Структурное программирование

ЛЕКЦИЯ №1

5 СЕНТЯБРЯ 2023

Знакомимся

- Борин Владислав Михайлович
- Лекции: вторник 09:00 312 гк.
- Почта для связи: V.M.Borin@inp.nsk.su
- Для особо экстренных вариантов: +7-903-933-33-66

Лабы/практика

- Лабы: Четверг 12:40 (1.5 пары)
- 3 подгруппы (терминальные классы 212, 305, 306)
- По программе 16 недель (в реальности может быть меньше).

Астафьева Алиса
Герлингер Евгений
Дрижак Иван
Иванова Александра
Малиновский Владислав
Машков Михаил
Паньшин Евгений
Труфанов Иван

Аудитория 212

Лабы/практика

- Лабы: Четверг 12:40 (1.5 пары)
- 3 подгруппы (терминальные классы 212, 305, 306)
- По программе 16 недель (в реальности может быть меньше).

Квинт Лукас
Купа Иван
Майстришин Арсений
Останин Федор
Погуляев Михаил
Русанов Илья
Шашкина Елизавета
Шешенин Георгий

Аудитория 305

Лабы/практика

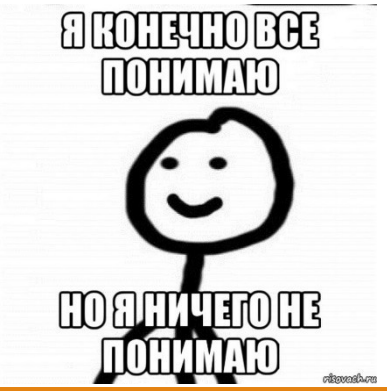
- Лабы: Четверг 12:40 (1.5 пары)
- 3 подгруппы (терминальные классы 212, 305, 306)
- По программе 16 недель (в реальности может быть меньше).

Антонов Максим
Бороздин Павел
Гемузов Артем
Климов Богдан
Позднякова Мария
Попов Дмитрий
Саликов Григорий
Хрисанов Константин

Аудитория 306

Новый семестр, новый язык программирования

- Быстрее
- Эффективнее
- Сложнее
- Опаснее



Язык С (Си)

1972-1974 - Dennis Ritchie (на данный момент стандарт регламентирован ISO/IEC 9899:2018)

- Язык для написания системы Unix.
- Язык высокого уровня с возможностью эффективного использования ресурсов.
- Высокая скорость работы программ
- Язык стал основой многих современных языков программирования
- Большая вероятность допустить ошибку, которая проявит себя во время работы (не обнаружится во время компиляции)

Краткое содержание Эвм

В простейшем случае содержит:

- центральный процессор (CPU),
- память,
- устройства ввода-вывода.

В простейшем случае содержит:

- устройство управления (CU),
- арифметико-логическое устройство (ALU),
- множество регистров.

Тип памяти	Время доступа	Объем
жесткий диск	~1 мс	~1 ТБ
оперативная память	~100 нс	~10 ГБ
регистр	~1 нс	~100 Б

Типы данных и их представление

Явное определение типа данных

ДИНАМИЧЕСКАЯ ТИПИЗАЦИЯ

Тип данных хранящихся в переменной определяется во время выполнения

```
def func(a, b):
    return a+b
```

СТАТИЧЕСКАЯ ТИПИЗАЦИЯ

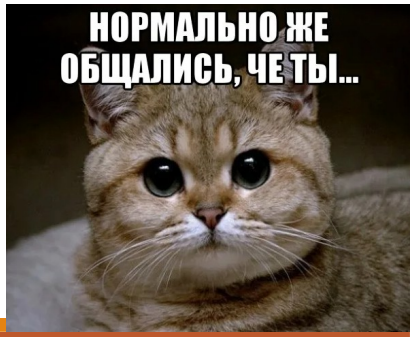
Тип данных хранящихся в переменной требует явного указания

```
int func(int a, int b)
{
    return a + b;
}
```

Тип ячейки данных

В байте записано следующее значение:

– это что?



Тип ячейки данных

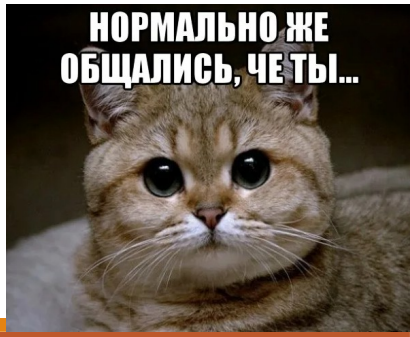
В байте записано следующее значение:

– это что?

