INTRODUCTION OF C++ SECTION 5 PART(2)

Dr/Ghada Maher
Eng/ Hossam Medhat

C++ DYNAMIC ALLOCATION OF ARRAYS

❖ **What is a Dynamic Array?**

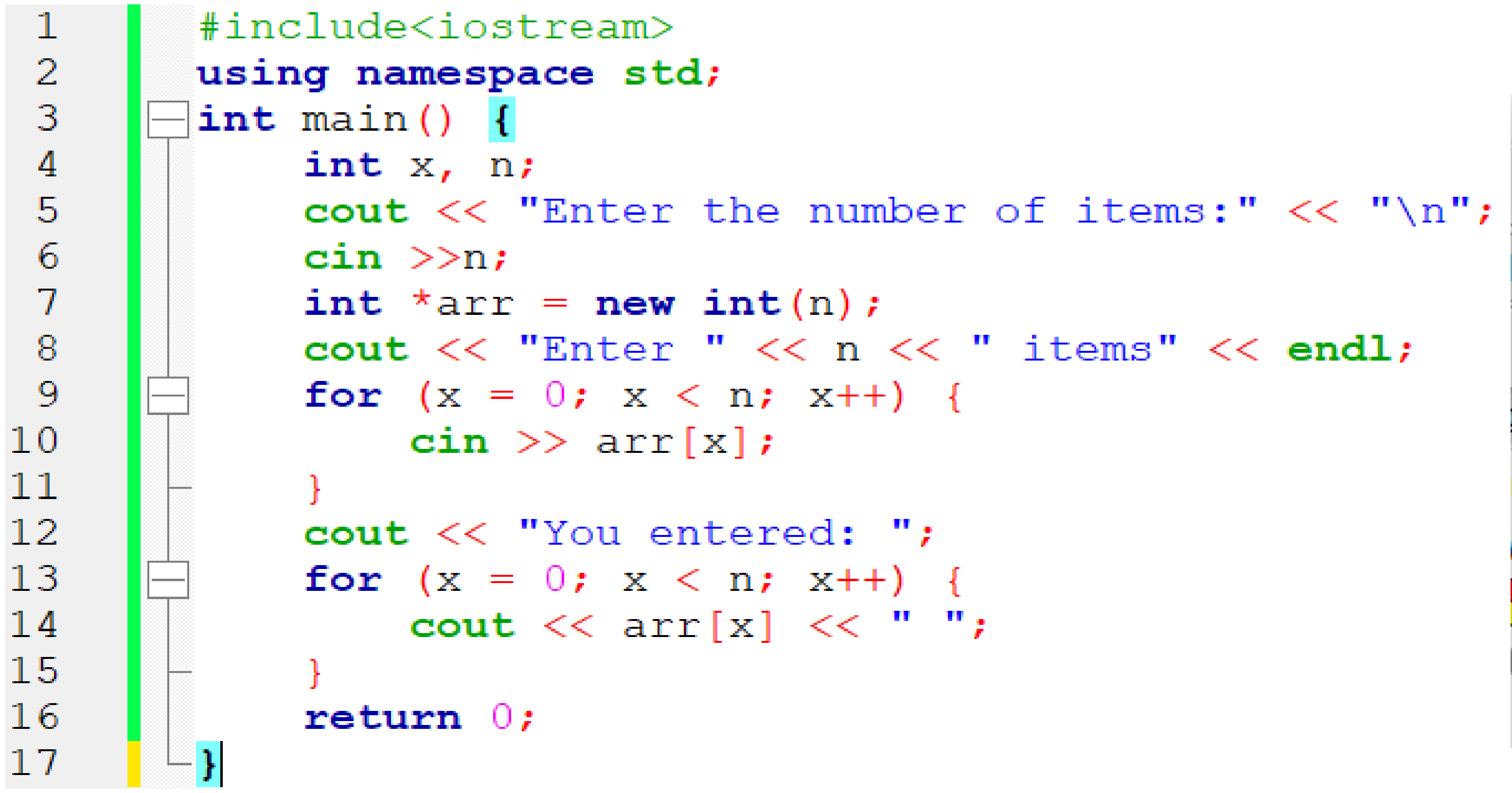
- **A dynamic array** is quite similar to a regular array, **but its size is modifiable during program runtime.**
- **DynamArray elements** occupy a contiguous block of memory.
- Once an array has been created, **its size cannot be changed**. However**, a dynamic array is different. A dynamic array can expand its size even after it has been filled.**
- In C++, we can **create a dynamic array using the new keyword.** The number of items to be allocated is specified within a pair of square brackets.
- **Syntax:**

```
pointer_variable = new data_type;
```

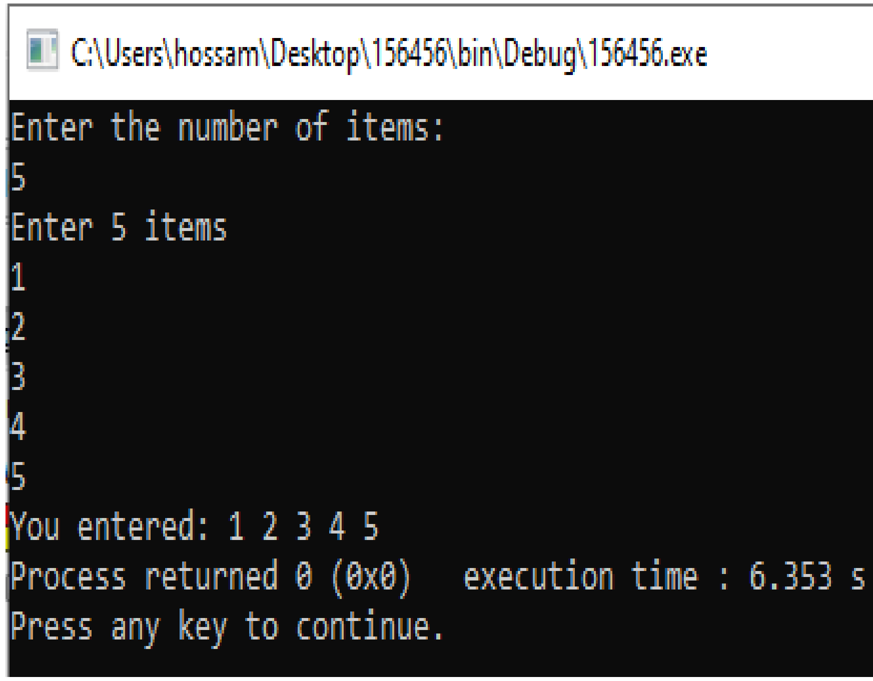
- **The pointer_variable** is the name of the pointer variable.
- **The data_type** must be a valid C++ data type.

