Lucrare de atestare profesională la

Informatică

FangliMESS

**Profesor îndrumător**

**Butyka Edmond**

**19 Mai, 2011**

Fangli Andrei

Documentaţia lucrării pentru atestatul profesional

Liceul teoretic „Lucian Blaga” Cluj-Napoca, Cluj

Specializare: Matematică-informatică intensiv informatică

Profil: REAL

**Cuprins**

|  |  |
| --- | --- |
| Detalii de început  Sintaxe .NET și C++  Programarea socket  Teoria aplicaţiei  Codul sursă  Utilizarea aplicaţiei  Posibile extinderi  Bibliografie | 3  4  12  13  15  32  34  34 |

**Detalii de început**

După cum se poate observa și din titlu, proiectul acesta se bazează pe transmiterea unor mesaje prin intermediul reţelei. Pentru aceasta este nevoie de doua aplicaţii, cel puţin, deoarece de când s-au stabilit primele protocoale de transmitere de date în reţea, aceste protocoale se bazează pe o structură client – server. Clientul se conectează la server după care se realizează comunicarea până la deconectarea unuia dintre cei doi. Acest proiect conţine două aplicaţii, una de tip server care acceptă un numar infinit de conexiuni și altă aplicaţie, de tip client, care se conectează, prin reţea, la aplicaţia server.

După cum spune protocolul TCP/IP, trebuie specificat un IP (Internet Protocol) care arată poziţia în reţea, o adresă unică. Dintr-un fișier text, aplicaţia client va citi IP-ul, după care va încerca să se conecteze. În cazul în care aplicaţia nu reușește să se conecteze va afișa un mesaj corespunzător, menţinând astfel utilizatorul final informat.

Un alt amănunt important este codul în care sunt scrise aplicaţiile, o bună parte este scrisă în .NET Framework pentru a oferii un aspect prietenos și pentru a conferi aplicaţiilor un mod ușor de abordat. Această documentaţie cuprinde și un capitol de „tranziţie” pentru cei care nu cunosc încă unele sintaxe folosite în scrierea acestor aplicaţii. Capitolul de „tranziţie” pornește de la alocarea dinamică și prezintă o parte mică a sintaxei .NET, C++ului și bine înţeles o mică teorie despre programarea pe reţea și ce înseamna aceasta.

**Sintaxe .NET și C++**

Pentru început voi explica sintaxele C++: spaţii de nume, clase, definirea macrourilor și a operatorului *scope* și voi continua cu sintaxa .NET.

**Spaţii de nume**

Spaţiile de nume au fost introduse pentru a evita conflictele de nume între mai multe funcţii care au aceeași denumire și preiau valori prin aceeași parametrii. Prin intermediul spaţiilor de nume se pot declara două funcţii cu același antet și care execută instrucţiuni total diferite sau se pot rescrie unele funcţii deja existente, dar care trebuie adaptate la codul sursă. Sintaxa spaţiilor de nume este urmatoarea:

|  |
| --- |
| namespace <nume>  {  <instrucţiuni>  } |

Pentru a putea accesa funcţiile, clasele, structurile, variabilele sau ori ce alt ceva ce am declarat într-un spaţiu de nume folosim sintaxa *using*. Această sintaxa arată ce domenii de vizibilitate conţine codul sursă.

|  |
| --- |
| using namespace <nume>; |

Referirea la domeniul de vizibilitate al unui spaţiu de nume se poate face și prin intermediul operatorului *scope*.

**Operatorul scope**

Operatorul scope ‚::’ are sintaxa următoare:

|  |
| --- |
| <spaţiu de nume>::<nume funcţie/variabilă/clasa etc.> |

Spaţiul de nume indică domeniul de vizibilitate la care ne referim, ceea ce este în stânga operatorului scope, iar ceea ce se află în dreapta acestuia este la ce ne referim în domeniul de vizibilitate specificat.

Codul de mai jos prezintă o referire la un spaţiu de nume prin sintaxa *using* și o altă referire prin operatorul *scope*.

|  |
| --- |
| //Folosind sintaxa using  using namespace std;  cout<<”Folosind using”;  //Folosind operatorul scope  std::cout<<”Folosind operatorul scope”; |

În Visual C++ spaţiul de nume *std* conţine toate funcţiile și toţi operatorii standar C++ cum ar fi atoi(), itoa(), cin, cout și așa mai departe.

**Definirea macrourilor**

Un macro este o constantă, care are sintaxa după cum urmează:

|  |
| --- |
| #define LEN 10 |

În exemplul de mai sus am definit o constantă denumită *LEN*, compilatorul când rulează codul înlocuiește toate apariţiile lui LEN cu valoarea specificată după *LEN*. De exemplu:

|  |
| --- |
| //In codul sursa  #define LEN 10  char a[LEN];  //In compilator  char a[10] |

**Sintaxa claselor**

Clasele sunt structuri de date, se definesc identic cu un *struct.* Diferenţa dintre *struct* și *class* este aceea că structul este specific lui C, iar clasele sunt specifice lui C++, așa că dacă cineva vrea sa programeze în C++ trebuie sa folosească clase nu structuri. După cum s-a mai observat, în C++ există multe instrucţiuni care sunt din C și asta pentru că C++ deriva din limbajul vechi de programare C și astfel s-au pastrat unele sintaxe. C++ul, încă, se află într-o perioadă de tranziţie. O alta diferenţă între struct și class este aceea că într-o structură datele sunt declarate, predefinit, publice pe când într-o clasa aceste date sunt declarate private.

Sintaxa unei clase dupa cum urmeaza:

|  |
| --- |
| class <nume clasă>  {  private:  <declaraţii>  protected:  <declaraţii>  public:  <declaraţii>  }; |

Cuvintele cheie private, protected, public specifică accesul la datele care urmează după acestea. Respectiv private oferă acces datelor numai și numai funcţiilor membru ale clasei. Protected oferă acces și funcţiilor prietene care pot fi dintr-o altă clasă, iar public oferă acces din toate părţiile, și din interior de la funcţiile membru și din exterior de la alte funcţii.

