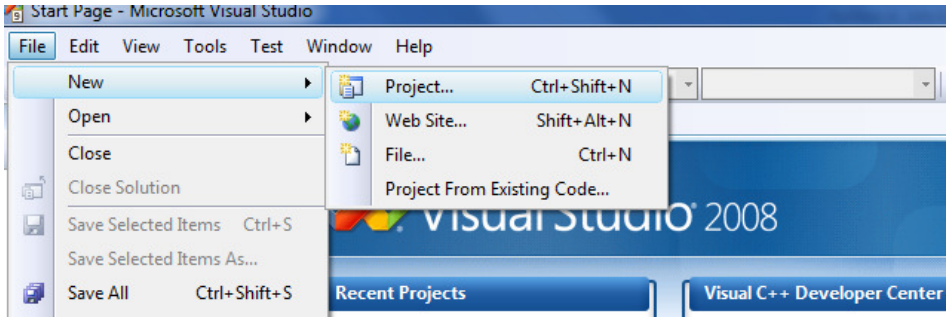
Aplikacja Sieciowa

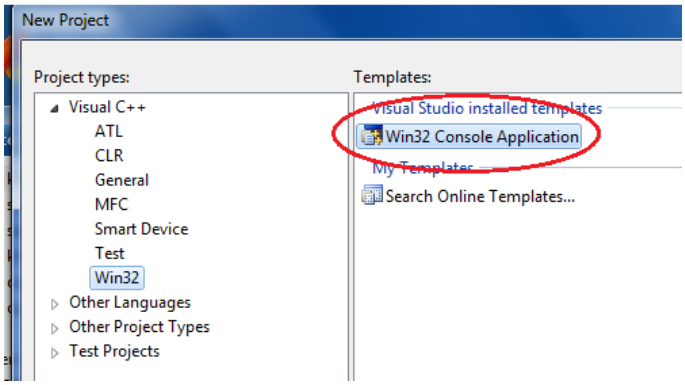
Jedną z fundamentalnych właściwości wielu aplikacji jest możliwość operowania pomiędzy jednostkami (np. PC), które włączone są do sieci. W Windows operacja ta jest możliwa przy wykorzystaniu „gniazdek” czyli „socketów”. Programiści z firmy Microsoft ułatwili nam zadanie i nie musimy pisać naszych „gniazdek” od zera. Wykorzystamy gotowe funkcje znajdujące się w bibliotece *winsock2*.

Zadanie na dzisiejszych zajęciach polega na stworzeniu aplikacji klienta, która będzie mogła wysyłać oraz odbierać dane od drugiego klienta. Dane zawsze wymieniane będą poprzez serwer. Serwer łączy kolejnych dwóch klientów w pary i pozwala im na prowadzenie wymiany danych. Serwer można pobrać z *moodle,* aplikacje klienta stworzymy według poniższych pleceń:

Najpierw tworzymy nowy projekt, tym razem pracować będziemy w konsoli, a zatem: *File->New->Project…*



Następnie:



Wpisujemy nazwę *„klient”* i wybieramy *„Finish”*.

Uzyskujemy pusty projekt, który po uruchomieniu wyświetla konsolę. Teraz musimy stworzyć odpowiednią klasę, która umożliwi całkowitą komunikację z serwerem.

Klasa nazywać będzie się `CClientWSA` natomiast plik nagłówkowy będzie miał postać:

```
#pragma once
#ifdef WIN32
#pragma comment(lib,"ws2_32")
#endif
```

```
#include "winsock2.h"

class CClientWSA
{
	public:
		CClientWSA(void);
	public:
		~CClientWSA(void);
		void initializeWSA(); // inicjalizuje Winsock.
		void socketCreate(); // tworzy socket.
		void connectToServer(const char * addr, unsigned short port); //połączenie z serwerem
		void sendData(); //wysyła sendbuff
		void receiveData(); //odbiera recvbuff
		void clearBuff(void);//czysci bufor wejsciowy i wyjsciowy
	public:
		SOCKET m_socket;//socket
		sockaddr_in clientService; // struktura zwiazana z serwerem - patrz connectToServer
		char sendbuf[32]; //bufor wyjściowy
		char recvbuf[32]; // bufor wejściowy
		int bytesRecv;//liczba otrzymanych bajtow;
};
```

