МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Децентралізований месенджер в експериментальному середовищі програмування Метапрог»

Завідувач випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Петров С.О.**

Студента групи ІН – 72 **Журакулов В.В.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-72 спеціальності "Інформатика" денної форми навчання Журакулова Владислава Вячеславовича.

Тема: "Децентралізований месенджер в експериментальному середовищі програмування Метапрог"

Затверджена наказом по СумДУ

№_____ от _____2021 г.

Зміст пояснювальної записки: 1) середовище програмування Метапрог: базові елементи; 2) створення бібліотеки для програми.

Дата видачі завдання "_____"_____2021 г.

Керівник випускної роботи _____ Петров С.О.

Завдання приняв до виконання _____ Журакулов В.В.

РЕФЕРАТ

Записка: 101 стор., 163 рис., 6 джерел.

Об’єкт дослідження — децентралізований чат на експериментальному графічному середовищі програмування «Метапрог».

Мета роботи — розробити децентралізований чат на експериментальному графічному середовищі програмування «Метапрог».

Методи дослідження — експеримент й наукове моделювання.

Результати — розроблено децентралізований чат на експериментальному графічному середовищі програмування «Метапрог». Упродовж розробки написана бібліотека під назвою «Техночат» з різноманітними функціями.

ДЕЦЕНТРАЛІЗОВАНИЙ МЕСЕНДЖЕР В ЕКСПЕРИМЕНТАЛЬНОМУ СЕРЕДОВИЩІ ПРОГРАМУВАННЯ МЕТАПРОГ.

ЗМІСТ

ВСТУП

Упродовж останніх чотирьох тисяч років людство активно розробляло й покращувало різні види обміну інформацією й взаємодії з нею. Так одним з найбільш розповсюджених способів взаємодії є текст що може складатися з символів, ієрогліфів й інших графічних зображень. З плином часу з'являлися нові методи запису інформації і спектр взаємодії з текстом значно збільшився. Відомо, що в давніх державах вміння читати було цінною навичкою, а особливо важливі повідомлення люди навчилися шифрувати. Більш того різні мови й варіації стилів тексту призвели до урізноманітнення текстової інформації.

У наш час необхідність в обміні інформацією, зокрема текстовою, є дуже актуальною. Однак можна помітити що людство технологічно дійшовши до миттєвого обміну інформацією хоче досягти нових висот, а саме зробити обмін текстовими повідомленнями приватним. Також з'явилися нові інструменти для роботи з інформацією.
Візьмемо програмування – процес, що надає можливість створити різноманітні програми й програмні інструменти, зокрема написати месенджер. Очевидно що для створення такої програми необхідно вибрати середу програмування. Серед океану мов програмування, компіляторів і середовищ було обране графічне середовище Метапрог через його унікальність, універсальність, перспективність, багатомовність, кросплатформовість, простоту, логічність і навіть візуалізацією кодогенерації.

Так з'явилась необхідність в такому інструменті, який зможе реалізувати приватний обмін повідомленнями в мережі інтернет. Очевидно, що прототипом такого месенджеру може стати написаний власноруч чат в надійності елементів якого можна бути впевненим знаючи всі його особливості. Постає питання, чи є вже реалізовані месенджери для обміну інформацією в світі які б в повній мірі

задовольнили визначеним потребам? Упродовж життя ознайомившись з низкою меседжерів, таких як Line, Signal, Facebook Messenger, Telegram, qTox, uTox, World Talk, Viber, WhatsApp, Discord та іншими не було знайдено такого, що б задовільнив виставленим вимогам. Нажаль, таких месенджерів не було знайдено, тому було прийнято рішення створити свій у середовищі Метапрог.

1. Дроти послідовності

Дроти послідовності мають 4 призначення:

1.1 Задання черговості виконання блоків

Початок послідовності (вхід в послідовність)- зелена стрілка вгору, кінець послідовності (вихід) – червона стрілка вниз. (рис. 1.1)

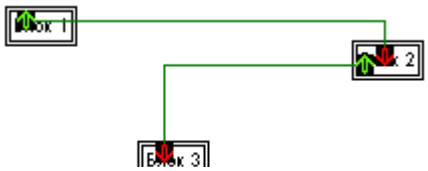

Рисунок 1.1 – Послідовності блоків

Дроти послідовності гарантують що спочатку виконається блок 1 потім 2 і тільки потім блок 3.

А от в даному випадку блоки можуть виконатися в будь якій черговості: (рис. 1.2)

Рисунок 2.2 – Блоки без послідовностей

Зазвичай першими виконуються ті блоки, які були поставлені, але в складніших випадках буває інакше.

1.2 Зараження умовою

Від блоку умови визначає те які блоки будуть виконані.

Блок умови має єдиний вхід для дроту з даними. Дроти з даними будуть описані далі, в даному випадку обмежимось тим фактом, що блок умови подано “так”. Це означає що виконається блок 1 адже на нього веде послідовність від зеленої (верхньої) половини блоку умови. Зате блок 3 не виконається адже на нього іде послідовність від червоної (нижньої) половини блоку умови. (рис. 1.3)

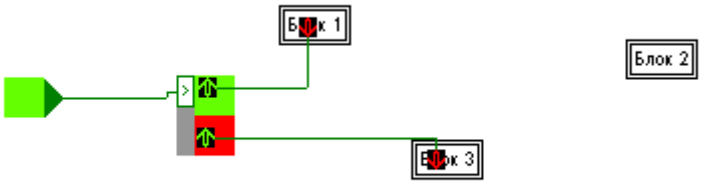

Рисунок 4.3 – Приклад логіки блоків «True»

А от в даному випадку навпаки виконається блок 3 , а блок 1 не виконається. (рис. 1.4)

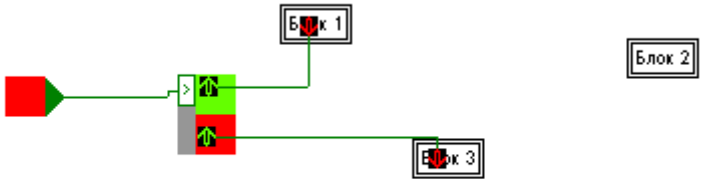

Рисунок 3.4 – Приклід логіки блоків «False»

В обох випадках виконається блок 2 адже він не заражений жодною з умов. І тут постає питання що виконається в першу чергу блоки заражені умовами чи блок 2 не заражений жодною з умов? ***Однозначної відповіді нема:*** ця схема не має принципової однозначності в цьому питанні.

А от в цій схемі вже є конкретна однозначність. (рис. 1.5)

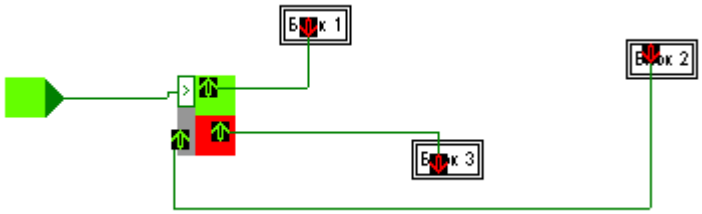

Рисунок 5.5 – Приклід логіки в обох випадках «True»

Блок 2 виконається строго після блоків, заражених умовою.

Приклад коли навпаки. (рис. 1.6)

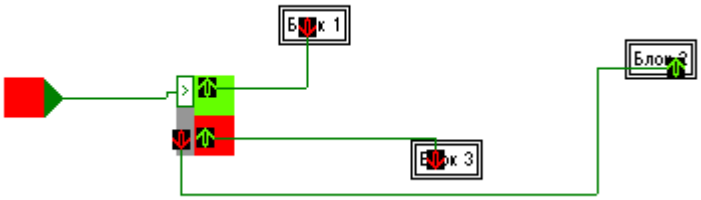

Рисунок 6.6 – Приклід логіки в обох випадках «False»

Спочатку виконається блок 2, потім виконаються(або не виконаються) блоки заражені умовою.

1.3 Забезпечення циклу

Цикл — процес, який повторюється кілька разів.

В метапрогу цикл забезпечують початковий та кінцевий блоки — квадрати зі стрілками. Вони обов’язково зв’язані дротом послідовності з кінцевого до початкового блоку. (рис. 1.7)

Рисунок 7.7 – Цикл

По дроту з даними на цикл подано число 10. Це означає, що цикл повинен повторитись десять разів. Блок 1 виконається лише один раз, тому, що він не заражений циклом

1.4 Зараження циклом

А тепер Блок 1 заражений циклом, тому він виконається у циклі, тобто 10 разів. (рис. 1.8)

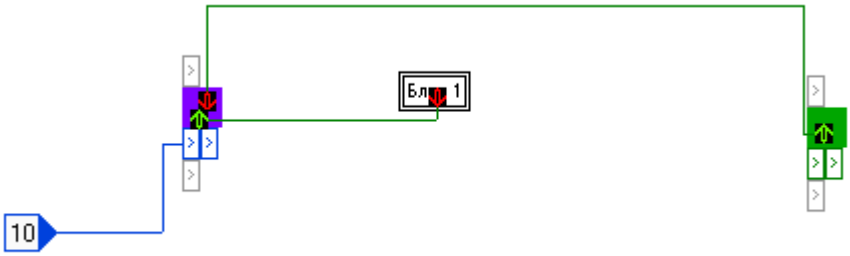

Рисунок 8.8 – Цикл з блоком

Зараження циклом відбувається від початкового (фіолетовий квадрат) блоку до блоків, які повинні виконатись у циклі.

2. Типи даних

Всі дані в метапрогу мають певний тип. Це може бути число певної розрядності, логічний тип або різні види складних типів, зібраних з елементарних числових.

2.1 Числові типи

Числові типи мають декілька видів.Числа можуть бути дробовими (з комою) та цілими (без коми).

Цілі числа можуть бути знаковими та беззнаковими. Знакові можуть мати мінус, беззнакові починаються з нуля.

Всі числові типи мають певну розрядність (кількість одиниць та нулів — бітів). Наприклад, «д128» - дробове число зі 128 бітів, «б64» - беззнакове ціле число зі 64 бітів, «з8» - знакове ціле число з 8 бітів.

Зазвичай в сучасних комп`ютерах розмір пам`яті вимірюється в байтах. Байт — одиниця інформації, що складається з 8 бітів. (рис. 1.9)

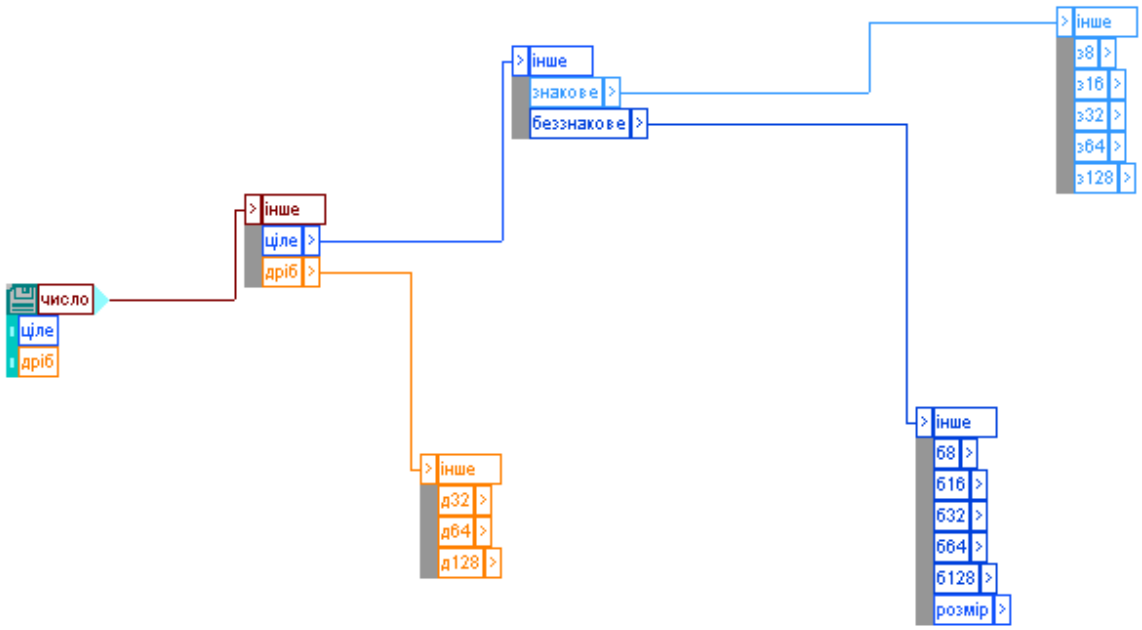

Рисунок 9.9 – Типи чисел

Як ви вже помітили, числа підсвічені різними кольорами. Дробові помаранчеві, знакові світло-сині, беззнакові темно-сині. В метапрогу типи фарбуються за кольорами.

Так виглядає константа дробового типу. (рис. 1.10)

Рисунок 10.10 – Дробове

Так виглядає константа цілого знакового типу (може мати мінус). (рис. 1.11)

Рисунок 11.11 – Ціле, знакове

Так виглядає константа цілого беззнакового типу (починається з нуля та

вище). (рис. 1.12)

Рисунок 12.12 – Ціле, беззнакове

2.2 Тип даних Логіка

Логіка — тип даних, що має значення «так» або «ні». (рис. 1.13)

Рисунок 13.13 – Блок логіки у стані «НІ»

2.3 Ускладнені типи даних

Масив

Масив — це сукупність будь-яких елементів одного типу. Це може бути масив чисел будь-якого типу, масив масивів і т.д. (рис. 1.14)

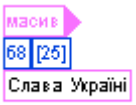

Рисунок 14.14 – Масив

В цьому контейнері з даними міститься текст «Слава Україні». Текст — це масив з байтів. Нехай вас не дивує невідповідність кількості букв та байтів в масиві(25). В кодуванні UTF — 8 кирилічні символи займають 2 байти, але

символ «пробіл» займає 1 байт.

Поіменований перелік

Виглядає поіменований перелік так. (рис. 1.15)

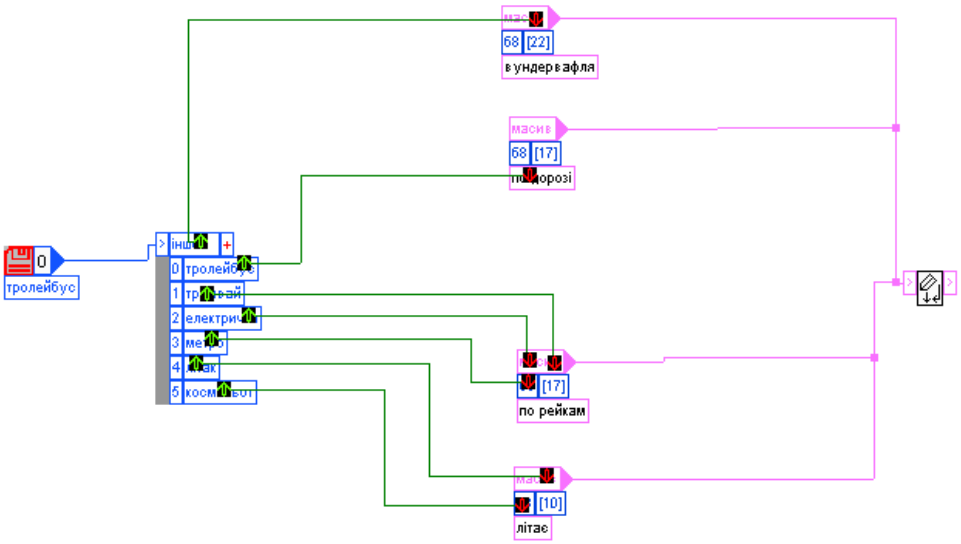

Рисунок 15.15 – Поіменований перелік

Поіменований перелік – табличний числовий тип, утворений з цілого числа будь-якої розрядності. Деяким певним числовим значенням відповідає назва. Наприклад 0 – тролейбус, 3 – метро і т.д. (рис. 1.16)

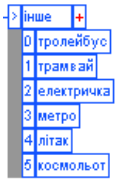

Рисунок 16.16 – Блок даних поіменованого переліку

Будь-якому числовому значенню можна надати будь-яку назву. Це має значення лише в конструкторі схем

Структура

Структура — це конструкція з кількох контейнерів з даними. Дані можуть бути різного типу. В даній структурі є масив та число типу «б128» (беззнакове ціле число зі 128 бітів). (рис. 1.17)

Рисунок 18.17 – Структура

Ця структура містить ім`я та вік. (рис. 1.18)

Рисунок 17.18 – Структура

Ця структура може містити або ім`я або вік.

Багатотиповий

Багатотиповий тип даних може містити в собі дані різних типів. В даному випадку структуру (певного типу) або масив. (рис. 1.19)

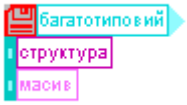

Рисунок 21.19 – Багатотипова структура

В даному випадку містить цілі або дробові числа вже згадані вище. (рис. 1.20)

Рисунок 20.20 – Структура з числами

В даному випадку різні види дробових чисел. (рис. 1.21)

Рисунок 23.21 – Структура з дробовими числами

Багатотиповий тип даних застосовуєтья там, де допускається використання кількох різних типів даних. Наприклад в функціях, які оперують числовими типами (що можуть бути різними, наприклад множенням). (рис. 1.22)

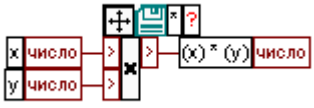

Рисунок 22.22 – Функціїя множення

Цей блок є блоком функції, яка в даному випадку виконує операцію

множення. Множити можна різні числа різних типів, тому вона приймає будь-які з них.

Структура умовного вибору типу (СУВТ)

СУВТ теж може містити дані різних типів. Відмінність від багатотипового в тому, що тип визначається під час роботи програми. Багатотиповий жорстко задається під час компіляції. Тип даних який містить СУВТ задає перемикач. Перемикачем може бути ціле число або логічний тип(так\ні). (рис. 1.23)

Рисунок 24.23 – СУВТ

Ця схема є простим прикладом застосування СУВТ. (рис. 1.24)

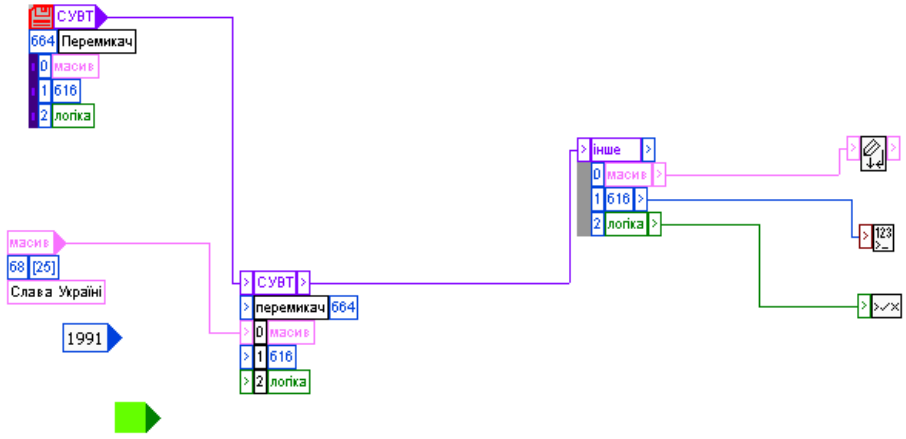

Рисунок 25.24 – СУВТ і різні типи даних

В цій половині іде виконання блоків залежно від типу даних в СУВТ. В цьому випадку друк в консоль тексту числа або логічного типу (так\ні).

Зверніть увагу що підключений лише контейнер з текстом «Слава Україні», контейнер з числом «1991» й логічним так (зеленого коліру) не підключені та, відповідно, не грають ролі в роботі схеми. (рис. 1.25)

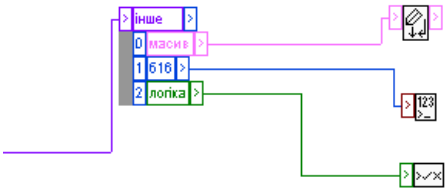

Рисунок 26.25 – Уточнююче зображення

В цьому випадку задається текст (Слава Україні). В консоль буде видано «Слава Україні». (рис. 1.26)

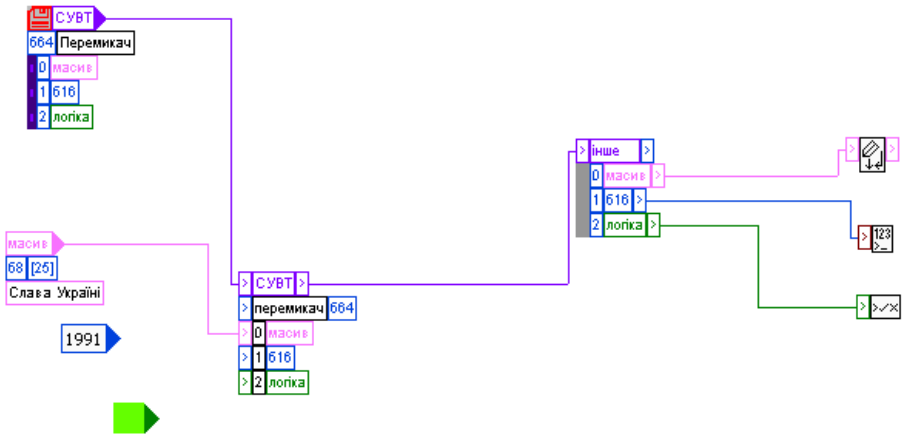

Рисунок 27.26 – Випадок з виводом тексту «Слава Україні»

В цьому випадку задається число (1991). В консоль буде видано «1991». (рис. 1.27)

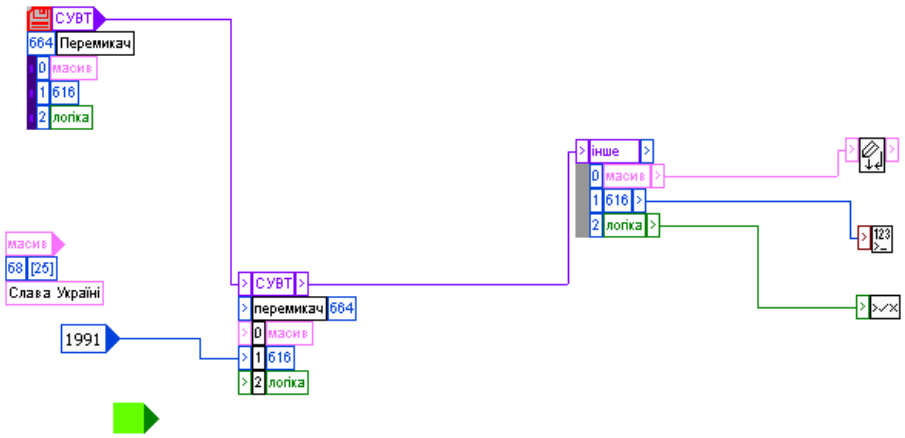

Рисунок 28.27 – Випадок з виводом тексту «1991»

В цьому випадку задається логіка (так). В консоль буде видано «так». (рис. 1.28)

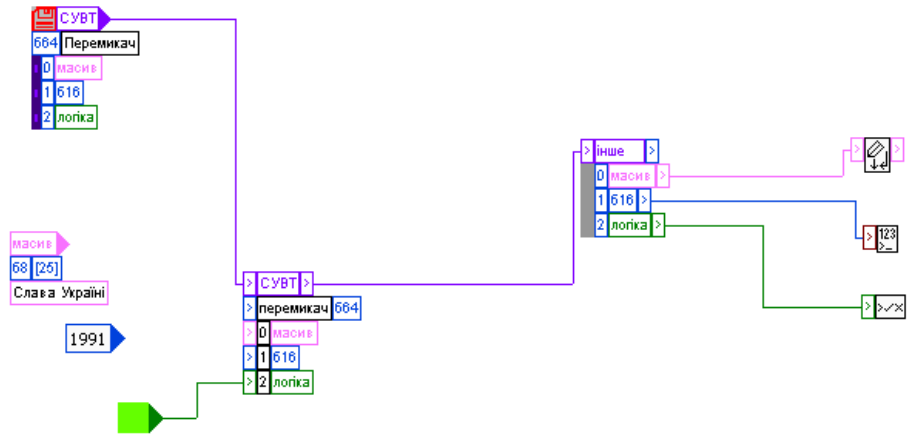

Рисунок 29.28 – Випадок з виводом тексту «так»

3. Дроти з даними

Дроти з даними несуть в собі дані певного типу. (рис. 1.29)

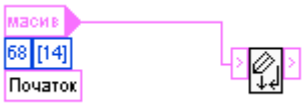

Рисунок 30.29 – Дроти з даними

Блок зі словом «Початок» є блоком константи — місце, де задані дані. В даному випадку рядок «Почати». В цьому блоці написано масив, тому, що рядок є масивом байтів. Данні з нього подаються на функцію і друкуються у консоль. Тут ми вводимо данні до масиву у лівій частині якого можна побачити набори бітів для кожного символу і самі символи тексту. (рис. 1.30)

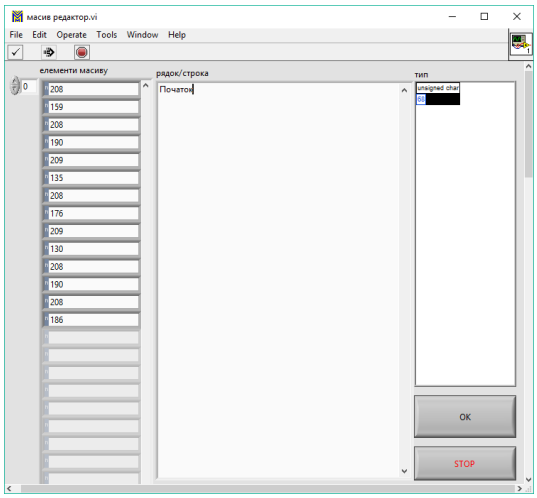

Рисунок 31.30 – Вигляд тексту й його байтів

Покажчик

В Метапрогу можна користуватись вказівниками на дані. Ці покажчики подібні до покажчиків на мові Сі і при кодогенерації перетворюються на них. Як і в Сі, в Метапрогу можливі покажчики на покажчики. У тексті Метапрогу

покажчик може бути записане словом «вказівник». (рис. 1.31)

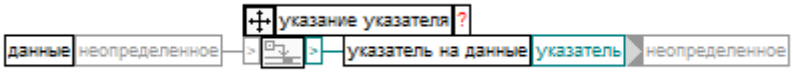

Рисунок 32.31 – Покажчик

При використанні вставок на Сі або виклику функцій на Сі (з імпортованих бібліотек) можна безпосередньо на Метапрогу працювати з вказівниками, які взаємодіють з тими функціями. Ця можливість застосовується, зокрема, в прикладах наведених далі по тексту.

4. Каст типу

Блок «каст типу» має наступний вигляд. (рис. 1.32)

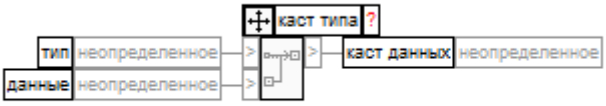

Рисунок 34.32 – Каст типу

Каст типу – базовий елемент метапрогу, що дозволяє перетворювати один тип даних в інший. (рис. 1.33)

Рисунок 33.33 – Каст типу без опису вхідних даних

Наприклад число одного типу в інший(в даному випадку число типу «д64»

в число типу «з16»). Отже на виході отримаємо число «152» з «152.564», що на вході. Таким чином дробна частину відкидується повністю без правил округлень. (рис. 1.34)

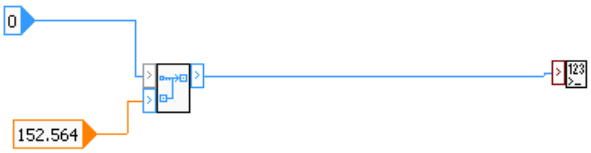

Рисунок 35.34 – Приклад роботи блоку «каст типу»

5. Компіляція

На даний момент Метапрог генерує код на С, котрий ми компілюємо й запускаємо в залежності від використаної нами операційної системи на компіляторах для мови С. (рис. 1.35, 1.36)

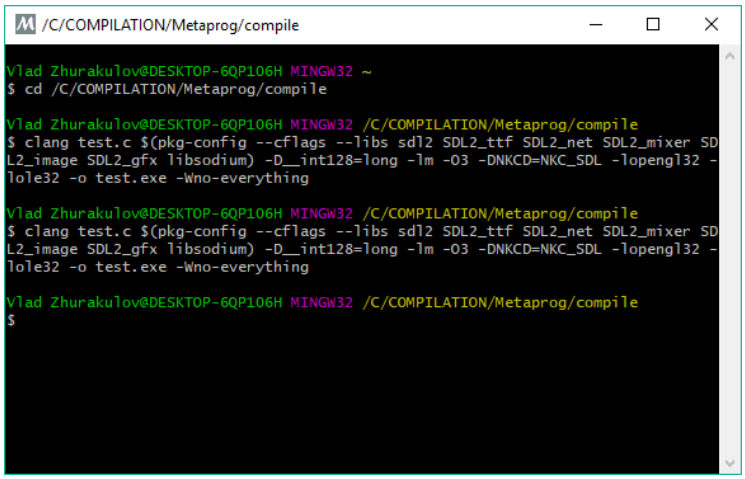

Рисунок 36.35 – Компіляція в компіляторі

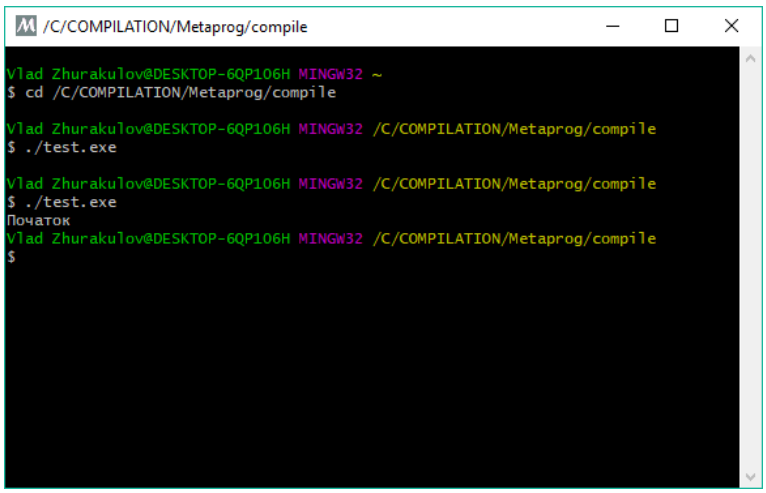

Рисунок 37.36 – Запуск test.exe файлу

Першочергово Метапрог розроблявся на операційній системі Linux, тому використовуючи його у Windows була реалізована компіляція через MSYS2 й MinGW.

