

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

Институт информатики, математики и робототехники

Отчёт по лабораторной работе №1 по дисциплине
«Архитектура вычислительных систем и
компьютерных сетей»

Приложение для получения адресных параметров сети

Выполнили:

Студенты гр. ПРО-341Б

Репин Д.А.

Репин И.А.

Проверил:

Валеев Р.С.

Содержание

1	Введение	3
2	Ход работы	4
2.1	Алгоритм работы	4
2.2	Демонстрация работы программы	4
2.3	Код программы	6
2.3.1	Доменный слой	6
2.3.2	Слой приложения	10
2.3.3	Слой представления	12
2.3.4	Функция main	21
3	Выводы	22

1 Введение

Цель работы

Изучить устройство IP-адресации, масок подсетей и их расчёт, разработать программу для определения адресных параметров сети по диапазону IP-адресов.

Задачи работы

Программа должна обладать графическим интерфейсом для ввода и отображения данных.

На вход подается диапазон ip адресов (начальный адрес и конечный)

Требуется рассчитать и вывести 4 параметра:

1. Адрес сети
2. Broadcast адрес
3. MAC-адрес узла сети
4. Маску сети

2 Ход работы

2.1 Алгоритм работы

1. Перевод IP адресов в байты
2. Проверка правильности порядка границ диапазона
3. Расчет сетевой маски, на вход подаем начало и конец диапазона в байтах
4. Расчет сетевого адреса с помощью первого адреса и сетевой маски в байтах
5. Расчет широковещательного адреса с помощью сетевой маски и сетевого адреса в байтах
6. Получение MAC-адресов устройств на компьютере

Функция расчета сетевой маски - Используя побитовые операции, определяется префикс начального и конечного IP адресов

Функция расчета сетевого адреса - Применяется побитовое И между IP-адресом и маской сети.

Функция расчета широковещательного адреса - Выполняется побитовое отрицание маски сети и объединение с сетевым адресом.

Функция получения MAC адресов - Извлекает MAC-адреса всех доступных сетевых интерфейсов.

2.2 Демонстрация работы программы

Ввод	Результаты
Диапазон IP-адресов	Параметры сети
Начало: 192.168.1.1	Network Address: 192.168.0.0
Конец: 192.168.3.3	Broadcast Address: 192.168.3.255
	Network Mask: 255.255.252.0
Расчитать Сброс	MAC-address: Bluetooth Device (Personal Area Network) — 208-197-211-126-68-66 TAP-Windows Adapter V9 — 0-255-32-85-99-35 Microsoft Wi-Fi Direct Virtual Adapter #2 — 208-197-211-126-68-67 Microsoft Wi-Fi Direct Virtual Adapter — 210-197-211-126-68-67 Realtek RTL8723DE 802.11b/g/n PCIe Adapter — 208-197-211-126-68-67 Realtek PCIe GbE Family Controller — 244-57-9-234-109-125

Рис. 1: Результат при 192.168.1.1 — 192.168.3.3

Ввод		Результаты
Диапазон IP-адресов		Параметры сети
Начало:	<input type="text" value="144.0.0.1"/>	Network Address: 128.0.0.0
Конец:	<input type="text" value="192.168.2.3"/>	Broadcast Address: 255.255.255.255
		Network Mask: 128.0.0.0
		MAC-address: Bluetooth Device (Personal Area Network) — 208-197-211-126-68-66 TAP-Windows Adapter V9 — 0-255-32-85-99-35 Microsoft Wi-Fi Direct Virtual Adapter #2 — 208-197-211-126-68-67 Microsoft Wi-Fi Direct Virtual Adapter — 210-197-211-126-68-67 Realtek RTL8723DE 802.11b/g/n PCIe Adapter — 208-197-211-126-68-67 Realtek PCIe GbE Family Controller — 244-57-9-234-109-125
<input type="button" value="Рассчитать"/> <input type="button" value="Сброс"/>		

Рис. 2: Результат при 144.0.0.1 — 192.168.2.3

Ввод		Результаты
Диапазон IP-адресов		Параметры сети
Начало:	<input type="text" value="129.0.0.1"/>	Network Address: 128.0.0.0
Конец:	<input type="text" value="134.1.2.3"/>	Broadcast Address: 135.255.255.255
		Network Mask: 248.0.0.0
		MAC-address: Bluetooth Device (Personal Area Network) — 208-197-211-126-68-66 TAP-Windows Adapter V9 — 0-255-32-85-99-35 Microsoft Wi-Fi Direct Virtual Adapter #2 — 208-197-211-126-68-67 Microsoft Wi-Fi Direct Virtual Adapter — 210-197-211-126-68-67 Realtek RTL8723DE 802.11b/g/n PCIe Adapter — 208-197-211-126-68-67 Realtek PCIe GbE Family Controller — 244-57-9-234-109-125
<input type="button" value="Рассчитать"/> <input type="button" value="Сброс"/>		

