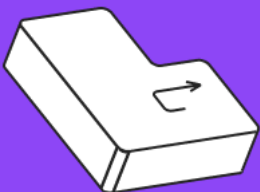




Урок 4.

Циклы

Программирование на языке Си
(базовый уровень)



Оглавление

Введение	2
Термины, используемые в лекции	3
Оператор выбора switch	4
Циклы	5
Цикл с условием while	6
Особенности циклов while	8
Цикл с постусловием do...while	10
Цикл с известным числом шагов for	11
Особенности цикла for	11
Замена for на while	13
Оператор continue	13
Оператор break	14
Оператор перехода goto	16
Область видимости переменных	17
Описание целых констант enum	19
Описание констант через #define	20
Как выбрать имя переменной	20
Форматирование кода для цикла	21
Вычисление корней квадратного уравнения (продолжение)	22
Подведение итогов	23
Дополнительные материалы	23
Используемые источники	24

Введение

На предыдущей лекции вы узнали:

- Спецификаторы фиксированной длины в C99
- Узнаем про явное и неявное приведение типов

- Логические операции
- Условный оператор
- Тернарные операции $a ? b : c$;
- Побитовые операции
- Приоритет выполнения операций
- Что такое оператор присваивания его побочный эффект и точка следования
- Форматирование кода

На этой лекции вы найдете ответы на такие вопросы как / узнаете:

- Рассмотрим оператор выбора `switch`
- Узнаем, что такое циклы и какие типы циклов бывают
- Изучим оператор перехода `goto`
- Поймем, что такое область видимости переменных
- Описание целых констант через `enum`
- Описание констант через `#define`
- Разберём, как правильно именовать переменные

Термины, используемые в лекции

Цикл — это многократное выполнение одинаковых действий.

Заголовок цикла — блок проверки условия повторения цикла

Тело цикла — последовательность инструкций, предназначенная для многократного исполнения

continue — оператор, который используется для немедленного перехода к следующей итерации цикла, пропуская оставшуюся часть тела цикла.

break — оператор, завершающий выполнение ближайшего внешнего оператора `do` , `for` , `switch` или `while` , в котором он находится.

Область видимости переменных (scope) — это участок программы, в рамках которого можно использовать переменную.

Глобальные переменные — переменные, объявленные вне определения функций.

Локальные переменные — переменные, объявленные внутри функции или блока операторов.

Оператор выбора switch

Оператор выбора switch является очень удобной заменой множественного использования операторов if. Оператор switch сравнивает значение одной переменной с несколькими константами. Основной формат для использования оператора множественного выбора switch case показан — ниже.

```
switch (expression) //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
}
```

Оператор выбора **switch** позволяет:

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

После ключевого слова *switch* в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями после оператора case. После того, как одно из значений подошло, происходит переход по соответствующей метке case и начинается выполнение тела switch с оператора после этой метки. Остальные операторы switch также выполняются в порядке их записи, поэтому для прекращения выполнения оператора выбора обязательно нужно использовать оператор *break*:

```
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;
}
```

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



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

Циклы

Цикл – это многократное выполнение одинаковых действий.

Каждый цикл состоит из

- блока проверки условия повторения цикла
- тела цикла

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

В языке Си следующие виды циклов:

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

Задача. Напечатать 5 раз слово “Hello”.

Особенность: одинаковые действия выполняются 5 раз.

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



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

Цикл с условием while

В случае, если количество операций заранее не известно, лучше использовать цикл *while*. Проверка условия осуществляется в заголовке перед телом цикла. Слева цикл с нарастающим счетчиком, справа цикл с убывающим.

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

```
int main()
{
    int n = 5;
    while (n != 0) {
        printf("Hello\n");
        n--;
    }
    return 0;
}
```

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++;
```

Задача. Что будет напечатано?

