FEJLETT PROGRAMOZÁSI NYELVEK, 2009
5. GYAKORLAT – Osztálysablonok készítése

Cél:

- Osztálysablonok készítése
- Referencia típusú attribútum
- Konstruktortípusok: implicit, másoló és más
- Operátorok túlterhelése (index és értékadó)
- Konstans tagfüggvények (`const`)
- Ismeretségi kapcsolat osztályok között (`Array<T>` és `ArrayIterator<T>`)
- Beágyazott osztály készítése

1. Osztálysablon

Készítsünk egy Array *osztálysablont*, amely a C stílusú tömbhöz hasonlóan működik. Osztálysablon esetén nem készítünk külön fejállományt és implementációs állományt, hanem csak egyiket a kettő közül, amely mindent tartalmaz. Ellenkező esetben összeszerkesztési hibát kapunk (linker error).

Array.h

```
template < class T >
class Array{
        T * data;
        int size;
public:
        Array( int size= 5 );
        Array( const Array<T>& a );
        Array( T* first, T* last );
        Array<T>& operator=( const Array<T>& a );
        int getSize() const;
        T& operator[]( int index );
        T operator[]( int index ) const;
        ~Array();
};
```

Hozzunk létre egy 10 elemű **string** típusú elemeket tartalmazó Array példányt, töltsük fel a ”**string1”, ”string2”,...”string10”** kezdőértékekkel, majd írassa ki a standard kimenetre. A kért karakterláncok előállítására használjuk a **`stringstream`** osztályt (**`#include <sstream>`**).

2. Iterátor– külső osztályos megvalósítás

Készítsünk az Array osztályhoz egy *„Java stílusú”* külső osztállyal megvalósított iterátorosztályt (*osztálysablon lesz!!!*). A külső osztályos megvalósítás előnye, hogy nem kell módosítani a tároló osztályt, arra rá lehet építeni.

FEJLETT PROGRAMOZÁSI NYELVEK, 2009
5. GYAKORLAT – Osztálysablonok készítése

ArrayIterator.h

```
template <class T>
class ArrayIterator{
        //Adattagok
public:
        ArrayIterator( Array<T>& a ){//kiegesziteni}
        bool hasNext(){//kiegesziteni}
        T next(){//kiegesziteni}
};
```

Vigyázat!!!
*Az iterátor (`ArrayIterator`) referencia szerint fér hozzá a tömb (`Array`) objektumhoz. A tömböt ábrázoló adattagot is **referencia típusú adattagnak** kell deklarálni. A referencia típusú adattagokat kötelező a konstruktorban inicializálni.*

3. Iterátor– beágyazott osztályos megvalósítás

Készítsünk az `Array` osztályhoz egy C++ stílusú, beágyazott osztállyal megvalósított iterátor osztályt. Az `Array` osztályt lássuk egészítsük ki egy `begin()` és egy `end()` művelettel, úgy, hogy lehetővé váljon a következő stílusú használat:

```
int x[] ={1, 2, 3, 4, 5};
Array<int> a( x+2, x+5 );
Array<int>::iterator it;
for( it= a.begin(); it != a.end(); ++it )
        cout<< *it<<' ';
cout<<endl;
```

Egy lehetséges osztály váz a következő lenne:

```
template < class T >
class Array{
      //....
public:
      class iterator{//kiegesziteni};
      iterator begin(){//kiegesziteni}
      iterator end  (){//kiegesziteni}
};
```

Szükséges-e új beágyazott osztályt bevezetni, vagy megoldható ez esetben egyszerűbben is? Ha igen, hogyan?