Compilers Documentation

Az első feladatom a félkész console-os alkalmazásunk in-outputjának vizuális megjelenítése, ehhez hozzáadtam egy **WinForms** -t a már meglévő projecthez.

Ezt nem a project forráskódjának átmásolásával egy új projectbe értem el, hanem indítottam egy külön szálat a winforms-nak, így a program indításakor mind a console-os felület, mind a winForm-s elérhető számunkra.

Célok/Terv :

Funkciók: Adatbázis beolvasása, az alkalmazásban készített adatbázis kimentése **.csv** fájlformátumba.

Tetszőleges méretű szabály táblázat generálása, kitöltése, majd a program ezt értékelje ki.

Regexxel az [0-9], [a-z], [A-Z] inputok a nyelvtanunk által meghatározott tetszőleges karakterré alakuljanak (ez a mi példánkban az **i** volt).

A lefutás lépésenkénti leírása (debug mód messageBoxokkal) és/vagy egyből kiírja hogy sikeres e.

Az output célját megerősítő színnel megieleníteni (piros: sikertelen, sikeres volt: zöld)

először a jsonnel, majd a xml el próbálkoztam, mind tul bonyolultan, serailizekkel mukodott, a xml

Egyértelmű, jól átlátható design/ megjelenés, gazdag információ megjelenítéssel rendelkező felhasználói felület.

A lekérdezést automatikusan kiegészíti a # karakterrel.

Az alábbi szabály táblázatot használva:

	$+$	$*$	$($	$)$	i	$\#$
E			$(TE', 1)$		$(TE', 1)$	
E'	$(+TE', 2)$			$(\varepsilon, 3)$		$(\varepsilon, 3)$
T			$(FT', 4)$		$(FT', 4)$	
T'	$(\varepsilon, 6)$	$(*FT', 5)$		$(\varepsilon, 6)$		$(\varepsilon, 6)$
F			$((E), 7)$		$(i, 8)$	
$+$	pop					
$*$		pop				
$($			pop			
$)$				pop		
i					pop	
$\#$						$elfogad$

Megvalósítás :

Először **JSON** adat struktúrában szerettem volna tárolni az adatokat, ehhez viszont a népszerűsége ellenére is elég sok serializálást kellett volna alkalmazni, ezért egyszerűbbnek tűnő lehetőségként a **xml**-re esett a választásom, ennél viszont oszlop fejléc nevekként semmiképen sem megadható speciális karakter, ami a mi esetünkben pedig szükséges lenne(szükséges lenne (+,*) input jellel végezhető műveletek miatt).

Szöveget beolvassa a program textként, ezt követően datatable-é alakítjuk es azt adom át a datagridview-nek datasource-kent.

A messageboxok pedig vezetnek az utonkon, ha elakadnánk, a hibákat, kivételeket kiváltó tényezők le lettek minimalizálva, amíg pl nem választottunk szabály rendszert, inaktív a analizálás gombja.

Amíg a felhasználó nem töltötte be a szabály táblázatot, addig nem tud analizálni sem.

„Okos" file mentés is helyet kapott, ha egyből mentünk, temp.csv file-ba menti ha nem akkor az előzőleg választott, kiböngészett helyen írogatja felül a fileunkat.

Mentés előtt kiüríti a kimeneti file-t, hogy a többszörös mentés következtében ne történjen duplikálás.

Ilyen módon viszont magát a datagridview-t nem tudtam teljes egészében manipulálni, mivel a datatable-t átadva neki, nem jönnek létre sem a header-jei sem a soraira, oszlopaira nem lehet hivatkozni közvetlenül, a headerek pedig megváltoztathatatlanul „beleégnek" a datatable-be, ezért minden írásnál új példányt kell ebből inicializálni.

A dataTable egyfajta összekötő szerepét tölti be az adatok és a dataGridView között.

A beviteli mező „hülye biztos" lett, azaz az user nyugodtan vihet be szóközöket az inputba, üres mezőre nem engedi tovább, ha pedig épp nem a nyelvtanunknak megfelelő betűt visz be a felhasználó (most nálunk i a megfelelő), akkor azt i-re alakítja (a a-z,A-Z,1-9 karaktereket).

A felhasználó informálását a Messagebox-ok biztosítják, szinte mindenről kap visszajelzést a felhasználó.

A felhasználó felület az alábbi képen néz ki: