Programozási alapismeretek

Alapok

A programkészítés folyamata 9 lépésből áll:

1. Specifikálás
   - Megértjük a feladatot, majd **specifikációt** készítünk.
2. Tervezés
   - Elkészítjük a specifikációból az **algoritmust**.
3. Kódolás
   - Kódban **implementáljuk** az algoritmusunkat.
4. Tesztelés
   - **Hibalistát** készítünk, kiderítjük, hogy hibás-e a kód.
5. Hibakeresés
   - A hibalistában leírt **hibák helyét** megkeressük.
6. Hibajavítás
   - Kijavítjuk a hibákat, így egy **helyes programot** kapunk.
7. Minőségvizsálat,hatékonyság
   - Megpróbáljuk **"jobbítani a programot"**, így egy **jó programot** kapunk.
8. Dokumentálás
   - Leírjuk a program működését, ezáltal a programunk **használható** lesz.
9. Használat,karbantartás
   - Karbantartjuk kódunkat, hogy **időtálló program** maradjon.

- A specifikáció a *feladat formális megragadása*, avagy szerződés a megbizó és a fejlesztő között.

Legalább 4 részből áll:

1. Bemenet
   - Adat/adatok amitől függ a megoldás.
2. Kimenet
   - Adat/adatok ami a megoldás.
3. Előfeltétel
   - Megszorítás vagyis a mindig/mikor van megoldás?-ra a válasz.
4. Utófeltétel
   - A bemenet és kimenet összekapcsolása, az eredményt meghatározó logikai állítás-

- Az algoritmus az a megoldás **elemi lépésekre/tevékenységekre** bontása.
  - Elemi tevékenységek az: értékadás, beolvasás és kiírás.

Algoritmust összeállíthatunk 4 féle módon:

- Szekvencia (egymás utáni végrehajtás)
- Elágazás
- Ciklus (ismétlés adott darabszámszor vagy adott feltételtől függően)
- Alprogram (absztrakció)

A feladat specifikációja:
- Valós világbeli objektum → valós világbeli eredmény

A klaszikus programok 3 fő lépése a(z):
- adatok beolvasása
- az eredmény kiszámítása
- eredmény kiirása

[deklaráció, beolvasás, feladatmegoldás, kiírás]

- A specifikációbeli címkhez **változókat** hozunk létre, az bemeneti változók felveszik a bemeneti adatok értékeit.
- Az algoritmus a megoldás során **módosíthatja** a változók értékét, majd a legvégén a kimeneti változók a kimeneti adatok értékét kell tartalmazzák.

Kódban így néz ki:

```
namespace program {
  internal class Program {
    static void Main(string[] args) {
      // Deklaráció (Specifikáció be,ki)

      // Beolvasás (Specifikáció be)

      // Feldolgozás / feladatmegoldás (Stuki alapján)

      // Kiírás (Specifikáció ki)
    }
  }
}
```

KONSTANS érték: nem változhat meg az állapothalmaza.
VÁLTOZÓ érték: végrehajtás során megváltozhat az állapothalmaza.

Változó lehet:
- Bementi változó
- Eredmény
- Részeredmény

Típus: olyan **absztrakt kategória** amely rögzítik bizonyos adatok állapothalmazát és az elvégezhető műveletek készletét

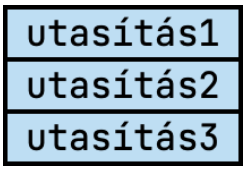
Típus lehet strukturálatlan vagy strukturált:
- Strukturálatlan : skalár
- Strukturált : ha elemibb összetevőkre bontjuk

1. Vezérlési szerkezetek

1.1. Szekvencia

Utasítások egymás utáni végrehajtása‼

Példa:
- egy recept (ismétléseket leszámítva)
- meghatározni egy szám első és második számjegyét

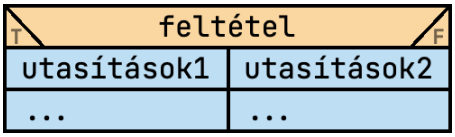


1.2. Elágazás:

Választás 2 vagy több utasítás közül

Példa:
- ügyintézés
- vércsoport meghatározása

Fajtáját tekintve 3 féle elágazásról beszélünk:
- Kétirányú
- Sokirányú (általános if-else)
- Sokirányú (speciális switch-case)

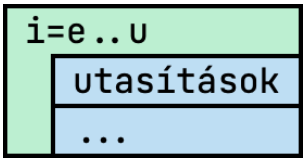


1.3. Ciklus

Ismételt végrehajtás (adott darabszámtól / utasítástól függően)

Példa:
- Recept (ismétléseket már beleértve)
- Összegzéses feladatok
- Maximumkereséses feladatok
- Osztó(k) keresése

Fajtáját tekintve 2 féle ciklusról beszélünk:
- Feltételes (azon belül lehet elől-vagy hátultesztelős)
- Számlálós



2. Adatszerkezetek

2.1. Rekord

- Rekordok használata nélkül a specifikációban bizonyos adatok között lehet nem lesz **szemantikus** kapcsolat, ennek megteremtésére szolgál a **rekord**.
- Rekord: létrehozza a szemantikus egységet, és *különböző funkciójú* adatokat egybezár.

