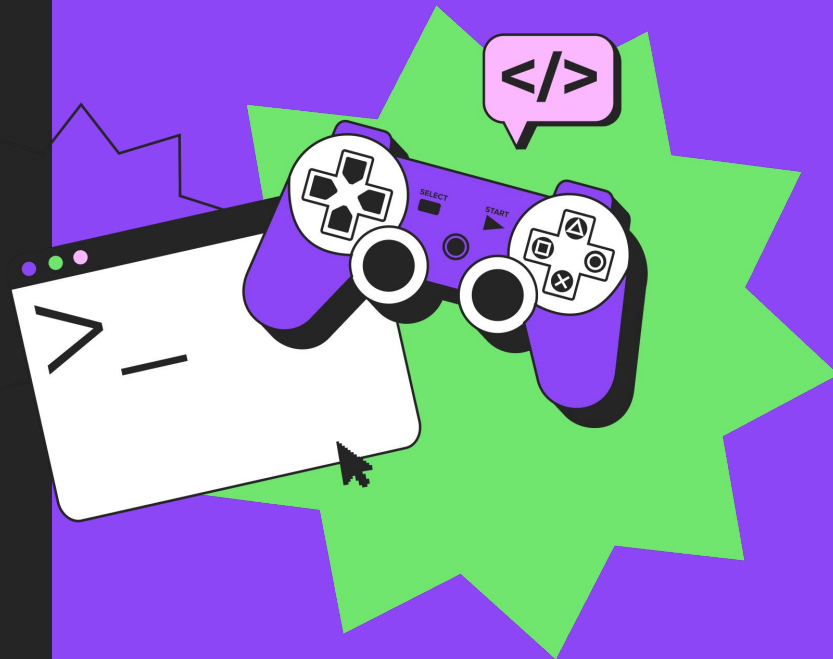


# Циклы

## Урок 4

Программирование на языке Си (базовый уровень)





# План курса

1

Введение в язык Си

2

Типы данных. Операторы и выражения

3

Ветвления и побитовые операции

4

Циклы

5

Буферный ввод-вывод.  
Функции

6

Область видимости.  
Указатели. Рекурсия

7

Вещественные типы  
данных. Массивы

8

Адресная арифметика.  
Массивы, строки

9

.Структурные типы данных.  
Файлы

10

Многомодульные  
программы

11

Аргументы командной  
строки. Препроцессор.  
Отладка программ



На этой лекции вы узнаете про:

- Операторы выбора `switch`
- Циклы и какие типы циклов бывают
- Оператор перехода `goto`
- Область видимости переменных
- Описание целых констант через `enum`
- Описание констант через `#define`
- Как правильно выбрать имя переменной




# Оператор выбора switch



# Оператор выбора switch

Оператор выбора switch позволяет:

- выполнить один из операторов в зависимости от значения целочисленного выражения
- если ни один из вариантов значения не подошел, то выполнить оператор по умолчанию (default),

 В операторе выбора нельзя использовать логические операции и сложные условия, т. е. мы сравниваем значение переменной с константой



## Синтаксис оператора выбора switch

```
switch (variable) //целочисленное выражение
{
    case expression-equal_1:
        statement_1; //оператор который будет выполнен
    break;
    case expression-equal_2:
        statement_2;
    break;
    default:
        statement_3; //оператор по умолчанию
}
```



Код будет выполняться до тех пор пока не встретиться оператор *break* или не закончится секция *switch*

## Пример switch(expr)



```
#include <stdio.h>
int main()
{
    int a = 1, b = 1, expr = 2;
    switch (expr)
    {
        case 2:
            a *= 2 ; // Если expr == 2, то выполнится a += 5; из следующей ветки
        case 3:
            a += 5;
            break; // А здесь произойдет выход
        case 4:
            a -= b;
            break;
        default:
            break;
    }
    printf("a = %d",a);
    return 0;
}
```



## Пример switch(input)

```
#include <stdio.h>

int main()
{
    int input;
    scanf("%d",&input);
    switch(input)
    {
        case 1: printf("one\n"); break;
        case 2: printf("two\n");
        case 3: printf("three\n"); break;
        case 4: printf("four\n"); break;
        default: printf("default\n");
    }
    return 0;
}
```





# Циклы

- for — с известным числом шагов;
- while — с предусловием;
- do...while — с постусловием;



# Циклы

**Цикл** — это многократное выполнение одинаковых действий.

Состоит из:

- **заголовка** — блока проверки условия повторения цикла
- **тела цикла** — последовательности инструкций, предназначенных для многократного исполнения

Цикл выполняется до тех пор, пока блок проверки условия возвращает истинное значение.