Funcţiile membru se declară identit cu funcţiile globale, doar că acestea sunt delcarate în interiorul clasei.

|  |
| --- |
| class <nume clasă>  {  <dată de tip> <nume funcţie>(<parametrii formali>)  {  <instrucţiuni>  }  }; |

Sau cu ajutorul operatorului scope.

|  |
| --- |
| class <nume clasă>  {  <dată de tip> <nume funcţie>(<parametrii formali>);  };  <nume clasă>::<dată de tip> <nume funcţie>(<parametrii formali>);  {  <instrucţiuni>  } |

Printre funcţiile membru se află și constructorii și destructorii, sintaxa după cum urmează:

|  |
| --- |
| class <nume clasă>  {  public:  //Constructor  <nume clasă>(<parametrii formali>)  {  <instrucţiuni>  }  //Destructor  ~<nume clasă>(void)  {  <instrucţiuni>  }  }; |

Destructorul se apelează când se întâlnește cuvântul cheie *delete* sau la finalul execuţiei programului.

Constructorii sunt de mai multe feluri, sintaxa de mai sus prezintă un constructor ce se apelează la declararea unei variabile sau la alocarea unui spaţiu de memorie în cazul pointerilor. Constructorii se pot supradefinii, pot exista mai mulţi constructori, destructorul este unic și nu ia argumente. La scrierea unei clase, dacă autorul acelei clase nu definește constructorii, aceștia sunt definiţi de către compilator. La definirea unui constructor de iniţializare (cel prezentat în exemplu) compilatorul nu mai declară constructorul predefinit de iniţializare care este următorul:

|  |
| --- |
| <nume clasă>(void)  {} |

Adică o funcţie vidă. Constructorul este o funcţie care se apelează obligatoriu! Nu se poate omite acest apel prin nici o metodă!

Printre constructori se numără și constructorul de copiere care predefinit copiază toate valorile. Sintaxa:

|  |
| --- |
| class <nume clasă>  {  public:  <nume clasă>(<nume clasă> const &<nume variabilă>)  {  <instrucţiuni>  }  }; |

Aici parametrul formal se transmite prin referinţă pentru a evita declararea unei noi variabile care nu își are rostul. Cuvăntul cheie *const* specifică faptul că variabila, deși este transmisă prin referinţă, nu poate fi modificată în niciun fel. Un exemplu al constructorului de copiere care copiază numai valoarea *x*:

|  |
| --- |
| class punct  {  public:  int x, y;  punct(punct const &pct2)  {  x = pct2.x;  }  }; |

Sau folosind cuvântul cheie *this*.

|  |
| --- |
| class punct  {  public:  int x, y;  punct(punct const &pct2)  {  this.x = pct2.x;  }  }; |

Cuvântul cheie this este un pointer către variabila curentă sau către variabila care apelează funcţia.

Într-o clasă se mai pot declara și operatori, sintaxa acestora:

|  |
| --- |
| class <nume clasă>  {  public:  <nume clasă> operator<operator>(const <dată de tip> &pct2)  {  <instrucţiuni>  }  }; |

Un exemplu care adună coordonatele a două puncte:

|  |
| --- |
| class punct  {  public:  int x, y;  punct operator+(const punct &pct2)  {  punct pct3;  pct3.x = this.x + pct2.x;  pct3.y = this.y + pct2.y;  return pct3;  }  }; |

Un alt amănunt important pentru clase este faptul că acestea se pot *moștenii*, adică se poate scrie o clasă de bază din care, mai târziu, vor deriva mai multe clase la care se vor adăuga și alte date. Sintaxa:

|  |
| --- |
| class <nume clasă> : <nivel de acces> <nume clasă moștenită>  {  <declaraţii>  }; |

Nume clasă reprezintă numele noii clase, nivelul de acces este nivelul de acces pe care îi vor avea membri moșteniţi (privaţi, protejaţi sau publici), iar nume clasă moștenită este numele clasei care este moștenită. Prin moștenire se transmit toate datele clasei de bază cu cexcepţia constructorilor și datelor private ale acesteia.

**Despre .NET Framework**

Câteva cuvinte despre .NET Framework de la Microsoft. Pentru început trebuie menţionat dezavantajul major, programele în .NET nu sunt portabile, aceste programe funcţionează numai pe sistemele de operare Windows care au instalat .NET Framework, de preferat ultima versiune. În ciuda acestui dezavantaj major, .NET oferă o interfaţă grafică extrem de prietenoasă, iar designerul care vine cu Visual Studio face ca programarea .NET să fie și mai ușoară. Designerul va scrie codul pentru orice element ce este adăugat din meniul de obiecte (labeluri, căsuţe text, meniuri și multe multe altele). Ba chiar mai mult designerul va face și legaturile necesare executarii unei funcţii la accesarea unui obiect, de exemplu când se apasă un buton.

Deși .NET Framework este disponibil împreună cu Microsoft Visual Studio, în sistemele de dezvoltare C++, C# sau Basic, este de preferat ca .NET să fie tratat precum un limbaj de programare similar cu C++ul dar nu identic. Acest proiect a fost scris în Microsoft Visual Studio 2008 C++ Express Edition.

Platforma aceasta de la Microsoft aduce împreună cu interfaţa grafică și schimbari în sintaxă ca de exemplu cuvântul cheie *gcnew* sau noţiunea de *handle* sau referinţă.

**Cuvântul cheie gcnew**

O mică schimbare pentru alocarea spaţiului de memorie, nu se mai realizează prin cuvantul cheie new ci prin cuvântul cheie gcnew care vine de la **G**arbage**C**ollected**NEW**. O dată cu gcnew scăpăm și de grija eliberarii spaţiului de memorie la finalul execuţiei programului, tot ce a fost alocat cu gcnew va fi dealocat la finalul execuţiei programului fară să fie scris în cod, nu există valoarea NULL în .NET! Numărul de elemente trebuie să fie reţinut!

