



Массивы, циклы

и

операторы принятия решений

Презентация разработана в рамках гранта «Грант на обучение студентов по образовательным программам высшего образования для топ-специалистов в сфере информационных технологий. Договор № 70-2025-000850 с АНО «Аналитический центр при Правительстве Российской Федерации»

Александра Волосова,
к.т.н., доцент кафедры
ИУ5

План лекции

1. Операторы принятия решений: if, else, switch
2. Тернарный оператор
3. Циклы: for, while, do-while
4. Управление циклами: break, continue
5. Массивы: объявление, инициализация, доступ
6. Многомерные массивы
7. Range-based for loops (C++11)
8. Алгоритмы обработки массивов
9. Структурное программирование
10. Лучшие практики

1. Оператор if

```
// Базовый синтаксис:
if (condition) {
    // выполняется если condition == true
}

// Примеры:
int score = 85;

if (score >= 90) {
    std::cout << "Отлично!" << std::endl;
}

if (temperature > 30) {
    std::cout << "Жарко!" << std::endl;
    turn_on_air_conditioner();
}

// Условное выражение:
• Должно возвращать bool или преобразовываться к bool
• Любое ненулевое значение = true
• Ноль = false

// Рекомендация MIT:
Всегда используйте фигурные скобки, даже для одного оператора
```

2. Оператор if-else

```
// Синтаксис:
if (condition) {
    // выполняется если condition == true
} else {
    // выполняется если condition == false}

// Пример:
int number = 7;

if (number % 2 == 0) {
    std::cout << "Четное число" << std::endl;
} else {
    std::cout << "Нечетное число" << std::endl;}

// Цепочка else-if:
int score = 85;

if (score >= 90) {
    std::cout << "Оценка: A" << std::endl;
} else if (score >= 80) {
    std::cout << "Оценка: B" << std::endl;
} else if (score >= 70) {
    std::cout << "Оценка: C" << std::endl;
} else {
    std::cout << "Оценка: F" << std::endl;}

// Вложенные if:
if (is_connected) {
    if (has_permission) {
        download_file();}}}
```

3. Оператор switch

```
// Синтаксис для дискретных значений:
switch (variable) {
    case value1:
        // код для value1
        break;
    case value2:
        // код для value2
        break;
    default:
        // код если ни один case не подошел}
// Пример:
char grade = 'B';

switch (grade) {
    case 'A':
        std::cout << "Отлично!" << std::endl;
        break;
    case 'B':
        std::cout << "Хорошо!" << std::endl;
        break;
    case 'C':
        std::cout << "Удовлетворительно" << std::endl;
        break;
    default:
        std::cout << "Неизвестная оценка" << std::endl;}

// Важно: break предотвращает "проваливание"
// switch работает с: char, int, enum (C++11: и с std::string)
```

4. Тернарный оператор

```
// Синтаксис: condition ? expression1 : expression2
// Возвращает expression1 если true, иначе expression2

// Примеры:
int a = 5, b = 10;
int max = (a > b) ? a : b; // max = 10

int number = 7;
std::string parity = (number % 2 == 0) ? "четное" : "нечетное";

// Эквивалентно:
int max;
if (a > b) {
    max = a;
} else {
    max = b;
}

// Вложенные тернарные операторы (осторожно!):
int x = 5, y = 10, z = 15;
int largest = (x > y) ? ((x > z) ? x : z) : ((y > z) ? y : z);

// Рекомендация MIT:
Используйте для простых условий, избегайте вложенных

// Modern C++: constexpr тернарный оператор
constexpr int size = (DEBUG_MODE) ? 100 : 1000;
```