## Типы циклов

В языке Си есть следующие виды циклов:

- Цикл с известным числом шагов - *for*
- Цикл с неизвестным числом шагов (цикл с условием) - *while*
  - *while* с предусловием
  - *do {...} while* с постусловием



## Задача

```
#include <stdio.h>
int main(void)
{
    printf("Hello\n");
    printf("Hello\n");
    printf("Hello\n");
    printf("Hello\n");
    printf("Hello\n");
    return 0;
}
```



Нарушается принцип DRY — это "Don't Repeat Yourself" (не повторяйся)



# Цикл с условием while



## Цикл с условием while

Цикл `while` проверяет истинность некоторого условия, и если условие истинно, то есть не равно 0, то код цикла выполняется.

→ В цикле `while` можно использовать сложные условия

```
while ( a < b && b < c )  
{ ... }
```

→ Если в цикле только один оператор, то скобки `{ }` можно не писать

```
while ( a < b )  
    a++;
```



## Примеры while

```
#include <stdio.h>
int main()
{
    int n;
    n=0;
    while (n!=5) { // лучше n<5
        printf("Hello\n");
        n++; // n = n+1
    }
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    int n;
    n=5;
    while (n!=0) {
        printf("Hello\n");
        n--;
    }
    return 0;
}
```

## Вопрос: Что будет напечатано?

```
#include <stdio.h>
int main()
{
    int n;
    n=1;
    while (n <= 5) {
        printf("%d\n", n);
        n++;
    }
    return 0;
}
```





## Ответ: Что будет напечатано?

```
int main()  
{  
    int n;  
    n=1;  
    while (n <= 5) {  
        printf("%d\n", n);  
        n++;  
    }  
    return 0;  
}
```

1  
2  
3  
4  
5





## Особенности циклов `while`

- Условие пересчитывается каждый раз при входе в цикл
- Если условие ложно, цикл не выполнится ни разу
- Если условие никогда не станет ложным, то цикл зациклится

```
while ( Условие )  
{ ... }
```

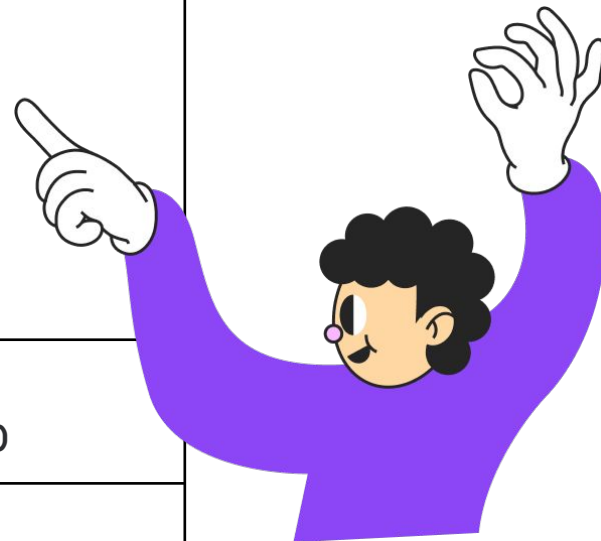
# Сколько раз выполнится цикл? Чему равно значение a и b?

<pre>int a = 4, b = 6; while (a &lt; b) a++;</pre>	? раз a = ?
<pre>int a = 4, b = 6; while (a &lt; b) a+=b;</pre>	? раз a = ?
<pre>int a = 4, b = 6; while (a &gt; b) a++;</pre>	? раз a = ?