Înainte de a aloca spaţiu trebuie să existe o dată de tip care acceptă așa ceva, în .NET nu există pointeri, există handle sau referinţă. Această referinţă se marchează prin operatorul *top level* ‚^’ la declararea variabilei. O altă modificare în sintaxă vine la nivelul claselor, ca o clasă să poată să ofere declararea datelor de tip referinţă aceasta trebuie să fie menajată. O clasă menajată se declară prin adăugarea cuvântului cheie *ref* înainte de cuvântul cheie *class*. Sintaxe:

|  |
| --- |
| //Un sir de caractere in .NET  //Static  <dată de tip> <nume variabilă>;  //Dinamic  <dată de tip> ^<nume variabilă>;  //Alocarea unui sir de caractere dinamic in .NET  <nume variabilă> = gcnew <dată de tip>(<variabila sau sir de caractere>);  //Declararea unei clase menajate  ref class <nume clasă>  {  <declaraţii>  }; |

Un exemplu care transformă un șir de caractere standard (char\*) într-un șir de caractere .NET:

|  |
| --- |
| char a[13] = „Transformare”;  System::String ^str;  str = gcnew System::String(a); |

Mai multe exemple pe codurile sursă ale aplicaţiilor.

**Programarea socket**

Programarea pe reţea sau programarea socket, se bazează destul de mult pe UNIX în ceea ce privește conceptul de socket. În UNIX, socketul este un descriptor de fișier unde fișierul poate să fie ori ce, de la imprimantă la Internet, de la tastatură la mouse. Programarea socket se bazează și pe protocoalele foarte bine stabilite acum ceva vreme și anume protocoalele TCP/IP care datează de la apariţia reţelei și rămân valabile și în ziua de astăzi. Asta presupune o comunicare client – server care se poate realiza în mai multe moduri:

* + prin intermediul unui flux continu de transmitere de date, socketurile de tip stream.
  + prin intermediul unui pachet de volum mai mare în ceea ce privește dimensiunea digitală la intervale de timp și nu continu, acestea sunt socketurile de tip datagram.

Diferenţa dintre un stream și un datagram este aceea că, în cazul unui stream, datele se transmit și se primesc, dacă se primesc, în exact aceeași ordine în care au fost trimise. Pe când la un datagram, datele se transmit, acestea pot să ajungă, pot să nu, pot să ajungă într-o altă ordine decât cea în care au fost trimise. Din acest punct de vedere un stream este mai sigur decât un datagram, dar un stream nu va reuși nici odată să trimită un volum prea mare de informaţi, puntea maximă este de 1 Kb. Socketurile stream se folosesc la aplicaţiile care transmit mesaje pe reţea și probabil la jocurile simple. Socketurile datagram se folosesc mai mult la transmiterea de fișiere, la aplicaţiile care folosesc FTP (**F**ile **T**ransfer **P**rotocol).

Acest proiect folosește un socket de tip stream deoarece doar transmite mesaje pe reţea, iar valoarea maximă care poate fi trimisă o dată este de 0.5 Kb, deci mult sub limita maximă.

**Teoria aplicaţiei**

Teoria acestui proiect este simplă, aplicaţia de tip server acceptă conexiuni teoretic infinite, iar tot ce primește (mesaje și eroare de transmitere) trimite la toţi cei conectaţi. Aplicaţia de tip client se conectează și afișează tot ceea ce primește de la server.

**Clientul**

Aplicaţia de tip client este formată din 4 butoane (închidere, minimizare, conectare la server și trimitere mesaj la server), 4 labeluri (bară de titlu, casuţă pentru mesaj în raport cu conectarea la server, mesaj informativ în josul ferestrei, mesaj descriptiv pentru ceea ce trebuie completat), 2 casuţe text (pentru nume și pentru tipărirea mesajului care urmează să fie transmis), o casuţă text pentru afișarea mesajelor primite, și o lista care indică cine este conectat în momentul curent, un meniu care are obţiunile: conectare, deconectare, ieșire. Prin intermediul unui timer cu intervalul de 1ms, clientul verifică dacă a primit mesaje noi.

Meniul de conectare și butonul de conectare execută același cod și anume cel de a se conecta la IPul descris în fișierul text *set.txt*, în cazul în care reușește să se conecteze, aplicaţia schimbă puţin interfaţa pentru a face disponebile casuţele în care primește mesaje de la server, în care se scriu mesaje care urmează să fie trimise la server și lista cu cine este conectat în acel moment. În meniu se activează opţiunea deconectare și se dezactivează opţiunea conectare. În cazul în care numele este în uz sau conexiunea nu a reușit se afișează un mesaj corespunzător.

Când clientul primește un mesaj, acesta este verificat dacă conţine caracterul ‚:’, în cazul în care îl conţine înseamnă că s-a trimis un mesaj, iar în cazul în care nu conţine, ori cineva s-a conectat, ori cineva s-a deconectat. Asta este determinat dacă numele primit este sau nu în lista de conectaţi.

**Serverul**

Serverul este mai simplu decât clientul, în ceea ce privește aspectul grafic, deoarece acesta începe ascultarea pe un anumit port încă de la lansarea aplicaţiei, iar ascultarea se încheie o data cu aplicaţia. Forma este formată dintr-un buton de transmitere de mesaje la toţi cei conectaţi, o casuţă text pentru tipărirea mesajelor primite și afișarea evenimentelor de conectare și deconectare a utilizatorilor și o altă casuţă text pentru scrierea mesajelor ce urmează a fi transmise.

Serverul reţine, într-o listă simplu înlanţuită, descriptori de fișier sau socketurile celor încă conectaţi, și la intervale de 1ms, prin intermediul unui timer, verifică dacă s-au primit mesaje noi de la fiecare utilizator conectat. În cazul în care s-au primit mesaje, acestea se transmit tuturor utilizatorilor conectaţi în acel moment. În cazul în care unul dintre utilizatori s-a deconectat, numele acestuia se trimite la toţi cei conectaţi în acel moment dupa care este eliminat din lista de conectaţi.

**Structura unui mesaj**

Structura unui mesaj transmis pe reţea de către client sau server începe cu lungimea șirului de caractere care constituie mesajul, urmat de caracterul spaţiu iar apoi mesajul. Această structură este obligatorie deoarece bufferul de reţea reţine mesaje fară caracterul NULL, asta înseamna că citirea mesajelor trebuie să fie exactă! Abia după ce s-a citit mesajul i se adaugă caracterul NULL la final pentru a semnala capăt de șir.