C++ DYNAMIC ALLOCATION OF ARRAYS

❖**Example_1 :**



```cpp
#include<iostream>
using namespace std;
int main() {
    int x, n;
    cout << "Enter the number of items:" << "\n";
    cin >>n;
    int *arr = new int(n);
    cout << "Enter " << n << " items" << endl;
    for (x = 0; x < n; x++) {
        cin >> arr[x];
    }
    cout << "You entered: ";
    for (x = 0; x < n; x++) {
        cout << arr[x] << " ";
    }
    return 0;
}
```



C:\Users\hossam\Desktop\156456\bin\Debug\156456.exe

```
Enter the number of items:
5
Enter 5 items
1
2
3
4
5
You entered: 1 2 3 4 5
Process returned 0 (0x0)   execution time : 6.353 s
Press any key to continue.
```

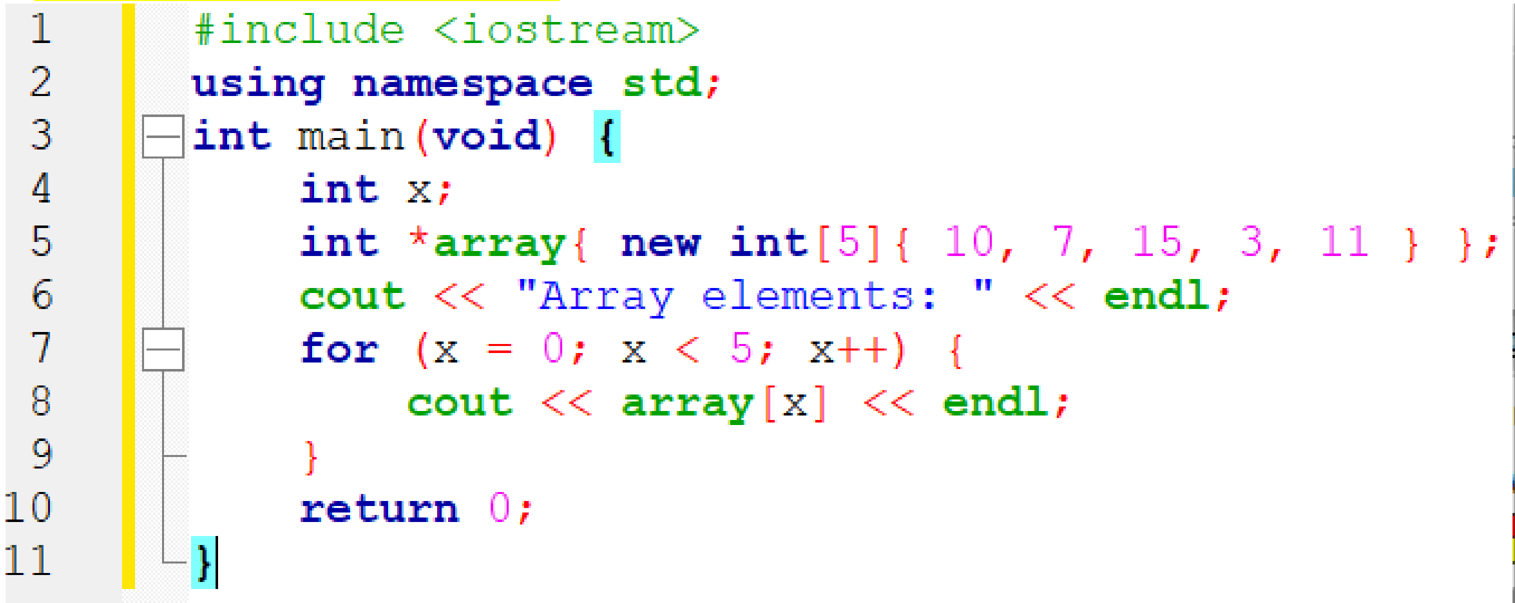
➢**NOTE:** **In the above example, the user is allowed to specify any size for the array during run time**

C++ Initializing Dynamically Allocated Arrays

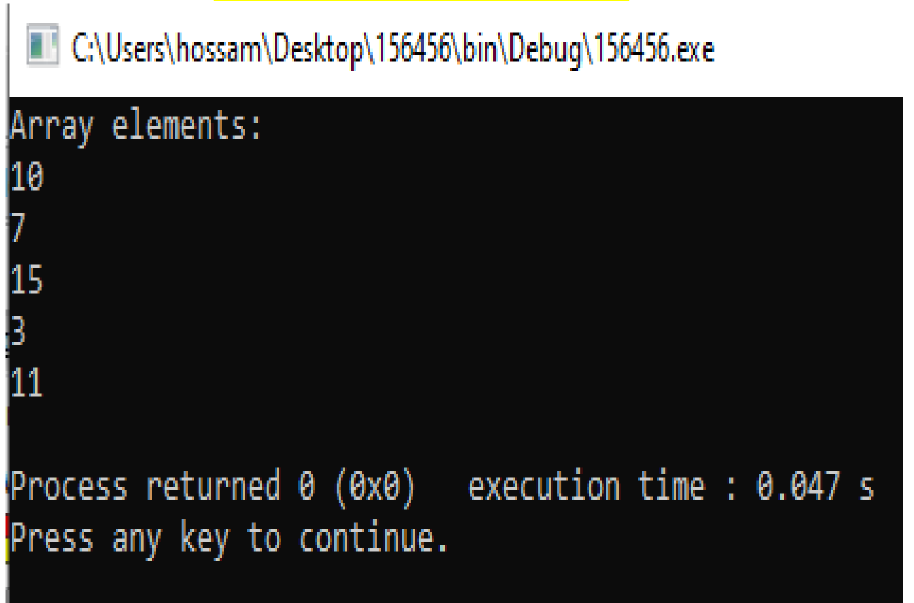
- ❖ It's easy to **initialize a dynamic array to 0.**
- ➢ **Syntax:**

```
int *array{ new int[length]{} };
```

