HÁZI FELADAT

Szoftver laboratórium 2.

Végleges

Teszt Elek
ELEK07

2011. március 22.

TARTALOM

1. Feladat

```
Szoftver laboratórium II. házi feladat
Teszt Elek (ELEK07) részére:

   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. Pontosított 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. 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. Hibás indexeléskor 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.

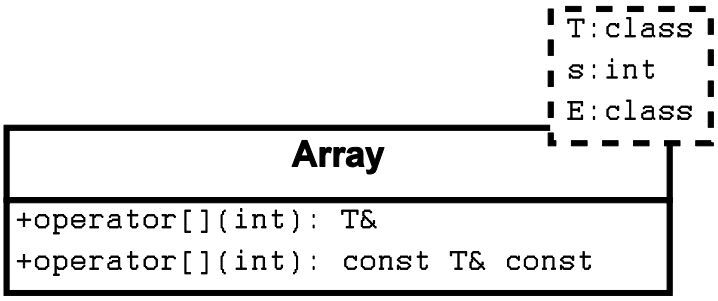
3. Terv

A feladat egy objektum és a tesztprogram megtervezését igényli.

3.1. 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.



1. ábra: A tömb sablonosztály osztálydiagramja

3.2. Algoritmusok

3.2.1. Tartományon kívüli elemek lekérése

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

```
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.

3.2.2. Tesztprogram algoritmusai.

A tesztprogram a standard inputról egy teszteset sorszámot, majd index és érték párokat olvas be. Az elsőként beolvasott szám dönti el, hogy melyik teszteset fut. Ezt követően a tesztesetnek megfelelő index és értékpárokat olvas, melyekkel indexeli a létrehozott tömböt.

4. Megvalósítás

A feladat megoldása egyetlen sablonosztály és egy tesztprogram elkészítését igényelte. A sablon felhasználja az *std::string, std:sstream* és az *std::exception* osztályokat. Az osztályok végleges interfésze a tervezési lépésben meghatározott módon alakult.[1] Csupán annyi változott a tervezéshez képest, hogy a 3. sablonparamétert logikusabbnak éreztem típus paraméterként átvenni osztályparaméter helyett. A tesztelés algoritmusai azonban kicsit módosultak a tesztadatok pontosítása során.

A sablon forrása a *gen_array3.hpp*, míg a tesztprogram a *gen_array3_main.cpp* állományba került. A továbbiakban a bemutatom a fontosabb interfészeket és algoritmusokat a program forrása alapján generált dokumentáció felhasználásával.

[1] Nem lenne baj, az sem, ha kiderült volna, hogy módosítani kellett utólag az interfészen. A lényeg, hogy a végleges legyen ledokumentálva.

4.1. Az elkészített osztálysablon bemutatása

A sablon a paraméterként átvett méretű tömbben tárolja a generikus adatokat. Így nincs dinamikus adata. Ennek köszönhetően minden implicit függvénye jól működik, nincs szükség átdefiniálásra. A generikus adatról feltételezzük, hogy van, és helyesen működik a:

- default konstruktora
- másoló konstruktora
- értékadó operátora
- destruktora

A 3. sablon paraméterként megadott típussal szemben az a követelmény, hogy legyen *std::string* paraméterű konstruktora. Indexelési hiba esetén a megadott típus egy példányát a hibára utaló szöveggel példányosítja, és ezt dobja hibaként.

class Array< T, s, E > (gen_array3.hpp)

Tömb osztálysablon

Paraméterek:

T - adattípus
s - méret, default: 50
E - hibaosztály (indexelési hiba esetén hívódik) defalult: std::out_of_range

Publikus tagfüggvényei:

- T & **operator[]** (int i)
  index operátor
  Paraméterek:
  i - index
  Visszatérési érték:
  - referencia az adott indexű elemre
  - hibás indexérték esetén E() kivételt dob

Definíció a(z) gen_array3.hpp fájl 36. sorában.

```
36                                              {
37     if (i < 0 || i >= s) {
38         // megpróbálunk összerakni egy értelmes hibajelzést
39         // persze ez így nagyon fapados és ronda
40         std::stringstream err;
41         err << " **Indexhiba** idx: " << i << " [0," << s-1 << "]";
42         throw E(err.str()); // template paraméterként kapott osztály példányát dobjuk
43     }
44     return t[i];    // megfelelõ elem referenciájával térünk vissza
45 }
```

- const T & **operator[]** (int i) const
  Konstans példány index operátora
  Paraméterek:
  i - index
  Visszatérési érték:
  - referencia az adott indexű elemre
  - hibás indexérték esetén E() kivételt dob