6. Вставки на C

В середовищі Метапрог є можливість додавати вставки коду на мові C. Влучним прикладом такої вставки є реалізація функції зсуву бітів, що стала основою для зсуву бітів у 8, 16, 32 й більше бітних чисел. (рис. 1.37)

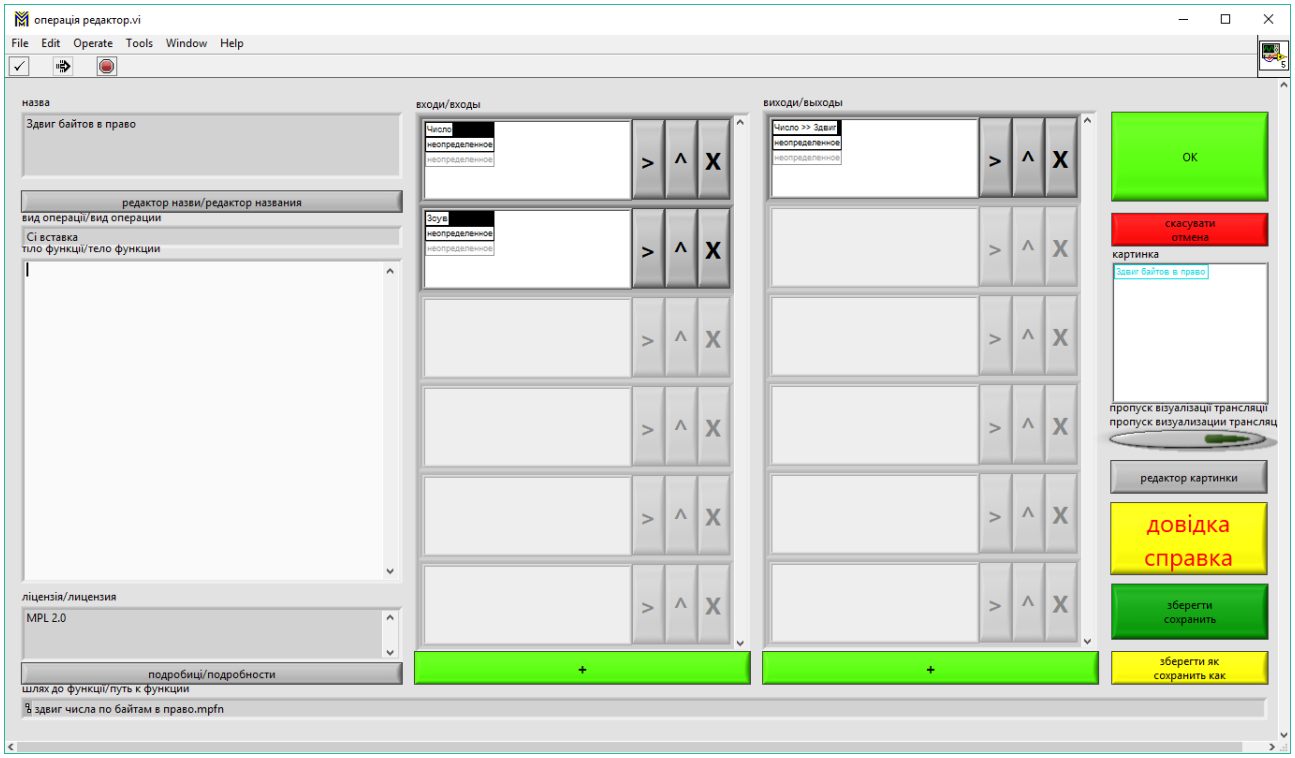

Рисунок 38.37 – Вставка коду з мови С

2 СТВОРЕННЯ БІБЛІОТЕКИ ДЛЯ ПРОГРАМИ

На початку розробки Техночату у стандартній бібліотеці Метапрогу не було достатнього числа функцій, структур й інших елементів для реалізації програми. Тому необхідно було розробити для програми свою бібліотеку. Таким чином була розроблена бібліотека Техночат у котрій було додано безліч необхідних й корисних функцій, найцікавіші з яких представлені нижче.

Слід зазначити що бібліотека Техночату вміщує в собі всі складові децентралізованого месенджеру: функції, примітиви на Сі, вставки, структури й типи. Також всі схеми й скомпільовані файли є в останніх версіях Метапрог, тому опис бібліотеки є еквівалентом опису програми.

1. Відкриття потоків

Стандартні функції для відкриття потоків існували й раніше, але в них не було семафорів для роботи з даними. Без семафорів неможливо контролювати синхронізацію потоків даних. Тому до бібліотеки Техночату були додані функції з бібліотеки SDL2, які працюють з семафорами. Ці функції були додані навіть до стандартної бібліотеки середовища Метапрог під час виконання лабораторних робіт з предмету «Обробка даних й Багатопотоковість». Функція відкриття потоку на основі SDL_CreateThread яка використовує callback (вказівники на мові С). (рис. 2.1)

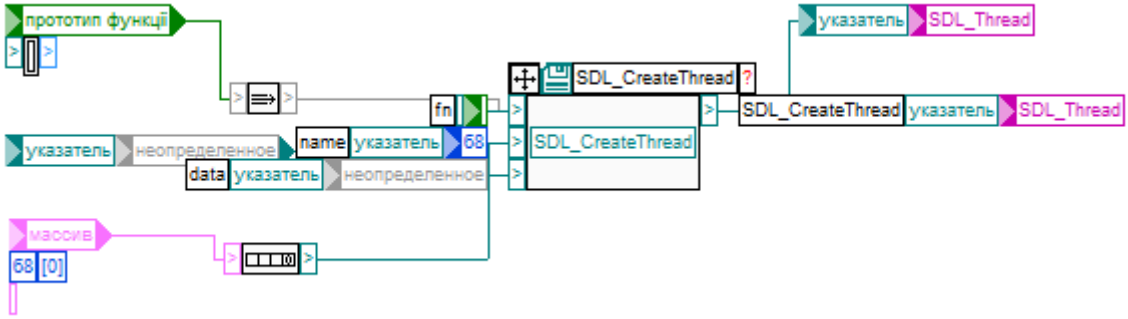

Рисунок 2.1 – Відкритя потоків

2. Семафори

Першочергово були реалізовані мьютекси, але надалі ж було вирішено повністю замінити їх на семафори. Малюнки приклади семафорів і м’ютексів й дрібних елементів багатопотоковості. (рис. 2.2 – 2.12)

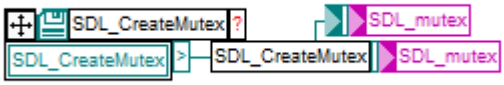

Рисунок 2.2 – Створити м’ютекс

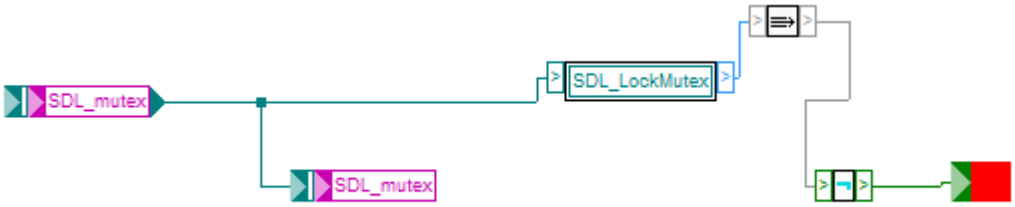

Рисунок 2.3 – Заблокувати м’ютекс

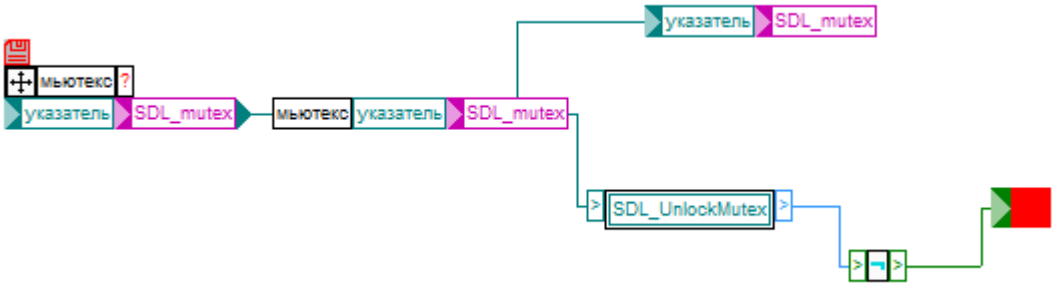

Рисунок 2.4 – Розблокувати м’ютекс

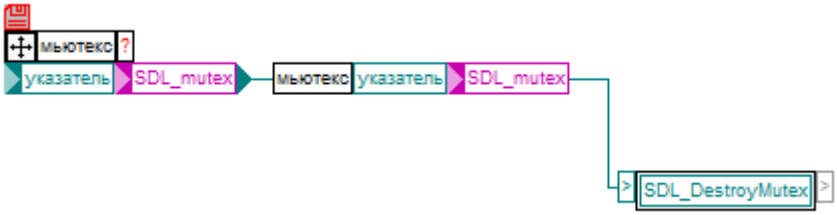

Рисунок 2.5 – Видалити м’ютекс

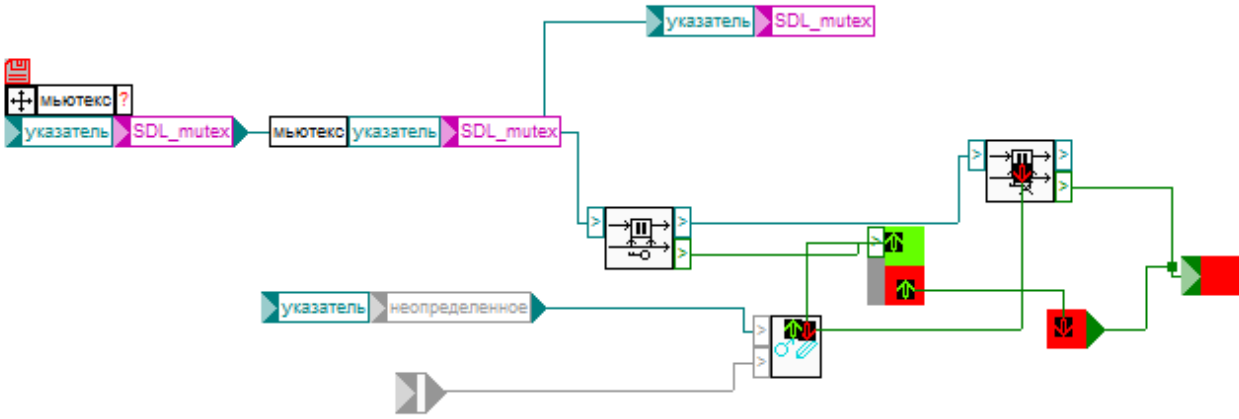

Рисунок 2.6 – Записати вказівник під мьютексом.

Рисунок 2.7 – Створити условний сигнал

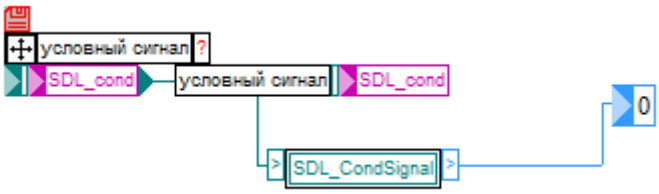

Рисунок 2.8 – Розблокувати условний сигнал.

Рисунок 2.9 – Створення семафору.

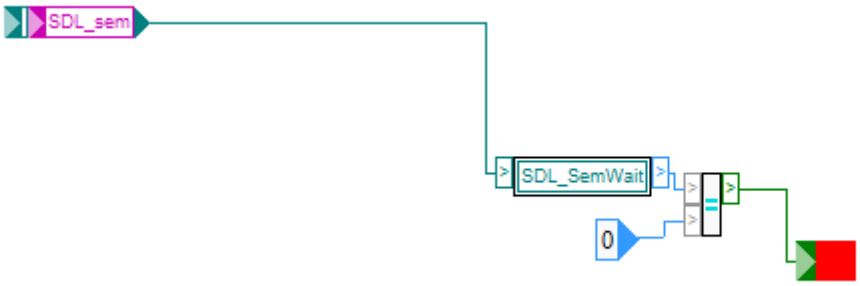

Рисунок 2.10 – Блокування семафору. семафору.

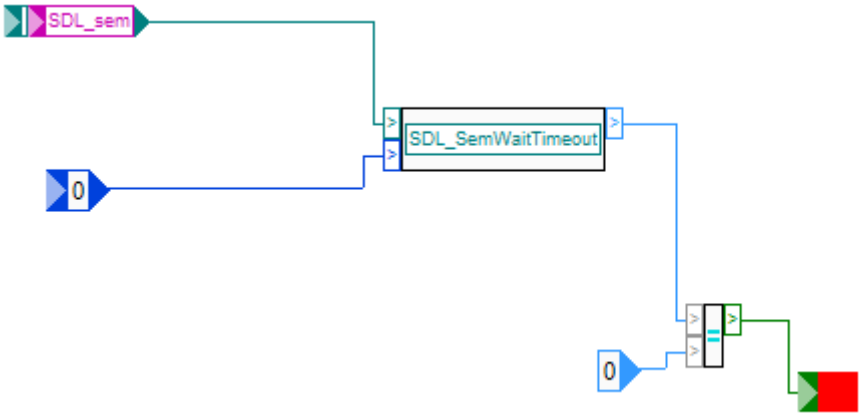

Рисунок 2.11 – Блокування семафору з очікуванням.

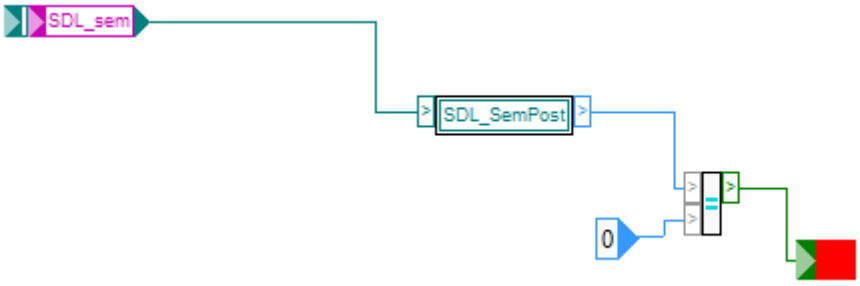

Рисунок 2.12 – Розблокування семафору.

3. Підняття TOR onion.

Дана функція досить велика, всі її складові елементи описані в назвах и описі самої функції. До неї входить складові функції поменше. На першому малюнку є функція відкриття onion-адреси в мережі TOR. (рис. 2.13 – 2.24)

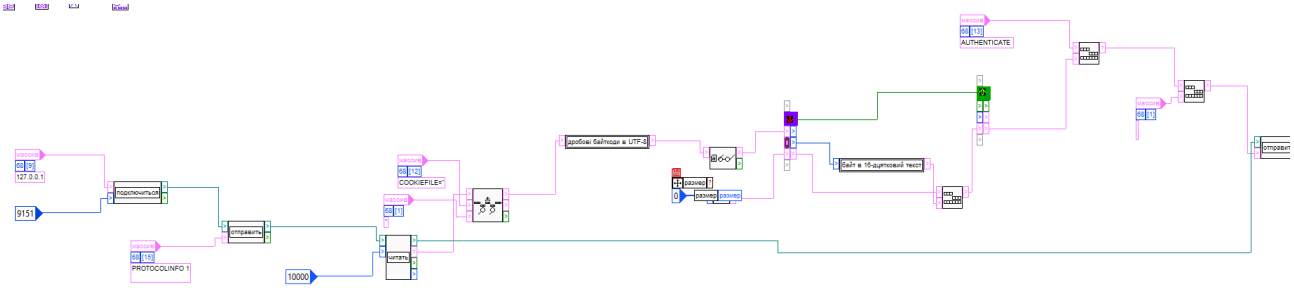

Рисунок 2.13 – Частина функції підняття TOR onion

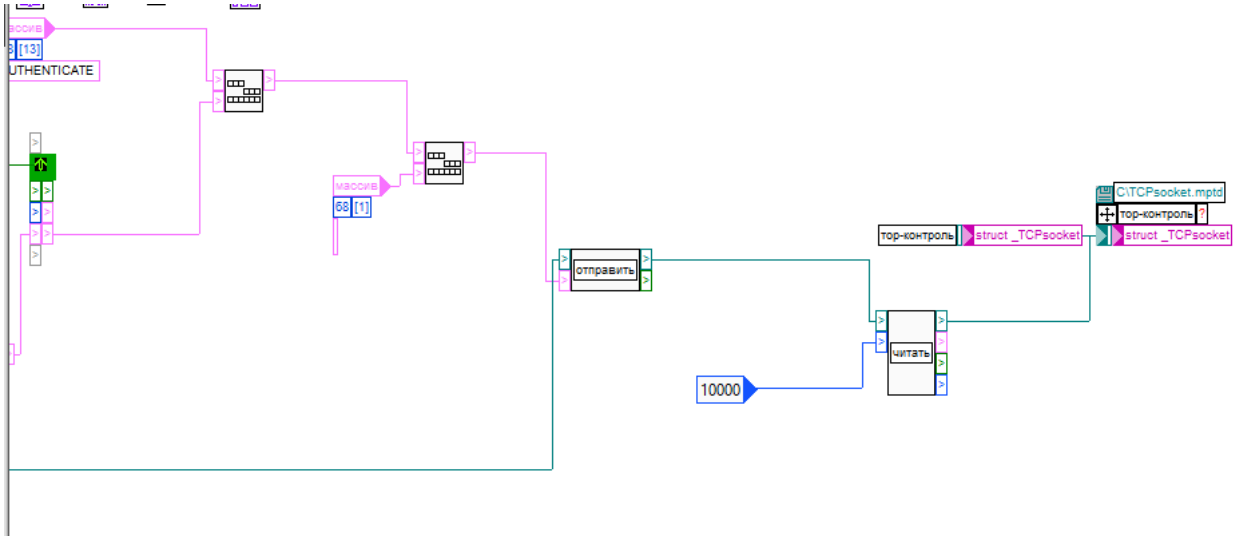

Рисунок 2.14 – Частина функції підняття TOR onion

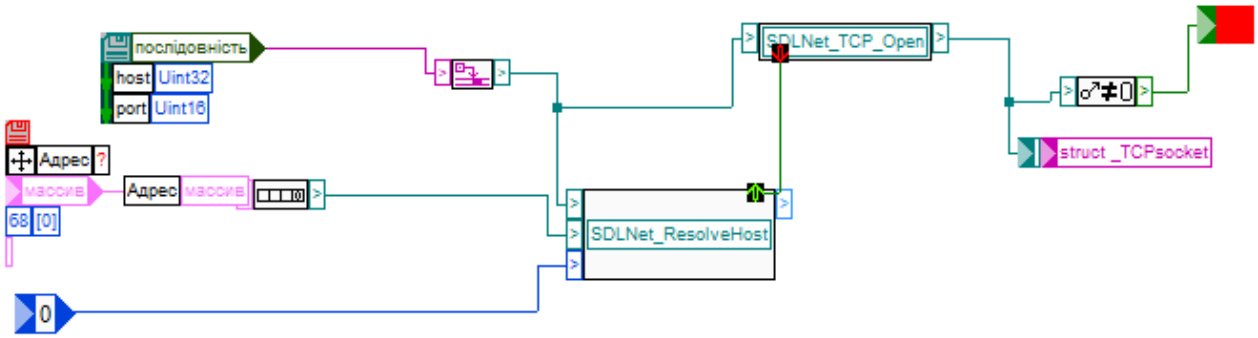

Рисунок 2.15 – Відкриття TCP-з’єднання.

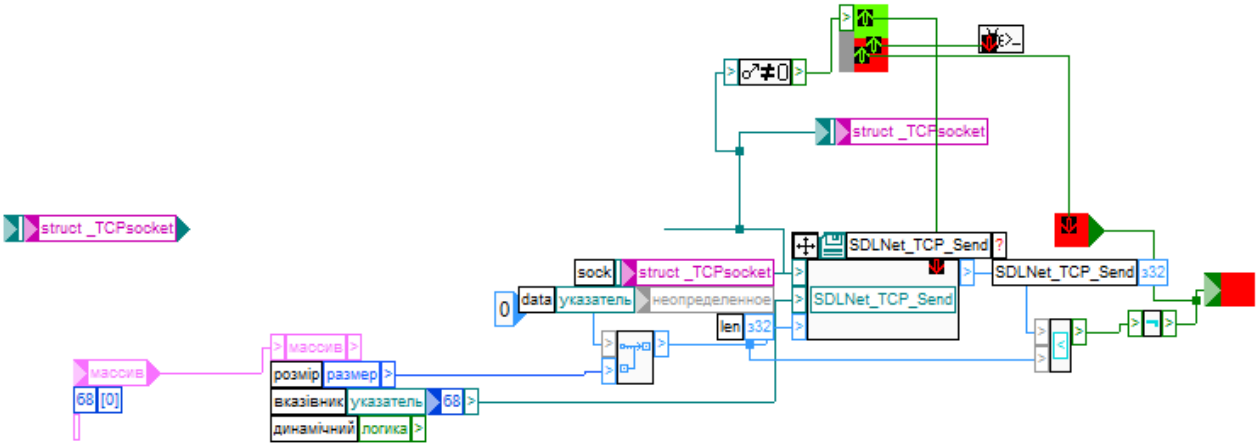

Рисунок 2.16 – Відправка TCP сокету.

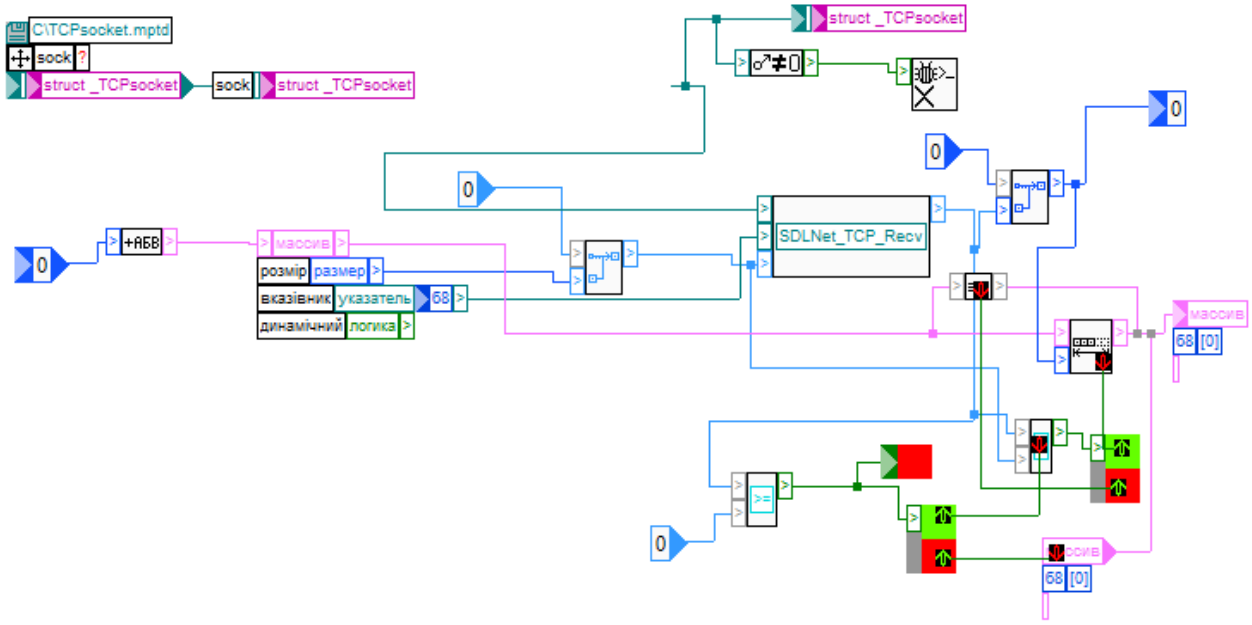

Рисунок 2.17 – Читання TCP сокету

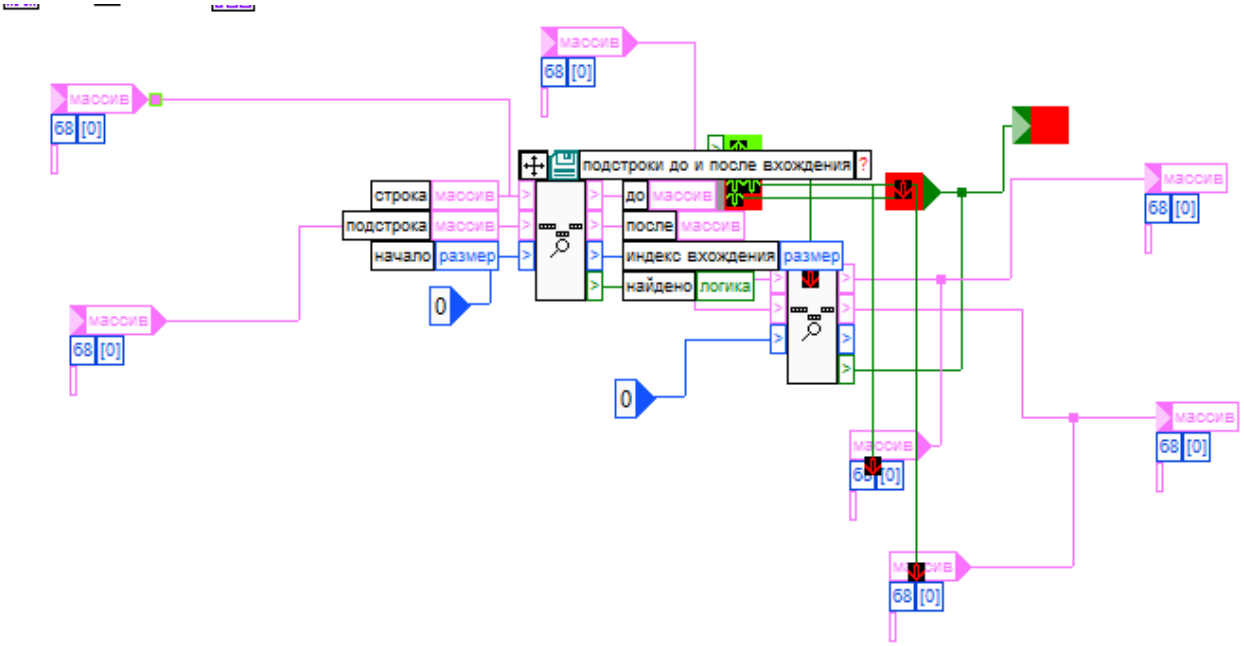

Рисунок 2.18 – Рядок між двох виразів

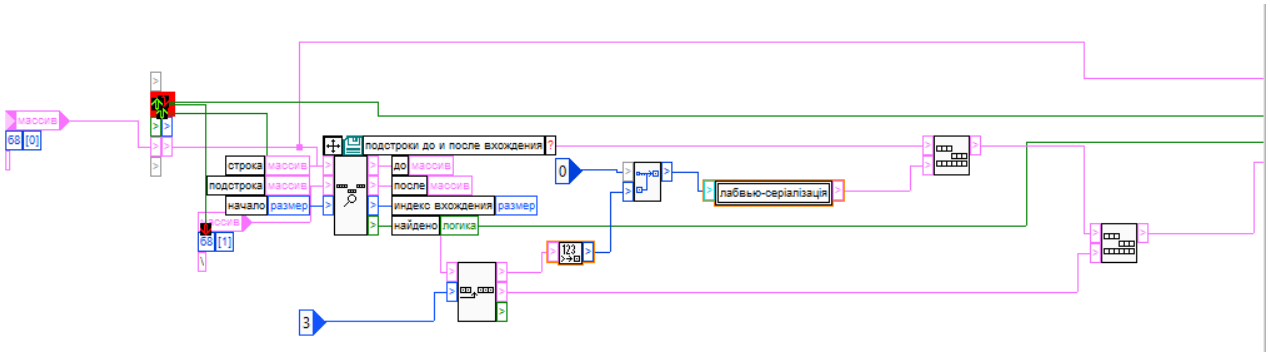

Рисунок 2.19 – Перетворення вісімкових “экранувань” (escaping) у байти

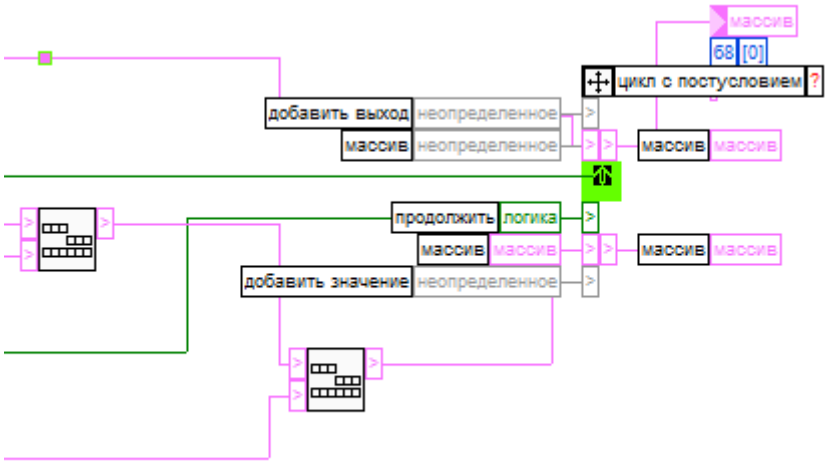

Рисунок 2.20 – Escaping друга частина схеми

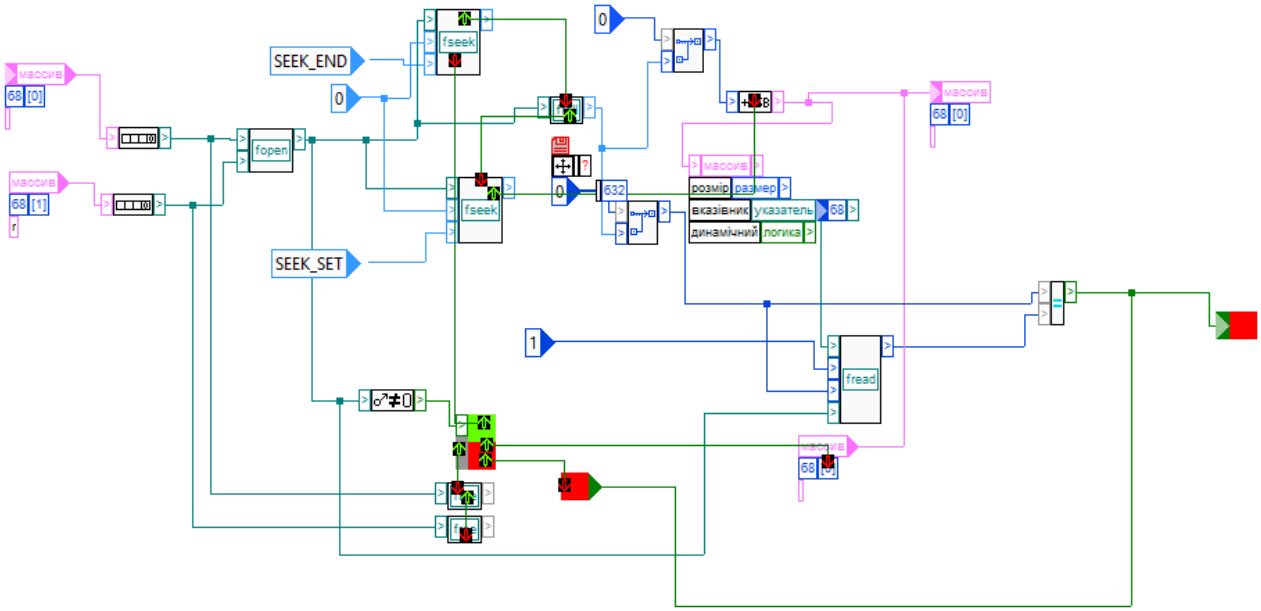

Рисунок 2.21 – Читання з файлу

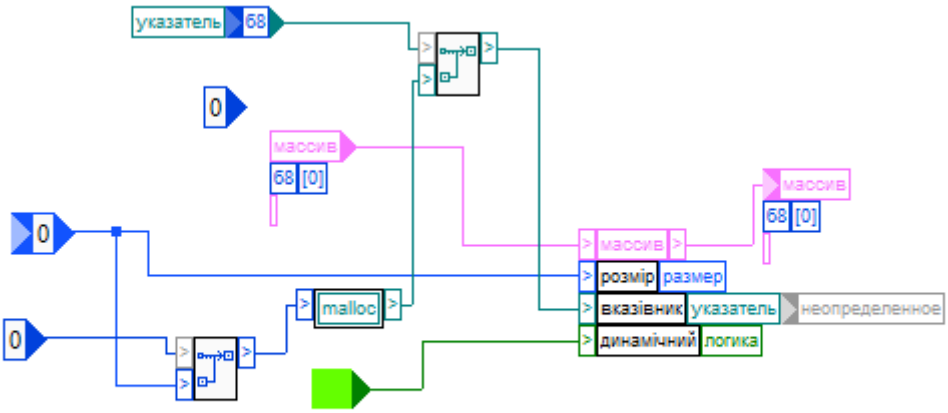

Рисунок 2.22 – Створити буфер

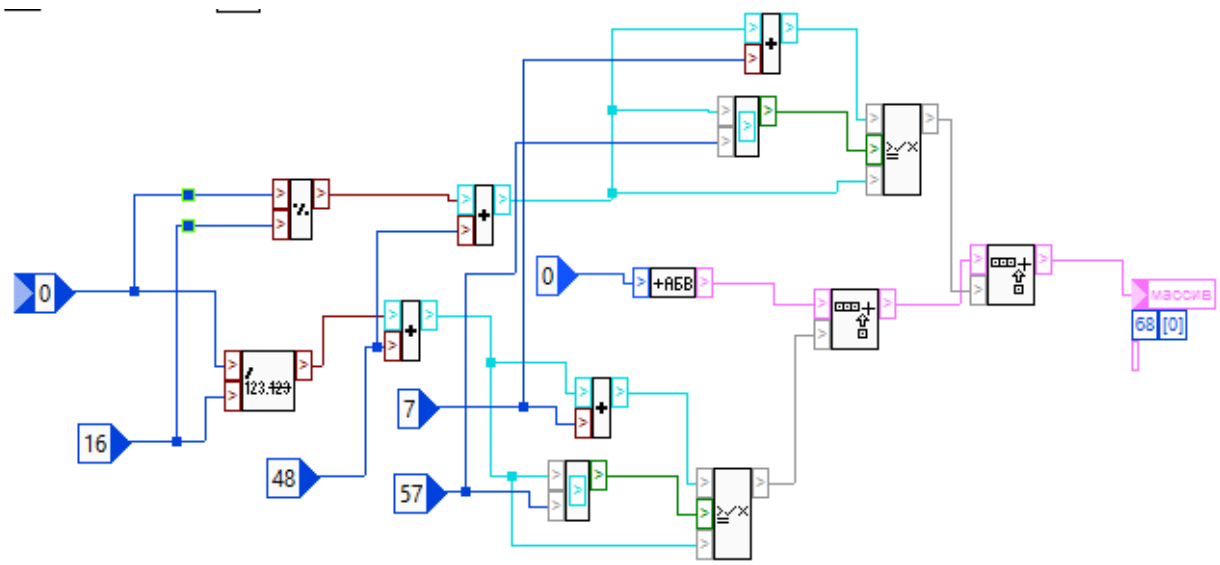

Рисунок 2.23 – Створення 16-дцяткового тексту з байтів.

