Unique Pointer to Array

We can have a unique pointer pointing to the array of a number of data elements of any types by using `‘MK_U_ARRAY<Type>(unsigned int) ‘` as a library function provided by the “mixutil.h “ header file.

The type of which to match to the returned output of `‘MK_U_ARRAY<Type>(Nx)’` is a wrapper of the standard `std::unique_ptr<T[]>` which is redefined to be `‘U_ARRAY<Type>’`.

For example:

```
U_ARRAY<int> unp = MK_U_ARRAY<int>(3); // similar to int* p = new int[3];

U_ARRAY<Bucket> ubc = MK_U_ARRAY<Bucket>(2); // Bucket* pb = new Bucket[2];
```

Shared Pointer to Array

The same scenario is applied to the shared pointer for holding any number of data elements of any types by using `‘MK_S_ARRAY<Type>(unsigned int) ‘`provided as a library function from the “mixutil.h “ header file.

For example:

```
S_ARRAY<int> sp = MK_S_ARRAY<int>(5);

S_ARRAY<Bucket> sb = MK_S_ARRAY<Bucket>(1);
```

Initializing the Array Owned by a Unique Pointer and a Shared Pointer

To easy initialize the unique array, we don't have to use the **"for..loop"** statement for accessing each element of the array and set it to a new data value or object. Instead we can use the static object's function of **"_init_p<ptr> "** which has the static member method **"initialize()".** And this is also completely provided by the "mixutil.h" header file.

The arguments to the **"initialize()"** method are first, a pointer to the target array to which data is to be stored and second is the **std::initializer_list<T>** parameter type which can be directly initialized a set range of data elements within its pair of curly braces **" { … } ".**

The definition and implementation signature of the "_init_p<ptr>" can be seen as follows.

```
template < class Ty >
using iList = std::initializer_list<Ty>;
.     .     .
.     .     .

template < class P >
struct _init_p {

      using value_type = typename std::remove_pointer_t<P>;
      using list_iterator = typename iList<value_type>::iterator;
      using LIST = iList<value_type>;

      inline static void initialize(P _p, LIST const& _ls) {
            for (list_iterator _it = _ls.begin(); _it != _ls.end(); _it++)
                  *_p++ = *_it;
      }

};
```

For example:
Considering from the above given examples:

```
U_ARRAY<int> unp = MK_U_ARRAY<int>(3);

_init_p<int*>::initialize( unp.get(),{ 200, 400, 600 } ); //initializes the unp


U_ARRAY<Bucket> ubc = MK_U_ARRAY<Bucket>(2);

_init_p<Bucket*>::initialize( ubc.get(),{ Bucket("0"), Bucket("1") }); //initializes the ubc


S_ARRAY<int> sp = MK_S_ARRAY<int>(5);

_init_p<int*>::initialize(sp.get(),{ 110, 509, 315, 1001, 790 });


S_ARRAY<Bucket> sb = MK_S_ARRAY<Bucket>(1);

_init_p<Bucket*>::initialize(sb.get(), { Bucket("One") } );
```

Printing the Content of the Initialized Array

To print all the contents of the array just initialized in the above given, again, we don't have to use any looping mechanism to access to each element of the array. Instead there is a **"`smart_print(first, last)`"** library function provided by the "mixutil.h" header file.

For example are: (considering the above given)

```
smart_print(unp.get(), unp.get() + 3);      std::cout << ub[0].data() << "\n\n";

smart_print(ubc.get(), ubc.get() + 2);

smart_print(sp.get(), sp.get() + 5);
```