Definíció a(z) gen_array3.hpp fájl 49. sorában.

```
49                                              {
50     if (i < 0 || i >= s) {
51         // megpróbálunk összerakni egy értelmes hibajelzést
52         // persze ez így nagyon fapados és ronda
53         std::stringstream err;
54         err << " **Indexhiba** idx: " << i << " [0," << s-1 << "]";
55         throw E(err.str()); // template paraméterként kapott osztály példányát dobjuk
56     }
57     return t[i];    // megfelelõ elem referenciájával térünk vissza
58 }
```

4.2. Tesztprogram bemutatása

A *gen_array3_main.cpp* fájlban levő tesztprogram a standard inputról beolvasott egész szám alapján különböző teszteseteket hajt végre. A főprogram az egész szám beolvasása után a sor hátralevő részét eldobja, majd meghívja a megfelelő tesztesetet, ami önállóan kezeli a bemenetet és kimenetet.

A demonstrációs céllal készített tesztprogram a Cporta tesztelési eljárásaihoz igazodva nem csak a standard inputról olvas adatokat, hanem egy .dat fájlból is.[2] Ezt azonban az eredeti feladat (generikus tömb) teszteléséhez nem használja fel (test_4).

A főprogram a *setlocale()* függvényhívással a futási környezet default nyelvi beállításait állítja be.

A kivételeket is a főprogram kezeli. A saját kivételosztály teszteléséhez létrehozott osztály a következő:

class BajVan

Saját hibaosztály a teszteléshez

Publikus tagfüggvényei:

- **BajVan** (const string& s)

  Paraméterek:

  s – nem használt

4.2.1. Az egyes teszteseteket megvalósító függvények:

void test_1 ()

1. tesztesetet megvalósító függvény: 50 elemű *int* tömböt hoz létre és ennek elemeit írja ill. olvassa a standard input alapján.

void test_2 ()

2. tesztesetet megvalósító függvény: 100 elemű *double* tömböt hoz létre és ennek elemeit írja ill. olvassa a standard input alapján.

void test_3 ()

3. tesztesetet megvalósító függvény: 11 elemű *string* tömböt hoz létre és ennek elemeit írja ill. olvassa a standard input alapján.

void test_4()

4. tesztesetet megvalósító függvény: Az összes magyar ékezetes karaktert tartalmazó szöveg kiírása után megnyitja a *valami_adat.dat* fájlt, és annak tartalmát kiírja a standard kimenetre.

5. Tesztelés

[2] A NHF automatikus ellenőrzését segítő felületen 20 db fájl és 5 bemenet adható meg. A 20 fájl mindegyike lehet fordítandó forrás, nem fordítandó forrás, dokumentáció (HTML, PDF), vagy adat (DAT). Ez utóbbi a program futtatásakor elérhető. Az ún. bemenetek megadásával lehet definiálni a teszteseteket. A feltöltött programot a Cporta annyiszor futtatja le, ahány bemenetet megadtunk (1-5). Minden futás indításakor a .dat fájlt a munkakatalógusba másolja és a standard inputot a megfelelő bemenetről veszi.

A feladat során elkészített generikus osztály tesztjeit a következő szempontok alapján terveztem meg:

- Interfész teszt.
  - paraméterek és default paraméterek tesztje
  - generikus működés tesztje
  - alkalmazás konstans objektumra
- Funkcionális teszt
  - a tömbbe beírt adat kiolvasható legyen
  - hibás indexelés esetén legyen kivétel

5.1. Interfész teszt

Az előző fejezetben bemutatott tesztprogram 3 különböző adatra, különböző paraméterekkel és konstans objektummal is vizsgálja az elkészített osztályt. A fordítás hibamentes, ami az interfész tesztnek való megfelelést igazolja.

5.2. A funkcionális tesztek

teszt_1

```
1       // teszt sorszáma. A következő sorokban pedig (index érték) párok
2 3
1 556
0 45
2 88
23 11
-1 4
-------- stdout --------

1. teszteset indítása:
default (50) elemű int tömb
arr[2]=3
arr[1]=556
arr[0]=45
arr[2]=88
arr[23]=11
-------- stderr --------
 **Indexhiba** idx: -1 [0,49]

--------- end ----------
```

teszt_2

```
2       // teszt sorszáma. A következő sorokban pedig (index érték) párok
49 1.3
23 78.5
3  123.56
150 2.5
-------- stdout --------

2. teszteset indítása:
100 elemű double tömb
arr[49]=1.3
arr[23]=78.5
arr[3]=123.56

-------- stderr --------
```

```
 **Indexhiba** idx: 150 [0,99]

--------- end ----------
```