Рис. 3: Результат при 129.0.0.1 — 134.1.2.3

Ввод		Результаты
Диапазон IP-адресов		Параметры сети
Начало:	<input type="text" value="129.0.0.1.1"/>	Network Address:
Конец:	<input type="text" value="134.1.2.3"/>	Broadcast Address:
		Network Mask:
		MAC-address:
<input type="button" value="Рассчитать"/> <input type="button" value="Сброс"/>		Invalid IP address format

Рис. 4: Обработка ошибки

Ввод		Результаты
Диапазон IP-адресов		Параметры сети
Начало:	<input type="text" value="456.1.1.1"/>	Network Address:
Конец:	<input type="text" value="345.1.1.1"/>	Broadcast Address:
		Network Mask:
		MAC-address:
<input type="button" value="Рассчитать"/> <input type="button" value="Сброс"/>		IP octet out of range

Рис. 5: Обработка ошибки

Ввод		Результаты
Диапазон IP-адресов		Параметры сети
Начало:	<input type="text" value="...."/>	Network Address:
Конец:	<input type="text" value="345.1.1.1"/>	Broadcast Address:
		Network Mask:
		MAC-address:
<input type="button" value="Рассчитать"/> <input type="button" value="Сброс"/>		Wrong format!

Рис. 6: Обработка ошибки

2.3 Код программы

2.3.1 Доменный слой

```

1 #pragma once
2
3 #include <string>
4 #include <unordered_map>
5 #include <utility>
6
7 #include <winsock2.h>
8 #include <iphlpapi.h>
9 #include <array>
10 #include <stdint>
11 #include <functional>
12 #include <iostream>
13 #include <memory>
14 #include <sstream>
15 #include <stdexcept>
16 #include <string>
17
18 #pragma comment(lib, "IPHLPAPI.lib")

```

```

19
20 namespace domain {
21
22     struct IP {
23         std::string address;
24     };
25
26     struct IpRange {
27         IP begin;
28         IP end;
29     };
30
31     struct NetworkSettings {
32         std::string network_address;
33         std::string broadcast_address;
34         std::unordered_map<std::wstring, std::string> mac_addresses;
35         std::string subnet_mask;
36     };
37
38     class Network {
39     public:
40         NetworkSettings GetNetworkSettings(IpRange range) const;
41
42     private:
43         std::array<uint8_t, 4> GetNetworkAddress(const std::array<uint8_t, 4>&
44             begin_ip,
45             const std::array<uint8_t, 4>& network_mask) const;
46         std::array<uint8_t, 4> GetBroadcastAddress(const std::array<uint8_t,
47             4>& network_address,
48             const std::array<uint8_t, 4>& network_mask) const;
49         std::array<uint8_t, 4> GetNetworkMask(const std::array<uint8_t, 4>&
50             begin, const std::array<uint8_t, 4>& end) const;
51         std::unordered_map<std::wstring, std::string> GetMacAddresses() const;
52
53         std::array<uint8_t, 4> ConvertIpToBytes(const std::string& ip) const;
54     };
55 } // namespace domain
56

```

Листинг 1: Доменный слой (Заголовочный файл)

```

1 #include "network.h"
2 #include <iptypes.h>
3
4 namespace domain {
5
6     NetworkSettings Network::GetNetworkSettings(IpRange range) const {
7         NetworkSettings result{};
8
9         std::array<uint8_t, 4> begin_bytes = ConvertIpToBytes(range.begin.
10             address);
11         std::array<uint8_t, 4> end_bytes = ConvertIpToBytes(range.end.address)
12             ;
13
14         for (size_t i = 0; i < 4; ++i) {
15             if (begin_bytes[i] > end_bytes[i]) {
16                 // throw std::runtime_error("Wrong order!");
17             } else if (begin_bytes[i] == end_bytes[i]) {
18                 continue;
19             } else {
20                 break;
21             }
22         }
23     }
24