Specifikációban így deklaráljuk:

Az × a direkt szorzatot jelenti:

$$p \in Pont,\ Pont=(x:R \times y:R)$$

A *Pont* az itt egy új adattípus, részeire nevük által meghatározott **szelektorokkal** hivatkozunk. Például:

$$p = (p.x,\ p.y)$$

Rekordok használata kódban:

```
struct Pont
{
  public double x,y;
}
...

Pont p;
```

2.2. Tömb

- Vannak olyan feladatok amiket nem tudunk csak elemi típusokkal vagy rekordokkal elvégezni, az ilyen feladatok megoldásához **tömböket** használunk.
- A tömb, egy homogén adatszerkezet, *ugyanolyan funkciójú* adatok sokasága (mint a rekord, ugyanúgy szemantikus egységet hoz létre)
- A tömbben található elemek csak is *közvetve* **indexeken** keresztül érhetőek el

2.2.1. Sorozat:

- Azonos funckiójú elemek egymásutánja, elemei indexelhetők

2.2.2. Tömb:

- Véges hosszúságú sorozat *algoritmikus* párja, i-edik tagjával végezhetünk műveleteket. Azonos funkciójú elemek egyirányú sorozata.

Specifikációban a tömb definiálása:

n∈N, számok∈N[1..n]
vagy
színek∈S[1..4]=[“zöld”, “piros”, “tök”, “makk”]

Kódban (**statikus tömb** - igény szerinti méret):

```
int n;
int.TryParse(Console.Readline(), out n);
string[] vendegek = new string[n];
```

- Specifikációban és algoritmusban (struktogram) a tömbök 1-től indexelődnek!!

2.2.3. Mátrix:

- Azonos funkciójú elemek kétirányú sorozata!

Specifikációban:
n∈N, m∈N, x∈Z[1..n, 1..m]
Két index kell egy elem kiválasztásához: x[i,j]

Példa:

- Hónapokhoz számok rendelése

3. Programozási minták

3.1. Analóg programozás

Analóg problémamegoldás:

- Ha egy olyan feladatot kell megoldanunk, aminek megoldása hasonlít egy korábbi feladatéhoz, akkor a korábbi feladat megoldása alapján járunk el.

Analóg programozás: A konkrét feladatot egy korábbi feladat megoldása alapján állítjuk elő.

- A **visszavezetés** során a *mintaprogramot* sablonként használva állítjuk elő a konkrét feladat programját, és figyelembe vesszük a két feladat specifikációbeli különbségeit.

Példa:

- N elemű tömb elemeinek összege -> Vektorok skaláris szorzata

Skaláris szorzat specifikációja:
Be: n∈N, a∈Z[1..n], b∈Z[1..n]
Ki: s∈Z
Ef: -
Uf: s=SZUMMA(1..n, b∈Z[1..n])

- A programok ad hoc módon is előállhatnak, azonban vannak olyan megoldásak amelyek nagyjából ugyanazt a feladatot oldják meg kicsit másképp.

3.1.1. Programozási tétel:

- Célja:
  - Bizonyíthatóan helyes sablon, amelyre lehet építeni a megoldást
- Szerkezete:
  1. absztrakt feladat specifikáció
  2. absztrakt algoritmus

3.1.2. Programozási tétel felhasználásának menete:

1. A konkrét feladat specifikálása
2. A programozási tételek **“megsejtése”**
3. A konkrét és absztrakt feladat paramétereinek egymáshoz rendelése
4. A konkrét algoritmus “generálása”, a megsejtett tételek alapján, 3. szerint átparaméterezve
5. “Hatékonyítás” programtranszformációkkal

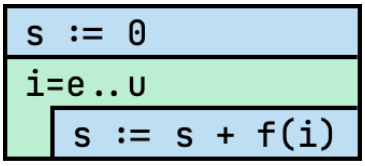
4. Programozási tételek részletesen

4.1. Összegzés tétele:

- Kulcsfontosságú részlet -> N szám összegét kell kiszámolni
- Adott egész számok [e..u] intervalluma és egy f: [e..u] -> H függvény, határozzuk meg a $\sum_{i=e}^{u} f(i)$ kifejezés értékét!

Specifikáció (sablon):

Be: e∈N, u∈Z
Ki: s∈H
Ef: -
Uf: s=SZUMMA(i=e..u, f(i))



Példafeladat: Mennyi pénze lesz valakinek n hónap végére?