Bufferul de reţea primește mesaje care sunt puse unul după altul, iar când se citește din acesta se citește până rămâne vid sau până la atingerea numarului maxim de valori specificat. Dacă serverul trimite 10 mesaje, iar clientul începe să citească din buffer abia după ce mesajele au fost trimise, el trebuie să știe să le interpreteze surplusul din buffer precum 10 mesaje diferite, nu ca unul singur, de aceea trebuie să se știe exact câte caractere are un mesaj.

**Codul sursă**

Pentru început, ambele coduri sursă pornesc de la șablonul predefinit care vine cu Microsoft Visual C++ 2008 Express Edition pentru Windows Forms. Pentru a obţine proiectul de start se începe cu **File->New->Project (Categoria Visual C++) ->Windows Forms Application**. De aici pornesc ambele aplicaţii, iar singurul fișier sursă modificat este *Form1.h*.

**Clientul**

Partea de început, macrouri, includeuri și încărcare de librarii

|  |
| --- |
| //Macrouri  #ifndef WIN32\_LEAN\_AND\_MEAN  #define WIN32\_LEAN\_AND\_MEAN  #endif  #define PORT "15193"  #define TV\_SEC 0  #define TV\_uSEC 0  #define LEN 512  #define NUME\_LEN 30  #define FISIER "set.txt"  //Includeuri  #include <fstream>  #include <windows.h>  #include <winsock2.h>  #include <ws2tcpip.h>  //Librarii  #pragma comment (lib, "ws2\_32.lib") |

Includerea spaţiilor de nume

|  |
| --- |
| namespace FangliMESS  {  //Spatii de nume  using namespace std;  using namespace System;  using namespace System::ComponentModel;  using namespace System::Collections;  using namespace System::Windows::Forms;  using namespace System::Data;  using namespace System::Drawing; |

Clasa mouse

|  |
| --- |
| ref class mouse  {  public:  bool Apasa;  ref class delta  {public:  int X, Y;  delta(){this->X = this->Y =0;}  }Delta;  mouse(){this->Apasa = false;}  }; |

Clasa client

|  |
| --- |
| ref class client  {  public:  System::String ^IP, ^Nume;  SOCKET Sock;  client(){this->Sock = INVALID\_SOCKET;}  bool Conectare(){  //Declaratii  int i;  char ip[21];  addrinfo \*rez, init;  //Conversie  for (i = 0; i < this->IP->Length; i++)  ip[i] = static\_cast<char>(this->IP[i]);  ip[this->IP->Length] = 0;  //Pregatire getaddrinfo()  ZeroMemory(&init, sizeof init);  init.ai\_family = AF\_UNSPEC;  init.ai\_socktype = SOCK\_STREAM;  init.ai\_protocol = IPPROTO\_TCP;  //getaddrinfo()  if (getaddrinfo(ip, PORT, &init, &rez))  return false;  //socket()  if ((this->Sock = socket(rez->ai\_family, rez->ai\_socktype, rez->ai\_protocol)) == INVALID\_SOCKET)  return false;  //connect()  if (connect(this->Sock, rez->ai\_addr, static\_cast<int>(rez->ai\_addrlen)) == SOCKET\_ERROR){  closesocket(this->Sock);  this->Sock = INVALID\_SOCKET;  return false;}  //Curatare  freeaddrinfo(rez);  //Returnare  return true;}  }; |

Clasa Form1

|  |
| --- |
| //Definitii  public ref class Form1 : public System::Windows::Forms::Form{  private:  mouse Mouse;  client ^Client;  System::Windows::Forms::Timer ^timer1;  System::Windows::Forms::ListBox ^listBox1;  System::Windows::Forms::RichTextBox ^richTextBox1;  System::Windows::Forms::PictureBox ^pictureBox1;  System::Windows::Forms::TextBox ^textBox1, ^textBox2;  System::Windows::Forms::Label ^label1, ^label2, ^label3, label4;  System::Windows::Forms::Button ^button1, ^button2, ^button3, button4;  System::Windows::Forms::MenuStrip ^menuStrip1;  System::Windows::Forms::ToolStripSeparator ^toolStripSeparator1;  System::Windows::Forms::ToolStripMenuItem ^programToolStripMenuItem, ^deconectareToolStripMenuItem, ^iesireToolStripMenuItem, ^conectareToolStripMenuItem;  System::ComponentModel::IContainer ^components;  protected:  ~Form1(){  //Curatare  if (components)  delete components;  if (this->Client->Sock != INVALID\_SOCKET)  closesocket(this->Client->Sock);  WSACleanup();}  public:  Form1(){  //Declaratii  char ip[21];  fstream f(FISIER, ios::in);  WSAData wsa;  //Initializari  InitializeComponent();  this->Client = gcnew Form1::client;  this->textBox1->Focus();  //WSAStartup()  if (WSAStartup(MAKEWORD(2,2), &wsa))  this->label3->Text = L"Eroare WSAStartup()";  //Citire IP  f.get(ip, 20);  if (f)  this->Client->IP = gcnew System::String(ip);  else  this->Client->IP = L"localhost";  //Curatare  f.close();} |

Funcţii

|  |
| --- |
| // cif()  int cif(int n){  int s = 0;  do s++; while(n = n/10);  return s;}  // Schimb()  void Schimb(){  this->textBox1->Enabled = !this->textBox1->Enabled;  this->textBox1->Visible = !this->textBox1->Visible;  this->textBox2->Enabled = !this->textBox2->Enabled;  this->textBox2->Visible = !this->textBox2->Visible;  this->richTextBox1->Enabled = !this->richTextBox1->Enabled;  this->richTextBox1->Visible = !this->richTextBox1->Visible;  this->pictureBox1->Enabled = !this->pictureBox1->Enabled;  this->pictureBox1->Visible = !this->pictureBox1->Visible;  this->listBox1->Enabled = !this->listBox1->Enabled;  this->listBox1->Visible = !this->listBox1->Visible;  this->label2->Enabled = !this->label2->Enabled;  this->label2->Visible = !this->label2->Visible;  this->label3->Enabled = !this->label3->Enabled;  this->label3->Visible = !this->label3->Visible;  this->button1->Enabled = !this->button1->Enabled;  this->button1->Visible = !this->button1->Visible;  this->button4->Enabled = !this->button4->Enabled;  this->button4->Visible = !this->button4->Visible;  this->conectareToolStripMenuItem->Enabled = !this->conectareToolStripMenuItem->Enabled;  this->deconectareToolStripMenuItem->Enabled = !this->deconectareToolStripMenuItem->Enabled;  this->timer1->Enabled = !this->timer1->Enabled;  this->Update();}  // Deconectare()  System::Void Deconectare(System::Object ^sender, System::EventArgs ^e)  {  //Schimb()  this->Schimb();  //Inchidere conexiune  closesocket(this->Client->Sock);  //Reinitializare  this->AcceptButton = this->button1;  this->Client->Sock = INVALID\_SOCKET;  this->listBox1->Items->Clear();  this->richTextBox1->Text = L"";  this->textBox2->Text = L"";  this->label1->Text = L"FangliMESS";  this->label3->Text = L"";  this->Text = L"FangliMESS";  this->textBox1->Focus();  } |