Число 97? Или символ ‘a’ или часть большего значения, хранящегося в нескольких байтах?

Неизвестно!

Тип данных определяет операции, которые предполагают некоторую кодировку данных двоичным кодом.



Целые числа: представление и операции

Двоичная система счисления

Самый простой метод записи целых чисел, использующий только две цифры: 0 и 1 (бит).

С помощью n бит можно записать чисел.

Перевод в десятичную систему счисления

Напоминание!

Перевод из любой позиционной системы счисления в десятичную очень прост:

Например:

Перевод в десятичную систему счисления

Напоминание!

Перевод из любой позиционной системы счисления в десятичную очень прост:

Например:

Кодировка беззнаковых чисел

1 байт состоит из 8 бит и может кодировать различных чисел.

Например, число 337 кодируется минимум 2 байтами:

00000001 01010001

Предельные значения:

8 бит: 0…255

16 бит: 0…65 535

32 бита 0…4 294 967 295 64 бита: 0…18 446 744 073 709 551 615

Шестнадцатеричная система счисления

Цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Перевод в двоичную и обратно по 4 бита:

=

Удобная и «понятная» запись:

8 бит: 32 бита:

Знаковые целые числа: простой вариант

Пусть старший бит хранит значение знака:

Диапазон значений уменьшается:

При переполнении (выход за пределы значений):

Знаковые целые числа: простой вариант

При выполнении арифметических операций, необходимо учитывать значение знакового бита.

Иначе получим бредовые результаты!

Знаковые целые числа: простой вариант

При выполнении арифметических операций, необходимо учитывать значение знакового бита.

Иначе получим бредовые результаты!

Если знаки разные, то результат имеет знак большего числа, после чего необходимо произвести вычитание чисел

Можно ли унифицировать сложение и вычитание? Да!

Знаковые целые числа: вариант сложнее

Посмотрим на проблему под другим углом:

, Чему равно X?

Знаковые целые числа: вариант сложнее

Посмотрим на проблему под другим углом:

, Чему равно X?

теперь будет «-1»

Аналогично:

теперь будет «-2»

Двоичный дополнительный код

Таким образом положительные числа остались прежними:

Отрицательные же числа идут следующим образом:

Такая запись – запись числа в двоичном дополнительном коде.

В двоичном дополнительном коде вычитание и сложение – одна операция.

Двоичный дополнительный код

Как получить дополнительный код числа?

Способ простой (человеческий):

, где N – число бит в записи чисел.

Допустим X=6, в системе из 4 разрядов

Тогда

Сложим 6 и -6:

Вычитание заменяется сложением!

Дополнительный двоичный код

Возьмем исходное число

Обратный код числа (все биты, за исключением старшего инвертируются):

Обратный код + 1 = Дополнительный код:

Задача на подумать

Вычисление модуля введенного числа:

```
if (input < 0) {
    output = -input;
} else {
    output = input;
}
```

Исполнение:

```
Input: -2147483648
Output: -2147483648
```

Начинаем изучать новый язык

Далее много формальностей нового для вас языка

Среда разработки

Microsoft visual studio (Она же и в терминалках)
https://visualstudio.microsoft.com/ru/thank-you-downloading-visual-studio/?sku=Community&rel=16

Code::Blocks https://www.codeblocks.org/downloads/

Clion (Продукт платный)
https://www.jetbrains.com/clion/download/#section=windows

Больше можно поискать тут:

https://ru.stackoverflow.com/questions/3592/ide-%D0%B4%D0%BB%D1%8F-c-%D0%B8-c

Структура программы на языке Си

```
#include <stdio.h>

int main()
{
    printf("Hello World!");
    return 0;
}
```

Подключение библиотек, содержащих реализацию функций.

Область глобальных переменных

Объявление функции main (С нее начинается работа программы)

Команды выполняющиеся в данной функции (все команды заканчиваются символом ;)

Основные формальности и проблемы

- Отступы больше не имеют силы, вместо них блоки кода выделяются { }
- Каждая команда завершается ;
- Не сразу будет полностью понятен смысл некоторых вещей (void, *,&…)
- //Комментарий (аналог #) /* Блочный комментарий (аналог тройных кавычек*/
- Программа может компилироваться, но делать все не так, как это было задумано

Структура функции на языке Си

Как написать функцию?

```
Тип Название(тип_аргумента аргумент)
{
Некий код;
return что-то;
}
```

Тип – тип возвращаемого значения.