- ➢ In the above syntax, **the length denotes the number of elements to be added to the array.** Since we need to initialize the array to 0, **this should be**



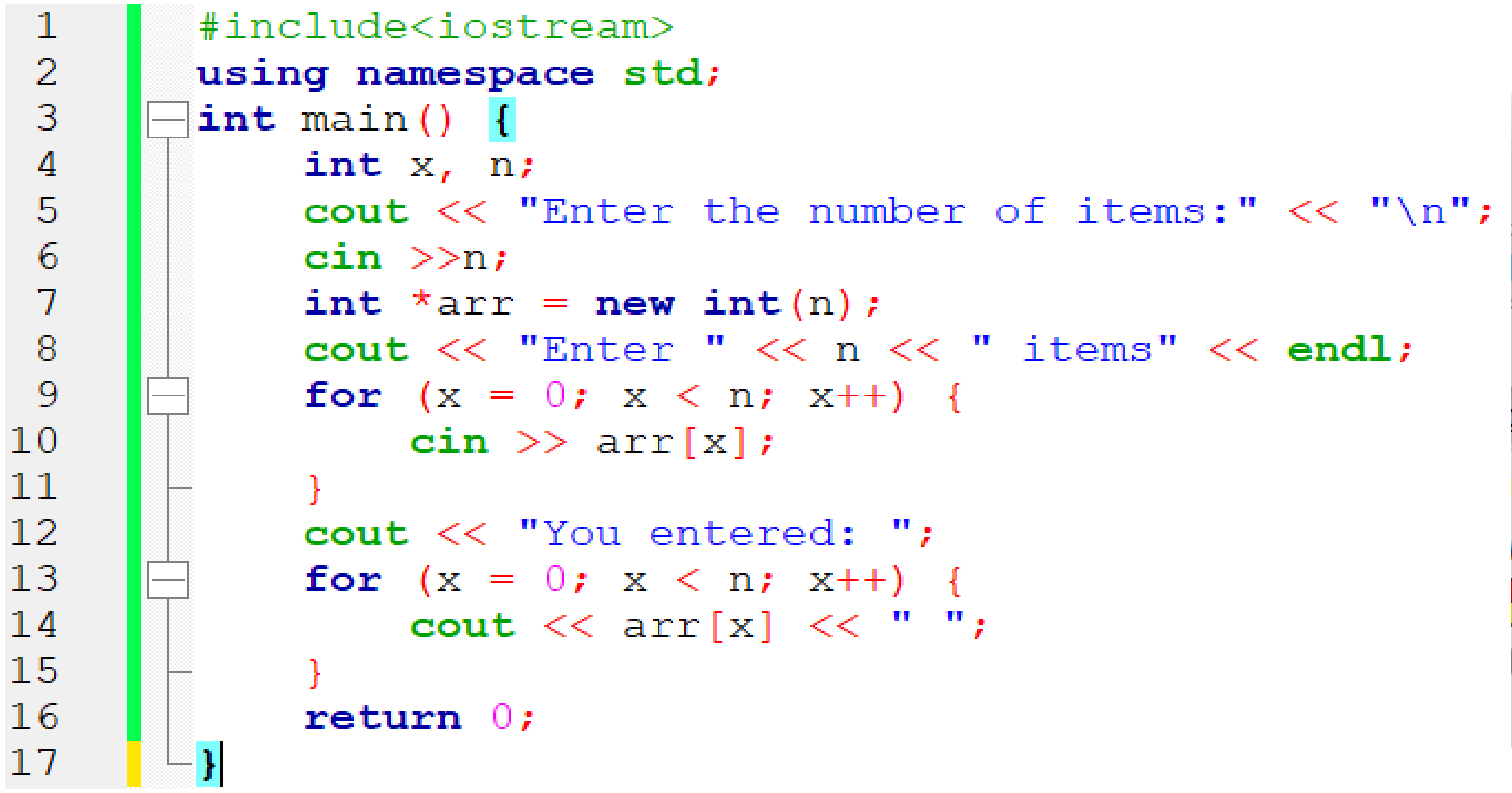
```
#include <iostream>
using namespace std;
int main(void) {
    int x;
    int *array{ new int[5]{ 10, 7, 15, 3, 11 } };
    cout << "Array elements: " << endl;
    for (x = 0; x < 5; x++) {
        cout << array[x] << endl;
    }
    return 0;
}
```