5. Цикл for

```
// Синтаксис:
for (инициализация; условие; инкремент) {
    // тело цикла
}

// Классический пример:
for (int i = 0; i < 10; i++) {
    std::cout << i << " "; // 0 1 2 3 4 5 6 7 8 9
}

// Несколько переменных:
for (int i = 0, j = 10; i < j; i++, j--) {
    std::cout << "i=" << i << ", j=" << j << std::endl;
}

// Бесконечный цикл:
for (;;) {
    // требует break для выхода
    if (condition) break;
}

// Область видимости:
for (int i = 0; i < 10; i++) {
    // i видна только внутри цикла
}
// i не видна здесь

// Рекомендация: используйте size_t для индексов
for (size_t i = 0; i < array_size; i++)
```

6. Цикл while

// Синтаксис:

```
while (condition) {  
    // тело цикла}
```

// Пример:

```
int count = 0;  
while (count < 5) {  
    std::cout << "Count: " << count << std::endl;  
    count++;}
```

// Чтение до конца ввода:

```
int number;  
while (std::cin >> number) {  
    std::cout << "Прочитано: " << number << std::endl;}
```

// Обработка событий:

```
bool running = true;  
while (running) {  
    process_events();  
    if (should_quit()) {  
        running = false;}}
```

// Особенности:

- Условие проверяется ПЕРЕД каждой итерацией
- Может не выполниться ни разу
- Подходит когда количество итераций неизвестно

7. Цикл do-while

// Синтаксис:

```
do {  
    // тело цикла  
} while (condition);
```

// Пример:

```
int choice;  
do {  
    std::cout << "Выберите опцию (1-3): ";  
    std::cin >> choice;  
} while (choice < 1 || choice > 3);
```

// Особенности:

- Тело выполняется ХОТЯ БЫ один раз
- Условие проверяется ПОСЛЕ каждой итерации
- Точка с запятой обязательна

// Практическое применение:

- Меню и пользовательский ввод
- Повторение операций до успеха
- Инициализация с последующей проверкой

// Сравнение с while:

```
int x = 10;
```

```
while (x < 5) {    // Не выполнится ни разу  
    std::cout << x;}
```

```
do {  
    // Выполнится один раз  
    std::cout << x; // Выведет 10} while (x < 5);
```

8. break и continue

```
// break: немедленный выход из цикла или switch
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        break; // выход при i == 5
    }
    std::cout << i << " "; // 0 1 2 3 4
}
```

```
// continue: переход к следующей итерации
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        continue; // пропустить четные числа
    }
    std::cout << i << " "; // 1 3 5 7 9
}
```

```
// Метки для вложенных циклов (редко):
outer_loop:
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (i * j == 4) {
            break outer_loop; // выход из обоих циклов
        }
    }
}
```

// Рекомендация:

Используйте break и continue умеренно, предпочитайте ясные условия

9. Массивы в C++

```
// Объявление массива:  
// type name[size];  
  
int numbers[5];           // Массив из 5 int  
double temperatures[24];  // Массив из 24 double  
  
// Инициализация:  
int primes[5] = {2, 3, 5, 7, 11}; // Явная инициализация  
int values[10] = {};             // Все элементы = 0  
int auto_size[] = {1, 2, 3};     // Размер = 3 (автоматически)  
  
// Доступ к элементам:  
primes[0] = 2;    // Первый элемент (индекс 0)  
primes[4] = 11;   // Последний элемент (индекс size-1)  
  
// Размер массива:  
int size = sizeof(primes) / sizeof(primes[0]); // 5  
  
// Важно:  
• Индексы от 0 до size-1  
• Выход за границы - неопределенное поведение  
• Массивы статического размера (известного на этапе компиляции)
```

10. Работа с массивами

// Заполнение массива:

```
const int SIZE = 5;
int arr[SIZE];
for (int i = 0; i < SIZE; i++) {
    arr[i] = i * 2; // 0, 2, 4, 6, 8}
```

// Поиск элемента:

```
int target = 4;
bool found = false;
for (int i = 0; i < SIZE; i++) {
    if (arr[i] == target) {
        found = true;
        break;}}
```

// Вычисление суммы:

```
int sum = 0;
for (int i = 0; i < SIZE; i++) {
    sum += arr[i];
}
```

