* ***Purpose* :** **Classwork.**

* ***Date* :** **29/12/2025**

* ***Author* :** **Vikas Srivastava**

* ***ID* :** **55984**

* ***Batch ID* :** **25SUB4505**

---

1. ***Code :*** Write a program to implement a custom MyArray class in C++ that demonstrates abstraction, constructors (default, parameterized, copy constructor), and member functions to initialize and display array elements, showcasing object-oriented programming principles.

```
Welcome ×    abstraction.cpp ×
abstraction.cpp > main()
#include <iostream>
using namespace std;
class MyArray{ //abstraction
            int arr[100];
            int size;
public:
            MyArray();
            MyArray(int data);
            MyArray(const MyArray&);
            void printArray();
            MyArray& extend(MyArray&);
};
int main(){
            MyArray objOne;
            MyArray objTwo(100);
            MyArray objThree = objTwo;
            objOne.printArray();
            objTwo.printArray();
            objThree.printArray();
            //objFour = objTwo.extend(objThree);
            //objFour.printArray();
}
MyArray::MyArray(){
            size = 100;
            for (int cnt=0;cnt< size; cnt++)
                        arr[cnt] = 0;
}
MyArray::MyArray(int data){
            size = 100;
            for (int cnt=0;cnt< size; cnt++)
                        arr[cnt] = data + cnt;
}
```

```
Welcome    abstraction.cpp ×
abstraction.cpp > main()
int main(){
            //objFour = objTwo.extend(objThree);
}
MyArray::MyArray(){
            size = 100;
            for (int cnt=0;cnt< size; cnt++)
                        arr[cnt] = 0;
}
MyArray::MyArray(int data){
            size = 100;
            for (int cnt=0;cnt< size; cnt++)
                        arr[cnt] = data + cnt;
}
MyArray::MyArray(const MyArray& rhsObj){
            size = rhsObj.size;
            for (int cnt=0;cnt< size; cnt++)
                        arr[cnt] = rhsObj.arr[cnt];
}
void MyArray::printArray(){
            cout<<"Size: "<<size<<"\tarray: "<<endl;
            for(int cnt=0;cnt < size; cnt++)
                        cout<<arr[cnt]<<"  ";
            cout<<endl;
}
MyArray& MyArray::extend(MyArray& rhsObject){
            //For future implementation with dynamic memory allocation
            return rhsObject;
}
```

Output :

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ abstraction.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Size: 100       array:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0
Size: 100       array:
100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125
126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151
152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177
178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199
Size: 100       array:
100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125
126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151
152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177
178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork>
```

2. ***Code*** : Write a program to demonstrate the invocation order of constructors and destructors in C++, including the default constructor, parameterized constructor, and copy constructor, and observe how the destructor is called automatically when objects go out of scope.

copyCtorOne.cpp

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

class Test{
  public:
    Test(){cout<<"Test()--> Default Ctor"<<endl; }
    Test(int){cout<<"Test(int)--> Parameterized Ctor"<<endl; }

    ~Test(){cout<<"~Test()--> Destructor"<<endl; }
    Test(const Test &) {cout<<"Test(const Test&) ---> Copy Ctor"<<endl;}
};


int main(){
  Test objOne ;//single parameterized ctor
  Test objTwo = 100;//single parameterized ctor
  Test objThree = objTwo;//single parameterized ctor
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\copyCtorOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Test()--> Default Ctor
Test(int)--> Parameterized Ctor
Test(const Test&) ---> Copy Ctor
~Test()--> Destructor
~Test()--> Destructor
~Test()--> Destructor
```

3. ***Code :*** Write a program to demonstrate passing an object by reference to a function in C++, showing that no copy constructor is called when objects are passed by reference, and only the existing object is accessed and returned.

copyCtorThree.cpp

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

class Test{
  public:
    Test(){cout<<"Test()--> Default Ctor"<<endl; }
    Test(int){cout<<"Test(int)--> Parameterized Ctor"<<endl; }

    ~Test(){cout<<"~Test()--> Destructor"<<endl; }
    Test(const Test &) {cout<<"Test(const Test&) ---> Copy Ctor"<<endl;}
};