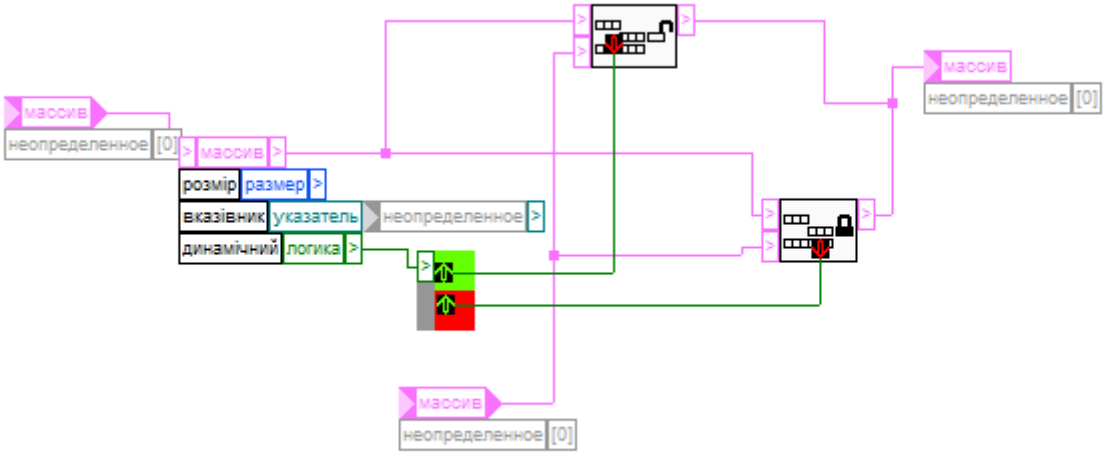

Рисунок 2.24 – Злиття масивів.

4. Робота з вікнами й «підвікнами»

Функція роботи з вікнами є необхідною частиною графічного інтерфейсу і використовує кросплатформену бібліотеку SDL2. (рис. 2.25)

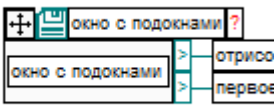

Рисунок 2.25 – Блок функції вікна з підвікнами

4.1. Вікно з підвікнами

Найбільшою функцією для роботи з вікнами є «Вікно з підвікнами». Вона виглядає так: (рис. 2.26)

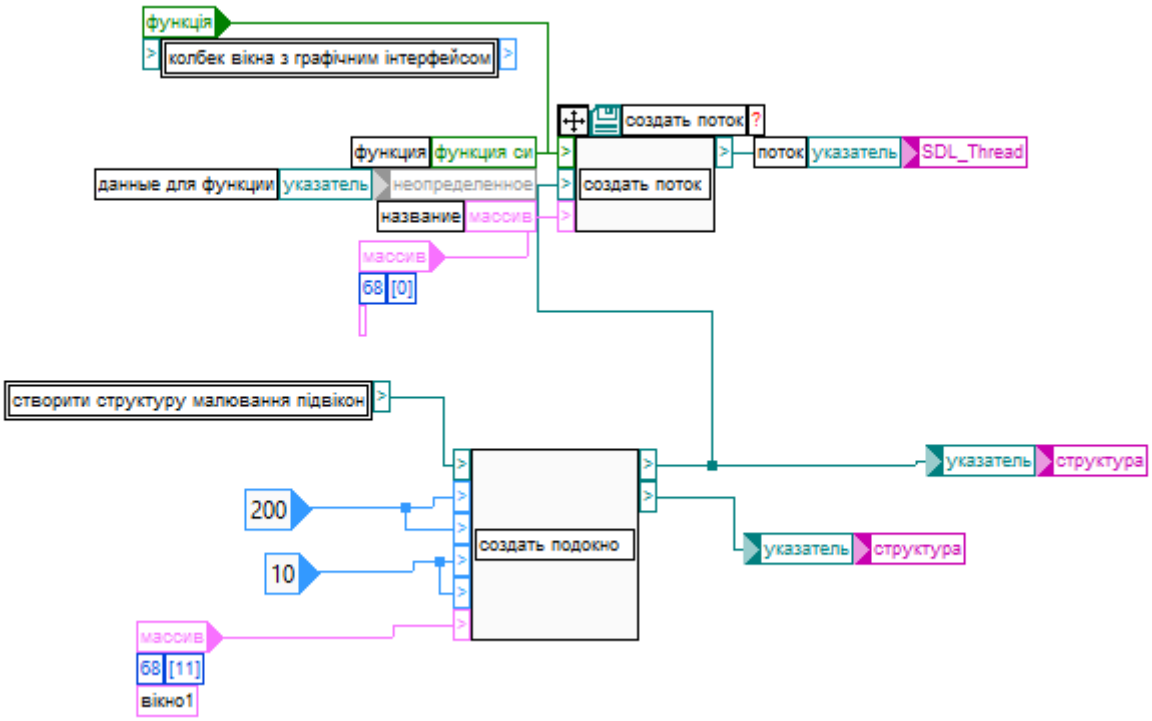

Рисунок 2.26 – Функція «Вікно з підвікнами»

4.2. Callback

Першою її складовою частиною є “callback” (виклик функції Сі з вказівника) вікна з графічним інтерфейсом. (рис. 2.27)

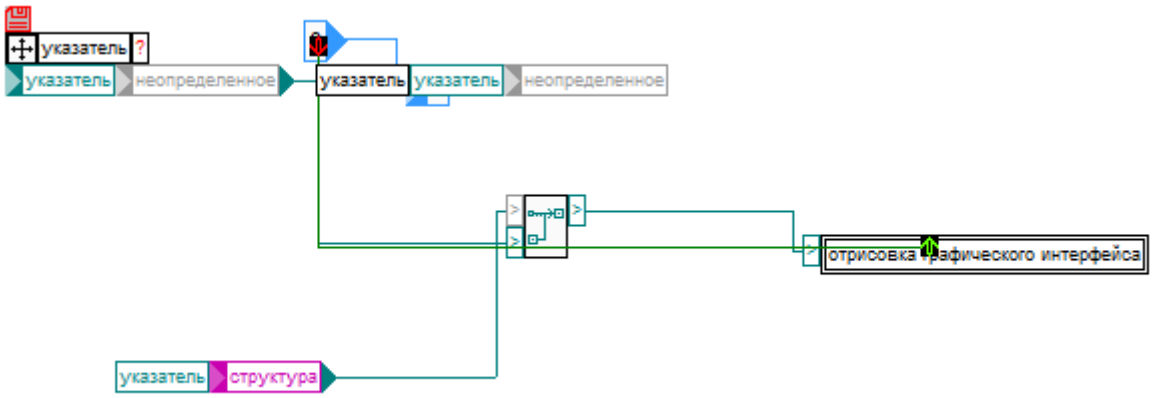

Рисунок 2.27 – Callback

4.3. Малювання графічного інтерфейсу

Графічний інтерфейс вміщує в себе функцію малювання графічного інтерфейсу. (рис. 2.28)

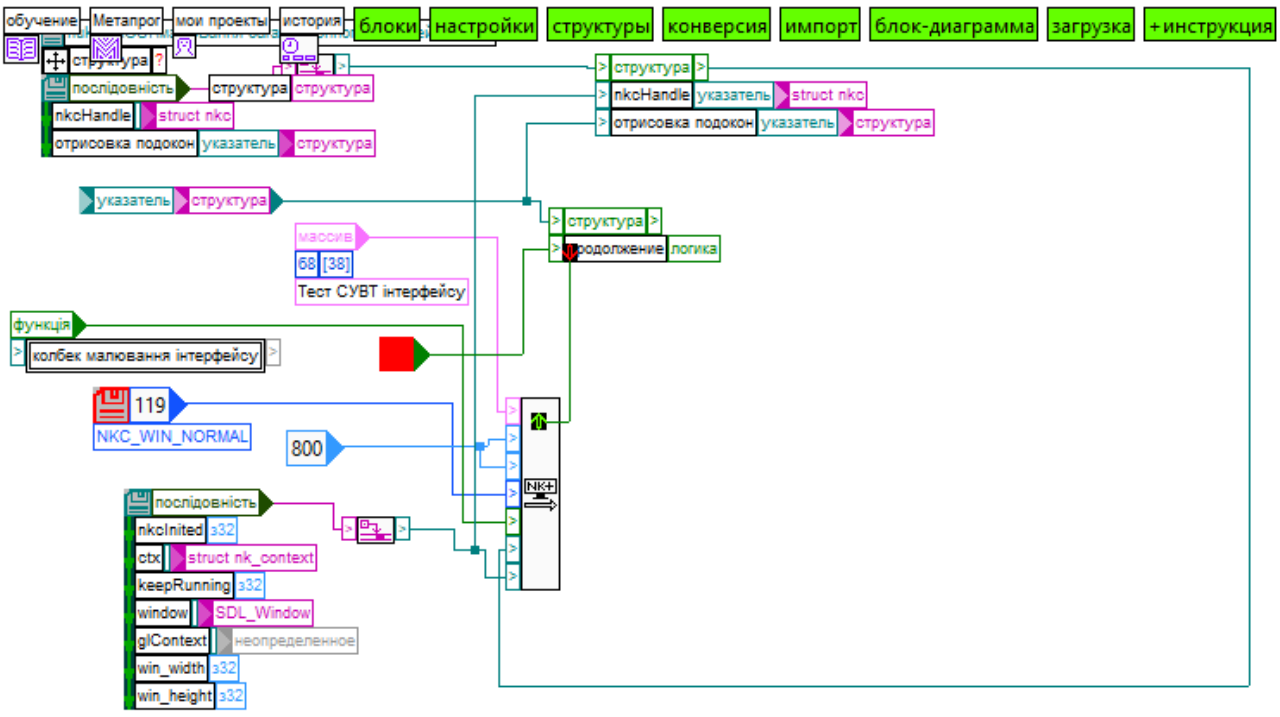

Рисунок 2.28 – Функцію «Малювання графічного інтерфейсу»

5. Callback малювання графічного інтерфейсу

Функція малювання графічного інтерфейсу має свій “callback”. (рис. 2.29 – 2.32)

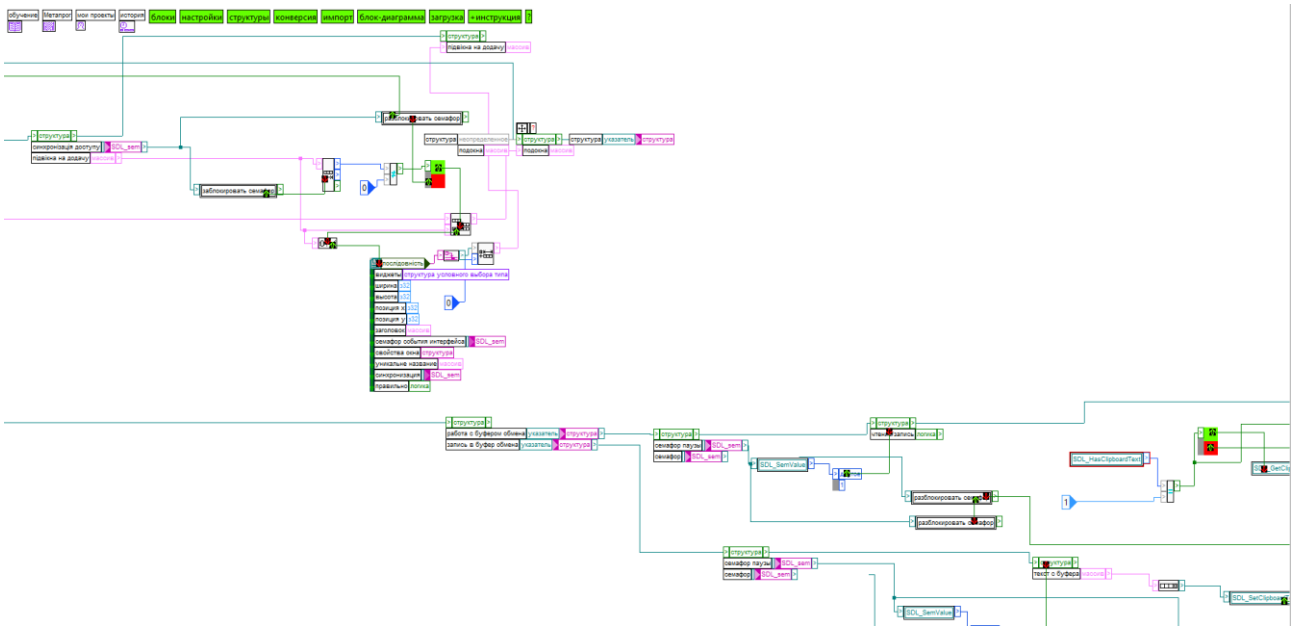

Рисунок 2.29 – Callback малювання графічного інтерфейсу (частина)

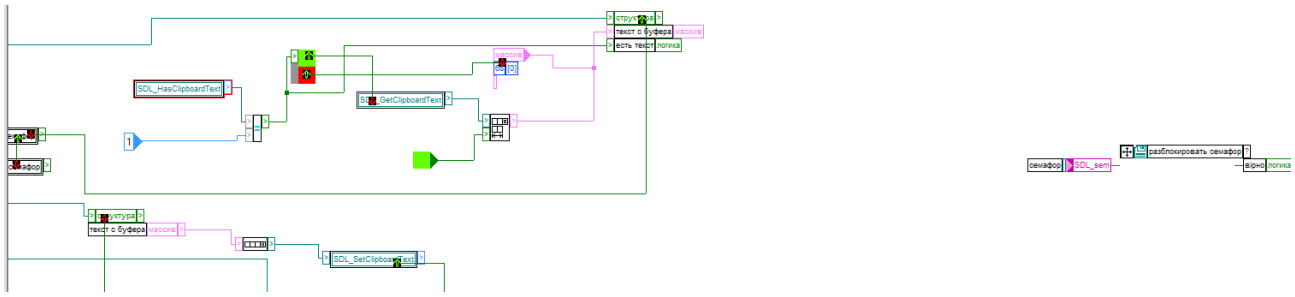

Рисунок 2.30 – Callback малювання графічного інтерфейсу (частина)

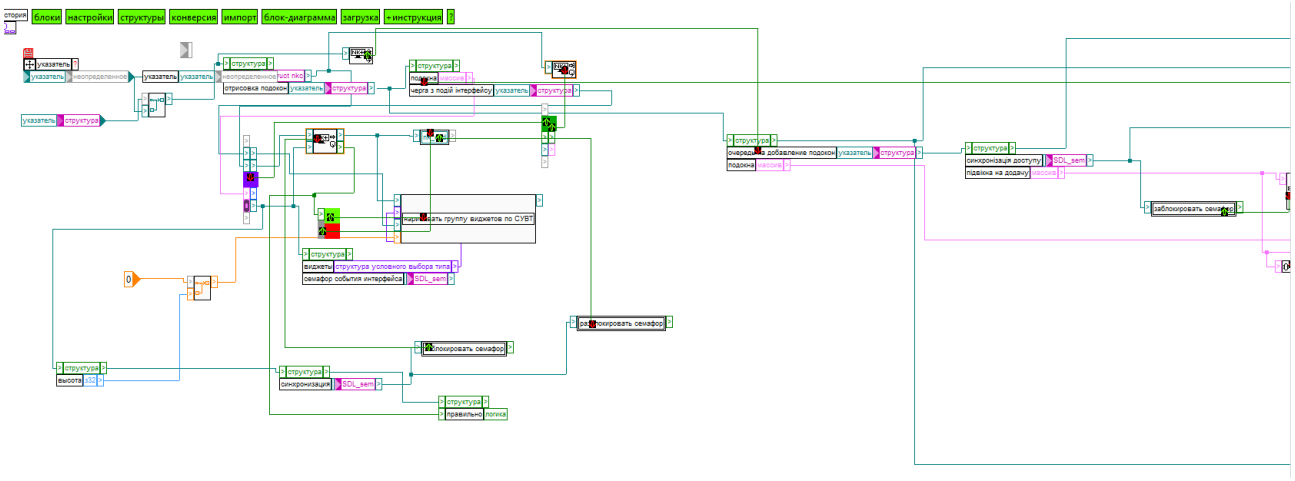

Рисунок 2.31 – Callback малювання графічного інтерфейсу (частина)

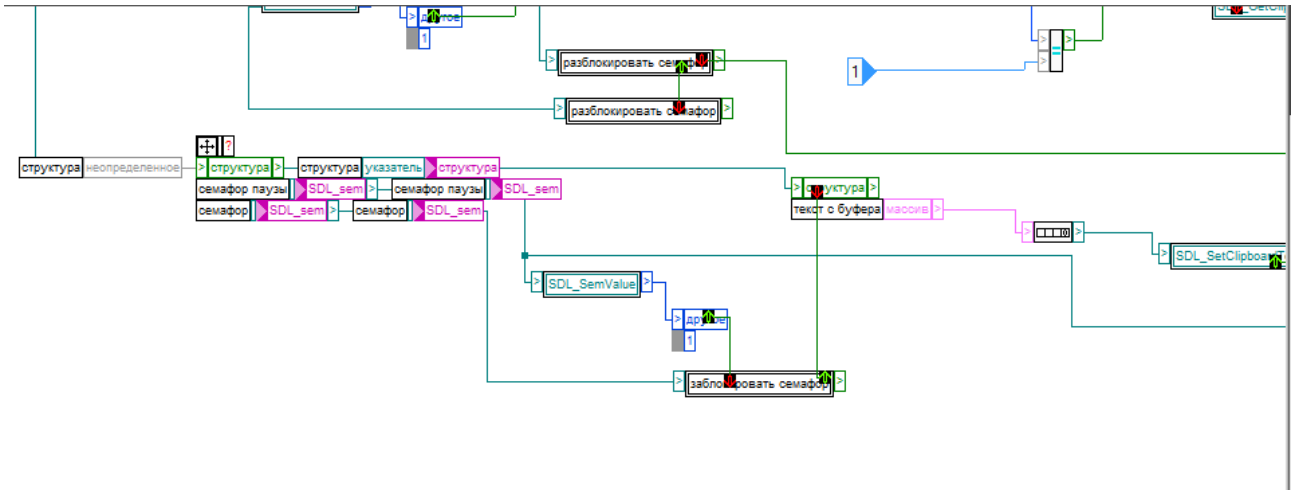

Рисунок 2.32 – Callback малювання графічного інтерфейсу (частина)

6. Малювання віджетів по СУВТ

Ця функція має в своєму складі меншу функцію «малювання віджетів по СУВТ» - важливу складову що малює віджети у вікнах й під вікнах и виглядає наступним чином. (рис. 2.33)

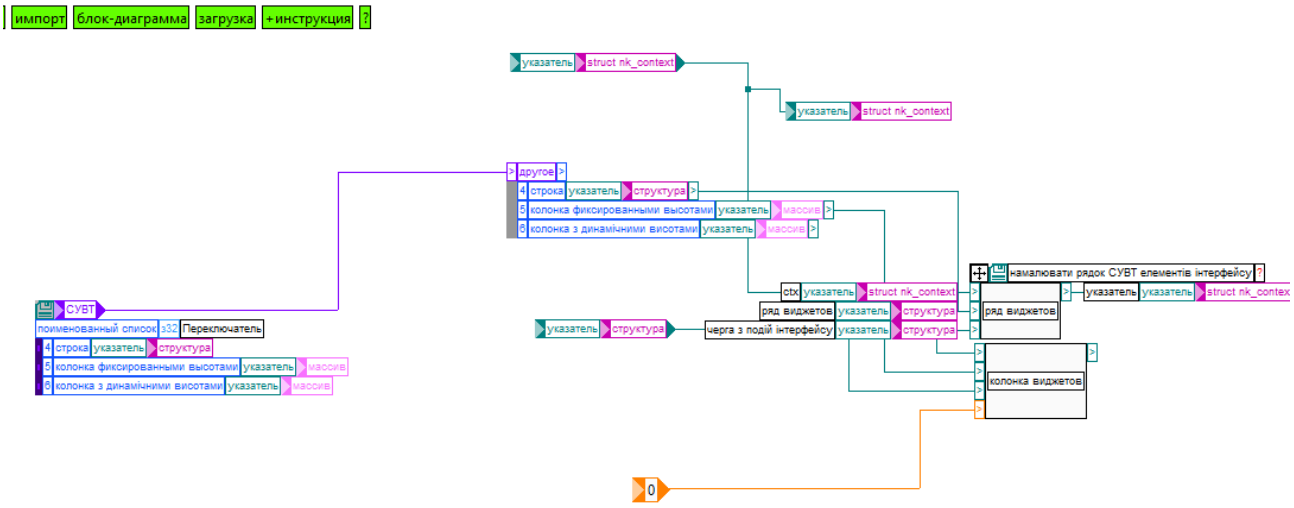

Рисунок 2.33 – Функція «малювання віджетів по СУВТ»

Нагадаю, що СУВТ – це структура умовного вибору типу – це легко побачити дивлячись на її складові змінні: «строка», «колонка фиксированных высот», «колонка з динамічними висотами». Цифри на малюнку нижче – це номери селектору що визначають (вказують. Teg-ують) тип СУВТ (де факто це юніон з цілих чисел, в тому числі enum). (рис. 2.34)

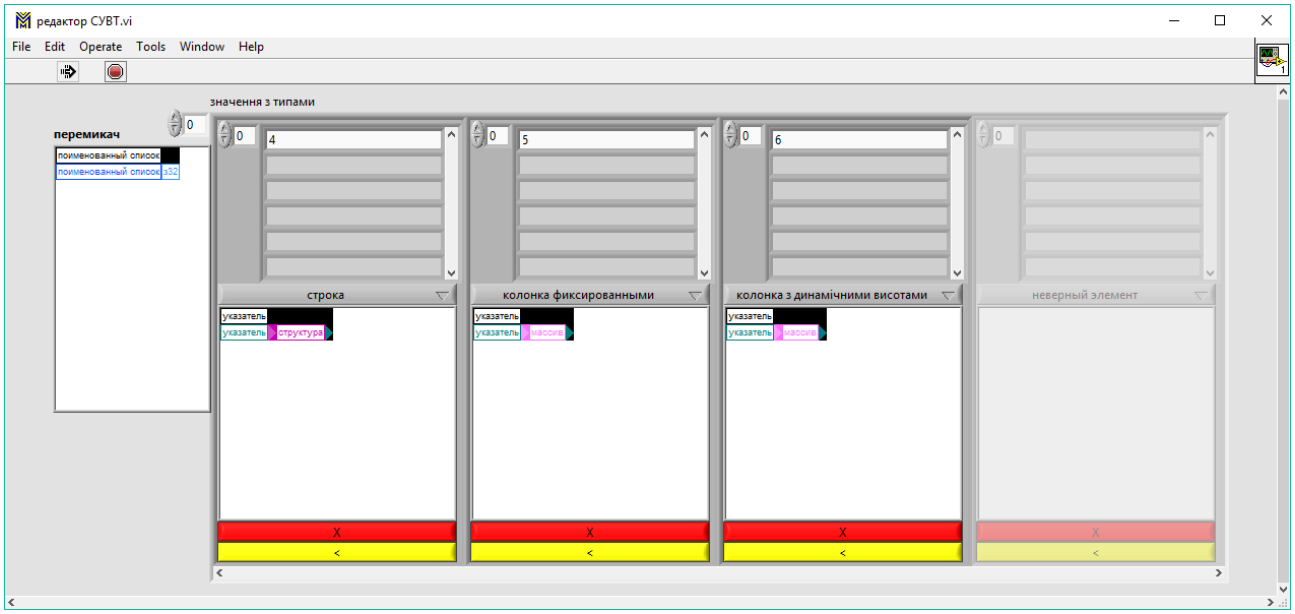

Рисунок 2.34 – Складові функції «малювання віджетів по СУВТ»

Цікаво що у даному випадку не має різниці на якій мові звуться змінні у коді, що надає можливість називати їх українською, російською й англійськими мовами тому що в Метапрогу майже всі назви призначені для читання людьми багатомовні. (рис. 2.35)

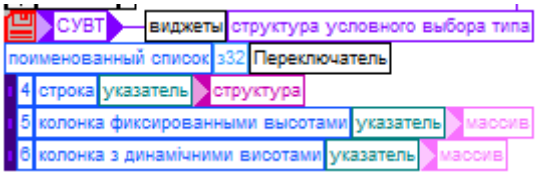

Рисунок 2.35 – Назви змінних на різних мовах

6.1. Ряд віджетів

Віджети визначаються і упорядковуються даною функцією в ряд. Складовою частиною даної функції є функція «один елемент інтерфейсу». (рис. 2.36)

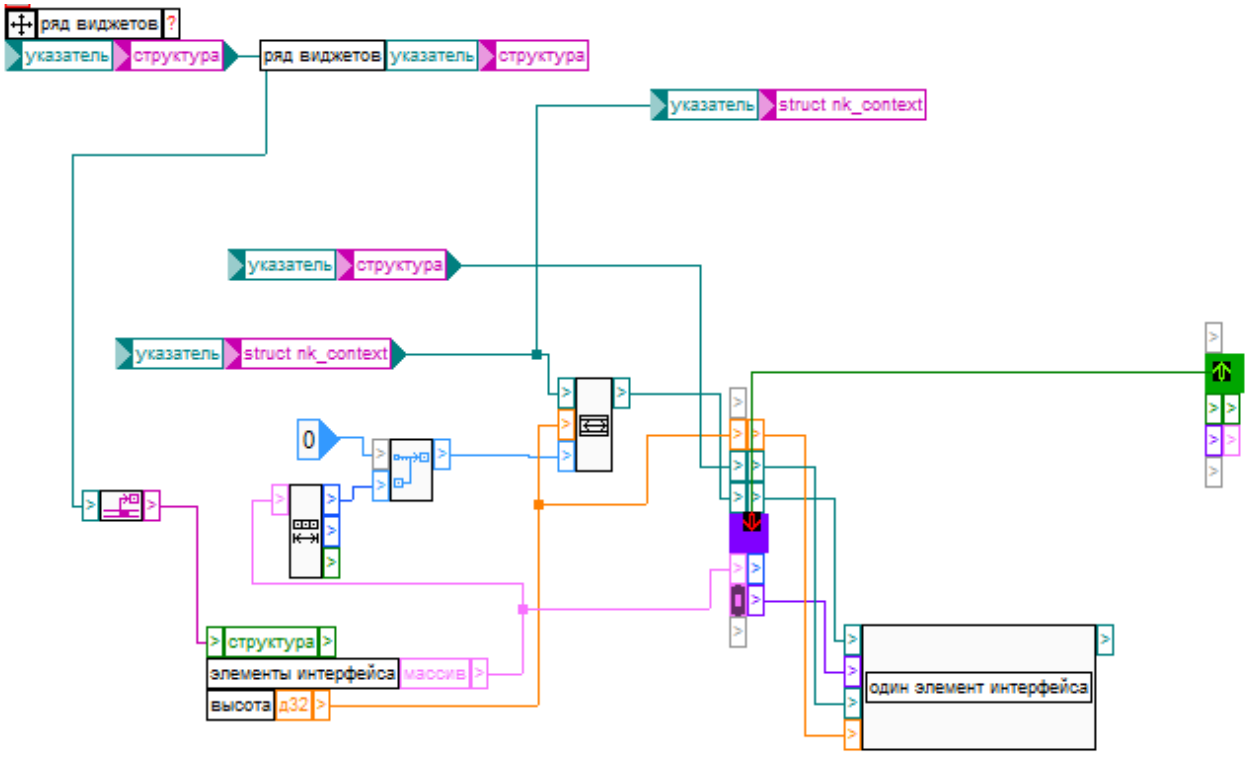

Рисунок 2.36 – Ряд віджетів

6.2 Один елемент інтерфейсу

Ця функція є перемикачем - еквівалентом функції «switch» у мові С різними елементами інтерфейсу. (рис. 2.37)

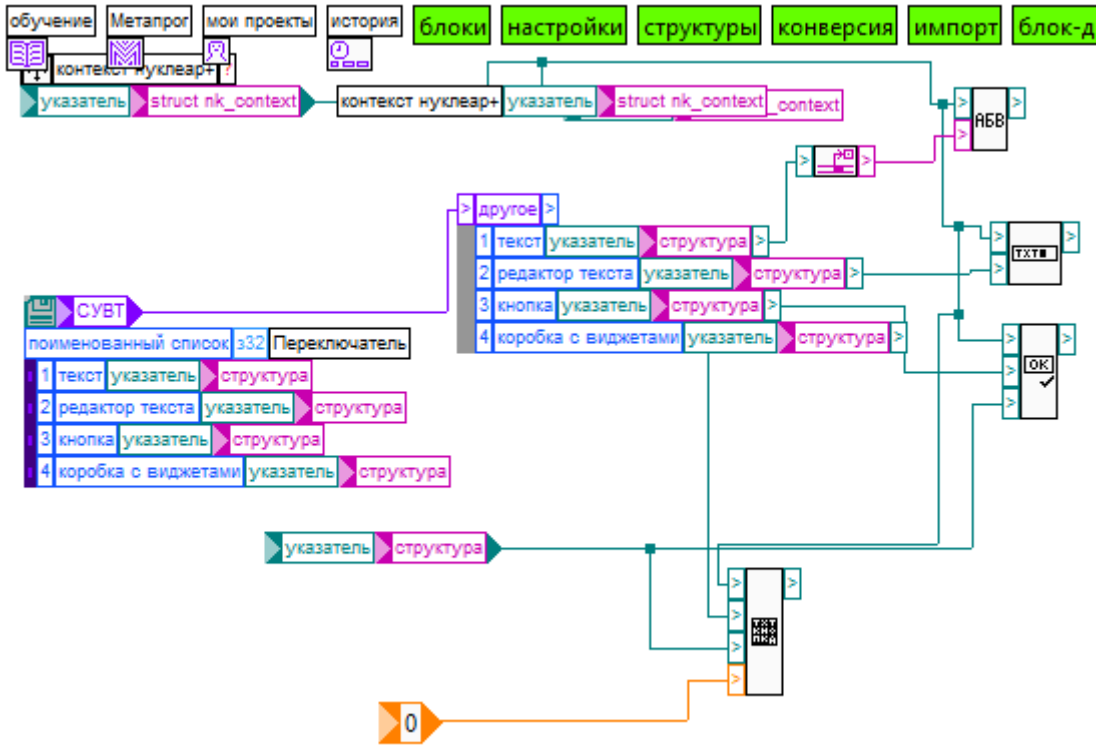

Рисунок 2.37 – Один елемент інтерфейсу

6.3 Малювання текстового лейблу.

