References

Chapter 9.8 – Dynamic Memory Allocation

Starting Out With C++. From Control Structures through Objects (Eighth Edition)

When using Dynamically Allocated Memory in array, there are mainly two different ways:

(Type 1) **Dynamically allocated array** - An entire array is dynamically allocated at a time.

(Type 2) **Array for storing dynamically allocated variables or objects** – Normally the array does not store objects directly. Instead, the array is used for storing the **pointers** which link to dynamically allocated variables or objects. The objects are created in runtime depending on the needs.

Type 1 - Dynamically allocated array

As shown below the memory for a variable can be dynamically allocated in runtime.

```
int *p = nullptr;

p = new int;

*p = 100;
```

And it can be deleted after it is no longer needed.

```
delete p;
```

In a similar way memory for an array can also be dynamically allocated like the following.

```
const int SIZE = 5;
```

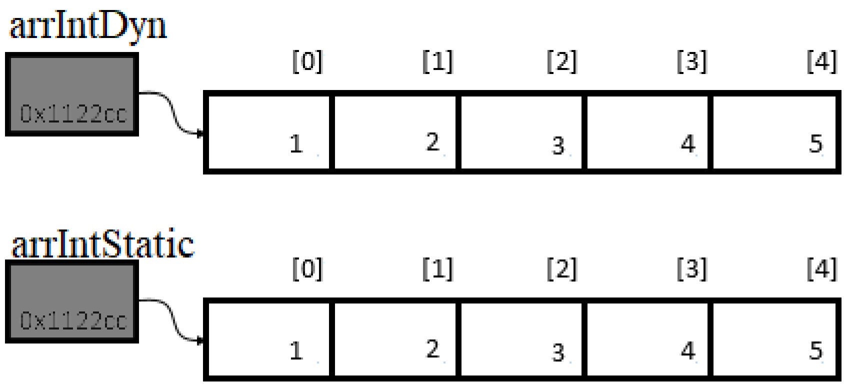
```
    int  *arrIntDyn = nullptr; // arrIntDyn is an integer pointer

    arrIntDyn = new int[SIZE]; // arrIntDyn points to the first element of the array
```

The third statement above indicates that arrIntDyn is the name of an integer array of size 5. Compared with the statically allocated array:

```
    int arrIntStatic[SIZE];
```

we know that arrIntStatic is also a name of the array. Both the prefixed "*" of arrIntDyn and the post-fixed "[ ]" of arrIntStatic lets us know that the variables are similar in a certain way. They store address, not regular value, such as integer or string.

After being assigned some int values they are like the following figures in the memory:



That is, the ways that they are represented in the memory are exactly the same. The only difference is their life span. The statically allocated array is released from the memory when it is out of scope, which the dynamically allocated array has to be deleted in the following way.

```
   delete []arrIntDyn;
```

The prefixed [] indicates that the variable arrIntDyn is an array, not a regular variable. Thus, system knows that the memory allocated for the entire array has to be released.

Exercise 1

Use sizeof() function to get the sizes of the two arrays, i.e. arrIntDyn and arrIntStatic. The two arrays are declared with a const SIZE of 5. Will they return the same value when applying sizeof() to them? Could you tell the reason why?

Type 2 - Array for storing dynamically allocated variables or objects

When using an array for storing dynamically allocated variables or objects, normally the array is first statically declared. (Of course, just like the description the array can also be dynamically declared. But let us use the static version as the example for now.) The following code segment shows how this is done.

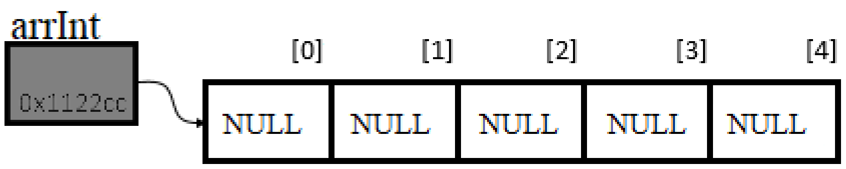
```
const int SIZE = 5;

int * arrInt[SIZE];
```

Please be aware that the array arrInt is prefixed with the "*" and post-fixed with the "[ ]." That "[ ]" indicates that arrInt is an array. And the "*" indicates that the values stored in the array are not int. Instead, the values in the array are pointers pointing to integers.