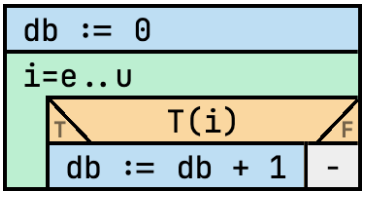
Be: n∈N, jöv∈Jövedelem[1..n], Jövedelem=(be: N × ki: N)
Ki: s∈Z
Ef: -
Uf: s=SZUMMA(i=1..n, jöv[i].be-jöv[i].ki)

4.2. Megszámolás tétele

- Kulcsfontosságú részlet: -> N darab “valamire” kell megadni, hogy hány adott tulajdonságú van közöttük!
- Adott egész számok [e..u] intervalluma és egy T: [e..u] -> Logikai feltétel. Határozzuk meg hogy az [e..u] intervallumon a T feltétel hányszor veszi fel az igaz értéket!

Specifikáció (sablon):

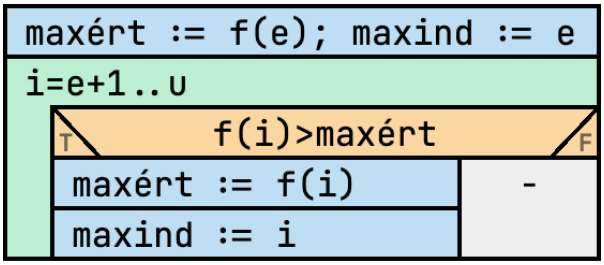
Be: e∈Z, u∈Z
Ki: db∈N
Ef: -
Uf: db=DARAB(i=e..u, T(i))



4.3. Maximumkiválasztás

- Kulcsfontosságú részlet: -> N darab “valami” közül kell megadni a legnagyobb (vagy legkisebbet) - (“leg” szórészletnek kell szerepelnie)
- Adott egész számok [e..u] intervalluma és f:[e..u] -> H függvény. A H halmaz elemein értelmezett egy *teljes rendezési reláció*. Határozzuk meg, hogy az f függvény hol veszi fel az [e..u] nem üres intervallumon a legnagyobb értéket, és mondjuk meg mekkora ez az érték!

Specifikiáció (sablon): Be: e∈Z, u∈Z
Ki: maxind∈Z, maxért∈H
Ef: e<=u
Uf: maxind∈[e..u] és ∀i∈[e..u]:(f(maxind)>=f(i) és maxért = f(maxind))
Röviden: (maxind, maxért) = MAX(i=e..u, f(i))



Specifikáció (sablon):

Példafeladat:

Adjuk meg n napi statisztika alapján a legmelegebb napot!

Specifikáció:

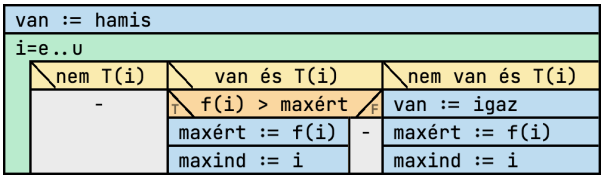
Be: n∈N, hőm∈R[1..n]
Ki: nap∈N
Ef: n>0 és ∀i∈[1..n]:(-100<=hőm[i]<=100)
Uf: nap∈[1..n] és ∀i∈[1..n]:(hőm[nap]>=hőm[i])

4.4. Feltételes maximumkeresés

- Kulcsfontosságú részlet: → N darab valami adott tulajdonságú elemei közül kell megadni a legnagyobbat

Adott egész számok $[e..u]$ intervalluma, $f : [e..u] \to H$ függvény és egy $T : [e..u] \to$ Logikai feltétel. A H halmaz elemén értelmezett egy teljes rendezési reláció. Határozzuk meg, hogy az $[e..u]$ intervallum T feltételt kielégítő elemi közül az f függvény hol veszi fel a legnagyobb értéket, és mondjuk meg mekkora ez az érték!

Specifikáció (sablon):
Be: e∈Z, u∈Z
Ki: van∈L, maxind∈Z, maxért∈H
Ef: -
Uf: van = ∃i[e..u]:(T(i)) és
—- van → (maxind∈[e..u] és
—- maxért = f(maxind) és T(maxind) és
—- ∀i∈[e..u]:(T(i) → maxért ≥ f(i))

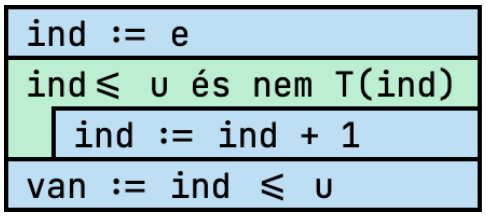


4.5. Keresés

- Kulcsfontosságú részlet → N darab "valami" közül kell megadni egy adott tulajdonságút, ha nem tudjuk, hogy ilyen elem van-e! (Adjunk meg egyet)

Adott az egész számok [e..u] intervalluma és egy T:[e..u] → Logikai feltétel. Határozzuk meg az [e..u] intervallumban balrók az első olyan számot, ha van, amely kielégíti a T feltételt!