Test& fun(Test& arg){ //call by reference

  return arg;
}

int main(){
  Test objTwo = 100;//single parameterized ctor

  fun(objTwo);
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\copyCtorThree.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Test(int)--> Parameterized Ctor
~Test()--> Destructor
```

4. ***Code :*** Write a program to demonstrate passing an object by value to a function in C++, where a copy of the object is created, resulting in the copy constructor being invoked, and the copied object is destroyed when it goes out of scope.

copyCtorTwo.cpp

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

class Test{
  public:
    Test(){cout<<"Test()--> Default Ctor"<<endl; }
    Test(int){cout<<"Test(int)--> Parameterized Ctor"<<endl; }

    ~Test(){cout<<"~Test()--> Destructor"<<endl; }
    Test(const Test &) {cout<<"Test(const Test&) ---> Copy Ctor"<<endl;}
};

Test fun(Test arg){//call by value

  return arg;
}

int main(){
  Test objTwo = 100;//single parameterized ctor

  fun(objTwo);
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\copyCtorTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Test(int)--> Parameterized Ctor
Test(const Test&) ---> Copy Ctor
Test(const Test&) ---> Copy Ctor
~Test()--> Destructor
~Test()--> Destructor
~Test()--> Destructor
```

5. ***Code :*** Write a program to demonstrate deep copy in C++ using a copy constructor, where dynamically allocated memory is duplicated for each object so that both objects manage separate memory, preventing issues like double deletion and dangling pointers.

DeepCopy.cpp

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

class Resource {
public:
    char* ptr;

    // Constructor
    Resource(const char* str) {
        ptr = new char[strlen(str) + 1];
        strcpy(ptr, str);
        cout << "Memory allocated: " << ptr << endl;
    }

    // Deep Copy Constructor
    Resource(const Resource& other) {
        ptr = new char[strlen(other.ptr) + 1];
        strcpy(ptr, other.ptr);
        cout << "Deep copy created: " << ptr << endl;
    }

    // Destructor
    ~Resource() {
        cout << "Deleting memory: " << ptr << endl;
        delete[] ptr;
    }
};

int main() {
    Resource r1("Network");
    Resource r2 = r1;   // Deep copy now

    return 0;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\DeepCopy.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Memory allocated: Network
Deep copy created: Network
Deleting memory: Network
Deleting memory: Network
```

6. ***Code :*** Write a program to demonstrate the concept of a friend class in C++, where one class (UsingTest) is declared as a friend of another class (Test), allowing it to access the private data members and member functions of the class, illustrating controlled access and encapsulation.

friendClassOne.cpp

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

//class UsingTest;

class Test{//all members of a class is private by default
  int data ;
  Test(int x=10): data(x){ cout <<"constructor called"<<endl;}
  void disp(){cout<<"Data: "<<data<<endl; }
  friend class UsingTest;
};

class UsingTest{
  Test obj;
public:
  UsingTest(int x=100): obj(Test(x)) {}
  void print(){ obj.disp() ; }
};

int main(){
  UsingTest obj=100;
  obj.print();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\friendClassOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
constructor called
Data: 100
```

7. Code : Write a program to demonstrate the use of a friend function in C++, where the main() function is declared as a friend of a class, allowing it to access the class's private constructor and private member function, illustrating controlled access beyond normal encapsulation rules.

friendOne.cpp

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

class Test{//all members of a class is private by default
  int data ;
  Test(int x=10): data(x){ cout <<"constructor called"<<endl;}
  void disp(){cout<<"Data: "<<data<<endl; }

  friend int main();
};

int main(){
  Test obj=100;
  obj.disp();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\friendOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
constructor called
Data: 100
```

8. ***Code :*** Write a program to demonstrate the use of a friend function in C++, where a non-member function is granted access to the private data members of a class to display internal object information.

friendTwo.cpp

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

class Box{
  int length;
public:
  Box(int len = 10): length(len) {}

  friend void showLength(Box &b);
};

void showLength(Box &b){
  cout<<"Length : "<<b.length<<endl;
}

int main(){
  Box gift;
  showLength(gift);
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\friendTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Length : 10
```

9. ***Code :*** Write a program to demonstrate single inheritance in C++, where a derived class inherits public member functions from a base class and accesses them through a derived class object.

inherOne.cpp

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

class Base{
public:
  void funOne(){cout<<"Base::funOne()"<<endl; }
  void funTwo(){cout<<"Base::funTwo()"<<endl; }
};

class Derived:public Base{};

int main(){
  Derived dObj;
  dObj.funOne();
  dObj.funTwo();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\inherOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Base::funOne()
Base::funTwo()
```

10. ***Code :*** Write a program to implement a custom array class (MyArray) in C++ that demonstrates abstraction, default constructor, parameterized constructor, and copy constructor, along with a member function to display array elements, illustrating object creation and copying behavior.