After the array is initialized with NULL pointers,

```
for (int i=0; i <SIZE; i++)

    arrInt[i] = nullptr;
```

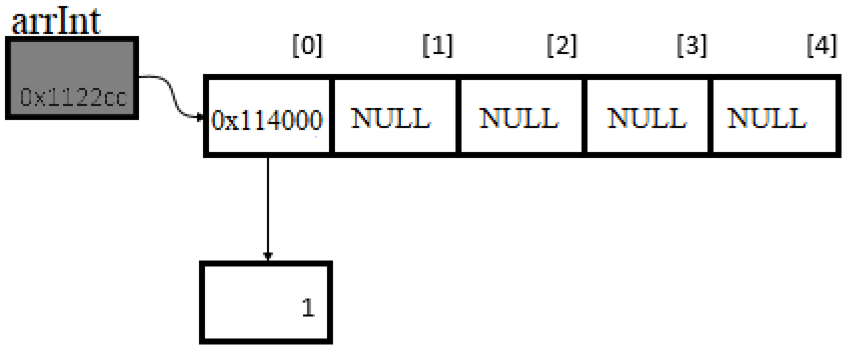
the array looks like the following figure in the memory.



Now we can dynamically create integer variables and link them to each element in the array. Let us see the following example.

```
arrInt[0] = new int;

*arrInt[0] = 1;
```

The first statement creates a new box with the address 0x14000 and assigns the address to the first slot of the array. The second statement then assign an integer 1 to the box. Hence now we have:



We can continue to do this until all the slots are filled with integer pointers. But in the real application, each slot may link to an object, such as student record which consists of first name, last name, address, phone number, and so on.

Finally, the memory blocks allocated in runtime have to be released. The following for statement will release the memory blocks. It checks if the value of every array element is NULL. If it is not, the system knows that it links to an integer box and hence deletes it.

```
for (int i=0; i <SIZE; i++)

        if (arrInt[i] != nullptr)

              delete arrInt[i];
```

Exercise 2

Create a static array which stores integer pointers. Dynamically add five integers, e.g. 11, 22, 33, 44 and 55, to the first five slots. Use a loop statement to display the values and delete all pointers before the program terminates.

A comparison of these two types

The following table summarize these methods are presented in C++ code.

	Type 1	**Type 2**

Declare array	const int SIZE = 5;
	
	int * arrInt1 = nullptr;
	
	arrInt1 = new int[SIZE];
	
	// new the entire array at once
		const int SIZE = 5;
		
		int * arrInt2[SIZE];

Initialize and populate the array	arrInt1[1] = 0;	for (int i=0; i <SIZE; i++)
		
		    arrInt2[i] = nullptr;
		
		arrInt2[1] = new int;
		
		*arrInt2[0] = 1;
		
		// new the array element one at a time

Release memory	delete [] arrInt1;
	
	//The entire array is released at
		for (int i=0; i <SIZE; i++)
		
		    if (arrInt2[i] != nullptr)

	a time	delete arrInt2[i]; //Each element has to be checked first. If it is not NULL, release the memory.

Type 1 is important since it shows that an entire array can be dynamically allocated in runtime.

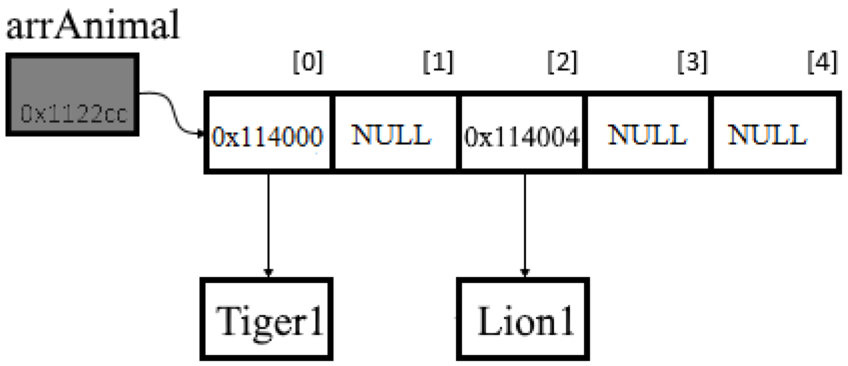
Type 2 is important because of at least the following two reasons:

1	Using array to store pointers can significantly save the memory. Let us do a comparison. We know that objects can be stored in array. For example, the declaration below stores 10,000 Rectangle objects in the array.
	
	`const SIZE  = 10000;`
	`Rectangle arrRect[SIZE];`
	
	If one Rectangle takes 1 kilobytes (1,000 bytes) of memory, then when the array is declared, totally 10 megabytes (10,000,000 bytes) of memory is allocated. What makes it even worse is that most of these objects may not be used at all. One possible alternative is that array of Rectangle pointers, not array objects, is declared.
	