Funcţii

|  |
| --- |
| // Conectare()  System::Void Conectare(System::Object ^sender, System::EventArgs ^e)  {  //Declaratii  int n, i = -1;  char snd[LEN], rcv[LEN];  //Initializare  this->Client->Nume = this->textBox1->Text;  //Cazuri  // Exista nume  if (System::String::Compare(this->textBox1->Text, ""))  {  //Conectare la server  if (this->Client->Conectare())  {  //Conversie  for (i = 0; i < this->Client->Nume->Length; i++)  snd[i] = static\_cast<char>(this->Client->Nume[i]);  snd[this->Client->Nume->Length] = 0;  //Trimitere mesaj  send(this->Client->Sock, snd, static\_cast<int>(strlen(snd)), 0);  //Primire mesaj  i = recv(this->Client->Sock, rcv, 1, 0);  //Cazuri  // Nume in uz  if (i <= 0)  {  closesocket(this->Client->Sock);  this->label3->Text = L"Nume in uz";  }  // Primire lista de conectati  else  {  do if (rcv[0] != ' ')  {  //Citire lungime de sir  n = 0;  do{  n = n \* 10 + rcv[0] - '0';  i = recv(this->Client->Sock, rcv, 1, 0);  }while(rcv[0] != ' ' && i > 0); |

|  |
| --- |
| //Cazuri  // Sir primit  if (i > 0)  {  //Citire nume de lungime n  recv(this->Client->Sock, rcv, n, 0);  rcv[n] = 0;  //Adaugare in lista de conectati  this->listBox1->Items->Add(gcnew System::String(rcv));  //Preluare 'inceput' de mesaj  i = recv(this->Client->Sock, rcv, 1, 0);  }  // Server deconectat  else  {  //Inchidere conexiune  closesocket(this->Client->Sock);  //Mesaj  this->label3->Text = L"Conexiune intrerupta";  }  }  else  i = 0;  while (i > 0);  //Cazuri  // Conexiune reusita  if (!i)  {  this->Schimb();  this->listBox1->Items->Add(this->Client->Nume);  this->AcceptButton = this->button4;  this->label1->Text += L" - " + this->Client->Nume;  this->Text += L" - " + this->Client->Nume;  this->textBox2->Focus();  }  // Conexiune nereusita  else  {  this->listBox1->Items->Clear();  this->textBox1->Focus();  this->label3->Text = L"Inchis";  }  }  }  // Conexiune nereusita  else  this->label3->Text = L"Conexiune nereusita";  }  // Nume inexistent  else  this->label3->Text = L"Introduceti un nume";} |
| //Cazuri  // Sir primit  if (i > 0)  {  //Citire nume de lungime n  recv(this->Client->Sock, rcv, n, 0);  rcv[n] = 0;  //Adaugare in lista de conectati  this->listBox1->Items->Add(gcnew System::String(rcv));  //Preluare 'inceput' de mesaj  i = recv(this->Client->Sock, rcv, 1, 0);  }  // Server deconectat  else  {  //Inchidere conexiune  closesocket(this->Client->Sock);  //Mesaj  this->label3->Text = L"Conexiune intrerupta";  }  }  else  i = 0;  while (i > 0);  //Cazuri  // Conexiune reusita  if (!i)  {  this->Schimb();  this->listBox1->Items->Add(this->Client->Nume);  this->AcceptButton = this->button4;  this->label1->Text += L" - " + this->Client->Nume;  this->Text += L" - " + this->Client->Nume;  this->textBox2->Focus();  }  // Conexiune nereusita  else  {  this->listBox1->Items->Clear();  this->textBox1->Focus();  this->label3->Text = L"Inchis";  }  }  }  // Conexiune nereusita  else  this->label3->Text = L"Conexiune nereusita";  }  // Nume inexistent  else  this->label3->Text = L"Introduceti un nume";} |
| // Iesire()  System::Void Iesire(System::Object ^sender, System::EventArgs ^e)  {  this->~Form1();  }  // Minimizare  System::Void Minimizare(System::Object^ sender, System::EventArgs^ e)  {  //Cazuri  // Conectat  if (this->Client->Sock != INVALID\_SOCKET)  this->textBox2->Focus();  // Neconectat  else  this->textBox1->Focus();  //Minimizare  this->WindowState = System::Windows::Forms::FormWindowState::Minimized;  //Update  this->Update();  }  //Mouse  // Mouse up  System::Void Mouse\_Up(System::Object ^sender, System::Windows::Forms::MouseEventArgs ^e)  {  this->Mouse.Apasa = false;  }  // Mouse move  System::Void Mouse\_Move(System::Object ^sender, System::Windows::Forms::MouseEventArgs ^e)  {  if (this->Mouse.Apasa)  this->Location = System::Drawing::Point(this->Cursor->Position.X - this->Mouse.Delta.X, this->Cursor->Position.Y - this->Mouse.Delta.Y);  }  // Mouse down  System::Void Mouse\_Down(System::Object ^sender, System::Windows::Forms::MouseEventArgs ^e)  {  this->Mouse.Apasa = true;  this->Mouse.Delta.X = this->Cursor->Position.X - this->Location.X;  this->Mouse.Delta.Y = this->Cursor->Position.Y - this->Location.Y;  } |

