Experiment 4: Write programs to understand dynamic memory allocation and array of objects.

NEW: The **new** operator is used to **allocate memory at runtime**. The memory is allocated in bytes. Let's first see how to allocate a variable dynamically.
int *ptr = new int;

By writing **new int**, we allocated the space in memory required by an integer. Then we assigned the address of that memory to an integer pointer **ptr**. We assign value to that memory as follows:
***ptr = 4;**

We can initialize a variable while dynamical allocation in the following two ways.
int *ptr = new int (4);
Or
int *ptr = new int {4};

Program:

```
#include <iostream>

int main()
{
        int *ptr = new int;
        *ptr = 4;
        std::cout << *ptr << std::endl;
        return 0;
}
```

Output: 4

Explanation: So, we have just seen how to dynamically allocate a variable. Now let's allocate arrays dynamically.

Dynamically Allocating Arrays: The main use of the concept of dynamic memory allocation is for allocating arrays when we have to declare an array by specifying its size but are not sure about the size.

Consider a situation when we want the user to enter the name but are not sure about the number of characters in the name that the user will enter. In that case, we will declare an array of characters for the name with some array size such that the array size should be sufficient enough to hold any name entered. Suppose we declared the array with the array size 30 as follows.
char name[30];

And if the user enters the name having only 12 characters, then the rest of the memory space which was allocated to the array at the time of its declaration would become waste, thus unnecessary consuming the memory.

In this case, we will be using the **new** operator to dynamically allocate the memory at runtime. We use the new operator as follows.
char *arr = new char[length];

Program:

```
#include <iostream>
using namespace std;

int main()
{
        int length, sum = 0;
        cout << "Enter the number of students in the group" << endl;
        cin >> length;
        int *marks = new int[length];
        cout << "Enter the marks of the students" << endl;
        for( int i = 0; i < length; i++ )          // entering marks of students
        {
                cin >> *(marks+i);
        }
        for( int i = 0; i < length; i++ )          // calculating sum
        {
                sum += *(marks+i);
        }
        cout << "sum is " << sum << endl;
        return 0;
}
```

Output:

```
Enter the number of students in the group
4
Enter the marks of the students
50
45
42
78
sum is 215
```

Explanation: In this example, we are calculating the sum of the marks of all the students of a group. Since different groups have a different number of students, therefore we are asking the number of students (i.e.the size of the array) every time we are running the program. In this example, the user entered the size as 4.

int *marks = new int[length]; - We declared an array of integer and allocated it some space in memory dynamically equal to the size which would be occupied by **length** number of **integers**. Thus it is allocated a space equal to **'length * (size of 1 integer)'** and assigned the address of the assigned memory to the pointer **marks**. The rest of the steps must be clear to you.

We call this array **dynamic** because it is being assigned memory when the program runs. We made this possible by using the new operator. This dynamic array is being allocated memory from heap unlike other fixed arrays which are provided memory from stack. We can give any size to these dynamic arrays and there is no limitation to it.
Now what if the variable to which we dynamically allocated some memory is not required anymore?
In that case, we use the **delete** operator.

Delete: Suppose we allocated some memory to a variable dynamically and then we realize that the variable is not needed anymore in the program. In that case, we need to free the memory which we had assigned to that variable. For that, we use the **delete** operator.

It is advised to free the dynamically allocated memory after the program finishes so that it becomes available for future use.

To delete the memory assigned to a variable, we simply need to write the following code.
delete ptr;

Here ptr is the pointer to the dynamically allocated variable.

The delete operator simply returns the memory allocated back to the operating system so that it can be used again.

Program:

```
#include <iostream>

int main()
{
        int *ptr = new int;
        *ptr = 4;
        std::cout << *ptr << std::endl;
        delete ptr;
        return 0;
}
```

Output: 4

Explanation: After printing the value 4, we deleted the pointer i.e. deleted the address of the allocated memory thus freeing it.
Once deleted, a pointer will point to deallocated memory and will be called a **dangling pointer**.
If we further try to delete a dangling pointer, we will get some undefined behavior.

Deleting Array: To delete an array which has been allocated in this way, we write the following code.

delete[] ptr;

Here ptr is a pointer to an array which has been dynamically allocated.

```
#include <iostream>

using namespace std;

int main()
{
        int length, sum = 0;
        cout << "Enter the number of students in the group" << endl;
        cin >> length;
        int *marks = new int[length];
        cout << "Enter the marks of the students" << endl;
        for( int i = 0; i < length; i++ )          // entering marks of students
        {
                cin >> *(marks+i);
        }
        for( int i = 0; i < length; i++ )          // calculating sum
        {
                sum += *(marks+i);
        }
        cout << "sum is " << sum << endl;
        delete[] marks;
        return 0;
}
```

Output:

```
Enter the number of students in the group
4
Enter the marks of the students
5
2
8
5
sum is 20
```

Explanation: We just wrote **delete[ ] marks;** at the end of the program to release the memory which was dynamically allocated using new.

Dynamic Memory Allocation for Objects: We can also dynamically allocate objects.

As we know that **Constructor** is a member function of a class which is called whenever a new object is created of that class. It is used to initialize that object. **Destructor** is also a class member function which is called whenever the object goes out of scope.
Destructor is used to release the memory assigned to the object.

Program:

```
#include <iostream>
using namespace std;

class Box {
  public:
    Box() {
      cout << "Constructor called!" <<endl;
    }
    ~Box() {
      cout << "Destructor called!" <<endl;
    }
};
int main() {
  Box* myBoxArray = new Box[4];
  delete [] myBoxArray; // Delete array

  return 0;
}
```

Output:

Constructor called!
Constructor called!
Constructor called!
Constructor called!
Destructor called!
Destructor called!
Destructor called!
Destructor called!

Explanation: The Constructor will be called four times since we are allocating memory to four objects of the class 'Box'. The Destructor will also be called four times during each of these objects.