Так виглядає лейбл текст. (рис. 2.38, 2.39)

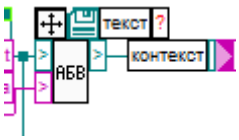

Рисунок 2.38 – Блок лейблу тексту

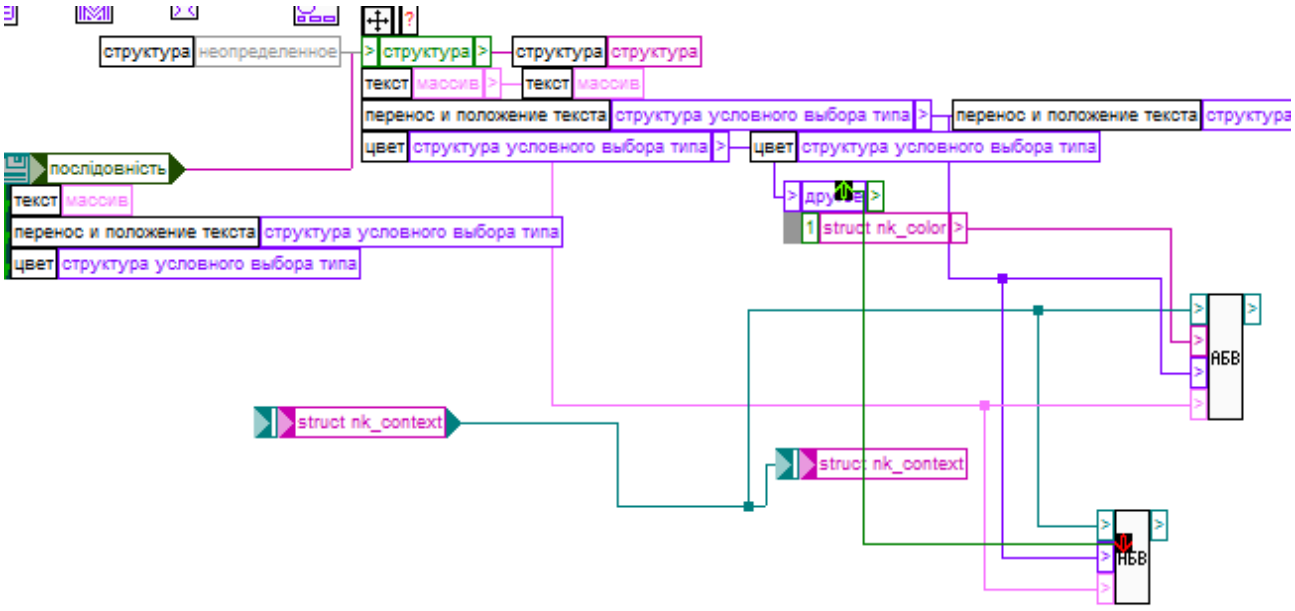

Рисунок 2.39 – Лейбл текст

Який у свою чергу може виводити кольоровий текст (верхній блок «АБВ»). (рис. 2.40)

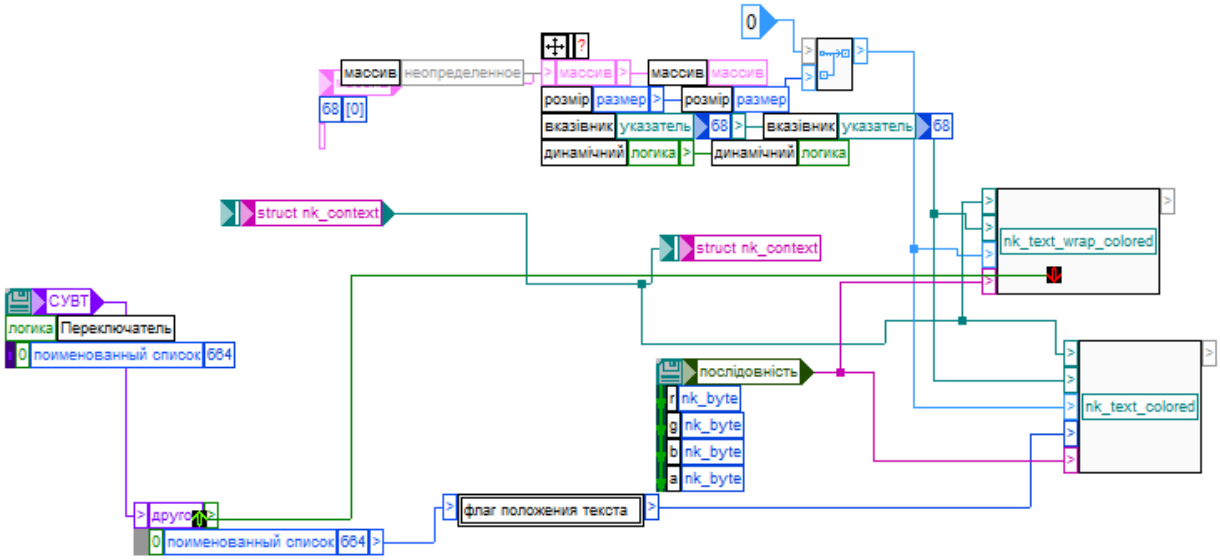

Рисунок 2.40 – Кольоровий текст

Або не кольоровий (інший блок). (рис. 2.41, 2.42)

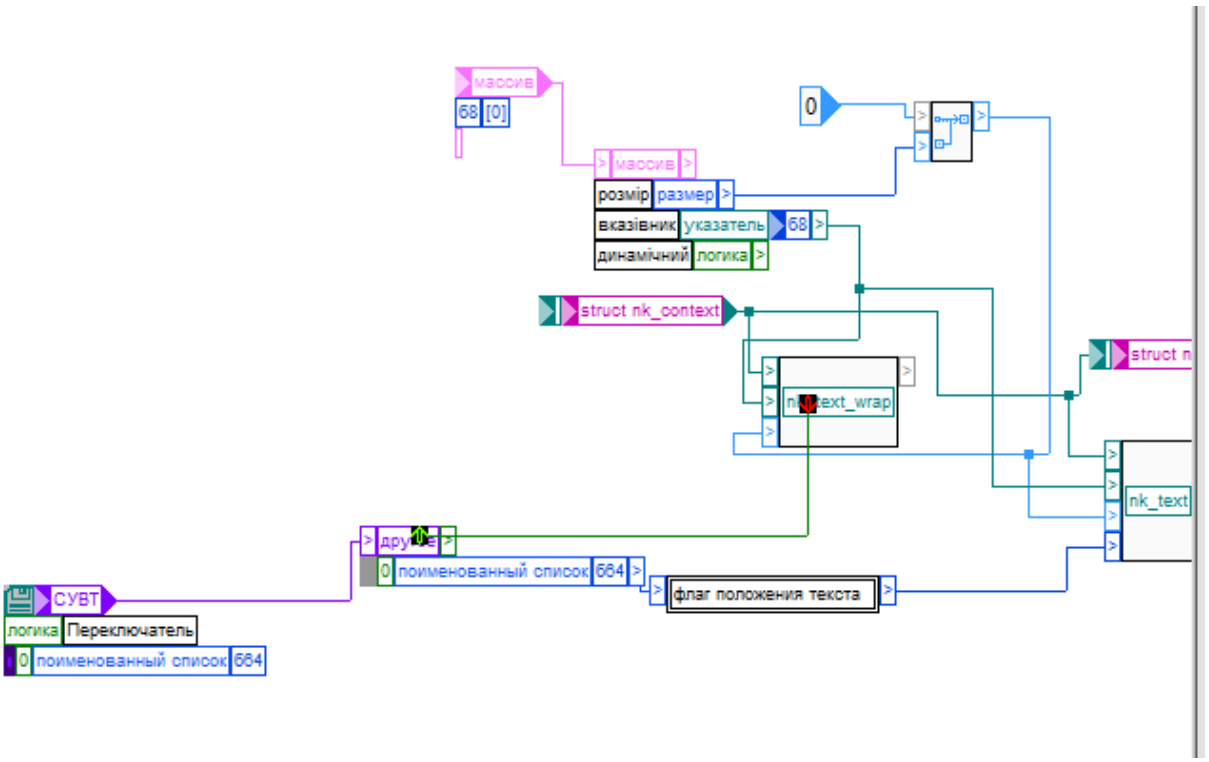

Рисунок 2.41 – Не кольоровий текст частина

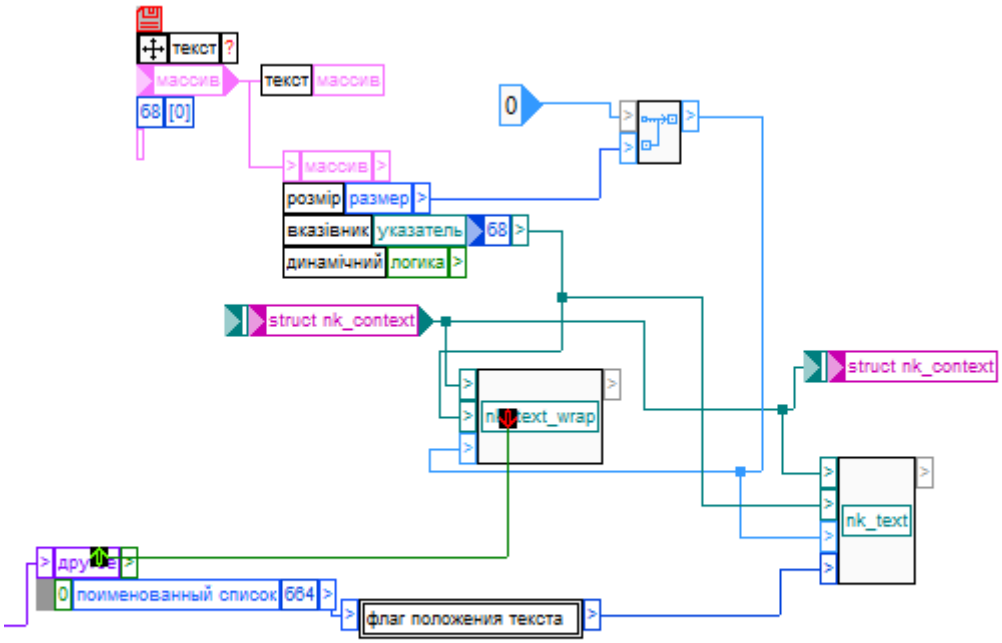

Рисунок 2.42 – Не кольоровий текст частина

Інший блок це текстове поле (підписане поле як редактор тексту) що використовую функції й примітиви C. (рис. 2.43, 2.44)

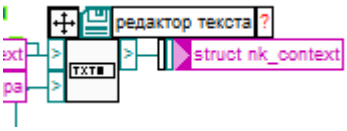

Рисунок 2.43 – Блок функції текстове поле

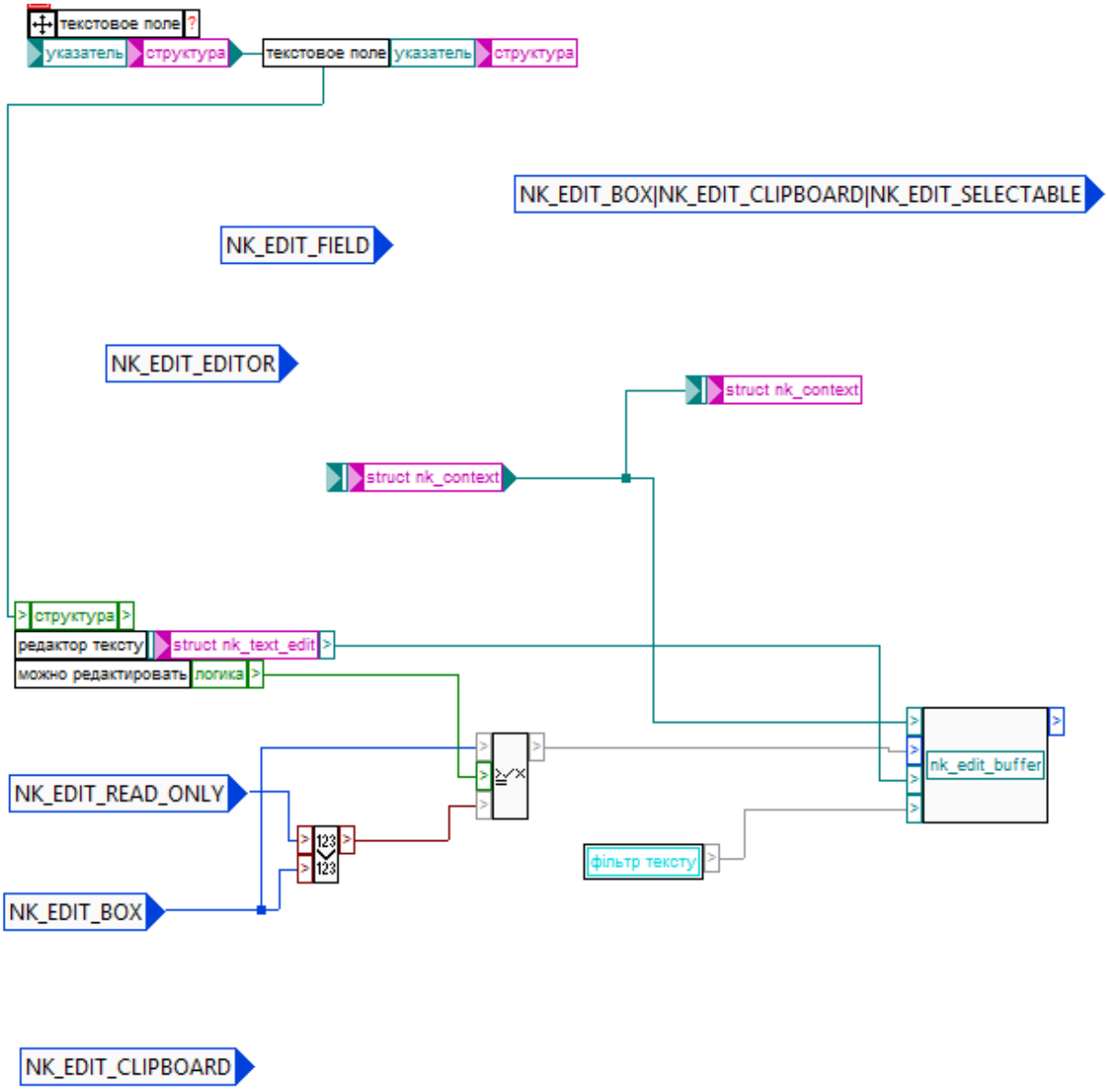

Рисунок 2.44 – Функція текстове поле

Третій і останній блок це вказівник на функцію з бібліотеки бібліотеки nuklear+ що відповідає за малювання в графічному інтерфейсі. (рис. 2.45)

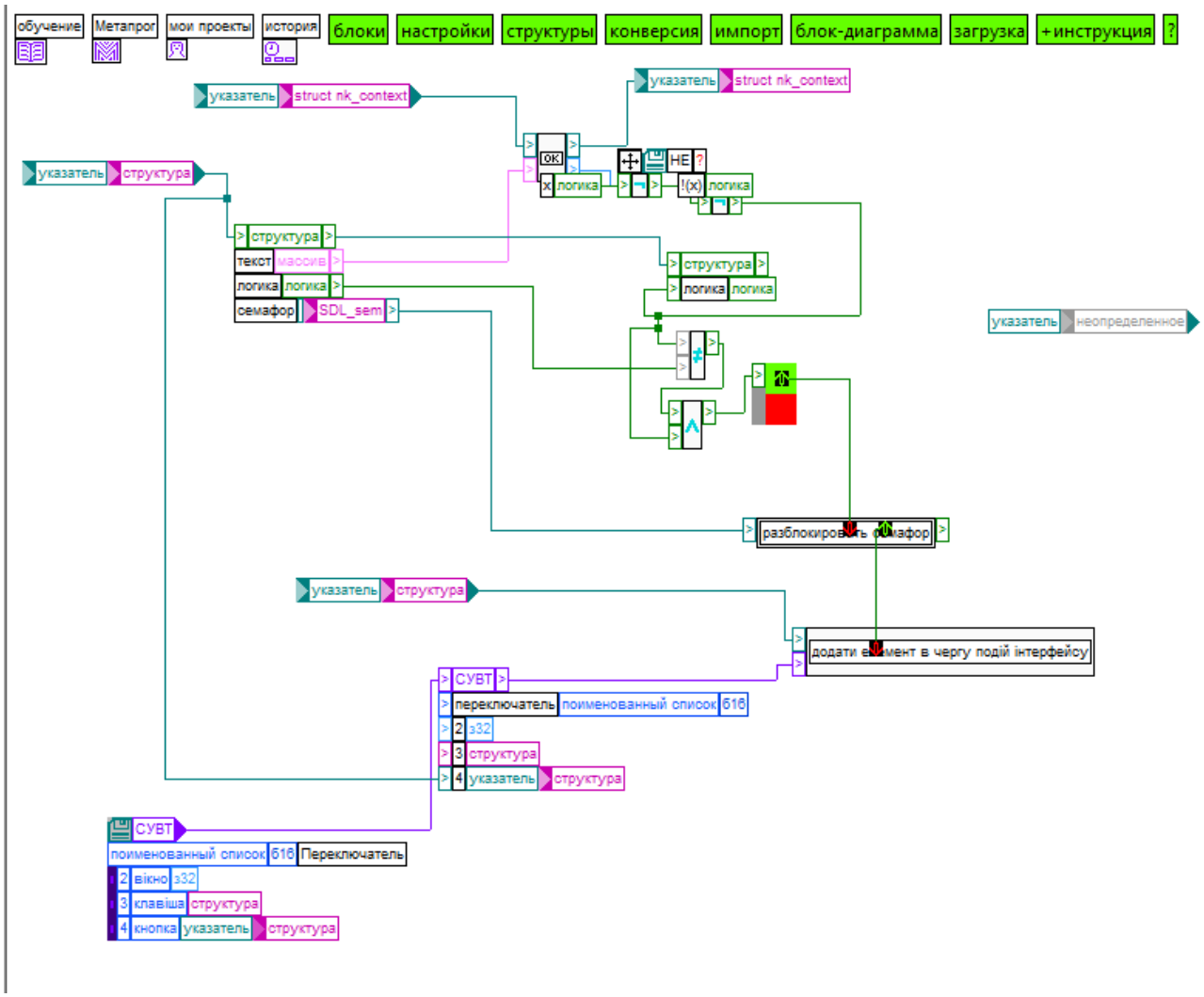

Рисунок 2.45 – Вказівник на функцію з бібліотеки бібліотеки nuklear+

При роботі з графікою у даному блоці ми використовуємо свої семафори про які вже зазначено вище для синхронізації подій у інтерфейсі. (рис. 2.46)

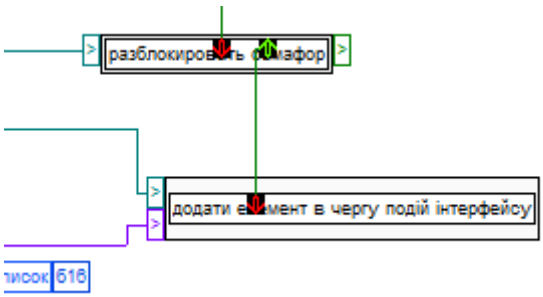

Рисунок 2.46 – Використання семафорів в Техночаті

Четвертий блок – це група віджетів яка має окрему смугу прокрутки. (рис. 2.47, 2.48)

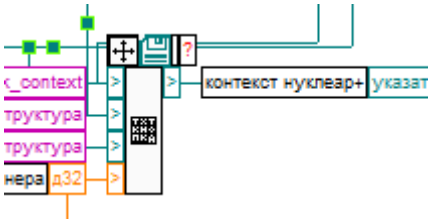

Рисунок 2.47 – Блок групи віджетів

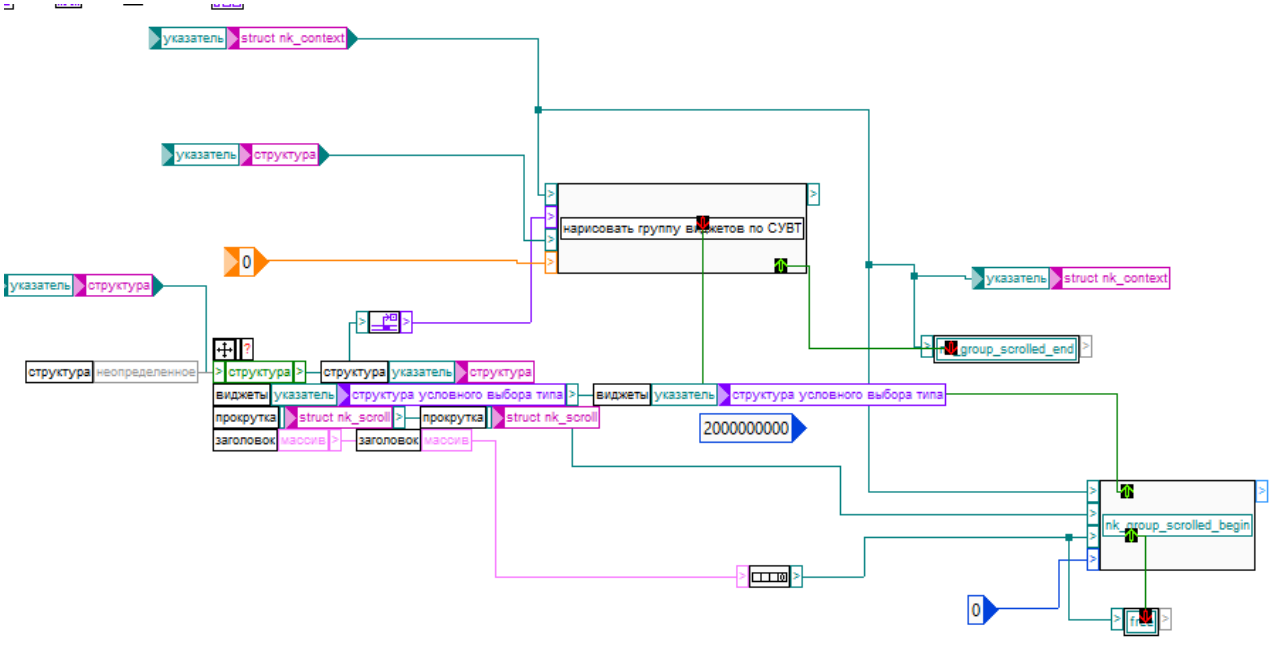

Рисунок 2.48 – Група віджетів

6.4 Функція шрифтів, ініціалізація з nuklear+

В ній ми можемо знайти функцію ініціалізації nuklear+, що відповідає за шрифти тексту. В ній також використовуються примітиви з мови С. (рис. 2.49)

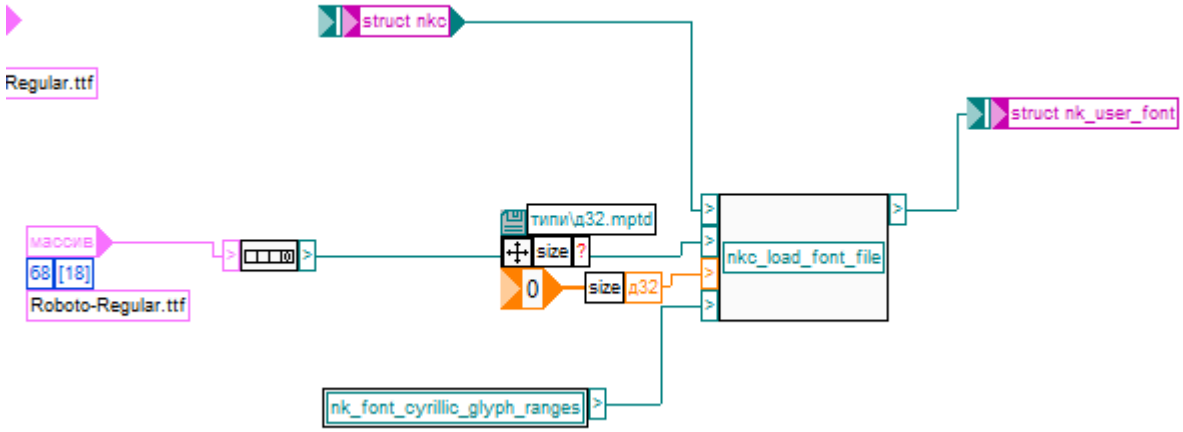

Рисунок 2.49 – Функція шрифтів

6.5 Створення нових потоків використовуючи SDL_CreateThread

Перейдемо до функції створення потоку що основана на SDL_CreateThread й закидує туди вказівники на масиви з нультермінованних рядків. (рис. 2.50, 2.51)

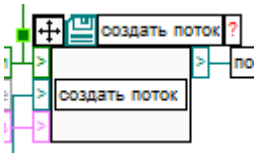

Рисунок 2.50 – Блок створення нових потоків

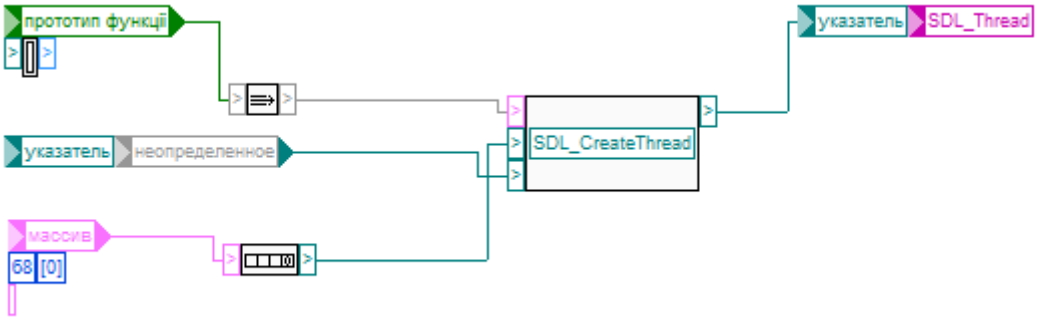

Рисунок 2.51 – Функція створення нових потоків

6.6 Функція малювання підвікон

Функція малювання підвікон (не плутати з функціями «малювання вікон», «вікно в вікні» й іншими) за допомогою семафорів й вказівників на віджити й підвікна малює підвіконця. (рис. 2.52 – 2.55)

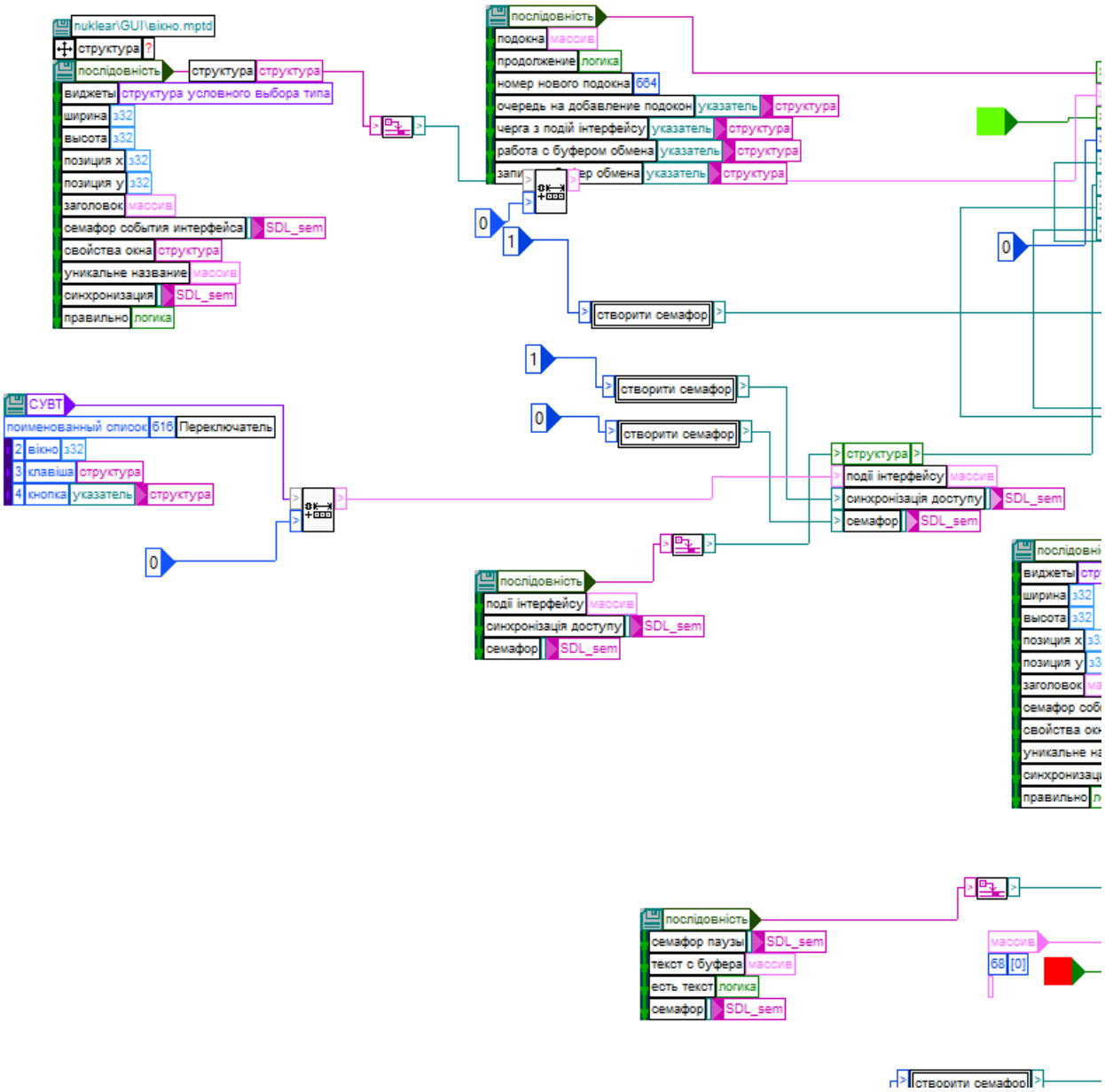

Рисунок 2.52 – Функція малювання підвікон (частина)