|  |
| --- |
| // Trimitere  System::Void Trimitere(System::Object^ sender, System::EventArgs^ e)  {  //Declaratii  int i, len, s;  char snd[LEN];  //Cazuri  // Avem ce trimite?  if (System::String::Compare(this->textBox2->Text, ""))  {  //Transformare  itoa(this->Client->Nume->Length + this->textBox2->Text->Length +2, snd, 10);  s = cif(this->Client->Nume->Length + this->textBox2->Text->Length +2);  //Adaugare spatiu  snd[s] = ' ';  //Conversie nume  for (i = 0; i < this->Client->Nume->Length; i++)  snd[i + s +1] = static\_cast<char>(this->Client->Nume[i]);  //Adaugare ": "  snd[this->Client->Nume->Length + s +1] = ':';  snd[this->Client->Nume->Length + s +2] = ' ';  //Conversie mesaj  for (i = 0; i < this->textBox2->Text->Length; i++)  snd[i + this->Client->Nume->Length + s + 3] = static\_cast<char>(this->textBox2->Text[i]);  snd[this->textBox2->Text->Length + this->Client->Nume->Length + s +3] = 0;  //Trimitere mesaj  send(this->Client->Sock, snd, this->textBox2->Text->Length + this->Client->Nume->Length + s +3, 0);  //Reinitializare  this->textBox2->Text = L"";  }  } |

|  |
| --- |
| //Timer  System::Void timer1\_Tick(System::Object^ sender, System::EventArgs^ e){  //Declaratii  int i, n = 0;  char rcv[LEN];  System::String ^str;  fd\_set fd;  timeval ti;  //Pregatire select()  fd.fd\_array[0] = this->Client->Sock;  fd.fd\_count = 1;  ti.tv\_sec = TV\_SEC;  ti.tv\_usec = TV\_uSEC;  //select()  if (select(1, &fd, 0, 0, &ti)){  //Primire mesaj  i = recv(this->Client->Sock, rcv, 1, 0);  if (i > 0){  do{  //Conversie  n = n\*10 + rcv[0] - '0';  //Citirea urmatorului caracter  i = recv(this->Client->Sock, rcv, 1, 0);  }while (i > 0 && rcv[0] != ' ');  //Cazuri  // Inca conectat  if (i > 0){  //Citire mesaj de lungime n  i = recv(this->Client->Sock, rcv, n, 0);  rcv[n] = 0;  //Cazuri  // Am primit un mesaj  if (strchr(rcv, ':')){  this->richTextBox1->Text += gcnew System::String(rcv) + L"\n";  this->richTextBox1->Focus();  this->richTextBox1->SelectionStart = this->richTextBox1->Text->Length;  this->textBox2->Focus();}  // S-a (de)conectat cineva  else{  //Conversie  str = gcnew System::String(rcv);  // Conectat  if (!this->listBox1->Items->Contains(str))  this->listBox1->Items->Add(str);  // Deconectat  Else this->listBox1->Items->Remove(str);}}  // Deconectat  else this->Deconectare(sender, e);}  else this->Deconectare(sender, e);}} |

**Serverul**

Partea de început este identică cu cea a clientului. Două clase definite:

|  |
| --- |
| ref class socklist{  public:  System::String ^Nume;  SOCKET Sock;  socklist ^Urmator;  socklist(System::String ^nume, SOCKET sock){  this->Nume = nume;  this->Sock = sock;}};  ref class server{  public:  int len;  socklist ^SockList, ^Ultim;  server(){this->len = 0;}}; |

Clasa Form1

|  |
| --- |
| public ref class Form1 : public System::Windows::Forms::Form  {  private:  server ^Server;  System::Windows::Forms::Button^ button1;  System::Windows::Forms::Label^ label1;  System::Windows::Forms::RichTextBox^ richTextBox1;  System::Windows::Forms::TextBox^ textBox1;  System::Windows::Forms::Timer^ timer1;    System::Windows::Forms::MenuStrip^ menuStrip1;  System::ComponentModel::IContainer^ components;  protected:  ~Form1()  {  //Declaratii  int i;  //Curatare  if (components)  delete components;  for (i = 0; i < this->Server->len; i++, this->Server->SockList = this->Server->SockList->Urmator)  closesocket(this->Server->SockList->Sock);  } |

|  |
| --- |
| public:  Form1(){  //Declaratii  addrinfo \*rez, init;  SOCKET srv;  WSADATA wsa;  //Initializari  InitializeComponent();  this->Server = gcnew Form1::server();  //Pornire server  // Pregatire getaddrinfo()  ZeroMemory(&init, sizeof init);  init.ai\_family = AF\_INET;  init.ai\_socktype = SOCK\_STREAM;  init.ai\_protocol = IPPROTO\_TCP;  init.ai\_flags = AI\_PASSIVE;  // WSAStartup()  if (WSAStartup(MAKEWORD(2, 2), &wsa))  this->richTextBox1->Text += L"Eroare WSAStartup()\n";  // getaddrinfo()  if (getaddrinfo(0, PORT, &init, &rez))  this->richTextBox1->Text += L"Eroare getaddrinfo()\n";  // socket()  if ((srv = socket(rez->ai\_family, rez->ai\_socktype, rez->ai\_protocol)) == INVALID\_SOCKET)  this->richTextBox1->Text += L"Eroare socket()\n";  // bind()  if (bind(srv, rez->ai\_addr, static\_cast<int>(rez->ai\_addrlen)) == SOCKET\_ERROR)  this->richTextBox1->Text += L"Eroare bind()\n";  // listen()  if (listen(srv, SOMAXCONN) == SOCKET\_ERROR)  this->richTextBox1->Text += L"Eroare listen()\n";  // Curatare  freeaddrinfo(rez);  // Cazuri  // Pornire nereusita  if (srv == INVALID\_SOCKET)  this->richTextBox1->Text += L"Eroare de pornire";  // Pornire reusita  else  {  this->richTextBox1->Text += L"Server pornit\n--------------------------\n";  this->Server->SockList = gcnew Form1::socklist("FangliMESS Admin", srv);  this->Server->Ultim = this->Server->SockList;  ++this->Server->len;  this->timer1->Enabled = true;  }  } |