```
myArrayOne.cpp ×
myArrayOne.cpp > ...
#include <iostream>
using namespace std;

class MyArray{ //abstraction
    int arr[100];
    int size;
public:
    MyArray();
    MyArray(int data);
    MyArray(const MyArray&);
    void printArray();
    MyArray& extend(MyArray&);
};


int main(){
    MyArray objOne;
    MyArray objTwo(100);
    MyArray objThree = objTwo;
    objOne.printArray();
    objTwo.printArray();
    objThree.printArray();

    //objFour = objTwo.extend(objThree);
    //objFour.printArray();
}

MyArray::MyArray(){
    size = 100;
    for (int cnt=0;cnt< size; cnt++)
        arr[cnt] = 0;
}
```

```
myArrayOne.cpp ×
myArrayOne.cpp > ...

MyArray::MyArray(int data){
    size = 100;
    for (int cnt=0;cnt< size; cnt++)
        arr[cnt] = data + cnt;
}


MyArray::MyArray(const MyArray& rhsObj){
    size = rhsObj.size;
    for (int cnt=0;cnt< size; cnt++)
        arr[cnt] = rhsObj.arr[cnt];

}
void MyArray::printArray(){
    cout<<"Size: "<<size<<"\tarray: "<<endl;
    for(int cnt=0;cnt < size; cnt++)
        cout<<arr[cnt]<<"  ";
    cout<<endl;
}

MyArray& MyArray::extend(MyArray& rhsObject){
    //For future implementation with dynamic memory allocation

    return rhsObject;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\myArrayOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Size: 100       array:
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0
Size: 100       array:
100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129
 130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  1
59  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188
 189  190  191  192  193  194  195  196  197  198  199
Size: 100       array:
100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129
 130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  1
59  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188
 189  190  191  192  193  194  195  196  197  198  199
```

11. ***Code :*** Write a program to demonstrate deep copy in C++ for a class that uses dynamic memory allocation, where a copy constructor allocates separate memory for each object to prevent dangling pointers and double deletion when objects go out of scope.

```
myArrayPtrOne.cpp X
myArrayPtrOne.cpp > ...
#include <iostream>
using namespace std;
//code has an issue - dangling pointer
class MyArray{
    int *arr;
    int size;
public:
    MyArray();
    MyArray(int data);
    MyArray(const MyArray& );
    void printArray();

    ~MyArray(){
        if (size){
            delete [] arr;
        }
        size=0;
    }
};

int main(){
    MyArray objTwo(100);//single parameterized ctor
    objTwo.printArray();
    {
        MyArray objThree = objTwo;//copying one object to another
        objThree.printArray();
    }
    objTwo.printArray();
}

MyArray::MyArray(const MyArray& rhs){
    size = rhs.size;
    arr = new int[size]; //separate memory allocation
    for (int cnt = 0;cnt < size; cnt++)
        arr[cnt] = rhs.arr[cnt]; //data being copied
```

```
        arr[cnt] = rhs.arr[cnt]; //data being copied
}
MyArray::MyArray():size(10){
    arr = new int[size];
    for (int cnt=0;cnt< size; cnt++)
        arr[cnt] = 0;
}

MyArray::MyArray(int sz):size(sz) {
    int data = 101;
    arr = new int[size];
    for (int cnt=0;cnt< size; cnt++)
        arr[cnt] = data + cnt;
}

void MyArray::printArray(){
    cout<<"Size: "<<size<<"\tarray: "<<endl;
    for(int cnt=0;cnt < size; cnt++)
        cout<<arr[cnt]<<"  ";
    cout<<endl;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\myArrayPtrOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Size: 100       array:
101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126
127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152
153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178
179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200
Size: 100       array:
101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126
127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152
153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178
179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200
Size: 100       array:
101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126
127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152
153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178
179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200
```

12. ***Code :*** Write a program to demonstrate the use of the default copy constructor in C++, where an object containing a statically allocated array is copied to another object, resulting in a member-wise (shallow) copy that works safely because no dynamic memory allocation is involved.

myArrayThree.cpp

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

class MyArray{ //abstraction
  int arr[100]; //array
  int size; //integer
public:
  MyArray();
  MyArray(int data);
  void printArray();
};

int main(){
  MyArray objTwo(100);//single parameterized ctor
  MyArray objThree = objTwo;//copying one object to another

  objTwo.printArray();
  objThree.printArray();
}

MyArray::MyArray(){
  size = 100;
  for (int cnt=0;cnt< size; cnt++)
    arr[cnt] = 0;
}
```

```cpp

MyArray::MyArray(int data){
    size = 100;
    for (int cnt=0;cnt< size; cnt++)
      arr[cnt] = data + cnt;
  }

void MyArray::printArray(){
    cout<<"Size: "<<size<<"\tarray: "<<endl;
    for(int cnt=0;cnt < size; cnt++)
      cout<<arr[cnt]<<"  ";
    cout<<endl;
  }
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\myArrayThree.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Size: 100       array:
100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125
126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151
152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177
178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199
Size: 100       array:
100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125
126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151
152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177
178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199
```

