УНИВЕРСИТЕТ СИНЕРГИЯ

РАЗРАБОТКА НА C++

Урок 4.
Цикл “for”

Синтаксис цикла «for»

Цикл for имеет следующий общий синтаксис:

```
for (инициализация; условие; выражение обновления) {
    // Тело цикла
}
```

Оператор "инициализация" выполняется **один раз** в начале цикла и используется для инициализации переменных, которые будут участвовать в цикле.

Условие задает условие продолжения выполнения цикла. Пока условие истинно, цикл будет выполняться.

"Выражение обновления" выполняется после каждой итерации цикла и обычно используется для изменения переменных контроля цикла.

Параметры цикла «for»: инициализация

Инициализация - это объявление переменной, которая будет использоваться в цикле. По-другому эту переменную называют “счетчик цикла”.

Например:

```
for (int i = 0; i < 5; i++) {
    // Тело цикла
}
```

```
//в качестве счетчика можно использовать уже существующую переменную (также можно поменять её значение)

int i = 5;

for (i=3; i < 5; i++) {
    // Тело цикла
}
```

Параметры цикла «for»: условие

Условие определяет, когда цикл будет выполняться и когда он завершится. Цикл будет выполняться, пока это условие истинно.

Например:

```
for (int i = 0; i < 5; i++) {
        // Тело цикла будет выполняться пять раз (i = 0, 1, 2, 3, 4)
}
```

//Условия могут быть такими же сложными как в конструкции if, но Важно! Цикл будет выполнятся пока всё условие будет true и завершится, когда false

```
for(int i = 0; i<5 && i>-2; i++){} \\Тело цикла будет выполняться пять раз (i = 0, 1, 2, 3, 4)

for(int i = 17; i<5 && i>-2; i++){} \\Тело цикла не будет выполнятся

for(int i = -1; i<5 && i>-2; i++){} \\Тело цикла будет выполняться шесть раз (i = -1, 0, 1, 2, 3, 4)
```

Параметры цикла «for»: выражение обновления

Это выражение, которое выполняется после каждой итерации цикла. Обычно оно используется для изменения переменных цикла.

Например:

```
for (int i = 0; i < 5; i++) {
    // После каждой итерации i увеличивается на 1
   // 0 1 2 3 4
}

for (int i = 0; i < 5; i+=2) {
    // После каждой итерации i увеличивается на 2
    // 0 2 4
}

for (int i = 0; i < 5; i--) {
    // После каждой итерации i уменьшается на 1 (Осторожно! это приведет к бесконечному циклу)
    // 0 -1 -2 -3 -4… и так до бесконечности (ведь цикл продолжается пока i < 5)
}
```

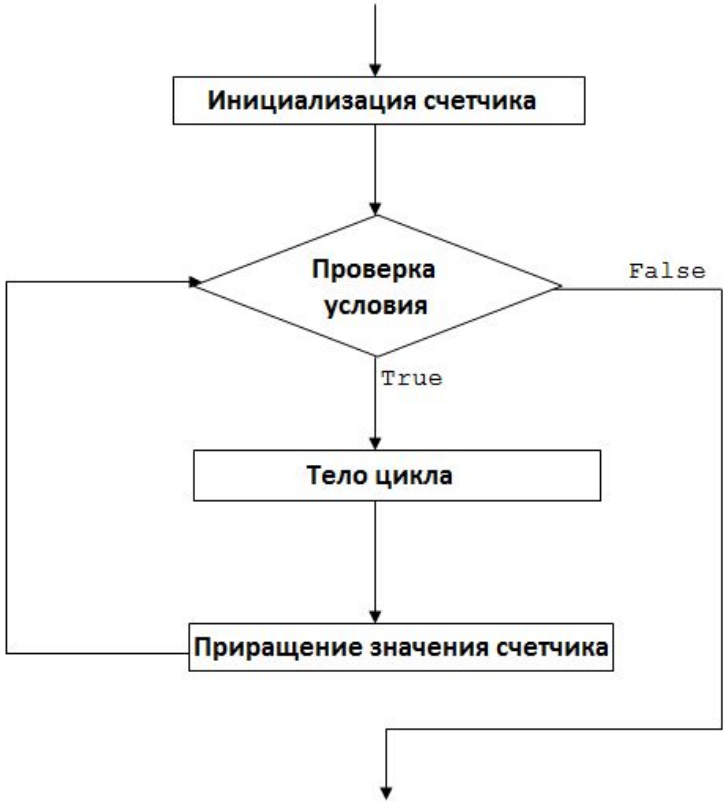
Порядок цикла «for»

