Politechnika Poznańska
Wydział Budowy Maszyn i Zarządzania

Programowanie systemów informatycznych – laboratorium

Ćw. 3: Akwizycja danych. RS – 232 cd.

1) Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z podstawową metodą akwizycji danych z wykorzystaniem protokołu komunikacji RS - 232.

2) Zakres niezbędnych wiadomości

- Akwizycja danych
- Zapis do pliku
- Odczyt z pliku
- Ramka komunikacji

3) Wstęp

Akwizycja danych służy do zapisu podstawowych ustawień programu oraz istotnych danych, które w wyniku pracy aplikacji zostają zapisywane w celu od tworzenia przebiegu np. procesu technologicznego. Zapis danych do pliku może zostać wykorzystany do monitorowania oraz nadzorowania awarii które programista przewidzi. Visual Studio może operować na podstawowym zapisie do pliku z wykorzystaniem plików bibliotecznych:

```
#pragma once
#using <System.dll>
#include <string.h>
#include <iostream>
#include <fstream>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::IO::Ports;
using namespace System::IO;
using namespace System::Threading;
using namespace System::Reflection;
using namespace std;
```

Tryb	Opis trybu
ios::app	**app** – (dopisywanie danych do pliku) ustawia wewnętrzny wskaźnik zapisu pliku na jego koniec. Plik otwarty w trybie tylko do zapisu. Dane mogą być zapisywane tylko i wyłącznie na końcu pliku.
ios::ate	**at** - Ustawia wewnętrzny wskaźnik pliku na jego koniec w chwili otwarcia pliku.
ios::binary	**binary** - Informacja dla kompilatora, aby dane były traktowane jako strumień danych binarnych, a nie jako strumień danych tekstowych.
ios::in	**in** - Zezwolenie na odczytywanie danych z pliku.

ios::out	**out** - Zezwolenie na zapisywanie danych do pliku.
ios::trunc	**trunc** - Zawartość pliku jest tracona, plik jest obcinany do 0 bajtów podczas otwierania.

Wszystkie parametry trybów pracy można łączyć ze sobą. Przykład otwarcia pliku:

```
std::fstream plik;
plik.open( "nazwa_pliku.txt", std::ios::in | std::ios::out );
plik.close();
```

Oznacza to, że dany plik może zostać odczytywany jak również zapisywany.

Zapis danych do pliku:

```
std::fstream plik;
plik.open( "nazwa_pliku.txt", std::ios::out );
char *bufor = "nazwa";
plik.write( bufor, strlen(bufor) ); //wczytuje tyle danych ile się zmieści do bufora
plik.close();
```

Odczyt danych z pliku:

```
std::fstream plik;
plik.open( "nazwa_pliku.txt", std::ios::in);
char bufor[ 1024 ];
plik.read( bufor, 1024 ); //wczytuje tyle danych ile się zmieści do bufora
plik.close();
```

Zapis i odczyt z pliku można realizować również z wykorzystaniem strumieni „<<” oraz „>>”.

```
char* zmienna;
fstream f("x.txt",ios::app);
f << zmienna <<" ";
f.close();
```

Visual Studio operuje również na zapisie plików w postaci:

```
StreamWriter^ write=gcnew StreamWriter("plik.txt");
write->WriteLine("napis");
write->Close();
```

Odczyt plików w postaci:

```
StreamReader^ file= File::OpenText("plik.txt");
textBox1->Text=file->ReadLine();
file->Close();
```

Konwersja z “char” do “string”

```
char* strChar="text";
String^ strNew;
strNew = gcnew String(strChar);//Here charString is ur char*
label1->Text = strNew;
```

Konwersja z “string” do “char”

```
using namespace System;
using namespace System::Runtime::InteropServices;

#include <iostream>                                    // for printf


int main()
{
            // Create a managed string.
            String^ managedString = "Hello unmanaged world (from the managed world).";

            // Marshal the managed string to unmanaged memory.
            char* stringPointer = (char*) Marshal::StringToHGlobalAnsi(managedString ).ToPointer();

            printf("stringPointer = %s\n", stringPointer);

            // Always free the unmanaged string.
             Marshal::FreeHGlobal(IntPtr(stringPointer));

             return 0;
}
```

4) Zadania.