# Сколько раз выполнится цикл? Чему равно значение a и b?

<pre>#include &lt;stdio.h&gt; int main(void) {     int a = 4, b = 6, counter=0;     while (a &lt; b)     {         a++;         counter++;     }     printf("a=%d,b=%d counter=%d",a,b,counter); }</pre>	2 раза a = 6
<pre>int a = 4, b = 6; while (a &lt; b) a+=b;</pre>	1 раз a = 10
<pre>int a = 4, b = 6; while (a &gt; b) a++;</pre>	0 раз a = 4





## Задача: Ввести целое число и определить количество цифр в нем

```
#include <stdio.h>
int main()
{
    int input, n, count;
    printf("Input number: ");
    scanf("%d", &input);
    count = 0;
    n = input;
    while (n != 0)
    {
        count ++;
        n = n / 10; // Отбросили одну цифру
    }
    printf("In %d found %d digits", input, count);
    return 0;
}
```



Задача: Дано натуральное число n. Получить все его натуральные делители

```
#include <stdio.h>
#include <inttypes.h>
int main(void)
{
    uint32_t n, i=2;
    scanf("%u" PRIu32 , &n);
    while(i<=n) {
        if(n%i == 0) {
            printf("%u" PRIu32 " ", i);
            n/=i;
        }
        else {
            i++;
        }
    }
    printf("\n");
    return 0;
}
```




Цикл с постусловием `do...while`



## Цикл с постусловием do...while

Такой цикл всегда выполняется хотя бы один раз.

- Условие проверяется каждый раз после выполнения тела цикла
- Если условие ложно, цикл все равно выполнится один раз
- Если условие никогда не станет ложным, то цикл зациклится

 `do { ... } while (a < b);` Ставим точку с запятой в конце





## Пример

```
#include <stdio.h>

int main(void)
{
    int a = 4, b = 6, counter=0;
    do
    {
        a++;
        counter++;
    }while (a < b); // Ставим точку с запятой в конце
    printf("a=%d,b=%d counter=%d",a,b,counter) ;
    return 0;
}
```



Цикл с известным числом шагов for



## Цикл с известным числом шагов `for`

Цикл `for` является самым мощным видом цикла в Си.

- Перед началом выполнения цикла вычисляется выражение `expression1`
- Тело цикла – оператор `statement` – выполняется до тех пор, пока значение выражения `expression2` оказывается не ложным
- После окончания итерации цикла вычисляется выражение `expression3`

```
for (expression1; expression2; expression3) statement;
```



Любое из выражений *expression*, а также оператор *statement*, может быть пустым.



## Особенности цикла for

- условие проверяется в начале очередного шага цикла,
- если оно ложно, то цикл не выполняется;
- изменения счетчиков выполняются в конце очередного шага цикла;
- если условие никогда не станет ложным, цикл может зациклиться;

```
for (expression1; expression2; expression3) statement;
```



Переменную-счетчик можно объявить прямо в заголовке цикла в C99



**Пример: выведем квадраты первых ста чисел.**

```
#include <stdio.h>
int main(void)
{
    for (int i = 1; i < 101; i++)
    {
        printf("%05d ", i*i);
        if(i%10==0)
            printf("\n");
    }
    return 0;
}
```

# Сколько раз выполнится цикл? Чему равно значение a?

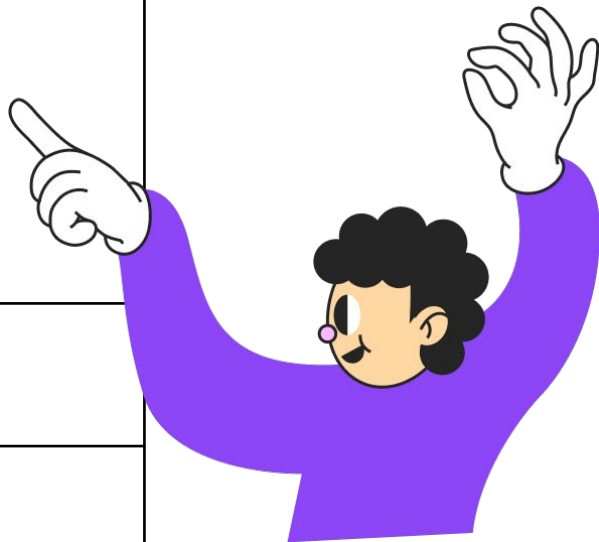
<pre>int i, a = 1; for(i = 1; i &lt; 4; i++) a++;</pre>	? раз a = ?
<pre>int i, a = 1; for(i = 1; i &lt; 4; i++) a = a + i;</pre>	? раз a = ?
<pre>int i, a = 1, b = 2; for(i = 3; i &gt;= 1; i--) a += b;</pre>	? раз a = ?