Specifikáció (sablon):
Be: e∈Z, u∈Z
Ki: van∈L, ind∈Z
Ef: -
Uf: (van, ind)=KERES(i=e..u, T(i))



Példafeladat:

- Legkisebb osztó

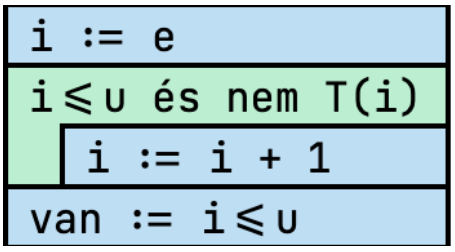
Be: n∈N
Ki: o∈N, van∈L
Ef: n>1
Uf: (van, o)=KERES(i=2..n-1, i | n)

4.6. Eldöntés tétele

- Kulcsfontosságú részlet → N "valami" között van-e adott tulajdonsággal rendelkező elem!

Adott az egész számok [e..u] intervalluma és egy T:[e..u] → Logikai feltétel. Határozzuk meg, hogy van-e az [e..u] intervallumnak olyan elemen amely kielégíti a T feltételt!

Specifikáció (sablon):
Be: e∈Z, u∈Z
Ki: van∈L
Ef: -
Uf: van = ∀i∈[e..u]:(T(i))
Rövidítve: Uf: van=VAN(i=e..u, T(i))



2. Bukott-e

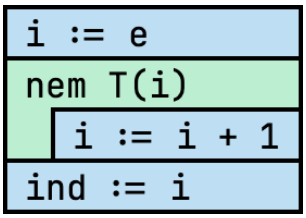
Be: n∈N, jegyek∈N[1..n]
Ki: bukott∈L
Ef: ∀i∈[1..n]:(1<=jegyek[i]<=5)
Uf: bukott=VAN(i=1..n, jegyek[i]=1)

4.7. Kiválasztás tétele

- Kulcsfontosságú részlet → N "valami" közül kell megadni egy adott tulajdonságút, ha tudjuk, hogy ilyen elem biztosan van‼

Adott egy "e" egész szám és egy "e"-től jobra értelmezett `T:Egész`→Logikai feltétel. Határozzuk meg "e"-től jobbra eső első olyan számot, amely kielégíti a T feltételt, ha tudjuk, hogy ilyen biztosan van‼

Specifikáció:



`Be: e∈Z`
`Ki: ind∈Z`
`Ef:∃i∈[e..∞]:(T(i))`
`Uf: ind>=e és T(ind) és ∀i∈[e..ind-1]:`
`(nem T(i))`
rövidítve: `Uf: ind=KIVÁLASZT(i>=e, T(i))`

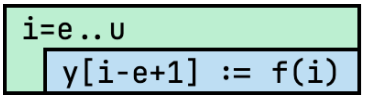
(később magánhangzó-e feladat)

4.8. Másolás tétele

- Kulcsfontosságú részlet → N darab "valamihez" kell hozzárendelni másik N darab "valamit"

Adott egész számok egy `[e..u]` intervalluma és egy `f:[e..u]→ H függvény`. Rendeljük az `[e..u]` intervallum minden értékéhez az f függvény hozzá tartozó értékét!

Specifikáció:



`Be: e∈Z, u∈Z`
`Ki: y∈H[1..u-e+1]`
`Ef: -`
`Uf: ∀i∈[e..u]:(y[i-e+1]=(f(i))`
röviden: `Uf: y=MÁSOL(i=e..u, f(i))`

4.9. Kiválogatás

- Kulcsfontosságú részlet -> N darab valami közül kell megadni az összes adott T tulajdonsággal rendelkezőt!

Példafeladat: Adjuk meg egy év azon napjait, amikor délben nem fagyott!

Be: n∈N, hőm∈R[1..n]
Ki: db∈N, poz∈N[1..db]
Ef: ∀i∈[1..n]:(-100<=hőm[i]<=100)
Uf: db=DARAB(i=1..n, hőm[i]>0) és ∀i∈[1..db]:(hőm[poz[i]]>0) és poz⊆[1..n]

Specifikáció (sablon):
Be: e∈Z, u∈Z
Ki: db∈N, y∈H[1..db]
Ef: -
Uf: (db, y)=KIVÁLOGAT(i=e..u, T(i), f(i))