```
int n;
n=1;
while (n <= 5) {
```

?

<pre>printf("%d\n", n); n++; }</pre>	
<pre>int n; n=1; while (n <= 5) { printf("%d\n", n); n = n+2; }</pre>	?
<pre>int n; n=2; while (n != 5) { printf("%d\n", n); n = n+2; }</pre>	?
<pre>int n; n=5; while (n >= 1) { printf("%d\n", n*n*n); n--; }</pre>	?



Ответ

<pre>int n; n=1; while (n <= 5) { printf("%d\n", n); n++; }</pre>	1 2 3 4 5
--	-----------------------

<pre> }</pre>	
<pre> int n; n=1; while (n <= 5) { printf("%d\n", n); n = n+2; }</pre>	1 3 5
<pre> int n; n=2; while (n != 5) { printf("%d\n", n); n = n+2; }</pre>	2 4 6 8 10 ...
<pre> int n; n=5; while (n >= 1) { printf("%d\n", n*n*n); n--; }</pre>	125 64 28 8 1

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

Задача. Сколько раз выполнится цикл? Чему равно значение a и b?

<pre> int a = 4; b = 6; while (a < b) a++;</pre>	? раз a = ?
<pre> int a = 4; b=6; while (a < b) a+=b;</pre>	? раз a = ?
<pre> int a = 4; b = 6; while (a > b) a++;</pre>	? раз a = ?
<pre> int a = 4; b = 6; while (a < b) b = a - b;</pre>	? раз b = ?
<pre> int a = 4; b = 6; while (a < b) a--;</pre>	? раз



Ответ

<pre>int a = 4; b = 6; while (a < b) a++;</pre>	2 раза a = 6
<pre>int a = 4; b=6; while (a < b) a+=b;</pre>	1 раз a = 10
<pre>int a = 4; b = 6; while (a > b) a++;</pre>	0 раза a = 4
<pre>int a = 4; b = 6; while (a < b) b = a - b;</pre>	1 раз b = -2
<pre>int a = 4; b = 6; while (a < b) a--;</pre>	зацикливание

Задачи

1. Ввести целое число и определить количество цифр в нем.

```
#include <stdio.h>

int main()
{
    int n, count;
    printf("Input number: ");
    scanf("%d", &n);
    count = 0;
    while (n != 0)
    {
        count ++;
```

```

        n = n / 10; // Отбросили одну цифру
    }
    printf("In %d found %d digits", n, count);
    return 0;
}

```

2. Ввести целое число и определить сумму цифр в нем.
3. Ввести целое число и определить произведение цифр в нем.
4. Найти наименьший общий делитель (НОД) двух чисел. Алгоритм Евклида.
 $\text{НОД}(14, 35) = \text{НОД}(14, 35-14) = \text{НОД}(14, 21) = \text{НОД}(14, 21-14) = \text{НОД}(14, 7) = \text{НОД}(7, 7) = 7$
5. Модифицированный алгоритм Евклида.
 $\text{НОД}(14, 35) = (14, 35\%14) = \text{НОД}(14, 7) = \text{НОД}(14\%7, 7) = \text{НОД}(0, 7) = 7$
6. 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;
}

```

Решение задач 3-5 будет разобрано на семинаре

Цикл с постусловием do...while

Такой цикл всегда выполняется хотя бы один раз.

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

```
int a = 4, b = 6;
do {
    a++;
}while (a < b); // Ставим точку с запятой в конце
```



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

Цикл с известным числом шагов for

Цикл **for** является самым мощным видом цикла в Си. Аналогично реализован этот цикл в языках C++, C# и других языках программирования.

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

Перед началом выполнения цикла вычисляется выражение *expression1*, обычно, используемое для инициализации счетчиков цикла. При этом переменную-счетчик можно объявить прямо в **заголовке цикла**.

Тело цикла – оператор *statement* – выполняется до тех пор, пока значение выражения *expression2* оказывается не ложным, это выражение вычисляется перед началом каждой итерации цикла.

После окончания итерации цикла вычисляется выражение *expression3*, обычно содержащее обновление счетчика цикла.



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



Переменную-счетчик можно объявить прямо в заголовке цикла в C99

Особенности цикла for

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

Задача. Сколько раз выполнится? Чему будет равно значение a?

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



<pre>int i, a = 1; for(i = 1; i < 4; i++) a++;</pre>	3 раза a=4
<pre>int i, a = 1; for(i = 1; i < 4; i++) a = a + i;</pre>	3 раза a=7
<pre>int i, a = 1, b = 2; for(i = 3; i >= 1; i--) a += b;</pre>	3 раза a=7
<pre>int i, a = 1; for(i = 1; i >= 3; i--) a++;</pre>	0 раз a=1
<pre>int i, a = 1; for(i = 1; i <= 4; i--) a++;</pre>	зацикливание



Для того чтобы создать бесконечный цикл, можно записать цикл так:
`for(;;) {};`



Для создания бесконечного цикла *while* необходимо просто создать условие, которое будет всегда истинным, например, вот так:

```
while(1) {
    // main loop в IoT и Embedded
}
```

Замена for на while

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

Для каких-то задач удобнее использовать *for*, для других - *while*.