teszt_3

```
3  // teszt sorszáma. A következő sorokban pedig (index érték) párok
1  alma
2  korte
10 uborka
13 hiba
9  idenemjut
-------- stdout --------

3. teszteset indítása:
11 elemű string sajat kivételosztállyal
arr[1]=alma
arr[2]=korte
arr[10]=uborka

-------- stderr --------
Saját kivétel jött

--------- end ----------
```

teszt_4

```
4       // teszt sorszáma.
Ez csak demo a .dat file bemutatásához
-------- stdout --------

0. teszteset indítása:
A mintafeladat bemutatja az ékezetes karakterek használatát is.
Az alábbi szöveg windows 1250-es (ISO 8859-2/latin 2) kódolású:
árvíztűrőtükörfúrógép ÁRVÍZTŰRŐTÜKÖRFÚRÓGÉP
Konstans tömb 0. eleme: 12

-------- stderr --------

--------- end ----------
```

5.3. Memóriakezelés tesztje

A memóriakezelés ellenőrzését a laborgyakorlatokon használt MEMTRACE modullal végeztem. Ehhez minden önálló fordítási egységben include-oltam a ”memtrace.h” állományt a standard fejlécállományok után. Memóriakezelési hibát nem tapasztaltam a futtatások során.

5.4. Lefedettségi teszt

A funkcionális tesztek a program minden ágát lefedték.
A Cporta rendszer 4 olyan kódrészletet jelölt meg, ami nem futott a programban. Ezek a következők:

- az 1., 2. és 3. tesztesetet megvalósító függvények utolsó sorai, amelyek a hibás input miatt nem futottak,
- valamint az ismeretlen kivétel elkapásához tartozó („Nagy baj van”) ág (2. ábra).

```
/// TESZT 1
void test_1() {
    Array<int> arr;                 // default (50) elemű int tömb
    int idx;
    int val;
    cout << "default (50) elemű int tömb" << endl;
    while ( cin >> idx >> val) {
      arr[idx] = val;
      cout << "arr[" << idx << "]=" << arr[idx] << endl;
    }
}

/// TESZT 2
void test_2() {
    Array<double, 100> arr;         // 100 elemű double tömb
    int idx;
    double val;
    cout << "100 elemű double tömb" << endl;
    while ( cin >> idx >> val) {
      arr[idx] = val;
      cout << "arr[" << idx << "]=" << arr[idx] << endl;
    }
}

/// TESZT 3
void test_3() {
    Array<string, 11, BajVan> arr;  // 11 elemű string saját kivétellel
    int idx;
    string val;
    cout << "11 elemű string sajat kivételosztállyal" << endl;
    while ( cin >> idx >> val) {
      arr[idx] = val;
      cout << "arr[" << idx << "]=" << arr[idx] << endl;
    }
}
```

●●●

```
    // kivétel elkapása
    } catch (exception& e) {
        // kiírjuk, hogy milyen kivétel jött
        cerr << e.what() << endl;
    } catch (BajVan&) {
        cerr << "Saját kivétel jött" << endl;
    } catch (...){
        cout << "**** Nagy baj van! ****"; << endl;
    }
#ifdef _CRTDBG_MAP_ALLOC
```

2. ábra: Cporta lefedettségi teszt eredménye

6. Mellékletek

6.1. gen_array3.cpp

```
00011 #include <sstream>
00012
00017 template <class T, int s = 50, typename E = std::out_of_range>
00018 class Array {               // osztálysablon
00019     T t[s];
00020 public:
00025     T& operator[](int i);
00026
00031     const T& operator[](int i) const;
00032 };
00033
00035 template <class T, int s, typename E>
00036 T& Array<T, s, E>::operator [](int i) {
00037     if (i < 0 || i >= s) {
00038         // megpróbálunk összerakni egy értelmes hibajelzést
00039         // persze ez így nagyon fapados, és ronda
00040         std::stringstream err;
00041         err << " **Indexhiba** idx: " << i << " [0," << s-1 << "]";
00042         throw E(err.str()); // template paraméterként kapott osztály példányát dobjuk
00043     }
00044     return t[i];    // megfelelő elem referenciájával térünk vissza
00045 }
00046
00048 template <class T, int s, typename E>
00049 const T& Array<T, s, E>::operator [](int i) const {
00050     if (i < 0 || i >= s) {
00051         // megpróbálunk összerakni egy értelmes hibajelzést
00052         // persze ez így nagyon fapados, és ronda
00053         std::stringstream err;
00054         err << " **Indexhiba** idx: " << i << " [0," << s-1 << "]";
00055         throw E(err.str()); // template paraméterként kapott osztály példányát dobjuk
00056     }
00057     return t[i];    // megfelelő elem referenciájával térünk vissza
00058 }
```

