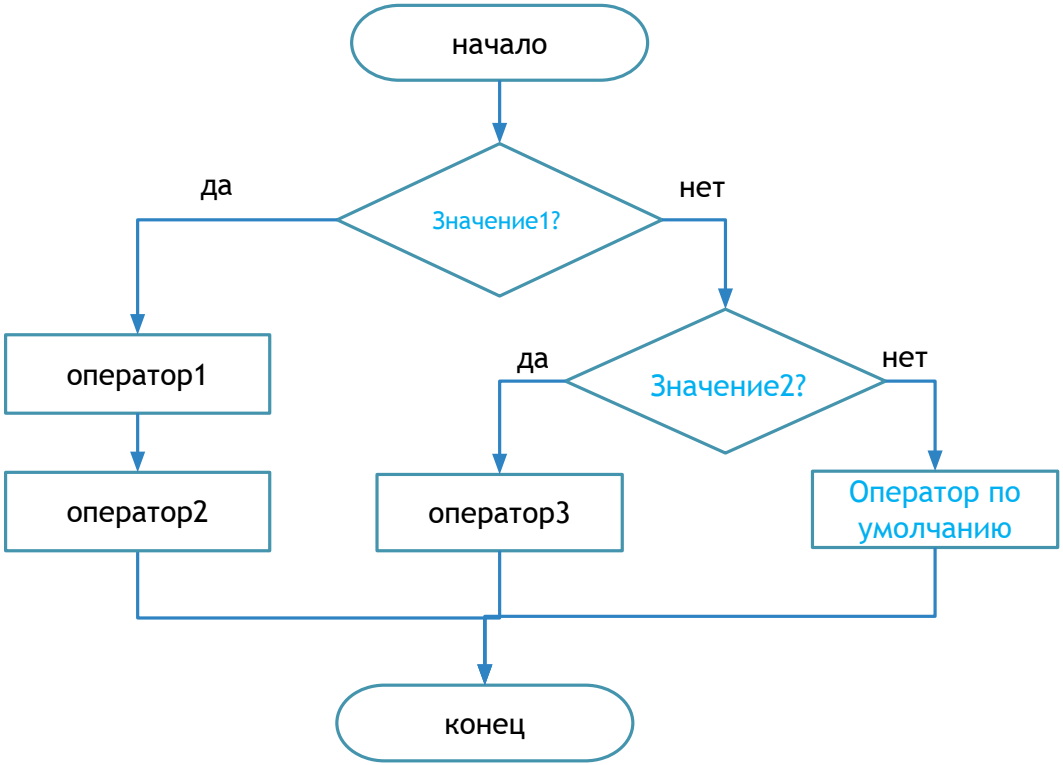
```
if (grade > 40) {
    puts("You got 5!");
}
if (grade > 30 && grade <= 40) {
    puts("You got a 4!");
}
if (grade > 20 && grade <= 30) {
    puts("You got a 3!");
}
```

# Оператор множественного выбора switch

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

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

# Оператор множественного выбора `switch`



# Напечатать название месяца по его номеру

```
int monthNo=0;
puts("Enter month number: ");
scanf("%d",  &monthNo);
switch (monthNo)
{
case 1:
    puts("January");
case 2:
    puts("February");
…
case 12:
    puts("December");
default:
    puts("Wrong month number");
}
```

```
Enter month number:
9
September
October
November
December
Wrong month number
```

# Напечатать название месяца по его номеру

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

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

# Напечатать название сезона по номеру месяца

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

```
Enter month number:
9
Fall
```

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

# Напечатать название сезона по номеру месяца

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

```
Enter month number:
9
Fall
```