Для обхода массива, например, цикл *for* гораздо удобнее.

while удобен, когда не знаешь количество итераций (проходов цикла), например приход конца файла.

<pre>for(i = 1; i <= 5; i++) { statement; }</pre>	<pre>i = 1; while(i <= 5) { statement; i++; }</pre>
<pre>for(i = a; i >= b; i--) { statement; }</pre>	<pre>i=a; while(i>=b) { statement; i--; }</pre>

Оператор continue

Оператор **continue** — используется для немедленного перехода к следующей итерации цикла, пропуская оставшуюся часть тела цикла, при этом выражение *expression3* не пропускается и выполняется до начала следующей итерации.

Использование оператора *continue* в цикле эффективно, если нужно пропустить некоторые итерации в зависимости от условия.

Требуется пропустить третью итерацию цикла и не выводить на печать цифру “3”.

<pre>#include <stdio.h> int main(void) {</pre>	<pre>i = 1 i = 2</pre>
--	------------------------

<pre> int i; for(i = 1; i < 5; i++) { if (i == 3) continue; printf("i = %d\n", i); } return 0; } </pre>	<pre> i = 4 </pre>
--	--------------------

Оператор break

Бывает, что цель выполнения цикла достигается раньше, чем он будет прекращен по условию выхода.

Оператор **break** завершает выполнение ближайшего внешнего оператора *do* , *for* , *switch* или *while* , в котором он находится. Управление передается оператору, который расположен после заверщенного оператора. См. задачу 5.

Код ниже реализует выход из системы меню по клавише q

<pre> #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 </pre>

Задачи.

1. Напечатать квадраты целых чисел от 1 до 10.

<pre> // Используя цикл while int i=1; while (i<=10) { printf("%d\n", i*i); i++; } </pre>	<pre> // Используя цикл for int i; for(i=1; i<=10; i++) { printf("%d\n", i*i); } </pre>
--	--

2. Подсчитать количество натуральных чисел n ($102 \leq n \leq 987$), в которых все три цифры различны.

3. Дано натуральное число n . Получить наименьшее число вида 2^k , превосходящее n . (Используйте побитовые операции).

```
unsigned int i,n, pow2=1;
scanf("%u",&n);
for(i=0; pow2<n; i++)
    pow2 <<= 1; // pow2 = pow2 * 2
printf("%u\n",pow2);
```

4. Подсчитать количество натуральных чисел n ($11 \leq n \leq 999$), являющихся палиндромами, и распечатать их.

```
#include <stdio.h>
#include <inttypes.h>

int main()
{
    uint32_t count=0;
    for(uint32_t i=11;i<=999;i++) {
        if ( i<100 && i%10 == i/10 ) {
            count++;
            printf("%" PRIu32 " ",i);
        } else if(i>=100 && i%10 == i/100) {
            count++;
            printf("%" PRIu32 " ",i);
        }
    }
    printf("Total polindroms %" PRIu32 "\n",count);
    return 0;
}
```

5. Распечатать первые n простых чисел (n - простое число, если $n \geq 2$ и делится только на 1 и на себя).

```
#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;
}

```

6. Распечатать первые n чисел Фибоначчи.

Задачи 2 и 6 для самостоятельной работы.

Оператор перехода goto


Оператор перехода по метке **goto label**; используется для организации сложного потока управления. В качестве метки может выступать идентификатор, областью видимости которого оказывается вся функция. Место перехода помечается в программе как **label:**. Метка прижимается к левому краю.


<pre> #include <stdio.h> int main() { printf("Hello"); goto skip; printf("World"); skip: return 0; } </pre>	<p>Hello</p>
--	---------------------

В рекомендациях написания драйверов под Linux (“Драйверы Устройств Linux, 3-я редакция”) говорится:

“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.”

 **Внимание!** 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

Область видимости переменных

Глобальные переменные объявляются вне определения функций, а **локальные** – внутри функции или блока операторов.