```

```

19     }
20 }
21
22     std::array<uint8_t, 4> network_mask_bytes = GetNetworkMask(begin_bytes
, end_bytes);
23
24     std::stringstream network_mask;
25     for (size_t i = 0; i < 4; ++i) {
26         network_mask << static_cast<int>(network_mask_bytes[i]) << (i < 3 ?
"." : "");
27     }
28
29     result.subnet_mask = network_mask.str();
30
31     std::array<uint8_t, 4> network_address_bytes = GetNetworkAddress(
begin_bytes, network_mask_bytes);
32
33     std::stringstream network_address;
34     for (size_t i = 0; i < 4; ++i) {
35         network_address << static_cast<int>(network_address_bytes[i]) << (i
< 3 ? "." : "");
36     }
37
38     result.network_address = network_address.str();
39
40     std::array<uint8_t, 4> broadcast_address_bytes = GetBroadcastAddress(
network_address_bytes, network_mask_bytes);
41
42     std::stringstream broadcast_address;
43     for (size_t i = 0; i < 4; ++i) {
44         broadcast_address << static_cast<int>(broadcast_address_bytes[i]) <<
(i < 3 ? "." : "");
45     }
46
47     result.broadcast_address = broadcast_address.str();
48
49     auto mac_addresses = GetMacAddresses();
50
51     result.mac_addresses = mac_addresses;
52
53     return result;
54 }
55
56 std::unordered_map<std::wstring, std::string> Network::GetMacAddresses()
57 {
58     const {
59         ULONG buf_len = 15000;
60
61         std::unique_ptr<IP_ADAPTER_ADDRESSES, std::function<void(
IP_ADAPTER_ADDRESSES*)>> addresses(
62             nullptr, [](IP_ADAPTER_ADDRESSES* ptr) { HeapFree(GetProcessHeap(), 0,
ptr); });
63
64         addresses.reset(reinterpret_cast<IP_ADAPTER_ADDRESSES*>(HeapAlloc(
GetProcessHeap(), 0, buf_len)));
65
66         DWORD adapters = GetAdaptersAddresses(AF_UNSPEC,
67             GAA_FLAG_INCLUDE_PREFIX, nullptr, addresses.get(), &buf_len);
68
69         if (adapters != NO_ERROR) {
70             throw std::runtime_error("GetAdaptersAddresses failed with error: "

```



```

68 + std::to_string(adapters));
69 }
70 std::unordered_map<std::wstring, std::string> mac_addresses;
71
72 auto current_address = addresses.get();
73
74 while (current_address) {
75     if (current_address->PhysicalAddressLength != 0) {
76         std::string mac_address;
77
78         for (unsigned int i = 0; i < current_address->
PhysicalAddressLength; i++) {
79             mac_address += std::to_string(static_cast<int>(current_address->
PhysicalAddress[i])) +
80                 (i == current_address->PhysicalAddressLength - 1 ? "" : "-");
81         }
82
83         mac_addresses[current_address->Description] = mac_address;
84     }
85
86     current_address = current_address->Next;
87 }
88
89 return mac_addresses;
90 }
91
92 std::array<uint8_t, 4> Network::GetNetworkMask(const std::array<uint8_t,
4>& begin,
93 const std::array<uint8_t, 4>& end) const {
94     std::array<uint8_t, 4> mask = {0, 0, 0, 0};
95
96     bool edge = false;
97
98     for (size_t i = 0; i < 4; ++i) {
99         for (uint8_t b = 128; b >= 1; b /= 2) {
100             if (!edge && ((begin[i] & b) == (end[i] & b))) {
101                 mask[i] |= b;
102             } else {
103                 edge = true;
104                 mask[i] &= ~b;
105             }
106         }
107     }
108
109     return mask;
110 }
111
112 std::array<uint8_t, 4> Network::GetBroadcastAddress(const std::array<
uint8_t, 4>& network_address,
113 const std::array<uint8_t, 4>& network_mask) const {
114     std::array<uint8_t, 4> broadcast_address{};
115
116     for (size_t i = 0; i < 4; ++i) {
117         broadcast_address[i] = network_address[i] | (~network_mask[i]);
118     }
119
120     return broadcast_address;
121 }
122