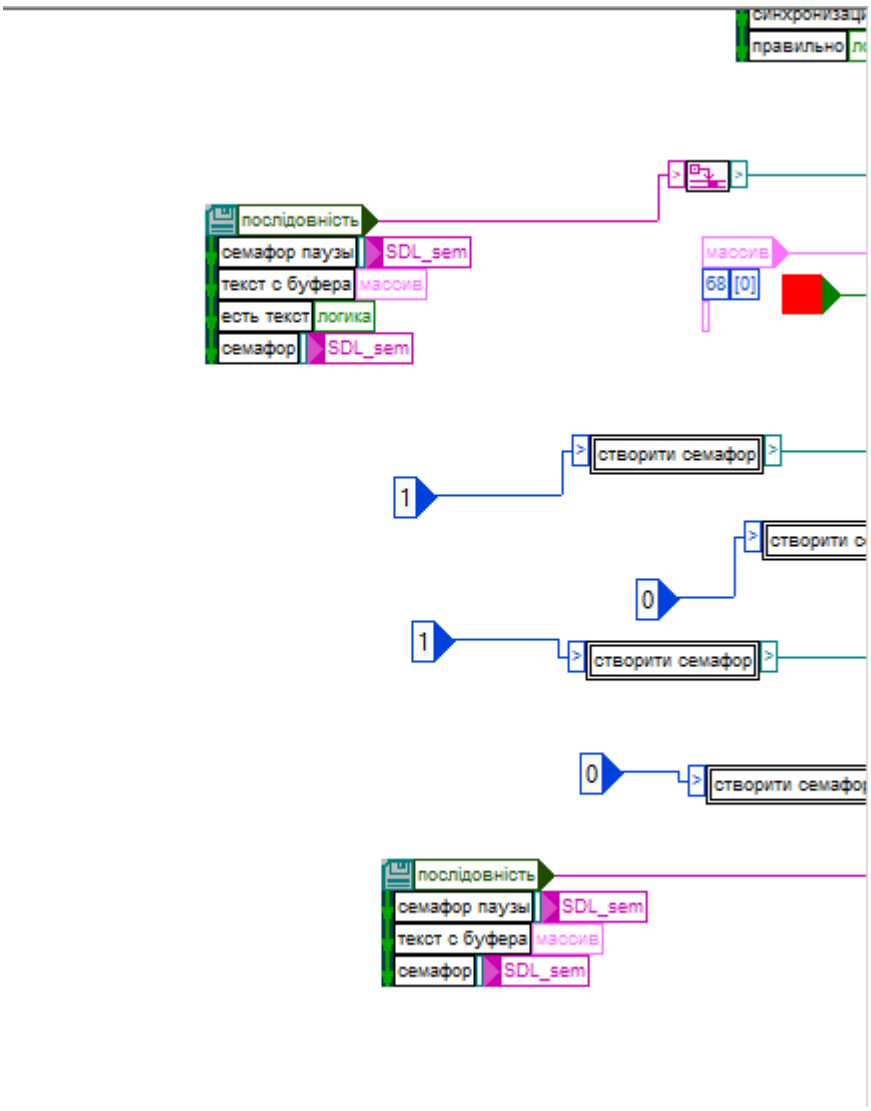

Рисунок 2.53 – Функція малювання підвікон (частина)

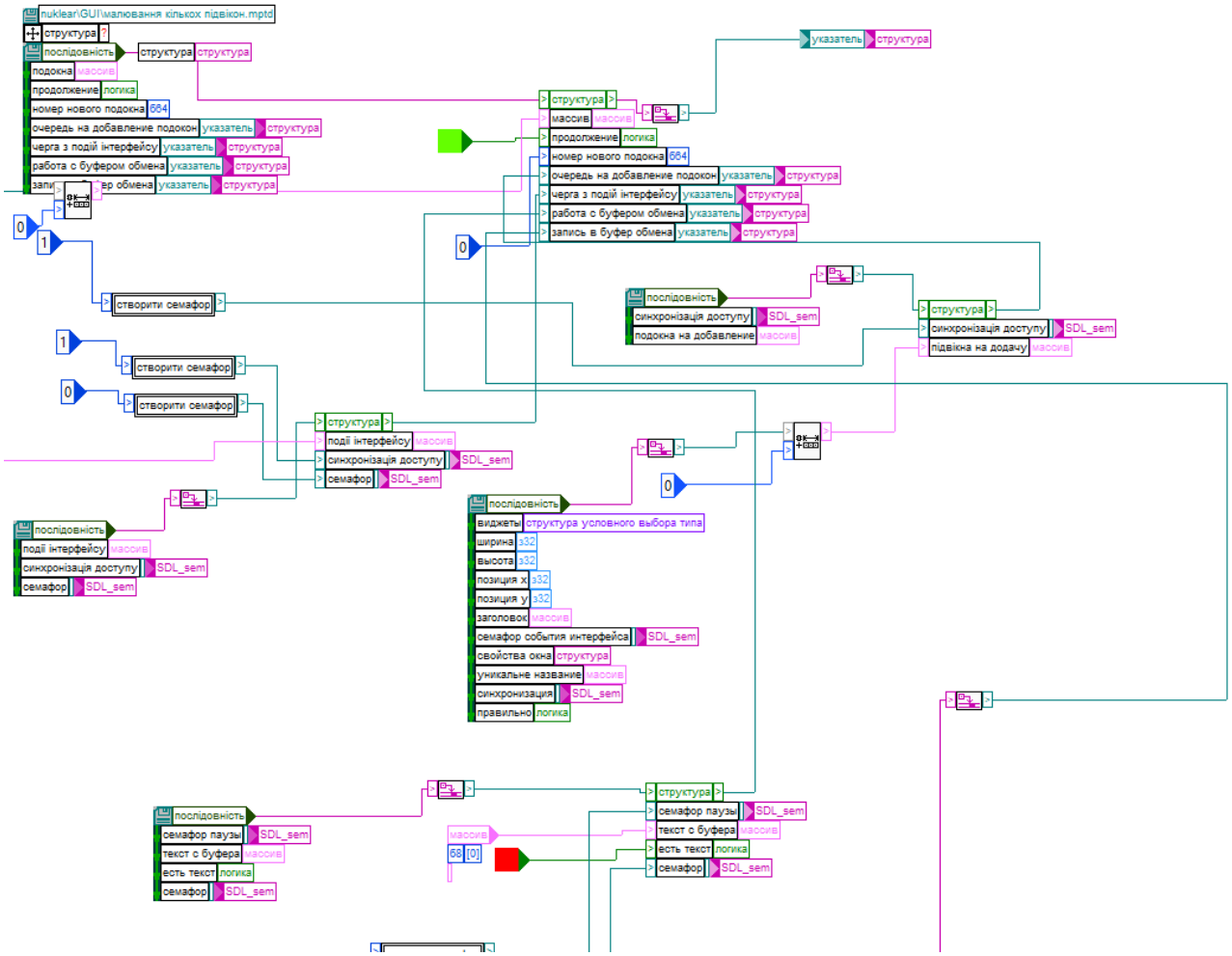

Рисунок 2.54 – Функція малювання підвікон (частина)

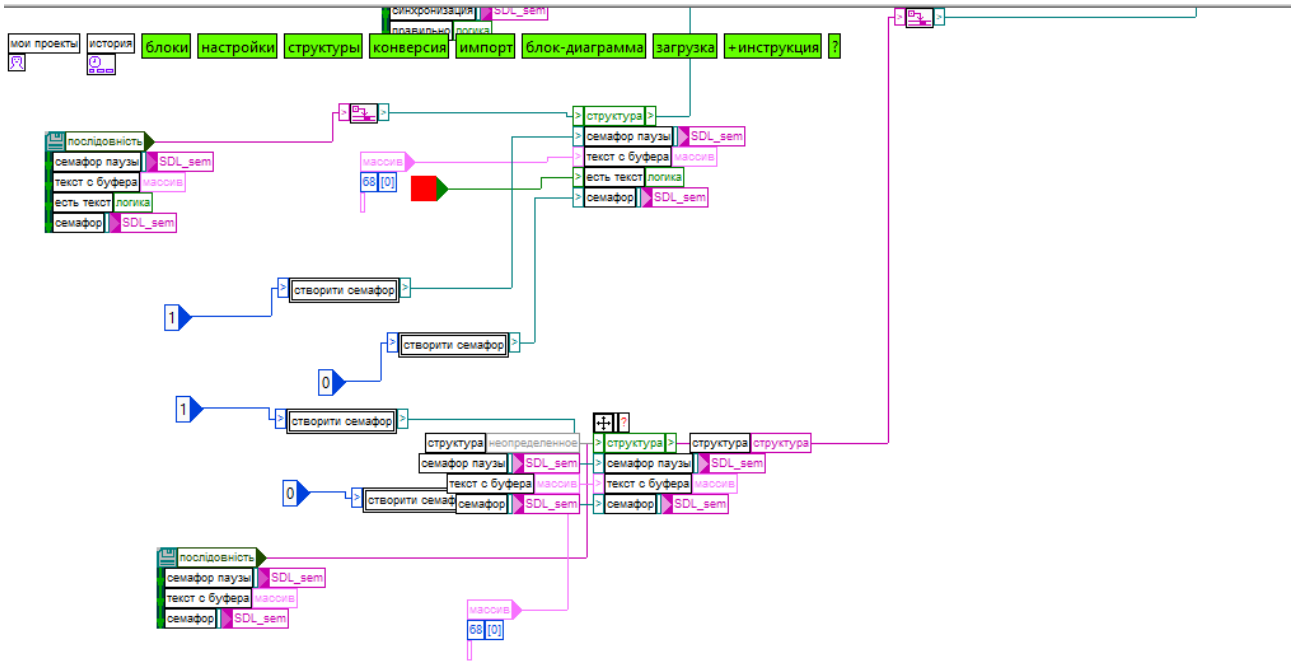

Рисунок 2.55 – Функція малювання підвікон (частина)

Також для їх малювання вона використовує структури: (рис. 2.56, 2.57)

Рисунок 2.56 – Структури даних для Функції малювання підвікон

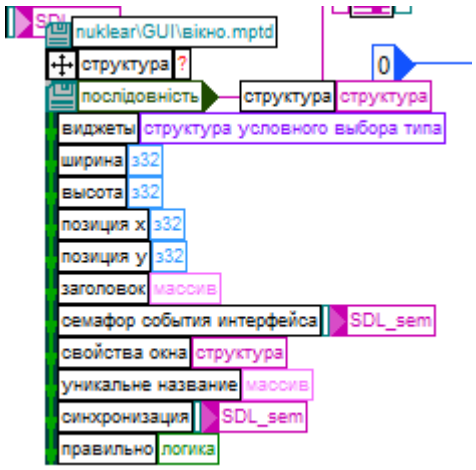

Рисунок 2.57 – Функція малювання підвікон (частина)

Саме структури і вказівники на них задають властивості підвікон. Від цього залежить буде підвікно активним, на передньому плані, скільки в ньому буде довжини й ширини, які віджити будуть додані до вікна.

7. Функція читання конфігурації

Так виглядає блок функції «прочитати конфіг» у бібліотеці «Техночат» і самому чаті. (рис. 2.58)

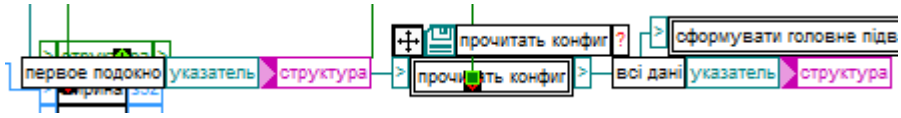

Рисунок 2.58 – Блок функції читання конфігурації

І сама функція. (рис. 2.59, 2.60)

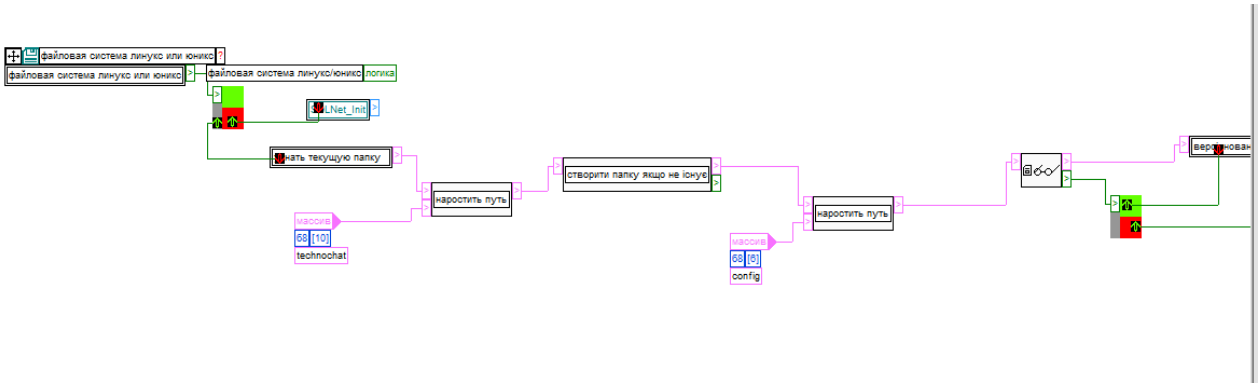

Рисунок 2.59 – Функція читання конфігурації (частина)

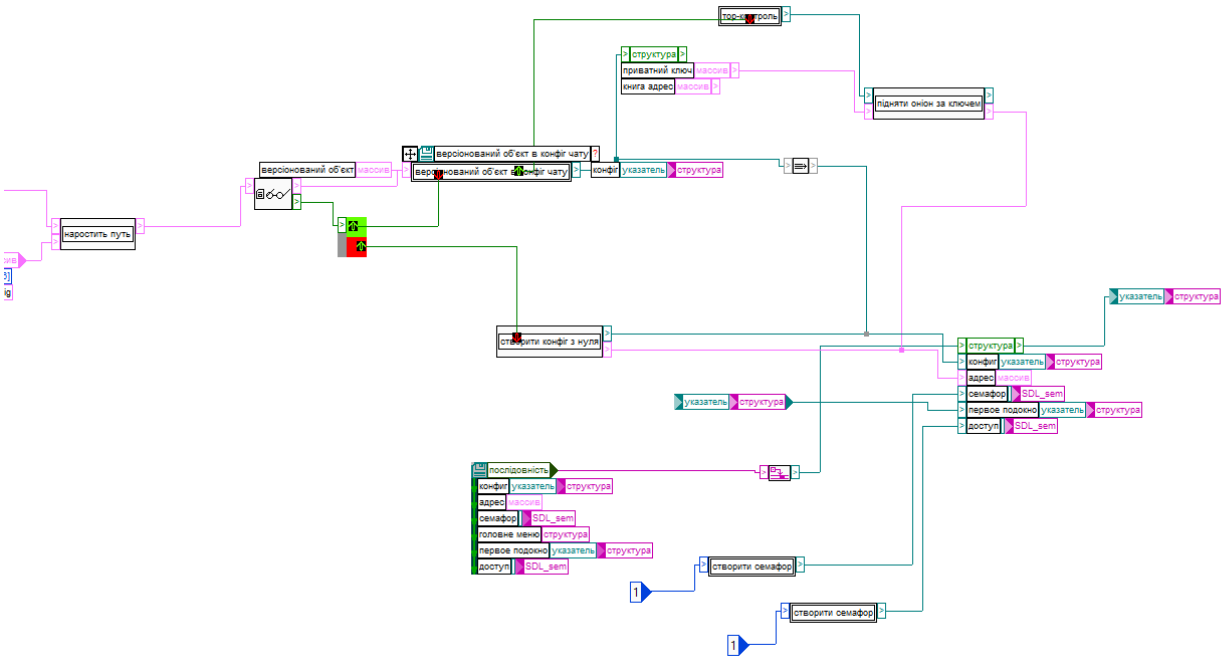

Рисунок 2.60 – Функція читання конфігурації (частина)

7.1 Функція «Файлова система лінукс або юнікс»

Блок функції в схемі Техночату віглядає так. (рис. 2.61)

Рисунок 2.61 – Блок функції «Файлова система лінукс або юнікс»

Першочергово згідно схеми ми визначаємо яку операційну систему використовує користувач. Схема функції наступна. (рис. 2.62, 2.63)

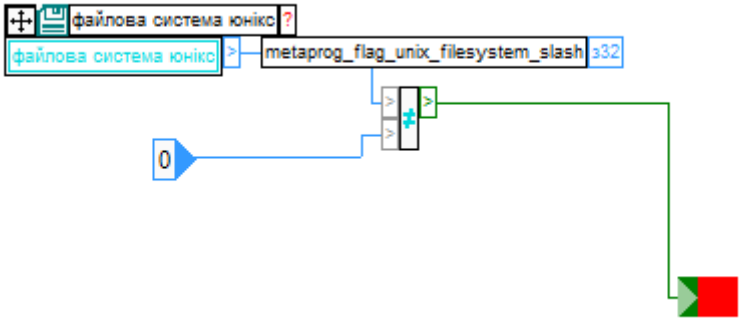

Рисунок 2.62 – Логіка функції

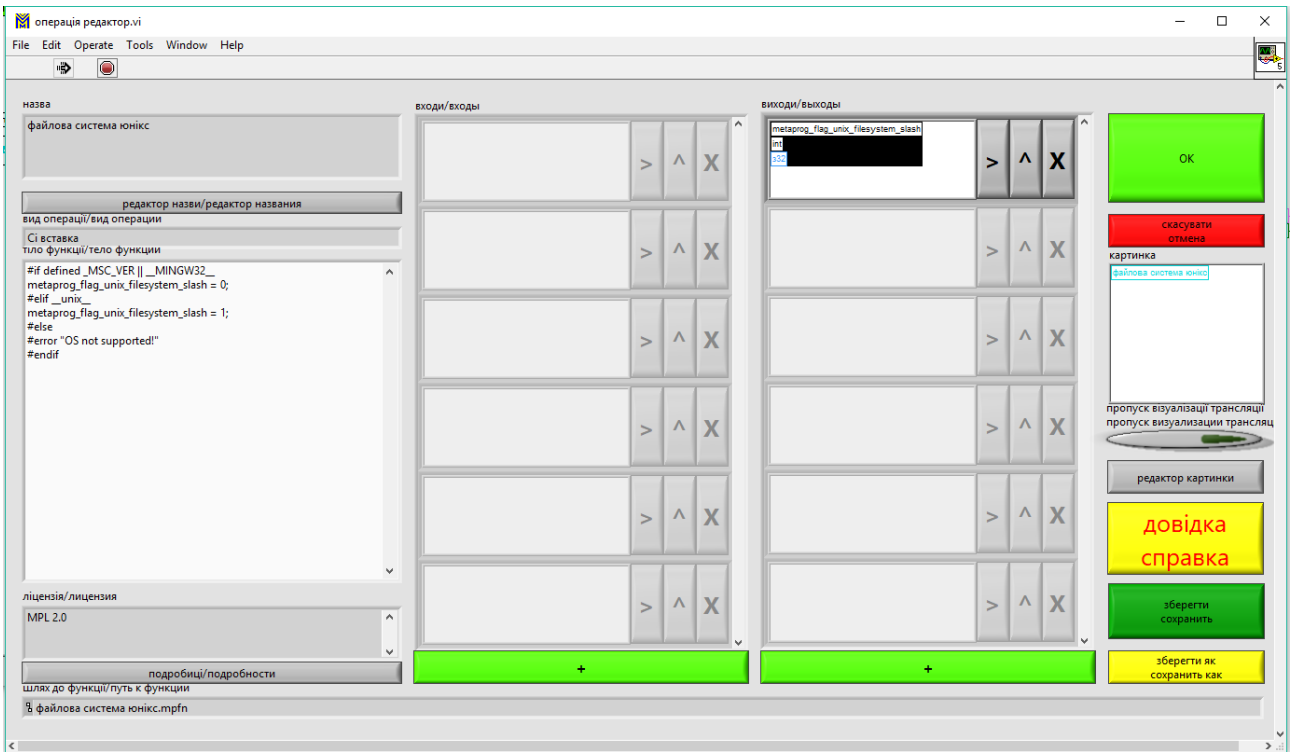

Рисунок 2.63 – Вставка з мови С.

Макроси на мові С для встановлення операційної системи необхідні для того щоб програма чату була кросплатформовою. Справа в тому що на різних операційних системах бувають різні види запису шляхів й структури даних. Так наприклад шляхи до папок й файлів на Лінуксі і в Віндоувс це різні набори слешів «//» і «\» відповідно.

```
#if defined _MSC_VER || __MINGW32__
metaprog_flag_unix_filesystem_slash = 0;
#elif __unix__
metaprog_flag_unix_filesystem_slash = 1;
#else
#error "OS not supported!"
#endif
```

Цікаво, що першочергово ця функція була зовсім іншою тому що визначала операційну систему за її ознаками і особливостями роботи, наприклад по запису шляхів до файлів і папок. Пишучи роботу було викладено декілька нових версій тому деякі функції з зазначених у роботі в момент її здачі вже еволюціонують.

7.2 Функція «Вказати поточну папку»

Блок функції «Вказати поточну папку» в схемі виглядає так. (рис. 2.64)

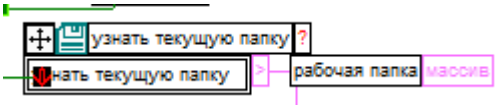

Рисунок 2.64 – блок Функції «Вказати поточну папку»

Згідно назви ця функція знаходить перебором рекурсивного циклу папку для конфігу. (рис. 2.65)

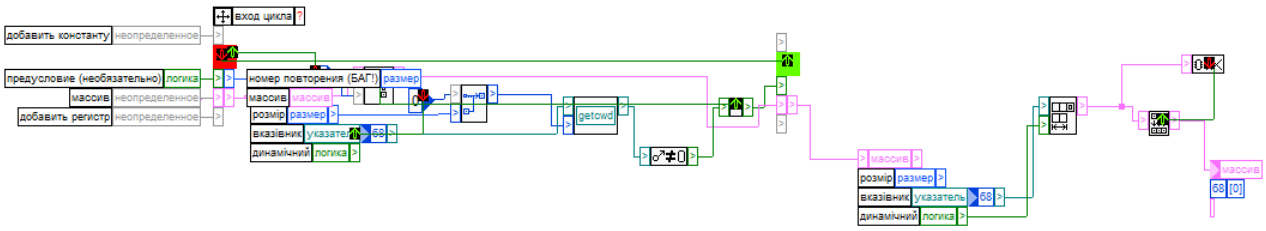

Рисунок 2.65 – Функція «Вказати поточну папку»

У даній функції є ліміт на розмір шляху у буфері обміну рівний 500

одиницям символів. Це необхідний ліміт якого має вистачити користувачу при задані шляху для конфігу Техночату.

7.3 Функція «Наростити шлях»

Блок функції «Наростити шлях» в схемі Техночату виглядає так. (рис. 2.66, 2.67)

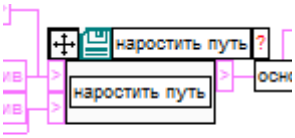

Рисунок 2.66 – Блок функції «Наростити шлях»

Це проста функція додавання назв папок до повного шляху враховуючи оформлення шляхів в різних операційних системах.

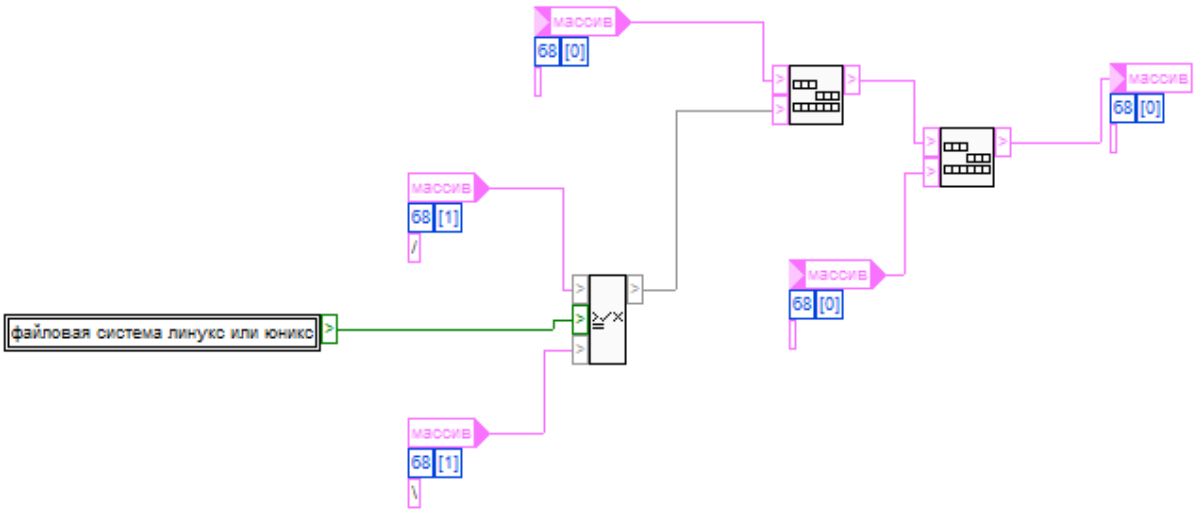

Рисунок 2.67 – Функція «Наростити шлях»

Якщо ви уважно роздивились пункт «Функція читання конфігурації» то могли помітити що масив з назвою technochat підключений до даної функції і називає робочу папку конфігу. (рис. 2.68)

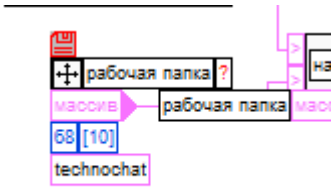

Рисунок 2.68 – Функція читання конфігурації

7.4 Функція «Створити папку якщо не існує»

Блок функції «створити папку якщо не існує» виглядає наступним чином. (рис. 2.69)

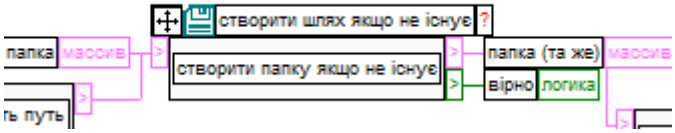

Рисунок 2.69 – Блок Функції «Створити папку якщо не існує»

Назва даної функції відповідає її призначенню – вона перевіряє чи є папка конфігу і створює її якщо такої не знайдено за шляхом зазначеним в функції «Наростити шлях». (рис. 2.70)

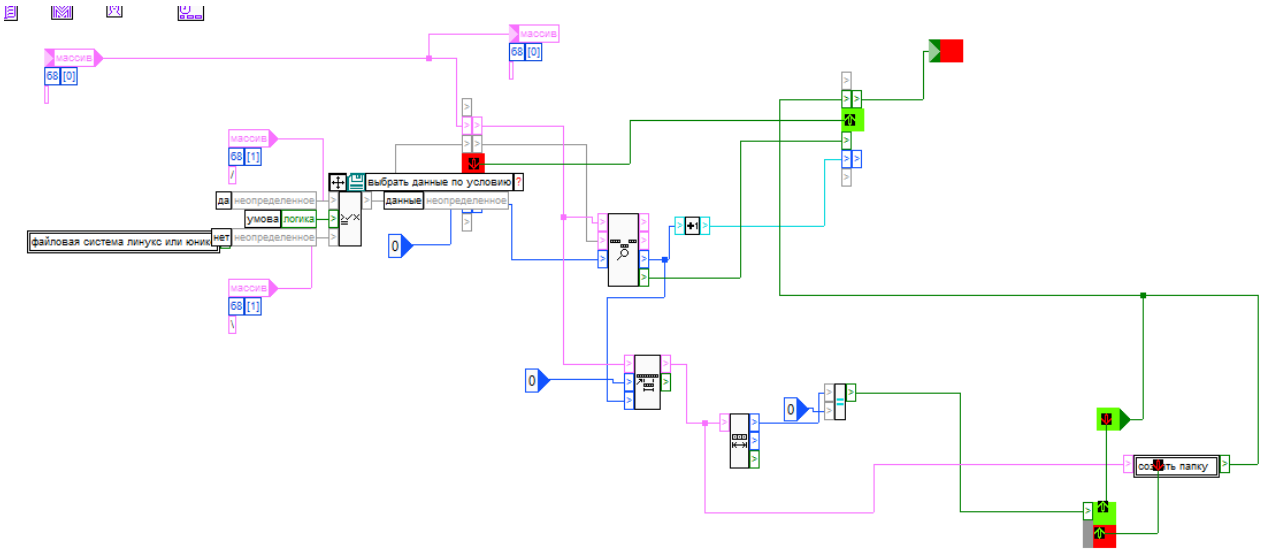

Рисунок 2.70 – Функція «Створити папку якщо не існує»

Логіка функції дуже проста і зрозуміла якщо на неї подивитися. Якщо червоне (0, False, логічне ні) то створюємо папку, інакше зелене – йдемо за послідовністю. (рис. 2.71)

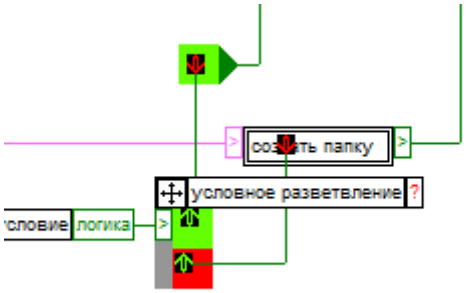

Рисунок 2.71 – Логіка функції «Створити папку якщо не існує»

Складнішими елементами є підстроки для входження. (рис. 2.72)

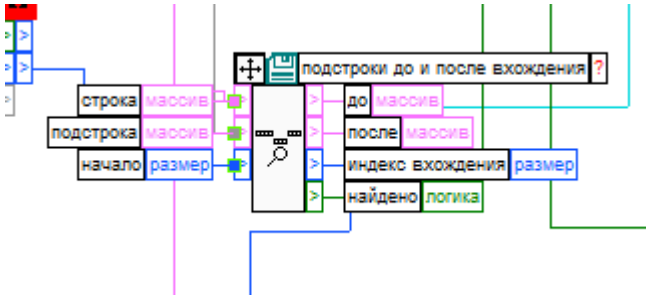

Рисунок 2.72 – Входження функції «Створити папку якщо не існує»

В ній ми пишемо в циклі рядки-підрядки. (рис. 2.73)

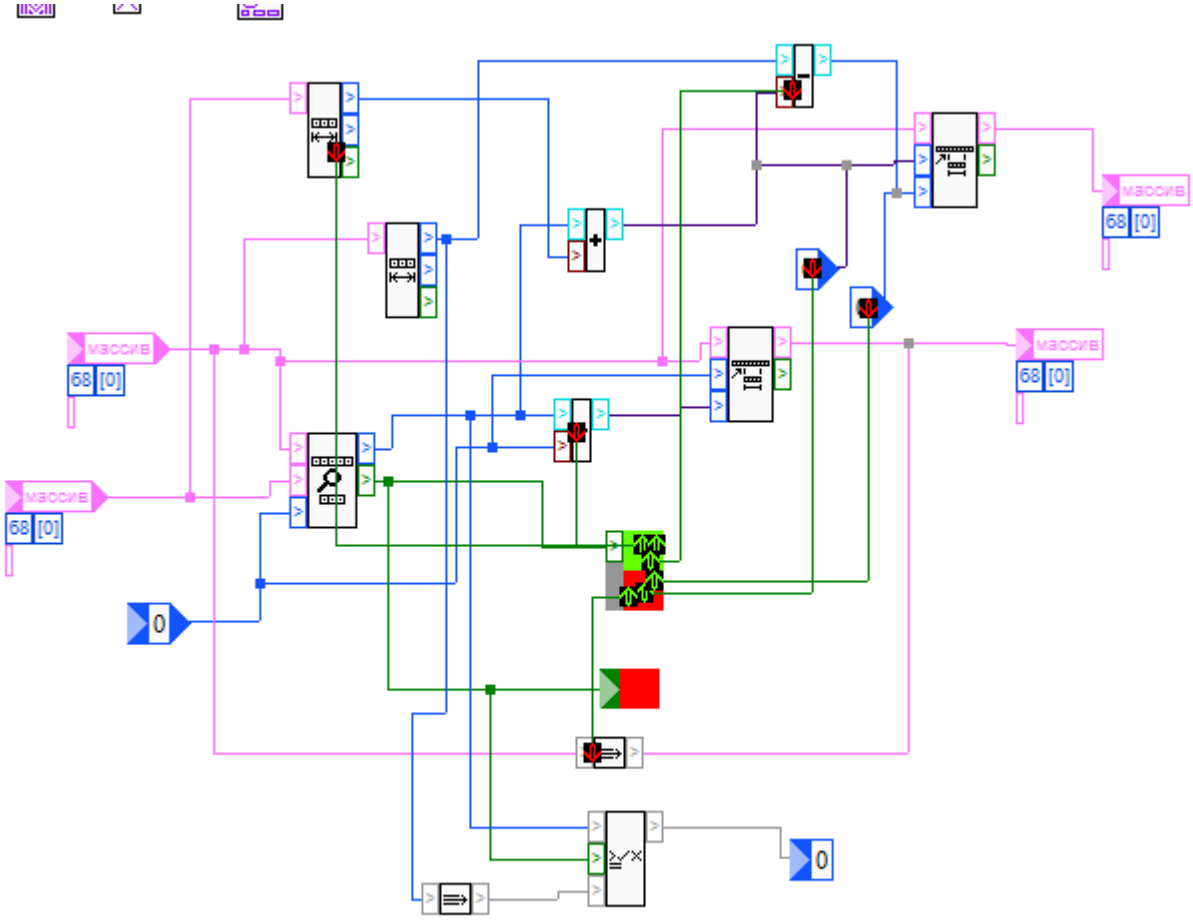

Рисунок 2.73 – Цикл рядки-підрядки

І з неї беремо «підмножину» масив з яким працює логічне порівняння. (рис. 2.74, 2.75)

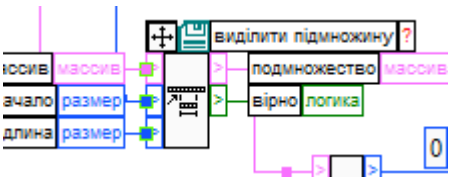

Рисунок 2.74 – Блок виділити підмножину.