Областью видимости переменной является для:

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

Внутри одной области видимости может быть лишь одна переменная с определенным именем, при этом возможно перекрытие имен – если имена локальной и глобальной переменной совпадают, то в локальной области видимости используется локальная переменная, а не глобальная.

<pre>#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; }</pre>	<pre>a = 10</pre>
--	-------------------

<pre>#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; }</pre>	<pre>a = 20 a = 10</pre>
---	--------------------------



Не стоит называть переменные также как и во внешней области видимости

В примере справа переменная `i` видна только внутри цикла `for`, поэтому при компиляции возникнет ошибка.

<pre>int main() { int i, sum=0; for(i=0; i<5; i++) { sum+=i; } printf("%d\n",i); // i = 5 return 0; }</pre>	<pre>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; }</pre>
--	---



По-возможности избегайте использования не константных глобальных переменных.

Недостатки глобальных переменных:

1. Глобальные переменные в большинстве случаев нарушают инкапсуляцию. К ним открыт неконтролируемый доступ отовсюду.
2. В большом проекте при обилии глобальных переменных возникает путаница в именах. Глобальную переменную же видно отовсюду, надо, чтобы отовсюду было понятно, зачем она.
3. Глобальные переменные ухудшают масштабируемость проекта.
4. Глобальные переменные ухудшают читаемость кода (в каком-то конкретно взятом месте непонятно, нужна ли какая-то конкретная глобальная переменная, или нет).
5. Глобальные переменные приводят к трудно уловимым ошибкам.



Нежелательное изменение значения в другом месте,



Ошибочное использование глобальной переменной для промежуточных вычислений из-за совпадения имен,



Возвращение функцией неправильного значения при тех же параметрах (оказывается, она зависима от глобальной переменной, а ее кто-то поменял).

6. Глобальные переменные увеличивают число прямых и косвенных связей в системе, делая ее поведение трудно предсказуемым, а её саму - сложной для понимания.

Описание целых констант `enum`

Константы имеющие целое значение описываются следующим образом.

```
enum { N = 100 };
```

На самом деле это — определение **перечислимого типа**, содержащего единственную константу N, имеющую значение 100.

Константы, объявленные таким образом, подчиняются правилам видимости идентификаторов языка Си.

Описание констант через #define

Хороший код от плохого отличается в том числе тем, что в нем отсутствуют непонятные константы в середине исходного файла. Для создания констант часто применяется директива `#define`. Общая форма записи при этом выглядит следующим образом:

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

Это удобно, когда одно и то же значение используется множество раз в одном и том же файле.

<pre>// 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++)</pre>	<pre>// Плохо unsigned int a, h; char ln[25], f[25]; for(int i=0;i<25;i++)</pre>
---	---

Сравнение const, #define и enum

В C++ предпочтительней использовать `const` и `enum` использованию `#define`, поскольку `#define` зачастую вообще не относят к языку C++. Это правило лучше было бы назвать «Компилятор предпочтительнее препроцессора».

`const int var = 5;` C++ стиль

`#define var 5` macro - assembler стиль

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

Как выбрать имя переменной

Необходимо помнить, что программа в первую очередь предназначена для прочтения человеком и лишь потом для исполнения на компьютере. Даже если код написан правильно, с точки зрения исполнения и проходит трансляцию, но при этом написан безграмотно с точки зрения стилистики, то такой код нельзя считать корректным.

- Правильно выбирайте имена для переменных, имя переменной должно отражать суть того для чего данная переменная используется.
- Не бойтесь составлять имена из нескольких слов. Лучше английских.

- Избегайте использования однобуквенных переменных, если только это не сугубо локальные переменные или их применение ограничено несколькими строчками.
- Если переменная используется во всей программе, то она должна состоять из нескольких букв или даже слов.
- Переменные и идентификаторы пишутся в camelCase.
- Константы пишутся в SCREAMING_CASE.
Это важное разграничение, позволяющее с первого взгляда определять, какие переменные неизменяемые и по своей природе используются только для чтения.