```

```

123     std::array<uint8_t, 4> Network::GetNetworkAddress(const std::array<
        uint8_t, 4>& begin_ip,
124     const std::array<uint8_t, 4>& network_mask) const {
125         std::array<uint8_t, 4> network_address{};
126
127         for (size_t i = 0; i < 4; ++i) {
128             network_address[i] = begin_ip[i] & network_mask[i];
129         }
130
131         return network_address;
132     }
133
134     std::array<uint8_t, 4> Network::ConvertIpToBytes(const std::string& ip)
        const {
135         std::stringstream ss(ip);
136
137         std::array<uint8_t, 4> bytes{};
138
139         for (size_t i = 0; i < 4; ++i) {
140             std::string part;
141
142             std::getline(ss, part, '.');
143
144             int value;
145             try {
146                 value = std::stoi(part);
147             } catch (const std::exception& e) {
148                 throw std::runtime_error("Wrong format!");
149             }
150
151             if (value < 0 || value > 255) {
152                 throw std::out_of_range("IP octet out of range");
153             }
154
155             bytes[i] = static_cast<uint8_t>(value);
156         }
157
158         if (ss.rdbuf()->in_avail() != 0) {
159             throw std::invalid_argument("Invalid IP address format");
160         }
161
162         return bytes;
163     }
164
165 } // namespace domain
166
167

```

Листинг 2: Доменный слой (Файл реализации)

2.3.2 Слой приложения

```

1 #pragma once
2
3 #include <iostream>
4 #include <memory>
5 #include <queue>

```

```

6 #include <string>
7 #include <utility>
8 #include <vector>
9
10 #include "domain/network.h"
11 #include "gui/window.h"
12
13 namespace app {
14
15     class Application {
16     public:
17         Application();
18         void Start();
19
20     private:
21         void Reset();
22
23     private:
24         gui::Window window_;
25         domain::Network network_;
26     };
27
28 } // namespace app

```

Листинг 3: Слой приложения (заголовочный файл)

```

1 #include "application.h"
2 #include <sstream>
3
4 namespace app {
5
6     Application::Application() : window_(ApplicationConstants::kTitle) {
7     }
8
9     void Application::Start() {
10         while (window_.IsOpen()) {
11             gui::UserChoice user_choice = window_.Tick();
12
13             if (user_choice.type == gui::RESET) {
14                 Reset();
15             } else if (user_choice.type == gui::CALCULATE) {
16                 try {
17                     auto result = network_.GetNetworkSettings({user_choice.ip_range.
18 first, user_choice.ip_range.second});
19
20                     std::wstringstream mac_result;
21
22                     for (auto& address : result.mac_addresses) {
23                         mac_result << address.first << L" - " << std::wstring(address.
24 second.begin(), address.second.end());
25                         mac_result << L"\n" << std::wstring(Labels::kMacAddress.size()
26 * 2, L' ');
27                     }
28
29                     window_.SetResult(result.network_address, result.
30 broadcast_address, mac_result.str(), result.subnet_mask);
31                 } catch (const std::exception& ex) {
32                     window_.ShowError(ex.what());
33                 }
34             }
35         }
36     }
37
38     void Application::Reset() {
39         window_.Reset();
40     }
41 } // namespace app

```

Листинг 4: Слой приложения (Файл реализации)

2.3.3 Слой представления

```

1 #pragma once
2
3 #include <optional>
4 #include <string>
5 #include <utility>
6 #include <vector>
7
8 #include "SFML/Graphics/Font.hpp"

```

```

9 #include "SFML/Graphics/RectangleShape.hpp"
10 #include "SFML/Graphics/RenderWindow.hpp"
11 #include "SFML/Graphics/Text.hpp"
12 #include "SFML/Window/Event.hpp"
13
14 #include "constants_storage.h"
15
16 namespace gui {
17
18     enum Event { NOTHING, CALCULATE, RESET };
19
20     struct UserChoice {
21         Event type = NOTHING;
22         std::pair<std::string, std::string> ip_range;
23     };
24
25     class Window {
26     public:
27         explicit Window(const std::string& title);
28
29         // Methods
30         void Init();
31         UserChoice Tick();
32         void ShowError(const std::string& error);
33         void Reset();
34
35         // Setters
36         void SetResult(std::string network_address, std::string
broadcast_address, std::wstring mac_address,
37             std::string subnet_mask);
38
39         // Predicates
40         bool IsOpen() const noexcept;
41
42     private:
43         UserChoice HandleMouseButtonPressed(sf::Vector2i cursor_position);
44         void Update();
45         void ResizeWindow(sf::Event event);
46         void Draw();
47         void Clear();
48         void UpdateLabels();
49
50         // Init
51         void InitWindow();
52         void InitLabels();
53         void InitInputFields();
54         void InitResultLabels();
55         void InitButtons();
56
57         // Predicates
58         bool IsMouseClicked(const sf::Event& event);
59         bool IsBackspacePressed(const sf::Event& event);
60         bool IsDigitInput(const sf::Event& event);
61         bool IsDotEntered(const sf::Event& event);
62
63         // Typing
64         void AddSymbol(const sf::Event& event);
65         void RemoveSymbol(const sf::Event& event);
66
67     private:

