# 【C++】Day53(3)

| | |
|---|---|
| Class | C++ |
| Date | @February 7, 2022 |
| Material | |
| Series Number | |
| Summary | |

## 【Ch12】Dynamic Memory

### 12.2 Dynamic Arrays

The language defines a second kind of new expression that allocates and initializes an array of objects. The library includes a template class named `allocator` that lets us separate allocation from initialization

*Best Practices: Most applications should use a library container rather than dynamically allocated arrays. Using a container is easier, less likely to contain memory-management bugs, and is likely to give better performance.*

#### 12.2.1 new and Arrays

We ask `new` to allocate an array of objects by specifying the number of objects to allocate in a pair of square brackets after a type name. In this case, `new` allocates the requested number of objects and (assuming the allocation succeeds) returns a pointer to the first one:

```
//call get_size to determine how many ints to allocate
int *pia = new int[get_size()]; //pia points to the first of these ints
```

We can also allocate an array by using a type alias to represent an array type. In this case, we omit the brackets:

```
typedef int arrT[42];
int *p = new arrT; //allocates an array of 42 ints; p points to the first one
```

### *Allocating an Array Yields a Pointer to the Element Type*

When we use `new` to allocate an array, we do not get an object with an array type. Instead, we get a pointer to the element type of the array. Even if we use a type alias to define an array type, `new` does not allocate an object of array type. `new` returns a pointer to the element type.

Because the allocated memory does not have an array type, we cannot call `begin` or `end` on a dynamic array. These functions use the array dimension to return pointers to the first and one past the last elements, respectively.

For the same reason, we also cannot use a range for to process the elements in a (so-called) dynamic array.

*Warning: It is important to remember that what we call a dynamic array does not have an array type.*

### *Initializing an Array of Dynamically Allocated Objects*

By default, objects allocated by new-whether allocated as a single object or in an array, are default initialized. We can value initialize the elements in an array by following the size with an empty pair of parentheses:

```
int *pia = new int[10];
int *pia2 = new int[10](); //block of ten ints value initialized to 0
string *psa = new string[10];
string *psa2 = new string[10](); //block of ten empty strings
```

Under the new standard, we can also provide a braced list of element initializers:

```
// block of ten ints each initialized from the corresponding initializer
int *pia3 = new int[10]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
// block of ten strings, the first four are initialized from the given initializers.
// remaining elemenst are value initialized
string *psa3 = new string[10]{"a", "an", "the", string(3, 'x')};
```

If there are more initializers than the given size, then the new expression fails and no storage is allocated. In this case, new throws an exception of type `bad_array_new_length`.

### *It Is Legal to Dynamically Allocate an Empty Array*

We can use an arbitrary expression to determine the number of objects to allocate:

```
size_t n = get_size(); //get_size returns the number of elements needed
int *p = new int[n]; //allocate an array to hold the elements
for(int *q = p; q != p + n; ++q)
```

Our code works find if n equals to 0.