Учреждение образования

«Белорусский государственный технологический университет»

**Реферат**

по дисциплине «Основы алгоритмизации и программирования»

по теме «Операторы цикла»

Работу выполнил:

Студент ИСиТ 1-1

**Палазник А.В.**

Минск, 2021

**Задача**: познакомиться с операторами цикла, рассказать о каждом из них, также узнать об операторах break и continue, и в заключение сделать определенные выводы

**Алгоритм решения:**

1)Определение цикла

2)Цикл while

3)Цикл for

4)Цикл do while

5)Где какой лучше использовать

6)Вложенные циклы

7)Операторы break и continue

8)Вывод

**Определение цикла**

Цикл – это участок кода, который выполняется такое количество раз, пока условие является истинным. Таким образом, чтобы не писать какое-то выражение огромное количество раз, мы можем поместить его в цикл с определенным количеством повторений.

**Цикл while**

Цикл while, как и цикл for, является предусловным. Это значит, что сначала проверяется условие на истинность, а уже потом выполняется тело цикла.

Синтаксис:

while (условие) {

выражения

}

В теле цикла обязательно нужно добавить любой из операторов изменения переменной, иначе у нас выйдет бесконечный цикл. Также, перед самим циклом эту переменную необходимо объявить и инициализировать.

Пример:

#include <iostream>

using namespace std;

void main() {

int m;

double h, y, r, c, j;

m = 7, r = 4e-4, c = 2.1, j = 0;

while (j <= 1.7) {

h = (10 \* r - j) / (pow(c, 2) + exp(-m));

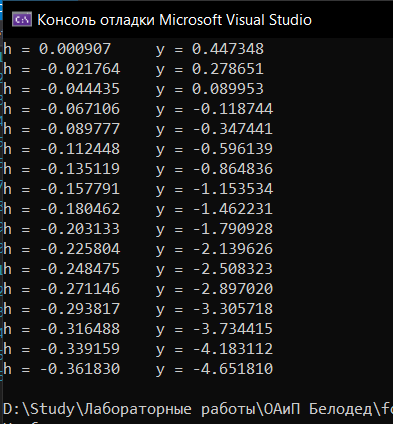
y = (h \* m - pow(j, 2)) + (0.1 \* pow(c, 2));

printf("h = %f\t y = %f\n", h, y);

j = j + 0.1;

}

}



Псевдокод для этого примера:

ОБЪЯВЛЕНИЕ m

ОБЪЯВЛЕНИЕ h, y, r, c, j

присвоение значений переменным

НАЧАЛО ЦИКЛА

ДЛЯ j от 0 до 1.7 с шагом 0.1

ПОВТОРЯТЬ

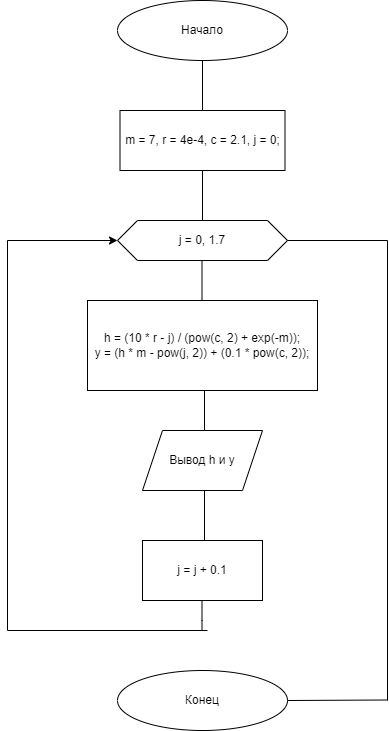
вычисляем неинициализированные переменные

выводим в консоль текст со значениями переменных

меняем значение переменной из условия

КОНЕЦ ЦИКЛА

Блок-схема:



**Цикл for**

Синтаксис:

for (инициализация; условие; изменение переменной) {

выражения;

}

Параметры цикла можно опускать. Таким образом цикл for можно преобразовать в цикл while

int i = 0;

for(; i < numb; ) {

выражения;

i++;

}

Также, можно с помощью оператора «запятая» добавить дополнительные параметры. Ошибки не будет, т.к. компилятор будет считать их за одно целое выражение

for (int i = 0, j = 0; i < numb1, j < numb2; i++, j--) {

выражения;