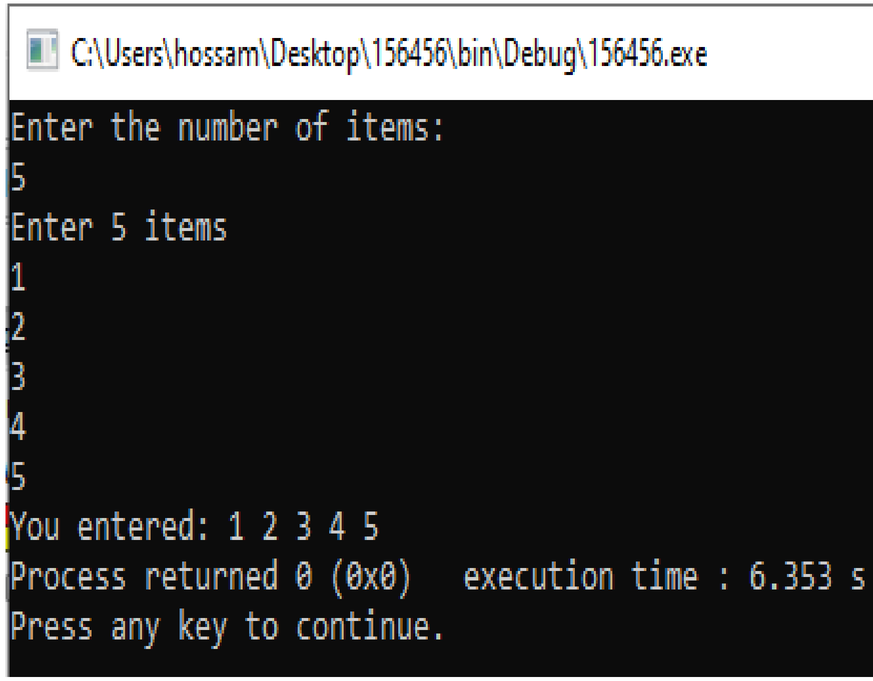
```
C:\Users\hossam\Desktop\156456\bin\Debug\156456.exe
Array elements:
10
7
15
3
11

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

C++ DYNAMIC ALLOCATION OF ARRAYS

❖ **Example_1 :**



```cpp
#include<iostream>
using namespace std;
int main() {
     int x, n;
     cout << "Enter the number of items:" << "\n";
     cin >>n;
     int *arr = new int(n);
     cout << "Enter " << n << " items" << endl;
     for (x = 0; x < n; x++) {
         cin >> arr[x];
     }
     cout << "You entered: ";
     for (x = 0; x < n; x++) {
         cout << arr[x] << " ";
     }
     return 0;
}
```



```
C:\Users\hossam\Desktop\156456\bin\Debug\156456.exe
Enter the number of items:
5
Enter 5 items
1
2
3
4
5
You entered: 1 2 3 4 5
Process returned 0 (0x0)   execution time : 6.353 s
Press any key to continue.