```

```

68     std::string network_address_;
69     std::string broadcast_address_;
70     std::string subnet_mask_;
71     std::wstring mac_address_;
72
73     // Flags
74     bool show_error_ = false;
75     bool calc_button_pressed_ = false;
76     bool reset_button_pressed_ = false;
77     bool focus_on_start = true;
78     bool focus_on_finish = false;
79
80     /* GUI */
81     sf::RenderWindow window_;
82     sf::Font font_;
83
84     // Labels
85     sf::Text results_zone_label_;
86     sf::Text error_label_;
87     sf::Text input_zone_label_;
88     sf::Text ip_range_label_;
89     sf::Text network_parameters_label_;
90
91     // Input fields
92     sf::Text start_ip_range_label_;
93     sf::Text finish_ip_range_label_;
94     sf::Text start_input_text_;
95     sf::Text finish_input_text_;
96     sf::RectangleShape start_input_field_;
97     sf::RectangleShape finish_input_field_;
98
99     // Results Labels
100    sf::Text network_address_result_label_;
101    sf::Text broadcast_address_result_label_;
102    sf::Text mac_address_result_label_;
103    sf::Text subnet_mask_result_label_;
104
105    // Buttons
106    sf::Text calculate_button_label_;
107    sf::Text reset_button_label_;
108 };
109
110 } // namespace gui
111
112

```

Листинг 5: Слой представления (Заголовочный файл)

```

1  #include "window.h"
2  #include <variant>
3
4  namespace gui {
5
6      Window::Window(const std::string& title)
7      : window_({ApplicationConstants::kWidth, ApplicationConstants::kHeight},
8                title) {
9          Init();
10     }
11
12     void Window::Init() {
13         InitWindow();
14     }
15
16 }

```

```

13     InitLabels();
14     InitInputFields();
15     InitResultLabels();
16     InitButtons();
17 }
18
19 void Window::InitWindow() {
20     window_.setFramerateLimit(144);
21
22     if (!font_.loadFromFile(ApplicationConstants::kPathToFont.data())) {
23         throw std::runtime_error("Failed to load font!");
24     }
25 }
26
27 void Window::InitLabels() {
28     results_zone_label_.setFont(font_);
29     results_zone_label_.setCharacterSize(30);
30     results_zone_label_.setFillColor(sf::Color::Magenta);
31     results_zone_label_.setString(Labels::kResults.data());
32
33     error_label_.setFont(font_);
34     error_label_.setCharacterSize(40);
35     error_label_.setFillColor(sf::Color::Red);
36
37     input_zone_label_.setFont(font_);
38     input_zone_label_.setCharacterSize(30);
39     input_zone_label_.setFillColor(sf::Color::Magenta);
40     input_zone_label_.setString(Labels::kInput.data());
41
42     ip_range_label_.setFont(font_);
43     ip_range_label_.setCharacterSize(25);
44     ip_range_label_.setFillColor(sf::Color::Cyan);
45     ip_range_label_.setString(Labels::kRange.data());
46
47     network_parameters_label_.setFont(font_);
48     network_parameters_label_.setCharacterSize(25);
49     network_parameters_label_.setFillColor(sf::Color::Cyan);
50     network_parameters_label_.setString(Labels::kNetworkParameters.data());
51     ;
52 }
53
54 void Window::InitInputFields() {
55     start_ip_range_label_.setFont(font_);
56     start_ip_range_label_.setCharacterSize(25);
57     start_ip_range_label_.setFillColor(sf::Color::White);
58     start_ip_range_label_.setString(Labels::kStart.data());
59
60     finish_ip_range_label_.setFont(font_);
61     finish_ip_range_label_.setCharacterSize(25);
62     finish_ip_range_label_.setFillColor(sf::Color::White);
63     finish_ip_range_label_.setString(Labels::kFinish.data());
64
65     start_input_field_.setFillColor(sf::Color::Black);
66     start_input_field_.setOutlineThickness(2);
67     start_input_field_.setOutlineColor(sf::Color::White);
68
69     finish_input_field_.setFillColor(sf::Color::Black);
70     finish_input_field_.setOutlineThickness(2);
71     finish_input_field_.setOutlineColor(sf::Color::White);