}

Пример:

#include <iostream>

#define SIZE 7

using namespace std;

void main() {

int n, i;

double a, b, y, z;

n = 2, a = 2e-3, b = 8.5;

for (i = 1; i < SIZE; i++) {

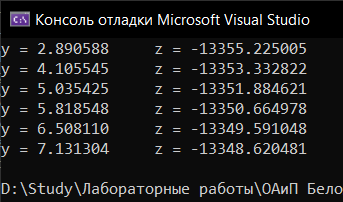
y = sqrt(i \* b - (pow(b, 2) \* a));

z = y \* tan(n / 2) - exp(b + 1);

printf("y = %f\t z = %f\n", y, z);

}

}



Псевдокод:

ОБЪЯВЛЕНИЕ n и i

ОБЪЯВЛЕНИЕ a, b, y, z

присваивание значений переменным

НАЧАЛО ЦИКЛА

ДЛЯ i от 1 до SIZE с увеличением

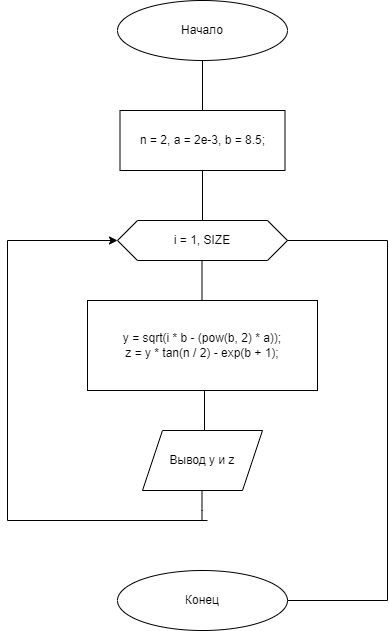
ПОВТОРЯТЬ

вычисляем неинициализированные переменные

выводим в консоль текст со значениями переменных

КОНЕЦ ЦИКЛА

Блок-схема:



**Цикл do while**

Цикл do while является постусловным, т.е. он сначала выполняет операции в теле цикла, а после проверяет истинность условия. Таким образом, тело цикла сработает как минимум один раз независимо от условия

Синтаксис:

do {

выражения;

}

while (проверка условия);

Пример:

#include <iostream>

using namespace std;

void main() {

int m;

double h, y, r, c, j;

m = 7, r = 4e-4, c = 2.1, j = 0;

do {

h = (10 \* r - j) / (pow(c, 2) + exp(-m));

y = (h \* m - pow(j, 2)) + (0.1 \* pow(c, 2));

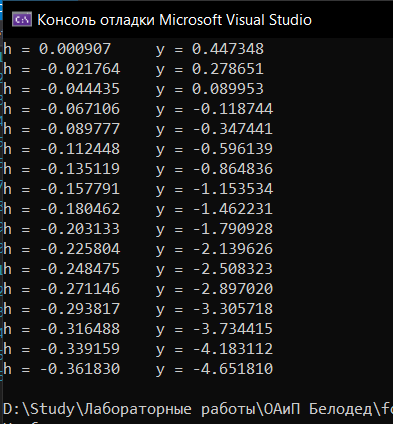
printf("h = %f\t y = %f\n", h, y);

j = j + 0.1;

}

while (j <= 1.7);

}



Псевдокод:

ОБЪЯВЛЕНИЕ m

ОБЪЯВЛЕНИЕ h, y, r, c, j

присвоение значений переменным

НАЧАЛО ЦИКЛА

ПОВТОРЯТЬ

вычисляем неинициализированные переменные

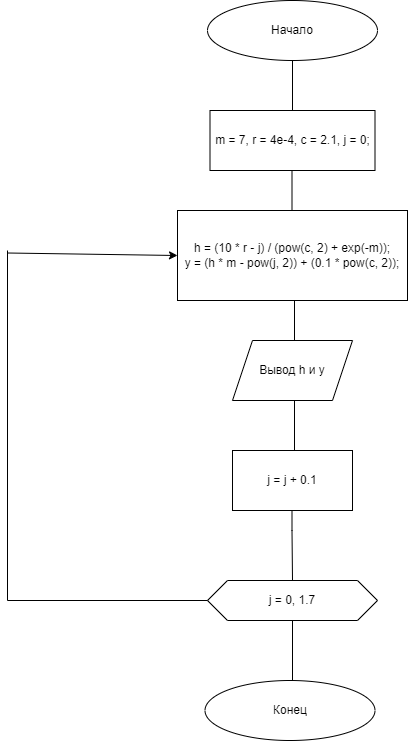
выводим в консоль текст со значениями переменных