Название

тип_аргумента – тип аргумента функции. (аргументы перечислять через запятую)

return – выход из функции с передачей значения. (значение строго ОДНО)

Переменные

Переменная – именованное место в памяти, отведенное для хранения данных.

У переменной есть имя – то как мы обращаемся к этому участку памяти.

Переменная может содержать строго определенный тип данных.

Общий вид объявления переменной:

Тип название;	int a;
Тип название = значение;	int b = 0;
Тип название1, название 2;	int c, d;

Базовые типы переменных

int – целочисленное (123) signed char, short int, int, long int, long long int

float – вещественные числа (3.14, но это не точно)

double – вещественные числа удвоенной точности (и другие)

Целочисленные могут быть без знака:

unsigned int Есть «длинные типы» и «короткие типы»

long int, long double… short int, short double

char – символ (‘a’)

Печать в консоль

Синтаксис

printf(“Строка для вывода”);

printf(“a=%d, b=%d”, a, b) – заменит %d на значение переменной интерпретируемое как целочисленное значение.

%d – целочисленное (int)

%f – вещественное (float)

%c – символ (char)

%lf – вещ. удвоенной точности (double)

%% - вывод символа %

\n – перевод строки

\t – табуляция

\” – двойная кавычка

\\ – вывод символа \

\v – вертикальная табуляция

```
int int_val = 5;
float float_val = 3.1;
double double_val = 3.5;
char symbol = 'c';
printf("a = %d\n", int_val);
printf("b = %f\n", float_val);
printf("c = %lf\n", double_val);
printf("d = %c\n", symbol);
printf("float as float = %f \nfloat as int = %d\n", float_val, float_val);
```

```
a = 5
b = 3.100000
c = 3.500000
d = c
float as float = 3.100000
float as int = -1073741824
Для продолжения нажмите любую клавишу . . .
```

Считываем значение из консоли

scanf(“%d”, &a) – Записанное пользователем значение будет интерпретировано как целочисленное значение и записано в переменную a.

Возвращает число присвоенных значений.

```c
int main()
{
    int a = 5;
    int b;
    printf("b = ");
    scanf("%d", &b);
    printf("%d\n", a + b);
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
b = 5
10
Для продолжения нажмите любую клавишу . . .
```

Функция не контролирует правильность ввода!

```c
int main()
{
    char b;
    printf("b = ");
    scanf("%c", &b);
    printf("%d\n", (int)b);
    return 0;
}
```

```
Выбрать C:\Windows\system32\cmd.exe
b = a
97
Для продолжения нажмите любую клавишу . . .
```

Основные арифметические операции

Все стандартные операции в пояснении не нуждаются: + - * /

a++ a-- увеличивает/уменьшает значение a на 1 (можно ставить перед или после переменной)

a=b++ и a=++b = разные результаты! В первом случае b, во втором b+1

a%b – остаток от деления a на b

```
int a = 2;
int b = 5;
int c;
c = a + b;
printf("c = %d\n", c);
c = c*a;
printf("c = %d\n", c);
printf("5%2 = %d\n", b%a); // можно писать выражения вместо конкретной переменной
printf("2*2 = %d\n", 2 * 2); // можно даже так
printf("%d\n", a++);
printf("%d\n", a); // a уже не равен 2
printf("%d\n", 5 / 2);
printf("%f\n", 11.5 / 2.5);
return 0;
```

```
c = 7
c = 14
5 = 1
2*2 = 4
2
3
2
4.600000
Для продолжения нажмите любую клавишу . . .
```

Подробнее о делении и приведении типов

```c
int main()
{
    int a = 5;
    int b = 3;
    printf("a/b = %d\n", a / b);
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
a/b = 1
Для продолжения нажмите любую клавишу . . .
```

```c
int main()
{
    double a = 5;
    double b = 3;
    printf("a/b = %lf\n", a / b);
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
a/b = 1.666667
Для продолжения нажмите любую клавишу . . .
```

А если мы присвоим результат деления двух целых чисел некой переменной типа double?