```

➢ **NOTE: In the above example, the user is allowed to specify any size for the array during run time**

C++ Dynamically Deleting Arrays

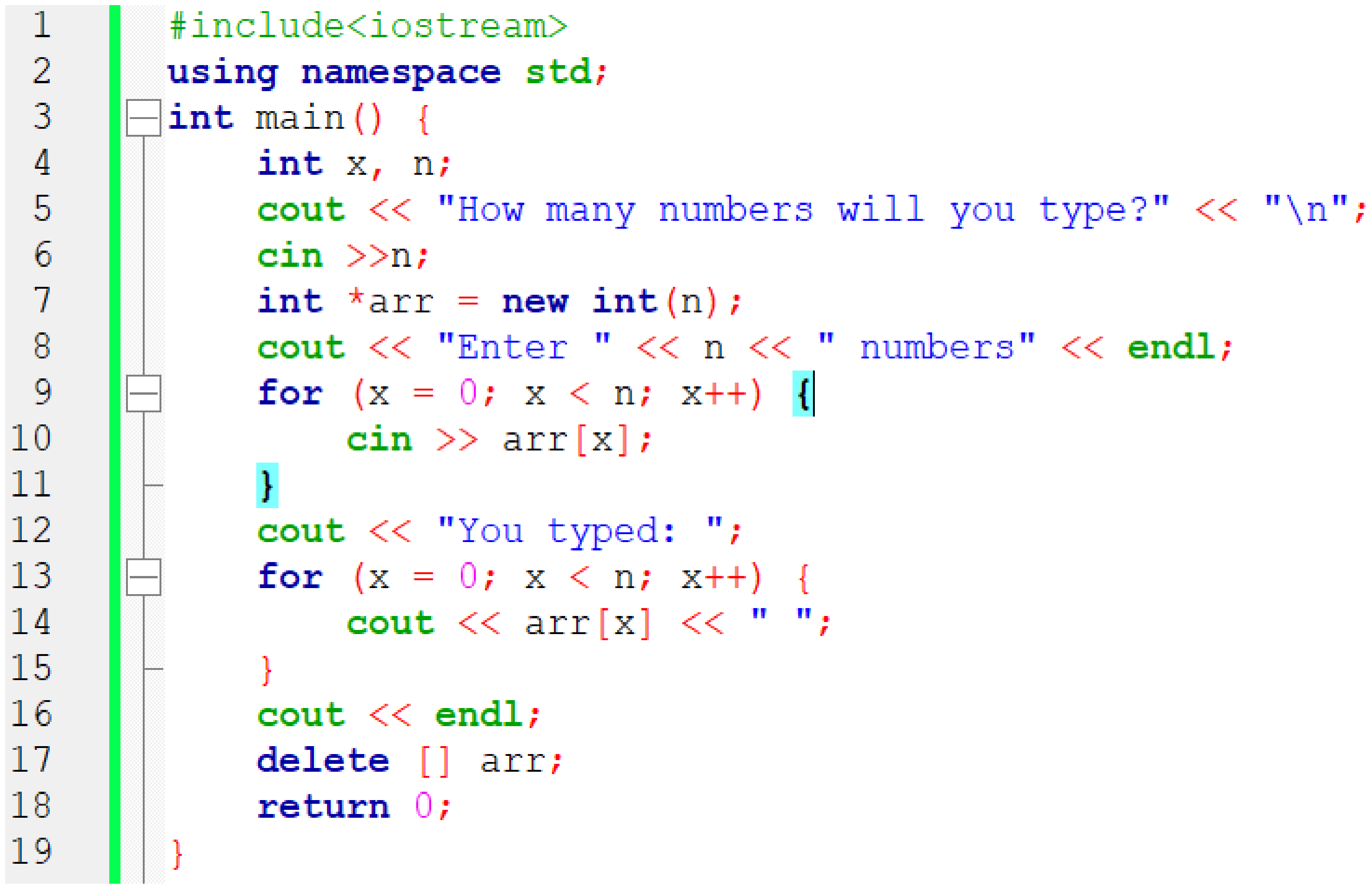
- A dynamic array should be deleted from the computer memory once its purpose is fulfilled.

- The delete statement can help you accomplish this. The released memory space can then
 be used to hold another set of data. However, even if you do not delete the dynamic array from the computer memory, it will be deleted automatically once the program terminates.

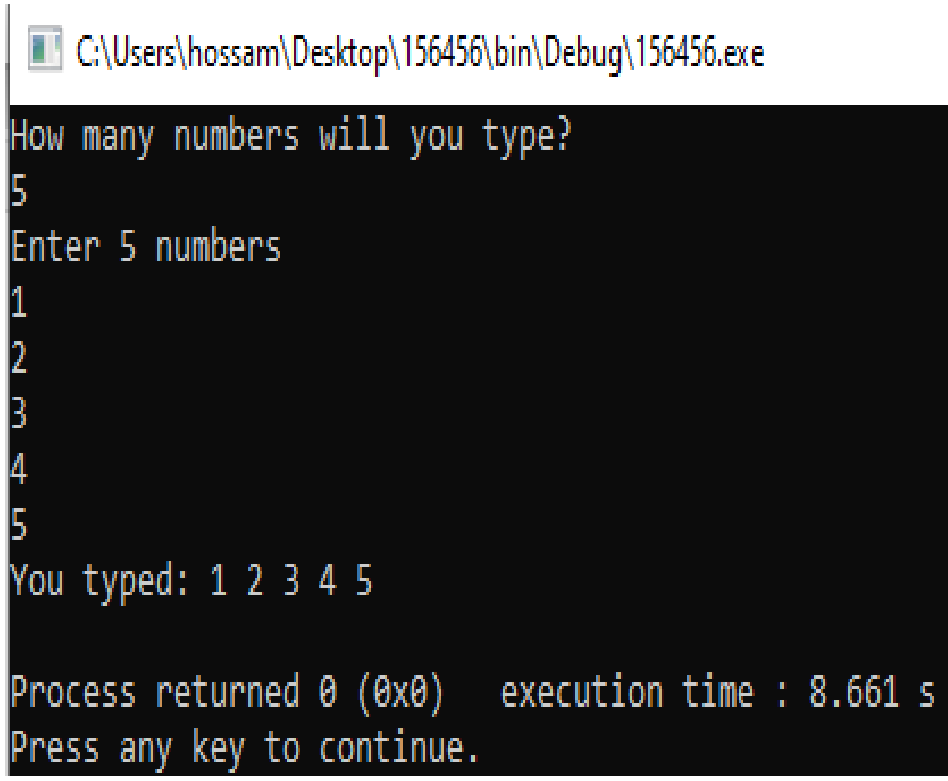
- Note: To delete a dynamic array from the computer memory, you should use delete[ ], instead of delete.
- The [ ] instructs the CPU to delete multiple variables rather than one variable.

C++ DYNAMICALLY DELETING ARRAYS “CONT”

❖ **Example_3 :**