Объединение нескольких констант

# Операторы цикла

Циклы позволяют повторять
один и тот же кусок кода
любое количество раз

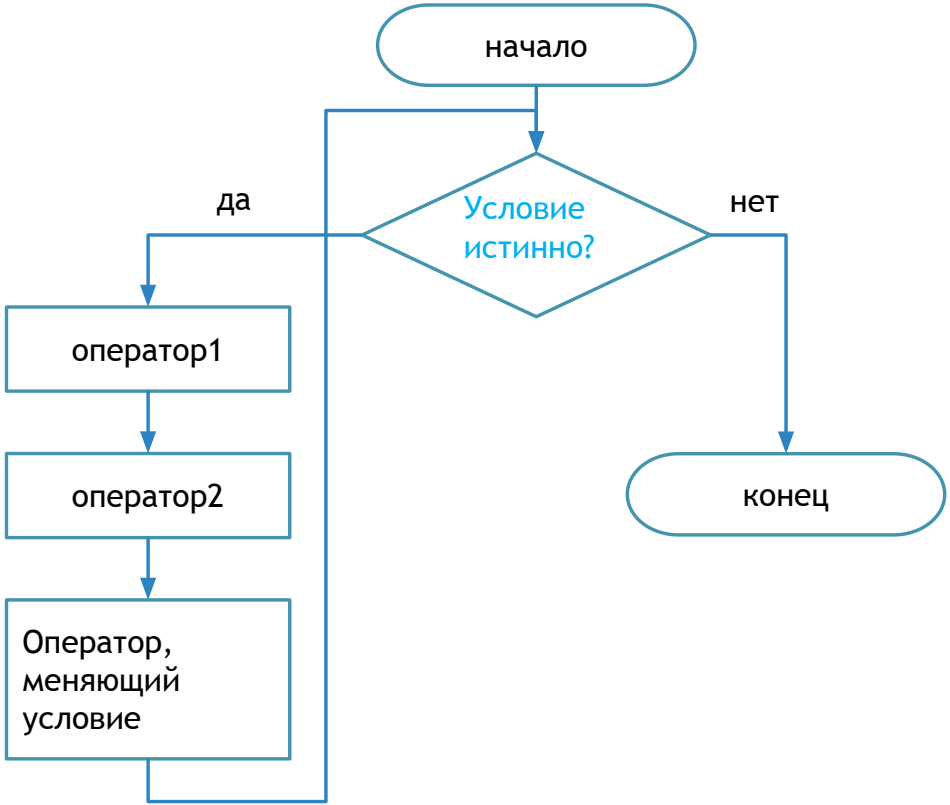
Виды циклов:

- С предусловием
  - `for`
  - `while`
- С постусловием
  - `do/while`

# Оператор цикла `while`

- Алгоритм цикла `while`:
  - Сначала проверяется условие цикла `while`
  - Если условие истинное, выполняется тело цикла `while`. Возврат к первому пункту
  - 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;
puts("Enter the grade or -1 for stopping: ");
scanf("%d", &grade);
while (grade != -1) {
     total += grade;
     ++counter;
     puts("Enter the grade or -1 for stopping: ");
   scanf("%d", &grade);
}
average = (float)total / counter;
printf("The average grade is %.2f\n", average);
```