Подробнее о делении и приведении типов

```c
int main()
{
    int a = 5;
    int b = 3;
    printf("a/b = %d\n", a / b);
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
a/b = 1
Для продолжения нажмите любую клавишу . . .
```

```c
int main()
{
    double a = 5;
    double b = 3;
    printf("a/b = %lf\n", a / b);
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
a/b = 1.666667
Для продолжения нажмите любую клавишу . . .
```

А если мы присвоим результат деления двух целых чисел некой переменной типа double?

```c
int main()
{
    int a = 5;
    int b = 3;
    double result;
    result = a / b;
    printf("a/b = %lf\n", result);
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
a/b = 1.000000
Для продолжения нажмите любую клавишу . . .
```

Подробнее о делении и приведении типов

А если мы присвоим результат деления двух целых чисел некой переменной типа double?

```c
int main()
{
    int a = 5;
    int b = 3;
    double result;
    result = a / b;
    printf("a/b = %lf\n", result);
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
a/b = 1.000000
Для продолжения нажмите любую клавишу . . .
```

Результат будет иметь тип вещественного числа, если хотя бы один из операндов будет иметь тип вещественного числа. Можно привести к типу double явно:

```c
int main()
{
    int a = 5;
    double b = 3;
    double result;
    result = a / b;
    printf("a/b = %lf\n", result);
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
a/b = 1.666667
Для продолжения нажмите любую клавишу . . .
```

```c
int main()
{
    int a = 5;
    int b = 3;
    double result;
    result = (double) a / b;
    printf("a/b = %lf\n", result);
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
a/b = 1.666667
Для продолжения нажмите любую клавишу . . .
```

Оператор ветвления IF

Выполняет проверку истинности условия.

Если условие верно, выполняет определенный код:

```
if ( условие )
    оператор;

if ( условие )
{
    оператор1;
    оператор2;
}
```

== равенство

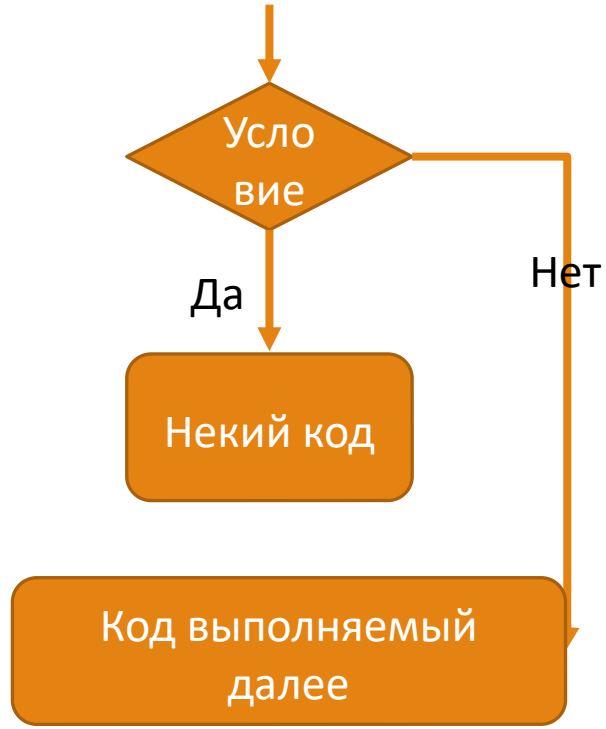
> больше

< меньше

!= не равно

&& И

|| ИЛИ



Оператор ветвления IF

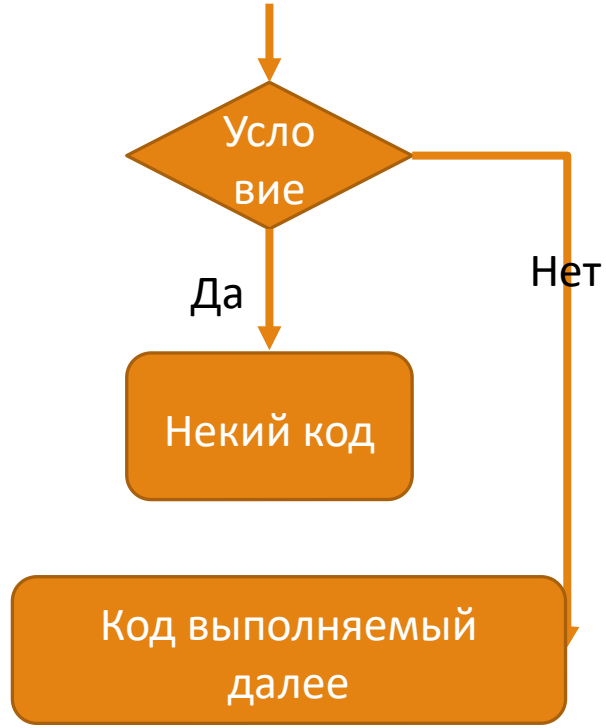
Выполняет проверку истинности условия.