```cpp
#include<iostream>
using namespace std;
int main() {
    int x, n;
    cout << "How many numbers will you type?" << "\n";
    cin >>n;
    int *arr = new int(n);
    cout << "Enter " << n << " numbers" << endl;
    for (x = 0; x < n; x++) {
        cin >> arr[x];
    }
    cout << "You typed: ";
    for (x = 0; x < n; x++) {
        cout << arr[x] << " ";
    }
    cout << endl;
    delete [] arr;
    return 0;
}
```



C:\Users\hossam\Desktop\156456\bin\Debug\156456.exe

```
How many numbers will you type?
5
Enter 5 numbers
1
2
3
4
5
You typed: 1 2 3 4 5

Process returned 0 (0x0)   execution time : 8.661 s
Press any key to continue.
```

C++ STRUCTURES (STRUCT)

- Structures (also called structs) are a way to group several related variables into one place.
- Unlike an array, a structure can contain many different data types (int, string, bool, etc.).
- Create a Structure :
- To create a structure, use the struct keyword and declare each of its members inside curly braces. After the declaration, specify the name of the structure variable

- Example

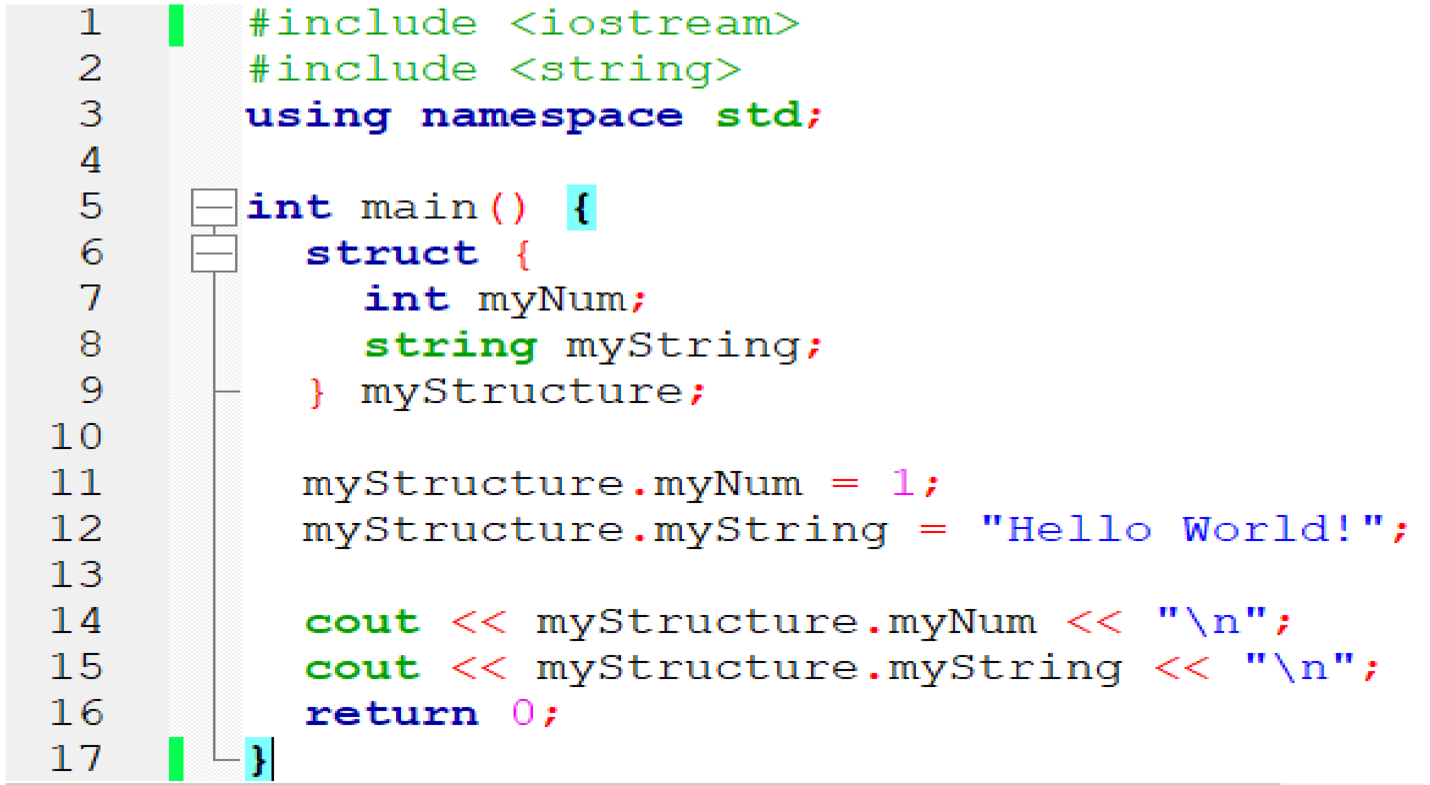
```
struct {                    // Structure declaration
  int myNum;              // Member (int variable)
  string myString;   // Member (string variable)
} myStructure;         // Structure variable
```

C++ STRUCTURES (STRUCT)

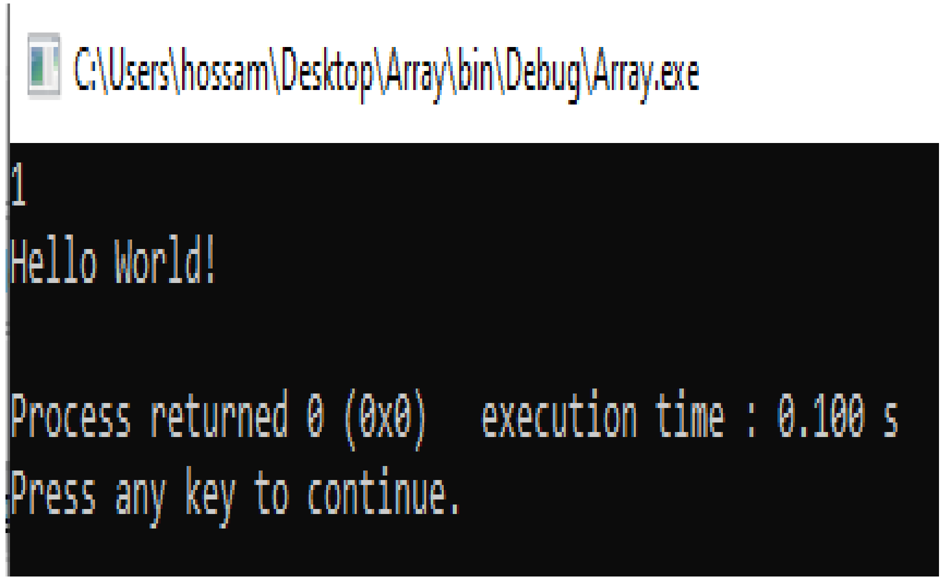
❖**Access Structure Members :**

➢**To access members of a structure, use the dot syntax (.).**

➢**Example :**



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

int main() {
  struct {
    int myNum;
    string myString;
  } myStructure;

  myStructure.myNum = 1;
  myStructure.myString = "Hello World!";

  cout << myStructure.myNum << "\n";
  cout << myStructure.myString << "\n";
  return 0;
}
```