меняем значение переменной из условия

КОНЕЦ ЦИКЛА

повторять пока истинно

Блок-схема:



**Где и какой цикл лучше использовать**

Цикл do while обычно используют не часто, из-за его постусловности, которое редко где применимо. Основное различие циклов for и while заключается в том, что цикл for обычно используют когда мы точно знаем, сколько должно произойти итераций цикла, цикл while – когда не знаем.

Никто не выяснил, какой же из циклов все же лучше, однако мне больше нравится цикл for, т.к. с помощью запятой можно увеличить гибкость цикла. К тому же, при необходимости его можно преобразовать в цикл while, опустив выражения в круглых скобках

**Вложенные циклы**

Циклы можно вкладывать друг в друга. Таким образом можно расширять разнообразие результатов

Пример:

#include <iostream>

#define iMAX 4

#define jMAX 6

using namespace std;

void main() {

setlocale(LC\_ALL, "ru");

int j = 1;

for (int i = 1; i < iMAX; i++) {

printf("\n%d. Виды", i);

while (j < jMAX) {

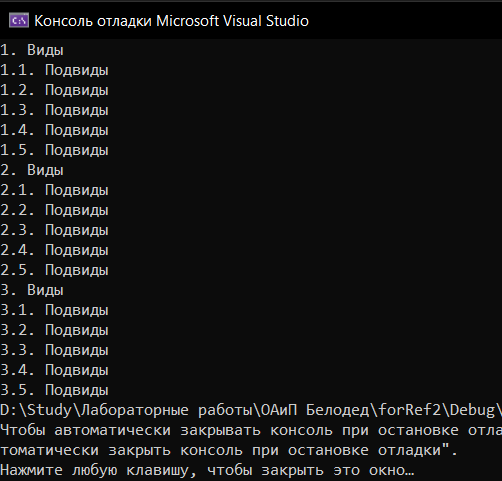
printf("\n%d.%d. Подвиды", i, j);

j++;

}

}

}



Псевдокод:

объявление и инициализация переменной

НАЧАЛО ПЕРВОГО ЦИКЛА

ДЛЯ i от 1 до iMAX

ПОВТОРЯТЬ

ВЫВОД ТЕКСТА

повторное присвоение значения переменной

НАЧАЛО ВТОРОГО ЦИКЛА

ДЛЯ j от 1 до jMAX

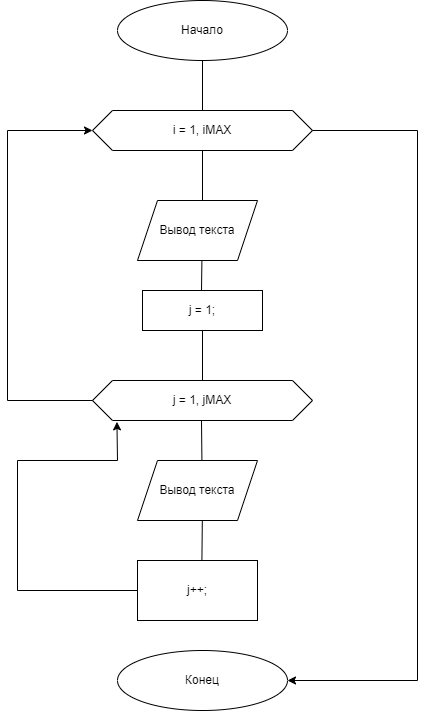
ПОВТОРЯТЬ

ВЫВОД ТЕКСТА

КОНЕЦ ВТОРОГО ЦИКЛА

КОНЕЦ ПЕРВОГО ЦИКЛА

Блок-схема:



**Операторы break и continue**

Существует метод, благодаря которому можно принудительно выйти из цикла. Таким чудом является оператор break. Также, существует оператор continue, который начинает новую итерацию цикла.