Ćw. 1 Należy utworzyć pliki i przetestować zapis oraz odczyt danych z wykorzystaniem obu metod przedstawionych w wstępie do ćwiczenia. Proszę zamieścić pliki z zapisanymi danymi oraz odczytanymi.

Ćw. 2 Należy rozbudować aplikację komunikacyjną RS -232 o możliwość wyboru:
- liczby bitów na sekundę
- bitów danych
- parzystość
- bity stopu
- sterowanie przepływem

Do tego celu należy wykorzystać kontrolki „comboBox" i w odpowiednim polu należy umieścić możliwe do wyboru parametry. W celu dopasowania parametrów możliwych do wyboru w komunikacji należy wykorzystać kontrolkę „serialPort". Aplikacja użytkownika po zatwierdzeniu odpowiednim klawiszem powinna zapisać parametry komunikacji do pliku *„ustawienia.txt"* i ładować je przy włączaniu aplikacji . Aplikacja powinna posiadać wszystkie funkcjonalności z lab nr 2.

W przypadku zapisu danych taki jak:
- port
- liczby bitów na sekundę
- bitów danych

Należy wykorzystać zapis pliku z wykorzystaniem `"comboBox1->Text"`.

W przypadku zapisu danych taki jak:
- parzystość
- bity stopu
- sterowanie przepływem

Należy wykorzystać zapis pliku z wykorzystaniem `" comboBox4->SelectedIndex".` Ponieważ te parametry są przyjmowane jako liczby.

Odczyt parametrów:
- port
- liczby bitów na sekundę
- bitów danych

```
comboBox1->Text=file->ReadLine();
```

Odczyt parametrów:
- parzystość
- bity stopu
- sterowanie przepływem

```
comboBox4->SelectedIndex=Int16::Parse(file->ReadLine());
```

Najważniejszym krokiem jest odpowiednie przypisanie odczytanych danych do kontrolki „serialPort”:

```
serialPort1->PortName=file->ReadLine();
serialPort1->BaudRate=System::Int64::Parse(file->ReadLine());
serialPort1->DataBits=System::Int64::Parse(file->ReadLine());
serialPort1->Parity=(Parity)Int32::Parse( file->ReadLine());
serialPort1->StopBits=(StopBits)Int32::Parse( file->ReadLine());
serialPort1->Handshake=(Handshake)Int32::Parse(file->ReadLine());
```

Przykładowy wygląd zapisanego pliku z ustawieniami:
COM1
19200
8
0
1
0

- odbiór danych z komunikacji należy zrealizować z wykorzystaniem przerwania:

```
String^ odbior; //deklaracja lokalna wewnątrz klasy
private: System::Void serialPort1_DataReceived(System::Object^  sender,
System::IO::Ports::SerialDataReceivedEventArgs^  e)
{

    char litera=serialPort1->ReadChar();
    odbior += Char::ToString(litera);
    Odczyt_znak^ dane=gcnew Odczyt_znak(this,&rs::Form1::wyswietl_znak);
    dane(odbior); //wywołanie funkcji umożliwiającej odczytanie danych z portu COM
}

private: System::Void wyswietl_znak( String^ ciag )
{
    //np. na kontrolce richTextBox1 w parametrze „text” umieszczamy „string^ ciag”;
    SetControlPropertyValue(richTextBox1, "Text",ciag);
}
```

Przykład wyglądu okna z możliwymi parametrami:

Następnie aplikacja powinna umożliwić zapis do pliku „dane.txt" odebranych informacji po protokole RS – 232.

Wygląd programu dowolny z zachowanie wymienionych obiektów oraz poprawności działania.

5) Sprawozdanie

Sprawozdanie powinno zawierać:
- fragmenty funkcji wykonanych przykładów ze wstępu do ćwiczenia
- program, jaki grupa napisała na zajęciach
- za każdą linijką programu, jeżeli będzie taka potrzeba, należy umieścić bieżący komentarz na temat danej linii kodu
- pod każdym programem należy umieścić własny opis działania programu i wnioski.
- ogólne wnioski i spostrzeżenia na temat wykonanego ćwiczenia i zdobytej wiedzy