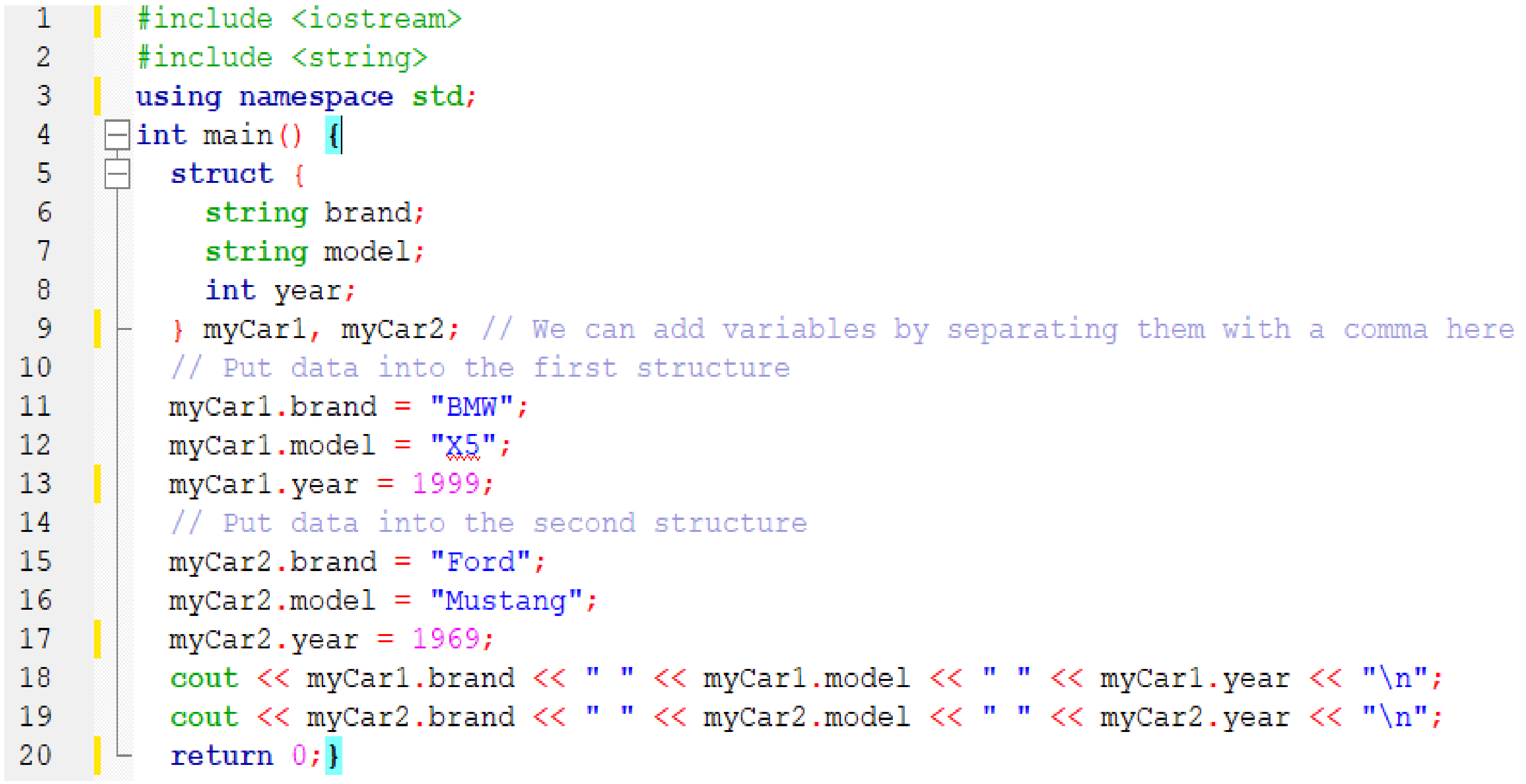
```
C:\Users\hossam\Desktop\Array\bin\Debug\Array.exe
1
Hello World!

Process returned 0 (0x0)   execution time : 0.100 s
Press any key to continue.
```

C++ STRUCTURES (STRUCT)

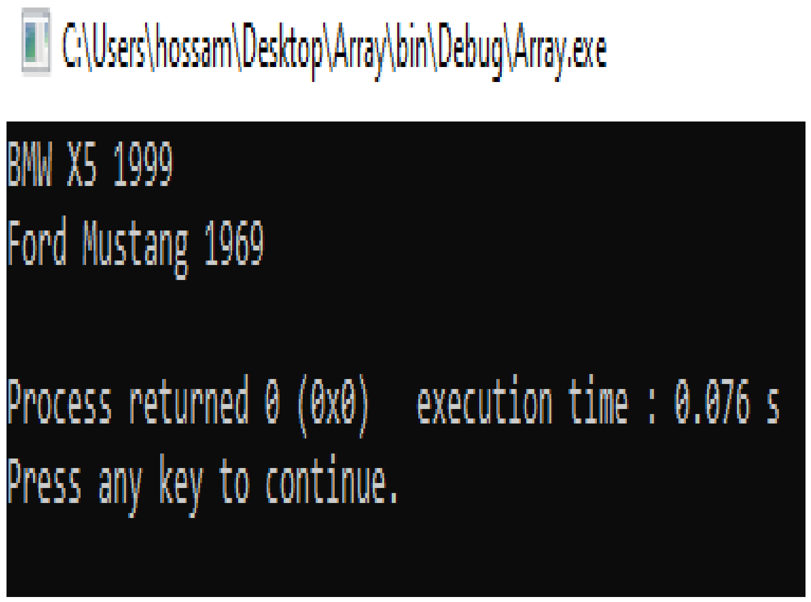
❖ **One Structure in Multiple Variables :**

➢ **You can use a comma (,) to use one structure in many variables.**

➢ **Example : shows how to use a structure in two different variables.**



```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
  struct {
    string brand;
    string model;
    int year;
  } myCar1, myCar2; // We can add variables by separating them with a comma here
  // Put data into the first structure
  myCar1.brand = "BMW";
  myCar1.model = "X5";
  myCar1.year = 1999;
  // Put data into the second structure
  myCar2.brand = "Ford";
  myCar2.model = "Mustang";
  myCar2.year = 1969;
  cout << myCar1.brand << " " << myCar1.model << " " << myCar1.year << "\n";
  cout << myCar2.brand << " " << myCar2.model << " " << myCar2.year << "\n";
  return 0;}
```



```
C:\Users\hossam\Desktop\Array\bin\Debug\Array.exe

BMW X5 1999
Ford Mustang 1969

Process returned 0 (0x0)   execution time : 0.076 s
Press any key to continue.
```

C++ Structures (Struct)

❖ **Named Structures :**

➢ To create a named structure, put **the name of the structure right after the struct keyword**.