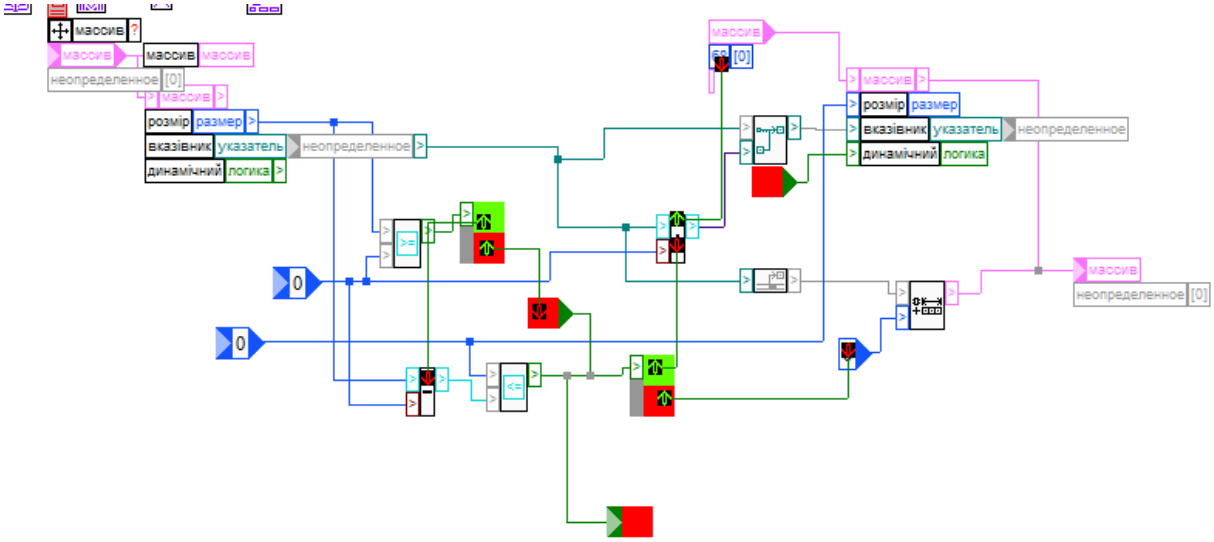

Рисунок 2.75 – Функція «виділити підмножину»

7.5 Функція «Читання з файлу»

Блок функціх «читання з файлу» виглядає наступним чином. (рис. 2.76)

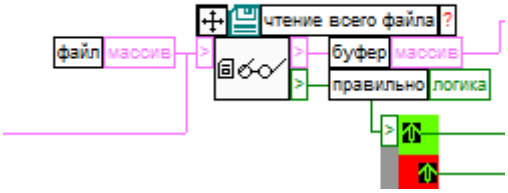

Рисунок 2.76 – блок Функції «Читання з файлу»

В даній функції ми за визначеним шляхом дійшовши до конфігу читаємо з файлу. (рис. 2.77)

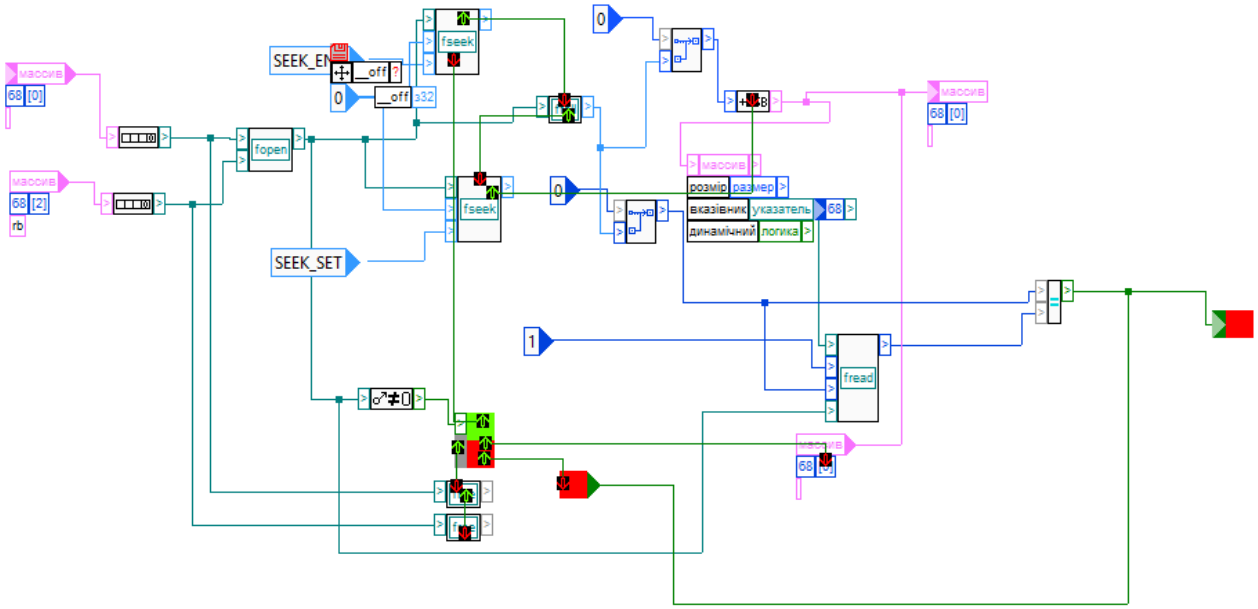

Рисунок 2.77 – Функція «Читання з файлу»

Відкриває файл стандартна функція мови С «fopen», встановлює вказівник «fseek» і читає «fread» надіслав котрим необхідні аргументи ми отримуємо конфіг.

7.6 Функція «Версіонований об’єкт в конфіг чату»

Блок функції «Версіонований об’єкт в конфіг чату» виглядає так. (рис. 2.78)

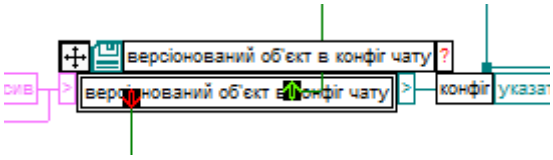

Рисунок 2.78 – Блок функції «Версіонований об’єкт в конфіг чату»

Дана функція версіонує об’єкти конфігу чату якщо той був успішно прочитаний функцією «Читання з файлу». (рис. 2.79, 2.80)

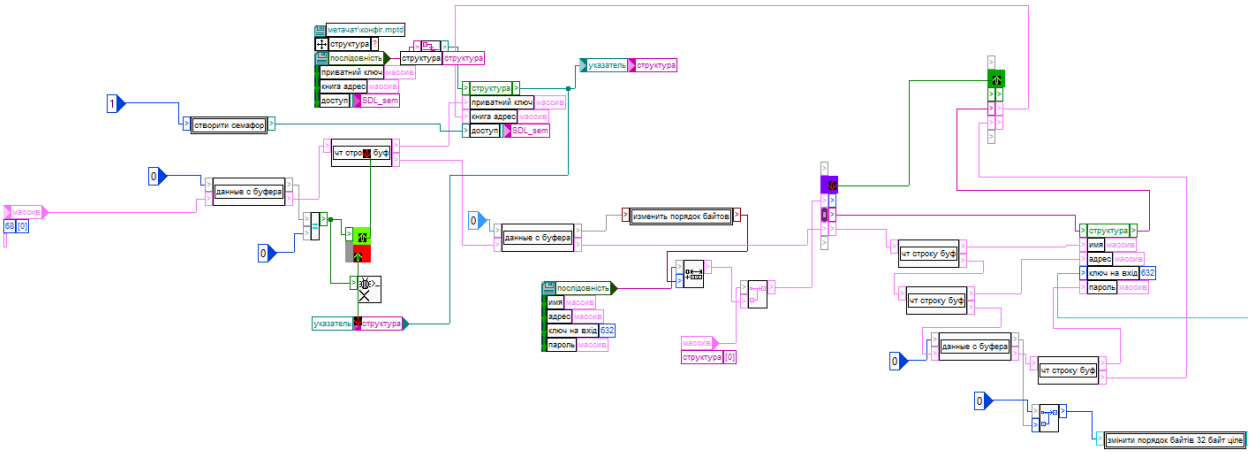

Рисунок 2.79 – Функція «Версіонований об’єкт в конфіг чату» (частина)

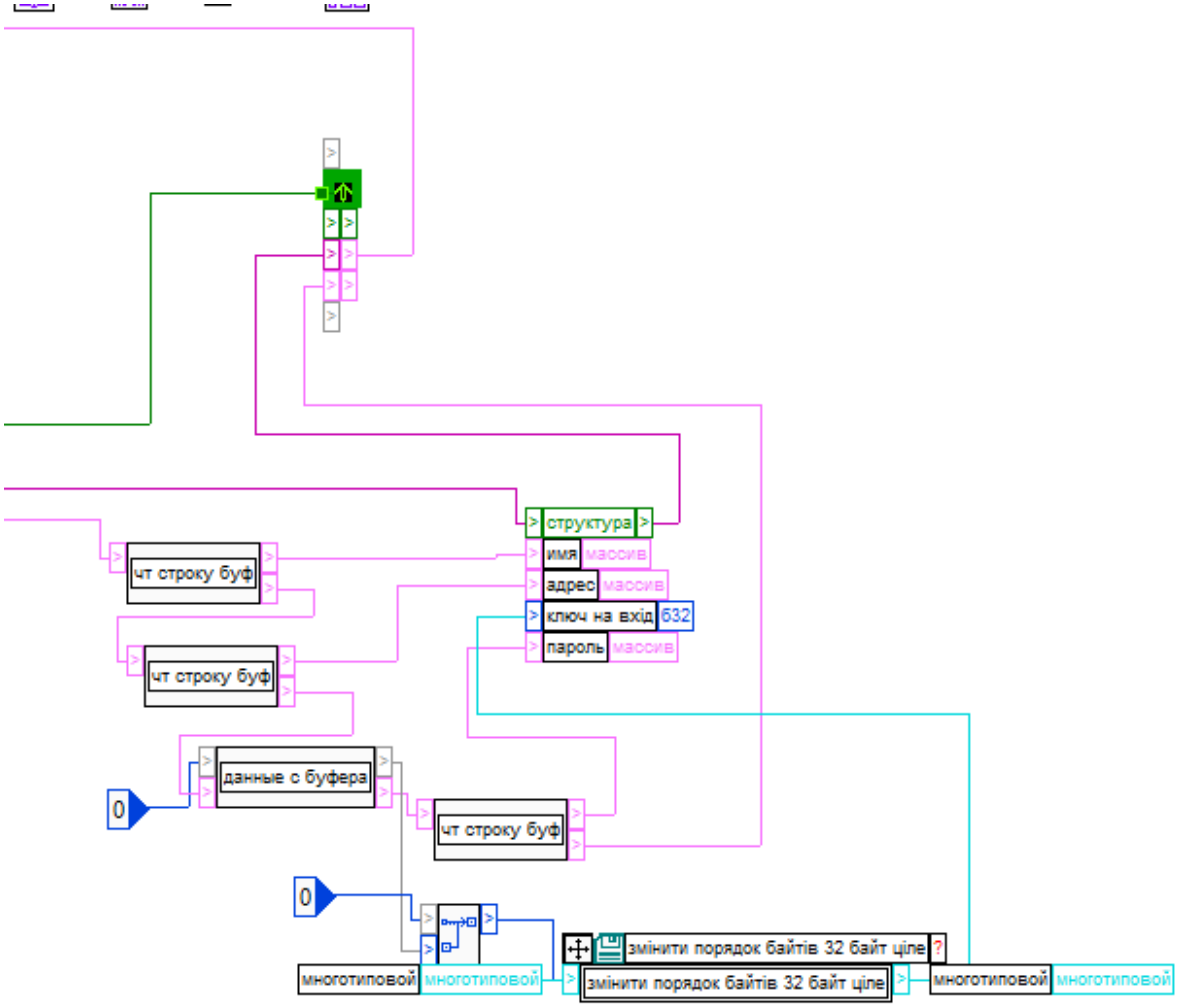

Рисунок 2.80 – Функція «Версіонований об’єкт в конфіг чату» (частина)

У функції «дані з буферу» є цікавий блок дебагу що залишився при дебагу роботи логіки у випадках коли видавало логічне НІ. (рис. 2.81, 2.82)

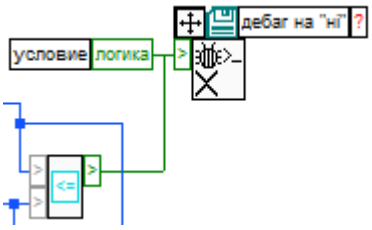

Рисунок 2.81 – Блок дебагу з’єднаний з функцією «дані з буферу»

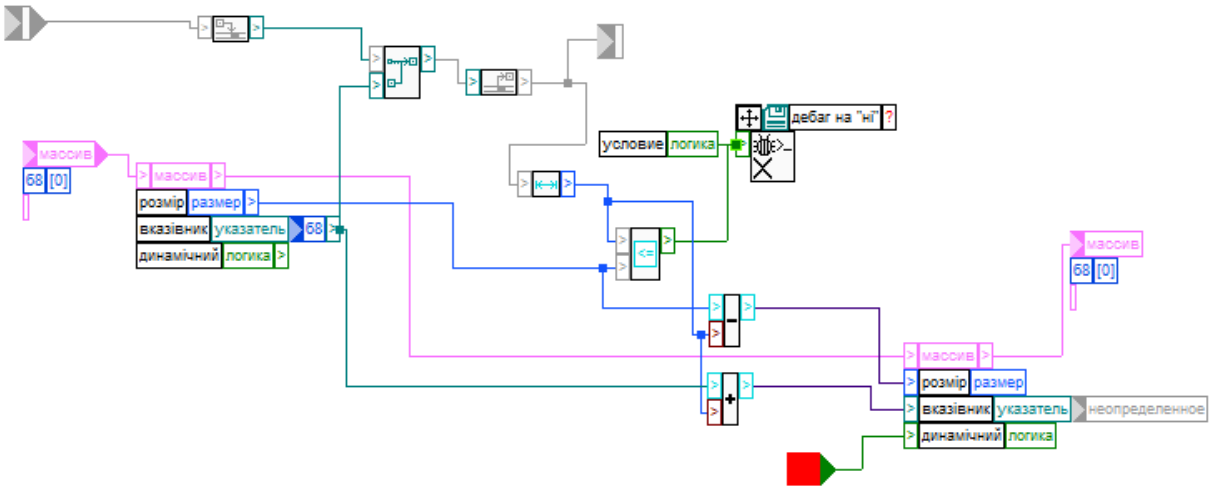

Рисунок 2.82 – Функція «дані з буферу»

7.7 Функція «тор контроль»

TOR – це децентралізована анонімна мережа яку можна використовувати як проксі. Клієнт TOR з відкритим кодом, що полегшує роботу з ним і гарантує прозорість розробці програми і відсутність backdoor, як це було з JavaAnonProxy з закритим кодом де був backdoor на вимогу німецької поліції.

TOR постачається, зокрема, у вигляді TOR browser bundel, що включає в себе спеціально адаптовану версію Firefox. Цей пакунок є зручним для користувачів без великого досвіду. В цьому пакунку TOR запускається разом з браузером і надає socks5 proxy на порті 9150. За замовчуванням TOR проводить

трафік через три вузли, що утримуються добровольцями (цей вузол може піднять кожен).

Крім анонімного доступу до звичайного інтернету мережа TOR надає можливість швидко й безкоштовно хостити власні onion-адреси. Ці адреси хостятся безкоштовно і можуть бути підняті анонімно будь яким клієнтом, підключеним до мережі TOR. Адреса починає працювати приблизно за хвилину і має вигляд 56-ти символьного ідентифікатору. Через це адреси в техночаті мають вигляд:
technochat_address:lo2q7tozea633d42zxw4jgyjq6cqaqcohdbjvahffuosmvrj3y2uz5id:technochat_address

В цій адресі адреса знаходиться між двох «technochat_address», відділених двокрапками. Ця адреса унікальна і може бути піднята тільки за приватним ключем, який генерується при підняті нової адреси. Далі цей ключ можна запам’ятати і використовувати повторно для підняття тієї самої адреси. Це використовується в техночаті: приватний ключ записується в конфіг після першого підняття адреси. Без приватного ключа неможливо підняти певну адресу. Генерація адрес і ключів здійснюється алгоритмом ED-25519. Даний алгоритм для onion-адрес прийшов на заміну застарілому RSA-1024, що використовувався раніше.

У TOR є спеціальний контрольний протокол. Через нього можна, зокрема, підіймати onion-адреси.

Блок функції «тор контроль», відповідальний за відкриття контрольного з’єднання виглядає так. (рис. 2.83)

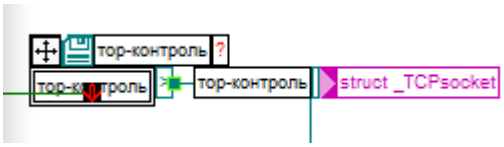

Рисунок 2.83 – Блок функції «тор контроль»

Дана функція відповідає за роботу з тором й onion-адресами. Вона проводить підключення що на момент написання можливе лише при включені TOR browser або (контрольне з’єднання - порт 9151). (рис. 2.84 – 2.85)

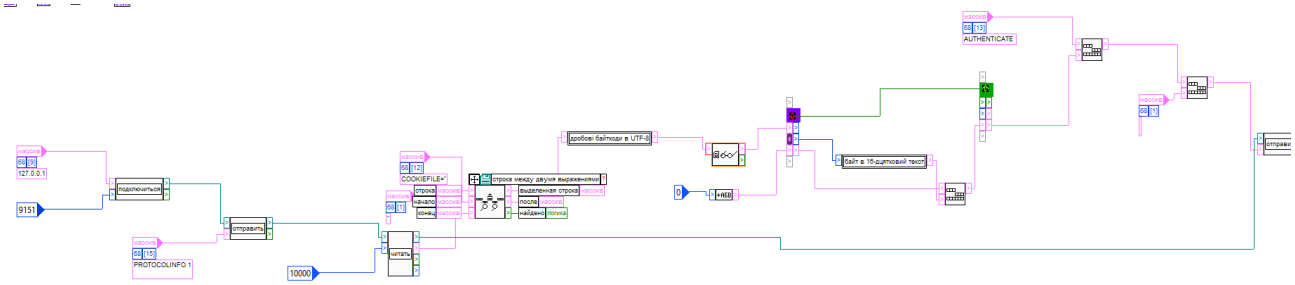

Рисунок 2.84 – Функція «тор контроль» (частина)

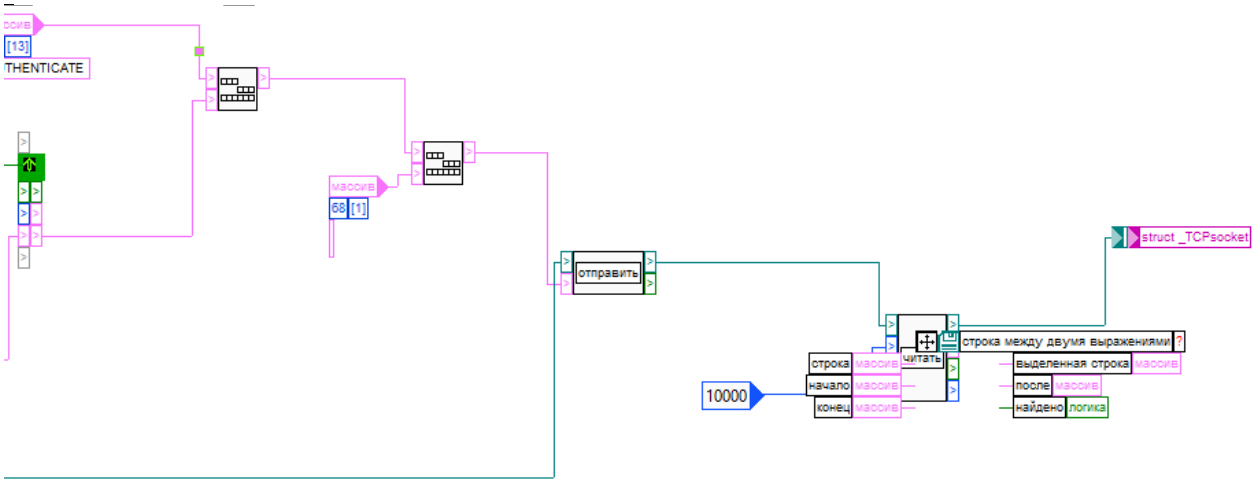

Рисунок 2.85 – Функція «тор контроль» (частина)

Саме підключення йде до вказаного порту і реалізовану функцією підключення. (рис. 2.86, 2.87)

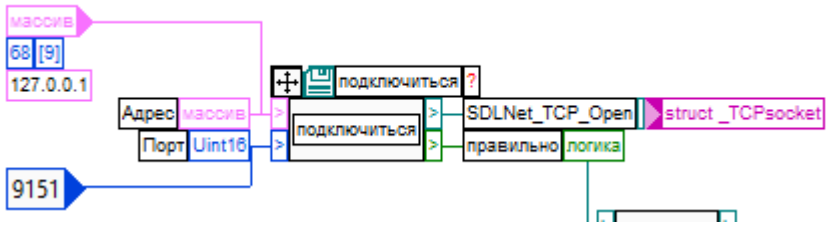

Рисунок 2.86 – Блок функції «Підключиться»

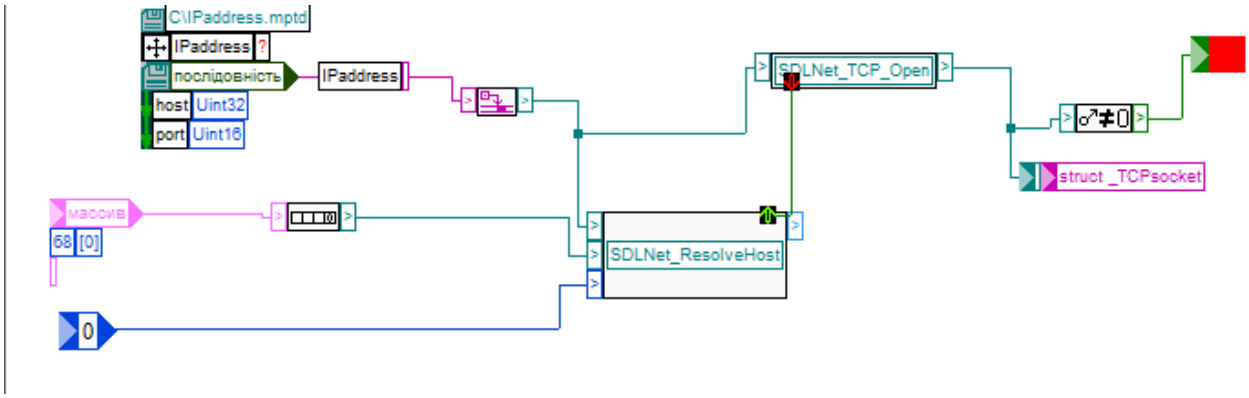

Рисунок 2.87 – Функція «Підключиться»

Для обміну інформацією було використано TCP socket відправка котрих основана на функції мови С «SDLNet_TCP_Send». (рис. 2.88 – 2.89)

Рисунок 2.88 – Блок функції «Відправити»

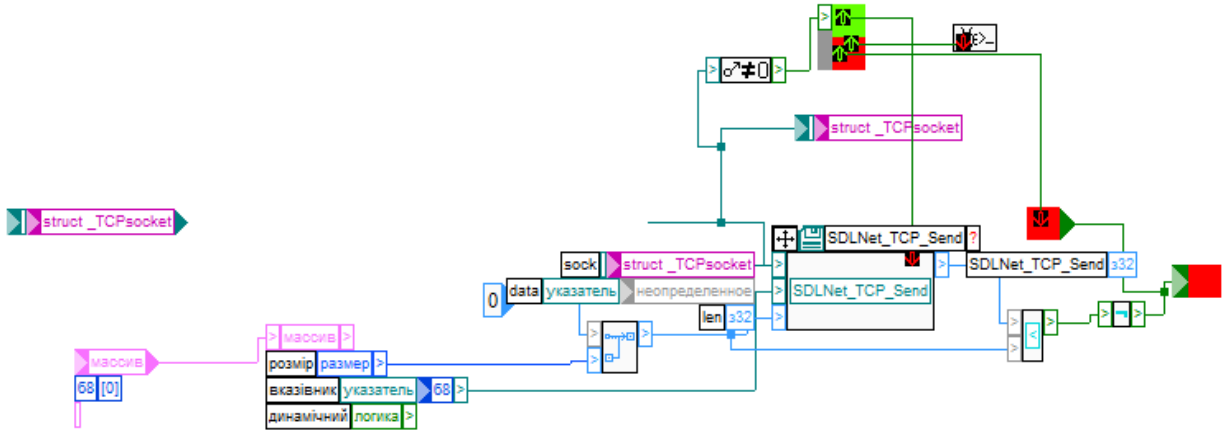

Рисунок 2.89 – Функція «Відправити»

Далі беручи її з буферу обміну вона прочитується за допомогою «SDLNet_TCP_Recv». (рис. 2.90 – 2.91)

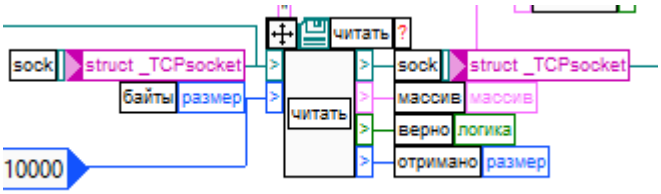

Рисунок 2.90 – Блок функції «Читати»

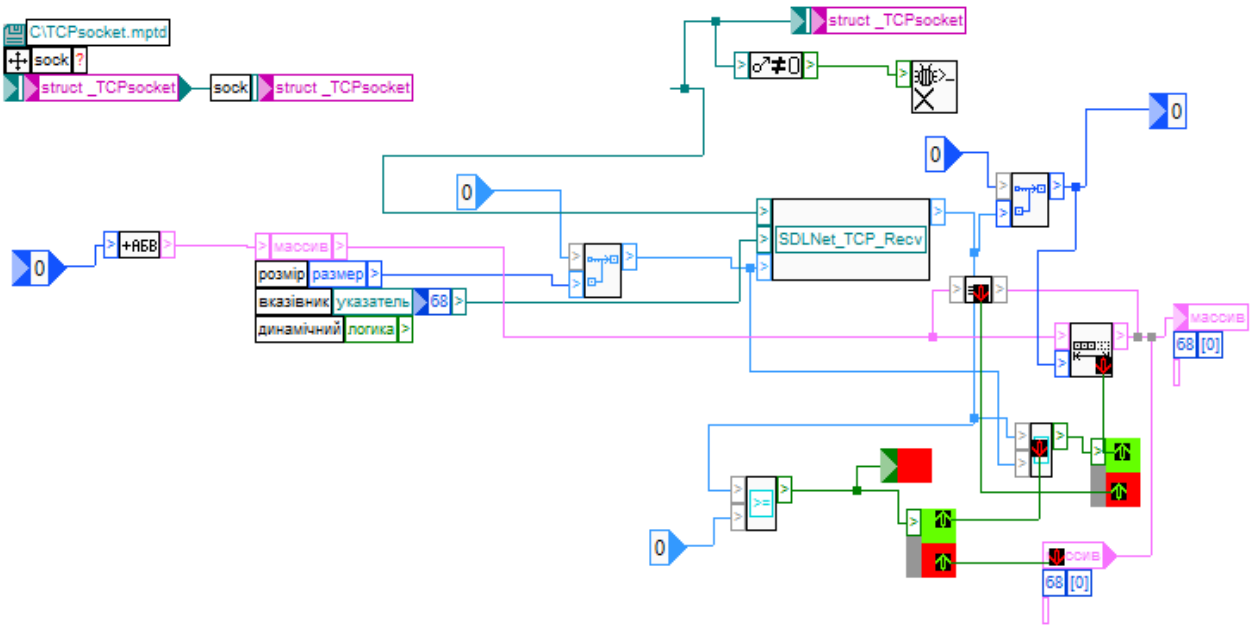

Рисунок 2.91 – Функція «Читати»

Наступним кроком є робота з підрядками до й після входження до масиву. (рис. 2.92, 2.93)

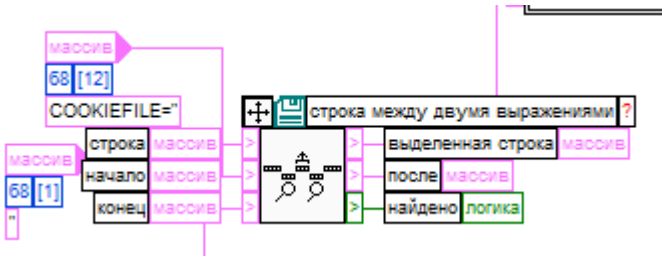

Рисунок 2.92 – Блок функції «строка между двумя выражениями»

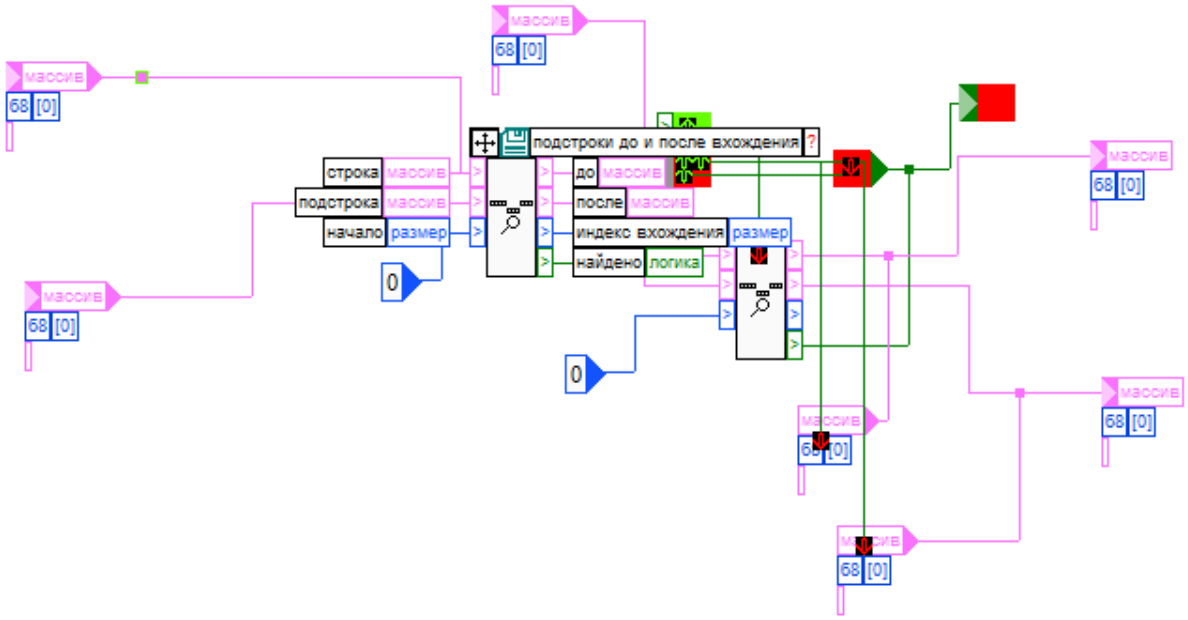

Рисунок 2.93 – Функція «строка между двумя выражениями»

7.8 Функція «дробові байткоди з UTF-8»

Блок функції «дробові байткоди» виглядає наступним чином. (рис. 2.94)

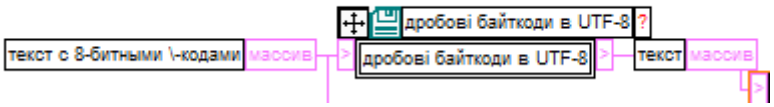

Рисунок 2.94 – Блок функції «дробові байткоди з UTF-8»

Ця функція реалізує необхідна для роботи з контрольним протоколом. В одній зі стадій відкриття контрольного socket TOR клієнт відповідає текстом які деякі символи, зокрема кириличні, кодуються символом \ та трьома вісімковими числами на байт.

