2. Tétel

a) Magasszintű programozási nyelvek I.

Tömbök és listák

- összetett adattípus

- homogén: minden elemének ugyanaz a típusa

- tetszőleges elérésű

- folytonos: az elemek folytonos memória területen tárolódnak

- Tömb VS Lista

- statikus: a hosszát előre kell deklarálni

- Tömböt akkor használjunk: elemszám ismert init előtt

- dinamikus: elem hozzáadás/törlés bármikor

- Listát: elemszám nem ismert

- Tömbök:

- Deklaráció: típus[] var;

- Inicilaizáció: var = new típus[méret]

- mérete futási időben megadható

- méret lekérdezése: var.Length

- elem elérése: var[index] (0, Length-1)

- tömbökre jól alkalmazható for foreach és while ciklusok

- többdimenziós tömbök: 2-> típus[,] var = new típus[méret1,méret2] {}/ n->típus[,,]

- var.GetLength

- Listák:

- Deklaráció: List<típus> var;

- Inicializálás: var = new List<típus>();

- méret lekérdezése: var.Count (Length VS Count)

- elem eléreése: var[index]

- var.Add(érték), var.Insert(index, érték), var.RemoveAt(index), var.Remove(érték)

- Java-ban a tömb ugyanaz, a lista csak egy interfész, ArrayList a megvalósítása

- Pythonban csak Lista

Rekordok, class és struct

- összetett adatípus, heterogén, véletlen elérésű, folytonos

- elemeket mezőnek nevezzük

- C#: class (referencia)->rövid életű, kis mennyiségű adathoz, struct->nagyobb méretű adathoz (érték)

- Inicializálás new kulcsszóval

- mező elérése a '.' operátorral-al

Felsorolásos típusok

- Enum (nevesített konstans értékeket tartalmaz)

- enum Napok {Het, Kedd ..};

- alapértelmezett érték/típus (0/int)

Metódusok

- könnyebb olvashatóság és újrafelhasználhatóság érdekében

- Eljárás: nincs visszatérési érték->utasítás végrehajtása / Függvény: van v. é.->kiszámítás

- return kulcsszó: Függvénynél kötelező, Eljárásnál nem

- public és névtér

- felparaméterezhetők = lokális változó

Paraméterátadás módjai, változó paraméterszám

- Metódus fejléce: formális paraméterlista

- Metódushívás: aktuális paraméter

- egyenlő számú, típuskompatibilis

- opcionális és nevesített paraméterek

Programozási nyelvek fordítási és futtatási megoldásai

- Fordító (compiler) + linker (Forráskód->fordító->Gépi kód->CPU) (C, C++, Pascal)

- Előnye gyors, hátránya nem biztonságos

- Interpreter (Forráskód->interpreter->CPU) (szkriptnyelvek)

- Előnye biztonságos, hátránya lassú

- Just-in-time (JIT) fordítás (Forráskód->fordító->Virtuális gépi kód->interpreter->CPU) (Java)

- Előnye: kellően gyors és biztonságos hátránya: virtuális CPU implementálása

.NET keretrendszer felépítése, más programozási nyelvek és keretrendszerek

- C#->CLS(alapszabályok)->.NET Class Library(I/O)->CLR(Garbage C.)->OS

- Common Intermediate Language(.exe)

- Java keretrendszerek: Oracle, OpenJDK / fejlesztői környezetek: NetBeans, Eclipse, IntelliJ IDEA

- C, C++ keretrendszerek: GNU, Visual C++ / f. k.: Eclipse, Visual Studio

- Python k.: Cpython, PyPy / f. k.: PyCharm, Spyder

b) Adatszerkezetek és algoritmusok

Algoritmus fogalma

- matematika és informatika szintjén

- probléma megoldására irányuló elemi lépéssorozat (Turing-gép)

Algoritmus tulajdonságai, eszközei, strukturált algoritmus szerkezete (programkészítés lépései)

- Tulajdonságai: Végesség, Determinisztikus, Bemenet/Kimenet, Hatékonyság (3)

- Eszközei: Folyamatábra, Pseudokód, Programozási nyelvek

- Strukturált algoritmus szerkezete: (3 vezérlési szerkezet)

- Szekvencia

- Szelekció (logikai feltételen alapul)

- Iteráció (ismételt végrehajtás)

- Programkészítés lépései: Specifikáció->Tervezés->algoritmus<->adatszerkezet<->programozási nyelv->Kódolás->Tesztelés, Hibajavítás->Dokumentáció készítése

- Spec. -> feladat célkitűzések

- Tervezés -> adatszerkezetek és algoritmus kölcsönös kapcsolata majd befolyásolja a programozási nyelvet

Programozási tételek

- Alap-algoritmusok:

- Összegzés (sorozathoz elemi értéket)

- Megszámlálás (sorozathoz elemi értéket)

- Kiválogatás (sorozathoz sorozatot)

- Minimum és maximumkiválaszátás érték/hely (sorozathoz elemi értéket)

- Keresés (while) - függvény: (4db)

- Eldöntés (logikai->megmondja hogy benne van-e T-tul. vagy nincs)

- Lineáris keresés (vissza is adja a helyét)

- Kiválasztás (első előfordulás helyét adja vissza)

- Strázsás keresés

- Lineáris keresés rendezett sorozatban

- Bináris keresés (rendezetten működik csak)

- Rendezés (for) - eljárás: ()

- Közvetlen kiválasztás

- Sorozat rendezése közvetlen kiválasztással

- Minimum kiválasztásos rendezés (kevesebb csere, mert csak akkor hajtja végre a cserét, ha megvan a legkisebb elem)

- Buborékrendezés (n-1 lépés) n-szer hajtjuk végre

- Javított buborékrendezés (VoltCere, UtolsóCsereH)

- Koktélrendezés (nyulak-teknősök)

- Beszúró rendezés (azt nézzük meg, hogy melyik részsorozatba szúrjuk be)

- Shell rendezés (min. továbbfejlesztése) (apró részsorozatokra bontjuk és azokon min. rendezés, majd összefésüljük)

- Összefuttatás (két már rendezett sorozatot mos össze egy harmadikká, figyelembe véve a hosszok különbségét is)

- Visszalépéses keresés:

- visszalép, ha a részmegoldás zsákutcába jut és másik utat választunk

- pl. N királynő probléma (NxN-es sakktáblán N királynő elhelyezése támadás nélkül)

- Előnyök: rugalmas, sokféle problémára alkalmazható, könnyen érthető

- Hátrányok: nagy keresési tér esetén időigényes

- Ha tudjuk hogy biztosan van megoldás

Elemi algoritmusok alkalmazása

- Alap-algoritmusok

A halmaz adatszerkezet különböző konstrukciói

- Rendezetlen lista vagy tömb: bármilyen sorrendben tárolódnak (egyszerű)

- HashSet: az elemek hash táblában (ennek előnye gyors keresés, beszúrás és törlés)

- Rendezett lista vagy tömb: pl. növekvő sorrend

- TreeSet: az elemek bináris keresőfában

- Karakterisztikus függvény alapú megközelítés:

- BitSet: 0/1 ha a halmazban van vagy ha nincs -> memóriahatékony

- Karakterisztikus függvény: ugyanez, absztrakt matematikai megközelítés