Поставьте видео  
на паузу и  
решите задачу

# Сколько раз выполнится цикл? Чему равно значение a?

<pre>#include &lt;stdio.h&gt; int main(void) {     int a = 1, counter=0,b=2;     for(int i = 1; i &lt; 4; i++)     {         a++;         counter++;     }     printf("a=%d,b=%d counter=%d",a,b,counter);     return 0; }</pre>	3 раза a=4
<pre>int i, a = 1; for(i = 1; i &lt; 4; i++) a = a + i;</pre>	3 раза a=7
<pre>int i, a = 1, b = 2; for(i = 3; i &gt;= 1; i--) a += b;</pre>	3 раза a=7





## Замена for на while

Всегда можно произвести замену цикла for на цикл while и наоборот.

```
for (i = 1; i <= 5; i++)  
{  
    statement;  
}
```

```
i = 1;  
while (i <= 5) {  
    statement;  
    i++;  
}
```

```
for (i = a; i >= b; i--)  
{  
    statement;  
}
```

```
i=a;  
while (i>=b) {  
    statement;  
    i--;  
}
```





# Замена for на while

Напечатать квадраты целых чисел от 1 до 10.

```
#include <stdio.h>
int main(void)
{
    // Используя цикл while
    int i=1;
    while (i<=10) {
        printf("%d\n",i*i);
        i++;
    }
    return 0;
}
```

```
#include <stdio.h>
int main(void)
{
    // Используя цикл for
    for(int i=1; i<=10; i++)
        printf("%d\n",i*i);
    return 0;
}
```



# Операторы перехода (break, continue, goto)



# Оператор continue

Continue – используется для немедленного перехода к следующей итерации цикла.

Для цикла *for*:

- пропускается оставшаяся часть тела цикла, идущая после continue,
- при этом выражение expression3 не пропускается и выполняется до начала следующей итерации
- далее начинается новая итерация

```
for (expression1; expression2; expression3) statement;
```



# Оператор `continue while`

Для цикла *while*:

- Выполняется первая часть тела цикла **statement1**
- Пропускается оставшаяся часть тела цикла **statement2**, идущая после `continue`
- Далее начинается новая итерация

```
while (expression)
{
    statement1;
    continue;
    statement2;
}
```



## Пример continue

Не выводить на печать цифру “3”.

```
#include <stdio.h>
int main(void)
{
    int i;
    for(i = 1; i < 5; i++) {
        if (i == 3)
            continue;
        printf("i = %d\n", i);
    }
    return 0;
}
```

```
i = 1
i = 2
i = 4
```



# Оператор break

- Оператор break завершает выполнение ближайшего внешнего оператора **do** , **for** , **switch** или **while** , в котором он находится
- Управление передается оператору, который расположен после соответствующего завершенного оператора **do** , **for** , **switch** или **while**

```
// Выходим из бесконечного цикла, когда нажата Q
for(;;) { // бесконечный цикл
    if (c == 'Q')
        break;
}
```



## Пример break: выход из системы меню

```
#include <stdio.h>
int main() {
    char c;
    for(;;) { //бесконечный цикл
        printf( "\nPress any key, Q to quit: " );
        // Считываем введенные символ
        scanf("%c", &c);
        if (c == 'Q' || c == 'q') //Не важно Q или q
            break;
        printf("%x\n", c); //0x0a - перевод строки '\n'
    }
    return 0;
} // Выходим из бесконечного цикла, когда нажата Q или q
```

