#include <iostream>

#include <string>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <conio.h>

#include <sstream>

#include <iomanip>

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

class Client

{

private:

//client

SOCKET s;

WSADATA WsaData;

SOCKADDR\_IN sin;

char buff[BUFSIZ];

int nsize;

//s2vna

double range;

string ch;

double positionMark1, positionMark2;

void SetMarkersFromCenter(); //первый вариант выполнения задания - пользователь ввел только диапазон

void SetMarkersByTwoPoints(); //второй вариант - пользователь ввел две точки

void GetSettingsStatistic();

void GetCH();

void GetRange();

void GetMarks();

void Continue();

string GetStatisticInCp();

string DoubleToString(double);

public:

void GetClientSettings();

void ConnectToServer();

void StartСommunication();

};

void Client::GetClientSettings()

{

WSAStartup(0x0101, &WsaData); //Инициализируем процесс библиотеки ws2\_32, вызвав функцию WSAStartup

s = socket(AF\_INET, SOCK\_STREAM, 0); //Теперь объявление переменную типа SOCKET

//Задаем параметры для сокета(сервера)

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(5025);

inet\_pton(AF\_INET, "127.0.0.1", &(sin.sin\_addr));

}

void Client::ConnectToServer()

{

if (!connect(s, (struct sockaddr\*)&sin, sizeof(sin)))

cout << "Server connection has been established" << endl;

else

{

cout << "Server connection not established" << endl; exit(0);

}

}

//////////////////////////////////////////////////////////////////////////////////////////

string Client::DoubleToString(double val)

{

string result;

ostringstream ost;

ost << fixed << setprecision(10) << val;

result = ost.str();

return result;

}

void Client::GetCH()

{

cout << "CH: ";

cin >> ch;

}

void Client::Continue()

{

char val;

cout << "Continue? 1 - Yes; Any key - exit." << endl;

val = \_getch();

switch (val)

{

case '1':

{

\_getch();

StartСommunication();

break;

}

default:

exit(0);

}

}

string Client::GetStatisticInCp()

{

int size; //кол-во принятых байтов

string q = ""; //запросы

string p = ""; //ответы

q = "CALC"+ch+":MSTatistics:DATA?\n";

send(s, q.c\_str(), q.length(), 0);

size = recv(s, &buff[0], sizeof(buff), 0);

for (int i = 0; i < size; ++i) p += buff[i];

return p;

}

void Client::SetMarkersFromCenter()

{

int size; //кол-во принятых байтов

string q = ""; //запросы

string p = ""; //ответы

//получаем центр

q = "SENS" + ch + ":FREQ:CENT?\n";

send(s, q.c\_str(), q.length(), 0);

size = recv(s, &buff[0], sizeof(buff), 0); //получаем ответ

for (int i = 0; i < size; ++i) p += buff[i];

double center = atof(p.c\_str()); //центр

//определим позицию маркеров 1 и 2 относительно центра

range \*= pow(10, 6); //переводим диапазон в Hz

positionMark1 = center - (range / 2);

positionMark2 = center + (range / 2);

//создаем два маркера

q = "CALC" + ch + ":MARK1 ON;:CALC1:MARK2 ON\n";

send(s, q.c\_str(), q.length(), 0);

//Задаем позицию первому

q = "CALC" + ch + ":MARK1:X " + DoubleToString(positionMark1) + "HZ\n";

send(s, q.c\_str(), q.length(), 0);

//Задаем позицию второму

q = "CALC" + ch + ":MARK2:X " + DoubleToString(positionMark2) + "HZ\n";

send(s, q.c\_str(), q.length(), 0);

}

void Client::SetMarkersByTwoPoints()

{

int size; //кол-во принятых байтов

string q = ""; //запросы

string p = ""; //ответы

/\*q = "SENSe1:FREQuency:STARt?\n";

send(s, q.c\_str(), q.length(), 0);

size = recv(s, &buff[0], sizeof(buff), 0); //получаем ответ

for (int i = 0; i < size; ++i) p += buff[i];

double start = atof(p.c\_str()); //стимул - старт

q = "SENSe1:FREQuency:STOP?\n"; p = "";

send(s, q.c\_str(), q.length(), 0);

size = recv(s, &buff[0], sizeof(buff), 0); //получаем ответ

for (int i = 0; i < size; ++i) p += buff[i];

double stop = atof(p.c\_str()); //стимул - стоп

if ((start < positionMark1 < stop) && (start < positionMark2 < stop))

cout << "Warning. Markers are not included in the range of the graph from " << start << " to " << stop << endl;

\*/

//создаем два маркера

q = "CALC" + ch + ":MARK1 ON;:CALC1:MARK2 ON\n";

send(s, q.c\_str(), q.length(), 0);

//Задаем позицию первому

q = "CALC" + ch + ":MARK1:X " + DoubleToString(positionMark1) + "HZ\n";

send(s, q.c\_str(), q.length(), 0);

//Задаем позицию второму

q = "CALC" + ch + ":MARK2:X " + DoubleToString(positionMark2) + "HZ\n";

send(s, q.c\_str(), q.length(), 0);

}

void Client::GetSettingsStatistic()

{

string q = ""; //запросы

//включаем статистику

q = "CALCulate" + ch + ":MSTatistics 1\n";

send(s, q.c\_str(), q.length(), 0);

//включаем диапазон

q = "CALCulate" + ch + ":MSTatistics:DOMain 1\n";

send(s, q.c\_str(), q.length(), 0);

//выставляем начало диапазона

q = "CALCulate" + ch + ":MSTatistics:DOMain:STARt 1\n";

send(s, q.c\_str(), q.length(), 0);

//выставляем конец диапазона

q = "CALCulate" + ch + ":MSTatistics:DOMain: STOP 2\n";

send(s, q.c\_str(), q.length(), 0);

}

void Client::GetRange()

{

cout << "================" << endl;

cout << "Range (MHz): ";

cin >> range;

}

void Client::GetMarks()

{

cout << "================" << endl;

cout << "Marker 1 and Marker 2 (MHz): ";

cin >> positionMark1;

cin >> positionMark2;

positionMark1 \*= pow(10, 6);

positionMark2 \*= pow(10, 6);

range = abs(positionMark1 - positionMark2);

}

void Client::StartСommunication()

{

char val;

while (1)

{

cout << "================" << endl;

cout << "1: Range from center(MHz)" << endl;

cout << "2: Two point range(MHZ)" << endl;

cout << "3: Exit" << endl;

val = \_getch();

switch (val)

{

case '1':

{

\_getch();

GetCH();

GetRange(); //3.2.Ввод с консоли диапазона частот(в МГЦ) для получения данных математической статистики

SetMarkersFromCenter(); //3.3.Передача в S2VNA команд SCPI, задающих на графике два маркера, обозначающих начало и конец диапазона частот

GetSettingsStatistic(); //3.4.Передача в S2VNA команд SCPI, устанавливающих диапазон частот для расчета математической статистики (номера маркеров) и запускающих расчет данных математической статистики в выбранном диапазоне частот.

cout << GetStatisticInCp(); //3.5.Считать данные математической статистики и вывести их на консоль.

Continue();

break;

}

case '2':

{

\_getch();

GetCH();

GetMarks();

SetMarkersByTwoPoints();

GetSettingsStatistic();

cout << GetStatisticInCp();

Continue();

break;

}

case '3':

{

exit(0);

}

default:

{

\_getch();

system("cls");

}

}

}

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

Client Client1;

Client1.GetClientSettings();

Client1.ConnectToServer();

Client1.StartСommunication();

return 0;

}