Для отримання тексту ми в циклі серіалізуємо дані (байти в структури і навпаки – десиріалізація). У нашому випадку серіалізація супроводжується злиттям до масиву байтів на виході. (рис. 2.95, 2.96)

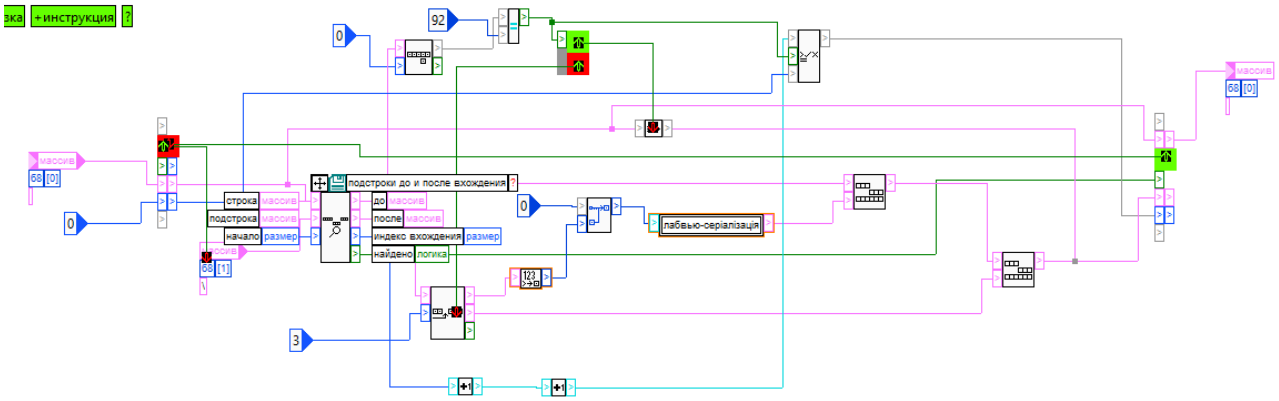

Рисунок 2.95 – Функція «дробові байткоди з UTF-8» (частина)

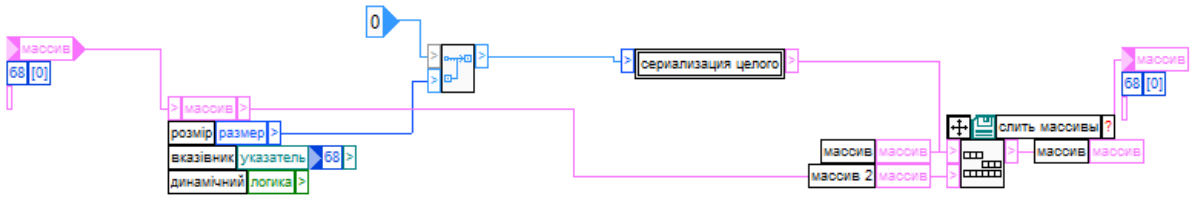

Рисунок 2.96 – Функція «дробові байткоди з UTF-8» (частина)

7.9 Функція «Байти в 16-дцяткове число»

Блок функції «байти в 16-дцяткове число». (рис. 2.97)

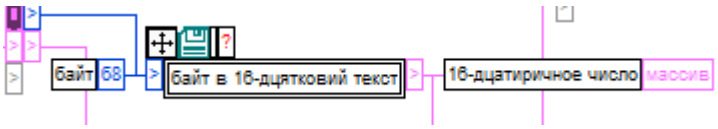

Рисунок 2.97 – Блок функції «Байти в 16-дцяткове число»

Про цю функцію та деякі інші вже описано вище (пункт «Підняття TOR onion»). (рис. 2.98)

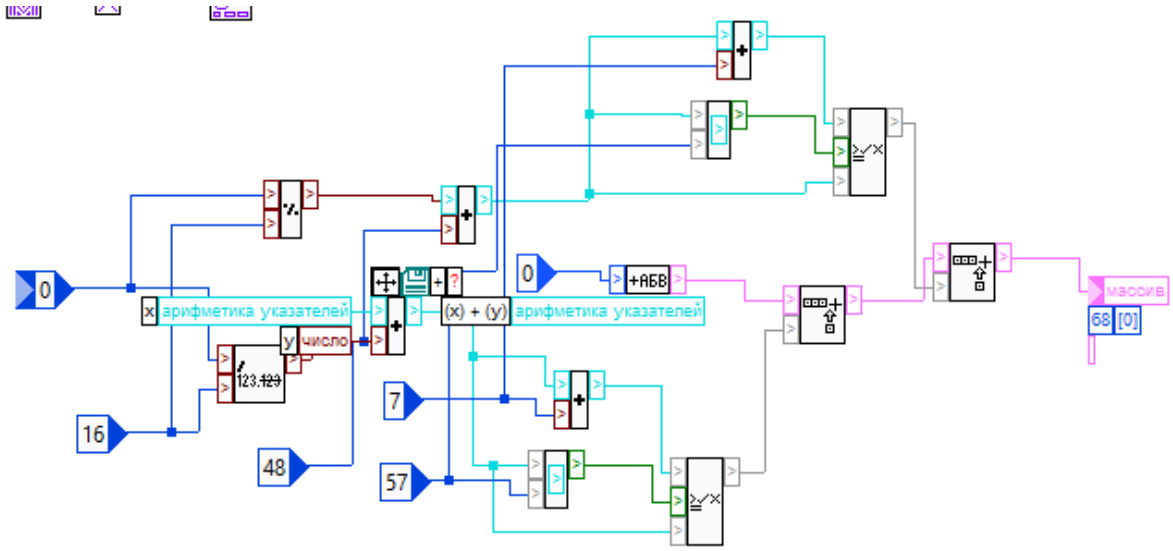

Рисунок 2.98 – Функція «байти в 16-дцяткове число»

8. Функція «Сформувати нове підвікно»

Блок функціх «Сформувати нове підвікно». (рис. 2.99 – 2.101)

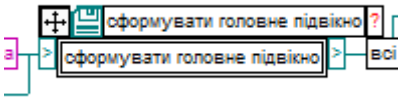

Рисунок 2.99 – Блок функції «Сформувати нове підвікно»

Ця функція відповідає за формування головного вікна і його елементів. Про більшість з них вже було згадано (віджети, текст).

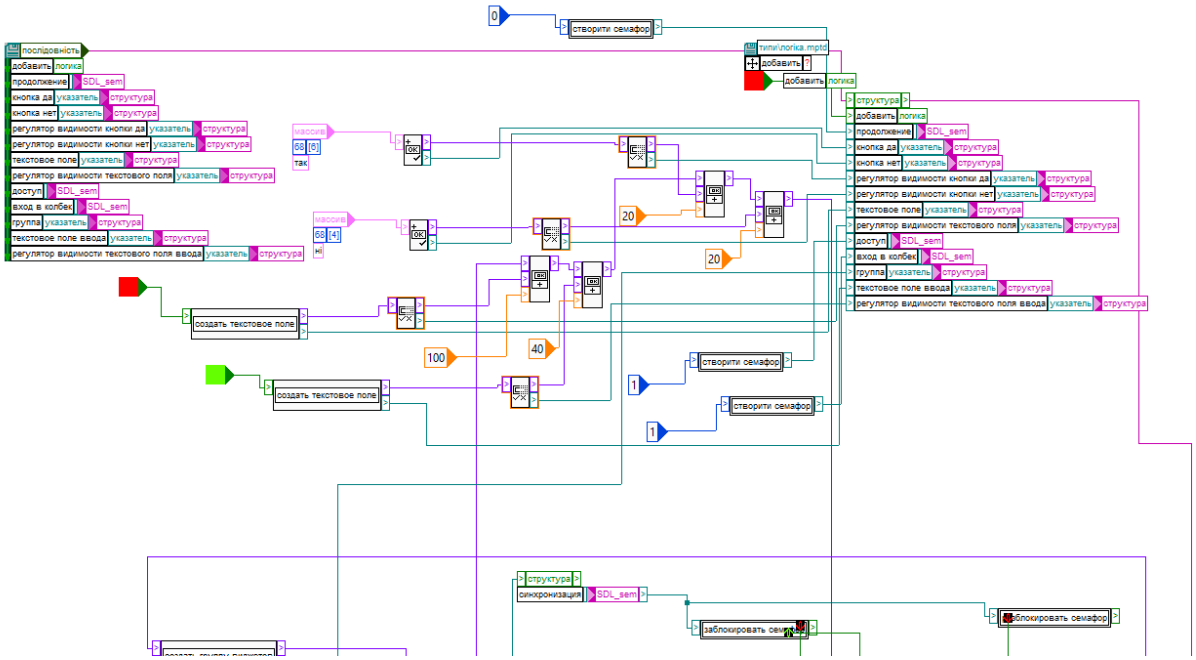

Рисунок 2.100 – Функція «Сформувати нове підвікно» (частина)

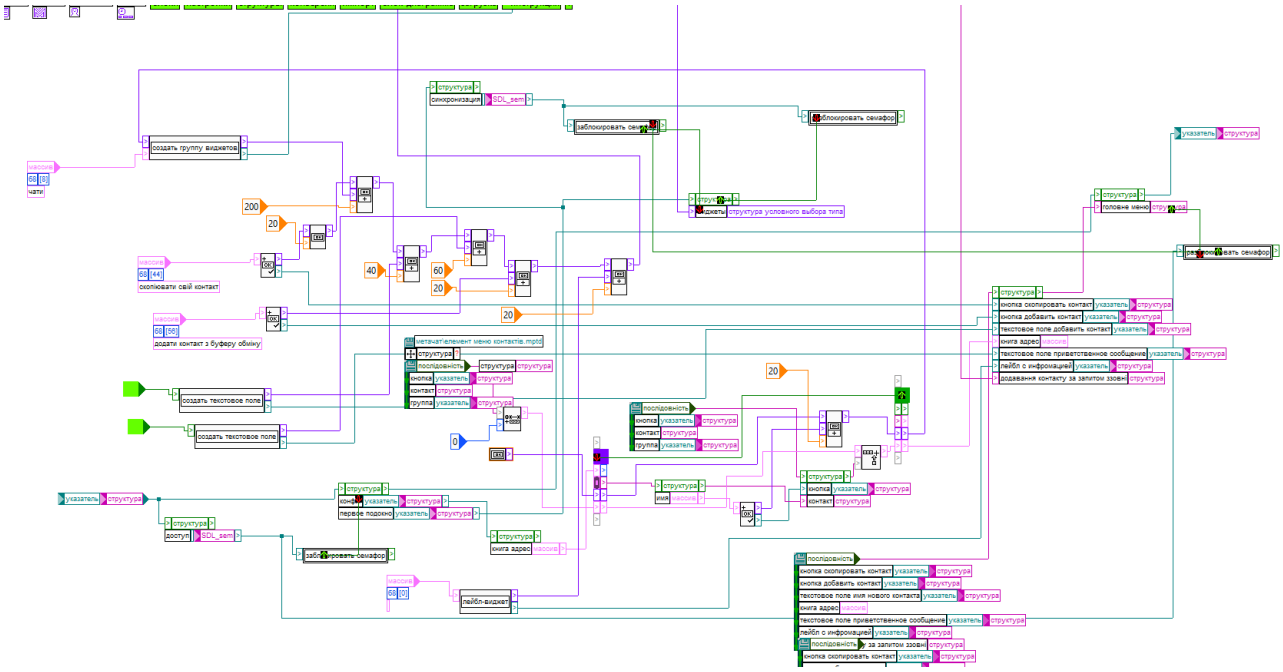

Рисунок 2.101 – Функція «Сформувати нове підвікно» (частина)

8.1 Функція «Додавання елементу в колонку»

Функція «Додавання елементу в колонку» виглядає так. (рис. 2.102)

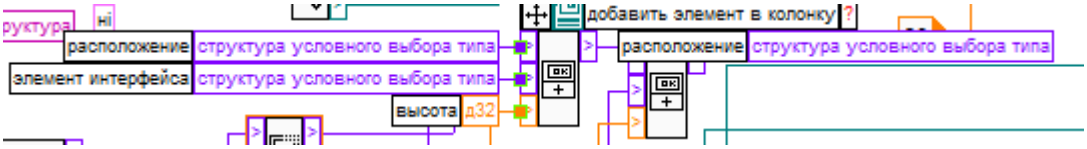

Рисунок 2.102 – Блок функції «Додавання елементу в колонку»

Функція «Додавання елементу в колонку» додає віджет до групи, якщо група є вертикальною колонкою.

За СУВТ (Структурою Умовного Вибору Типу) дана функція за вказівником додає елементи. (рис. 2.103)

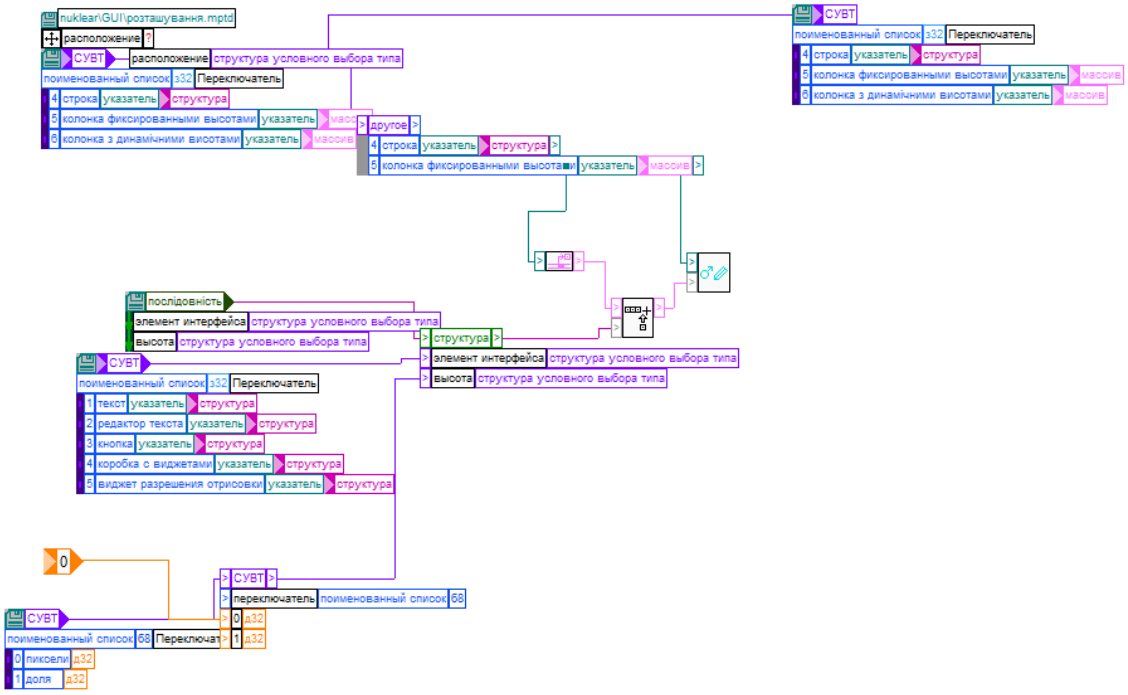

Рисунок 2.103 – Функція «Додавання елементу в колонку»

8.2 Функція «лейбл-віджет»

Блок функції «лейбл віджет» виглядає наступним чином. (рис. 2.104)

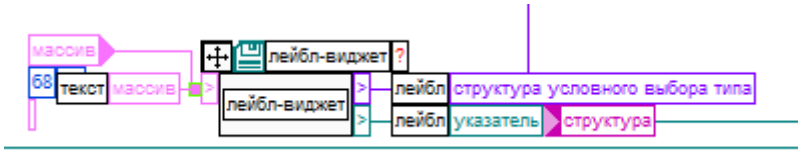

Рисунок 2.104 – Блок функції Функція «лейбл-віджет»

Інформація про наявність й використання лейблів є в пункті 6.3 про блок «АБВ». У даному випадку функція за даними з СУВТ малює необхідні лейбл-віджети. (рис. 2.105)

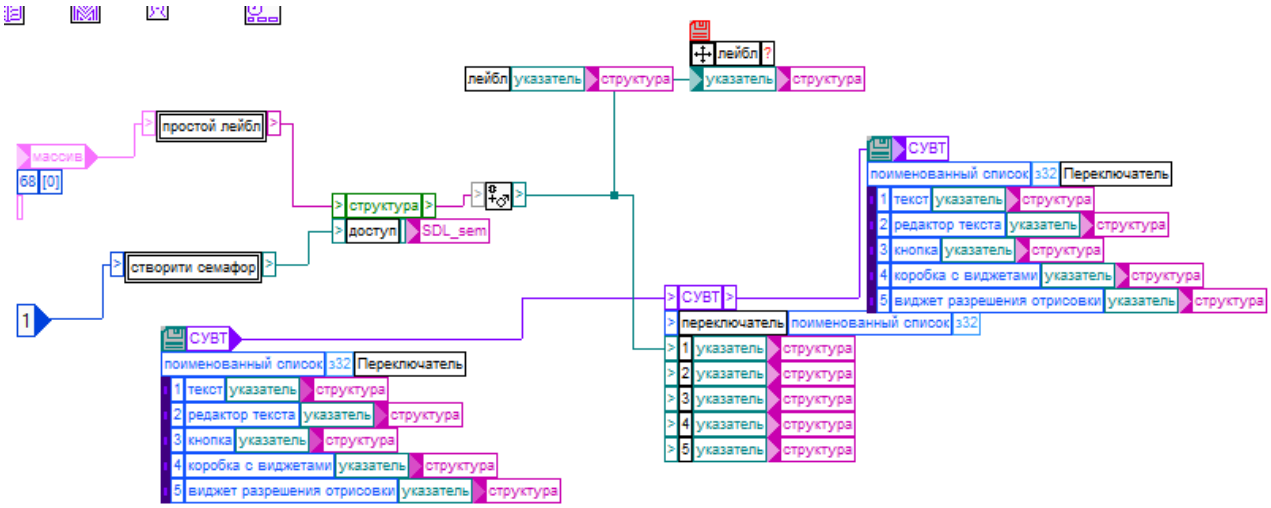

Рисунок 2.105 – Малюваня по СУВТ

Використовується наступний шаблон «Просто лейбл» зі всіма даними з СУВТ. (рис. 2.106, 2.107)

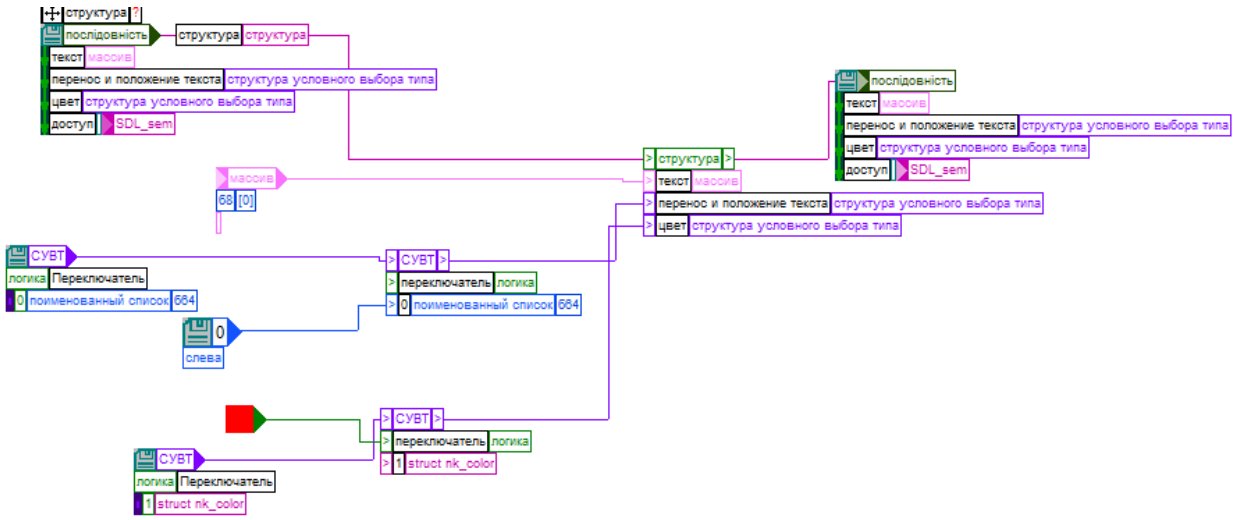

Рисунок 2.106 – шаблон «Просто лейбл»

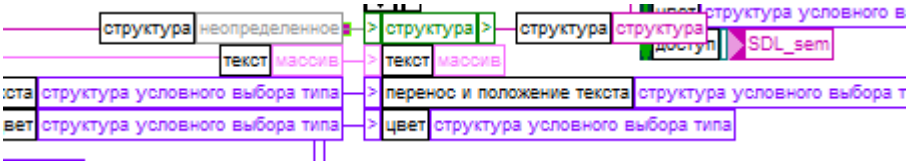

Рисунок 2.107 – Приклад аргументів СУВТ.

9. Функція «Підняти слухача оніон»

Блок функції «Підняти слухача оніон» виглядає наступним чином. (рис. 2.108)

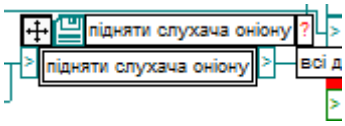

Рисунок 2.108 – Блок функції «Підняти слухача оніон»

Техночат – це децентралізований месенджер що не має серверів, це означає що для функціонування кожен клієнт має втілювати не лише клієнтський, а і певний серверний функціонал. Піднявши onion-адресу Техночат слухає локальний порт 22001. Це дає змогу приймати запити від інших клієнтів, зокрема на додавання в контакти й надсилання повідомлення. Функція підняття на малюнку нижче запускає потік, що слухає порт й обробляє вхідні запити. (рис. 2.109)

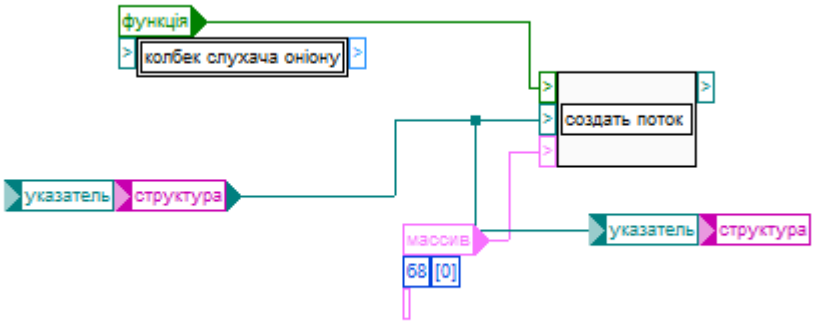

Рисунок 2.109 – Функція «Підняти слухача оніон»

Вона використовує callback слухача оніону. (рис. 2.110)

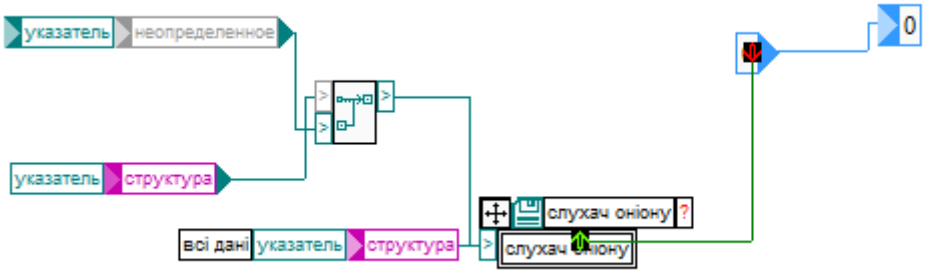

Рисунок 2.110 – Callback слухача оніону

Який в свою чергу слухає порт TCP 22001. (рис. 2.111)

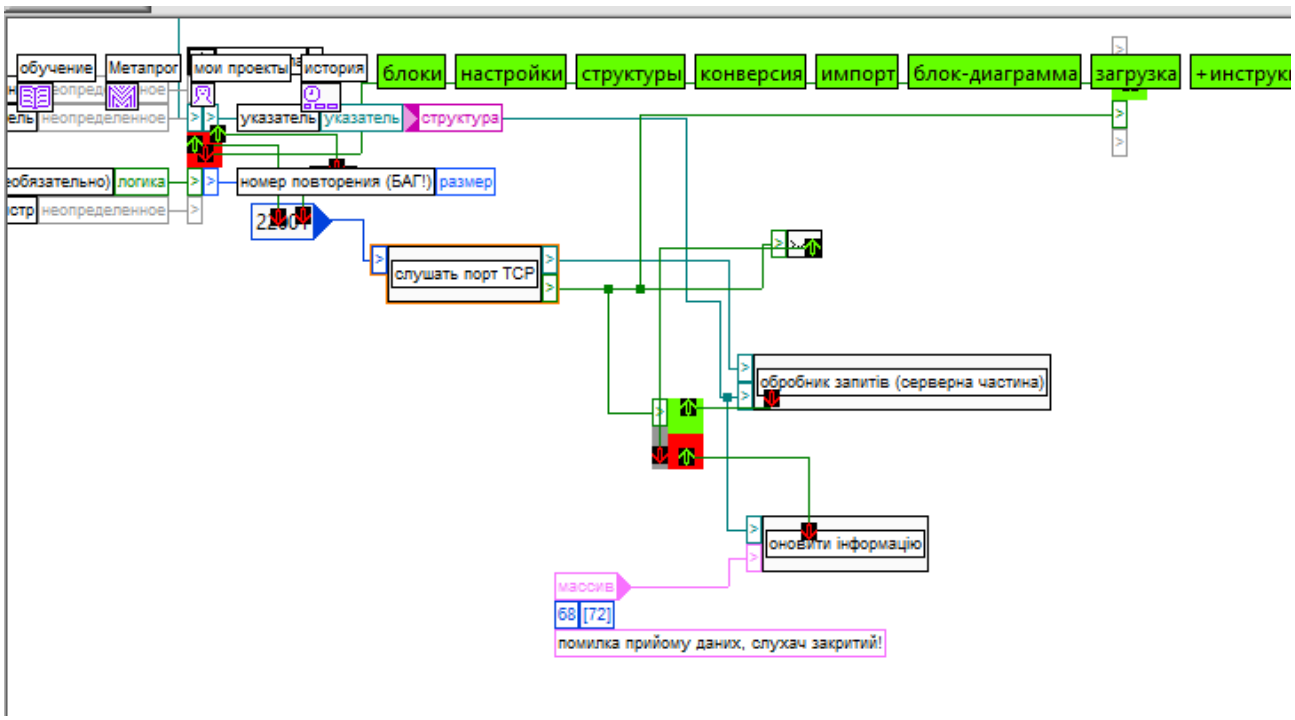

Рисунок 2.111 – Прослуховування порт TCP

10. Функція «Очікування на події інтерфейсу»

Блок функції «Очікування на події інтерфейсу» виглядає наступним чином. (рис. 2.112)

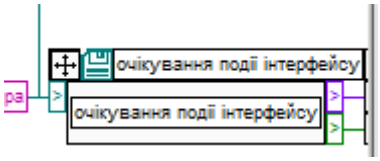

Рисунок 2.112 – Блок функції «Очікування на події інтерфейсу»

Функція циклічно бере події з інтерфейсу й відповідно до дій реалізовує їх в графічному інтерфейсі. (рис. 2.113)

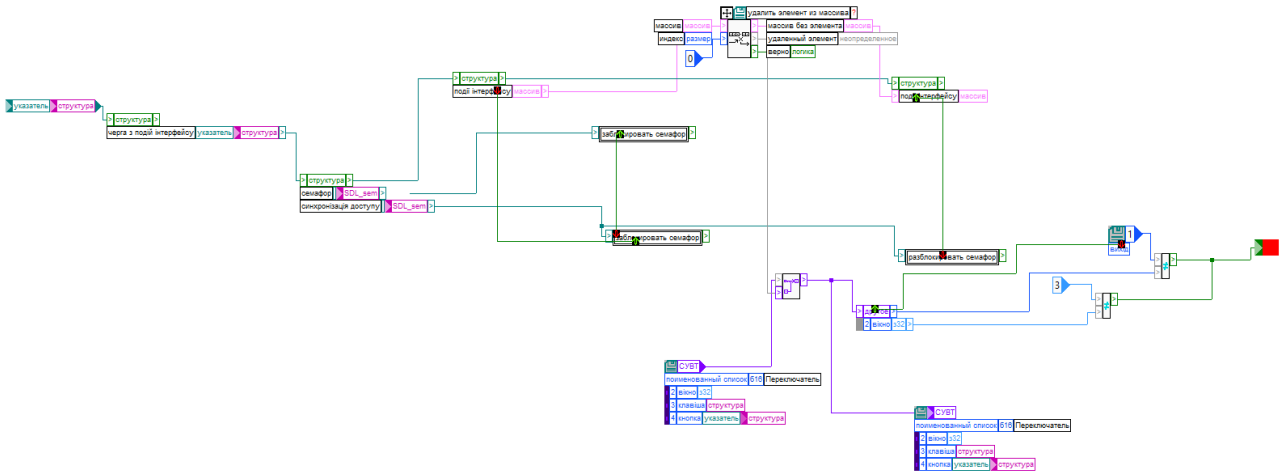

Рисунок 2.113 – Функція «Очікування на події інтерфейсу»

11. Функція «запис до буферу обміну»

Коли користувач хоче дізнатися свій id йому достатньо клікнути на кнопку і тоді в його буфер обміну внесеться його id. Звичайно це можливо зробити лише якщо був інтернет, працював TOR і конфіг отримав його id. (рис. 2.114)

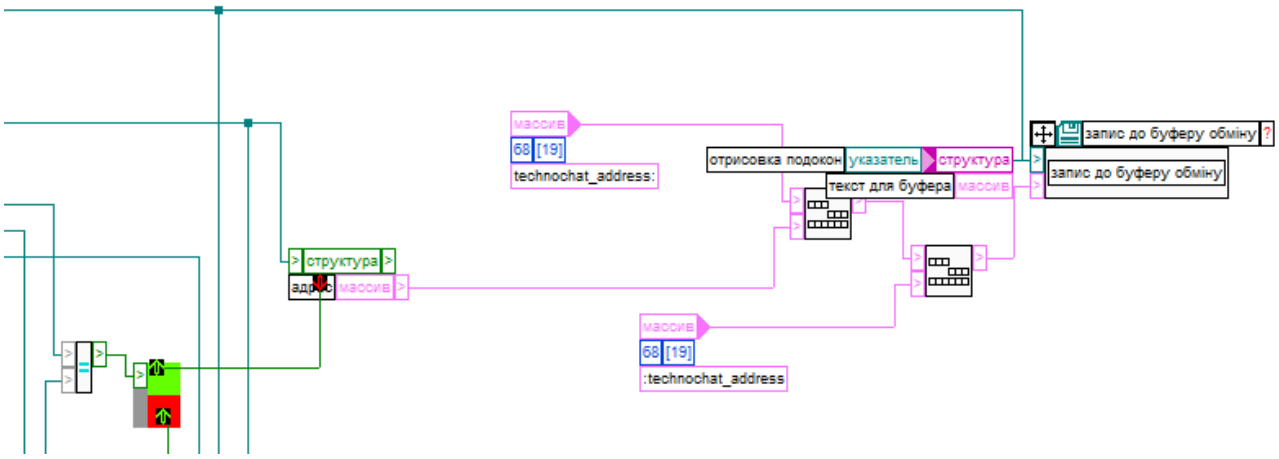

Рисунок 2.114 – Функція «запис до буферу обміну»

12. Додавання вхідних запитів

Після отримання вхідного запиту в головоному підвікні з’являється відповідний діалог додавання контакту. Користувач може відхилити новий контакт ибо призначити йому ім’я та додати.