```

```

72     start_input_text_.setFont(font_);
73     start_input_text_.setCharacterSize(25);
74     start_input_text_.setFillColor(sf::Color::White);
75
76     finish_input_text_.setFont(font_);
77     finish_input_text_.setCharacterSize(25);
78     finish_input_text_.setFillColor(sf::Color::White);
79 }
80
81 void Window::InitResultLabels() {
82     network_address_result_label_.setFont(font_);
83     network_address_result_label_.setCharacterSize(25);
84     network_address_result_label_.setFillColor(sf::Color::White);
85     network_address_result_label_.setString(Labels::kNetworkAddress.data());
86
87     broadcast_address_result_label_.setFont(font_);
88     broadcast_address_result_label_.setCharacterSize(25);
89     broadcast_address_result_label_.setFillColor(sf::Color::White);
90     broadcast_address_result_label_.setString(Labels::kBroadcastAddress.
data());
91
92     mac_address_result_label_.setFont(font_);
93     mac_address_result_label_.setCharacterSize(25);
94     mac_address_result_label_.setFillColor(sf::Color::White);
95     mac_address_result_label_.setString(Labels::kMacAddress.data());
96
97     subnet_mask_result_label_.setFont(font_);
98     subnet_mask_result_label_.setCharacterSize(25);
99     subnet_mask_result_label_.setFillColor(sf::Color::White);
100    subnet_mask_result_label_.setString(Labels::kSubnetMask.data());
101 }
102
103 void Window::InitButtons() {
104     calculate_button_label_.setFont(font_);
105     calculate_button_label_.setCharacterSize(35);
106     calculate_button_label_.setFillColor(sf::Color::Black);
107     calculate_button_label_.setString(Labels::kCalculate.data());
108     calculate_button_label_.setOutlineColor(sf::Color::White);
109     calculate_button_label_.setOutlineThickness(3);
110
111     reset_button_label_.setFont(font_);
112     reset_button_label_.setCharacterSize(35);
113     reset_button_label_.setFillColor(sf::Color::Black);
114     reset_button_label_.setString(Labels::kReset.data());
115     reset_button_label_.setOutlineColor(sf::Color::White);
116     reset_button_label_.setOutlineThickness(3);
117 }
118
119 bool Window::IsOpen() const noexcept {
120     return window_.isOpen();
121 }
122
123 UserChoice Window::Tick() {
124     sf::Event event{};
125     Update();
126
127     if (window_.waitEvent(event)) {
128         if (event.type == sf::Event::Closed) {
129             window_.close();

```



```

130     } else if (event.type == sf::Event::Resized) {
131         ResizeWindow(event);
132     } else if (IsBackspacePressed(event)) {
133         RemoveSymbol(event);
134     } else if (IsDigitInput(event) || IsDotEntered(event)) {
135         AddSymbol(event);
136     } else if (IsMouseClicked(event)) {
137         return HandleMouseButtonPressed(sf::Mouse::getPosition(window_));
138     }
139 }
140
141 return {NOTHING};
142 }
143
144 void Window::RemoveSymbol(const sf::Event& event) {
145     if (focus_on_start) {
146         std::string data_str = start_input_text_.getString();
147         start_input_text_.setString(data_str.empty() ? data_str : data_str.
substr(0, data_str.size() - 1));
148     } else if (focus_on_finish) {
149         std::string data_str = finish_input_text_.getString();
150         finish_input_text_.setString(data_str.empty() ? data_str : data_str.
substr(0, data_str.size() - 1));
151     }
152 }
153
154 void Window::AddSymbol(const sf::Event& event) {
155     if (focus_on_start) {
156         start_input_text_.setString(start_input_text_.getString() +
static_cast<char>(event.text.unicode));
157     } else if (focus_on_finish) {
158         finish_input_text_.setString(finish_input_text_.getString() +
static_cast<char>(event.text.unicode));
159     }
160 }
161
162 UserChoice Window::HandleMouseButtonPressed(sf::Vector2i cursor_position
) {
163     if (calculate_button_label_.getGlobalBounds().contains(static_cast<
float>(cursor_position.x),
164 static_cast<float>(cursor_position.y))) {
165         std::string start = start_input_text_.getString().toAnsiString();
166         std::string finish = finish_input_text_.getString().toAnsiString();
167
168         calc_button_pressed_ = true;
169         focus_on_start = false;
170         focus_on_finish = false;
171
172         if (start.empty() || finish.empty()) {
173             return {NOTHING};
174         }
175
176         return {CALCULATE, {start, finish}};
177     } else if (reset_button_label_.getGlobalBounds().contains(static_cast<
float>(cursor_position.x),
178 static_cast<float>(cursor_position.y))) {
179         reset_button_pressed_ = true;
180         focus_on_start = false;
181         focus_on_finish = false;
182