Ciekawostką jest tu jawne dodanie biblioteki winsock'a : *"ws2_32"*

Mamy już deklaracje, teraz dodajemy definicje (w odpowiednio nazwanym pliku):

W konstruktorze oraz destruktorze nic się specjalnego nie dzieje więc zostawiamy puste. W inicjalizacji dodajemy kod, który umożliwi inicjalizację biblioteki obsługi gniazdek:

```
WSADATA wsaData;
	int iResult = WSAStartup( MAKEWORD(2,2), &wsaData );//initialize Windows Sockets API

		if ( iResult != NO_ERROR )
		printf("Error at WSAStartup()\n");
```

W metodzie odpowiedzialnej za tworzenie gniazdek wpisujemy:

```
//tworzone jest gniazdo dla rodziny adresów (AdresFamily) sieci internet dla trybu
//strumieniowego przesyłania danych), do obsługi tego trybu w seci internet służy protokół
//TCP/IP (IPPROTO_TCP)
	m_socket = socket( AF_INET, SOCK_STREAM, 0 ); // tworzony jest socket
		if ( m_socket == INVALID_SOCKET ) { // obsługa błędu
		printf( "Error at socket(): %ld\n", WSAGetLastError() );
		WSACleanup();
		return;
		}
```

Kolejna metoda odpowiada za nawiązanie połączenia. Tutaj dopisujemy:

```
//inicjowana jest struktura zawierające adres serwera i numer gniazda na którym on nasłuchuje
//(10000) do własnych celów należy wykozystywać gniazda o numerze >1024. Adres IP
//127.0.0.1 odpowiada lokalnemu komputerowi (localhost)
	clientService.sin_family = AF_INET;
	clientService.sin_addr.s_addr = inet_addr( addr);
	clientService.sin_port = htons( port );//htons( 10000 );- konwersja zapsiu bitowego w bajtach
	if ( connect( m_socket, (SOCKADDR*) &clientService, sizeof(clientService) ) == SOCKET_ERROR) {
```

```
	printf( "Failed to connect.\n" ); //obsługa błędu
	WSACleanup();//w przypadu błedu należy również odłączyć bibliotekę
	return;
	}
```

Metoda wysyłająca dane może mieć postać:

```
//wysyła dane do gniazda serwera określonego przez m_socket. Z bufora sendbuff wysyłana
//jest ilość bajtów podana przez strlen(sendbuf);
	int bytesSent = send( m_socket, sendbuf, strlen(sendbuf), 0 );
	printf( "Bytes Sent: %ld\n", bytesSent );
```

Oczywiście musimy również odebrać dane (w odpowiedniej metodzie):

```
//Odebrane dane są zapisywane do bufora recbuf, którego maksymalny rozmiar jet określony
//w wywołaniu funkcji recv.Ilość danych odczytanych jest zwracana do zmiennej bytesRecv
	bytesRecv = SOCKET_ERROR;

	while( bytesRecv == SOCKET_ERROR ) {
		bytesRecv = recv( m_socket, recvbuf, 32, 0 );
		if ( bytesRecv == 0 || bytesRecv == WSAECONNRESET ) {
			printf( "Connection Closed.\n");
			break;
		}

		if(bytesRecv < 0) return;

	printf( "Bytes Recv: %ld\n", bytesRecv );
	}
```

Na samym końcu musimy dodać odpowiednią definicję metody czyszczącej bufory.

Uwaga! W pliku definicji metod musimy zostać zawarta następująca preambuła:

```
#include "stdafx.h"
#include "ClientWSA.h"
#include <stdio.h>
```

Skończyliśmy pracę nad klasami. Teraz obsługa pliku gdzie zawarty jest nasz wątek główny czyli „main”. Najpierw dodajemy odpowiednie pliki oraz informujemy linker o dołączeniu biblioteki winsock’a:

```
#include "stdafx.h"
#ifdef WIN32
#pragma comment(lib,"ws2_32")
#endif

#include "winsock2.h"
#include "ClientWSA.h"
#include "conio.h"
#include <string>
#include <iostream>
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
```

```
#include <winsock.h>  // our thread for recving commands
using namespace std;
```