	`const SIZE  = 10000;`
	`Rectangle * arrRect[SIZE];`
	
	With such declaration, only 40,000 bytes (4 * 10000) is needed. (A pointer takes only 4 bytes.) The ratio of memory used between the two is
	
	250 : 1

2	Type 2 is the way which we conduct Polymorphism, one of the most important features in C++. For example, the declaration `const SIZE  = 10000;`

Animal * arrAnimal[SIZE];

allows the Animal pointers to be stored in the array arrAnimal. At a later time, depending on the scenarios, the objects of any **subtypes of Animal**, e.g. Lion, Tiger, and Monkey, can be stored in the array.



This will enable Polymorphism to be conducted. The details will be covered in later chapters.

<u>**Double pointer**</u>

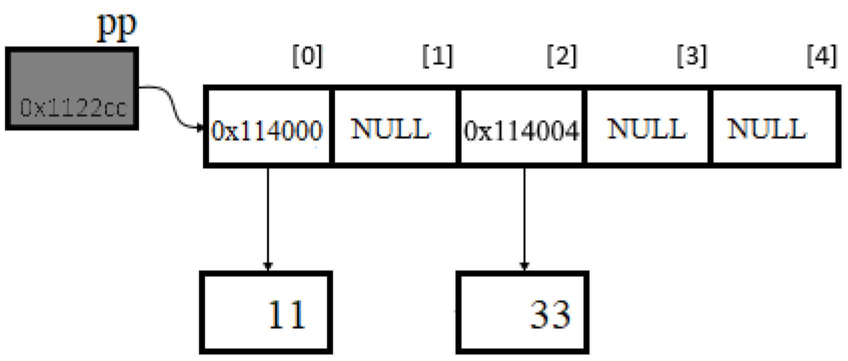
Occasionally we may see a declaration like:

```
int ** pp;
```

in a program. What do the double asterisks stand for? We can first replace one prefixed “*” with a post-fixed “[].”

```
int *pp[];
```

Now we can see it is actually the pointers in Type 2, which we described above. That is, pp is a pointer which points to a static array of integer pointers.



Exercise 3

Create a static array named arrInt, which stores integer pointers. Dynamically add five integers, e.g. 11, 22, 33, 44 and 55, to the first five slots. Declare the pointer pp in the following way and assign arrInt to it:

```
int ** pp = nullptr;

pp = arrInt;
```

Use pp to display the integer values in the array and delete all pointers before the program terminates.

Exercise 4

Write a program which causes more than 1GB bytes of memory leak in your system.

Exercise 5

/*

Goals: (A) Learn how to use array for storing objects

(B) Learn how to use array for storing pointers

(C) Compare the differences of these two methods

Requirements:

Follow the guidelines intertwined with the code to complete the works.

Write comments on the top of the program to answer these questions:

(1) Should we delete the data stored in studArr1? Why?

(2) Should we delete the data stored in studArr2? Why? When we delete the pointers, do we actually release the memory which was allocated for (1) the studArr2 array, (2) the pointer stored in the array, or (3) the student objects pointed by the pointers?

```
*/

#include <iostream>

using namespace std;

struct student

{

   int ID = 0;

   string name = "";

   long long phone = 0;

   string addr = "";

};


int main()

{

   const int SIZE = 10;
```

```
    /// TO DO: Declare a student object array here.



    /// Check the size of the array
    cout << "The size of the studArr1: " << sizeof(studArr1) << endl;

    /// TO DO: Hard-code some data for two students
    /// Populate the 4 fields of studArr1[0] with the data: 12345, Amy, 9256776789,
"123 Main St, Livermore, CA, 94550"
    /// Populate the 4 fields of studArr1[3] with the data: 12666, Bill, 9256776000,
"29 Johnson Dr, Livermore, CA, 94550"

    cout << "Display the data in studArr1" << endl;
    /// TO DO: Use a for loop to display the data of studArr1[0] and studArr1[3]
    /// Use one line to display the information of a person

    /// TO DO: Declare a student pointer array (Type 2 in the document) here. How
can we initialize the pointer with nullptr?



    /// Check the size of the array
    cout << "The size of the studArr2: " << sizeof(studArr2) << endl;
    /// TO DO: Hard-code some data for two students
   /// Populate the 4 fields of studArr2[0] with the data: 12345, Amy, 9256776789,
"123 Main St, Livermore, CA, 94550"
   /// Populate the 4 fields of studArr2[3] with the data: 12666, Bill, 9256776000,
"29 Johnson Dr, Livermore, CA, 94550"


   cout << "Display the data in studArr2" << endl;
   /// TO DO: Use a for loop to display the data of studArr2[0] and studArr2[3]
   /// Use one line to display the information of a person


   /// TO DO: Release data in studArr1, if needed. Answer the #1 question above.


   /// TO DO: Release data in studArr2, if needed. Answer the #2 question above.


   return 0;
}
```