Funcţii

|  |
| --- |
| // cif()  int cif(int n){  int s = 0;  do s++; while (n /= 10);  return s;}  // cauta()  bool cauta(System::String ^str){  //Declaratii  int i;  Form1::socklist ^j = this->Server->SockList;  //Cautare  for (i = 0; i < this->Server->len; i++, j = j->Urmator)  if (!System::String::Compare(str, j->Nume))  return false;  //Returnare  return true;}  // Trimite  System::Void Trimite(System::Object ^sender, System::EventArgs ^e){  //Declaratii  int i, len;  char snd[LEN];  Form1::socklist ^j = this->Server->SockList->Urmator;  //Conversie lungime mesaj  len = this->Server->SockList->Nume->Length + this->textBox1->Text->Length +2;  itoa(len, snd, 10);  len = cif(len);  //Adaugare spatiu liber  snd[len] = ' ';  //Conversie nume  for (i = 0; i < this->Server->SockList->Nume->Length; i++)  snd[i + len +1] = static\_cast<char>(this->Server->SockList->Nume[i]);  //Adaugare ": "  snd[this->Server->SockList->Nume->Length + len +1] = ':';  snd[this->Server->SockList->Nume->Length + len +2] = ' ';  //Conversie mesaj  for (i = 0; i < this->textBox1->Text->Length; i++)  snd[i + this->Server->SockList->Nume->Length + len +3] = static\_cast<char>(this->textBox1->Text[i]);  //Semnalare final de sir  snd[this->textBox1->Text->Length + this->Server->SockList->Nume->Length + len +3] = 0;  //Memorare lungiem de sir  len = static\_cast<int>(strlen(snd));  //Trimitere mesaj la toti cei conectati  for (i = 1; i < this->Server->len; i++, j = j->Urmator)  send(j->Sock, snd, len, 0);  //Afisare mesaj  this->richTextBox1->Text += this->Server->SockList->Nume + L": " + this->textBox1->Text + L"\n";  //Reinitializare  this->textBox1->Text = L"";} |
| // conect()  void conect(){  //Declaratii  int i, k, len;  char rcv[LEN], snd[LEN];  Form1::socklist ^j = this->Server->SockList->Urmator;  System::String ^str;  SOCKET client;  fd\_set fd;  timeval ti;  //Pregatire select()  fd.fd\_array[0] = this->Server->SockList->Sock;  fd.fd\_count = 1;  ti.tv\_sec = TV\_SEC;  ti.tv\_usec = TV\_uSEC;  //select()  if (select(1, &fd, 0, 0, &ti)){  //accept()  client = accept(this->Server->SockList->Sock, 0, 0);  //Primire nume  i = recv(client, rcv, LEN, 0);  rcv[i] = 0;  //Conversie  str = gcnew System::String(rcv);  //Cautare  if (this->cauta(str)){  //Trimitere lista de conectati  for (i = 1; i < this->Server->len; i++, j = j->Urmator){  //Formare mesaj  len = cif(j->Nume->Length);  itoa(j->Nume->Length, snd, 10);  strncat\_s(snd, " ", LEN);  for (k = 0; k < j->Nume->Length; k++)  snd[k + len +1] = static\_cast<char>(j->Nume[k]);  snd[j->Nume->Length + len +1] = 0;  //Trimitere  send(client, snd, static\_cast<int>(strlen(snd)), 0);  }  //Trimitere mesaj de final  send(client, " ", 1, 0);  //Formare mesaj  itoa(static\_cast<int>(strlen(rcv)), snd, 10);  strncat\_s(snd, " ", LEN -1);  strncat\_s(snd, rcv, LEN -1);  len = static\_cast<int>(strlen(snd));  //Trimitere la toti cei conectati  j = this->Server->SockList->Urmator;  for (i = 1; i < this->Server->len; i++, j = j->Urmator)  send(j->Sock, snd, len, 0);  //Adaugare in lista  this->Server->Ultim->Urmator = gcnew Form1::socklist(str, client);  this->Server->Ultim = this->Server->Ultim->Urmator;  ++this->Server->len;  //Mesaj  this->richTextBox1->Text += L" --> S-a conectat " + str + L"\n";}  // Nume existent  else closesocket(client);}}  // Timer  System::Void Timer(System::Object ^sender, System::EventArgs ^e)  {  //Declaratii  int k, ok, n;  char rcv[LEN], snd[LEN];  bool p;  fd\_set fd;  timeval ti;  Form1::socklist ^i, ^j = this->Server->SockList;  //S-a conectat cineva?  this->conect();  //Am primit mesaje?  ti.tv\_sec = TV\_SEC;  ti.tv\_usec = TV\_uSEC;  //Parcurgere lista de conectati  for (k = 1; k < this->Server->len; k++, j = j->Urmator)  {  //Pregatire select()  fd.fd\_array[0] = j->Urmator->Sock;  fd.fd\_count = 1;  p = true;  //select()  if (select(1, &fd, 0, 0, &ti))  {  //Primire mesaj de inceput  ok = recv(j->Urmator->Sock, rcv, 1, 0);    //Cazuri  // Inca conectat  if (ok > 0)  {  //Obtinere lungime de sir  n = 0;  do if(rcv[0] != ' ')  {  n = n \* 10 + rcv[0] - '0';  ok = recv(j->Urmator->Sock, rcv, 1, 0);  }  else  ok = 0;  while (ok > 0); |

|  |
| --- |
| //cazuri  // Inca conectat  if (!ok)  {  //Citire sir de lungime n  recv(j->Urmator->Sock, rcv, n, 0);  rcv[n] = 0;  //Afisare mesaj  this->richTextBox1->Text += gcnew System::String(rcv) + L"\n";  }  // Deconectat  else  p = false;  }  // Deconectat  else  p = false;  //Cazuri  // Se trimite mesajul  if (p)  {  //Formare mesaj  itoa(n, snd, 10);  strncat\_s(snd, " ", LEN);  strncat\_s(snd, rcv, LEN);  //Memorare lungime de sir  n = static\_cast<int>(strlen(snd));  //Se da vestea-n targ  i = this->Server->SockList->Urmator;  for (ok = 1; ok < this->Server->len; ok++, i = i->Urmator)  send(i->Sock, snd, n, 0);  }  // Deconectat  else  {  //Mesaj  this->richTextBox1->Text += L" --> S-a deconectat " + j->Urmator->Nume + L"\n";  //Formare mesaj  n = j->Urmator->Nume->Length;  itoa(n , snd, 10);  n = cif(n);  snd[n] = ' '; |