Ім’я задає сам користувач. (рис. 2.115)

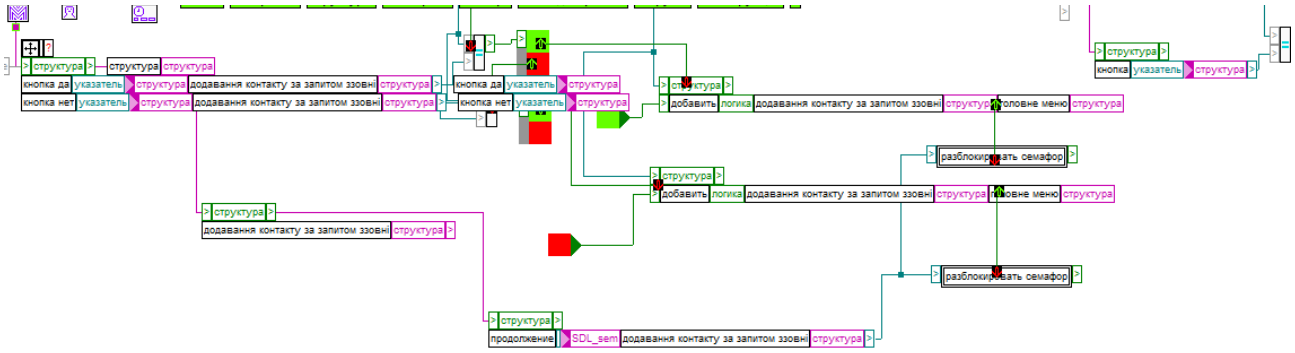

Рисунок 2.115 – Функція «запис до буферу обміну»

13. Функція «відкрити вікно чату»

Блок функції «відкрити вікно чату» виглядає наступним чином (рис. 2.116)

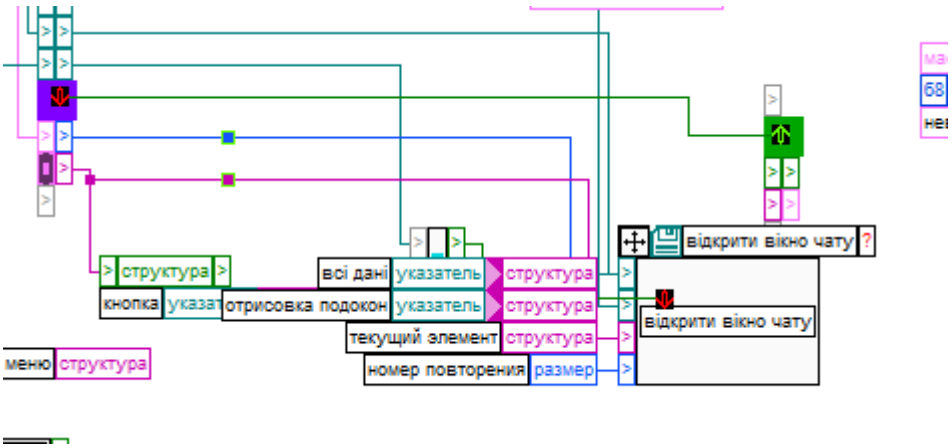

Рисунок 2.116 – Блок функції «відкрити вікно чату»

Після цього можна відкрити вікно чату разом з контактом. (рис. 2.117)

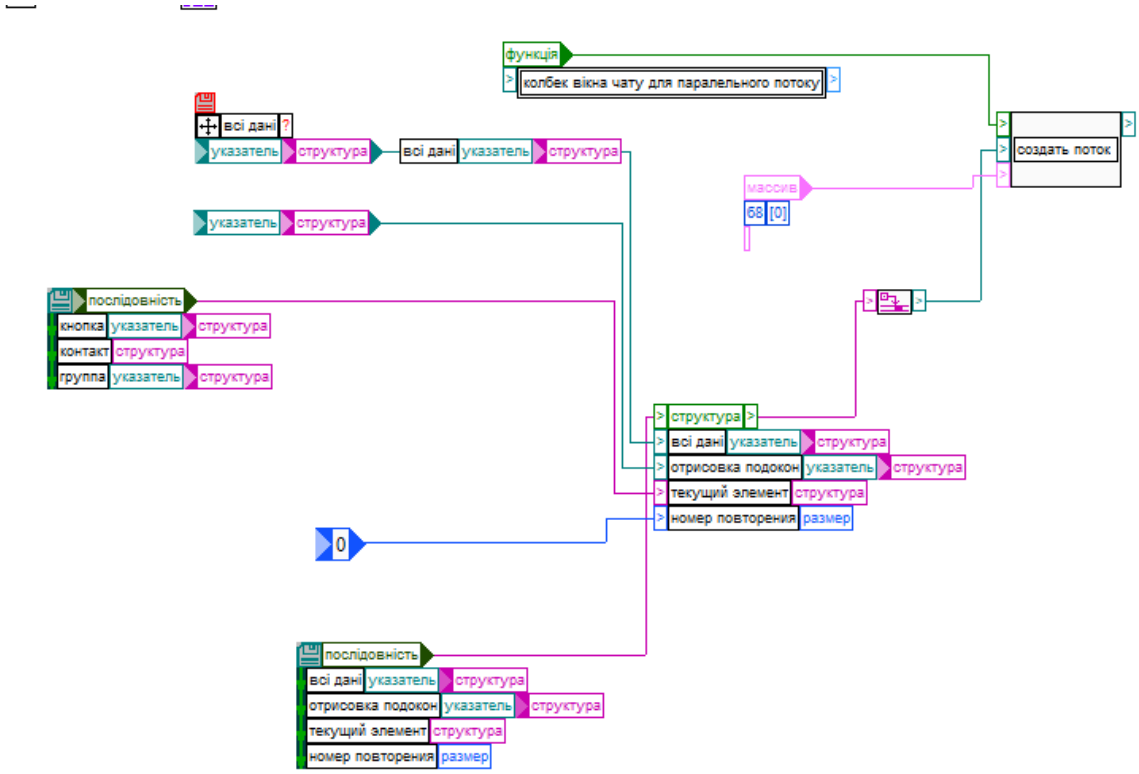

Рисунок 2.117 – Функція «відкрити вікно чату»

14. Функція «оновити інформацію»

Блок функції «оновити інформацію» виглядає наступним чином. (рис. 2.118)

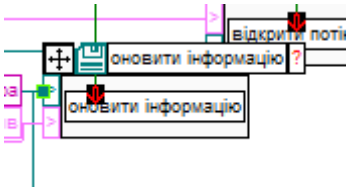

Рисунок 2.118 – Блок функції «відкрити вікно чату»

Дана функція оновлює інформацію в інтерфейсі. Очевидно, що після таких дій як додавання контактів необхідно додати сам контакт у меню і заставити лейбл робити за оновленою структурою, що і робить ця функція. (рис. 2.119)

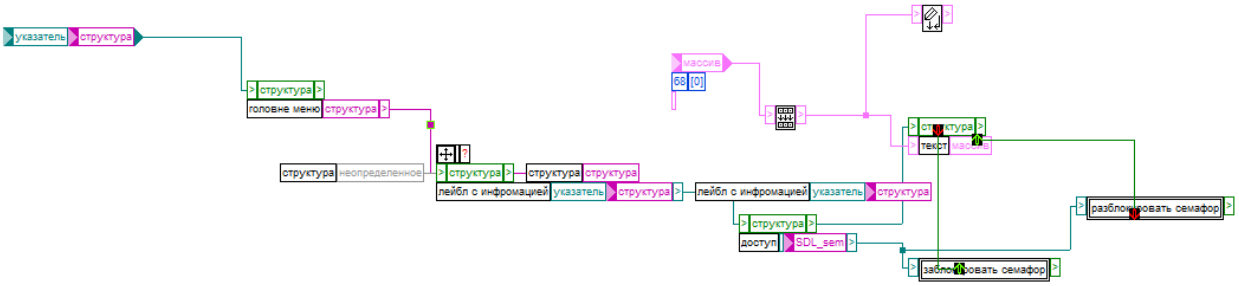

Рисунок 2.119 – Функція «оновити інформацію»

14.1 Функція «клонувати масив»

Блок функції «клонувати масив» виглядає так. (рис. 2.120)

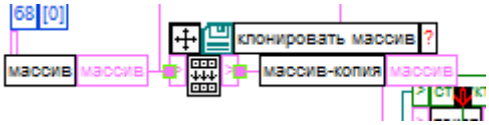

Рисунок 2.120 – Блок функції «клонувати масив»

Цікавим тут є блок клонування масивів, що використовує функцію мови С «memcpy» після надання аргументів вказівника й виділення пам’яті клонує масив. (рис. 2.121)

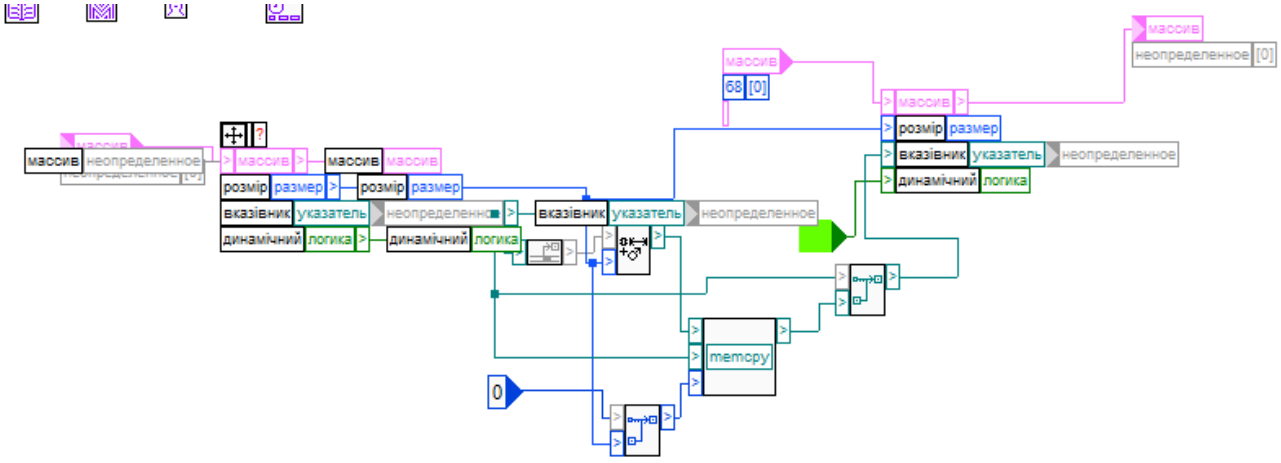

Рисунок 2.121 – Функція «клонувати масив»

15. Функція «відкрити потік надсилання запиту на додавання»

Блок функції «відкрити потік надсилання запиту на додавання» виглядає так. (рис. 2.122)

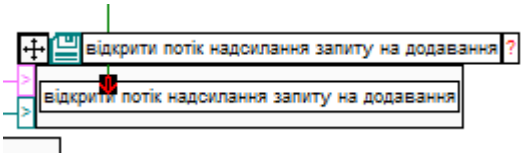

Рисунок 2.122 – Блок функції «клонувати масив»

Отримавши структура даних й id користувача з буферу обміну дана функція відкриває потік використовуючи вище зазначену функцію створення нового потоку. (рис. 2.123)

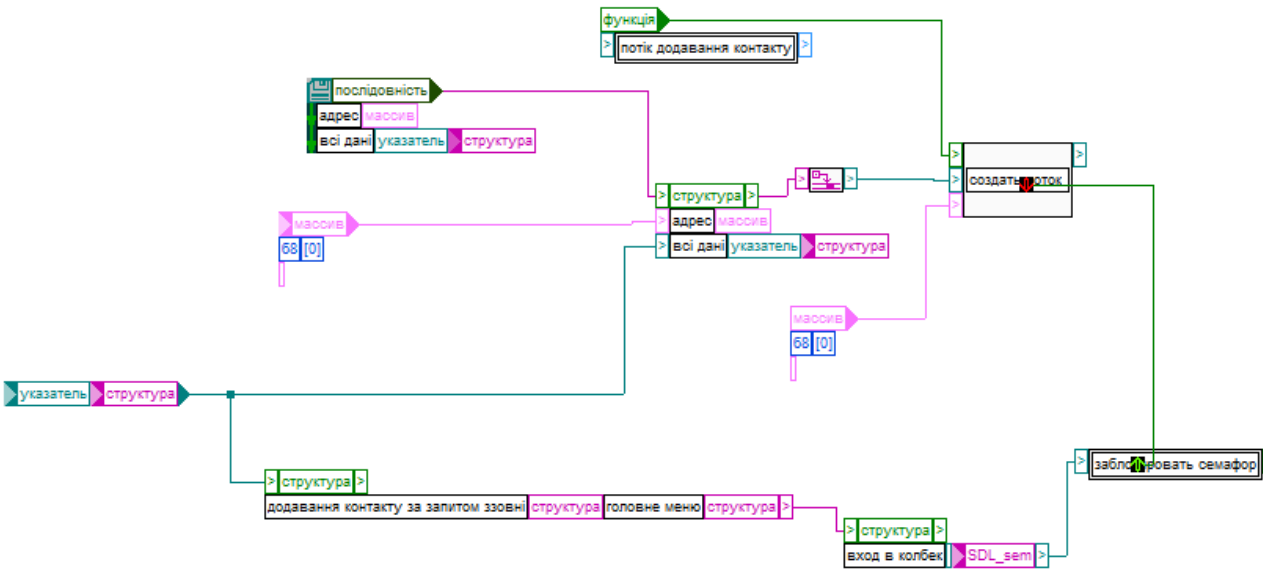

Рисунок 2.123 – Функція «клонувати масив»

16. Месенджер Техночат

Як вже було написано на початку цього розділу – бібліотека Техночату і є самим месенджером. Створивши всі необхідні функції і зібравши їх в єдину

функцію Техночату був створений сам месенджер. Виглядає він в роботі наступним чином. (рис. 2.124 – 2.126)

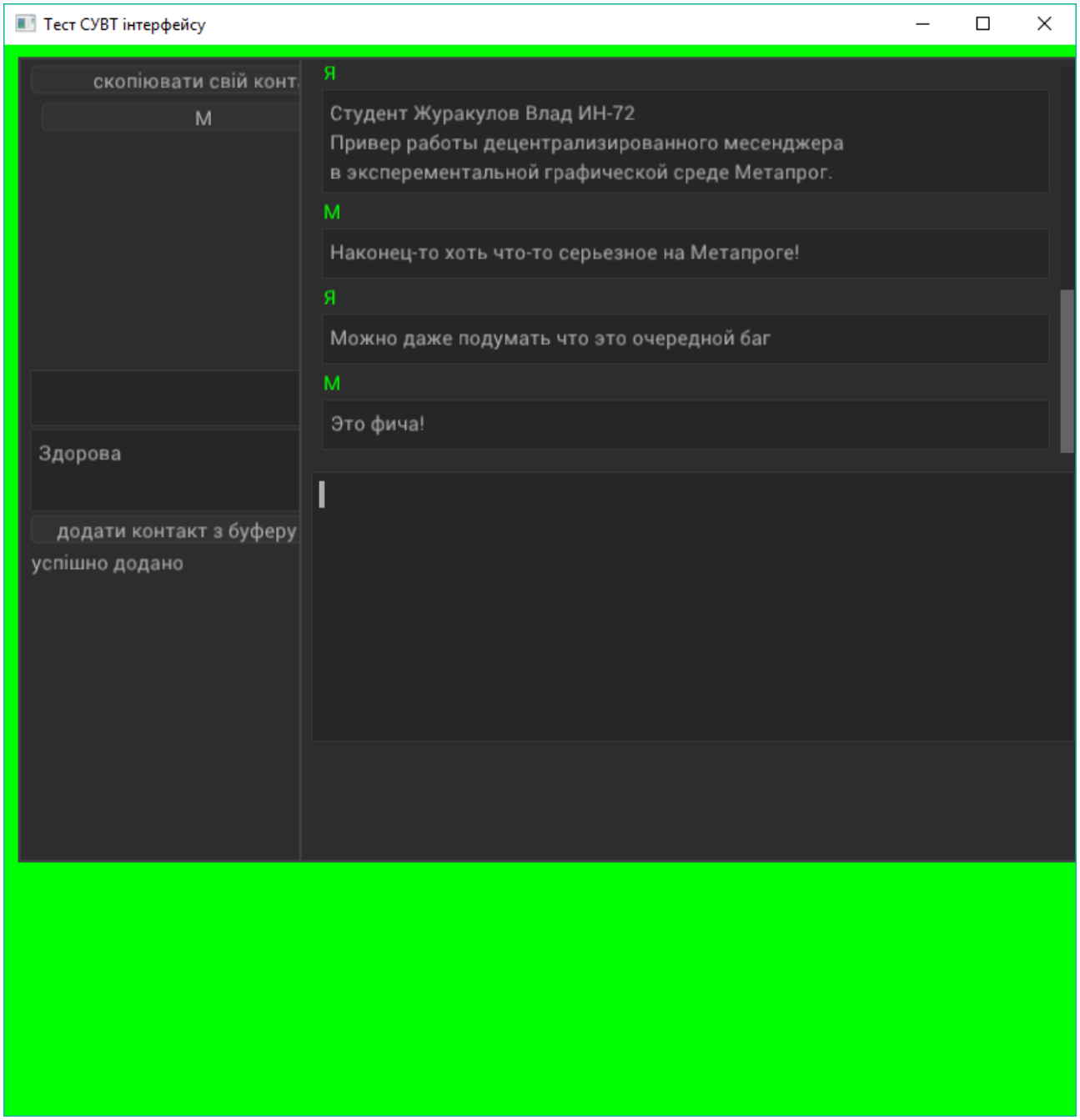

Рисунок 2.124 – Стара версія 62

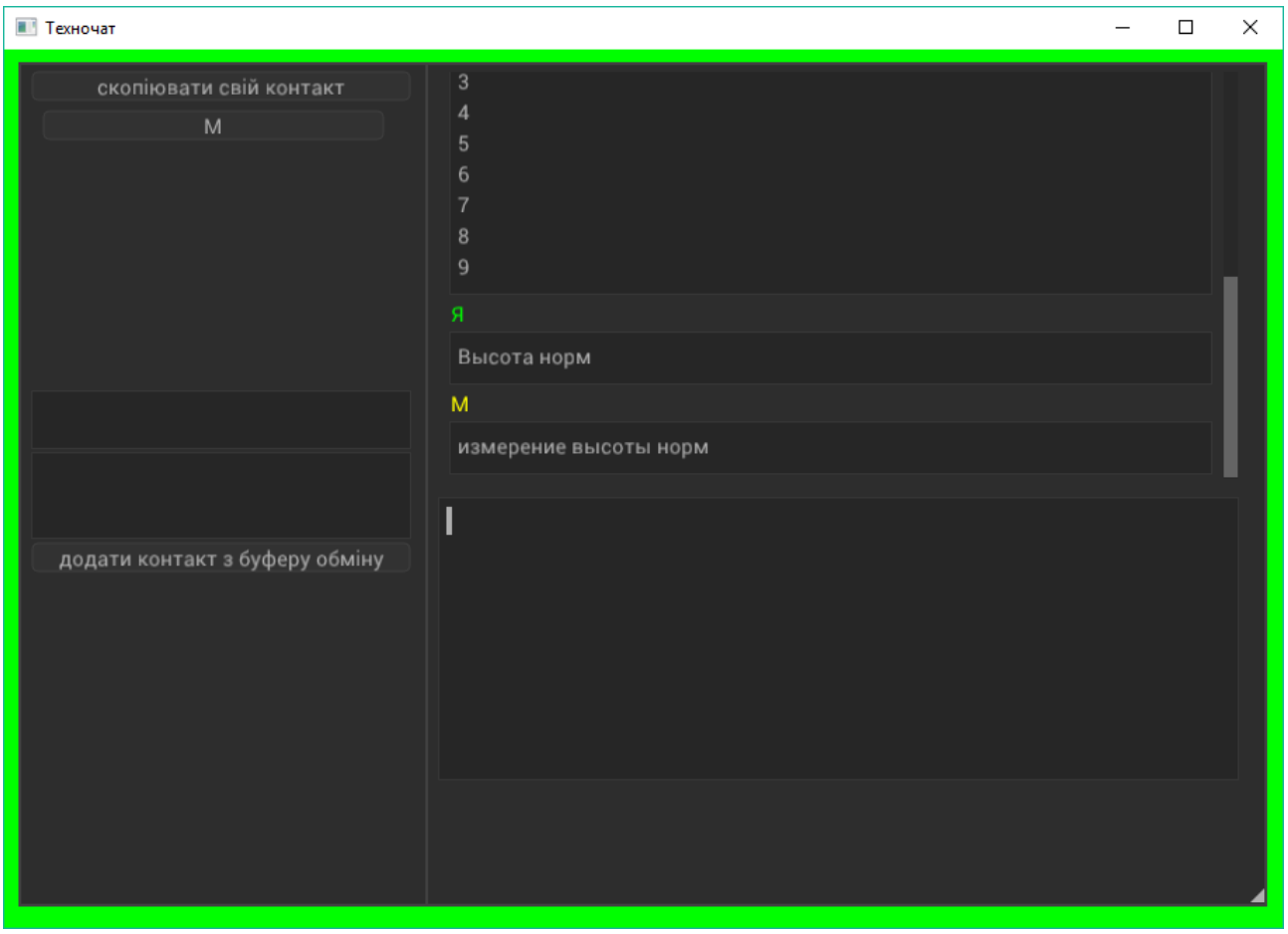

Рисунок 2.125 – Новіша версія 63

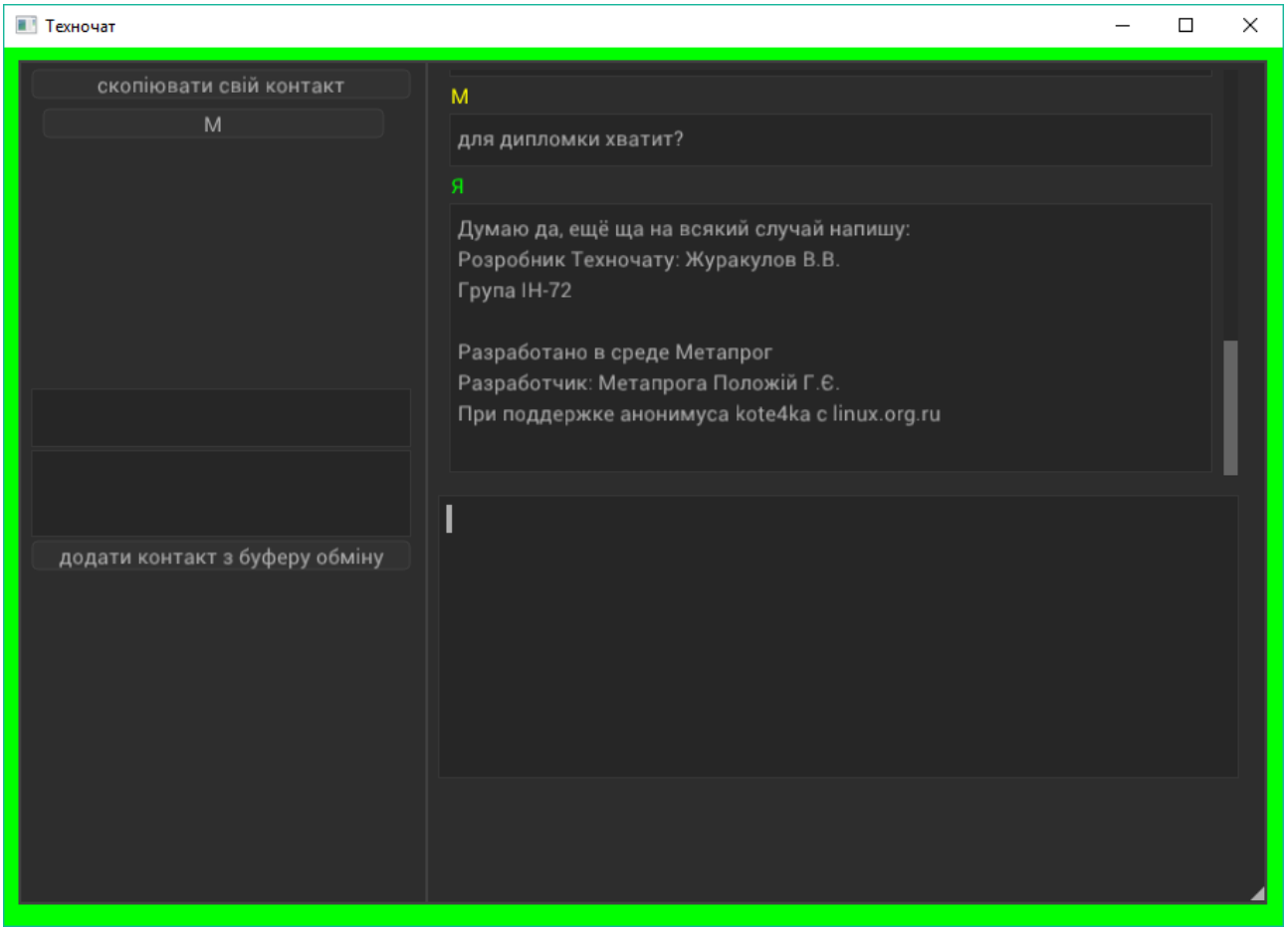

Рисунок 2.126 – Версія 63

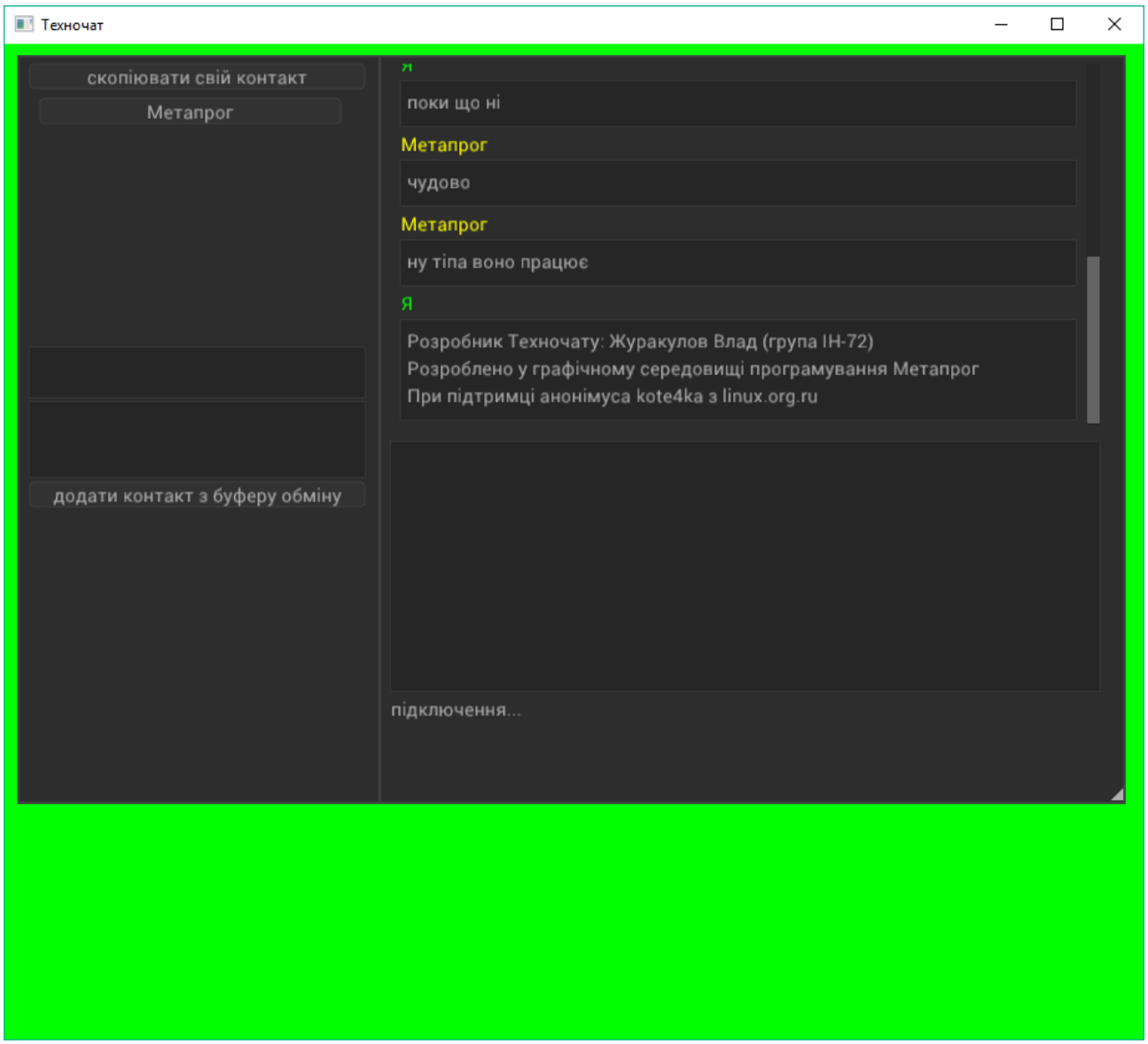

Рисунок 2.126 – Нова версія 73

ВИСНОВКИ

Завершуючи розробку децентралізованого чату у експериментальному середовищі Метапрог зазначу наступне:

- бібліотека з необхідним набором функцій була успішно написана;
- месенджер успішно розроблений і відповідає заданим вимогам;
- існуючий функціонал буде довершений новим функціоналом у найближчому майбутньому;
- я використав широкий спектр знань отриманих в університеті й інших джерелах і успішно реалізував свій досвід в даній роботі.

Ця робота є основою для вже існуючих та нових проектів. На момент закінчення написання роботи з існуючим прототипом вже взаємодіяли й проводили тестування близько п’яти людей, ще більша кількість ним зацікавлена. На мою думку це свідчить про актуальність й успішність моєї роботи.

СПИСОК ЛІТЕРАТУРИ

1. Robert Cecil Martin, Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall. ISBN 978-0132350884. 2009.
2. Codd Hooker, Brad. Rationality, rules, and utility: new essays on the moral philosophy of Richard B. Brandt. Boulder: Westview Press. ISBN 9780813315683. 1994.
3. Josef Albers. Interaction of Color. 1963.
4. Ellen Lupton. Type On Screen, Princeton Architectural Press. ISBN 978-1616891701. 2014.
5. Michael Bierut. How to Use Graphic Design to Sell Things, Explain Things, Make Things Look Better, Make People Laugh, Make People Cry, and (Every Once in a While) Change the World. Harper Design; ISBN 978-0062413901. 2015.
6. Metaprog. Метапрог-прототип 34 + СУВТ по логическому типу - https://www.linux.org.ru/forum/development/15824737

ДОДАТОК

Код Техночату версії 63 має 38 312 рядків і, як було зазначено в роботі вище, він автоматично генерується з примітивів мови С графічною середою Метапрог. Бібліотеки (include) є основою для роботи Техночату, на їх базі були створені деякі з описаних в роботі функції месенджеру.

```
#define _GNU_SOURCE

#include <SDL_net.h>
#include <SDL_mixer.h>
#include "nuklear_cross/nuklear_cross.c"
#include <stdlib.h>
#include <stdio.h>
#include <sodium.h>
#include <SDL.h>
#include <SDL_ttf.h>
#include <SDL_image.h>
#include <SDL2_gfxPrimitives.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <locale.h>
#include <wchar.h>
#include <stdint.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <assert.h>
#include <limits.h>
#include <string.h>
#include <unistd.h>
#include "tinyfiledialogs/tinyfiledialogs.h"
#include "tinyfiledialogs/tinyfiledialogs.c"
#include <gmp.h>
int metaprog_directory_creation_return;
int metaprog_flag_unix_filesystem_slash;
char metaprog_big_endian_flag;
#if defined _MSC_VER || __MINGW32__
#define __uint32_t unsigned int
#define __uint16_t uint16_t
#endif
```