1. Инициализируем счетчик, например int i = 0; (Только один раз!)
2. Проверяем условие, например i < 5;
3. Если условие - правда
   a. Выполняем код внутри фигурных скобок { }
   b. Выполняем “выражение обновления”, например i++
   c. Переход к шагу 2
4. Если условие - ложь
   a. Выход из цикла

```
for(int i = 0; i<5; i++){
        cout << i << “ “;
}

// Вывод: 0 1 2 3 4
```

```
i = 0;   0<5 правда   вывод: 0   i = 0+1
i = 1;   1<5 правда   вывод: 1   i = 1+1
i = 2;   2<5 правда   вывод: 2   i = 2+1
i = 3;   3<5 правда   вывод: 3   i = 3+1
i = 4;   4<5 правда   вывод: 4   i = 4+1
i = 5;   5<5 ложь     выход из цикла
```



Вложенные циклы

Вложенные циклы - это конструкция, которая позволяет использовать один цикл внутри другого цикла. Их используют для обработки многократных операций над многомерными структурами данных и матрицами. В этом разделе мы рассмотрим основы вложенных циклов в C++.

Вложенные циклы имеют следующий синтаксис:

```
for (инициализация1; условие1; выражение обновления1) {
    for (инициализация2; условие2; выражение обновления2) {
        // Тело внутреннего цикла
    }
    // Тело внешнего цикла
}
```

Внешний цикл управляет выполнением внутреннего цикла. Внутренний цикл полностью выполняется для каждой итерации внешнего цикла.

Вложенные циклы

С помощью вложенных циклов удобно, например, разработать небольшую программу для вывода таблицы умножения:

```
#include <iostream>

int main() {
    int n = 10; // Размер таблицы умножения (можно изменить по желанию)

    // Внешний цикл для умножаемых чисел (от 1 до n)
    for (int i = 1; i <= n; i++) {
        // Внутренний цикл для множителей (от 1 до n)
        for (int j = 1; j <= n; j++) {
            // Выводим результат умножения i на j
            std::cout << i << " * " << j << " = " << (i * j) << "\t";
        }
        // Переход на следующую строку после завершения строки таблицы
        std::cout << std::endl;
    }

    return 0;
}
```

Операторы управления циклом

break

Инструкция «break» используется для немедленного завершения выполнения цикла, в котором она находится. Когда команда «break» выполнена, управление передается следующей инструкции за циклом.

Не используйте этот оператор, если без него можно обойтись!

```
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break; // Завершить цикл, когда i равно 5
    }
    std::cout << i << " ";
}
// Вывод: 0 1 2 3 4
```

```
for (int i = 0; i < 5; i++) {
    std::cout << i << " ";
}
// Вывод: 0 1 2 3 4
```

Операторы управления циклом

continue

Инструкция «continue» используется для пропуска текущей итерации цикла и перехода к следующей итерации. Когда «continue» выполнена, управление возвращается в начало цикла, и следующая итерация начинается сразу. Этот оператор полезен, когда необходимо пропустить часть кода внутри цикла для определенных условий.

```
for (int i = 0; i < 5; i++) {
    if (i == 2) {
        continue; // Пропустить итерацию при i равном 2
    }
    std::cout << i << " ";
}
// Вывод: 0 1 3 4
```

Практические советы по использованию циклов

1. Избегайте бесконечных циклов
   Бесконечные циклы — это циклы, которые никогда не завершаются по условию. Они могут привести к зависанию программы. Убедитесь, что условие цикла в какой-то момент станет ложным, чтобы избежать бесконечного выполнения.

   Пример бесконечного цикла:
   ```
   for (int i = 0; i >= 0; i++) {
       std::cout << i << " ";
   }
   ```

2. Правильный выбор между циклами for, while и do-while
   - «for» обычно используется, когда заранее известно количество итераций.
   - «while» подходит, когда условие для выполнения цикла известно заранее, но количество итераций может варьироваться.
   - «do-while» полезен, когда вы хотите, чтобы цикл выполнился хотя бы один раз, даже если условие не выполнено с самого начала.

Практические советы по использованию циклов

3. Использование булевых флагов

Иногда циклами удобно управлять с помощью булевых флагов (переменных-условий).

Пример использования булевого флага:

```
bool condition_met = false;
while (!condition_met) {
    // ...
    if (some_condition) {
        condition_met = true; // Выход из цикла при выполнении условия
    }
    // ...
}
```

Тернарный оператор

```
if(выражение_условия)
{
    cout << значение_true << endl;
}
else
{
    cout << значение_false << endl;
}
```

можно заменить на:

```
cout << (выражение_условия ? значение_true : значение_false) << endl;
```

```
int a = 6;

// тернарный оператор
a = (a > 10 ? a*10 : a-100); //-94

// конструкция if else
if (a > 10) { //false, значит
    a = a * 10; //пропускаем
}                   //пропускаем
else {
    a = a - 100; //-94
}
```