Если условие верно, выполняет определенный код:

```
if ( условие )
    оператор;
```

```
if ( условие )
{
    оператор1;
    оператор2;
}
```

```
int main()
{
    int a;
    printf("a = ");
    scanf("%d", &a);
    if (a > 2)
        printf("a>2\n");
    printf("some text\n");
    return 0;
}
```



Оператор ветвления IF

Выполняет проверку истинности условия.

Если условие верно, выполняет определенный код:

```
if ( условие )
    оператор;
```

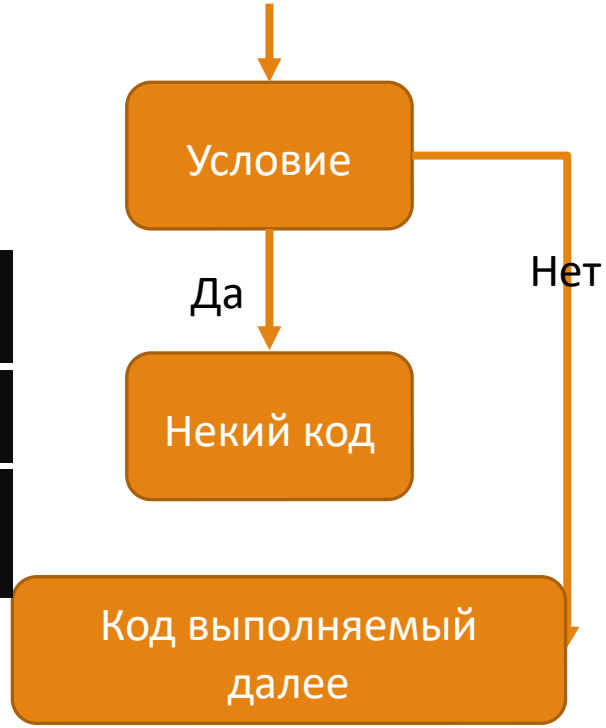
```
if ( условие )
{
    оператор1;
    оператор2;
}
```

```
int main()
{
    int a;
    printf("a = ");
    scanf("%d", &a);
    if (a > 2)
        printf("a>2\n");
    printf("some text\n");
    return 0;
}
```

```
a = 0
some text
Для продолжения нажмите любую клавишу . . .
```

```
a = n
some text
Для продолжения нажмите любую клавишу . . .
```

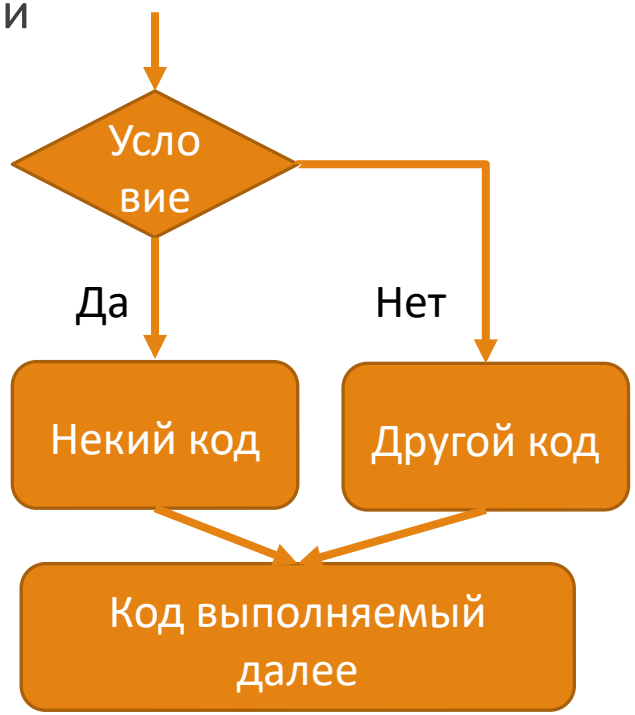
```
a = 7
a>2
some text
Для продолжения нажмите любую клавишу . . .
```