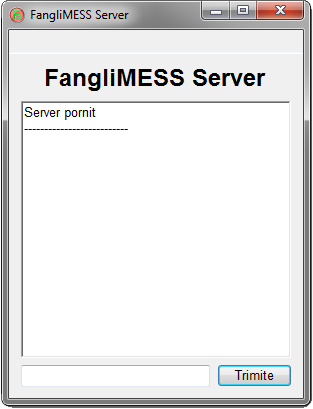
|  |
| --- |
| //Conversie  for (ok = 0; ok < j->Urmator->Nume->Length; ok++)  snd[ok + n +1] = static\_cast<char>(j->Urmator->Nume[ok]);  snd[j->Urmator->Nume->Length + n +1] = 0;  //Memorare lungime sir  n = static\_cast<int>(strlen(snd));  //Se da vestea-n targ  i = this->Server->SockList->Urmator;  for (ok = 1; ok < this->Server->len; ok++, i = i->Urmator)  send(i->Sock, snd, n, 0);  //Stergem ultimul din lista?  if (this->Server->Ultim == j->Urmator)  this->Server->Ultim = j;  //Eliminare din lista  j->Urmator = j->Urmator->Urmator;  --this->Server->len;  }  }  }  } |

**Utilizarea aplicaţiei**

Pentru utilizarea clientului sau a serverului este nevoie de un sistem de operare Windows cu .NET Framework instalat, de preferat ultima versiune (se poate realiza prin Windows Update). Nu există cerinţe minime hardware deoarece acestea sunt impuse de cerinţele minime pentru a putea rula sistemul de operare Windows.

Singura cerinţă software este pentru server, este nevoie de acces de administrator pentru ca aplicaţia server să poată să accepte conexiuni din exteriorul sistemului de calcul.

**Serverul**

Aplicaţia de tip server are nevoie de drepturi de administrator pentru a putea accepta conexiuni din afara sistemului de calcul pe care rulează. O dată ce a fost rulat, acesta începe să asculte pe portul *15193* pentru eventuale conexiuni, iar această ascultare se înceie odata cu aplicaţia.

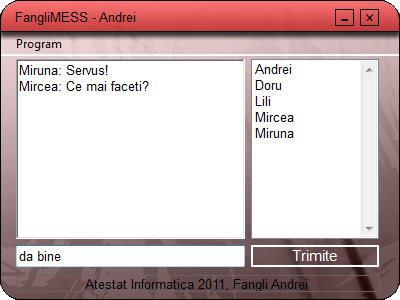
Deasupra butonului *Trimite* se afla o casuţă text în care vor fi afișate mesajele trimise de utilizatorii conectaţi și în momentul în care un utilizator se conectează sau se deconectează se va afișa un mesaj corespunzător.

În stânga butonului *Trimite* se afla o altă casuţa text în care se poate scrie, iar la apăsarea butonului **Trimite** se va trimite mesajul scris în acea casuţă text, dacă există unul.

**Clientul**

Pentru a folosi clientul, este nevoie de crearea fișierului text *set.txt* în același directoriu cu aplicaţia, dacă fișierul nu există. În fișier se scrie, pe prima linie, începand cel mai din stânga, adresa IPv4 a serverului. Aplicaţia se conectează pe portul *15193*. Dacă **firewallul** din Windows blochează conexiunea, trebuie să fie permis traficul catre IPul scris în fișierul text prin portul *15193* pentru ca aplicaţia sa se poată conecta la server. În cazul în care apar erori, aplicaţia va afișa mesaje corespunzătoare. Autorul aplicaţiei, eu, Fangli Andrei, nu răspund pentru orice evenimente, nedorite sau dorite, care apar după rularea aplicaţiei, rulaţi aplicaţia la riscul dumneavoastră.

După lansarea aplicaţiei va apărea o fereastră similară cu cea din stânga, completaţi câmpul *Nume* după care apăsaţi **Conectare** sau accesaţi din meniul *Program* -> **Conectare** sau folosiţi scurtătura pentru conectare și anume **ALT + C**. În cazul în care aplicaţia nu se poate conecta sau numele este în uz, se va afișa un mesaj corespunzător.

După o conexiune reușită veţi primii, prin intermediul casuţei din dreapta de deasupra butonului *Trimite*, lista celor conectaţi în momentul curent. Această listă va adăuga și va elimina utilizatori după starea conexiunii lor.

În stânga listei de utilizatori conectaţi veţi primii mesaje, iar în stânga butonului *Trimite* puteţi scrie, iar în momentul în care apăsaţi butonul **Trimite** se va trimite mesajul scris în casuţa text, în cazul în care este scris unul.

Pentru a vă deconecta accesaţi din meniul *Program* funcţia **deconectare** sau folosiţi scurtătura **ALT + D**.

**Posibile extinderi**

**Clientul**

Una dintre posibile extinderi ale clientului ar putea fi posibilitatea de a putea discuta în privat cu unul sau mai mulţi dintre utilizatorii conectaţi sau de a face posibilă crearea de conturi, iar fiecare utilizator să aibă propria lui listă de utilizatori înregistraţi cu care să comunice.

O altă extindere ar fi posibilitatea de a trimite fișiere între utilizatori sau posibilitatea de a suna unul sau mai multi utilizatori.

**Serverul**

Pentru server s-ar putea adăuga câteva comenzi pentru o administrare mai ușoară a utilizatorilor conectaţi, acceptarea de înregistrăre a numelui de utilizator și alte informaţii pe care aceștia ar dorii să le adauge.

**Bibliografie**

**C++ Ghid pentru începători**  - Brian Overland

**Beej.us** – Guide to network programming

**MSDN.com** – Getting started with winsock

**Cplusplus.com** – Windows programming