Operations on arrays:

The basic operations provide for creating, destroying and updating array are shown below. Detailed explanation is not given as these function are just create and destroying array and just positioning the various pointers of the memory pool and array at the right place after insertion of elements...

Creating an Array: ngx_array_create

Function definition:

```
ngx_array_t *
ngx_array_create(ngx_pool_t *p, ngx_uint_t n, size_t size)
{
    ngx_array_t *a;
    a = ngx_palloc(p, sizeof(ngx_array_t));
    if (a == NULL) {
        return NULL;
    }
    if (ngx_array_init(a, p, n, size) != NGX_OK) {
        return NULL;
    }
    return a;
}
```

Parameters :

1. p : The address of the pool from which the memory will be allocated.
2. n : number of elements for which memory is to be allocated.
3. size : The size in bytes of each element

Return value:

A pointer to the newly allocated array if successful otherwise NULL if allocation fails.Useful to check whether creation was successful or not.

Eg :

```
        ngx_array_t  new_arr = ngx_array_create(pool,10,sizeof(int));
        if( new_arr == NULL )
                return  NGX_ERROR
```

Allocating memory to already created array : ngx_array_init

Function Definition :

```
static ngx_inline ngx_int_t
ngx_array_init(ngx_array_t *array, ngx_pool_t *pool, ngx_uint_t n, size_t size)
{
    /*
     * set "array->nelts" before "array->elts", otherwise MSVC thinks
     * that "array->nelts" may be used without having been initialized
     */
    array->nelts = 0;
    array->size = size;
    array->nalloc = n;
    array->pool = pool;
    array->elts = ngx_palloc(pool, n * size);
    if (array->elts == NULL) {
        return NGX_ERROR;
    }
    return NGX_OK;
}
```

Ngx_array_init is used if the memory for ngx_array_t is already allocated and if we want to allocate memory for its elements.

Parameters :
array : pointer to already allocated ngx_array_t structure
p : The pointer to the pool which is used to allocate memory for the array structure and elements
n: number to elements to allocate memory for
size : The size of individual element of the array
Return value :
NGX_OK if the allocation is successful otherwise NGX_ERROR

Push a element into the array :   ngx_array_push

This function pushes a new element to the end of the array.

Function Definition :

```
void *
ngx_array_push(ngx_array_t *a)
{
    void        *elt, *new;
    size_t       size;
    ngx_pool_t  *p;

    if (a->nelts == a->nalloc) {
        size = a->size * a->nalloc;
        p = a->pool;

        if ((u_char *) a->elts + size == p->d.last
            && p->d.last + a->size <= p->d.end)
        {
            p->d.last += a->size;
            a->nalloc++;
        } else {
            /* allocate a new array */

            new = ngx_palloc(p, 2 * size);
            if (new == NULL) {
                return NULL;
            }

            ngx_memcpy(new, a->elts, size);
            a->elts = new;
            a->nalloc *= 2;
        }
    }
    elt = (u_char *) a->elts + a->size * a->nelts;
    a->nelts++;
    return elt;
}
```

Parameter :
a : Pointer to array to which the element is to be added
Return value :
The pointer to the allocated element if successful otherwise NULL

Push n elements into the array :

Function Definition :

```
void *
ngx_array_push_n(ngx_array_t *a, ngx_uint_t n)
{
    void        *elt, *new;
    size_t       size;
    ngx_uint_t   nalloc;
    ngx_pool_t  *p;
    size = n * a->size;
    if (a->nelts + n > a->nalloc) {
        p = a->pool;
        if ((u_char *) a->elts + a->size * a->nalloc == p->d.last
            && p->d.last + size <= p->d.end)
        {
            p->d.last += size;
            a->nalloc += n;
        } else {
            nalloc = 2 * ((n >= a->nalloc) ? n : a->nalloc);
            new = ngx_palloc(p, nalloc * a->size);
            if (new == NULL) {
                return NULL;
            }

            ngx_memcpy(new, a->elts, a->nelts * a->size);
            a->elts = new;
            a->nalloc = nalloc;
        }
    }

    elt = (u_char *) a->elts + a->size * a->nelts;
    a->nelts += n;
    return elt;
}
```

This function is used to insert n elements to the end of the array.

Parameters :

a : Pointer to array to which the element is to be added.

n : No of elements to be pushed.

Return Value :

The function return a pointer to the elements of the array

Destroying an array :

After using the array we can free the space allocated to the array using the ngx_array_destroy().

Source Code :

```
void
ngx_array_destroy(ngx_array_t *a)
{
    ngx_pool_t  *p;

    p = a->pool;

    if ((u_char *) a->elts + a->size * a->nalloc == p->d.last) {
        p->d.last -= a->size * a->nalloc;
    }

    if ((u_char *) a + sizeof(ngx_array_t) == p->d.last) {
        p->d.last = (u_char *) a;
    }
}
```

Parameters:

a : pointer to the array to be destroyed.

After freeing the array space the various pointers of the memory pool are set to their new positions.