Оператор ветвления IF

Есть дополнение “else” – код выполнится в случае, если условие не выполнено

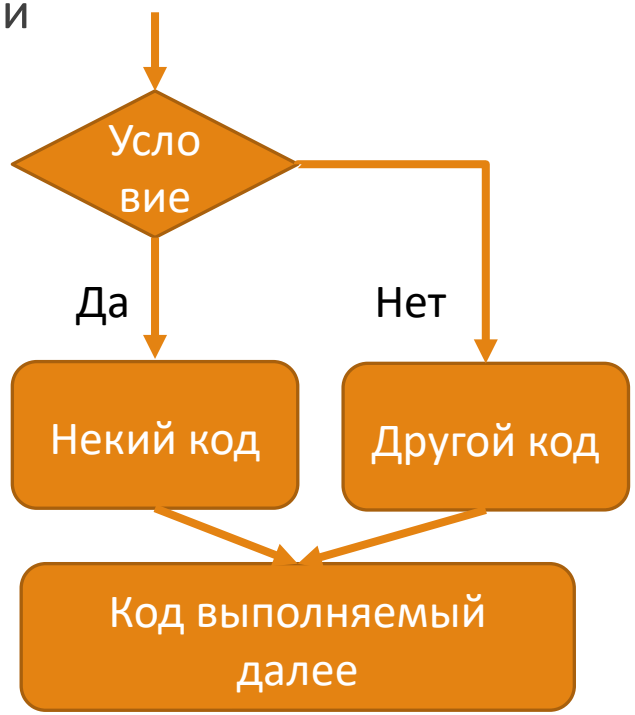
```
if ( условие )
    оператор;
else
    оператор2;
```

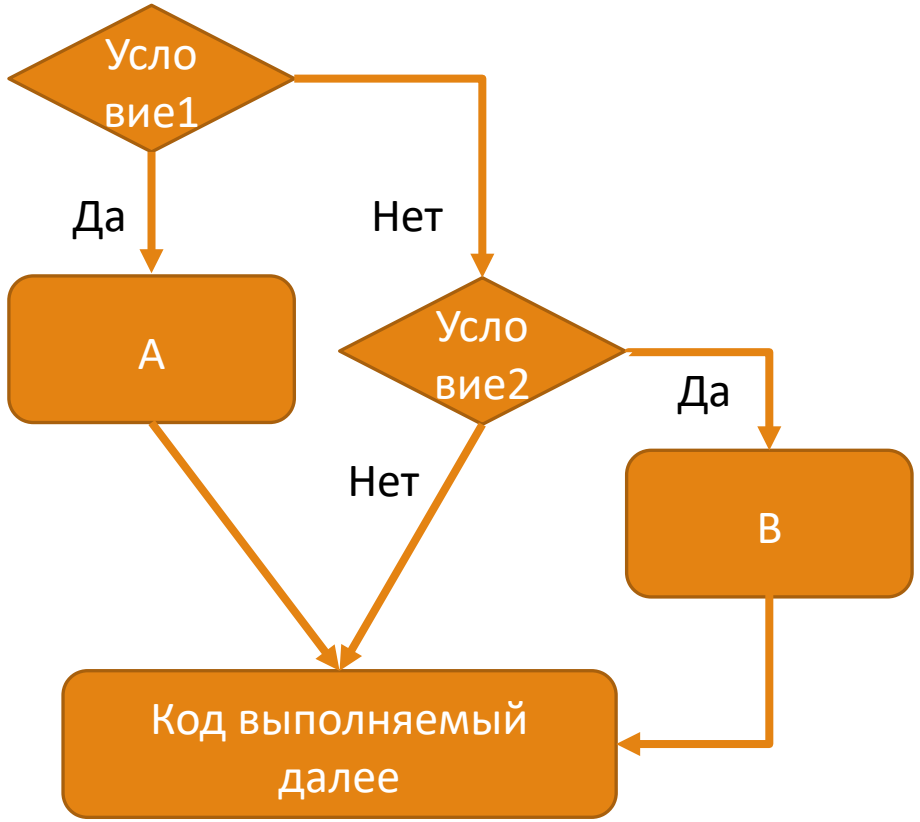


Оператор ветвления IF

Есть дополнение “else” – код выполнится в случае, если условие не выполнено

```
if ( условие )
    оператор;
else
    оператор2;
```





Ну на сегодня хватит с вас