// Поиск максимума:

```
int max = arr[0];
for (int i = 1; i < SIZE; i++) {
    if (arr[i] > max) {
        max = arr[i];}}
```

// Копирование массива:

```
int copy[SIZE];
for (int i = 0; i < SIZE; i++) {
    copy[i] = arr[i];}
```

11. Многомерные массивы

```
// Двумерный массив (матрица):  
// type name[rows][columns];  
  
int matrix[3][4]; // 3 строки, 4 столбца  
  
// Инициализация:  
int grid[2][3] = {  
    {1, 2, 3},      // Первая строка  
    {4, 5, 6}       // Вторая строка};  
  
// Доступ к элементам:  
matrix[0][0] = 1; // Верхний левый угол  
matrix[2][3] = 12; // Нижний правый угол  
  
// Обход матрицы:  
const int ROWS = 3;  
const int COLS = 4;  
  
for (int i = 0; i < ROWS; i++) {  
    for (int j = 0; j < COLS; j++) {  
        std::cout << matrix[i][j] << " ";  
    }  
    std::cout << std::endl;}  
  
// Трехмерный массив:  
int cube[2][3][4]; // 2 слоя, 3 строки, 4 столбца  
  
// Применение: изображения, игры, научные вычисления
```

12. Range-based for loop (C++11)

```
// Синтаксис: for (element : collection)
// Автоматический обход элементов

// С массивами:
int numbers[] = {1, 2, 3, 4, 5};

for (int num : numbers) {
    std::cout << num << " "; // 1 2 3 4 5}

// С ссылками (для модификации):
for (int& num : numbers) {
    num *= 2; // Удваиваем каждый элемент}

// С константными ссылками (эффективно):
for (const int& num : numbers) {
    std::cout << num << " "; // 2 4 6 8 10}

// С auto (современный стиль):
for (auto& num : numbers) {
    num += 1; // Увеличиваем на 1}

// Преимущества:


- Более читаемый код
- Автоматическое определение размера
- Исключает ошибки индексов



// Ограничение: не работает с указателями на массивы
```

13. Алгоритмы обработки массивов

// Линейный поиск:

```
int linear_search(int arr[], int size, int target) {  
    for (int i = 0; i < size; i++) {  
        if (arr[i] == target) {  
            return i; // Найден}}  
    return -1; // Не найден}
```

// Сортировка пузырьком:

```
void bubble_sort(int arr[], int size) {  
    for (int i = 0; i < size - 1; i++) {  
        for (int j = 0; j < size - i - 1; j++) {  
            if (arr[j] > arr[j + 1]) {  
                std::swap(arr[j], arr[j + 1]);}}}
```

// Реверс массива:

```
void reverse_array(int arr[], int size) {  
    for (int i = 0; i < size / 2; i++) {  
        std::swap(arr[i], arr[size - i - 1]);  
    }}
```

// Подсчет частот:

```
void frequency_count(int arr[], int size) {  
    const int MAX_VALUE = 100;  
    int freq[MAX_VALUE + 1] = {0};  
  
    for (int i = 0; i < size; i++) {  
        if (arr[i] >= 0 && arr[i] <= MAX_VALUE) {  
            freq[arr[i]]++;}}
```

14. Принципы структурного программирования

Три базовые структуры (теорема Бёма-Якопини):

1. Последовательность:

Выполнение операторов по порядку

```
{  
    statement1;  
    statement2;  
    statement3; }
```

2. Ветвление:

Выбор между альтернативами

```
if (condition) {  
    // true branch  
} else {  
    // false branch }
```

3. Цикл:

Повторение while условие истинно

```
while (condition) {  
    // loop body }
```

Запрещено:

- goto (в большинстве случаев)
- Неструктурированные переходы

Преимущества:

- Легче читать и понимать
- Легче тестировать и отлаживать
- Меньше ошибок
- Лучшая поддерживаемость