13. *Code* : Write a program to implement a custom MyArray class in C++ that demonstrates static array usage, default, parameterized, and copy constructors, and a member function to extend (merge) two array objects into a new object, illustrating object copying, array manipulation, and return-by-value behavior.

myArrayTwo.cpp

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

class MyArray{ //abstraction
  static const int max = 100;
  int arr[max];
  int size;
public:
  MyArray(); //size = 10
  MyArray(int data);//size = 20
  MyArray(const MyArray&);//copy ctor
  void printArray();
  MyArray extend(MyArray&);//10 + 20 --> less than 100(max)
};

int main(){
  MyArray objOne;
  MyArray objTwo(100);
  MyArray objThree = objTwo; //copy constructor called
  objOne.printArray();
  objTwo.printArray();
  objThree.printArray();

  MyArray objFour = objOne.extend(objThree);
  objFour.printArray();
}

MyArray::MyArray(){
  size = 10;
  for (int cnt=0;cnt< size; cnt++)
    arr[cnt] = 0;
}
```

myArrayTwo.cpp

```cpp

MyArray::MyArray(int data){
  size = 20;
  for (int cnt=0;cnt< size; cnt++)
    arr[cnt] = data + cnt;
}

MyArray::MyArray(const MyArray& rhsObj){
  size = rhsObj.size;
  for (int cnt=0;cnt< size; cnt++)
    arr[cnt] = rhsObj.arr[cnt];
}

void MyArray::printArray(){
  cout<<"Size: "<<size<<"\tarray: "<<endl;
  for(int cnt=0;cnt < size; cnt++)
    cout<<arr[cnt]<<"  ";
  cout<<endl;
}

MyArray MyArray::extend(MyArray& rhsObj){
  MyArray newObj;
  newObj.size = size + rhsObj.size;

  for (int cnt=0;cnt< size; cnt++)
    newObj.arr[cnt] = arr[cnt];

  for (int nCnt=0, cnt = size; cnt < newObj.size; cnt++)
    newObj.arr[cnt] = rhsObj.arr[nCnt++];

  return newObj;
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\myArrayTwo.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Size: 10        array:
0  0  0  0  0  0  0  0  0  0
Size: 20        array:
100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119
Size: 20        array:
100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119
Size: 30        array:
0  0  0  0  0  0  0  0  0  0  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119
```

14. ***Code :*** Write a program to implement the Singleton Design Pattern in C++, where a class ensures that only one object is created throughout the program by using a private constructor, a static pointer, and a static member function to provide controlled access to the single instance.

singleTon.cpp

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

class Singleton{
  private: //data members here
    static Singleton *ptr2Obj;
    Singleton(){
      cout<<"Object created..."<<endl;
    }
    Singleton(int x){
      cout<<"Object created...with "<<x<<endl;
    }
    Singleton(const Singleton &);
  public:
    static Singleton& getSingleton(){
      if (ptr2Obj==nullptr)
        ptr2Obj = new Singleton();

      return *ptr2Obj;
    }
    void disp(){
      cout<<"Singleton::disp()"<<endl;
    }
};

Singleton* Singleton::ptr2Obj=nullptr;
int main(){
  //Singleton objOne;//this is an error

  Singleton &refObj = Singleton::getSingleton();
  refObj.disp();
}
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\singleTon.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Object created...
Singleton::disp()
```

15. Code : Write a program to demonstrate static data members and static member functions in C++, where a class-level variable is shared among all objects and can be accessed using both the class name and object instances.

staticOne.cpp

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

class Employee{
  static string compName;//class member declaration
public:
  static string company(){return compName; }//class member
};

int main(){
  cout<<"Employee::company() --> "<< Employee::company()<<endl;
  Employee obj1;
  cout<<"obj1.company() --> "<< obj1.company()<<endl;
}

string Employee::compName = "Wipro Ltd";
```

Output :

```
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> g++ .\staticOne.cpp
PS C:\Users\VIKAS SRIVASTAVA\OneDrive\Desktop\C_CPP\Day_6\Classwork> .\a.exe
Employee::company() --> Wipro Ltd
obj1.company() --> Wipro Ltd
```