HÁZI FELADAT

Programozás alapjai 2.

Tervezés

Teszt Elek
ELEK07

2019. április 10.

TARTALOM

1. Feladat

```
   Készítsen generikus tömböt!

Demonstrálja a működést külön modulként fordított tesztprogrammal!
A megoldáshoz NE használjon STL tárolót vagy algoritmust!

A tesztprogramot úgy specifikálja, hogy az parancssoros batch
alkalmazásként (is) működjön, azaz a szabványos bemenetről olvasson, és a
szabványos kimenetre, és/vagy a hibakimenetre írjon!
Amennyiben a feladat teszteléséhez fájlból, vagy fájlokból kell input
adatot olvasnia, úgy a fájl neve *.dat alakú legyen!
```

2. Feladatspecifikáció

A feladat egy generikus tömb elkészítése. A feladat nem specifikálja, hogy ez fix vagy változó méretű tömb legyen, ezért az egyszerűbb megoldás, a fix méret mellett döntöttem[1]. A méretet sablon paraméterként lehet megadni.

A feladat nem írja elő, hogy milyen műveletei legyenek a tömbnek, így az automatikusan létrejövő tagfüggvények mellett (másolás, értékadás, címképzés, létrehozás, megszüntetés) egyedül az indexelést valósítom meg[2]. Hibás indexeléskor *std::out_of_range* kivétel keletkezik.

Csak olyan adatokkal lehet az elkészített tömböt használni, melyre értelmezve van az értékadás művelete.

A teszteléséhez egy olyan programot készítek, ami különböző adattípusokkal létrehozott tömbökkel a standard inputról beolvasott adatok alapján műveleteket végez. A tesztadatok között hibás indexelés is elő fog fordulni.

3. Pontosított feladatspecifikáció

A „megrendelővel” (laborvezető) folytatott konzultáció alapján a specifikációt pontosítani kell. Az így keletkezett specifikáció a végleges, amit a valóságban jóvá kell hagyatni a „megrendelővel”. Ez csak közös megegyezéssel módosítható. Ez fog megvalósulni.
Ennek megadása az előző specifikációhoz mellékelt kiegészítő dokumentációval, vagy a teljes már végleges specifikáció leírásával történhet. Ez utóbbi világosabb, és gyakran egyszerűbb kivitelezni is. Pl:

[1] Ezt egy ilyen egyszerű feladatnál nem fogja elfogadni a laborvezető.
[2] Ezt sem fogadható el. Meg kellene valósítani néhány generikus műveletet (pl. összeadás, merge, rendezés, stb…)

A feladat egy generikus tömb elkészítése, mely egy fix méretű dinamikusan allokált memóriaterületen tárolja az adatokat. A méretet sablonparaméterként lehet megadni.
A generikus osztály, az implicit generálódó tagfüggvényeken kívül megvalósítja az alábbi műveleteket:

- létrehozás,
- megszüntetés,
- másolás,
- értékadás,
- indexelés, nincs indexhatár ellenőrzés
- at, hibás indexeléskor *std::out_of_range* kivétel keletkezik.
- elemek száma és kapacitás lekérdezése (size, max_size)
- random access iterator létrehozása (begin, end)

A felsorolt műveletek a konstans környezetben is értelmezhető változatát is megvalósítom.

Csak olyan adatokkal lehet az elkészített tömböt használni, melyre értelmezve van az értékadás művelete.

A teszteléséhez egy olyan programot készítek, ami különböző adattípusokkal létrehozott tömbökkel a standard inputról beolvasott adatok alapján műveleteket végez. A tesztadatok között hibás indexelés is elő fog fordulni.

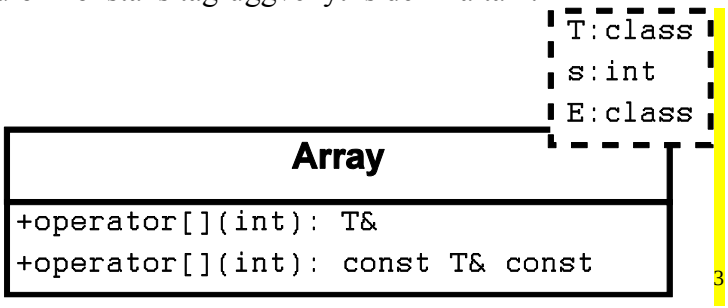
4. Terv

A feladat egy objektum és a tesztprogram megtervezését igényli.

4.1. Objektum terv

A generikus tömböt egyetlen sablonnal fogom megvalósítani. A sablon sablonparaméterként veszi át tömb elemeinek típusát és a tömb méretét, valamint azt az osztályt, amit a kivételkezelésben használ. A könnyebb felhasználhatóság érdekében a sablonparaméterként átvett méretnek és hibaosztálynak alapértelmezése is van.

Az indexeléshez külön konstans tagfüggvényt is definiáltam.



[3]

4.2. Algoritmusok

[3] A rajz nincs összhangban a szöveges leírással! Így élesben nem fogadható el!

4.2.1. Tartományon kívüli elemek lekérése

A feladat egyetlen összetett algoritmusa az operator[] paraméterének ellenőrzése.[4]

```
if i < 0 then
    exception    a l u l i n d e x e l é s
else  if i ≥ N  then
    exception    t ú l i n d e x e l é s
else
    normál működés
```

Mindkét hibaesetben a sablonparaméterként megadott osztályból generált objektumot dob az operátor. Ennek alapértelmezett értéke az *out_of_range* osztály.

4.2.2. Tesztprogram algoritmusai.

A tesztprogram a standard inputról file végéig olvas. Az első beolvasott adat egy teszteset sorszámot jelent. Ezt követően egy megjegyzés lehet az adott sorban. A beolvasott szám dönti el, hogy melyik teszteset fut a megjegyzés pedig az adott tesztesetre vonatkozhat.
Az 1-3. tesztesetek index és érték párokat olvasnak be, melyekkel indexelik a létrehozott tömböt. Az indexhatár megsértést is teszteli.

5. Megjegyzések a kidolgozott minta HF-hez

1. A minta HF kidolgozását a továbbiakban az első, elfogadhatatlanul egyszerű specifikáció alapján oldjuk meg.

2. A házi feladok tesztelési lehetőségeit bemutatandó, a végleges megoldásban (Kesz) egy külön teszteset (4. eset) mutatja be szöveges adtafájlok feldolgozását. Általános probléma, hogy a szöveges fájlokat a különböző operációs rendszerek eltérően kezeik. Vannak rendszerek (UNIX/LINUX), melyekben nem különböztetik meg fájlokat tartalmuk alapján. Más rendszerekben a szöveges fájlokat eltérően kezelik (ld. C nyelvnél tanult bináris és text megnyitási módokat). Ez mindaddig nem jelent problémát, amíg az adatok a feldolgozási rendszerrel azonos típusú operációs rendszerből származnak. Ellenkező esetben ezt a feldolgozásnál figyelembe kell venni.

   A 4. teszteset kapcsán megfigyelhető, hogy egy Windows 8-ból származó, a sorok végén \r\b sorozatot tartalmazó fájl beolvasásakor a Jporta operációsrendszere (LINUX) a \r karaktert nem „nyeli” le.

   A példához mellékelt *cp::getline()* a probléma egy megoldási lehetőségét mutatja be. Figyelje meg a beolvasott sor memóriaképét (*memtrace::mem_dump()*)!

[4] Ez az állítás sem feltétlenül igaz!