Правило : Одна точка входа и одна точка выхода у функций

15. Лучшие практики: Принятие решений

// Избегайте глубокой вложенности:

// ПЛОХО:

```
if (condition1) {  
    if (condition2) {  
        if (condition3) {  
            // сложный код}}  
}
```

// ХОРОШО:

```
if (!condition1) return;  
if (!condition2) return;  
if (!condition3) return;  
// сложный код
```

// Ранний возврат (early return):

```
bool is_valid_input(int value) {  
    if (value < 0) return false;  
    if (value > 100) return false;  
    if (value % 2 != 0) return false;  
    return true;}  
  
// Switch с enum:
```

```
enum class Color { RED, GREEN, BLUE };
```

```
void handle_color(Color color) {  
    switch (color) {  
        case Color::RED:    /* обработка */  
            break;  
        case Color::GREEN: /* обработка */  
            break;  
        case Color::BLUE:   /* обработка */  
            break;}}
```

// Константы вместо магических чисел:

```
const int MAX_RETRIES = 3;  
if (attempts < MAX_RETRIES) { }
```

16. Лучшие практики: Циклы

// Правильные имена переменных:

```
for (int student_index = 0; student_index < student_count; student_index++) {  
    // ХОРОШО: понятное имя  
}
```

```
for (int i = 0; i < n; i++) { // ПЛОХО: непонятные имена}
```

// Предвычисление границ цикла:

```
const size_t size = vector.size();  
for (size_t i = 0; i < size; i++) { // ХОРОШО}
```

```
for (size_t i = 0; i < vector.size(); i++) { // ПЛОХО: size() вызывается каждый раз}
```

// Итерация по контейнерам:

```
for (const auto& element : container) { // ХОРОШО}
```

// Избегайте изменять счетчик цикла в теле:

```
for (int i = 0; i < 10; i++) {  
    if (condition) {  
        i++; // ПЛОХО: сложно отслеживать }  
}
```

17. Лучшие практики: Массивы

```
// Используйте константы для размеров:
const int MAX_STUDENTS = 100;
int grades[MAX_STUDENTS];

// Проверка границ:
void safe_array_access(int arr[], int size, int index) {
    if (index >= 0 && index < size) {
        return arr[index];
    } else {
        throw std::out_of_range("Index out of bounds");}}

// Инициализация массивов:
int values[100] = {0}; // Все элементы = 0
int data[5] = {};      // Все элементы = 0

// Современная альтернатива - std::array:
#include <array>
std::array<int, 5> modern_array = {1, 2, 3, 4, 5};

// Преимущества std::array:


- Знает свой размер (modern_array.size())
- Проверка границ с at()
- Совместимость с STL алгоритмами
- Безопаснее обычных массивов



// Для динамических массивов - std::vector:
#include <vector>
std::vector<int> dynamic_array;
dynamic_array.push_back(10); // Автоматическое расширение
```

18. Распространенные ошибки

1. Выход за границы массива:

```
int arr[5];  
arr[5] = 10; // Неопределенное поведение
```

2. Бесконечные циклы:

```
while (true) { // Нет условия выхода  
    // забыли break или изменение условия}
```

3. Путаница = и ==:

```
if (x = 5) { // Присваивание вместо сравнения  
    // всегда true}
```

4. Пропущенный break в switch:

```
switch (value) {  
    case 1: do_something(); // Проваливается в case 2!  
    case 2: do_another(); break;}
```

5. Неинициализированные переменные в циклах:

```
int sum;  
for (int i = 0; i < 10; i++) {  
    sum += i; // sum не инициализирована}
```

19. Отладка и диагностика

// Вывод отладочной информации:

```
for (int i = 0; i < size; i++) {  
    std::cout << "arr[" << i << "] = " << arr[i] << std::endl;  
}
```

// Условная компиляция для отладки:

```
#define DEBUG 1
```

```
#if DEBUG
```

```
    std::cout << "Debug: i=" << i << ", value=" << value << std::endl;  
#endif
```