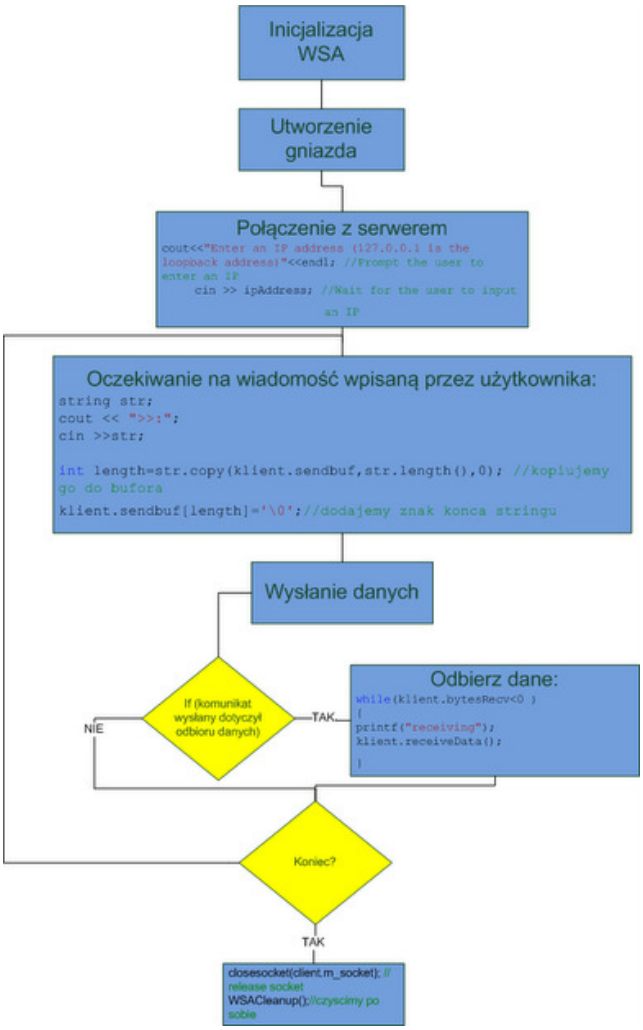
Ciało naszego main'a jest puste:

```
int _tmain(int argc, _TCHAR* argv[])
{

        …

        system("pause");
        return 0;
}
```

Teraz naszym zadaniem jest implementacja, następującego algorytmu w miejsce trzech kropek:



Poprawnie napisana aplikacja pozwala na komunikacje pomiędzy klientami według obowiązujących zasad:

a) Do uruchomionego serwer łączą się klient 1 oraz klient 2

b) Jeśli klient jako pierwszy bajt wyśle „1” wówczas ciąg znaków za „1” będzie traktowany jako dane do drugiego klienta (np. 1Hello – do klienta drugiego wysłane zostanie „Hello”)
c) Jeśli klient jako pierwszy bajt wyśle „2” wówczas uzyska dane, które wysłał do niego drugi klient (bajty za „2” nie są brane pod uwagę)
d) Komunikat nie zaczynający się na „1” lub „2” jest odbierany przez serwer, a następnie pomijany.

Aplikacja może działać w obrębie jednej jednostki. Wtedy łączymy klientów z danym serwerem wpisując adres: „127.0.0.1”.

Uwaga: jako pierwszy zawsze uruchamiamy serwer.

Sprawozdanie:

1. Chcemy, aby do serwera mogło połączyć się trzech klientów. Zmodyfikować kod serwera oraz klienta, tak, aby możliwym było przesyłanie informacji pomiędzy klientem 1, 2 oraz 3 według reguł:
   - jeśli pierwszy bajt jest równy „1” to dane są wysyłane do klienta 1,
   - jeśli pierwszy bajt jest równy „2” wówczas dane są przesyłane do klienta 2,
   - jeśli pierwszy bajt oznacza „3” dane są wysyłane do klienta 3,
   - jeśli pierwszy bajt jest równy „4” wówczas klient odbiera dane, które zostały do niego przesłane
   Dane wysyłane mogą być nadpisywane, natomiast odbiorca nie musi być informowany o tożsamości nadawcy.
   Dołączyć kod, który uległ modyfikacjom, wraz z komentarzem
2. Używając schematu blokowego proszę przedstawić algorytm interakcji pomiędzy klientem 1, klientem 2 oraz serwerem (dokładnie tak, jak ma to miejsce w naszym programie) uwzględniając fazę łączenia, wysyłania oraz odbierania informacji przez klientów (*klient 1->serwer->klient 2->serwer->klient 1*).