```
Enter the grade or -1 for stopping: 2
Enter the grade or -1 for stopping: 3
Enter the grade or -1 for stopping: 4
Enter the grade or -1 for stopping: 4
Enter the grade or -1 for stopping: 5
Enter the grade or -1 for stopping: 2
Enter the grade or -1 for stopping: 3
Enter the grade or -1 for stopping: -1
The average grade is 3.29
```

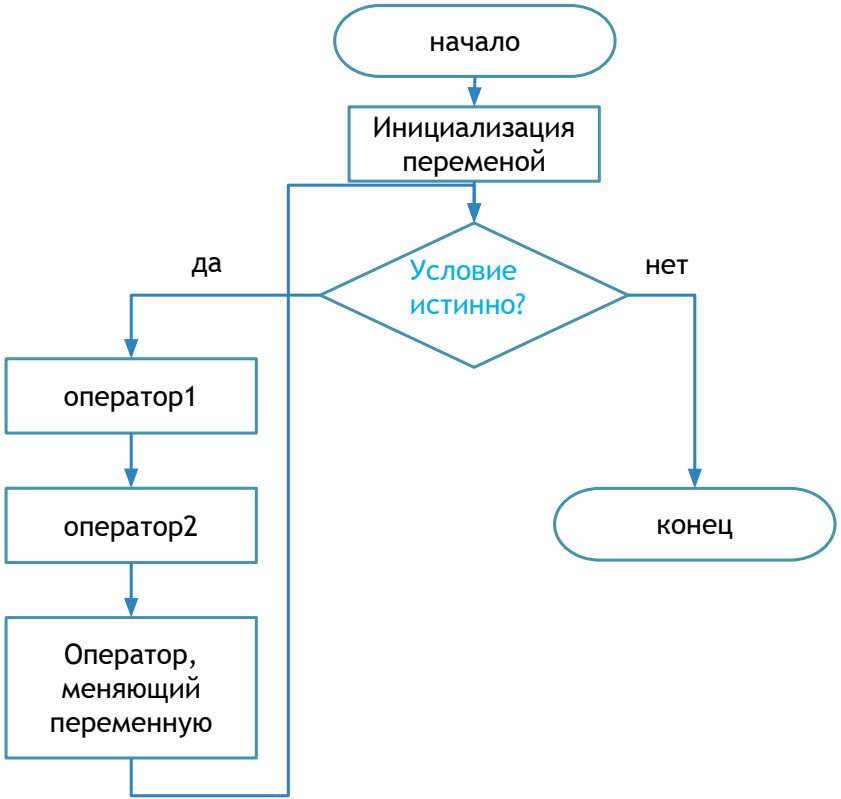
# Бесконечный цикл `while`

```
while(1) {
    puts("Doing something");
    …
}
```

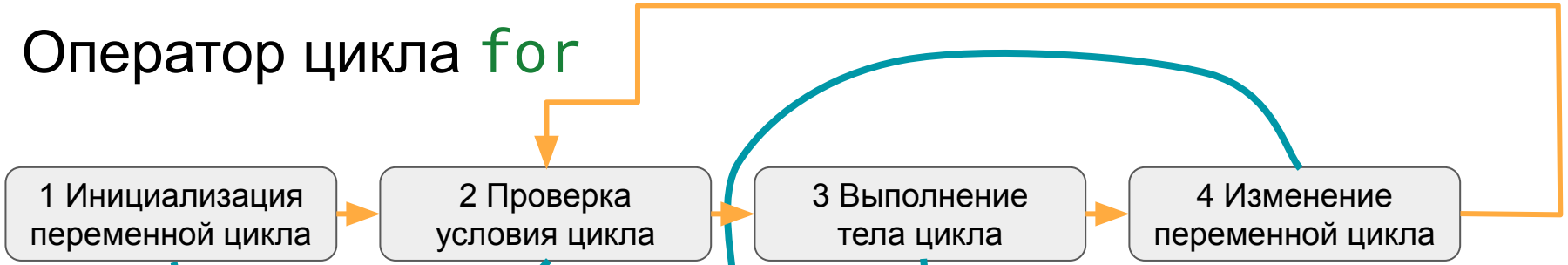
# Оператор цикла `for`

- Алгоритм цикла `for`:
  - Инициализируется переменная цикла `for`, от значения которой зависит истинность условия цикла `for`
  - Затем проверяется условие цикла `for`
  - Если условие истинное, выполняется тело цикла `for`. Иначе происходит переход к пункту 6.
  - Изменяется значение переменной цикла
  - Переход к пункту 2.
  - Если условие ложное, происходит 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