```

```

183     return {RESET};
184 } else if (start_input_field_.getGlobalBounds().contains(static_cast<
float>(cursor_position.x),
185 static_cast<float>(cursor_position.y))) {
186     focus_on_start = true;
187     focus_on_finish = false;
188 } else if (finish_input_field_.getGlobalBounds().contains(static_cast<
float>(cursor_position.x),
189 static_cast<float>(cursor_position.y))) {
190     focus_on_start = false;
191     focus_on_finish = true;
192 }
193
194 return {NOTHING};
195 }
196
197 void Window::Draw() {
198     results_zone_label_.setPosition((window_.getSize().x - window_.getSize()
().x / 3) / 64 + window_.getSize().x / 3, 20);
199     error_label_.setPosition((window_.getSize().x - window_.getSize().x /
3) / 64 + window_.getSize().x / 3,
200 window_.getSize().y - 70);
201     input_zone_label_.setPosition(window_.getSize().x / 64, 20);
202     ip_range_label_.setPosition(window_.getSize().x / 12, 60);
203     network_parameters_label_.setPosition((2 * window_.getSize().x / 3) /
2 + window_.getSize().x / 3, 60);
204
205     start_ip_range_label_.setPosition(window_.getSize().x / 64, 120);
206     finish_ip_range_label_.setPosition(window_.getSize().x / 64, 180);
207     start_input_field_.setPosition(window_.getSize().x / 64 + 100, 120);
208     start_input_field_.setSize(sf::Vector2f(window_.getSize().x / 3 - 150,
30));
209     finish_input_field_.setPosition(window_.getSize().x / 64 + 100, 180);
210     finish_input_field_.setSize(sf::Vector2f(window_.getSize().x / 3 -
150, 30));
211     start_input_text_.setPosition(window_.getSize().x / 64 + 100, 120);
212     finish_input_text_.setPosition(window_.getSize().x / 64 + 100, 180);
213
214     calculate_button_label_.setPosition((window_.getSize().x / 3) / 12,
window_.getSize().y - 70);
215     reset_button_label_.setPosition((window_.getSize().x / 3) - (window_
.getSize().x / 8), window_.getSize().y - 70);
216
217     if (calc_button_pressed_) {
218         calculate_button_label_.setOutlineColor(sf::Color::Red);
219         calc_button_pressed_ = false;
220         window_.draw(calculate_button_label_);
221     } else {
222         calculate_button_label_.setOutlineColor(sf::Color::White);
223     }
224
225     if (reset_button_pressed_) {
226         reset_button_label_.setOutlineColor(sf::Color::Red);
227         reset_button_pressed_ = false;
228         window_.draw(reset_button_label_);
229     } else {
230         reset_button_label_.setOutlineColor(sf::Color::White);
231     }
232
233     network_address_result_label_.setPosition((window_.getSize().x / 3) +

```

```

20, 120);
234 broadcast_address_result_label_.setPosition((window_.getSize().x / 3)
+ 20, 180);
235 subnet_mask_result_label_.setPosition((window_.getSize().x / 3) + 20,
240);
236 mac_address_result_label_.setPosition((window_.getSize().x / 3) + 20,
300);
237
238 if (show_error_) {
239     window_.draw(error_label_);
240 }
241
242 window_.draw(input_zone_label_);
243 window_.draw(results_zone_label_);
244 window_.draw(ip_range_label_);
245 window_.draw(network_parameters_label_);
246
247 window_.draw(start_ip_range_label_);
248 window_.draw(finish_ip_range_label_);
249 window_.draw(start_input_field_);
250 window_.draw(finish_input_field_);
251 window_.draw(start_input_text_);
252 window_.draw(finish_input_text_);
253
254 window_.draw(calculate_button_label_);
255 window_.draw(reset_button_label_);
256
257 window_.draw(network_address_result_label_);
258 window_.draw(broadcast_address_result_label_);
259 window_.draw(subnet_mask_result_label_);
260 window_.draw(mac_address_result_label_);
261
262 sf::Vertex vertical_1[] = {sf::Vertex(sf::Vector2f(window_.getSize().x
/ 3, 0)),
263     sf::Vertex(sf::Vector2f(window_.getSize().x / 3, window_.getSize().y
))});
264
265 sf::Vertex horizontal_1[] = {sf::Vertex(sf::Vector2f(0, window_.
getSize().y - 110)),
266     sf::Vertex(sf::Vector2f(window_.getSize().x / 3, window_.getSize().y
- 110))});
267
268 sf::Vertex horizontal_2[] = {sf::Vertex(sf::Vector2f(0, 110)), sf::
Vertex(sf::Vector2f(window_.getSize().x, 110))});
269
270 window_.draw(vertical_1, 4, sf::Lines);
271 window_.draw(horizontal_1, 4, sf::Lines);
272 window_.draw(horizontal_2, 4, sf::Lines);
273 }
274
275 void Window::UpdateLabels() {
276     network_address_result_label_.setString(std::string(Labels::
kNetworkAddress.data()) + network_address_);
277     broadcast_address_result_label_.setString(std::string(Labels::
kBroadcastAddress.data()) + broadcast_address_);
278     subnet_mask_result_label_.setString(std::string(Labels::kSubnetMask.
data()) + subnet_mask_);
279     mac_address_result_label_.setString(std::wstring(Labels::kMacAddress.
begin(), Labels::kMacAddress.end()) +
280     mac_address_);