6.2. gen_array3_main.cpp

```
00032 #include <iostream>
00033 #include <fstream>
00034 #include <stdexcept>
00035 #include <locale>
00036 #include "memtrace.h"           // a standard headerek után kell
00037 #include "gen_array3.hpp"       // sablon
00038 using namespace std;
00039
00041 class BajVan {
00042 public:
00044     BajVan(const string&) {}
00045 };
00046
00047 const char msg[] = "\
00048 A mintafaladat bemutatja az ékezetes karakterek használatát is.\n\
00049 Az alábbi szöveg windows 1250-es (ISO 8859-2/latin 2) kódolású:\n\
00050 árvíztűrőtükörfúrógép ÁRVÍZTŰRŐTÜKÖRFÚRÓGÉP";
00051
00052
00056 void test_4() {
00057     cout << msg << endl;
00058     char buf[100];
00059     fstream os;
00060     os.exceptions(ifstream::badbit);
00061
00062     // megnyitás
00063     os.open("valami_adat.dat", fstream::in);
00064
00065     // belvasas
```

```
00066     while (os.getline(buf, sizeof(buf)))
00067        cout << buf << endl;
00068     os.close();
00069 }
00070
00072 void test_1() {
00073     Array<int> arr;                 // default (50) elemű int tömb
00074     int idx;
00075     int val;
00076     cout << "default (50) elemű int tömb" << endl;
00077     while ( cin >> idx >> val) {
00078       arr[idx] = val;
00079       cout << "arr[" << idx << "]=" << arr[idx] << endl;
00080     }
00081 }
00082
00084 void test_2() {
00085     Array<double, 100> arr;         // 100 elemű double tömb
00086     int idx;
00087     double val;
00088     cout << "100 elemű double tömb" << endl;
00089     while ( cin >> idx >> val) {
00090       arr[idx] = val;
00091       cout << "arr[" << idx << "]=" << arr[idx] << endl;
00092     }
00093 }
00094
00096 void test_3() {
00097     Array<string, 11, BajVan> arr;  // 11 elemű string saját kivétellel
00098     int idx;
00099     string val;
00100     cout << "11 elemű string sajat kivételosztállyal" << endl;
00101     while ( cin >> idx >> val) {
00102       arr[idx] = val;
00103       cout << "arr[" << idx << "]=" << arr[idx] << endl;
00104     }
00105 }
00106
00110 int main() {
00111     setlocale(LC_ALL, "");  // a rendszer környezeti változójában megadott nyelvi
környezet beállítása
00112     try {
00113       int nr;
00114       cin >> nr;        // hanyadik teszteset
00115       cin.ignore(500, '\n'); // eldobunk max. 500 karaktert a sor végéig
00116       cout << endl << nr << ". teszteset indítása:" << endl;
00117       switch (nr) {
00118         case 4:
00119             test_4();   // csak demo, nincs funkciója a feladatban
00120         break;
00121
00122         case 1:
00123             test_1();   // default (50) elemű int tömb
00124         break;
00125
00126         case 2:
00127             test_2();   // 100 elemű double tömb
00128         break;
00129
00130         case 3:
00131             test_3();   // 11 elemű string saját kivételosztállyal
00132         break;
00133       }
00134
00135       // konstans tömb indexelésének tesztje
00136       Array<int> ia;
00137       ia[0] = 12;
00138       const Array<int> ia2 = ia;
00139       cout << "Konstans tömb 0. eleme: " << ia2[0] << endl;
00140
00141     // kivétel elkapása
00142     } catch (exception& e) {
00143         // kiírjuk, hogy milyen kivétel jött
00144         cerr << e.what() << endl;
00145     } catch (BajVan&) {
00146         cerr << "Saját kivétel jött" << endl;
00147     } catch (...) {
```

```
00148          cerr << "*** Nagy baj van! ****" << endl;
00149      }
00150
00151      // itt minden memóriaterületnek, objektumnak fel kell szabadulnia, hogy a
00152      // memóriaszivárgás ellenőrizhető legyen!
00153      _CrtDumpMemoryLeaks();  // ellenőrzi, hogy volt-e memóriaszivárgás
00154
00155      return 0;
00156 }
```