```
for(int i = 0; i<=16; i++){
    printf("%d\t%d\n", i, step);
    step*=2;
}
```

# Бесконечный цикл `for`

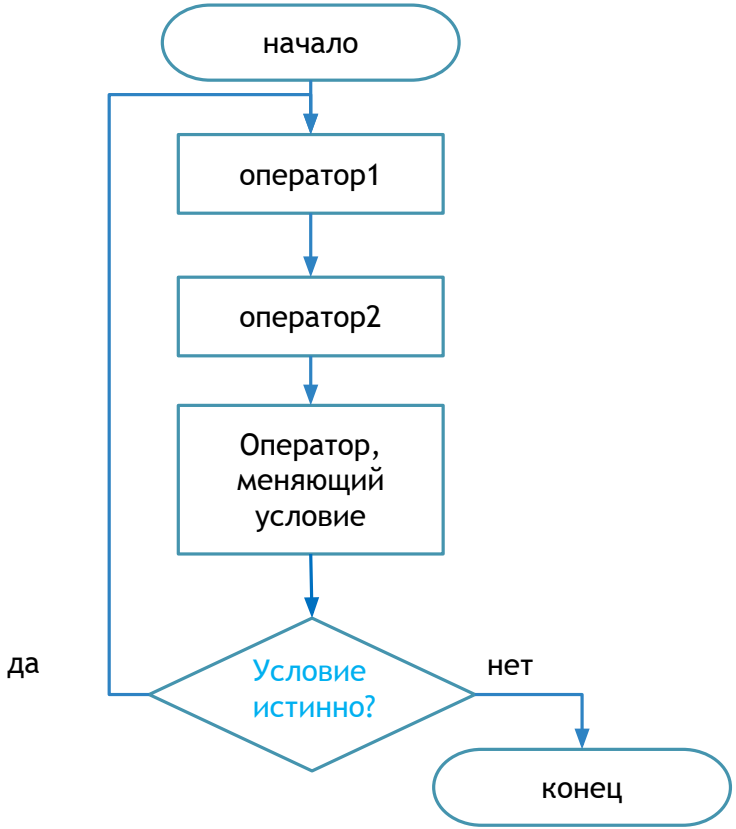
```
for(;;){
    puts("Doing something");
    …
}
```

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

# Оператор цикла `do/while`

- Алгоритм цикла `do/while`:
  - Выполняется тело цикла `do/while`.
  - Проверяется условие цикла `do/while`
  - Если условие истинное — возврат к первому пункту
  - 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;
printf("Enter a number: ");
scanf("%d", &number);
do {
     del*=10;
     step++;
} while(number/del);
printf("Digits quantity: %d\n", step);
```

```
Enter a number: 4
Digits quantity: 1
```

```
Enter a number: 2355234
Digits quantity: 7
```

```
Enter a number: -34234
Digits quantity: 5
```

# Вложенные операторы

- Все операторы можно вкладывать друг в друга
- Глубина вложенности определяется решаемой задачей

# Распечатать таблицу умножения

```
for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
        printf("%d\t", i * j);
    }
    printf("\n");
}
```

```
/Users/danilashubin/CLionProjects/untitl
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, то отдайте предпочтение циклу `for`
- Если число повторений неизвестно, то выбирайте цикл `while`
- Если требуется организовать бесконечный цикл, то 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`

```c
for (int i = 0; i < 10; i++) {
    if (5 == i)
        continue;
    printf("%d\n", i);
}
```

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

```c
int i = 0;
while (i < 10) {
    if (5 == i)
        continue;
    printf("%d\n", i++);
}
```

```
0
1
2
3
4
```

# Различия в циклах `while` и `for`

```c
int i = 0;
while (i < 10) {
    if (5 == i)  {
        ++i;
        continue;
    }
    printf("%d\n", i++);
}
```

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

```c
int i = 0;
while (i < 10) {
    if (5 == i)
        continue;
    printf("%d\n", i++);
}
```

```
0
1
2
3
4
```

# 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**