# Пример break: Распечатать первые n простых чисел



```
#include <stdio.h>
//( n - простое число, если n >= 2 и делится только на 1 и на себя)
int main() {
    _Bool is_prime;
    int n;
    scanf("%u",&n);
    for(int i=2; i<n; i++) {
        is_prime=1;
        for(int j=2; j*j<=i; j++) // j <= i
            if (i%j == 0) {
                is_prime=0;
                break; //прервать внутренний цикл
            }
        if (is_prime)
            printf("%u ",i);
    }
    return 0;
}
```





# Сравнение `break` и `continue`

Можно сказать, что оператор *continue* немного похож на *break*.

- *break* — вызывает прерывание цикла
- *continue* — переход к следующей итерации.



# Оператор перехода goto

Оператор перехода по метке **goto label:** используется для организации сложного потока управления.

- В качестве метки выступает идентификатор с двоеточием в конце
- Метка должна находиться в той же функции, что и **goto**,
- Место перехода помечается в программе как **label:**
- Метка в коде прижимается к левому краю.

```
goto label;  
...  
label:
```



## Особенности оператора перехода `goto`



Нужно использовать с осторожностью оператор `goto`. Так как это может сделать программу трудно читаемой и тяжелой для восприятия.



Переходить можно только внутри области видимости, нельзя перейти из одной функции в другую



## Рекомендации написания драйверов под Linux

“Возвращаемые ошибки иногда лучше обрабатывать инструкцией *goto*. Мы обычно ненавидим использовать *goto*, но на наш взгляд это та ситуация, в которой она полезна.

Осторожное использование *goto* в ошибочных ситуациях может устранить большую, сложную, высоко-вложенную, "структурированную" логику. Таким образом, как показано здесь, в ядре *goto* часто используется для исправления ошибки.”

“Драйверы Устройств Linux, 3-я редакция”



## Пример goto

```
#include <stdio.h>

int main()
{
    printf("Hello");
    goto skip;
    printf("World");
skip:
    return 0;
}
```

Hello



# Область видимости переменных (scope)



# Область видимости переменных

Является для:

- **локальной** переменной – блок (функция), в которой она объявлена,
- **глобальной** переменной – программный файл, начиная со строки объявления.

🔥 Не стоит называть переменные также как и во внешней области видимости

🔥 По-возможности избегайте использования не константных глобальных переменных.



## Пример видимости переменных

```
#include <stdio.h>
/*Глобальная переменная. Видна внутри всего файла.*/
int a=5;
int main()
{
/*Локальная переменная видна только внутри main. */
    int a=10;
    printf("a = %d",a);
    return 0;
}
```

a = 10





## Пример видимости переменных if

```
#include <stdio.h>
int a = 5;
int main()
{
    int a = 10;
    if (1 == 1) {
        int a = 20; /*Локальная переменная видна только
внутри if. Не стоит называть переменные также как и во
внешней области видимости */
        printf("a = %d\n",a);
    }
    printf("a = %d\n",a);
    return 0;
}
```

a = 20

a = 10



## Пример видимости переменных цикл for

```
#include <stdio.h>
int main()
{
    int i, sum = 0;
    for(i = 0; i<5; i++) {
        sum += i;
    }
    printf("%d\n",i); // i = 5
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    int sum = 0;
    // переменная i видна только
    // внутри for
    for(int i = 0; i<5; i++) {
        sum += i;
    }
    printf("%d\n",i); // ошибка
    return 0;
}
```



## Пример видимости переменных и замена цикла for на while

```
#include <stdio.h>
int main()
{
    int i, sum = 0;
    for(i = 0; i<5; i++) {
        sum += i;
    }
    printf("%d\n",i); // i = 5
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    int sum = 0;
    int i = 0;
    while(i<5) {
        sum += i;
        i++;
    }
    printf("%d\n",i); // 5
    return 0;
}
```



# Недостатки глобальных переменных

- Нарушают инкапсуляцию
- Вызывают путаницу в большом проекте
- Ухудшают масштабируемость проекта
- Ухудшают читаемость кода
- Приводят к трудно уловимым ошибкам
- Увеличивают число прямых и косвенных связей в системе, делая её поведение трудно предсказуемым, а её саму – сложной для понимания



# Константы: enum и #define



## Описание целых констант `enum`

```
enum { N = 100, K = 500 };
```

- Константы имеющие целое значение можно задавать через определение **перечислимого типа** `enum`.
- Подчиняются правилам видимости идентификаторов языка Си.