```

```

281 }
282
283 void Window::Update() {
284     Clear();
285     Draw();
286     UpdateLabels();
287
288     window_.display();
289 }
290
291 void Window::ResizeWindow(sf::Event event) {
292     auto new_width = event.size.width;
293     auto new_height = event.size.height;
294
295     if (new_width < ApplicationConstants::kMinWidth) {
296         new_width = ApplicationConstants::kMinWidth;
297     }
298
299     if (new_height < ApplicationConstants::kMinHeight) {
300         new_height = ApplicationConstants::kMinHeight;
301     }
302
303     window_.setView({sf::Vector2f(new_width / 2.0, new_height / 2.0), sf::
Vector2f(new_width, new_height)});
304
305     window_.setSize(sf::Vector2u(new_width, new_height));
306 }
307
308 void Window::Reset() {
309     start_input_text_.setString("");
310     finish_input_text_.setString("");
311     network_address_.clear();
312     broadcast_address_.clear();
313     mac_address_.clear();
314     subnet_mask_.clear();
315     show_error_ = false;
316     error_label_.setString("");
317 }
318
319 void Window::Clear() {
320     window_.clear(sf::Color::Black);
321 }
322
323 bool Window::IsDigitInput(const sf::Event& event) {
324     int backspace_code = 8;
325     int ascii_scope = 128;
326
327     return ((event.type == sf::Event::TextEntered) &&
(event.text.unicode < ascii_scope && event.text.unicode !=
backspace_code) &&
std::isdigit(static_cast<char>(event.text.unicode)));
329 }
330
331
332 bool Window::IsDotEntered(const sf::Event& event) {
333     int ascii_scope = 128;
334     int dot_code = 46;
335
336     return ((event.type == sf::Event::TextEntered) && (event.text.unicode
< ascii_scope && event.text.unicode == 46));
337 }

```

```

338
339 bool Window::IsBackspacePressed(const sf::Event& event) {
340     return (event.type == sf::Event::KeyPressed && event.key.code == sf::
Keyboard::Backspace);
341 }
342
343 bool Window::IsMouseClicked(const sf::Event& event) {
344     return (event.type == sf::Event::MouseButtonPressed && event.
mouseButton.button == sf::Mouse::Left);
345 }
346
347 void Window::SetResult(std::string network_address, std::string
broadcast_address, std::wstring mac_address,
348 std::string subnet_mask) {
349     show_error_ = false;
350     network_address_ = network_address;
351     broadcast_address_ = broadcast_address;
352     mac_address_ = mac_address;
353     subnet_mask_ = subnet_mask;
354 }
355
356 void Window::ShowError(const std::string& error) {
357     network_address_.clear();
358     broadcast_address_.clear();
359     mac_address_.clear();
360     subnet_mask_.clear();
361     error_label_.setString(error);
362     show_error_ = true;
363 }
364
365 } // namespace gui

```

Листинг 6: Слой представления (Файл реализации)

2.3.4 Функция main

```

1
2 #include <iostream>
3 #include "application/application.h"
4
5 int main() {
6     try {
7         app::Application app;
8         app.Start();
9     } catch (const std::exception& ex) {
10         std::cerr << ex.what() << std::endl;
11         return EXIT_FAILURE;
12     }
13
14     return EXIT_SUCCESS;
15 }
16

```

Листинг 7: Функция main

3 Выводы

В ходе выполнения лабораторной работы были изучены устройство IP-адресации, масок подсетей, методика получения параметров сети. Реализовано приложение с графическим интерфейсом, рассчитывающая адрес сети, broadcast адрес, MAC-адрес, маску сети