// Точки останова в циклах:

```
for (int i = 0; i < large_number; i++) {  
    if (i == breakpoint_index) {  
        // Установите точку останова здесь  
    }  
    // основной код  
}
```

// Валидация входных данных:

```
if (index < 0 || index >= array_size) {  
    std::cerr << "Invalid index: " << index << std::endl;  
    return ERROR_INVALID_INDEX;  
}
```

20. Практические примеры

// Калькулятор статистики:

```
void calculate_statistics(const int scores[], int count) {
    if (count == 0) return;

    int sum = 0;
    int min = scores[0];
    int max = scores[0];

    for (int i = 0; i < count; i++) {
        sum += scores[i];
        if (scores[i] < min) min = scores[i];
        if (scores[i] > max) max = scores[i];
    }

    double average = static_cast<double>(sum) / count;
    std::cout << "Average: " << average << ", Range: [" << min << ", " << max << "]" <<
    std::endl;
}
```

// Поиск простых чисел:

```
void find_primes(int limit) {
    for (int num = 2; num <= limit; num++) {
        bool is_prime = true;
        for (int div = 2; div * div <= num; div++) {
            if (num % div == 0) {
                is_prime = false;
                break;} }
        if (is_prime) {
            std::cout << num << " ";}}
}
```

21. Современные возможности C++

```
// std::array (C++11):
#include <array>
std::array<int, 5> arr = {1, 2, 3, 4, 5};
for (auto elem : arr) { /* ... */ }

// std::vector (динамический массив):
#include <vector>
std::vector<int> vec = {1, 2, 3};
vec.push_back(4); // Автоматическое расширение

// Algorithm library:
#include <algorithm>
#include <numeric>

std::vector<int> numbers = {5, 2, 8, 1, 9};
std::sort(numbers.begin(), numbers.end()); // Сортировка
auto it = std::find(numbers.begin(), numbers.end(), 8); // Поиск
int sum = std::accumulate(numbers.begin(), numbers.end(), 0); // Сумма

// Structured bindings (C++17):
std::array<int, 3> values = {1, 2, 3};
auto [a, b, c] = values; // a=1, b=2, c=3

// Range-based for c initializer (C++20):
for (std::array arr = {1, 2, 3}; auto elem : arr) {
    std::cout << elem << " ";
}
```

22. Вопросы для самопроверки

1. В чем разница между `while` и `do-while`?
2. Когда использовать `switch` вместо `if-else`?
3. Как избежать выхода за границы массива?
4. В чем преимущество `range-based for`?
5. Как работает короткое замыкание в логических операторах?
6. Когда использовать `break` и `continue`?
7. Как определить размер массива?
8. В чем разница между `std::array` и обычным массивом?
9. Как обработать пользовательский ввод с проверкой ошибок?
10. Какие принципы структурного программирования вы знаете?

Практические задания:

1. Реализуйте сортировку массива
2. Напишите программу для поиска в матрице
3. Создайте меню с обработкой пользовательского ввода
4. Реализуйте алгоритм для подсчета статистики
5. Напишите тесты для проверки граничных условий

23. Заключение и ключевые выводы

- Массивы, циклы и операторы решений - фундамент программирования
- Правильный выбор структур управления критически важен для читаемости
- Понимание границ массивов предотвращает ошибки
- Современный C++ предлагает безопасные альтернативы (`std::array`, range-based for)
- Структурное программирование улучшает maintainability кода
- Тестирование граничных условий обязательно

Ресурсы для изучения

Книги:

- "Язык программирования C++" - Бьярн Страуструп
- "Эффективный современный C++" - Скотт Мейерс
- "Учебник по C++" - Стэнли Липпман

Онлайн ресурсы:

- cppreference.com - полная информация
- learncpp.com - учебник для начинающих
- isocpp.org - официальный сайт C++

Статьи и руководства:

- **C++ Core Guidelines**
- Документация Microsoft C++