<pre>// Хорошо enum { NAME_LENGTH = 25 }; unsigned int age, height; char lastName[NAME_LENGTH]; char firstName[NAME_LENGTH];</pre>	<pre>// Плохо unsigned int a, h; char ln[25], f[25];</pre>
--	--

Если вы все же решились их использовать, то добавьте ей префикс, например: g_




<code>int myVar = 3; // плохо</code>	<code>int g_myVar = 3; // уже лучше</code>
--------------------------------------	--

Вопрос. 🙄 Какой наилучший префикс для глобальной переменной?

Ответ: //




Форматирование кода для цикла

Рассмотрим форматирование кода на примере цикла while. Правильное форматирование показано на примере 1. Открывающуюся скобку допустимо оставить на уровне заголовка цикла, но желательно переносить на новую строку. Закрывающуюся операторную скобку всегда необходимо переносить на новую строку (пример 2). Отступ скобок должен быть на уровне заголовка цикла.

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

В примере 2 и 5 неправильное форматирование закрывающей фигурной скобки. В примере 3 неверное форматирование открывающейся фигурной скобки.

Если телом цикла должен стать пустой оператор, лучше использовать пустые операторные скобки, пример 4. Недопустимо ставить точку с запятой в одну строку, пример 6.

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

Вычисление корней квадратного уравнения (продолжение)

Давайте добавим в нашу программу систему меню.

Оператор goto позволяет нам не делать сложных условий для проверки прихода символа перевода строки "0xa" и не печатать меню повторно

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

```

Подведение итогов

Итак, на этой лекции мы рассмотрели такие управляющие конструкции как [оператор выбора switch](#) и [циклы](#).

Изучили различные типы циклов: [циклы с условием while](#) и их [особенности](#), [цикл с постусловием do...while](#) и [цикл с известным числом шагов for](#) его [особенности](#).

[Операторы continue](#) и [break](#) для изменения логики работы циклов

Научились [заменять цикл for на while](#)

Рассмотрели [оператор перехода goto](#) его особенности и область его применения.

Изучили такие немаловажные вопросы как: [область видимости переменных](#), [описание целых констант](#) через `enum` и [через #define](#), а также [как выбрать имя переменной](#).

Заострили внимание на [форматировании кода для цикла](#).

Продолжили писать [вычисление корней квадратного уравнения](#), добавив систему меню

Дополнительные материалы

1. Оформление программного кода
<http://www.stolyarov.info/books/pdf/codestyle2.pdf>

2. Стандарт разработки программного обеспечения MISRA на языке C
http://easyelectronics.ru/files/Book/misra_c_rus.pdf
3. <http://cppstudio.com/post/6458/> Циклы for, while и do while в языке C
4. <http://cppstudio.com/post/6691/> Оператор выбора switch в C(Си)
5. <https://metanit.com/cpp/tutorial/2.17.php> Конструкция switch-case
6. <https://metanit.com/c/tutorial/2.12.php> Циклы
7. [https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB_\(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5\)](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) Цикл (программирование)
8. <https://www.cyberforum.ru/cpp-beginners/thread1497526.html> Циклы while и for: в чем суть отличия?
9. <https://learn.c.info/c/loop.html> Введение. Циклы с предусловием.
10. <https://cppprosto.blogspot.com/2017/09/break-continue-return-goto.html> Язык Си. Циклы. Операторы перехода (break, continue, return, goto)
11. <https://pas1.ru/goto> Операторы goto, break, continue и прекращения программы
12. <https://www.bestprog.net/ru/2019/11/17/c-statements-break-continue-goto-examples-of-use-ru/#contents> Операторы break, continue, goto. Примеры применения
13. https://professorweb.ru/my/csharp/charp_theory/level3/3_17.php Операторы перехода

Используемые источники

1. cprogramming.com - учебники по C и C++
2. free-programming-books - ресурс содержащий множество книг и статей по программированию, в том числе по C и C++ (там же можно найти ссылку на распространяемую бесплатно автором html-версию книги Eckel B. «Thinking in C++»)
3. tutorialspoint.com - ещё один ресурс с множеством руководств по изучению различных языков и технологий, в том числе содержит учебники по C

4. Юричев Д. [«Заметки о языке программирования Си/Си++»](#) - «для тех, кто хочет освежить свои знания по Си/Си++»
5. [Онлайн версия «Си для встраиваемых систем»](#)
6. [Руководство по Си](#) METANIT.COM <https://metanit.com/c/tutorial/2.12.php> Циклы
7. <https://habr.com/ru/post/114211/> goto