Array of Objects:

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

ClassName ObjectName[number of objects];

Different methods to initialize the Array of objects with parameterized constructors:

1. **Using malloc():** To avoid the call of non-parameterised constructor, use malloc() method. “malloc” or “memory allocation” method in C++ is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form.

   Example Code:

```
#include <iostream>
#define N 5

using namespace std;

class Test {
   // private variables
   int x, y;

public:
   // parameterised constructor
   Test(int x, int y)
   {
      this->x = x;
      this->y = y;
   }
```

```
    // function to print
    void print()
    {
        cout << x << " " << y << endl;
    }
};

int main()
{
    // allocating dynamic array
    // of Size N using malloc()
    Test* arr = (Test*)malloc(sizeof(Test) * N);

    // calling constructor
    // for each index of array
    for (int i = 0; i < N; i++) {
        arr[i] = Test(i, i + 1);
    }

    // printing contents of array
    for (int i = 0; i < N; i++) {
        arr[i].print();
    }

    return 0;
}
```

Output:

```
0 1
1 2
2 3
3 4
4 5
```

2. **Using new keyword:** The new operator denotes a request for memory allocation on the Heap. If sufficient memory is available, the new operator initializes the memory and returns the address of the newly allocated and

initialized memory to the pointer variable. Here, pointer-variable is the pointer of type data-type. Data-type could be any built-in data type including array or any user-defined data types including structure and class.
For dynamic initialization new keyword require non parameterised constructor if we add a parameterised constructor. So we will use a dummy constructor for it.

Example Code:

```
#include <iostream>
#define N 5

using namespace std;

class Test {
   // private variables
   int x, y;

public:
   // dummy constructor
   Test() {}

   // parameterised constructor

   Test(int x, int y)
   {
      this->x = x;
      this->y = y;
   }

   // function to print
   void print()
   {
      cout << x << " " << y << endl;
   }
};

int main()
```

```
{
    // allocating dynamic array
    // of Size N using new keyword
    Test* arr = new Test[N];

    // calling constructor
    // for each index of array
    for (int i = 0; i < N; i++) {
        arr[i] = Test(i, i + 1);
    }

    // printing contents of array
    for (int i = 0; i < N; i++) {
        arr[i].print();
    }

    return 0;
}
```

Output:

0 1
1 2
2 3
3 4
4 5

If we don't use the dummy constructor compiler would show the error given below

Compiler Error:

error: no matching function for call to 'Test::Test()'

```
Test *arr=new Test[N];
```

3. **Using Double pointer (pointer to pointer concept):** A pointer to a pointer is a form of multiple indirections, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.

Here we can assign a number of blocks to be allocated and thus for every index we have to call parameterised constructor using the new keyword to initialise.

Example Code:

```
#include <iostream>
#define N 5

using namespace std;

class Test {
   // private variables
   int x, y;

public:
   // parameterised constructor

   Test(int x, int y)
      : x(x), y(y)
   {
   }

   // function to print
   void print()
   {
      cout << x << " " << y << endl;
   }
};

int main()
{
   // allocating array using
   // pointer to pointer concept
   Test** arr = new Test*[N];

   // calling constructor for each index
   // of array using new keyword
```

```
    for (int i = 0; i < N; i++) {
        arr[i] = new Test(i, i + 1);
    }

    // printing contents of array
    for (int i = 0; i < N; i++) {
        arr[i]->print();
    }

    return 0;
}
```

Output:

```
0 1
1 2
2 3
3 4
4 5
```

4. **Using Vector of type class:** Vector is one of the most powerful element of Standard Template Library makes it easy to write any complex codes related to static or dynamic array in an efficient way. It takes one parameter that can be of any type and thus we use our Class as a type of vector and push Objects in every iteration of the loop.
   Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end.

   Example Code:

```
#include <iostream>
#include <vector>
#define N 5
using namespace std;
class Test {
    // private variables
    int x, y;
```

```
public:
    // parameterised constructor

    Test(int x, int y)
        : x(x), y(y)
    {
    }

    // function to print
    void print()
    {
        cout << x << " " << y << endl;
    }
};

int main()
{
    // vector of type Test class
    vector<Test> v;

    // inserting object at the end of vector
    for (int i = 0; i < N; i++)
        v.push_back(Test(i, i + 1));

    // printing object content
    for (int i = 0; i < N; i++)
        v[i].print();

    return 0;
}
```

Output:

```
0 1
1 2
2 3
3 4
4 5
```