Пример:

#include <iostream>

#define iMAX 4

#define jMAX 6

using namespace std;

void main() {

setlocale(LC\_ALL, "ru");

int j = 1;

for (int i = 1; i < iMAX; i++) {

if (i == 2)

continue;

printf("\n%d. Виды", i);

j = 1;

while (j < jMAX) {

if (j == 4)

break;

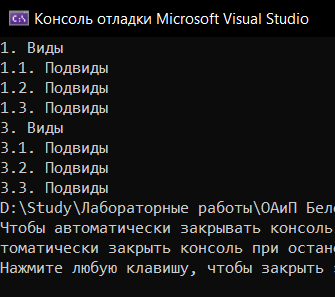
printf("\n%d.%d. Подвиды", i, j);

j++;

}

}

}



Псевдокод:

объявление и инициализация переменной

НАЧАЛО ПЕРВОГО ЦИКЛА

ДЛЯ i от 1 до iMAX

ПОВТОРЯТЬ

ЕСЛИ выражение истинно, ТО

новая итерация

КОНЕЦ ЕСЛИ

ВЫВОД СТРОКИ

повторное присвоение значения

НАЧАЛО ВТОРОГО ЦИКЛА

ДЛЯ j от 1 до jMAX

ПОВТОРЯТЬ

ЕСЛИ выражение истинно, ТО

ВЫХОД ИЗ ЦИКЛА

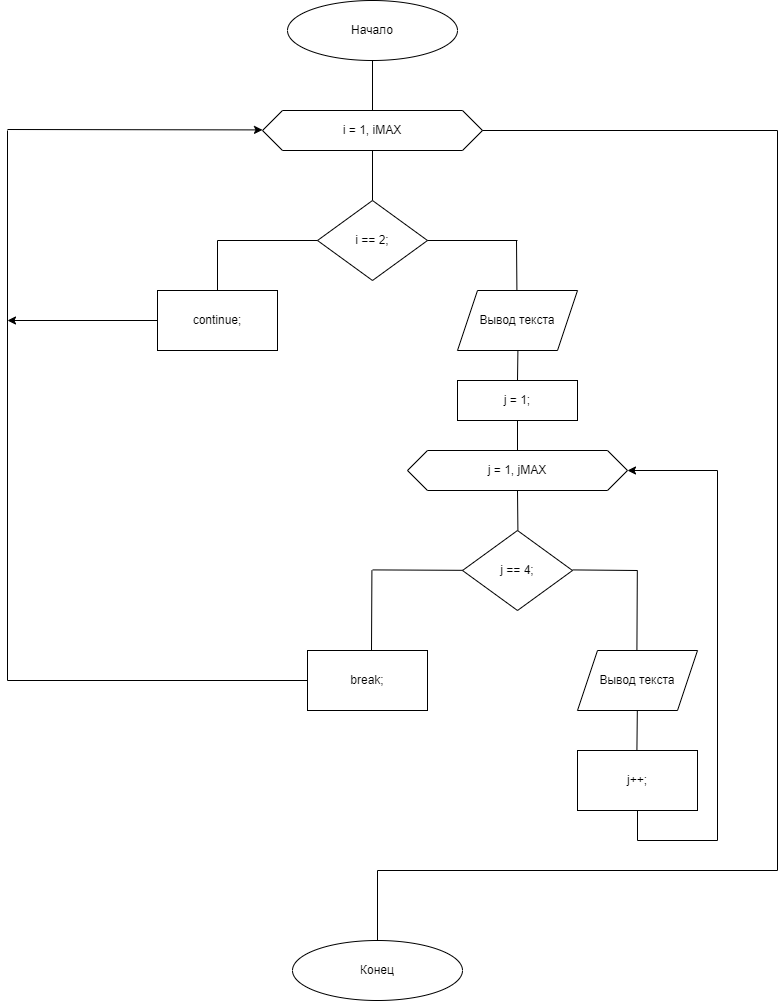
КОНЕЦ ЕСЛИ

ВЫВОД ТЕКСТА

КОНЕЦ ВТОРОГО ЦИКЛА

КОНЕЦ ПЕРВОГО ЦИКЛА

Блок-схема:



**Вывод:** циклы – это прекрасный инструмент, который заменит огромное количество повторений одних и тех же строк кода, на несколько команд, которые будут прогоняться определенное количество раз