## Пример enum

*// Плохо*

```
unsigned int a, h;  
char ln[25], f[25];
```

*// Хорошо*

```
enum { NAME_LENGTH = 25 };  
unsigned int age, height;  
char      lastName[NAME_LENGTH];  
char firstName[NAME_LENGTH];
```



# Описание констант через #define

```
#define [ИДЕНТИФИКАТОР] [ОТСТУП] [ЗАМЕНА]
```

- #define — это макрос. **Перед** компиляцией в коде будет произведена замена идентификатор на последовательность символов
- Удобно, когда одно и то же значение используется множество раз





## Пример #define

*// Плохо*

```
unsigned int a, h;  
char ln[25], f[25];  
for(int i=0; i<25; i++)
```

*// Хорошо*

```
#define NAME_LENGTH 100  
unsigned int age, height;  
char lastName[NAME_LENGTH];  
char firstName[NAME_LENGTH];  
for(int i=0; i<NAME_LENGTH; i++)
```



# Сравнение const, #define и enum в Си и Си++

`const int var = 5;` C++ стиль

`#define var 5` assembler стиль

`enum { var = 5 };` хорошо для машин состояний и  
перечислений

В С++ предпочтительней использовать const и enum использованию #define”, поскольку #define зачастую вообще не относят к языку С++

«Компилятор предпочтительнее препроцессора»



# Форматирование кода

Как выбрать имя переменной?  
Форматирование кода для цикла.






# Как выбрать имя переменной

- Должно отражать суть того, для чего данная переменная используется
- Составлять из нескольких английских слов
- Избегайте использования однобуквенных переменных
- Переменные и идентификаторы пишутся в camelCase
- Константы пишутся в SCREAMING\_CASE




Вопрос. 🙄 Какой наилучший префикс для глобальной переменной?

Ответ: //

## Форматирование кода для цикла

Пример 1	Пример 2	Пример 3
<pre>while (p) {     s+= p-&gt;data;     p = p-&gt;next; }</pre> 	<pre>while (p){     s+= p-&gt;data;     p = p-&gt;next; }</pre> 	<pre>while (p) {    s+= p-&gt;data;     p = p-&gt;next; }</pre> 

## Форматирование кода для цикла

Пример 4	Пример 5	Пример 6
<pre>while (wait(p) != -1) { }</pre>	<pre>while (p) {     s += p-&gt;data;     p = p-&gt;next; }</pre>	<pre>while (wait(p) != -1) ;</pre>
		



# Вычисление корней квадратного уравнения (продолжение)




```
#include <locale.h>
#include <stdio.h>
int main(int argc, char **argv)
{
    char Choice;
    setlocale(LC_ALL, "Rus");
    while(1)
    {
        printf("1. Вычисление корней
квадратного уравнения\n");
        printf("0. Выход\n");
        printf("Для выход нажмите Q\n");
    NO_PRINT:
        scanf("%c",&Choice);
        printf("%x\n",Choice);
    }
```

```
switch(Choice)
{
    case '1':
        printf("SquarEquation()\n");
        break;
    case '0':
    case 'q':
        return 0;
    break;
    case 0xa:/'\n':
        goto NO_PRINT;
    break;
    default:
        printf("Непонятный выбор\
%x\n",Choice);
        break;
}
}
return 0;
}
```



На этой лекции мы изучили:

- Операторы выбора `switch`
- Циклы и какие типы циклов бывают
- Изменение логики `break` и `continue`
- Оператор перехода `goto`
- Область видимости переменных
- Описание целых констант через `enum` и `#define`
- Узнали как форматировать код для цикла

**Спасибо**   
**за внимание**

Просто делай дело и перестань беспокоиться!

