I. Alapismeretek

I.1. A nyelv története

A C# programozási nyelv a Microsoft új fejlesztési környezetével, a 2002-ben megjelent Visual Studio.NET programcsomaggal együtt, annak részeként jelent meg.

Bár ez a nyelv hosszú múlttal nem rendelkezik, mindenképpen elődjének tekinthetjük a C++ nyelvet, annak szintaktikáját, szerkezeti felépítését.

A C, C++ nyelvekben készült alkalmazások elkészítéséhez gyakran hoszszabb fejlesztési időre volt szükség, mint más nyelvekben, például a MS Visual Basic esetében. A C, C++ nyelvek komplexitása, a fejlesztések hosszabb időciklusa azt eredményezte, hogy a C, C++ programozók olyan nyelvet keressenek, amelyik jobb produktivitást eredményez, ugyanakkor megtartja a C, C++ hatékonyságát.

Erre a problémára az ideális megoldás a C# programozási nyelv. A C# egy modern objektumorientált nyelv, kényelmes és gyors lehetőséget biztosítva ahhoz, hogy .NET keretrendszer alá alkalmazásokat készítsünk, legyen az akár számolás, akár kommunikációs alkalmazás. A C# és a .NET keretrendszer alapja a *Common Language Infrastructure(CLI).*

I.2. A nyelv jellemzői

A C# az új .NET keretrendszer bázisnyelve. Tipikusan ehhez a keretrendszerhez tervezték, nem véletlen, hogy a szabványosítási azonosítójuk is csak egy számmal tér el egymástól. A nyelv teljesen komponens orientált. A fejlesztők számára a C++ hatékonyságát, és a Visual Basic fejlesztés gyorsaságát, egyszerűségét ötvözték ebben az eszközben.

A C# nyelv legfontosabb elemei a következők:

- Professzionális, Neumann-elvű. Nagy programok, akár rendszerprogramok írására alkalmas.
- A program több fordítási egységből – modulból – vagy fájlból áll. Minden egyes modulnak vagy fájlnak azonos a szerkezete. Névterek, osztályok (partial) nem feltétlenül csak egy állományban helyezkedhetnek el.
- Egy sorba több utasítás is írható. Az utasítások lezáró jele a pontosvessző (;). Minden változót deklarálni kell. Változók, függvények elnevezésében az ékezetes karakterek használhatók, a kis- és nagybetűk különbözőek.
- A keretrendszer fordítási parancsa parancssorból is egyszerűen használható. (pl. *csc /out:alma.exe alma.cs*).

- Minden utasítás helyére összetett utasítás (blokk) írható. Az összetett utasítást a kapcsos zárójelek közé *{}* írt utasítások definiálják.
- Érték (alaptípusok, *enum, struct, value*), illetve referencia (*class*) típusú változók.
- Nincs mutatóhasználat; biztonságos a tömbhasználat (vektorhasználat).
- Boxing, unboxing. Minden típus őse az *object*, így például egy egész típust (*int*) csomagolhatunk objektumba (boxing), illetve visszaalakíthatunk (unboxing).
- Függvények definíciói nem ágyazhatók egymásba, egy függvény önmagát meghívhatja (rekurzió). Tehát függvénydefiníció esetén nincs blokkstruktúra. Blokkon belül statikus vagy dinamikus élettartamú változók deklarálhatók. Függvénypolimorfizmus megengedett.
- Érték, referencia (*ref*) és output (*out*) függvényparaméterek.
- Kezdő paraméter-értékadás, változó paraméterszámú függvény deklarálása.
- Delegáltak, események használata.
- Hierarchikus névterekben használt osztályok. Mivel minden osztály, ezért a „program", a ***Main*** függvény public *static* hozzáférésű. Több osztály is tartalmazhat ***Main*** függvényt, de ilyen esetben a fordításkor meg kell mondani, hogy melyik osztálybeli ***Main*** függvény legyen az aktuális induló (*/main:osztálynév*).
- Új operátorok: ***is*** operátor egy objektum típusát ellenőrzi (*x is Labda*), ***as*** operátor a bal oldali operandust jobb oldali típussá konvertálja (*Labda l = x as Labda;*). A hagyományos konverziós operátortól abban különbözik, hogy nem generál kivételt!
- Privát konstruktor, statikus konstruktor használata. (Privát konstruktort akkor használjuk, ha a típusból nem akarunk egy példányt se. Statikus konstruktor feladata a statikus mezők inicializálása, a statikus konstruktor mindig példány konstruktor előtt hívódik meg, futási időben az osztálybetöltő hívja meg.)
- Egyszeres öröklés, interface-ek használata.
- Nincs C++ elvű destruktor, helyette a keretrendszer szemétgyűjtési algoritmusa van, ami az esetlegesen definiált destruktort meghívja. Szükség esetén az IDisposable interface *Dispose* metódusa újradefiniálható.
- Operátorok definiálásának lehetősége, *property, indexer* definiálás, aszimmetrikus elérés.
- Extension metódusok (függvénykiterjesztés) definiálása.
- Kivételkezelés megvalósítása.
- Párhuzamos végrehajtású szálak definiálhatósága.
- Generic (típusparaméter) használhatósága.
- Lambda kifejezések (Lambda-calculus) használata.