➢ **Example :**

```
struct myDataType {           // This structure is named
"myDataType"
  int myNum;
  string myString;
};
```

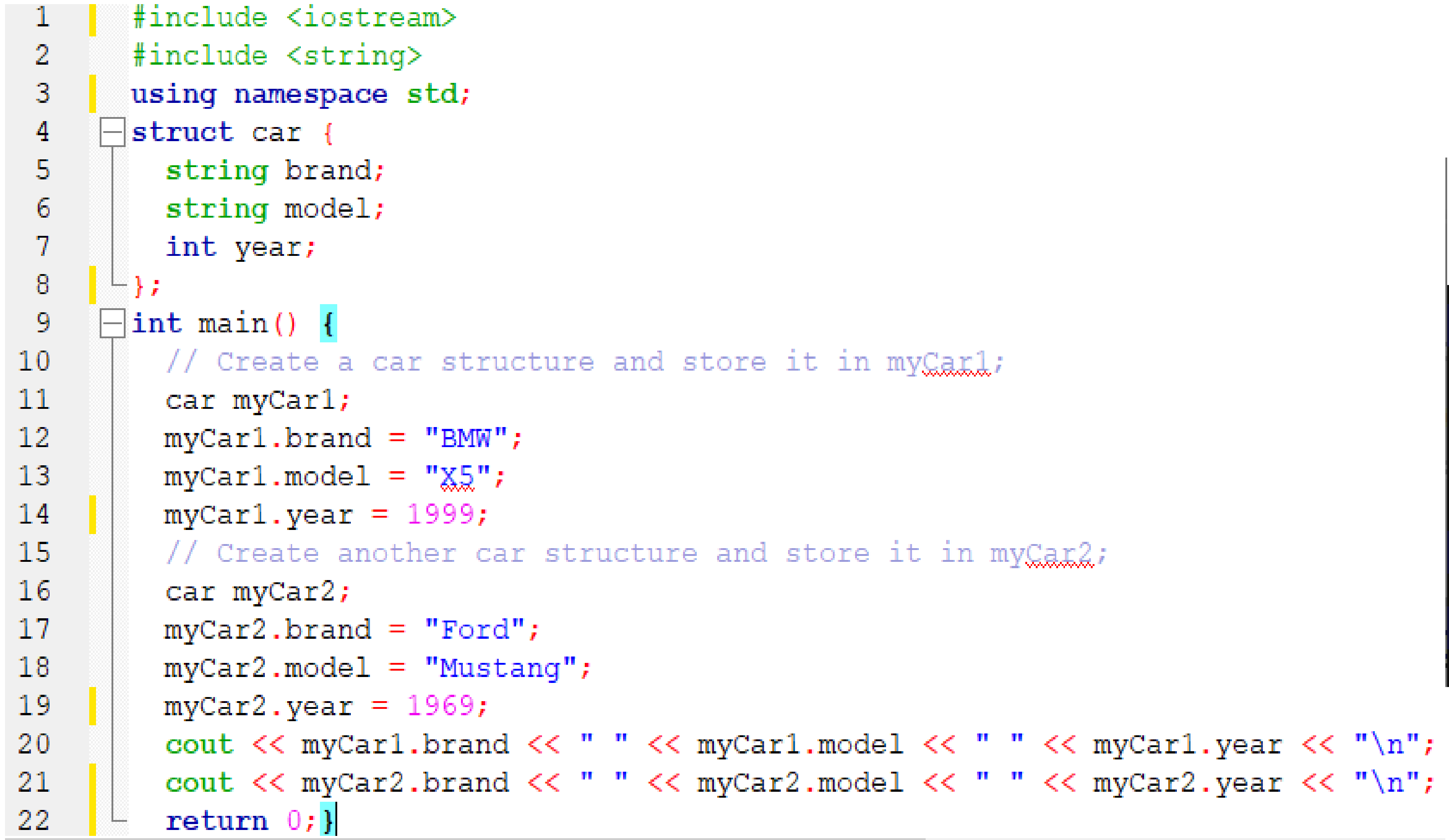
➢ use the name of the structure **as the data type** of the variable.

```
myDataType  myVar;
```

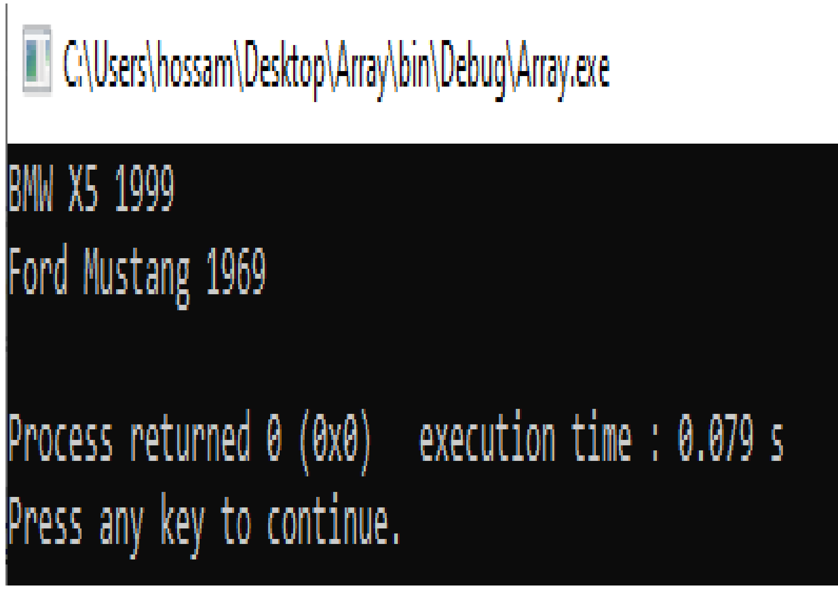
C++ STRUCTURES (STRUCT)

❖ **Named Structures :**

➢ **Example : Use one structure to represent two cars:**



```cpp
#include <iostream>
#include <string>
using namespace std;
struct car {
  string brand;
  string model;
  int year;
};
int main() {
  // Create a car structure and store it in myCar1;
  car myCar1;
  myCar1.brand = "BMW";
  myCar1.model = "X5";
  myCar1.year = 1999;
  // Create another car structure and store it in myCar2;
  car myCar2;
  myCar2.brand = "Ford";
  myCar2.model = "Mustang";
  myCar2.year = 1969;
  cout << myCar1.brand << " " << myCar1.model << " " << myCar1.year << "\n";
  cout << myCar2.brand << " " << myCar2.model << " " << myCar2.year << "\n";
  return 0;}
```



```
C:\Users\hossam\Desktop\Array\bin\Debug\Array.exe

BMW X5 1999
Ford Mustang 1969

Process returned 0 (0x0)   execution time : 0.079 s
Press any key to continue.
```

THANKS

Dr/Ghada Maher
Eng/ Hossam Medhat